Week 4 Reading Summaries: Deep Compression + Quantization Methods Survey

Amogh Patankar
University of California, San Diego
3869 Miramar Street Box #3037, La Jolla, CA- 92092
apatankar@ucsd.edu

Abstract

With these readings, we dive deep into optimization techniques for modern day AI, namely compression and quantization. The first paper looks at deep compression, which includes modeling pruning, trained quantization, and Huffman coding, all of which help to compress models by up to 49x. Moreover, it improves speedup for CPU, GPU, and mobile GPU. The second reading dives deep into quantization methods for efficient NN inference; namely, they compare and contrast quantization methods for deep NN computations.

1 Deep Compression

1.1 Introduction

Neural Newtorks struggle with storage and memory bandwidth at large dimensions/sizes, making it hard to deploy on mobile systems. Moreover, energy consumption is another issue, as large models require a lot of memory bandwidth, using up energy. Deep compression is a pipeline to reduce storage and energy through connection removals, quantization, and huffman coding.

1.2 Network Pruning

Network pruning in the past has involved reducing network complexity and overfitting, and previously occurred without losing accuracy. The authors learn connectivity via normal training, then prune small weight connections below a threshold, and finally retrain for remaining connections. They use CSR or CSC formats to store elements, and encode the index differences.

1.3 Trained Quantization and Weight Sharing

The authors then work with network quantization and weight sharing, i.e. limiting the effective weights, and fine tuning those weights. They quantize weights, with all weights in a bin containig the same value, meaning each weight can be stored in a table with a small index. When updated, the gradients are summed, multiplied by LR, and subtracted from the shared weights. Using the formula for compression rate, they only need $log2(k)$ bits, where k = # of clusters.

1.3.1 Weight Sharing

Using k-means clustering, they find shared weights for each layer, such that all weights in same cluster are equal. Weights don't get shared across layers, and they use WCSS to cluster weights within a layer.

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.

1.3.2 Initialization of Shared Weights

When sharing weights, centroid initilization matters, and they survey the following: forgy, density-based, and linear. Forgy initialization chooses k random centroids, then concentrates around peaks in the distribution. Density-based creates a CDF, then makes centroids near the peaks. Linear spaces centroids evenly, and is the most scattered.

1.3.3 Feedforward and Backprop

With respect to FF adn BP, indices are stored, and gradient for the shared weight is calculated and updates the weight.

1.4 Huffman Coding

1.4.1 Pruning + Quantization Together

Pruned networks alone see accuracy dropping significantly, as do quantized networks; together, they do not drop in accuracy.

1.4.2 Centroid Initializations

Linear initialization lets larger weights have a better chance of forming a large centroid.

1.4.3 Speedup + Energy Efficiency

Deep compression targets latency-focused applications like on-device ML, and so they consider a batch size = 1. FC layers dominate model sizes and get compressed the most. The memory access is $O(n^2)$ and the computation is $O(n^3)$. Finally, the speedup is approximately 3-4x on average due to a smaller memory footprint.

1.4.4 Ratio of Weights, Index, Codebook

Quantization allows for a codebook; codebooks barely make up any overhead.

1.5 Related Work

Overparametrization of neural networks results in a waste of computation and memory usage; many research papers exploit linear structures, while many more bin parameters into buckets.

1.6 Future Work

The authors focus on quantizing networks with weight sharing, as current libraries don't support matrix entry lookup.

1.7 Conclusion

Deep compression proves that NNs can be compressed without losing accuracy, through pruning, quantizing, and Huffman coding.

2 Survey of Quantization Methods

2.1 Introduction

Modern ML has shown improvements in NNs; accuracies have increased for over-parametrized models, yet these models are impractical for on-device ML. A lot of the literature falls into a few categories: efficient NN architectures, NN SW/HW co-design, Pruning, Distillation, and Quantization.

2.2 Quantization

While all these methods work to a degree, quantization seems to be foundational to optimize overparametrized models. Quantization has *technically* existed since the mid-20^{th} century, but realistically has become prominent in the last 20 years. Roundoff errors are the biggest problem, and there are still many issues associated with quantization as of now.

2.2.1 Quantization in NNs

In neural networks, many novel algorithms share the same ideas; however, due to inference and training being expensive, representing numerical values is key. Moreover, reducing bit precision is a wide field, but there may be a vast difference between quantized and the original models due to overparametrization. Mixed-precision approaches have been spurred due to varying layers having varyied impact on loss.

2.3 Basic Concepts of Quantization

2.3.1 Problem Setup + Notations

With L layers in a NN, the goal is to optimize the risk minimization function, and we have some parameters θ that are stored in FP precision.

2.3.2 Uniform Quantization

A popular choice for a quantization function is $Q(r) = Int(r/S) - Z$ which is unifrom quantization, i.e. quantized values are spaced uniformly.

2.3.3 Symmetric + Asymmetric Quantization

The value of S from uniform quantization is calculated as such: $S = \frac{\beta-\alpha}{2^b-1}$, with a clipping range of $[\alpha, \beta]$. A symmetric quantization would mean a range where $\alpha = -\beta$, and asymmetric would mean $-\alpha! = \beta$.

2.3.4 Range Calibration Algorithms- Static vs Dynamic

Dynamic quantization computes the clipping range dynamically and has highest accuracy, but due to high cost of calculation, we go with static quantization, with a fixed clipping range.

2.3.5 Quantization Granularity

For CV tasks, there are many channels/filters; as such, channelwise quantization is used for convolutional kernels, while layerwise and groupwise, and subchannelwise quantization are not used due to high overhead.

2.3.6 Non-Uniform Quantization

Non-uniform quantization allows us to get better information and assigns bits + parameter range non-uniformly, but is hard to deploy on HW. Hence, uniform quantization is still the primary method as it's simple and efficiently mapped to HW.

2.3.7 Fine-Tuning Methods

- Quantization Aware Training (QAT): The disadvantage is computational cost of retraining the model, where we requantize the parameters after every gradient update.
- Post-Training Quantization (PTQ): While it's quick, as we do all quantization without retraining the model, we have lower accuracy as compared to QAT.
- Zero-Shot Quantization (ZSQ): We perform quantization with training or validation data, meaning they use SDG mimicing the original distribution to quantize activation quantization parameters + weights.

2.3.8 Stochastic Quantization

Stochastic Quantization maps the floating point number up or down based on probabilities calculated from the weight update. Again, the overhead of random numbers for all weight update has stopped stochastic quantization from being widely adopted.

2.4 Advanced Concepts: Quantization below 8 Bits

2.4.1 Simulated + Int-only Quantization

Simulated quantization stores parameters in low-precision, but carries out ops with FP arithmatic. Int-ony quantization does everything in low-precision.

2.4.2 Mixed Precision Quantization

It's both efficient and effective method for low-precision quantization of NNs, and layers vary in low/high bits used, reducing accuracy degradation.

2.4.3 HW Aware Quantization

: Using an RL agent, you determine the best mixed-precision setting, through latency lookup tables with different bandwidths.

2.4.4 Distillation-Assisted Quantization

The loss function uses both student and distillation loss, and leverages knowledge form intermediate layers.

2.4.5 Extreme Quantization

The methods used have a high accuracy degradation as we use *extremely* low bit precision quantization.

2.5 Quantization and HW Processors

Edge devices have high resource constraints, and often cannot execute heavy NN models. There are new developments like RISC-V SoC and other SoCs for improved acceleration on-device, but NN models must still be quantized to execute on device without breaking memory, compute, and power budgets.

2.6 Future Directions for Research in Quantization

2.6.1 Quantization SW

TensorRT, TVM, and other SW libraries make it easier to quantize models that isn't sub-INT8.

2.6.2 Hardware + NN Architecture Co-Design

This line of work explores co-designing HW and NN Architecture, specific for different quantized values.

2.6.3 Coupled Compression Methods

Coupled compression simply refers to combining multiple methods (pruning, quantization, and other methods for efficient NN deployment).

2.6.4 Quantized Training

Quantized training allows faster and power-efficient logic, but pushing it to INT8 precision is hard. This area is high-impact and difficult.

2.7 Summary and Conclusions

This survey presents a useful look into broader quantization techniques, and more advanced quantization technqiues, as well as future research in this field. All of this is useful in the journey to more efficient on-device ML deployment.