Quantization-Aware Training of Graph Neural Networks

Meljohn Ugaddan
Department of Information Systems and Computer Science
Quezon City, Philippines
meljohn.ugaddan@obf.ateneo.edu

Patricia Angela Abu
Department of Information Systems and Computer Science
Quezon City, Philippines
patricia.reyes-abu@obf.ateneo.edu

***Abstract* — This study aims to explore the Graph Neural Networks and Quantization. Graph Neural Networks has been successful in modelling data that are non-uniform data such as Social Connection, Molecular Structure, Recommender system. Due to the model for being new, various quantization schemes can be implemented to such models.**

Recently Efficient AI models are introduced. Methods such as Pruning and Quantization are now widely used to various models such as CNNs for faster inference time. Pruning is a process of removing a weight connection in order for model to have a lower storage size and faster inference time. This method is often performed to weights that has little effect on the accuracy of the model. Quantization on the other hand, is a method of approximating floating-point number to integer. Both of these methods are being used in order to have a faster inference time and lower model storage size.

The study was able to conduct the viability of quantization on Graph Neural Networks by introducing a form of information loss, fake-quantization. The study was able to reduce the size of the model after the training with minimal loss of accuracy. For the dataset, the study uses Cora and Pubmed a citation network from torch geometric module on where it already has set of training and testing data. For the GNN architecture, the study use Graph Convulutional Networks with the task of edge prediction, a process of determining the likelihood of a node is connected to other node.

Keywords—Graph Neural Networks, Quantization, Efficient AI, Computer Architecture

I. Introduction

On recent years, the use of drones, smartphones and virtual reality devices becomes popular. At the same years, machine learning models are now being used widely from these devices, from selfies, drone deliveries, and self-driving cars. Machine Learning nowadays are becoming more larger, comparing models like ResNet 2015 and ResNet 2012 [6], the recent model is 16x larger because of having more layer to achieve higher accuracy. ResNet is one of many image-processing models that can be use in smaller devices. Which in turn, model efficiency under smaller devices becomes an interesting study for researchers and some of the methods that are often used is Pruning and Quantization [6].

Quantization has been well studied in CNNs and it shows how it makes models e.g., MobileNet, ResNet, Inception, FCN or Jasper smaller in terms for its memory size from 1 to 4 times using smaller integer bit wise INT4 and INT8, by performing various task e.g., Classification, Segmentation, Translation and Detection. Also, it reduces the training time of the models from 2 to 4 times from its original format by comprising from about 0 to 2 percent of the accuracy of the models.

Recently, Graph Neural Networks rose to popularity for its unique structures and higher accuracies in terms of modelling complex molecular structure for finding a cure of Covid-19 [2], recommender system for e-commerce sites such as Alibaba [1], and economics network [5]. These studies suggests that there is wide application of GNN, not only from research of molecular structure but also in business applications.

The study aims to explore the quantization of GNN by its own intricacy such as node or graph embedding, edge prediction, node classification, and graph classification.

II. Review of Related Literature and Referrences

Most recent work on Quantization is mostly done in CNNs, where they mostly perform data compression to reduces the size of the model with pruning [6]. The related study uses AlexNet model where it reduces its size from 233mb to 8.9mb.

Various Quantization schemes are also proposed such as Quantization-Aware Training, Static Quantization (Post-Training), Integer Quantization (Vanilla Scaling) and Dynamic Quantization. Quantization-Aware training is the commonly use because of its higher accuracies [7]. And we can see the two direct related quantization scheme is mostly done in QAT [4,8].

One related study of Quantization in GNN is where the method is performed in nodes that are not important [4]. Meaning, a node has less incoming degree from the other nodes, in order to preserved the accuracy of the model. The study was performed in 3 architectural model of GNN which is the GCN, GAT and GIN. Also, another study [4] where quantization is performed in different level of the model (components, topology and neural network layers). The said study use Layer-Wise Quantization which quantization is done differently on each layer of the GNN. Some layers would be quantized in INT8 and some layer will be converted to INT4 depending on how much each of those layer affects the accuracy of the model.

The use Straight-Through Estimation for quantization is a common technique to deal with its non-differentiable nature of value clipping, like from a direct related study [4]. This study will be going to use straight-through estimation without a value clipping for the backpropagation of the gradients of the model.

III. METHODOLOGY

A. Range Mapping

First, we let $[\beta, \alpha]$ to represent the actual value of the numbers that we will $x \in [\beta, \alpha]$. For our simulation, we represent β as the minimum value, and α as the maximum value. On code level, we use pytorch function `torch.min()` and `torch.max()` to get the value of $[\beta, \alpha]$ .Also, for the number of byte for integer, the study represent the number of integer as b. For our simulation we are going to use int8 so the range of our values is going to be $[127, -128]$ when x is quantized.

$$s = \frac{\alpha - \beta}{-(-2^{b-1}) + 2^{b-1} - 1} \tag{1}$$

The study derives its own notation for each quantization equation from a related study [8] to tailor this study needs. In order to get the equivalent integer of a floating point, use the equation 1 to get the scaling number of floating-point relative to the bit integer.

$$quantize(x, b, s) = \frac{x + 2^{b-1} * s - \beta}{s} \tag{2}$$

$$clip(x, l, u) = \begin{cases} l, & x < l \\ x, & l \le x \le u \\ u, & x > u \end{cases} \tag{3}$$

$$x_q = round(clip(quantize(x, b, s), 2^{b-1} - 1, -(-2^{b-1}))) \tag{4}$$

It then uses the equation 2 to scale the floating number point to integer. Then, we use clipping function to limit the number of integers within the x values. The variable l represents the minimum value and u as the maximum value. For the study use case, as it uses int8 integer, it has a limit of $[127, -128]$ where l represents number 127 and u represents the number -128. For code level, the study uses python function `clamp(x,min,max)` to limit the number of integers together python function `round(x)` to round off to the nearest integer. Formally we will be going to come out with a quantized number x_q as it shown in equation 4.

$$\hat{x} = dequantize(x_q, s, \beta) = \frac{x_q * s - \beta}{s} \tag{5}$$

For converting back to integer to the study uses the equation 5 to get the equivalent floating number from the quantized value. The figure 1 shows a more intuitive representation of the equation of quantization mapping. As we can see from the figure 1, this from a related study [8], anything that is more than the value of α will be clamp to maximum number presentation in bit. In this case it is 127. This type of mapping is called Affine-Quantization, where z is pointing to the number 0 from the direct integer

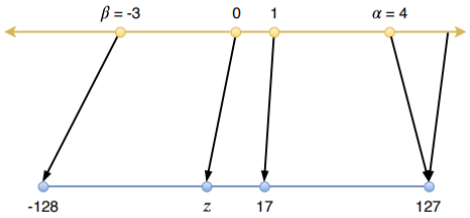

Figure 1: Quantization Mapping (Wu H., Judd P., Zhang X.,2020)

B. Model and Dataset

For our model, the study uses the sample code for Graph Convolutional Network (GCN) from torch geometric website [11]. For the formal definition of GCN [12], we have:

$$\mathbf{h}_{l+1}^{(i)} = \sum_{j \in \mathcal{N}(i) \cup \{i\}} \left(\frac{1}{\sqrt{d_i d_j}} \mathbf{W} \mathbf{h}_l^{(j)} \right) \tag{6}$$

Graph Convolutional Network has been used widely in graph classification, node prediction, and edge prediction. Graph Classification is when there's a clustering of group within the graph. The edge prediction or link prediction, is a GNN task to determine how a given node is connected to another node. Though, it could also how the graph will behave if we removed a node. Node prediction on the other hand is given a node and its neighbor, what will be the feature of the given node will be.

For our datasets, we use Cora and Pubmed, a commonly used dataset in GNN, and it is described as a citation network Cora has 2,708 nodes with 10,858 number of connections between those nodes. For Pubmed it has larger number of nodes which is 19,717 with 88,676 edges.

TABLE I. DATASET

Dataset	***Number of Node***	***Number of Edges***
Cora	2,708	10,858
Pubmed	19,717	88,676

C. Training

For the training, the study has two scenario one is the normal training without quantization, and one with quantization. For the quantized trained model, it is common methodology in quantized-aware training to insert a fake-quantization layer (6) each level of the GCN model. In this manner, the model is not only learning on the Graph Datasets but also to the information loss that we do in during the training.

$$\hat{x} = dequantize(quantize(x, b, s), s, \beta) \qquad (6)$$

One problem with Quantization-Aware Scheme is that it is not differentiable because of the nature of the function clamp(x,min,max) and round(x) .On this study we use straight-through each fake-quantization layer to skip the gradients within the quantization layer. In the code level, it is simple as passing the gradient from previous layer, skipping the current layer (quantization) since it is not differentiable, then pass the gradient to the next layer.

For more intuitive explanation, figure 2 shows how fake-quantization happens during the training and updating of the gradients. The figure shows after activation is being done, the activated input x will then enter the fake-quantization layer, then $\hat{x}$ will be given to the loss function. As backpropagation is done, STE will be implemented by skipping the quantization layer.

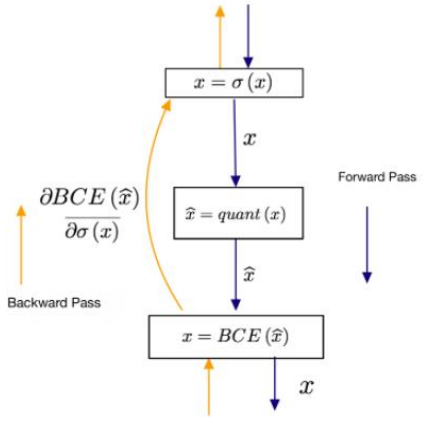

Figure 2: Fake-Quantization Diagram

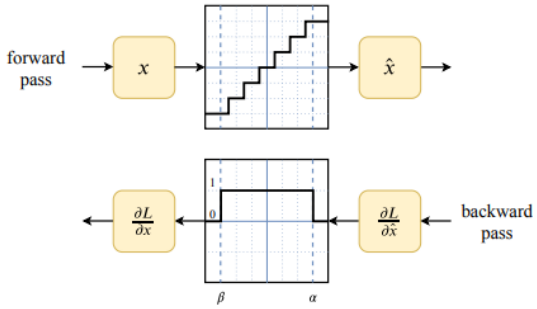

Figure 3: Graph View of Quantization (Wu H., Judd P., Zhang X.,2020)

A graphical form of Quantization is shown in Figure 3. We can see on the figure, the quantization is not differentiable because of the round() and clip() nature, therefore the gradient that will be return during the backward pass will be zero.

D. Environment Setup and Hardware Specification

The training is done in 200 epochs for both datasets. The GPU device that the study use is the GeForce RTX 2060 Super (8 gb). For the CPU, it uses Intel Core i5-9400F. For python we used version 3.8.8.

IV. Results and Discussions

A. Evaluation Result

TABLE II. Quantization Performance

Dataset	***Model***	***Accuracy (%)***
Cora	GCN(Full-Precision)	80.00 ±0.2
	GCN (Reduced-Precision)	79.80±0.3
Pubmed	GCN(Full-Precision)	78.80 ± 0.2
	GCN (Reduced-Precision)	78.70 ±0.2

As the fake-quantization or information loss introduced in each layer of the model, there no significant loss of accuracy of the GCN architecture in terms of accuracy as we can see on the table above. Full precision model is the trained in using FP32 datatype, while the Reduced-Precision model are the one who has a fake-quantized layer on the model.

For the the dataset, the study achieves lower accuracy on inference stage of the model. We assume this is because Pubmed has higher number of nodes and connections.

By comparing our result to other study [4], most of the accuracy result that the study has less accuracy for about two to five percent in terms of both full-precision model. The study assume that it might be because of the GPU device that is used is different from this study.

For the file size of the model, the model was reduced from 75kb to 38kb, which is approximately 2 times from the full-precision to reduced precision. This reduction of file size is good to resource limited device.

V. Conclusion and Recommendation

The study was able to demonstrate the effect of using the most common implementation of Quantization-Aware Training, Fake-Quantization. Based on the result the study was able to reduce the file size of the model for about 2times from its original floating-point precision with tangible loss of accuracy.

This study is done in a short duration of time, lots of experimentation can be further. First, the study uses the uniform quantization. This is applicable to data that is uniformly distributed which is not preferrable to data that is sparse. The sparsity of data is crucial to large dataset such as Amazon and Reddit. Next, the study can be further to other Graph Neural Network (GNN) architecture such as AGNN, GIN and GAT, in which each if these models have its own intricacies.

The use Straight-Through Estimation in fake-quantization layer can be also experimented in different scenario, like from a direct related study [4], value clipping is added within the from the STE for the updating of the gradients since there is a value clipping in the layer. Gradient Value clipping in Straight-Through estimation is not new to Quantization, but also in activation function such as clipped ReLu where the function is not differentiable which other studies that uses such activation layer.

For the benchmarking of result, the quantized model, inference time can be added from the experimentation, since this is crucial smaller device like embedded system. On which these devices are often resource-limited in terms of storage and computing power.

For the quantization, Full INT8 model can be added in the experimentation. INT4 can be considered for reference but base on previous studies it amounts to huge amount of accuracy loss in testing. Since there is new quantization scheme that is newly introduced, Layer-Wise Quantization and Granular Quantization should be considered.

For calibration of $[\beta, \alpha]$ during range mapping, the study was able to use the Min/Max calibration which can be extended to other mapping such as Percentile Mapping (99%) and Entropy. Though, based on previous studies some of these mapping cannot be entirely to all use case, some calibration can be applied to specific type of use-case and other can be used to different use-case. Entropy uses KL Divergence to minimize information loss between the original floating-point. Percentile mapping on other hand, maps the 99% of the data then clip the 1% percent representation of the distribution.

The Graph Neural Network task that is used in the study is only done in Edge Prediction. Edge prediction is used to know the likelihood of a node is being connected to a given node. Other GNN techniques can be introduced in the benchmarking of the results further this study, e.g., Graph classification and Node Classification. Graph classification is use of how likelihood a node belongs to a class of node. Node Classification on the other hand, is determining the labelling of a node based from its neighbors.

REFERENCES

[1] Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, Jingren Zhou "AliGraph: A Comprehensive Graph Neural Network Platform" arXiv:1902.08730

[2] Gysi D., Valle I., Zitnik M."Network Medicine Framework for Identifying Drug Repurposing Opportunities for COVID-19" arXiv:2004.07229

[3] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. "Quantization and training of neural networks for efficient integer-arithmetic-only inference, 2017"

[4] Tailor S., Fernandez-Marques J, Lane N,"Degree-Quant: Quantization-Aware Training for Graph Neural Networks" arXiv:2008.05000

[5] Shengjie Min, Zhan Gao, Jing Peng, Liang Wang, Ke Qina, BoFang , "STGSN — A Spatial–Temporal Graph Neural Network framework for time-evolving social networks" arXiv:2101.11846

[6] 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

[7] Feng B., Wang Y., Li X., Yang S., Peng X., Ding Y., "SGQuant: Squeezing the Last Bit on Graph Neural Networks with Specialized Quantization" arXiv:2007.05100

[8] Wu H., Judd P., Zhang X., Isaev M., Micikevicius P. ,"Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation" arXiv:2004.09602

[9] Paszke, Adam and Gross, Sam and Chintala, Soumith and Chanan, Gregory and Yang, Edward and DeVito, Zachary and Lin, Zeming and Desmaison, Alban and Antiga, Luca and Lerer, Adam," Automatic differentiation in PyTorch" 2017

[10] Yang Z., Cohen W., Salakhutdinov R.," Revisiting Semi-Supervised Learning with Graph Embeddings" arXiv:2004.09602

[11] Fey, Matthias and Lenssen, Jan E.," Fast Graph Representation Learning with PyTorch Geometric" 2019

[12] Kipf T., and Welling M.," Semi-Supervised Classification with Graph Convolutional Networks" arXiv:1609.02907,2017