Post-Training 4-bit Quantization on Embedding Tables

Hui Guan[1], **Andrey Malevich**[2], **Jiyan Yang**[2], **Jongsoo Park**[2], **Hector Yuen**[2]
[1]North Carolina State University
[2]Facebook, Inc
hguan2@ncsu.edu, {amalevich, chocjy, jongsoo, hyz}@fb.com

Abstract

Continuous representations have been widely adopted in recommender systems where a large number of entities are represented using embedding vectors. As the cardinality of the entities increases, the embedding components can easily contain millions of parameters and become the bottleneck in both storage and inference due to large memory consumption. This work focuses on post-training 4-bit quantization on the continuous embeddings. We propose row-wise uniform quantization with greedy search and codebook-based quantization that consistently outperforms state-of-the-art quantization approaches on reducing accuracy degradation. We deploy our uniform quantization technique on a production model in Facebook and demonstrate that it can reduce the model size to only 13.89% of the single-precision version while the model quality stays neutral.

1 Introduction

The success of word embeddings in Natural Language Processing (NLP) [23, 20] has promoted the wide adoption of continuous representations in recommender systems in recent years. Embedding-based approaches have achieved state-of-the-art performance in recommendation and ranking tasks and have been successfully applied in real-world applications [21, 22, 5, 16, 12]. In these recommendation models, a large number of entities such as page ids and user ids are encoded using *embedding tables* whose *row vectors* correspond to entities. As embedding tables scale with the number of entities and embedding dimensions, they can easily contain billions of parameters and usually contribute to 99.99% of the size of the models. For example, a single-precision embedding table with 50,000,000 number of ids and 64 embedding dimensions costs 12GB memory and a recommendation model could contain up to hundreds of embedding tables. Furthermore, due to memory-bandwidth limitations, embedding table lookups are one of the most time-consuming operations. Their proportion will increase due to the acceleration of other parts (e.g. FC) from faster increase of compute throughput than memory bandwidth, posing a great challenge to get real-time predictions [22, 21].

Quantization is one of the effective approaches to reduce model size. By quantizing floating-point values in embedding tables to low-precision numbers that use less number of bits, a large recommendation model can be reduced to a model with much smaller model size and memory bandwidth consumption during inference. Prior work on quantization has been focusing on quantization-aware training from scratch or a pre-trained floating-point model [7, 28, 26, 11, 18, 8, 15, 10]. Although these techniques have shown promising results, they are not always applicable in many practical scenarios where the training dataset might be no longer available during model deployment [27, 3, 6, 19]. In these cases, post-training quantization is a more desirable approach. Post-training quantization is simple to use and convenient for rapid deployment. Recent studies have shown post-training quantization using 8-bit precision can achieve accuracy close to that of single-precision models in a wide variety of DNN architectures [19, 14]. Post-training quantization using lower bit width (e.g. 4-bit), however, usually incurs significant accuracy drop [6].

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

Several state-of-the-art post-training quantization techniques that rely on the clipping have been proposed to mitigate accuracy degradation. Shin et al. [24] and Sung et al. [25] approximated the inputs as a histogram and adopt a clipping threshold that minimizes the ℓ_2 norm of the quantization error. Migacz et al. [19] proposed an iterative approach to search for the clipping threshold based on Kullback-Leibler Divergence measure for quantizing activations. Later, Banner et al. [3] proposed ACIQ, an analytic solution that computes the optimal clip threshold by assuming the input values are sampled from a Gaussian or Laplacian distribution. Although these approaches are demonstrated to reduce the accuracy drops to some extent, the problem of post-training 4-bit quantization without accuracy drop is still unsolved yet. Empirically, we also observe that the above-mentioned approaches can result in significant accuracy drops when applied to embedding table quantization.

In this paper, we explore a variety of post-training 4-bit quantization methods on embedding tables and propose novel quantization approaches that can reduce model size while incurring negligible accuracy degradation. Quantization on embedding tables is usually applied to row vectors (*row-wise quantization*) to reduce quantization error. Throughout the paper, quantization is applied to row vectors unless noted differently. We notice that the prior post-training quantization approaches approximate the inputs to quantize using either a histogram or some distributions. While these assumptions are beneficial to derive efficient algorithms for finding the optimal clipping thresholds for weights and activations of convolutional neural networks (CNNs), they are not suitable for embedding tables because their row vectors contain too few values to be well-characterized using either histograms or distributions. Inspired by these understandings, we design quantization algorithms that directly target at minimizing the mean square error after quantization. Specifically,

> Our exploration reveals that state-of-the-art post-training 4-bit quantization approaches are no better than the approach that uses the range the input without clipping when the input contains only tens or hundreds of values, as in the case of embedding table quantization.
>
> We propose two simple yet effective approaches to improve 4-bit quantization on embedding tables: 1) row-wise uniform quantization with greedy search that finds the best clipping thresholds from a gradually discovered set of local optima; 2) codebook-based quantization that maps inputs to indices of non-uniformly distributed values using k-means clustering.
>
> Moreover, for 4-bit quantized embedding tables using uniform quantization, we can achieve dequantization performance similar to the Caffe2 8-bit dequantization operators (e.g., SparseLengthSum[1]) that are already heavily optimized.

We empirically demonstrate the effectiveness of our proposed quantization approaches on DNN-based recommendation models [26, 21] and also a production model in Facebook. The results show that the proposed approaches consistently outperform state-of-the-art post-training quantization approaches in reducing quantization error and accuracy degradation. Row-wise uniform quantization with greedy search can reduce the model size to 13.3%-25.0% of the baseline single-precision models with negligible accuracy loss. Codebook-based quantization can reduce the model size to 18.5%-37.5% of the single-precision model with no accuracy loss. We deploy our uniform quantization technique on a production model in Facebook and demonstrate that it can reduce the model size to only 13.89% of the single-precision version while the model quality stays neutral.

2 Prior Quantization Methods and Their Limitations

Let x be a value clipped to the range $[x_{min}; x_{max}]$. Quantization using n bits maps x to an integer in the range $[0; 2^n - 1]$, where each integer corresponds to a quantized value. If the quantized values are a set of discrete, evenly-spaced grid points, the methods are called *uniform quantization*. Otherwise, they are called *non-uniform quantization*. This section reviews several state-of-the-art post-training uniform quantization methods and explains their limitations on embedding table quantization.

Let x_{int} and x_{float} be the quantized and dequantized values respectively. Uniform quantization proceeds as follows:

$$x_{int} = round\left(\frac{x - x_{min}}{x_{max} - x_{min}}(2^n - 1)\right) = round\left(\frac{x - bias}{scale}\right); \quad (1)$$

[1] https://caffe2.ai/docs/operators-catalogue.html#sparselengthssum

where $scale = \frac{x_{max} - x_{min}}{2^n - 1}$ and $bias = x_{min}$. The de-quantization operation is: $x_{float} = scale \cdot x_{int} + bias$. The quantization is *symmetric* if $x_{max} = -x_{min}$. Otherwise, it is *asymmetric*. To ease the description below, we define a quantization function[2] as: $x_{float} = Q(x; x_{min}; x_{max})$.

Without loss of generality, let $X \in \mathbb{R}^N$ be an input vector to quantize. Tensors with higher dimensions can be flattened to a vector. Because each value is scaled by $x_{max} - x_{min}$, the naive quantization using $x_{max} = max(X)$ and $x_{min} = min(X)$ is sensitive to outliers, i.e., values with large magnitude in the input X, and could cause large accuracy drops. We refer to this method **range-based asymmetric quantization (ASYM)**.

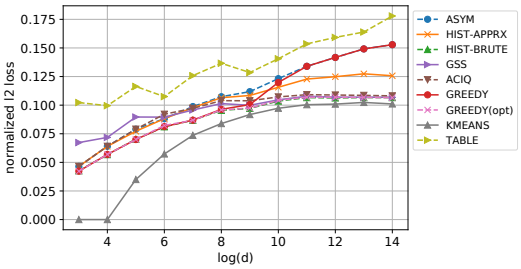

Figure 1: The normalized ℓ_2 loss of 4-bit quantization with different embedding dimensions on a FP32 embedding table with 10 row vectors. The values in the embedding table are randomly sampled from a normal distribution, which is in favor of GSS and especially ACIQ. TABLE applies range-based uniform quantization on the entire table while the other methods are on row vectors. HIST-APPRX and HIST-BRUTE use $b = 200$, GREEDY uses $b = 200, r = 0.16$; GREEDY (opt) uses $b = 1000, r = 0.5$.

State-of-the-art post-training quantization techniques rely on the clipping to mitigate accuracy degradation. They differ in their way to minimize the mean squared error (MSE) of the original values and the quantized values:

$$f(x_{min}; x_{max}) = \|X - Q(X; x_{min}; x_{max})\|_2^2 \quad (2)$$

Histogram-based Quantization (HIST) This method chooses the clipping thresholds which minimize the MSE between the histogram of floating-point inputs and that of the quantized versions [24]. Let x_i and $h(x_i)$, where $i = 1, \ldots, b$, be the bin value and the frequency of the i-th bin in the inputs' histogram. The optimization objective is defined as: $f_{hist}(x_{min}; x_{max}) = \frac{1}{b} \sum_{i=1}^{b} h(x_i) \cdot (x_i - Q(x_i; x_{min}; x_{max}))^2$. We used the approximate algorithm (**HIST-APPRX**) implemented in Caffe2 [1] that scales linearly with the number of bins to solve the above optimization problem. We also implemented a brute force approach (**HIST-BRUTE**) to find better solutions. Its time complexity is $O(b^3)$ (See Appendix A).

ACIQ Analytical Clipping for Integer Quantization (ACIQ) [3] derives the clipping thresholds analytically from the distribution of the tensor. It assumes that the values in the tensor are sampled from a Gaussian or a Laplacian distribution. After determining the distribution to use, it uses an approximate closed-form solution for the clip thresholds which minimizes MSE in Eq. 2. For example, if the tensor is closer to a Laplacian distribution, the clipping thresholds for 4-bit quantization are calculated using the formula: $x_{min} = \mathbb{E}(X) - \alpha; x_{max} = \mathbb{E}(X) + \alpha$, where $\alpha = 5.03 \cdot \mathbb{E}(|X - \mathbb{E}(X)|)$. We used the open-source code from the authors[3].

Quantization with Golden Section Search (GSS) Instead of approximating the floating-point inputs using a histogram or assuming it follows a certain distribution, this approach finds a range limit x_{thr} that minimizes MSE using golden section search (GSS) [13] for symmetric quantization. The objective function is simplified as: $f_{sym}(x_{thr}) = \frac{1}{N}\|X - Q(X, -x_{thr}; x_{thr})\|_2^2$. The method is applied to compress word embeddings in [17].

Their Limitations Quantization on embedding tables is commonly applied to row vectors to reduce the quantization error (See ASYM v.s. TABLE in Figure 1). The embedding dimension in recommendation models is usually 8 to 200 [21]. The above clipping-based approaches are better than the range-based asymmetric quantization (ASYM) method when quantizing weights and activations of CNNs to 4-bit. However, we empirically observed that they are no better than ASYM when the input X to quantize has a small dimension (i.e., a small number of values), as in the case of

[2] An alternative uniform quantization uses: $x_{int} = round(x/scale) - zero\_point$. This method is better when the inputs to quantize have lots of zeros, e.g., ReLU activations. We found that the mapping in Eq. 1 provides better accuracy for embedding table quantization.

[3] `https://github.com/submission2019/cnn-quantization`

embedding table quantization. Approximating row vectors in an embedding table using histograms or distributions could give large quantization error.

Figure 1 shows the normalized ℓ_2 loss of different quantization methods with various embedding dimensions. Normalized ℓ_2 loss is calculated as $\|X - Q(X; x_{min}, x_{max})\|_2 / \|X\|_2$. It measures relative quantization errors. Overall, when the embedding dimension is larger than 1024, GSS, ACIQ, HIST-APPRX, and HIST-BRUTE can achieve a smaller loss compared with ASYM. However, when the embedding dimension is small (e.g. 64), the advantage of GSS, ACIQ, and HIST-APPRX over ASYM is gone. GSS is even much worse than ASYM. Although HIST-BRUTE is still better than ASYM, it is very time-consuming (millions of times slower than ASYM) and too expensive to be applied in real-world recommendation applications that require continuous learning and thus periodic quantization for model serving (See Appendix A).

3 Proposed Quantization Approaches

This section elaborates the proposed uniform quantization with greedy search and codebook-based quantization with k-means clustering.

Uniform Quantization with Greedy Search (GREEDY) To overcome the limitations of the prior uniform quantization approaches, we propose a greedy search algorithm (see Algorithm 1) to find the optimal clipping thresholds. The algorithm is inspired by the 1-D golden section search (GSS) and directly targets at minimizing the MSE objective function in Eq. 2. Although 2-D GSS was proposed recently, it is not applicable in general as it is too consuming [4]. The basic idea of greedy search is to find as many local optima as possible and select the best one as the clipping thresholds. The algorithm takes the input vector X and two hyper-parameters b and r that balance the optimality of its solution and its time complexity.

The algorithm initializes x_{min} and x_{max} with the range of the input X (lines 1-2). It then gradually increases x_{min} or decreases x_{max} by one $stepsize$ to reduce the range and find a smaller loss calculated as Eq. 2 (lines 7-16). The algorithm stops when the current range is $1 - r$ percentage of the range of X (lines 5-6). The larger the b and r are, the better the found solution will be but the higher the time cost is ($O(b \cdot r)$ time complexity). The default value of b and r are set as 200 and 0.16 respectively.

Algorithm 1 greedy search

```
Input:  X     // a vector to quantize.
Input:  b     // default: 200
Input:  r     // default: 0.16
Output: xmin, xmax // range used for quantization
1:  xmin = cur_min = min(X)
2:  xmax = cur_max = max(X)
3:  loss = compute_loss(X, Q(X, xmin, xmax))
4:  stepsize = (xmax - xmin)/b
5:  min_steps = b * (1 - r) * stepsize
6:  while cur_min + min_steps < cur_max do
7:      loss_l = compute_loss(X, Q(X, cur_min + step-
        size, cur_max))
8:      loss_r = compute_loss(X,Q(X, cur_min, cur_max
        - stepsize))
9:      if loss_l < loss_r then
10:         cur_min = cur_min + stepsize
11:         if loss_l < loss then
12:             loss, xmin = loss_1, cur_min
13:     else
14:         cur_max = cur_max - stepsize
15:         if loss_r < loss then
16:             loss, xmax = loss_r, cur_max
17: return  xmin, xmax
```

Codebook-based Quantization Codebook-based quantization is a type of non-uniform quantization that maps each input value to the index of a value in the codebook. Quantizing a value to 4 bits means the number of values in a codebook cannot be larger than 16. We consider the following two codebook-based quantization variants: Quantization with Rowwise Clustering (KMEANS) and Quantization with Two-Tier Clustering (KMEANS-CLS) . KMEANS algorithm applies k-means clustering to produce a 16-value codebook for each row vector, and then maps each value in the row vector to the index of the codebook based on its cluster assignment. Although the algorithm has less model size reduction than uniform quantization due to the storage overhead of codebooks, it has the potential to achieve a lower MSE and avoid accuracy degradation.

To achieve a larger compression rate, KMEANS-CLS applies k-means clustering in a more coarse-grained way. The algorithm first groups similar row vectors in an embedding table into blocks (called tier-1 clustering) and then generates a 16-value codebook for each block (called tier-2 clustering).

Table 1: Computational throughput in billion sums per second for SparseLengthsSum operators. The performance is measured using a single core of Intel Xeon Gold 6138 CPU @ 2.0 GHz with turbo-mode off.

Data type	Cache non-resident case				Cache resident case			
	d=64	d=128	d=256	d=512	d=64	d=128	d=256	d=512
FP32	1.939	1.908	1.997	2.063	2.804	4.165	4.209	4.127
INT8	1.246	1.511	2.726	3.076	2.242	2.510	3.748	3.450
INT4	1.608	2.047	2.532	5.581	2.093	2.878	6.454	6.893

Both steps use k-means clustering. Let K_1 be the number of clusters in tier-1 clustering. After 4-bit quantization using KMEANS-CLS, the required number of bytes to store a d embedding table is $Nd/2 + N \log_2 K_1/8 + 64K_1$, where $\log_2 K_1/8$ is the number of bytes used to store tier-1 cluster assignment.

4 Efficient 4-bit Embedding Operation Implementation

A challenge for quantizing embedding tables in less than 8-bit is the overhead of dequantization when reading the tables. This is because, in most commonly used processors, 8-bit is the smallest granularity in which instructions can operate with, and less than 8-bit granularity requires bit manipulations like or, shift, and so on. Nevertheless, we found that we can sustain good enough dequantization throughput with careful use of vector instructions available in recent CPUs (e.g., AVX512 in Intel Skylake CPUs) as shown in Table 1. We measure computational throughput of SparseLengthsSum operator (the most time-consuming operator reading embedding tables in our recommendation models [21]) in FP32, INT8, and INT4, both in cache non-resident and cache resident cases. In cache non-resident cases, we flush the last level cache between benchmark runs, which is more representative of running big recommendation models with many huge embedding tables. The cache resident cases are to see upper bound computational throughputs (a worst case for 4-bit embedding tables). We can see that in most cases the speed of 4-bit SparseLengthsSum is on par or faster than its highly-optimized 8-bit or FP32 counterparts running in production.

5 Experiments

In this section, we present the experimental results of the proposed approaches. We use the Terabyte Criteo data [2]. It is a click prediction dataset that has a size of 1.3TB and contains more than 4.3 billion records. The dataset is a commonly used benchmark dataset for ranking applications.

The ranking problem is a binary classification problem. The models we used are DNN models [21]. For categorical features, following the same procedure as in [36], we transform them into dense vectors using embedding tables. The number of rows in embedding tables corresponds to the number of categorical features with a maximum of 50 million. The number of columns corresponds to the embedding dimension. We choose a variety of embedding dimensions: 8, 16, 32, 64, and 128, that are commonly used in ranking models. The dense embeddings of the categorical features, concatenated with the dense vector formed by dense features, are taken as the input to a neural network with 2 fully-connected (FC) layers. The FC layers have a width of 512. The models are trained using Adagrad [9] with a batch size of 100. The initial learning rate is set to 0.015 for embedding tables and 0.005 for the rest of the parameters. All the parameters are trained using single-precision (FP32). All the embedding tables are quantized to use 4 bits after model training is finished.

We compare the proposed approaches (GREEDY, KMEANS, KMEANS-CLS) with other uniform quantization approaches including SYM, GSS, ASYM, HIST-APPROX, HIST-BRUTE, ACIQ. Because k-means is sensitive to initialization, we initialize cluster centers using uniform quantization results from ASYM. The default hyperparameter settings ($b = 200; r = 0.16$) are used for greedy search. HIST-APPRX uses $b = 200$ as it gives the best performance after a grid-based hyper-parameter tuning. For KMEANS-CLS, we choose the K_1 such that it achieves the same compression rate as the uniform quantization approaches. If a method is appended with "(FP16)", it means that the scales and biases in uniform quantization and the codebooks in codebook-based quantization are stored using FP16 instead of FP32. Besides the baseline where embedding tables are not quantized (FP32), we include another baseline that uses range-based uniform quantization to quantize

Table 2: Normalized ℓ_2 loss with different quantization methods and embedding dimensions.

Methods	Description	d=8	d=16	d=32	d=64	d=128
ASYM-8BITS	Asymmetric, $x_{min} = \min(X)$; $x_{max} = \max(X)$	0.00260	0.00329	0.00376	0.00387	0.00400
SYM	Symmetric, $x_{min} = x_{max}$; $x_{max} = \max(\|X\|)$	0.05564	0.06296	0.06785	0.06836	0.06928
GSS	Symmetric with Golden Section Search	0.05269	0.05965	0.06328	0.06400	0.06423
ASYM	Asymmetric, $x_{min} = \min(X)$; $x_{max} = \max(X)$	0.04451	0.05479	0.06387	0.06608	0.06781
HIST-APPRX	Asymmetric with histogram-based approximation [1]	0.04452	0.05512	0.06409	0.06589	0.06768
HIST-BRUTE	Asymmetric with histogram-based brute force algorithm	0.04156	0.05082	0.05881	0.06083	0.06272
ACIQ	Analytical Clipping for Integer Quantization [3]	0.04451	0.05479	0.06387	0.06742	0.07665
GREEDY	Asymmetric with greedy search (ours)	0.03889	0.04878	0.05744	0.05991	0.06221
GREEDY (FP16)	Asymmetric with greedy search (ours)	0.03889	0.04879	0.05744	0.05991	0.06221
KMEANS-CLS (FP16)	Two-tier k-means clustering with uniform init (ours)	0.03948	0.05349	0.06826	0.07369	0.07287
KMEANS (FP16)	Rowwise kmeans clustering with uniform init (ours)	0	0	0.03670	0.05160	0.05781

"-8BITS": 8-bit quantization; otherwise, 4-bit quantization. "(FP16)": scales and biases or values of a codebook in FP16; otherwise, FP32.

Table 3: Model log loss and size with different quantization methods and embedding dimensions.

Methods	d=8		d=16		d=32		d=64		d=128	
	loss	size	loss	size	loss	size	loss	size	loss	size
FP32 (no quantization)	0.12522	8.07GB	0.12489	16.14GB	0.12468	32.27GB	0.12451	64.54GB	0.12454	129.09GB
ASYM-8BITS	0.12522	49.98%	0.12489	37.49%	0.12469	31.25%	0.12451	28.12%	0.12454	26.56%
SYM	0.12528		0.12507		0.13266		0.13107		0.12470	
GSS	0.12527		0.12504		0.13199		0.12843		0.12459	
ASYM	0.12526		0.12491		0.12496		0.12494		0.12455	
HIST-APPRX	0.12525	37.49%	0.12492	24.99%	0.12497	18.75%	0.12498	15.62%	0.12455	14.06%
HIST-BRUTE	0.12525		0.12490		0.12490		0.12489		0.12454	
ACIQ	0.12526		0.12491		0.12804		0.12514		0.12455	
GREEDY	0.12525		0.12490		0.12489		0.12485		0.12454	
GREEDY (FP16)	0.12525	24.99%	0.12490	18.74%	0.12489	15.62%	0.12485	14.06%	0.12454	13.28%
KMEANS (FP16)	-	-	-	-	0.12469	37.50%	0.12451	25.00%	0.12454	18.75%

all embedding tables to 8 bits (ASYM-8BITS). We evaluate the performance of the quantization approaches using three evaluation metrics: Normalized ℓ_2 loss, model log loss, and model size.

Table 2 lists the normalized ℓ_2 loss results on an embedding table from models with different embedding dimensions. Overall, our proposed approach GREEDY consistently gives the smallest loss among all 4-bit uniform quantization approaches. Using FP16 for scales and biases further reduces the embedding table size without affecting the loss. KMEANS achieves the smallest normalized ℓ_2 loss for the use of codebook. Even though KMEANS-CLS variants can achieve the same compression rate as the uniform quantization approaches, they suffer from larger losses, indicating the importance of row-wise quantization for embedding tables.

Table 3 lists the model log loss and model size for the models after 4-bit quantization. Overall, GREEDY consistently gives the smallest model log loss compared with other uniform quantization approaches while reducing the models to 13.3%-25.0% of the single-precision model size. The proposed KMEANS approach can even get the same model loss as the original single-precision model while reducing the models to 18.8%-37.5% of the single-precision model size.

We deployed GREEDY on one of the ranking applications at Facebook. The application uses a DNN model trained on billions of records. Being able to reduce the model size using post-training 4-bit quantization while preserving model accuracy is a challenging task. Our experimental results show that the 4-bit uniform quantization with greedy search can reduce the model size to only 13.89% of the single-precision version while the model quality stays neutral. This demonstrates the practicality of our approach in real applications.

6 Conclusions and Future Work

We proposed row-wise uniform quantization with greedy search and non-uniform quantization with k-means clustering to improve 4-bit post-training quantization on embedding tables. We empirically showed that the proposed approaches consistently outperform state-of-the-art quantization methods on reducing the quantization error and the model accuracy degradation. The model size reduction resulting from 4-bit quantization makes it possible to use even larger embedding tables for potentially better model accuracy. In the future, we want to explore how much accuracy gain can be achieved by increasing model size while applying 4-bit quantization to meet a certain space budget.

References

[1] caffe2 histogram-based norm minimization. https://caffe2.ai/doxygen-c/html/norm_minimization_8cc_source.html. Accessed: 2019-08-03.

[2] Criteo releases industry's largest-ever dataset for machine learning to academic community. https://www.criteo.com/news/press-releases/2015/07/criteo-releases-industrys-largest-ever-dataset/. Accessed: 2019-08-03.

[3] Ron Banner, Yury Nahshan, Elad Hoffer, and Daniel Soudry. Aciq: Analytical clipping for integer quantization of neural networks. arXiv preprint arXiv:1810.05723, 2018.

[4] Yen-Ching Chang. N-dimension golden section search: Its variants and limitations. In 2009 2nd International Conference on Biomedical Engineering and Informatics, pages 1–6. IEEE, 2009.

[5] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In Proceedings of the 1st workshop on deep learning for recommender systems, pages 7–10. ACM, 2016.

[6] Yoni Choukroun, Eli Kravchik, and Pavel Kisilev. Low-bit quantization of neural networks for ef cient inference. arXiv preprint arXiv:1902.06822, 2019.

[7] Matthieu Courbariaux and Yoshua Bengio. Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or- 1. arxiv 2016. arXiv preprint arXiv:1602.02830, 0.

[8] Christopher De Sa, Megan Leszczynski, Jian Zhang, Alana Marzoev, Christopher R Aberger, Kunle Olukotun, and Christopher Ré. High-accuracy low-precision training. arXiv preprint arXiv:1803.03383, 2018.

[9] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(Jul):2121–2159, 2011.

[10] Alexander Goncharenko, Andrey Denisov, Sergey Alyamkin, and Evgeny Terentev. Fast adjustable threshold for uniform neural network quantization. arXiv preprint arXiv:1812.07872, 2018.

[11] Qinyao He, He Wen, Shuchang Zhou, Yuxin Wu, Cong Yao, Xinyu Zhou, and Yuheng Zou. Effective quantization methods for recurrent neural networks. arXiv preprint arXiv:1611.10176, 2016.

[12] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative ltering. In Proceedings of the 26th international conference on world wide web, pages 173–182. International World Wide Web Conferences Steering Committee, 2017.

[13] Jack Kiefer. Sequential minimax search for a maximum. Proceedings of the American mathematical society, 4(3):502–506, 1953.

[14] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for ef cient inference: A whitepaper. arXiv preprint arXiv:1806.08342, 2018.

[15] Hao Li, Soham De, Zheng Xu, Christoph Studer, Hanan Samet, and Tom Goldstein. Training quantized nets: A deeper understanding. In Advances in Neural Information Processing Systems, pages 5811–5821, 2017.

[16] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 1754–1763. ACM, 2018.

[17] Avner May, Jian Zhang, Tri Dao, and Christopher Ré. On the downstream performance of compressed word embeddings. arXiv preprint arXiv:1909.01264, 2019.

[18] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. arXiv preprint arXiv:1710.03740, 2017.

[19] Szymon Migacz. 8-bit inference with tensorrt. In GPU technology conference, volume 2, page 7, 2017.

[20] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems, pages 3111–3119, 2013.

[21] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. Deep learning recommendation model for personalization and recommendation systems. arXiv preprint arXiv:1906.00091, 2019.

[22] Jongsoo Park, Maxim Naumov, Protonu Basu, Summer Deng, Aravind Kalaiah, Daya Khudia, James Law, Parth Malani, Andrey Malevich, Satish Nadathur, et al. Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications. arXiv preprint arXiv:1811.09886, 2018.

[23] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 1532–1543, 2014.

[24] Sungho Shin, Kyuyeon Hwang, and Wonyong Sung. Fixed-point performance analysis of recurrent neural networks. In 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 976–980. IEEE, 2016.

[25] Wonyong Sung, Sungho Shin, and Kyuyeon Hwang. Resiliency of deep neural networks under quantization. arXiv preprint arXiv:1511.06488, 2015.

[26] Jian Zhang, Jiyan Yang, and Hector Yuen. Training with low-precision embedding tables. In Systems for Machine Learning Workshop at NeurIPS, volume 2018, 2018.

[27] Ritchie Zhao, Yuwei Hu, Jordan Dotzel, Chris De Sa, and Zhiru Zhang. Improving neural network quantization without retraining using outlier channel splitting. In International Conference on Machine Learning, pages 7543–7552, 2019.

[28] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint arXiv:1606.06160, 2016.

A HIST-BRUTE

In this section, we present the algorithm and the complexity analysis of the brute force histogram-based quantization approach (HIST-BRUTE). Its pseudo-code is shown in Algorithm 2. For 4-bit quantization, the algorithm uses a histogram with 16 number of bins (`dst_nbins`) to approximate the histogram of the inputs with `b` number of bins. Lines 1-9 are for initialization. The algorithm selects different numbers of consecutive bins from the inputs' histogram and approximates these selected bins using 16 target bins (lines 10-36). The selected bins determine the clipping thresholds (lines 37-42). HIST-BRUTE has a time complexity of $O(b^3)$.

Algorithm 2 HIST-BRUTE

```
Input:  X // a vector to quantize.
Input:  b // number of bins used to generate histogram, default: 200
Output:  xmin, xmax // range used for quantization
 1: // Initialize
 2: xmin = min(X)
 3: xmax = max(X)
 4: histogram = get_histogram(X, b)
 5: dst_nbins = 16
 6: bin_width = (xmax - xmin)/b
 7: norm_min =1
 8: best_start_bin = -1
 9: best_nbins_selected = 1
10: for nbins_selected = 1to b do
11:     start_bin_begin = 0
12:     start_bin_end = b - nbins_selected + 1
13:     dst_bin_width = bin_width * nbins_selected / (dst_nbins - 1)
14:     for start_bin = start_bin_begin to start_bin_end do
15:         norm = 0
16:         // Go over each histogram bin and accumulate errors.
17:         for src_bin = 0to b do
18:             src_bin_begin = (src_bin - start_bin) * bin_width
19:             src_bin_end = src_bin_begin + bin_width
20:             // Determine which dst_bins the beginning and end of src_bin belong to
21:             dst_bin_of_begin = min(dst_nbins - 1, max(0,  oor((src_bin_begin + 0.5 * dst_bin_width) /
                dst_bin_width)))
22:             dst_bin_of_end = min(dst_nbins - 1, max(0,  oor((src_bin_end + 0.5 * dst_bin_width) /
                dst_bin_width)))
23:             dst_bin_of_begin_center = dst_bin_of_begin * dst_bin_width
24:             density = histogram[src_bin] / bin_width
25:             delta_begin = src_bin_begin - dst_bin_of_begin_center
26:             if dst_bin_of_begin == dst_bin_of_end then
27:                 delta_end = src_bin_end - dst_bin_of_begin_center
28:                 norm += get_l2_norm(delta_begin, delta_end, density)
29:             else
30:                 delta_end = dst_bin_width / 2
31:                 norm += get_l2_norm(delta_begin, delta_end, density)
32:                 norm  += (dst_bin_of_end  -  dst_bin_of_begin  -  1)  *  get_l2_norm(  -dst_bin_width  /  2,
                    dst_bin_width / 2, density)
33:                 dst_bin_of_end_center = dst_bin_of_end * dst_bin_width
34:                 delta_begin = -dst_bin_width / 2
35:                 delta_end = src_bin_end - dst_bin_of_end_center
36:                 norm += get_l2_norm(delta_begin, delta_end, density)
37:         if  norm < norm_minthen
38:             norm_min = norm
39:             best_start_bin = start_bin
40:             best_nbins_selected = nbins_selected
41: xmin = xmin + bin_width * best_start_bin
42: xmax = xmax + bin_width * (best_start_bin + best_nbins_selected)
43: return  xmin, xmax
```

Figure 2: Average time per row spent on 4-bit quantization. Time is shown in log10 scale.

Figure 2 shows the average time in milliseconds to quantize a row vector with different dimensions using 4 bits. To make a fair comparison, we implemented all the quantization algorithms in python. The results show that HIST-BRUTE is millions of times slower than ASYM. All the other clipping-based approaches take less than 100ms to quantize a row vector if its d is less than 2048. The times are measured on a computer with Ubuntu 16.04, 3.00GHz Intel Xeon CPU E5-1607 processor, and 8GB memory.

B Histograms After 4-bit Quantization

We show the histograms of a vector after 4-bit quantization using different approaches in Figure 3. The vector is of dimension 64 and its values are randomly sampled from a Gaussian distribution. The results echo the observations in Figure 1 that GREEDY and KMEANS have the smallest quantization error than state-of-the-art quantization approaches (HIST-APPRX, ACIQ, GSS).

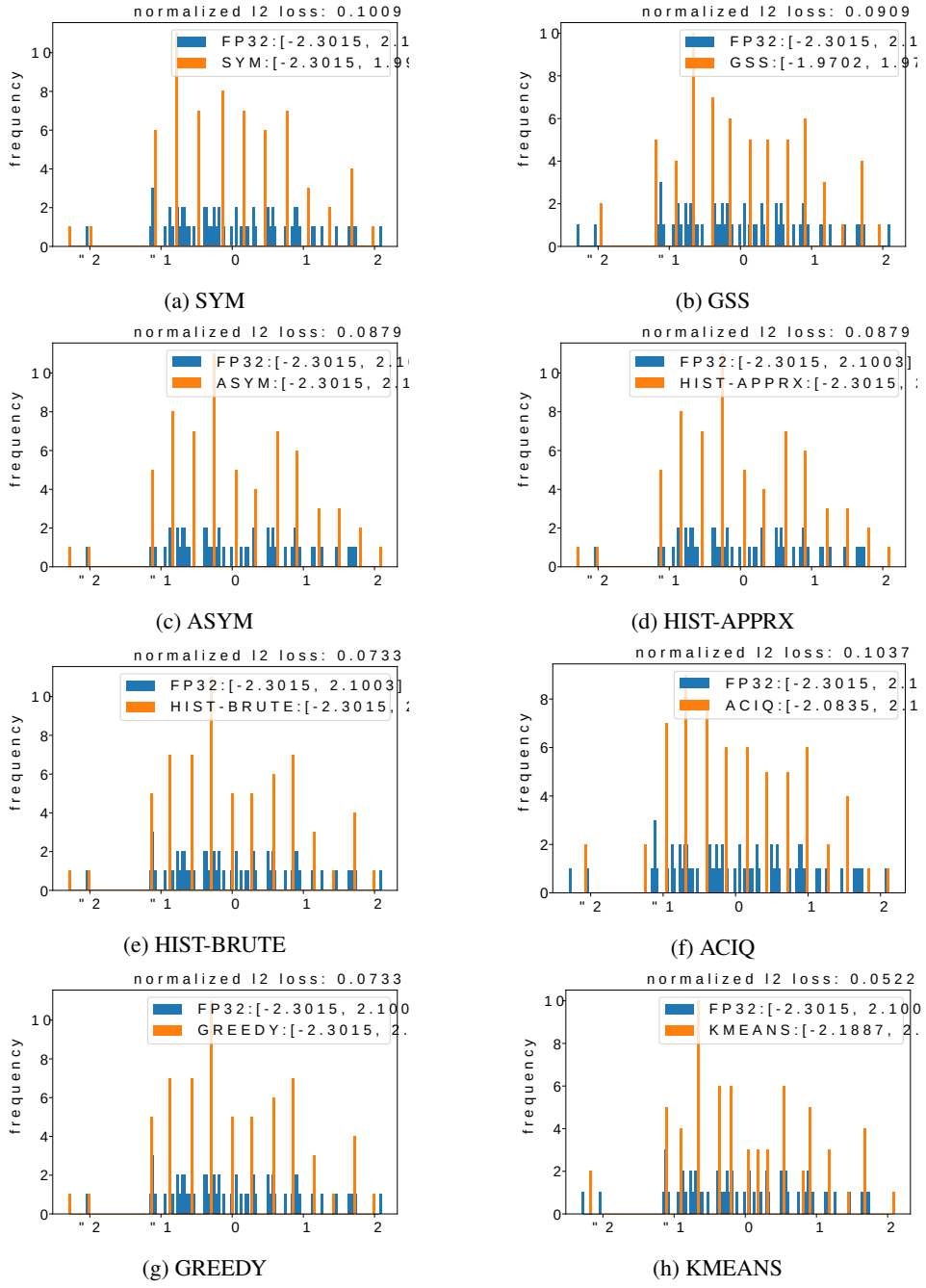

Figure 3: Histograms of a vector (d=64) before and after 4-bit quantization with different techniques. HIST-APPRX and HIST-BRUTE use $b = 200$, GREEDY uses $b = 200; r = 0.16$.