Hessian-aware Quantized Node Embeddings for Recommendation

Huiyuan Chen
hchen@visa.com
Visa Research
Palo Alto, CA, USA

Kaixiong Zhou
Kwei-Herng Lai
Kaixiong.Zhou@rice.edu
Rice University
Houston, TX, USA

Chin-Chia Michael Yeh
miyeh@visa.com
Visa Research
Palo Alto, CA, USA

Yan Zheng
yazheng@visa.com
Visa Research
Palo Alto, CA, USA

Xia Hu
xia.hu@rice.edu
Rice University
Houston, TX, USA

Hao Yang
haoyang@visa.com
Visa Research
Palo Alto, CA, USA

ABSTRACT

Graph Neural Networks (GNNs) have achieved state-of-the-art performance in recommender systems. Nevertheless, the process of searching and ranking from a large item corpus usually requires high latency, which limits the widespread deployment of GNNs in industry-scale applications. To address this issue, many methods compress user/item representations into the binary embedding space to reduce space requirements and accelerate inference. Also, they use the Straight-through Estimator (STE) to prevent vanishing gradients during back-propagation. However, the STE often causes the gradient mismatch problem, leading to sub-optimal results.

In this work, we present the Hessian-aware Quantized GNN (HQ-GNN) as an effective solution for discrete representations of users/items that enable fast retrieval. HQ-GNN is composed of two components: a GNN encoder for learning continuous node embeddings and a quantized module for compressing full-precision embeddings into low-bit ones. Consequently, HQ-GNN benefits from both lower memory requirements and faster inference speeds compared to vanilla GNNs. To address the gradient mismatch problem in STE, we further consider the quantized errors and its second-order derivatives for better stability. The experimental results on several large-scale datasets show that HQ-GNN achieves a good balance between latency and performance.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; • **Computing methodologies** → **Neural networks**.

KEYWORDS

Collaborative Filtering, Graph Neural Networks, Low-bit Quantization, Generalized Straight-Through Estimator

RecSys '23, September 18–22, 2023, Singapore, Singapore

ACM ISBN 979-8-4007-0241-9/23/09…$15.00
https://doi.org/10.1145/3604915.3608826

ACM Reference Format:
Huiyuan Chen, Kaixiong Zhou, Kwei-Herng Lai, Chin-Chia Michael Yeh, Yan Zheng, Xia Hu, and Hao Yang. 2023. Hessian-aware Quantized Node Embeddings for Recommendation. In *Seventeenth ACM Conference on Recommender Systems (RecSys '23), September 18–22, 2023, Singapore, Singapore.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3604915.3608826

1 INTRODUCTION

Recommender systems play an important role for e-commerce, such as display advertising and ranking products [5, 15]. Among different recommender models, Graph Neural Networks (GNNs) have achieved cutting-edge performance on top-k recommendations [14, 16, 30, 36]. For instance, Pinterest deploys a GNN model to train on a graph with 3 billion nodes and 18 billion edges, which has delivered state-of-the-art performance [36]. Despite the superior ability of GNNs, node representations are often stored in continuous embedding space (*e.g.*, 32-bit floating point (FP32)). This often requires huge memory consumption [23]. For example, the FP32 embeddings of 10 million items with a dimensional size of 256 will take up over 9.5 GB of storage space, which is hard to be deployed into devices with limited memory, especially under the federated learning settings [25, 37]. Therefore, searching and ranking from a large item corpus to generate top-k recommendations become intractable at scale due to their high latency [4, 26, 28, 29, 33].

Low-bit quantization [3, 12, 17, 21, 22] is a promising method to save the memory footprint and accelerate model inference for large-scale systems. By replacing FP32 values with lower precision values, e.g., 8-bit integer (INT8), quantization can shrink down the size of embeddings without modifying the original network architectures. Also, quantized operators are widely supported by modern hardwares, which allows to deploy very large networks to resource-limited devices [4, 17]. For example, NVIDIA Turing GPU architecture[1] supports the INT8 arithmetic operations.

Recently, several studies have adopted quantization in large-scale recommender systems [3, 20, 28, 32]. However, existing methods suffer from two drawbacks: 1) Most of them employ binary hash techniques to compress user/item embeddings into 1-bit quantized representations. Nevertheless, recent studies show that ultra low-bit quantizations (*e.g.*, 1 or 2 bits) can be much more challenging due to their significant degradation in the accuracy [12, 38]; 2)

[1] https://www.nvidia.com/en-us/geforce/turing/

They often use the Straight-through Estimator (STE) [2] to avoid zero gradients during the back-propagation. Specifically, the non-differentiable quantized function is replaced with a surrogate: the identity function [28] or the scaled tanh function [3, 20]. However, the use of different forward and backward functions results in a gradient mismatch problem, i.e., the modified gradient is certainly not the gradient of loss function, which makes the network training unstable [8, 35].

In this work, we propose the Hessian-aware Quantized GNN (HQ-GNN) for effective discrete representations of users and items for fast retrieval. Specifically, HQ-GNN consists of two components: a GNN encoder for learning continuous user/item embeddings, and a quantized module for compressing the full-precision embeddings into low-bit ones. Instead of 1-bit, HQ-GNN allows arbitrary bit quantization for better trade-offs between latency and performance. To address the gradient mismatch problem, we tailor the STE by further considering the quantized errors and second-order derivatives (e.g. Hessian) for better stability and accuracy. As such, HQ-GNN can benefit from both lower memory footprint and faster inference speed comparing to vanilla GNN. Experimental results on several large-scale datasets show the superiority of our HQ-GNN.

2 RELATED WORK

GNN-based Recommenders. GNNs have received a lot of attention in graph domains. GNNs learn how to aggregate messages from local neighbors using neural networks, which have been successfully applied to user-item bipartite graphs [6, 7, 14, 30, 31, 36]. Some representative models include PinSage [36], NGCF [30], LightGCN [14], etc. Although GNNs have great ability of capturing high-order collaborative signals between users and items, their node embeddings are stored in continuous space (*e.g.,* FP32), which is the major bottleneck for searching and ranking (e.g., high computational cost of similarity calculation between continuous embeddings). It is thus essential to improve the efficiency of generating top-k recommendations at scale [26, 28].

Network Quantizations. Quantization is a hardware-friendly approach by approximating real values with low-bit ones [3, 12, 17–19, 21, 22, 34]. Meanwhile, network inference can be performed using cheaper fixed-point multiple-accumulation operations. As a result, quantization can reduce the storage overhead and inference latency of networks [12, 22, 23, 38, 39]. In recommender systems, HashNet [3] proposes to binarize the embeddings by continuation method for multimedia retrieval. Similarly, CIGAR [20] learns binary codes to build a hash table for retrieving top-k item candidates. Recently, HashGNN [28] learns hash functions and graph representations in an end-to-end fashion. Our HQ-GNN builds on HashGNN. Specifically, we extend 1-bit quantization of HashGNN to arbitrary-bit one, and address the gradient mismatch issue of STE, resulting in better performance.

3 METHODOLOGY

3.1 Task Description

Generally, the input of recommender systems includes a set of users $\mathcal{U} = \{u\}$, items $\mathcal{I} = \{i\}$, and users' implicit feedback $\mathcal{O}^+ = \{(u, i) \mid u \in \mathcal{U}, i \in \mathcal{I}, y_{ui} = 1\}$, where $y_{ui} = 1$ indicates that user u has adopted item i before, $y_{ui} = 0$ otherwise. One can construct a corresponding bipartite graph $\mathcal{G} = (\mathcal{V} = \mathcal{U} \cup \mathcal{I}, \mathcal{E} = \mathcal{O}^+)$. The goal is to estimate the user preference towards unobserved items.

We next introduce our HQ-GNN that consists of two parts: a GNN encoder and a quantized module.

3.2 GNN-based Recommenders

Most GNNs fit under the message-passing schema [14, 30], where the representation of each node is updated by collecting messages from its neighbors via an aggregation operation $\mathrm{Agg}(\cdot)$ followed by an $\mathrm{Update}(\cdot)$ operation as:

$$
\begin{aligned}
\mathbf{e}_u^{(l)} &= \mathrm{Update}\left(\mathbf{e}_u^{(l-1)}, \mathrm{Agg}\left(\{\mathbf{e}_i^{(l-1)} \mid i \in \mathcal{N}_u\}\right)\right), \\
\mathbf{e}_i^{(l)} &= \mathrm{Update}\left(\mathbf{e}_i^{(l-1)}, \mathrm{Agg}\left(\{\mathbf{e}_u^{(l-1)} \mid u \in \mathcal{N}_i\}\right)\right),
\end{aligned}
\tag{1}
$$

where $\{\mathbf{e}_u^{(l)}, \mathbf{e}_i^{(l)}\} \in \mathbb{R}^d$ denote the embeddings of user and item in the l-th layer; $\mathcal{N}_u$ and $\mathcal{N}_i$ denote neighbors of user u and item i, respectively. By propagating L layer, a pooling operator is used to obtain the final representations:

$$
\mathbf{e}_u = \mathrm{Pool}(\mathbf{e}_u^{(0)}, \ldots, \mathbf{e}_u^{(L)}), \quad \mathbf{e}_i = \mathrm{Pool}(\mathbf{e}_i^{(0)}, \ldots, \mathbf{e}_i^{(L)}), \tag{2}
$$

where the final representations $\mathbf{e}_u \in \mathbb{R}^d$ and $\mathbf{e}_i \in \mathbb{R}^d$ can be used for downstream tasks. However, the full-precision embeddings, *e.g.*, FP32, usually require high memory cost and power consumption to generate top-k recommendations for the billion-scale graphs.

3.3 Low-bit Quantization

Quantization is a hardware-friendly technique to reduce memory footprint and energy consumption [13, 27, 39]. For a uniform b-bit quantization, one can clip and normalize a floating-point number x into a quantization interval, parameterized by an upper u and a lower l bounds, as:

$$
x_n = \frac{\mathrm{clip}(x, l, u) - l}{\Delta}, \tag{3}
$$

where x_n is the normalized output, $\mathrm{clip}(x, l, u) = \min(\max(x, l), u)$, $\Delta = \frac{u-l}{2^b-1}$ is the interval length, and b denotes the number of quantization levels, *e.g.*, $b = 8$ for 8-bit quantization. During training, the clipping interval (l, u) is often unknown beforehand, two strategies are commonly used to determine the upper/lower thresholds: exponential moving averages [17] and treating the thresholds as learnable parameters [9]. The normalized output x_n can be then converted to a discrete value x_b using a round function with post-scaling as [12, 38, 39]:

$$
x_b = x_q \cdot \Delta, \quad x_q = \mathrm{round}(x_n), \tag{4}
$$

where $\mathrm{round}(\cdot)$ maps a full-precision value to its nearest integer. The quantized tensor x_b can be then used for efficient computation by emergent accelerators (*e.g.*, NVIDIA TensorRT) that are able to handle Δ efficiently.

By combining Eq. (3) and Eq. (4), we can defined a quantization function $Q_b(\cdot)$ as: $x_b = Q_b(x)$. If the input is a vector/matrix, $Q_b(\cdot)$ would apply to each element of the vector/matrix. To this end, we can quantize the GNN embeddings $\mathbf{e}_u$ and $\mathbf{e}_i$ in Eq. (2) into:

$$
\mathbf{q}_u = Q_b(\mathbf{e}_u), \quad \mathbf{q}_i = Q_b(\mathbf{e}_i), \tag{5}
$$

where $\{\mathbf{q}_u, \mathbf{q}_i\} \in \mathbb{R}^d$ are the b-bit representations of user u and item i, respectively. Our model follows the mixed-precision quantization policy [24], where we only compress the *activations* of GNNs for faster inference, and leave the *weights* of GNNs at full precision. Since GNNs often contain less than three layers and have limited weights, the mixed-precision scheme could achieve good trade-offs between performance and memory size [11]. The mixed-precision quantization has also become more and more common in deep learning frameworks[2].

However, the non-differentiable quantized processes are undesirable for the standard back-propagation, i.e., the quantization function is intrinsically a discontinuous step function and nearly has zero gradients, which significantly affects the training of HQ-GNN. We next present a Generalized Straight-Through Estimator to address this problem.

3.4 Generalized Straight-Through Estimator

The main challenge of training our HQ-GNN arises from the discretized round function in Eq. (4), where its derivative is either infinite or zero at almost everywhere. One popular family of estimators are the so-called Straight-Through Estimators (STE) [2, 35]. In STE, the forward computation of round$(\cdot)$ is unchanged, but back-propagation is computed through a surrogate [3, 28, 38]: replacing round$(\cdot)$ with an identity function, *i.e.*, $\mathcal{G}_{\mathbf{x}_\mathbf{n}} = \mathcal{G}_{\mathbf{x}_\mathbf{q}}$ where $\mathcal{G}$ denotes the gradient operator. However, STE runs the risk of convergence to poor minima and unstable training [35]. For example, both values of 0.51 and 1.49 round to same integer 1 with different quantized errors. Moreover, STE forces to update both values equally with the same gradient at integer 1, which is likely to be biased with cumulative quantized errors. Moreover, a small decrement (*e.g.*, −0.2) for value 0.51 can largely change the quantized integer from 1 to 0, while a same decrement to 1.49 cannot.

To mitigate the impact of quantized errors, we generalize the STE as [22]:

$$\mathcal{G}_{\mathbf{x}_\mathbf{n}} = \mathcal{G}_{\mathbf{x}_\mathbf{q}} \odot \left(1 + \delta \cdot \mathrm{sign}(\mathcal{G}_{\mathbf{x}_\mathbf{q}}) \odot (\mathbf{x}_\mathbf{n} - \mathbf{x}_\mathbf{q})\right), \tag{6}$$

where $\odot$ denotes element-wise product; sign$(\cdot)$ is a sign function such that sign$(x) = +1$ if $x \geq 0$, -1 otherwise; δ is the scaling factor. Eq. (6) is able to scale up/down the gradient of $\mathcal{G}_{\mathbf{x}_\mathbf{q}}$ when the $\mathbf{x}_\mathbf{n}$ requires a larger/smaller magnitude for an update. Moreover, Eq. (6) is equivalent to vanilla STE when setting $\delta = 0$. It is thus crucial to determine the scaling factor δ during training.

Inspired by Hessian-aware quantized networks [10, 11], we use second-order information to guide the selection of δ. Let $\epsilon = \mathbf{x}_\mathbf{n} - \mathbf{x}_\mathbf{q}$ denote the quantized error for round function, where each element of ϵ is well bound by a small number, *i.e.*, $|\epsilon_i| \leq \frac{0.5}{2^b - 1}$, with element-wise Taylor expansion, we have:

$$\begin{aligned}
\mathcal{G}_{\mathbf{x}_\mathbf{n}} &= \mathcal{G}_{\mathbf{x}_\mathbf{q}} + \frac{\mathcal{G}_{\mathbf{x}_\mathbf{n}} - \mathcal{G}_{\mathbf{x}_\mathbf{q}}}{\mathbf{x}_\mathbf{n} - \mathbf{x}_\mathbf{q}} \odot (\mathbf{x}_\mathbf{n} - \mathbf{x}_\mathbf{q}) \\
&= \mathcal{G}_{\mathbf{x}_\mathbf{q}} + \frac{\mathcal{G}_{\mathbf{x}_\mathbf{q}+\epsilon} - \mathcal{G}_{\mathbf{x}_\mathbf{q}}}{\epsilon} \odot (\mathbf{x}_\mathbf{n} - \mathbf{x}_\mathbf{q}) \\
&\approx \mathcal{G}_{\mathbf{x}_\mathbf{q}} + \mathcal{G}'_{\mathbf{x}_\mathbf{q}} \odot (\mathbf{x}_\mathbf{n} - \mathbf{x}_\mathbf{q}),
\end{aligned}$$

[2]https://www.tensorflow.org/guide/mixed_precision

where $\frac{[\cdot]}{[\cdot]}$ is the element-wise division, $\mathcal{G}'_{\mathbf{x}_\mathbf{q}} = \frac{\partial \mathcal{G}_{\mathbf{x}_\mathbf{q}}}{\partial \mathbf{x}_\mathbf{q}}$ denotes the second-order derivative of a task loss with respect to $\mathbf{x}_\mathbf{q}$. The above equation can be represented as:

$$\mathcal{G}_{\mathbf{x}_\mathbf{n}} \approx \mathcal{G}_{\mathbf{x}_\mathbf{q}} \odot \left(1 + \frac{\mathcal{G}'_{\mathbf{x}_\mathbf{q}}}{|\mathcal{G}_{\mathbf{x}_\mathbf{q}}|} \odot \mathrm{sign}(\mathcal{G}_{\mathbf{x}_\mathbf{q}}) \odot (\mathbf{x}_\mathbf{n} - \mathbf{x}_\mathbf{q})\right), \tag{7}$$

where $|\cdot|$ denotes the absolute value. Comparing Eq. (6) and Eq. (7) suggests that we can connect δ with $\frac{\mathcal{G}'_{\mathbf{x}_\mathbf{q}}}{|\mathcal{G}_{\mathbf{x}_\mathbf{q}}|}$, but explicitly forming the Hessian matrix $\mathbf{H}$ (containing all $\mathcal{G}'_{\mathbf{x}_\mathbf{q}}$) is computationally infeasible in practice. Instead, recent quantized networks approximate the second-order information by the average Hessian Trace [10] or top Hessian eigenvalues [11]. In this work, we summarize the average trace of Hessian and $\frac{\mathcal{G}'_{\mathbf{x}_\mathbf{q}}}{|\mathcal{G}_{\mathbf{x}_\mathbf{q}}|}$ as scaling factor:

$$\delta = \frac{\mathrm{Tr}(\mathbf{H})/N}{G}, \tag{8}$$

where N is the number of diagonal elements in $\mathbf{H}$ and G is an average over the absolute values of gradients, *i.e.*, $\mathbb{E}[|\mathcal{G}_{\mathbf{x}_\mathbf{q}}|]$.

Algorithm 1: HQ-GNN

Input: A GNN f_{gnn}, bipartite graph $\mathbf{A}$, bit-width b, regularizer α.
Output: Model parameters $\mathbf{\Theta}$ of f_{gnn};

```
Initialize Θ ;
for each mini-batch do
    /* Forward pass                                   */
    Compute node embeddings e_u and e_i by Eq. (2);
    Normalize outputs ê_u = (clip(e_u, l, u) − l) / Δ (same for ê_i);
    Quantize values ē_u = round(ê_u) (same for ē_i);
    Post-scaling quantized values q_u = ē_u ⊙ Δ (same for q_i);
    Compute the BPR loss by Eq. (9);
    /* Backward propagation                           */
    Compute the gradients G_{ē_u} and G_{ē_i} via standard SGD;
    Adjust the gradients G_{ê_u} and G_{ê_i} by Eq. (6):
        G_{ê_u} = G_{ē_u} ⊙ (1 + δ · sign(G_{ē_u}) ⊙ (ê_u − ē_u)),
        G_{ê_i} = G_{ē_i} ⊙ (1 + δ · sign(G_{ē_i}) ⊙ (ê_i − ē_i)).
    Compute the trace of Hessian by Hutchinson method [1];
    Update GNN parameters Θ and the scaling factor δ by Eq. (8);
end
return Θ
```

We compute the trace of Hessian via Hutchinson's method [1] Given a random vector $\mathbf{v}$, whose elements are i.i.d. sampled from a Rademacher distribution such that $\mathbb{E}[\mathbf{v}\mathbf{v}^\top] = \mathbf{I}$. Then, we have:

$$\begin{aligned}
\mathrm{Tr}(\mathbf{H}) &= \mathrm{Tr}(\mathbf{H}\mathbb{E}[\mathbf{v}\mathbf{v}^\top]) = \mathbb{E}[\mathrm{Tr}(\mathbf{H}\mathbf{v}\mathbf{v}^\top)] \\
&= \mathbb{E}[\mathbf{v}^\top \mathbf{H}\mathbf{v}] \approx \frac{1}{m}\sum_{i=1}^{m}(\mathbf{v}^{(i)\top}\mathbf{H}\mathbf{v}^{(i)}),
\end{aligned}$$

where $\mathbf{I}$ is the identity matrix. The trace of $\mathbf{H}$ can be estimated by $\mathbb{E}[\mathbf{v}^\top\mathbf{H}\mathbf{v}]$, where the expectation can be obtained by drawing m random vectors. Note that we can first compute $\mathbf{H}\mathbf{v}$, then $\mathbf{v}^\top\mathbf{H}\mathbf{v}$ is a simple inner product between $\mathbf{v}$ and $\mathbf{H}\mathbf{v}$. Also, we can obtain $\mathbf{H}\mathbf{v}$ efficiently without computing an exact Hessian matrix as follows:

$$\frac{\partial(\mathcal{G}^\top_{\mathbf{x}_\mathbf{q}}\mathbf{v})}{\partial \mathbf{x}_\mathbf{q}} = \frac{\partial \mathcal{G}^\top_{\mathbf{x}_\mathbf{q}}}{\partial \mathbf{x}_\mathbf{q}}\mathbf{v} + \mathcal{G}^\top_{\mathbf{x}_\mathbf{q}}\frac{\partial \mathbf{v}}{\partial \mathbf{x}_\mathbf{q}} = \frac{\partial \mathcal{G}^\top_{\mathbf{x}_\mathbf{q}}}{\partial \mathbf{x}_\mathbf{q}}\mathbf{v} = \mathbf{H}\mathbf{v},$$

Table 1: Dataset statistics.

Dataset	Gowalla	Yelp2018	Amazon-Book	Alibaba
|User|	29,858	31,668	52,643	106,042
|Item|	40,981	38,048	91,599	53,591
|Interaction|	1,027,370	1,561,406	2,984,108	907,407

where the first equality is the chain rule, while the second is due to the independence of $\mathbf{v}$ and $\mathbf{x_q}$. As such, the cost of Hessian matrix-vector multiply is the same as one gradient back-propagation.

3.5 Model Optimization

3.5.1 Loss function. Based on the b-bit representations $\mathbf{q}_u$ and $\mathbf{q}_i$ from Eq. (5), we can adopt the inner product to estimate the user's preference towards the target item as: $\hat{y}_{ui} = \langle \mathbf{q}_u, \mathbf{q}_i \rangle$. Also, we use Bayesian Personalized Ranking loss to optimize the model [20]:

$$\mathcal{L}_{BPR}(\Theta) = \sum_{(u,i)\in O^+,(u,j)\notin O^+} -\ln \sigma\left(\hat{y}_{ui} - \hat{y}_{uj}\right) + \alpha\|\Theta\|_F^2, \quad (9)$$

where $\sigma(\cdot)$ denotes the sigmoid function, Θ denotes the model parameters of GNNs, and α controls the L_2 regularization strength. Finally, we briefly summarize our HQ-GNN in Algorithm 1.

3.5.2 Complexity. Compared to vanilla GNN, HQ-GNN has an extra time cost to perform gradient adjustments in Eq. (6). The computation of Hessian Trace only requires one gradient back-propagation, which is significantly faster than training the GNN encoder itself [10]. Thus, HQ-GNN has the same training complexity as its GNN encoder. However, during the inference, we can use integer-only node embeddings (without post-scaling) to generate the top-k candidates, which has both lower memory footprint and faster inference speed compared to the vanilla GNN.

4 EXPERIMENTS

4.1 Experimental Settings

4.1.1 ***Datasets.*** We evaluate our method on four public datasets [14, 16, 30]: Gowalla[3], Yelp-2018[4], Amazon-book[5], and Alibaba[6]. Their statistics are summarized in Table 1. For each dataset, we randomly select 80% of historical interactions of each user to construct the training set, and treat the remaining as the test set. From the training set, we randomly select 10% of interactions as the validation set to tune the hyper-parameters.

4.1.2 ***Baselines and Evaluations.*** To verify the effectiveness of HQ-GNN, we mainly compare with graph-based models: NGCF [30], LightGCN [14], HashNet [3] and HashGNN [28]. For HashNet, HashGNN and HQ-GNN, we can choose any GNN encoder to compute the continuous node embeddings in Eq. (2). The comparison against other methods (*e.g.*, factorization machines) is omitted, since most of them are outperformed by LightGCN. We choose the widely-used Recall@k and NDCG@k as the evaluation metrics [14, 16, 30]. We simply set $k = 50$ in all experiments [28].

[3]https://snap.stanford.edu/data/loc-gowalla.html
[4]https://www.yelp.com/dataset
[5]https://jmcauley.ucsd.edu/data/amazon/
[6]https://github.com/huangtinglin/MixGCF/tree/main/data/ali

4.1.3 ***Implementation Details.*** For all baselines, the embedding size of user/item is searched among $\{16, 32, 64, 128\}$. The hyper-parameters (*e.g.*, batch size, learning rate) of baselines are initialized as their original settings and are then carefully tuned to achieve the optimal performance. For HQ-GNN, we search L_2 regularizer α within $\{10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$. In addition, we determine the upper/lower thresholds (Eq. (3)) by exponential moving averages [17], and set the number of bits $b = 1$ in Eq. (5) for fair comparisons with binary hash methods: HashNet [3] and HashGNN [28].

4.2 Experimental Results

4.2.1 ***Overall Performance.*** We present a comprehensive performance comparison between full-precision GNNs and quantization-aware GNNs. We summarize the results in terms of Recall@50 and NDCG@50 for different datasets in Table 2. From the table, we have two major observations: 1) Among all 1-bit GNNs, our proposed HQ-GNN consistently outperforms both HashNet and HashGNN by a large margin on all four datasets. Clearly, this reveals that our HQ-GNNs provide a meaningful gradient adjustments for non-differentiable quantized function. For example, for LightGCN encoder, HQ-GNN has on average 15.80% improvement with respect to Recall@50 and over 15.63% improvement with respect to NDCG@50, comparing to the state-of-the-art HashGNN. 2) It is not surprised that full-precision GNNs perform better than quantization-aware GNNs in all cases. However, quantization-aware GNNs benefit from both lower memory footprint and faster inference speed comparing to vanilla GNN.

In terms of memory and inference speed, we have observed similar results as those reported in HashNet [3] and HashGNN [28]. This is because our HQ-GNN, with $b = 1$, inherits all the benefits of HashGNN. For instance, using binarized embeddings (1 bit) can significantly reduce memory usage as compared to using FP32 embeddings. Moreover, the inference speed of our HQ-GNNs is approximately 3.6 times faster than that of full-precision GNNs because the Hamming distance between two binary embeddings can be calculated efficiently [28]. These features make our HQ-GNN more desirable for large-scale retrieval applications in the industry.

4.2.2 ***Compared to GTE.*** The STE method propagates the same gradient from an output to an input of the discretizer, assuming that the derivative of the discretizer is equal to 1. In contrast, our GSTE method adopts the Hessian to refine the gradients. To evaluate the effectiveness of our GSTE method, we chose LightGCN as the backbone and quantized its embeddings into 1 bit. The performance on different datasets is summarized in Table 3. From the table, it is clear that our GSTE method performs better than STE for 1-bit quantization, with improvements ranging from 14.7% to 24.5%.

Regarding running time, during the training stage, our GSTE method requires computing the trace of Hessian using Hutchinson's method, which is however fast. From Table 3, we can see that our GSTE method is slightly slower than STE, which is negligible in practice. During inference, both our GSTE and STE methods have the same speed as both use 1-bit quantized embeddings for retrieval, and the trace of Hessian is not needed in the inference stage.

The left of Figure 1 also displays the training curves of GSTE and STE, and we clearly observe that training quantized LightGCN with GSTE is better than STE in terms of stability. This highlights

Table 2: Performance comparison (bold and underline represent the best full-precision and 1-bit quantized models).

	Gowalla		Yelp-2018		Amazon-Book		Alibaba	
Methods	Recall@50	NDCG@50	Recall@50	NDCG@50	Recall@50	NDCG@50	Recall@50	NDCG@50
NGCF	**0.159**	**0.130**	**0.114**	**0.054**	**0.092**	**0.065**	**0.071**	**0.033**
+HashNet	0.104	0.082	0.071	0.030	0.057	0.038	0.047	0.021
+HashGNN	0.122	0.098	0.091	0.042	0.073	0.043	0.054	0.023
+HQ-GNN	0.145	0.112	0.101	0.048	0.081	0.054	0.065	0.029
LightGCN	**0.163**	**0.134**	**0.118**	**0.059**	**0.098**	**0.072**	**0.076**	**0.036**
+HashNet	0.113	0.088	0.074	0.036	0.064	0.041	0.052	0.024
+HashGNN	0.128	0.112	0.094	0.047	0.075	0.053	0.062	0.029
+HQ-GNN	0.152	0.122	0.108	0.051	0.089	0.062	0.070	0.032

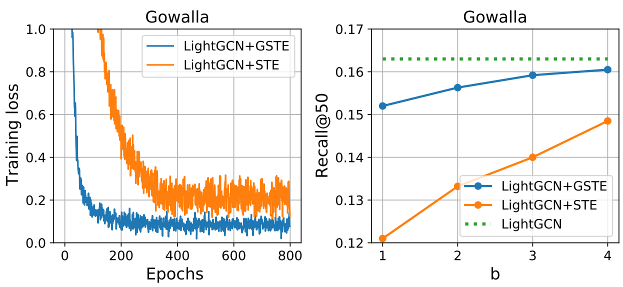

Figure 1: Left: GSTE vs. STE over training loss. Right: the impact of the number of bits in the HQ-GNN.

the effectiveness of utilizing Hessian information in the training process. The right of Figure 1 shows the impact of quantization levels by varying b within $\{1, 2, 3, 4\}$ for both GSTE and STE. As can be seen, aggressive quantization (less than 2-bit precision) can lead to significant degradation in the accuracy. When $b = 4$, HQ-GNN obtains 98.5% performance recovery of LightGCN. Comparing STE and GSTE, our GSTE consistently performance better than STE in all cases. In summary, HQ-GNN strikes a good balance between latency and performance.

5 CONCLUSION

Training graph neural networks on large-scale user-item bipartite graphs has been a challenging task due to the extensive memory requirement. To address this problem, we propose HQ-GNN that explores the issue of low-bit quantization of graph neural networks for large-scale recommendations. Additionally, we introduce a Generalized Straight-Through Estimator to solve the gradient mismatch problem that arises during the training of quantized networks. HQ-GNN is flexible and can be applied to various graph neural networks. The effectiveness of our proposed method is demonstrated through extensive experiments on real-world datasets.

REFERENCES

[1] Haim Avron and Sivan Toledo. 2011. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *J. ACM* (2011), 1–34.

[2] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* (2013).

[3] Zhangjie Cao, Mingsheng Long, Jianmin Wang, and Philip S Yu. 2017. Hashnet: Deep learning to hash by continuation. In *Proceedings of the IEEE international conference on computer vision*. 5608–5617.

[4] Huiyuan Chen, Xiaoting Li, Kaixiong Zhou, Xia Hu, Chin-Chia Michael Yeh, Yan Zheng, and Hao Yang. 2022. TinyKG: Memory-Efficient Training Framework for Knowledge Graph Neural Recommender Systems. In *Proceedings of the 16th ACM Conference on Recommender Systems*. 257–267.

[5] Huiyuan Chen, Yusan Lin, Fei Wang, and Hao Yang. 2021. Tops, bottoms, and shoes: building capsule wardrobes via cross-attention tensor network. In *Proceedings of the 15th ACM Conference on Recommender Systems*. 453–462.

[6] Huiyuan Chen, Chin-Chia Michael Yeh, Fei Wang, and Hao Yang. 2022. Graph neural transport networks with non-local attentions for recommender systems. In *Proceedings of the ACM Web Conference 2022*. 1955–1964.

[7] Huiyuan Chen, Kaixiong Zhou, Kwei-Herng Lai, Xia Hu, Fei Wang, and Hao Yang. 2022. Adversarial graph perturbations for recommendations at scale. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1854–1858.

[8] Yankai Chen, Huifeng Guo, Yingxue Zhang, Chen Ma, Ruiming Tang, Jingjie Li, and Irwin King. 2022. Learning binarized graph representations with multi-faceted quantization reinforcement for top-k recommendation. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 168–178.

[9] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. 2018. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085* (2018).

[10] Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2020. Hawq-v2: Hessian aware trace-weighted quantization of neural networks. *Advances in neural information processing systems*.

[11] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2019. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 293–302.

[12] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. 2019. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 4852–4861.

[13] Song Han, Huizi Mao, and William J Dally. 2016. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *International Conference on Learning Representations*.

[14] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 639–648.

[15] Jui-Ting Huang, Ashish Sharma, Shuying Sun, Li Xia, David Zhang, Philip Pronin, Janani Padmanabhan, Giuseppe Ottaviano, and Linjun Yang. 2020. Embedding-based retrieval in facebook search. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2553–2561.

[16] Tinglin Huang, Yuxiao Dong, Ming Ding, Zhen Yang, Wenzheng Feng, Xinyu Wang, and Jie Tang. 2021. MixGCF: An Improved Training Method for Graph Neural Network-Based Recommender systems. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 665–674.

[17] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2704–2713.

[18] Gangwei Jiang, Hao Wang, Jin Chen, Haoyu Wang, Defu Lian, and Enhong Chen. 2021. xLightFM: Extremely Memory-Efficient Factorization Machine. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 337–346.

Table 3: The performance and the running time of 1-bit quantized LightGCN with STE and GSTE.

	Gowalla		Yelp-2018		Amazon-Book		Alibaba	
LightGCN	Recall@50	Time(sec)	Recall@50	Time(sec)	Recall@50	Time(sec)	Recall@50	Time(sec)
+STE	0.122	30.4	0.092	41.7	0.074	103.6	0.061	22.2
+GSTE	0.152	32.9	0.108	45.1	0.089	110.7	0.070	23.9
Improv(%)	+24.5%	-	+17.3%	-	+20.2%	-	+14.7%	-

[19] Yongcheng Jing, Yiding Yang, Xinchao Wang, Mingli Song, and Dacheng Tao. 2021. Meta-Aggregator: Learning to Aggregate for 1-bit Graph Neural Networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 5301–5310.

[20] Wang-Cheng Kang and Julian McAuley. 2019. Candidate generation with binary codes for large-scale top-n recommendation. In *Proceedings of the 28th ACM international conference on information and knowledge management*. 1523–1532.

[21] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2021. I-bert: Integer-only bert quantization. In *International conference on machine learning*. 5506–5518.

[22] Junghyup Lee, Dohyung Kim, and Bumsub Ham. 2021. Network Quantization with Element-wise Gradient Scaling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 6448–6457.

[23] Defu Lian, Haoyu Wang, Zheng Liu, Jianxun Lian, Enhong Chen, and Xing Xie. 2020. Lightrec: A memory and search-efficient recommender system. In *Proceedings of The Web Conference 2020*. 695–705.

[24] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. Mixed Precision Training. In *International Conference on Learning Representations*.

[25] Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. 2020. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *International Conference on Artificial Intelligence and Statistics*.

[26] Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. 2020. Compositional embeddings using complementary partitions for memory-efficient recommendation systems. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 165–175.

[27] Xiao Sun, Naigang Wang, Chia-Yu Chen, Jiamin Ni, Ankur Agrawal, Xiaodong Cui, Swagath Venkataramani, Kaoutar El Maghraoui, Vijayalakshmi Viji Srinivasan, and Kailash Gopalakrishnan. 2020. Ultra-low precision 4-bit training of deep neural networks. *Advances in Neural Information Processing Systems*.

[28] Qiaoyu Tan, Ninghao Liu, Xing Zhao, Hongxia Yang, Jingren Zhou, and Xia Hu. 2020. Learning to Hash with Graph Neural Networks for Recommender Systems. In *Proceedings of The Web Conference 2020*. 1988–1998.

[29] Song Wang, Xingbo Fu, Kaize Ding, Chen Chen, Huiyuan Chen, and Jundong Li. 2023. Federated Few-shot Learning. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

[30] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*. 165–174.

[31] Yu Wang, Yuying Zhao, Yushun Dong, Huiyuan Chen, Jundong Li, and Tyler Derr. 2022. Improving fairness in graph neural networks via mitigating sensitive attribute leakage. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1938–1948.

[32] Wei Wu, Bin Li, Chuan Luo, and Wolfgang Nejdl. 2021. Hashing-accelerated graph neural networks for link prediction. In *Proceedings of the Web Conference 2021*. 2910–2920.

[33] Zhe Xu, Yuzhong Chen, Menghai Pan, Huiyuan Chen, Mahashweta Das, and Hao Yang. 2023. Kernel Ridge Regression-Based Graph Dataset Distillation. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

[34] Chin-Chia Michael Yeh, Mengting Gu, Yan Zheng, Huiyuan Chen, Javid Ebrahimi, Zhongfang Zhuang, Junpeng Wang, Liang Wang, and Wei Zhang. 2022. Embedding Compression with Hashing for Efficient Representation Learning in Large-Scale Graph. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4391–4401.

[35] Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley J. Osher, Yingyong Qi, and Jack Xin. 2019. Understanding Straight-Through Estimator in Training Activation Quantized Neural Nets. In *International Conference on Learning Representations*.

[36] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 974–983.

[37] Wei Yuan, Hongzhi Yin, Fangzhao Wu, Shijie Zhang, Tieke He, and Hao Wang. 2023. Federated unlearning for on-device recommendation. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*. 393–401.

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

[39] Feng Zhu, Ruihao Gong, Fengwei Yu, Xianglong Liu, Yanfei Wang, Zhelong Li, Xiuqi Yang, and Junjie Yan. 2020. Towards unified int8 training for convolutional neural network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1969–1979.