Quaternion Graph Neural Networks

Dai Quoc Nguyen
Monash University, Australia
dai.nguyen@monash.edu

Tu Dinh Nguyen
nguyendinhtu@gmail.com

Dinh Phung
Monash University, Australia
dinh.phung@monash.edu

Abstract

Recently, graph neural networks (GNNs) become a principal research direction to learn low-dimensional continuous embeddings of nodes and graphs to predict node and graph labels, respectively. However, Euclidean embeddings have high distortion when using GNNs to model complex graphs such as social networks. Furthermore, existing GNNs are not very efficient with the high number of model parameters when increasing the number of hidden layers.

Therefore, we move beyond the Euclidean space to a hyper-complex vector space to improve graph representation quality and reduce the number of model parameters. To this end, we propose quaternion graph neural networks (QGNN) to generalize GCNs within the Quaternion space to learn quaternion embeddings for nodes and graphs. The Quaternion space, a hyper-complex vector space, provides highly meaningful computations through Hamilton product compared to the Euclidean and complex vector spaces. As a result, our QGNN can reduce the model size up to four times and enhance learning better graph representations. Experimental results show that the proposed QGNN produces state-of-the-art accuracies on a range of well-known benchmark datasets for three downstream tasks, including graph classification, semi-supervised node classification, and text (node) classification. Our code is available at: https://github.com/daiquocnguyen/QGNN.

1 Introduction

Graph representation learning is one of the most important topics for graph-structured data [10, 47, 37, 44], where a goal is to learn vector embeddings for nodes and graphs. Recently, graph neural networks (GNNs) become an essential strand to learn low-dimensional continuous embeddings of nodes and graphs to predict node and graph labels, respectively [29, 14]. GNNs use an AGGREGATION function [14, 9, 35, 21] over neighbors of each node to update its vector representation iteratively, and then apply a graph-level READOUT pooling function to obtain graph embeddings [7, 45, 43, 36, 39]. We note that the GNNs have been producing state-of-the-art accuracy performances for node and graph classification tasks.

Nevertheless, as discussed in [28, 15, 2], Euclidean embeddings have high distortion when modeling complex graphs with scale-free and hierarchical structures such as protein interaction networks and social networks. It also has been noted in [39, 27] that the Euclidean embeddings of different nodes (or different graphs) can become increasingly more similar when constructing multiple GNN layers, hence degrading the graph representation quality. Furthermore, existing GNNs [14, 9, 35, 39] are not very efficient with the high number of model parameters when using more hidden layers.

While it has been considered under other contexts, in this paper, we address the following research question: Can we move beyond the Euclidean space to enhance learning better graph representations and reduce the number of model parameters?

To this end, we present an effective approach to learn node and graph embeddings within the Quaternion space – a hyper-complex vector space, where each quaternion is a hyper-complex number

NeurIPS 2020 workshop on Differential Geometry meets Deep Learning.

consisting of one real and three separate imaginary components. Some quaternion-based methods have been applied in image classification [6, 48], speech recognition [26, 25] and knowledge graph [46]. The closely related work is applying quaternion networks for natural language processing [34]. However, to the best of our knowledge, our work is the first to investigate quaternion embeddings for general graphs with diverse and different structures, requiring a different solution approach.

In general, our strategy can be flexibly applied to existing AGGREGATION functions such as in Graph Convolutional Network (GCN) [14] and Graph Isomorphism Network (GIN-0) [39]. Note that GCN is one of the current state-of-the-art GNNs for the semi-supervised node classification task. As noted in [39, 3], GCN also outperforms GraphSAGE [9] and GAT [35], and produces competitive accuracies compared to other up-to-date GNN models [39, 4] for the graph classification task. Therefore, we propose our quaternion graph neural networks (QGNN) to generalize GCNs within the Quaternion space.

The Quaternion space allows highly expressive computations through Hamilton product compared to the Euclidean and complex vector spaces, by sharing the inputs' quaternion components during multiplication. This characteristic helps to reduce the number of model parameters up to four times. Besides, any slight change in any of the quaternion input components results in an entirely different output [24], leading to a different performance. This innovation enhances capturing potential relations within each hidden quaternion layer and between the different hidden layers to increase the embedding quality. Our proposed QGNN has demonstrated to achieve superior performances through extensive experimental evaluation, making state-of-the-art performances on a wide range of benchmark datasets for three downstream tasks.

2 Quaternion Graph Neural Networks

2.1 Quaternion background

A quaternion $q \in \mathbb{H}$ is a hyper-complex number consisting of one real and three separate imaginary components [11] defined as: $q = q_r + q_i\mathbf{i} + q_j\mathbf{j} + q_k\mathbf{k}$ where $q_r, q_i, q_j, q_k \in \mathbb{R}$, and $\mathbf{i}, \mathbf{j}, \mathbf{k}$ are imaginary units that $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$. Correspondingly, a n-dimensional quaternion vector $\boldsymbol{q} \in \mathbb{H}^n$ is defined as: $\boldsymbol{q} = \boldsymbol{q}_r + \boldsymbol{q}_i\mathbf{i} + \boldsymbol{q}_j\mathbf{j} + \boldsymbol{q}_k\mathbf{k}$ where $\boldsymbol{q}_r, \boldsymbol{q}_i, \boldsymbol{q}_j, \boldsymbol{q}_k \in \mathbb{R}^n$.

Hamilton product. The Hamilton product $\otimes$ of two quaternions q and p is defined as:

$$\begin{aligned} q \otimes p &= (q_r p_r - q_i p_i - q_j p_j - q_k p_k) + (q_i p_r + q_r p_i - q_k p_j + q_j p_k)\mathbf{i} \\ &+ (q_j p_r + q_k p_i + q_r p_j - q_i p_k)\mathbf{j} + (q_k p_r - q_j p_i + q_i p_j + q_r p_k)\mathbf{k} \end{aligned} \tag{1}$$

Concatenation. In our approach, we further define a concatenation of two quaternion vectors $\boldsymbol{q}$ and $\boldsymbol{p}$ as: $[\boldsymbol{q}; \boldsymbol{p}] = [\boldsymbol{q}_r; \boldsymbol{p}_r] + [\boldsymbol{q}_i; \boldsymbol{p}_i]\,\mathbf{i} + [\boldsymbol{q}_j; \boldsymbol{p}_j]\,\mathbf{j} + [\boldsymbol{q}_k; \boldsymbol{p}_k]\,\mathbf{k}$

2.2 The proposed QGNN

Existing GNNs [14, 9, 35, 39] are not very efficient with the explosive number of parameters when increasing the number of hidden layers. Furthermore, as mentioned in [39, 27], the Euclidean embeddings of the different nodes (or the different graphs) become more and more similar when we increase the number of layers. This is partially explained in [28, 15, 2] that the Euclidean embeddings have high distortion when modeling the complex graphs with scale-free and hierarchical structures such as social networks. This motivates us to consider the hyper-complex vector space and propose QGNN, a generalized variant of GCNs within the Quaternion space, to deal with these issues.

As illustrated in Figure 2, the AGGREGATION function in our proposed QGNN is defined as:

$$\mathbf{h}_{\mathsf{v}}^{(l+1),Q} = \mathsf{g}\left(\sum_{\mathsf{u} \in \mathcal{N}_{\mathsf{v}} \cup \{\mathsf{v}\}} a_{\mathsf{v},\mathsf{u}} \mathbf{W}^{(l),Q} \otimes \mathbf{h}_{\mathsf{u}}^{(l),Q}\right), \forall \mathsf{v} \in \mathcal{V} \tag{2}$$

where we use the superscript ^{Q} to denote the Quaternion space; $a_{\mathsf{v},\mathsf{u}}$ is an edge constant between nodes v and u in the re-normalized adjacency matrix $\tilde{\mathbf{D}}^{\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{\frac{1}{2}}$; $\mathbf{W}^{(l),Q}$ is a quaternion weight matrix at the l-th layer; $\mathbf{h}_{\mathsf{v}}^{(0),Q}$ is the quaternion feature vector of node v; and g can be a nonlinear activation function such as ReLU and can be adopted to each quaternion element [25].

Correspondingly, we represent the quaternion vector $\mathbf{h}_{\mathsf{u}}^{(l),Q} \in \mathbb{H}^n$ and the quaternion weight matrix $\mathbf{W}^{(l),Q} \in \mathbb{H}^{m \times n}$ as:

$$\mathbf{h}_{\mathsf{u}}^{(l),Q} = \mathbf{h}_{\mathsf{u},r}^{(l)} + \mathbf{h}_{\mathsf{u},i}^{(l)}\mathbf{i} + \mathbf{h}_{\mathsf{u},j}^{(l)}\mathbf{j} + \mathbf{h}_{\mathsf{u},k}^{(l)}\mathbf{k} \tag{3}$$

$$\mathbf{W}^{(l),Q} = \mathbf{W}_{r}^{(l)} + \mathbf{W}_{i}^{(l)}\mathbf{i} + \mathbf{W}_{j}^{(l)}\mathbf{j} + \mathbf{W}_{k}^{(l)}\mathbf{k} \tag{4}$$

where $\mathbf{h}_{\mathsf{u},r}^{(l)}$, $\mathbf{h}_{\mathsf{u},i}^{(l)}$, $\mathbf{h}_{\mathsf{u},j}^{(l)}$, and $\mathbf{h}_{\mathsf{u},k}^{(l)} \in \mathbb{R}^n$; and $\mathbf{W}_{r}^{(l)}$, $\mathbf{W}_{i}^{(l)}$, $\mathbf{W}_{j}^{(l)}$, and $\mathbf{W}_{k}^{(l)} \in \mathbb{R}^{m \times n}$. We now express the Hamilton product $\otimes$ between $\mathbf{W}^{(l),Q}$ and $\mathbf{h}_{\mathsf{u}}^{(l),Q}$ derived from Equation 22 as:

$$\mathbf{W}^{(l),Q} \otimes \mathbf{h}_{\mathsf{u}}^{(l),Q} = \begin{bmatrix} 1 \\ \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \end{bmatrix}^{\top} \begin{bmatrix} \mathbf{W}_{r}^{(l)} & -\mathbf{W}_{i}^{(l)} & -\mathbf{W}_{j}^{(l)} & -\mathbf{W}_{k}^{(l)} \\ \mathbf{W}_{i}^{(l)} & \mathbf{W}_{r}^{(l)} & -\mathbf{W}_{k}^{(l)} & \mathbf{W}_{j}^{(l)} \\ \mathbf{W}_{j}^{(l)} & \mathbf{W}_{k}^{(l)} & \mathbf{W}_{r}^{(l)} & -\mathbf{W}_{i}^{(l)} \\ \mathbf{W}_{k}^{(l)} & -\mathbf{W}_{j}^{(l)} & \mathbf{W}_{i}^{(l)} & \mathbf{W}_{r}^{(l)} \end{bmatrix} \begin{bmatrix} \mathbf{h}_{\mathsf{u},r}^{(l)} \\ \mathbf{h}_{\mathsf{u},i}^{(l)} \\ \mathbf{h}_{\mathsf{u},j}^{(l)} \\ \mathbf{h}_{\mathsf{u},k}^{(l)} \end{bmatrix} \tag{5}$$

The four quaternion components $\mathbf{W}_{r}^{(l)}$, $\mathbf{W}_{i}^{(l)}$, $\mathbf{W}_{j}^{(l)}$, and $\mathbf{W}_{k}^{(l)}$ are shared when performing the Hamilton product; while in the Euclidean space, all the elements of the weight matrix are different parameter variables [34]. Thus, we can reduce the number of model parameters up to four times within the Quaternion space, similar to the parameter saving reported in [25, 34].

Note that the quaternion components of $\mathbf{W}^{(l),Q}$ are also shared across the four quaternion components $\mathbf{h}_{\mathsf{u},r}^{(l)}$, $\mathbf{h}_{\mathsf{u},i}^{(l)}$, $\mathbf{h}_{\mathsf{u},j}^{(l)}$, and $\mathbf{h}_{\mathsf{u},k}^{(l)}$. Therefore, if we use any slight change in the input $\mathbf{h}_{\mathsf{u}}^{(l),Q}$, we get an entirely different output [24], leading to a different performance. This phenomenon is one of the crucial reasons why the Quaternion space provides highly expressive computations through the Hamilton product compared to the Euclidean and complex vector spaces. The phenomenon enforces the model to learn the potential relations within each hidden layer and between the different hidden layers, hence increasing the graph representation quality.

QGNN for node classification. We consider $\mathbf{h}_{\mathsf{v}}^{(L),Q}$, which is the quaternion vector representation of node v at the *last* L-th QGNN layer. We vectorize $\mathbf{h}_{\mathsf{v}}^{(L),Q}$ to obtain the node representation $\mathbf{x}_{\mathsf{v}}^{(L)}$ as:

$$\mathbf{x}_{\mathsf{v}}^{(L)} = \textsc{Vec}\left(\mathbf{h}_{\mathsf{v}}^{(L),Q}\right) = \left[\mathbf{h}_{\mathsf{v},r}^{(L)}; \mathbf{h}_{\mathsf{v},i}^{(L)}; \mathbf{h}_{\mathsf{v},j}^{(L)}; \mathbf{h}_{\mathsf{v},k}^{(L)}\right] \tag{6}$$

where $\textsc{Vec}(.)$ denotes a concatenation of the four components of the quaternion vector. Note that *the learning process to predict the class labels is in the Euclidean space*. Therefore, to perform the semi-supervised node classification task, on top of the last L-th QGNN layer, we construct a GCN layer followed by a $\mathsf{softmax}$ activation function as follows:

$$\hat{\mathbf{y}}_{\mathsf{v}} = \mathsf{softmax}\left(\sum_{\mathsf{u} \in \mathcal{N}_{\mathsf{v}} \cup \{\mathsf{v}\}} a_{\mathsf{v},\mathsf{u}} \mathbf{W}_1 \mathbf{x}_{\mathsf{u}}^{(L)}\right), \forall \mathsf{v} \in \mathcal{V} \tag{7}$$

QGNN for graph classification. Following Equations 12 and 13, we employ a concatenation over the vector representations of node v at the different QGNN layers to construct the node embedding $\mathbf{e}_{\mathsf{v}}^{Q}$. And then we use the sum pooling to obtain the quaternion embedding $\mathbf{e}_{\mathcal{G}}^{Q}$ of the entire graph $\mathcal{G}$. To perform the graph classification task, we also use the $\textsc{Vec}(.)$ to vectorize $\mathbf{e}_{\mathcal{G}}^{Q}$ to obtain the final graph embedding $\mathbf{x}_{\mathcal{G}}$, which is fed to a single fully-connected layer followed by the $\mathsf{softmax}$ layer to predict the graph label. We then learn the model parameters for both the node and graph classification tasks by minimizing the cross-entropy loss function.

3 Experimental results

Table 1 presents the graph classification results of our QGNN and other up-to-date baselines.[1] In general, our QGNN produces state-of-the-art accuracies on most datasets; hence this demonstrates a notable impact of our model. In particular, QGNN outperforms all baseline models on IMDB-B,

[1] We present additional details in the appendix.

Table 1: Graph classification accuracies (%). The best scores are in **bold**.

Model	COLLAB	IMDB-B	IMDB-M	DD	PROTEINS	MUTAG	PTC
GK [33]	72.84 ± 0.28	65.87 ± 0.98	43.89 ± 0.38	78.45 ± 0.26	71.67 ± 0.55	81.58 ± 2.11	57.26 ± 1.41
WL [32]	79.02 ± 1.77	73.40 ± 4.63	49.33 ± 4.75	79.78 ± 0.36	74.68 ± 0.49	82.05 ± 0.36	57.97 ± 0.49
DGK [40]	73.09 ± 0.25	66.96 ± 0.56	44.55 ± 0.52	73.50 ± 1.01	75.68 ± 0.54	87.44 ± 2.72	60.08 ± 2.55
AWE [12]	73.93 ± 1.94	74.45 ± 5.83	51.54 ± 3.61	71.51 ± 4.02	–	87.87 ± 9.76	–
PSCN [23]	72.60 ± 2.15	71.00 ± 2.29	45.23 ± 2.84	77.12 ± 2.41	75.89 ± 2.76	**92.63 ± 4.21**	62.29 ± 5.68
GSAGE [9]	79.70 ± 1.70	72.40 ± 3.60	49.90 ± 5.00	65.80 ± 4.90	65.90 ± 2.70	79.80 ± 13.9	–
GAT [35]	75.80 ± 1.60	70.50 ± 2.30	47.80 ± 3.10	–	74.70 ± 2.20	89.40 ± 6.10	66.70 ± 5.10
GCAPS [36]	77.71 ± 2.51	71.69 ± 3.40	48.50 ± 4.10	77.62 ± 4.99	76.40 ± 4.17	–	66.01 ± 5.91
DGCNN [45]	73.76 ± 0.49	70.03 ± 0.86	47.83 ± 0.85	79.37 ± 0.94	75.54 ± 0.94	85.83 ± 1.66	58.59 ± 2.47
CapsGNN [38]	79.62 ± 0.91	73.10 ± 4.83	50.27 ± 2.65	75.38 ± 4.17	76.28 ± 3.63	86.67 ± 6.88	–
IEGN [18]	77.92 ± 1.70	71.27 ± 4.50	48.55 ± 3.90	–	75.19 ± 4.30	84.61 ± 10.0	59.47 ± 7.30
DSGC [31]	79.20 ± 1.60	73.20 ± 4.90	48.50 ± 4.80	77.40 ± 6.40	74.20 ± 3.80	86.70 ± 7.60	–
PPGN [17]	81.38 ± 1.42	73.00 ± 5.77	50.46 ± 3.59	–	77.20 ± 4.73	90.55 ± 8.70	66.17 ± 6.54
GIN-0 [39]	80.20 ± 1.90	75.10 ± 5.10	52.30 ± 2.80	–	76.20 ± 2.80	89.40 ± 5.60	64.60 ± 7.00
GFN [3]	81.50 ± 2.42	73.00 ± 4.35	51.80 ± 5.16	78.78 ± 3.49	76.46 ± 4.06	90.84 ± 7.22	–
GCN [14]	**81.72 ± 1.64**	73.30 ± 5.29	51.20 ± 5.13	79.12 ± 3.07	75.65 ± 3.24	87.20 ± 5.11	–
QGNN	81.36 ± 1.31	**77.56 ± 2.45**	**53.78 ± 3.83**	**79.92 ± 3.54**	**78.47 ± 3.30**	92.59 ± 3.59	**69.92 ± 2.59**

IMDB-M, DD, PROTEINS, and PTC. QGNN obtains competitive accuracies on COLLAB and MUTAG, but there are no significant differences between our QGNN and the best models on these two datasets. Furthermore, our QGNN achieves higher accuracies than GCN except for COLLAB, where we get similar results. Note that we found the best results of GCN in [3], where, after obtaining the graph embeddings, [3] used two fully-connected layers to predict the graph labels. This is different from other models such as GIN-0 and QGNN, where we constructed a single fully-connected layer.

Table 2: Node classification accuracies (%).

Dataset	GCN	QGNN
CORA	85.77	**87.48 ± 1.08**
CITESEER	73.68	**76.03 ± 1.89**
PUBMED	**88.13**	87.65 ± 0.47

Table 3: Text (node) classification accuracies (%).

Dataset	GCN	QGNN
20NG	86.34 ± 0.09	**86.40 ± 0.12**
R8	97.07 ± 0.10	**97.09 ± 0.15**
R52	93.56 ± 0.18	**93.97 ± 0.20**
OHSUMED	68.32 ± 0.56	**68.49 ± 0.42**
MR	**76.74 ± 0.20**	76.67 ± 0.21

Table 2 presents the node classification accuracies, where the results of GCN are also taken from [27]. As mentioned in the evaluation protocol, for a fair comparison, we use exactly the same experimental setting used in [27], where each dataset has 10 random data splits, wherein each data split has 60%, 20%, 20% numbers of nodes, equally distributed for each class, for training, validation and testing respectively. We achieve the accuracies of 87.48%, 76.03%, and 87.65% on CORA, CITESEER, and PUBMED respectively. In particular, our QGNN outperforms GCN on CORA and CITESEER, and produces competitive results on PUBMED.

Table 3 presents the text (node) classification accuracies for GCN and our QGNN, wherein the results of GCN are also taken from [42]. As mentioned in Section C.3, for a fair comparison, we also use exactly the same experimental setting used in [42]. In general, QGNN works better than GCN on four datasets, except MR.

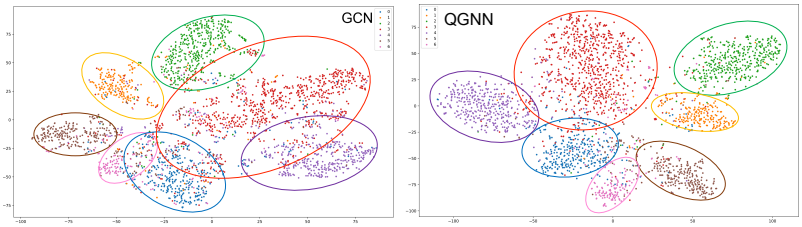

Figure 1: Visualization of node representations on CORA.

Qualitative result. To qualitatively demonstrate the effectiveness of our QGNN, we use t-SNE [16] to visualize the node representations learned at the first layer of GCN and QGNN on CORA in Figure 1, where colors denote the class labels. We concatenate the four components of the quaternion node embeddings to have the real-valued vector inputs for t-SNE. The figure shows that our QGNN has a better class separation, implying the higher quality of the learned node representations.

Acknowledgements

This research was partially supported by the ARC Discovery Projects DP150100031 and DP160103934.

References

[1] Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. Towards sparse hierarchical graph classifiers. *arXiv preprint arXiv:1811.01287*, 2018.

[2] Ines Chami, Zhitao Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 4869–4880, 2019.

[3] Ting Chen, Song Bian, and Yizhou Sun. Are powerful graph neural nets necessary? a dissection on graph classification. *arXiv:1905.04579*, 2019.

[4] Simon S Du, Kangcheng Hou, Russ R Salakhutdinov, Barnabas Poczos, Ruosong Wang, and Keyulu Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. In *Advances in Neural Information Processing Systems 32*, pages 5723–5733, 2019.

[5] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[6] Chase J Gaudet and Anthony S Maida. Deep quaternion networks. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2018.

[7] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural Message Passing for Quantum Chemistry. In *International Conference on Machine Learning*, pages 1263–1272, 2017.

[8] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[9] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[10] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv:1709.05584*, 2017.

[11] William Rowan Hamilton. Ii. on quaternions; or on a new system of imaginaries in algebra. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 25(163):10–13, 1844.

[12] Sergey Ivanov and Evgeny Burnaev. Anonymous walk embeddings. In *International Conference on Machine Learning*, pages 2191–2200, 2018.

[13] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015.

[14] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[15] Qi Liu, Maximilian Nickel, and Douwe Kiela. Hyperbolic graph neural networks. In *Advances in Neural Information Processing Systems*, pages 8230–8241, 2019.

[16] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9:2579–2605, 2008.

[17] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *Advances in Neural Information Processing Systems*, pages 2153–2164, 2019.

[18] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *International Conference on Learning Representations (ICLR)*, 2019.

[19] Galileo Mark Namata, Ben London, Lise Getoor, and Bert Huang. Query-driven active surveying for collective classification. In *Workshop on Mining and Learning with Graphs*, 2012.

[20] Dai Quoc Nguyen, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Phung. A capsule network-based model for learning node embeddings. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management*, pages 3313–3316, 2020.

[21] Dai Quoc Nguyen, Tu Dinh Nguyen, and Dinh Phung. Universal graph transformer self-attention networks. *arXiv preprint arXiv:1909.11855*, 2019.

[22] Dai Quoc Nguyen, Tu Dinh Nguyen, and Dinh Phung. A self-attention network based node embedding model. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, 2020.

[23] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning Convolutional Neural Networks for Graphs. In *International Conference on Machine Learning*, pages 2014–2023, 2016.

[24] Titouan Parcollet, Mohamed Morchid, and Georges Linarès. A survey of quaternion neural networks. *Artificial Intelligence Review*, pages 1–26, 2019.

[25] Titouan Parcollet, Mirco Ravanelli, Mohamed Morchid, Georges Linarès, Chiheb Trabelsi, Renato De Mori, and Yoshua Bengio. Quaternion recurrent neural networks. In *International Conference on Learning Representations (ICLR)*, 2019.

[26] Titouan Parcollet, Ying Zhang, Mohamed Morchid, Chiheb Trabelsi, Georges Linarès, Renato De Mori, and Yoshua Bengio. Quaternion convolutional neural networks for end-to-end automatic speech recognition. In *Interspeech*, pages 22–26, 2018.

[27] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-GCN: Geometric Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*, 2020.

[28] Erzsébet Ravasz and Albert-László Barabási. Hierarchical organization in complex networks. *Physical review E*, 67(2):026112, 2003.

[29] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[30] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93, 2008.

[31] Younjoo Seo, Andreas Loukas, and Nathanael Peraudin. Discriminative structural graph classification. *arXiv:1905.13422*, 2019.

[32] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

[33] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient Graphlet Kernels for Large Graph Comparison. In *AISTATS*, pages 488–495, 2009.

[34] Yi Tay, Aston Zhang, Anh Tuan Luu, Jinfeng Rao, Shuai Zhang, Shuohang Wang, Jie Fu, and Siu Cheung Hui. Lightweight and efficient neural natural language processing with quaternion networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1494–1503, 2019.

[35] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations (ICLR)*, 2018.

[36] Saurabh Verma and Zhi-Li Zhang. Graph capsule convolutional neural networks. *The Joint ICML and IJCAI Workshop on Computational Biology*, 2018.

[37] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv:1901.00596*, 2019.

[38] Zhang Xinyi and Lihui Chen. Capsule Graph Neural Network. *International Conference on Learning Representations (ICLR)*, 2019.

[39] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful Are Graph Neural Networks? *International Conference on Learning Representations (ICLR)*, 2019.

[40] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *SIGKDD*, pages 1365–1374, 2015.

[41] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, pages 40–48, 2016.

[42] Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *AAAI*, volume 33, pages 7370–7377, 2019.

[43] Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, pages 4805–4815, 2018.

[44] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Network representation learning: A survey. *IEEE Transactions on Big Data*, 6:3–28, 2020.

[45] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An End-to-End Deep Learning Architecture for Graph Classification. In *AAAI*, 2018.

[46] Shuai Zhang, Yi Tay, Lina Yao, and Qi Liu. Quaternion knowledge graph embeddings. In *Advances in Neural Information Processing Systems*, pages 2731–2741, 2019.

[47] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv:1812.08434*, 2018.

[48] Xuanyu Zhu, Yi Xu, Hongteng Xu, and Changjian Chen. Quaternion convolutional neural networks. In *ECCV*, pages 631–647, 2018.

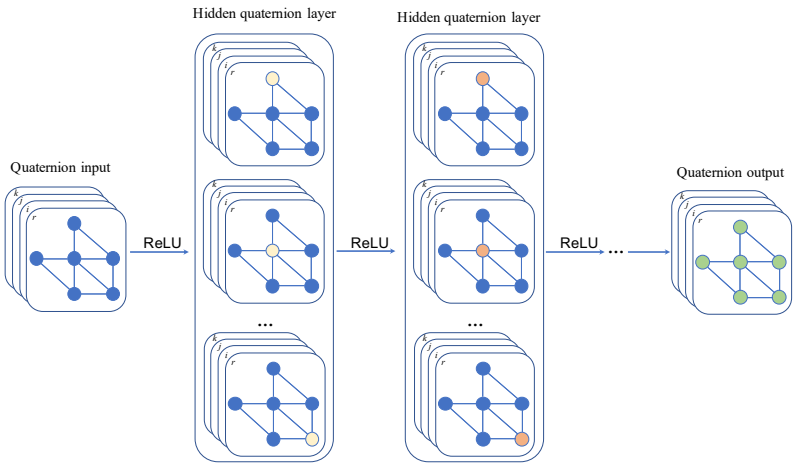

Figure 2: Illustration of our QGNN.

A Related work

We represent each graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \{\mathbf{h}_v\}_{v\in\mathcal{V}})$, where $\mathcal{V}$ is a set of nodes, $\mathcal{E}$ is a set of edges, and $\mathbf{h}_v$ (i.e., $\mathbf{h}_v^{(0)}$) represents the Euclidean feature vector of node $v \in \mathcal{V}$.

Node classification. We consider a graph $\mathcal{G}$, where each node belongs to one of the class labels. Given the labels of a subset of $\mathcal{V}$, the task is to predict the labels of remaining nodes.

Graph classification. Given a set of M disjoint graphs $\{\mathcal{G}_m\}_{m=1}^M$ and their corresponding class labels $\{y_m\}_{m=1}^M \subseteq \mathcal{Y}$, the task is to learn an embedding $\mathbf{e}_{\mathcal{G}_m}$ for each entire graph $\mathcal{G}_m$ to predict its label y_m.

Recent work on graph representation learning has focused on using graph neural networks (GNNs). In general, GNNs update the vector representation of each node by recursively aggregating and transforming the vector representations of its neighbors [14, 9, 35, 22, 20]. After that, GNNs use a READOUT pooling function to obtain the vector representation of the entire graph [7, 45, 43, 36, 39]. Mathematically, given a graph $\mathcal{G}$, we formulate GNNs as follows:

$$\mathbf{h}_v^{(l+1)} = \text{AGGREGATION}\left(\left\{\mathbf{h}_u^{(l)}\right\}_{u\in\mathcal{N}_v\cup\{v\}}\right) \tag{8}$$

$$\mathbf{e}_{\mathcal{G}} = \textsc{ReadOut}\left(\left\{\left\{\mathbf{h}_{\mathsf{v}}^{(l)}\right\}_{l=0}^{L}\right\}_{\mathsf{v}\in\mathcal{V}}\right) \tag{9}$$

where $\mathbf{h}_{\mathsf{v}}^{(l)}$ is the vector representation of node v at the l-th iteration/layer, $\mathcal{N}_{\mathsf{v}}$ is the set of neighbors of node v, and $\mathbf{h}_{\mathsf{v}}^{(0)} = \mathbf{h}_{\mathsf{v}}$.

There have been many designs for the AGGREGATION functions proposed in recent literature. The widely-used one is introduced in Graph Convolutional Network (GCN) [14] as:

$$\mathbf{h}_{\mathsf{v}}^{(l+1)} = \mathsf{g}\left(\sum_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}\cup\{\mathsf{v}\}} a_{\mathsf{v},\mathsf{u}}\mathbf{W}^{(l)}\mathbf{h}_{\mathsf{u}}^{(l)}\right), \forall \mathsf{v}\in\mathcal{V} \tag{10}$$

where $a_{\mathsf{v},\mathsf{u}}$ is an edge constant between nodes v and u in the re-normalized adjacency matrix $\tilde{\mathbf{D}}^{\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{\frac{1}{2}}$; $\mathbf{W}^{(l)}$ is a weight matrix; and g is a nonlinear activation function. Besides, a more powerful aggregation function based on multi-layer perceptrons (MLPs) (e.g., two fully-connected layers) is used in Graph Isomorphism Network (GIN-0) [39]:

$$\mathbf{h}_{\mathsf{v}}^{(l+1)} = \mathsf{MLP}^{(l)}\left(\sum_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}\cup\{\mathsf{v}\}} \mathbf{h}_{\mathsf{u}}^{(l)}\right), \forall \mathsf{v}\in\mathcal{V} \tag{11}$$

Following the GIN-0 architecture, we employ a concatenation over the vector representations of node v at the different layers to construct the node embedding $\mathbf{e}_{\mathsf{v}}$ as:

$$\mathbf{e}_{\mathsf{v}} = \left[\mathbf{h}_{\mathsf{v}}^{(1)}; \mathbf{h}_{\mathsf{v}}^{(2)}; ...; \mathbf{h}_{\mathsf{v}}^{(L)}\right], \forall \mathsf{v}\in\mathcal{V} \tag{12}$$

where L is the index of the last layer. The graph-level READOUT function can be a simple sum pooling or a complex pooling such as sort pooling [45], hierarchical pooling [1], and differentiable pooling [43]. As the sum pooling often produces competitive performances [39], we utilize the sum pooling to obtain the embedding $\mathbf{e}_{\mathcal{G}}$ of the entire graph $\mathcal{G}$ as:

$$\mathbf{e}_{\mathcal{G}} = \sum_{\mathsf{v}\in\mathcal{V}} \mathbf{e}_{\mathsf{v}} = \sum_{\mathsf{v}\in\mathcal{V}} \left[\mathbf{h}_{\mathsf{v}}^{(1)}; \mathbf{h}_{\mathsf{v}}^{(2)}; ...; \mathbf{h}_{\mathsf{v}}^{(L)}\right] \tag{13}$$

B Quaternion background

The hyper-complex vector space has recently been considered on the Quaternion space [11] consisting of one real and three separate imaginary axes. It provides highly expressive computations through the Hamilton product compared to the Euclidean and complex vector spaces. The Quaternion space has been applied to image classification [48, 6], speech recognition [26, 25], knowledge graph [46], and natural language processing [34].

We provide key notations and operations related to the Quaternion space required for our later development. Additional details can further be found in [24].

A quaternion $q \in \mathbb{H}$ is a hyper-complex number consisting of one real and three separate imaginary components [11] defined as:

$$q = q_r + q_i\mathbf{i} + q_j\mathbf{j} + q_k\mathbf{k} \tag{14}$$

where $q_r, q_i, q_j, q_k \in \mathbb{R}$, and $\mathbf{i}, \mathbf{j}, \mathbf{k}$ are imaginary units that $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$. Correspondingly, a n-dimensional quaternion vector $\boldsymbol{q} \in \mathbb{H}^n$ is defined as:

$$\boldsymbol{q} = \boldsymbol{q}_r + \boldsymbol{q}_i\mathbf{i} + \boldsymbol{q}_j\mathbf{j} + \boldsymbol{q}_k\mathbf{k} \tag{15}$$

where $\boldsymbol{q}_r, \boldsymbol{q}_i, \boldsymbol{q}_j, \boldsymbol{q}_k \in \mathbb{R}^n$. The operations for the Quaternion algebra are defined as follows:

Addition. The addition of two quaternions q and p is defined as:

$$q + p = (q_r + p_r) + (q_i + p_i)\mathbf{i} + (q_j + p_j)\mathbf{j} + (q_k + p_k)\mathbf{k} \tag{16}$$

Norm. The norm $\|q\|$ of a quaternion q is defined as:

$$\|q\| = \sqrt{q_r^2 + q_i^2 + q_j^2 + q_k^2} \tag{17}$$

And the normalized or unit quaternion $q^\triangleleft$ is defined as:

$$q^\triangleleft = \frac{q}{\|q\|} \tag{18}$$

Scalar multiplication. The multiplication of a scalar λ and a quaternion q is defined as:

$$\lambda q = \lambda q_r + \lambda q_i \mathbf{i} + \lambda q_j \mathbf{j} + \lambda q_k \mathbf{k} \tag{19}$$

Conjugate. The conjugate q^* of a quaternion q is defined as:

$$q^* = q_r - q_i \mathbf{i} - q_j \mathbf{j} - q_k \mathbf{k} \tag{20}$$

Hamilton product. The Hamilton product $\otimes$ (i.e., the quaternion multiplication) of two quaternions q and p is defined as:

$$\begin{aligned} q \otimes p \quad = \quad & (q_r p_r - q_i p_i - q_j p_j - q_k p_k) \\ + \quad & (q_i p_r + q_r p_i - q_k p_j + q_j p_k)\mathbf{i} \\ + \quad & (q_j p_r + q_k p_i + q_r p_j - q_i p_k)\mathbf{j} \\ + \quad & (q_k p_r - q_j p_i + q_i p_j + q_r p_k)\mathbf{k} \end{aligned} \tag{21}$$

We can express the Hamilton product of q and p in the following form:

$$q \otimes p = \begin{bmatrix} 1 \\ \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \end{bmatrix}^\top \begin{bmatrix} q_r & -q_i & -q_j & -q_k \\ q_i & q_r & -q_k & q_j \\ q_j & q_k & q_r & -q_i \\ q_k & -q_j & q_i & q_r \end{bmatrix} \begin{bmatrix} p_r \\ p_i \\ p_j \\ p_k \end{bmatrix} \tag{22}$$

We note that the Hamilton product is not commutative, i.e., $q \otimes p \neq p \otimes q$.

Concatenation. In our approach, we further define a concatenation of two quaternion vectors $\boldsymbol{q}$ and $\boldsymbol{p}$ as:

$$[\boldsymbol{q}; \boldsymbol{p}] = [\boldsymbol{q}_r; \boldsymbol{p}_r] + [\boldsymbol{q}_i; \boldsymbol{p}_i]\,\mathbf{i} + \left[\boldsymbol{q}_j; \boldsymbol{p}_j\right]\mathbf{j} + [\boldsymbol{q}_k; \boldsymbol{p}_k]\,\mathbf{k} \tag{23}$$

C Experimental setup

We conduct experiments to evaluate our QGNN on the tasks of graph classification, semi-supervised node classification, and text (node) classification. Table 4 reports the statistics of benchmark datasets used for the three tasks.

The quaternion feature vectors $\mathbf{h}_\mathsf{v}^{(0),Q}$. As shown in Equation 2, we consider how to initialize the quaternion feature vectors $\mathbf{h}_\mathsf{v}^{(0),Q}$ of nodes v. Note that we evaluate our QGNN on benchmark datasets where the Euclidean feature vectors $\mathbf{h}_\mathsf{v}$ are typically given and pre-fixed (as mentioned in Section A). We see that each vector element within the feature vector $\mathbf{h}_\mathsf{v}$ specifies an individual node attribute that lives in an independent space. Therefore, we also aim to learn latent relations among these node attributes to strengthen the graph representations. To this end, we set the same $\mathbf{h}_\mathsf{v}$ to the four components of $\mathbf{h}_\mathsf{v}^{(0),Q}$ as:

$$\mathbf{h}_{\mathsf{v},r}^{(0)} = \mathbf{h}_{\mathsf{v},i}^{(0)} = \mathbf{h}_{\mathsf{v},j}^{(0)} = \mathbf{h}_{\mathsf{v},k}^{(0)} = \mathbf{h}_\mathsf{v} \tag{24}$$

The four components of $\mathbf{h}_\mathsf{v}^{(0),Q}$ are shared when performing the Hamilton product; hence we could achieve our aim in higher layers. In our pilot experiments, we find that using this simple mapping scheme produces competitive accuracies.

Parameter initialization. [25] and [46] used a specialized scheme to initialize the parameters in the quaternion weight matrices, while [48] and [34] applied the Glorot initialization [8] that is also used in previous GNN works such as GCN [14]; hence we use the Glorot initialization for a fair comparison with the previous works.

Discussion. We refrained from constructing a complex architecture within the Quaternion space using the complicated AGGREGATION and READOUT functions, as our main goal is to introduce a simple and effective framework that works well and produces competitive performances on the benchmark datasets. Thus, it is reasonable to propose our QGNN as a generalized variant of GCNs within the Quaternion space.

To demonstrate the effectiveness of our QGNN, we use the similar architectures and the same optimal hyper-parameters taken from the original papers that we follow for the tasks of node classification and text (node) classification (i.e., we do not tune our QGNN's hyper-parameters for these two tasks), as presented in Sections C.2 and C.3.

Table 4: Statistics of the benchmark datasets. Avg#N denotes the average number of nodes per graph. #d denotes the dimension of feature vectors. #Cls denotes the number of class labels.

GRAPH	**#Graphs**	**Avg#N**	**#d**	**#Cls**
COLLAB	5,000	74.5	–	3
IMDB-B	1,000	19.8	–	2
IMDB-M	1,500	13.0	–	3
DD	1,178	284.3	82	2
PROTEINS	1,113	39.1	3	2
PTC	344	25.6	19	2
MUTAG	188	17.9	7	2
NODE	**#Nodes**	**#Edges**	**#d**	**#Cls**
CORA	2,708	5,429	1,433	7
CITESEER	3,327	4,732	3,703	6
PUBMED	19,717	44,338	500	3
TEXT (node)	**#Docs**	**#Words**	**#Nodes**	**#Cls**
20NG	18,846	42,757	61,603	20
R8	7,674	7,688	15,362	8
R52	9,100	8,892	17,992	52
OHSUMED	7,400	14,157	21,557	23
MR	10,662	18,764	29,426	2

C.1 Graph classification

Datasets. We use well-known datasets that are three social network datasets (consisting of COLLAB, IMDB-B and IMDB-M) [40] and four bioinformatics datasets (consisting of DD, MUTAG, PROTEINS, and PTC). The social network datasets do not have available node features; thus, we follow [23, 45] to use node degrees as features on these datasets.

Evaluation protocol. We follow [39, 38, 17, 31, 3] to use the same data splits and the same 10-fold cross-validation scheme to calculate the classification performance for a fair comparison. We compare our QGNN with up-to-date strong baselines and report the baseline results published in the original papers or reported in [12, 36, 38, 3, 31].

Training protocol. We vary the number of hidden layers in {1, 2, 3, 4, 5}, and the hidden size (i.e., the number of quaternions in the hidden layers) in {8, 16, 32, 64}. We set the batch size to 4 and use the Adam optimizer [13] with the initial learning rate $\in \left\{5e^{-5}, 1e^{-4}, 5e^{-4}, 1e^{-3}\right\}$. We run up to 100 epochs to evaluate our trained model.

C.2 Node classification

Datasets. We use three benchmark datasets consisting of CORA, CITESEER [30] and PUBMED [19] that are citation networks. Each node represents a document in each dataset, and each edge represents a citation link between two documents. Each node is also associated with a feature vector of a bag-of-words. Each node is assigned a class label representing the main topic of the document.

Evaluation protocol. As mentioned in [5, 27], the experimental setup used in [14, 35] is not fair to show the effectiveness of existing GNN models when only using one fixed data split of training, validation, and test sets from [41]. Therefore, for a fair comparison, we use the same 10 random data splits used in [27], where each data split consists of 60%, 20%, 20% numbers of nodes, equally distributed for each node class, for training, validation, and testing, respectively. We also follow [27] to report the average accuracy on the test sets across the 10 data splits.

Training protocol. The architecture used by [27] is a 2-layer GCN, wherein the hidden sizes are 16 for CORA and CITESEER, and 64 for PUBMED. Hence, for a fair comparison, here, we construct 1-layer QGNN followed by 1-layer GCN (referring to our QGNN for node classification described in Section 2). We use the corresponding hidden sizes of 4 for CORA and CITESEER, and 16 on PUBMED. We also set the same Adam initial learning rate to 0.05, and the same number of epochs to 100 for both CORA and CITESEER; while they are 0.1 and 200 respectively for PUBMED.

C.3 Text (node) classification

Datasets. [42] proposed to transform a collection of text documents into a graph that considers words and documents as nodes with one-hot feature vectors. Each edge between two word nodes is weighted using point-wise mutual information. Each edge between a document node and a word node is weighted using the term frequency-inverse document frequency, to construct the adjacency matrix. Hence we could use GNNs to predict the class labels of the document nodes (i.e., becoming the node classification task). We also follow [42] to use the same data splits and the evaluation scheme for five benchmarks – 20NG, R8, R52, OHSUMED, and MR – which are medical abstract, web mining, and social network datasets.

Evaluation protocol. We follow [42] to report the mean and standard deviation over 10 runs.

Training protocol. [42] also used 2-layer GCN to perform the text (node) classification with the hidden size of 200. Therefore, we use 1-layer QGNN followed by 1-layer GCN with the quaternion hidden size of 50. We also use the same hyper-parameter settings; those are 0.02 Adam learning rate and 200 training epochs.

D Model size

Table 5: Number of learnable parameters on IMDB-B with using the hidden size of 256 in GCN, and corresponding to the hidden size of 64 in our QGNN.

Model	GCN	QGNN
1-layer	17,152	17,152
2-layer	83,200	34,048
3-layer	149,248	50,944
4-layer	215,296	67,840
5-layer	281,344	84,736

We report the total number of learnable parameters for the GCN and QGNN using the similar architectures on IMDB-B in Table 5. The GCN's hidden size is 256; accordingly, the QGNN's hidden size is 64. As mentioned in Equation 24, we set the feature vectors $\mathbf{h}_{\mathsf{v}}$ to the four components of the quaternion feature vectors $\mathbf{h}_{\mathsf{v}}^{(0),Q}$ of nodes v, hence the 1-layer QGNN and the 1-layer GCN have the same number of parameters. For deeper architectures, we see a remarkable difference between the model sizes of GCN and QGNN, e.g., the 5-layer GCN size is approximately four times larger than the 5-layer QGNN size. For this reason, our proposal for employing the Quaternion space helps to reduce the model parameters significantly.

E Conclusion

In this paper, we aim to increase the graph representation quality and reduce the number of model parameters up to four times. Therefore, we propose to learn node and graph embeddings within

the Quaternion space. We study this strategy by introducing our quaternion graph neural networks (QGNN) to generalize GCNs within the Quaternion space. The experimental results demonstrate that our QGNN obtains state-of-the-art accuracies on a range of well-known benchmark datasets for the three tasks of graph classification, semi-supervised node classification, and text (node) classification.