Quaternion Graph Neural Networks

Dai Quoc Nguyen DAI.NGUYEN@MONASH.EDU
Department of Data Science and AI, Monash University, Australia

Tu Dinh Nguyen V.TUND21@VINAI.IO
VinAI Research, Vietnam

Dinh Phung DINH.PHUNG@MONASH.EDU
Department of Data Science and AI, Monash University, Australia

Editors: Vineeth N Balasubramanian and Ivor Tsang

Abstract

Recently, graph neural networks (GNNs) have become an important and active research direction in deep learning. It is worth noting that most of the existing GNN-based methods learn graph representations within the Euclidean vector space. Beyond the Euclidean space, learning representation and embeddings in hyper-complex space have also shown to be a promising and effective approach. To this end, we propose Quaternion Graph Neural Networks (QGNN) to learn graph representations within the Quaternion space. As demonstrated, the Quaternion space, a hyper-complex vector space, provides highly meaningful computations and analogical calculus through Hamilton product compared to the Euclidean and complex vector spaces. Our QGNN obtains state-of-the-art results on a range of benchmark datasets for graph classification and node classification. Besides, regarding knowledge graphs, our QGNN-based embedding model achieves state-of-the-art results on three new and challenging benchmark datasets for knowledge graph completion. Our code is available at: https://github.com/daiquocnguyen/QGNN.

1. Introduction

Graph representation learning has recently emerged as a new promising learning paradigm for graph-structured data (Hamilton et al., 2017b; Wu et al., 2019b), where the goal is to learn a parametric mapping function that embeds the nodes, the subgraphs, or the entire graph into low-dimensional continuous vector spaces; hence the learned vectors can be useful for downstream tasks. Recently, graph neural networks (GNNs) become an essential strand to learn graph representations (Zhang et al., 2020a; Nguyen, 2021). In general, GNNs utilize an aggregation function to update the vector representation of each node by aggregating those of its neighbors (Kipf and Welling, 2017; Veličković et al., 2018; Nguyen et al., 2019b). GNNs also use a graph-level readout function such as a simple sum pooling to obtain graph embeddings. As a result, GNNs achieve state-of-the-art performances for downstream tasks such as node classification and graph classification (Hamilton et al., 2017a; Xu et al., 2019).

It is noted that most of the existing GNNs learn graph representations within the Euclidean space. Recently, learning representation and embeddings in hyper-complex space have shown to be a promising and effective approach to move beyond the Euclidean space. For example, some Quaternion space-based methods have been applied in image classifi-

cation (Gaudet and Maida, 2018; Zhu et al., 2018), speech recognition (Parcollet et al., 2018, 2019b), knowledge graph (Zhang et al., 2019; Nguyen et al., 2020), and machine translation (Tay et al., 2019). As the Quaternion space allows highly expressive computations through Hamilton product compared to the Euclidean and complex vector spaces (Parcollet et al., 2019a), we introduce Quaternion Graph Neural Networks (QGNN) to learn graph representations within the Quaternion space. Our proposed model achieves effective performances through extensive experimental evaluation, benchmarking with current state-of-the-art methods, and post-analysis to demonstrate the merits of the proposed methodology, on a wide range of benchmark datasets for the tasks of node classification, graph classification, and knowledge graph completion. In summary, our contributions can be highlighted as follows:

• We propose Quaternion Graph Neural Networks (QGNN) to learn quaternion embeddings for graph-structured data. To the best of our knowledge, our work is the first to investigate quaternion embeddings for general graphs with diverse and different structures.

• QGNN produces state-of-the-art accuracies on citation network datasets for node classification as well as social network and bioinformatics datasets for graph classification.

• Regarding knowledge graphs, we present a new, simple yet effective method that utilizes vanilla GNNs directly on a single undirected graph of entities constructed from a given knowledge graph, followed by a score function to compute the triple score. Our QGNN-based knowledge graph embedding model, called SimQGNN, outperforms the existing models and obtains state-of-the-art results on three new and challenging benchmark datasets CoDEx-S, CoDEx-M, and CoDEx-L (Safavi and Koutra, 2020) for knowledge graph completion.[1]

2. Related work

There have been many designs for the aggregation functions proposed in recent literature. The widely-used one is introduced in Graph Convolutional Networks (GCNs) (Kipf and Welling, 2017). Furthermore, Simple Graph Convolution (Wu et al., 2019a) is a simplified variant of GCN without using the non-linear activation function g. Graph Isomorphism Network (Xu et al., 2019) is proposed to leverage an aggregation function based on a multi-layer perceptron (MLP) network of two fully-connected layers. Besides, Graph Attention Network (Veličković et al., 2018) extends GCN to compute edge weights following the standard attention technique (Bahdanau et al., 2015) and then employs the multi-head attention technique (Vaswani et al., 2017) to further stabilise the learning process, except the final (prediction) layer where GAT uses averaging. UGformer (Nguyen et al., 2019b) adapts the transformer self-attention network (Vaswani et al., 2017; Dehghani et al., 2019) to construct an advanced aggregation function to learn node and graph representations. Hyperbolic Graph Convolutional Neural Networks (HGCN) (Chami et al., 2019) leverages the expressiveness of GCNs and hyperbolic geometry to map Euclidean input features to embeddings in hyperbolic spaces to learn node representations. Moreover, Gated GNNs (Li et al., 2016) adopts GRUs (Cho et al., 2014), unrolls the recurrence for a fixed number of timesteps, and removes the need to constrain parameters to ensure convergence.

Following (Xu et al., 2018, 2019), we also employ a concatenation over the vector representations of node v at the different layers to construct a final vector representation $\mathbf{e}_{\mathsf{v}}$

1. We propose Gated Quaternion Graph Neural Networks (GQGNN) in Appendix A.

for each node $\mathsf{v} \in \mathcal{G}$. The graph-level readout function can be a simple sum pooling or a complex pooling such as sort pooling (Zhang et al., 2018), hierarchical pooling (Cangea et al., 2018), and differentiable pooling (Ying et al., 2018). As the sum pooling produces competitive results (Xu et al., 2019), we use the simple sum pooling to obtain the embedding $\mathbf{e}_{\mathcal{G}}$ of the entire graph $\mathcal{G}$.

3. Quaternion background

A quaternion $q \in \mathbb{H}$ is a hyper-complex number consisting of one real and three separate imaginary components (Hamilton, 1844) 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 $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$. The operations for the Quaternion algebra are defined as follows:

Addition. The addition of two quaternions q and p is defined as: $q + p = (q_r + p_r) + (q_i + p_i)\mathbf{i} + (q_j + p_j)\mathbf{j} + (q_k + p_k)\mathbf{k}$

Norm. The norm $\|q\|$ of a quaternion q is computed as: $\|q\| = \sqrt{q_r^2 + q_i^2 + q_j^2 + q_k^2}$. And the normalized or unit quaternion $q^{\triangleleft}$ is defined as: $q^{\triangleleft} = \frac{q}{\|q\|}$

Scalar multiplication. The multiplication of a scalar λ and q is computed as: $\lambda q = \lambda q_r + \lambda q_i\mathbf{i} + \lambda q_j\mathbf{j} + \lambda q_k\mathbf{k}$

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

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

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

$$q \otimes p = \begin{bmatrix} 1 \\ \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \end{bmatrix}^{\top} \begin{bmatrix} q_r & -q_i & -q_j & -q_k \\ q_i & q_r & -q_k & q_j \\ q_j & q_k & q_r & -q_i \\ q_k & -q_j & q_i & q_r \end{bmatrix} \begin{bmatrix} p_r \\ p_i \\ p_j \\ p_k \end{bmatrix} \tag{2}$$

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

Quaternion-inner product. The quaternion-inner product $\bullet$ of two quaternion vectors $\boldsymbol{q}$ and $\boldsymbol{p} \in \mathbb{H}^n$ returns a scalar as:

$$\boldsymbol{q} \bullet \boldsymbol{p} = \boldsymbol{q}_r^{\top}\boldsymbol{p}_r + \boldsymbol{q}_i^{\top}\boldsymbol{p}_i + \boldsymbol{q}_j^{\top}\boldsymbol{p}_j + \boldsymbol{q}_k^{\top}\boldsymbol{p}_k \tag{3}$$

Concatenation. We define a concatenation of two quaternion vectors $\boldsymbol{q}$ and $\boldsymbol{p}$ as:

$$[\boldsymbol{q} \parallel \boldsymbol{p}] = [\boldsymbol{q}_r \parallel \boldsymbol{p}_r] + [\boldsymbol{q}_i \parallel \boldsymbol{p}_i]\,\mathbf{i} + [\boldsymbol{q}_j \parallel \boldsymbol{p}_j]\,\mathbf{j} + [\boldsymbol{q}_k \parallel \boldsymbol{p}_k]\,\mathbf{k} \tag{4}$$

4. Graph representation learning within the Quaternion space

Most of the existing GNNs focus on learning graph representations within the Euclidean space. We, however, note that learning representation and embeddings in hyper-complex space have recently shown to be a promising and effective approach to move beyond the Euclidean space (Parcollet et al., 2019a). This motivates us to move beyond the Euclidean space and consider the Quaternion space – a hyper-complex vector space – which provides highly expressive computations through the Hamilton product.

4.1. Quaternion Graph Neural Networks (QGNN)

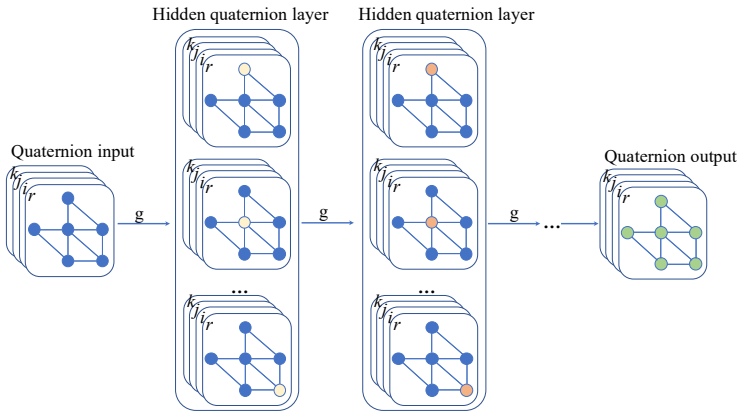

Figure 1: Illustration of our QGNN.

We propose Quaternion Graph Neural Networks (QGNN) to learn quaternion embeddings for graph-structured data. Our QGNN can be seen as a generalization of GCN within the Quaternion space. We represent each graph $\mathcal{G} = \left(\mathcal{V}, E, \{\mathbf{h}_{\mathsf{v}}^{(0),Q}\}_{\mathsf{v}\in\mathcal{V}}\right)$, where $\mathcal{V}$ is a set of nodes, E is a set of edges. In particular, the aggregation function in QGNN is defined as:[2]

$$\mathbf{h}_{\mathsf{v}}^{(l+1),Q} = \mathsf{g}\left(\sum_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}\cup\{\mathsf{v}\}} a_{\mathsf{v},\mathsf{u}}\mathbf{W}^{(l),Q} \otimes \mathbf{h}_{\mathsf{u}}^{(l),Q}\right), \forall \mathsf{v} \in \mathcal{V} \tag{5}$$

where $\mathcal{N}_{\mathsf{v}}$ is the set of neighbors of node v; $a_{\mathsf{v},\mathsf{u}}$ is an edge constant between nodes v and u in the re-normalized adjacency matrix $\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}$, wherein $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ where $\mathbf{A}$ is the adjacency matrix, $\mathbf{I}$ is the identity matrix, and $\tilde{\mathbf{D}}$ is the diagonal node degree matrix of $\tilde{\mathbf{A}}$. We use the superscript ^{Q} to denote the Quaternion space; $\mathbf{W}^{(l),Q}$ is a quaternion weight matrix; $\mathbf{h}_{\mathsf{v}}^{(0),Q}$ is the quaternion feature vector of node v; and g is a nonlinear activation function (such as ReLU or tanh) and can be adopted to each quaternion element (Parcollet et al., 2019b) as:

$$\mathsf{g}(q) = \mathsf{g}(q_r) + \mathsf{g}(q_i)\mathbf{i} + \mathsf{g}(q_j)\mathbf{j} + \mathsf{g}(q_k)\mathbf{k} \tag{6}$$

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:

$$\mathbf{h}_{\mathsf{u}}^{(l),Q} = \mathbf{h}_{\mathsf{u},r}^{(l)} + \mathbf{h}_{\mathsf{u},i}^{(l)}\mathbf{i} + \mathbf{h}_{\mathsf{u},j}^{(l)}\mathbf{j} + \mathbf{h}_{\mathsf{u},k}^{(l)}\mathbf{k} \tag{7}$$

$$\mathbf{W}^{(l),Q} = \mathbf{W}_r^{(l)} + \mathbf{W}_i^{(l)}\mathbf{i} + \mathbf{W}_j^{(l)}\mathbf{j} + \mathbf{W}_k^{(l)}\mathbf{k} \tag{8}$$

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 2 as:

2. In practice, we also implement QGNN efficiently using sparse matrix multiplications.

$$\mathbf{W}^{(l),Q} \otimes \mathbf{h}_{\mathsf{u}}^{(l),Q} = \begin{bmatrix} 1 \\ \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \end{bmatrix}^{\top} \begin{bmatrix} \mathbf{W}_r^{(l)} & -\mathbf{W}_i^{(l)} & -\mathbf{W}_j^{(l)} & -\mathbf{W}_k^{(l)} \\ \mathbf{W}_i^{(l)} & \mathbf{W}_r^{(l)} & -\mathbf{W}_k^{(l)} & \mathbf{W}_j^{(l)} \\ \mathbf{W}_j^{(l)} & \mathbf{W}_k^{(l)} & \mathbf{W}_r^{(l)} & -\mathbf{W}_i^{(l)} \\ \mathbf{W}_k^{(l)} & -\mathbf{W}_j^{(l)} & \mathbf{W}_i^{(l)} & \mathbf{W}_r^{(l)} \end{bmatrix} \begin{bmatrix} \mathbf{h}_{\mathsf{u},r}^{(l)} \\ \mathbf{h}_{\mathsf{u},i}^{(l)} \\ \mathbf{h}_{\mathsf{u},j}^{(l)} \\ \mathbf{h}_{\mathsf{u},k}^{(l)} \end{bmatrix} \tag{9}$$

Note that the quaternion components of $\mathbf{W}^{(l),Q}$ are 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, 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 (Parcollet et al., 2019a). The phenomenon enforces the model to learn the potential relations within each hidden layer and between the different hidden layers, hence increasing the representation quality. Furthermore, 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 (Tay et al., 2019). Thus, we can keep the same complexity and reduce the number of model parameters up to four times within the Quaternion space, similar to the parameter saving reported in (Parcollet et al., 2019b; Tay et al., 2019).

4.2. QGNN for node classification

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

We consider $\mathbf{h}_{\mathsf{v}}^{(L),Q}$, which is the quaternion vector representation of node v at the last L-th layer. To predict the label of node v, we simply feed $\mathbf{h}_{\mathsf{v}}^{(L),Q}$ to a prediction layer followed by a $\mathsf{softmax}$ layer as follows:

$$\hat{\mathbf{y}}_{\mathsf{v}} = \mathsf{softmax}\left(\sum_{\mathsf{u} \in \mathcal{N}_{\mathsf{v}} \cup \{\mathsf{v}\}} a_{\mathsf{v},\mathsf{u}} \mathbf{W}_1 \textsc{Vec}\left(\mathbf{h}_{\mathsf{v}}^{(L),Q} \right) \right), \forall \mathsf{v} \in \mathcal{V} \tag{10}$$

where $\textsc{Vec}(.)$ denotes a concatenation of the four components of the quaternion vector. For example,

$$\textsc{Vec}\left(\mathbf{h}_{\mathsf{v}}^{(L),Q} \right) = \left[\mathbf{h}_{\mathsf{v},r}^{(L)} \parallel \mathbf{h}_{\mathsf{v},i}^{(L)} \parallel \mathbf{h}_{\mathsf{v},j}^{(L)} \parallel \mathbf{h}_{\mathsf{v},k}^{(L)} \right] \tag{11}$$

We then learn the model parameters for the classification task by minimizing the cross-entropy loss function.

4.3. QGNN for graph classification

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

Following (Xu et al., 2019), we obtain the quaternion embedding $\mathbf{e}_{\mathcal{G}}^{Q}$ of the entire graph $\mathcal{G}$ as:

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

To perform the task, we feed $\mathbf{e}_{\mathcal{G}}^{Q}$ to a single fully-connected layer followed by a $\mathsf{softmax}$ layer as:

$$\hat{\mathbf{y}}_{\mathcal{G}} = \mathsf{softmax}\left(\mathbf{W}_2 \textsc{Vec}\left(\mathbf{e}_{\mathcal{G}}^{Q} \right) + \mathbf{b} \right) \tag{13}$$

where $\textsc{Vec}\left(\mathbf{e}_{\mathcal{G}}^{Q} \right) = [\mathbf{e}_{\mathcal{G},r} \parallel \mathbf{e}_{\mathcal{G},i} \parallel \mathbf{e}_{\mathcal{G},j} \parallel \mathbf{e}_{\mathcal{G},k}]$. We then also learn the model parameters by minimizing the cross-entropy loss function.

4.4. QGNN for knowledge graph completion (SimQGNN)

Knowledge graphs (KGs) can be viewed as directed multi-relational graphs to represent directional relationships between entities in the form of triples *(head, relation, tail)* denoted as *(h, r, t)*. However, large KGs are still incomplete, i.e., missing a lot of valid triples (West et al., 2014). To tackle this issue, research efforts have been made to predict whether a triple not in a knowledge graph is likely to be valid or not, which then helps to improve the graph completeness. More specifically, many KG embedding models have been proposed to learn entity and relation embeddings and return a score for each triple *(h, r, t)*, such that valid triples have higher scores than invalid ones (Bordes et al., 2011, 2013; Socher et al., 2013). For example, the score of the valid triple (Melbourne, city_Of, Australia) is higher than the score of the invalid one (Melbourne, city_Of, Germany).

It is worth mentioning that several KG embedding approaches have been proposed to adapt GNNs for knowledge graph completion, e.g., R-GCN (Schlichtkrull et al., 2018) and CompGCN (Vashishth et al., 2020). In general, these GNN-based models adopt an encoder-decoder architecture, wherein the encoder module aims to capture the relation-specific directions among entities, and then the decoder module adopts a score function to return the triple scores. Several examples of score functions are used in TransE (Bordes et al., 2013), ComplEx (Trouillon et al., 2016), ConvE (Dettmers et al., 2018), and ConvKB (Nguyen et al., 2019a).

We note that the encoder modules used in the existing GNN-based KG embedding models focus on modifying the vanilla GNNs. For example, R-GCN modifies GCNs to associate entities with relation-specific transformation matrices, and CompGCN customizes GCNs to combine composition operations between entities and relations with relation-type specific weight matrices. However, making these "more complicated" encoder modules is not completely effective at improving the performance of the existing GNN-based models. This is probably because the directions between entities in the knowledge graph can be well-encoded by an effective score function. In particular, the GNN-based models are outperformed by other conventional KG embedding models such as TuckER (Balažević et al., 2019) on benchmark datasets such as FB15k-237 (Toutanova and Chen, 2015) for knowledge graph completion (Bordes et al., 2013).

To this end, we propose a new method of exploring the vanilla GNNs (such as GCN or our QGNN) directly on a single *undirected* graph $\mathcal{G}$ of entities built from a given knowledge graph G, as illustrated in Figure 2, so that we construct a simple yet effective adaptive

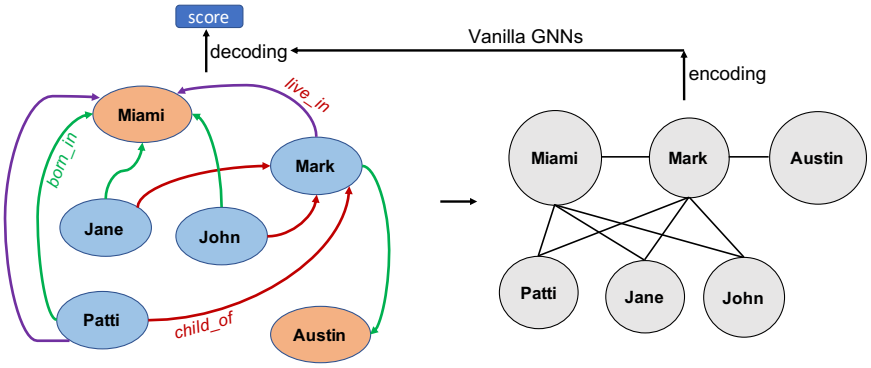

Figure 2: Illustration of adopting the vanilla GNNs on a single undirected graph $\mathcal{G}$ of entities built from a given knowledge graph G.

encoder module. As a result, we can easily adopt our QGNN on $\mathcal{G}$ as the encoder module as follows:

$$\mathbf{h}_{\mathsf{e}}^{(l+1),Q} = \mathsf{g}\left(\sum_{\mathsf{e}' \in \mathcal{N}_{\mathsf{e}} \cup \{\mathsf{e}\}} a_{\mathsf{e},\mathsf{e}'} \mathbf{W}^{(l),Q} \otimes \mathbf{h}_{\mathsf{e}'}^{(l),Q}\right) \tag{14}$$

where $\mathcal{N}_{\mathsf{e}}$ is the set of entity neighbors of entity node e in $\mathcal{G}$. We also adapt DistMult (Yang et al., 2015) as a decoder module to compute the score of *(h, r, t)* as:

$$f\left(h,r,t\right) = \left\langle \textsc{Vec}\left(\mathbf{h}_h^{(L),Q}\right), \textsc{Vec}\left(\boldsymbol{v}_r^{Q}\right), \textsc{Vec}\left(\mathbf{h}_t^{(L),Q}\right) \right\rangle$$

where $\mathbf{h}_h^{(L),Q}$ and $\mathbf{h}_t^{(L),Q}$ are taken from the last QGNN layer; $\mathbf{h}_h^{(0),Q} = \boldsymbol{v}_h^{Q}$ and $\mathbf{h}_t^{(0),Q} = \boldsymbol{v}_t^{Q}$ are the quaternion embeddings of head entity h and tail entity t, respectively; $\boldsymbol{v}_r^{Q}$ is the quaternion embedding of relation r; and $\boldsymbol{v}_h^{Q}$, $\boldsymbol{v}_r^{Q}$, and $\boldsymbol{v}_t^{Q}$ are learned during training. $\textsc{Vec}(.)$ denotes a concatenation of the four components of the quaternion vector as mentioned in Equation 11; and $\langle\rangle$ denotes a multiple-linear dot product.

Similarity, we can also adopt GCN as the encoder module. We name our QGNN-based KG embedding model as **SimQGNN** and its variant using GCN as **SimRGCN**.

We then employ the Adam optimizer (Kingma and Ba, 2014) to train our proposed model by minimizing the binary cross-entropy loss function (Dettmers et al., 2018) as:

$$\mathcal{L} = -\sum_{(h,r,t) \in \{G \cup G'\}} \left(\textsc{L}_{(h,r,t)} \log\left(p_{(h,r,t)}\right) + \left(1 - \textsc{L}_{(h,r,t)}\right) \log\left(1 - p_{(h,r,t)}\right)\right) \tag{15}$$

$$\text{in which, } \textsc{L}_{(h,r,t)} = \begin{cases} 1 & \text{for } (h,r,t) \in G \\ 0 & \text{for } (h,r,t) \in G' \end{cases}$$

where $p_{(h,r,t)} = \mathsf{sigmoid}\left(f(h,r,t)\right)$. G and G' are collections of valid and invalid triples, respectively.

5. Experimental results

5.1. QGNN for graph classification

Datasets We evaluate our QGNN on well-known datasets consisting of three social network datasets (i.e., COLLAB, IMDB-B and IMDB-M) and four bioinformatics datasets (i.e., DD, MUTAG, PROTEINS, and PTC) (Yanardag and Vishwanathan, 2015). The social network datasets do not have available node features; thus, we follow (Niepert et al., 2016; Zhang et al., 2018; Xu et al., 2019) to use node degrees as features on these datasets.

Evaluation protocol We follow (Xu et al., 2019; Maron et al., 2019a; Seo et al., 2019; Chen et al., 2019) 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 reported either in the original papers or in (Verma and Zhang, 2018; Xinyi and Chen, 2019; Chen et al., 2019; Seo et al., 2019; Xu et al., 2019).

Training protocol We usually evaluate the proposed models on benchmark datasets where the Euclidean feature vectors $\mathbf{h}_v$ are typically given and pre-fixed. Hence we set the same $\mathbf{h}_v$ to the four components of $\mathbf{h}_v^{(0),Q}$ as: $\mathbf{h}_{v,r}^{(0)} = \mathbf{h}_{v,i}^{(0)} = \mathbf{h}_{v,j}^{(0)} = \mathbf{h}_{v,k}^{(0)} = \mathbf{h}_v$ This is similar to the way applying the Quaternion space on grayscale images (Zhu et al., 2018). 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 $\{8, 16, 32, 64, 128\}$. We set the batch size to 4 and use the Adam optimizer (Kingma and Ba, 2014) with the initial learning rate in $\{5e^{-5}, 1e^{-4}, 5e^{-4}, 1e^{-3}\}$. We run up to 100 epochs to evaluate our trained model.

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

Model	COLLAB	IMDB-B	IMDB-M	DD	PROTEINS	MUTAG	PTC
PSCN (2016)	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
GCN (2017)	79.00 ± 1.80	74.00 ± 3.40	51.90 ± 3.80	–	76.00 ± 3.20	85.60 ± 5.80	64.20 ± 4.30
GraphSAGE (2017a)	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 (2018)	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 (2018)	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 (2018)	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 (2019)	79.62 ± 0.91	73.10 ± 4.83	50.27 ± 2.65	75.38 ± 4.17	76.28 ± 3.63	86.67 ± 6.88	–
GIN-0 (2019)	80.20 ± 1.90	75.10 ± 5.10	52.30 ± 2.80	–	76.20 ± 2.80	89.40 ± 5.60	64.60 ± 7.00
IEGN (2019b)	77.92 ± 1.70	71.27 ± 4.50	48.55 ± 3.90	–	75.19 ± 4.30	84.61 ± 10.0	59.47 ± 7.30
PPGN (2019a)	81.38 ± 1.42	73.00 ± 5.77	50.46 ± 3.59	–	77.20 ± 4.73	90.55 ± 8.70	66.17 ± 6.54
GFN (2019)	**81.50 ± 2.42**	73.00 ± 4.35	51.80 ± 5.16	78.78 ± 3.49	76.46 ± 4.06	90.84 ± 7.22	–
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**

Experimental results Table 1 presents the accuracy results of our QGNN and other up-to-date baselines. In general, our QGNN produces state-of-the-art accuracies on most datasets; hence this demonstrates a notable impact of our model. Especially, QGNN outperforms the existing baseline models and produces state-of-the-art accuracies on IMDB-B, IMDB-M, DD, PROTEINS, and PTC, respectively. QGNN also obtains competitive accuracies on COLLAB and MUTAG, respectively. Furthermore, compared to GCN, the obtained results demonstrate the effectiveness of QGNN to generalize GCN within the Quaternion space for the graph classification task.

5.2. QGNN for node classification

Datasets We use three well-known benchmark datasets consisting of Cora, Citeseer (Sen et al., 2008) and Pubmed (Namata et al., 2012) that are citation networks.

Evaluation protocol As mentioned in (Fey and Lenssen, 2019; Pei et al., 2020), the experimental setup used in (Kipf and Welling, 2017; Veličković et al., 2018) 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 (Yang et al., 2016). Therefore, for a fair comparison, we use the same 10 random data splits used in (Pei et al., 2020), 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 (Pei et al., 2020) to report the mean classification accuracy on the test sets over the 10 data splits.

Training protocol We also we set the same $\mathbf{h}_v$ to the four components of $\mathbf{h}_v^{(0),Q}$. The architecture used in (Pei et al., 2020) is a 2-layer GCN, wherein the hidden sizes are 16 for Cora and Citeseer, and 64 for Pubmed. Hence, we construct one QGNN layer followed by a prediction layer and then a $\mathsf{softmax}$ layer (referring to our QGNN for node classification as mentioned in Equation 10). 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. We also provide the accuracy results of Hyperbolic Graph Convolutional Neural Networks (HGCN) (Chami et al., 2019) following these evaluation and training protocols.

Table 2: Node classification accuracies (%).

Dataset	GAT	GCN	HGCN	QGNN
Cora	86.37	85.77	86.09	**87.48**
Citeseer	74.32	73.68	74.84	**76.03**
Pubmed	87.62	**88.13**	87.13	87.65

Experimental results Table 2 presents the node classification accuracies, where the results of GCN and GAT are also taken from (Pei et al., 2020). Our QGNN achieves the accuracies of 87.48%, 76.03%, and 87.65% on Cora, Citeseer, and Pubmed respectively. In particular, QGNN outperforms GCN, GAT and HGCN on Cora and Citeseer, and produces competitive results on Pubmed, e.g., QGNN obtains absolute gains of 1.39%, 1.19%, and 0.52% over HGCN on Cora, Citeseer, and Pubmed, respectively.

5.3. QGNN for knowledge graph completion (SimQGNN)

We evaluate our proposed method for the knowledge graph completion task, i.e., link prediction (Bordes et al., 2013), which aims to predict a missing entity given a relation with another entity, e.g., inferring a head entity h given $(?, r, t)$ or inferring a tail entity t given $(h, r, ?)$. The results are computed by ranking the scores returned by the score function f on triples in the test set.

Datasets Safavi and Koutra (2020) point out issues with existing KG completion datasets and thus present three new and more appropriately difficult benchmark datasets CoDEx-S, CoDEx-M, and CoDEx-L. These three open-domain CoDEx datasets are derived from Wikidata and Wikipedia to cover more diverse and interpretable content and make a more challenging prediction task. Therefore, we use these new datasets in our experiments.

Evaluation protocol Following Bordes et al. (2013), for each valid test triple (h, r, t), we replace either h or t by each of all other entities to create a set of corrupted triples. We also use the "Filtered" setting protocol (Bordes et al., 2013). We rank the valid test triple and corrupted triples in descending order of their scores to calculate mean reciprocal rank (MRR) and Hits@10. The final scores on the test set are reported for the model which obtains the highest Hits@10 on the validation set.

Training protocol We set the batch size to 1024 and employ the Adam optimizer (Kingma and Ba, 2014) to train our model up to 4,000 epochs on CoDEx-S and CoDEx-M, and 2,000 epochs on CoDEx-L. We set the same dimension value for both the quaternion embedding size and the hidden size, wherein we vary the dimension value in $\{32, 64, 128, 256\}$. We also vary the number of hidden layers in $\{1, 2, 3\}$ and the Adam initial learning rate in $\{1e^{-4}, 5e^{-4}, 1e^{-3}, 5e^{-3}, 1e^{-2}\}$. We use grid search to select the best model checkpoints, wherein we compute the Hits@10 scores after each training epoch on the CoDEx-S and CoDEx-M validation sets and after each 5 training epochs on the CoDEx-L validation set.

For other baseline models including DistMult (Yang et al., 2015), SACN (Shang et al., 2019), R-GCN (Schlichtkrull et al., 2018), and CompGCN (Vashishth et al., 2020), we apply the same evaluation protocol. The training protocol is the same w.r.t. the optimizer, the hidden layers, the initial learning rate values, and the number of training epochs.

Table 3: Experimental results on the test sets. Hits@10 (H@10) is reported in %. The best scores are in bold, while the second best scores are in underline. $*$ denotes the results taken from (Safavi and Koutra, 2020), while ♣ denotes our own results for other baseline models. We get an out-of-memory for SACN on the large dataset CoDEx-L.

Method	**CoDEx-S**		**CoDEx-M**		**CoDEx-L**	
	MRR	H@10	MRR	H@10	MRR	H@10
TransE*	0.354	63.4	0.303	45.4	0.187	31.7
ComplEx*	**0.465**	<u>64.6</u>	**0.337**	<u>47.6</u>	0.294	40.0
ConvE*	<u>0.444</u>	63.5	0.318	46.4	0.303	42.0
TuckER*	<u>0.444</u>	63.8	<u>0.328</u>	45.8	<u>0.309</u>	43.0
DistMult♣	0.435	64.6	0.320	46.8	0.307	42.7
R-GCN♣	0.275	53.3	0.124	24.1	0.073	14.2
SACN♣	0.374	59.4	0.294	44.3	–	–
CompGCN♣	0.395	62.1	0.312	45.7	0.304	42.8
SimRGCN	0.427	<u>64.7</u>	0.322	47.5	0.307	<u>43.2</u>
SimQGNN	0.435	**65.2**	0.323	**47.7**	**0.310**	**43.7**

Experimental results We report the results on the CoDEx datasets for our proposed SimQGNN and SimRGCN, and other baseline models in Table 3. SimQGNN performs better than the more complicated models R-GCN, SACN, and CompGCN, e.g., SimQGNN obtains absolute Hits@10 improvements of 3.1%, 2.0%, and 0.9% over CompGCN on CoDEx-S, CoDEx-M, and CoDEx-L, respectively. In general, our SimQGNN obtains the highest Hits@10 scores and achieves competitive MRR scores on CoDEx-S and CoDEx-M. Furthermore, SimQGNN produces the highest MRR and Hits@10 scores on the large and more challenging dataset CoDEx-L.

It is worth mentioning that our SimRGCN outperforms R-GCN, SACN, and CompGCN; hence this clearly shows the effectiveness of our proposed method of exploring the vanilla GNNs directly on the single undirected graph $\mathcal{G}$ of entities built from the given knowledge graph G. Besides, it is noted that, compared to SimQGNN, the results degrade for SimRGCN, implying the advantage of our QGNN over GCN.

6. Conclusion

We propose Quaternion Graph Neural Networks (QGNN) to learn graph representations within the Quaternion space. Our proposed QGNN obtains state-of-the-art accuracies on well-known benchmark datasets for node classification and graph classification. Furthermore, regarding knowledge graphs, our QGNN-based embedding model, named SimQGNN, outperforms the existing models and produces state-of-the-art results on three benchmark datasets CoDEx for knowledge graph completion.[3]

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *ICLR*, 2015.

Ivana Balažević, Carl Allen, and Timothy M Hospedales. Tucker: Tensor factorization for knowledge graph completion. In *EMNLP*, pages 5185–5194, 2019.

Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning Structured Embeddings of Knowledge Bases. In *AAAI*, pages 301–306, 2011.

Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating Embeddings for Modeling Multi-relational Data. In *NeurIPS*, pages 2787–2795, 2013.

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

Ines Chami, Zhitao Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks. In *NeurIPS*, pages 4869–4880, 2019.

3. We introduce Gated Quaternion Graph Neural Networks (GQGNN) to learn graph representations in Appendix A. Our GQGNN-based text classification model, called TextQGNN, performs better than the existing models on well-known benchmark datasets for inductive text classification.

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

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *EMNLP*, 2014.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal Transformers. *ICLR*, 2019.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2D Knowledge Graph Embeddings. In *AAAI*, pages 1811–1818, 2018.

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

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

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017a.

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

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.

Diederik Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated Graph Sequence Neural Networks. *ICLR*, 2016.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *NeurIPS*, 2019a.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *ICLR*, 2019b.

Galileo Mark Namata, Ben London, Lise Getoor, and Bert Huang. Query-driven active surveying for collective classification. In *MLG Workshop*, 2012.

Dai Quoc Nguyen. Representation Learning for Graph-Structured Data. 2021. doi: 10.26180/14450496.v1. URL https://bridges.monash.edu/articles/thesis/Representation_Learning_for_Graph-Structured_Data/14450496.

Dai Quoc Nguyen, Dat Quoc Nguyen, Tu Dinh Nguyen, and Dinh Phung. Convolutional Neural Network-based Model for Knowledge Base Completion and Its Application to Search Personalization. *Semantic Web*, 10(5):947–960, 2019a. doi: 10.3233/SW-180318.

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

Dai Quoc Nguyen, Thanh Vu, Tu Dinh Nguyen, and Dinh Phung. Quatre: Relation-aware quaternions for knowledge graph embeddings. *arXiv preprint arXiv:2009.12517*, 2020.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning Convolutional Neural Networks for Graphs. In *ICML*, pages 2014–2023, 2016.

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.

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

Titouan Parcollet, Mirco Ravanelli, Mohamed Morchid, Georges Linarès, Chiheb Trabelsi, Renato De Mori, and Yoshua Bengio. Quaternion recurrent neural networks. In *ICLR*, 2019b.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-GCN: Geometric Graph Convolutional Networks. In *ICLR*, 2020.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, 2014.

Tara Safavi and Danai Koutra. CoDEx: A Comprehensive Knowledge Graph Completion Benchmark. In *EMNLP*, pages 8328–8350, 2020.

Michael Schlichtkrull, Thomas Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607, 2018.

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.

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

Chao Shang, Yun Tang, Jing Huang, Jinbo Bi, Xiaodong He, and Bowen Zhou. End-to-end structure-aware convolutional networks for knowledge base completion. In *AAAI*, volume 33, pages 3060–3067, 2019.

Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning With Neural Tensor Networks for Knowledge Base Completion. In *NeurIPS*, 2013.

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 *ACL*, pages 1494–1503, 2019.

Kristina Toutanova and Danqi Chen. Observed Versus Latent Features for Knowledge Base and Text Inference. In *CVSC Workshop*, pages 57–66, 2015.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex Embeddings for Simple Link Prediction. In *ICML*, pages 2071–2080, 2016.

VAnonymous. ReGVD: Revisiting graph neural networks for vulnerability detection. In *Submitted to ICLR 2022*, 2021. URL https://openreview.net/forum?id=wVFkD13GpeX. under review.

Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. In *ICLR*, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *NeurIPS*, pages 5998–6008, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *ICLR*, 2018.

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

Robert West, Evgeniy Gabrilovich, Kevin Murphy, Shaohua Sun, Rahul Gupta, and Dekang Lin. Knowledge Base Completion via Search-based Question Answering. In *Proceedings of the 23rd International Conference on World Wide Web*, pages 515–526, 2014.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *ICML*, pages 6861–6871, 2019a.

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

Zhang Xinyi and Lihui Chen. Capsule Graph Neural Network. *ICLR*, 2019.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. In *ICML*, pages 5453–5462, 2018.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful Are Graph Neural Networks? *ICLR*, 2019.

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

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *ICLR*, 2015.

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

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

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.

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

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

Shuai Zhang, Yi Tay, Lina Yao, and Qi Liu. Quaternion knowledge graph embeddings. In *NeurIPS*, pages 2731–2741, 2019.

Yufeng Zhang, Xueli Yu, Zeyu Cui, Shu Wu, Zhongzhen Wen, and Liang Wang. Every document owns its structure: Inductive text classification via graph neural networks. In *ACL*, pages 334–339, 2020b.

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

Appendix A. Gated Quaternion Graph Neural Networks (GQGNN)

We propose Gated Quaternion Graph Neural Networks (GQGNN), which can be seen as a generalization of Gated GNNs (Li et al., 2016) within the Quaternion space as follows:

$$
\begin{aligned}
\mathbf{a}_{\mathsf{v}}^{(l+1),Q} &= \sum_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}} a_{\mathsf{v},\mathsf{u}}\mathbf{h}_{\mathsf{u}}^{(l),Q} \\
\mathbf{z}_{\mathsf{v}}^{(l+1),Q} &= \sigma\left(\mathbf{W}^{z,Q}\otimes\mathbf{a}_{\mathsf{v}}^{(l+1),Q} + \mathbf{U}^{z,Q}\otimes\mathbf{h}_{\mathsf{v}}^{(l),Q}\right) \\
\mathbf{r}_{\mathsf{v}}^{(l+1),Q} &= \sigma\left(\mathbf{W}^{r,Q}\otimes\mathbf{a}_{\mathsf{v}}^{(l+1),Q} + \mathbf{U}^{r,Q}\otimes\mathbf{h}_{\mathsf{v}}^{(l),Q}\right) \\
\widetilde{\mathbf{h}_{\mathsf{v}}^{(l+1),Q}} &= \mathsf{g}\left(\mathbf{W}^{o,Q}\otimes\mathbf{a}_{\mathsf{v}}^{(l+1),Q} + \mathbf{U}^{o,Q}\otimes\left(\mathbf{r}_{\mathsf{v}}^{(l+1),Q}\odot\mathbf{h}_{\mathsf{v}}^{(l),Q}\right)\right) \\
\mathbf{h}_{\mathsf{v}}^{(l+1),Q} &= \left(1-\mathbf{z}_{\mathsf{v}}^{(l+1),Q}\right)\odot\mathbf{h}_{\mathsf{v}}^{(l),Q} + \mathbf{z}_{\mathsf{v}}^{(l+1),Q}\odot\widetilde{\mathbf{h}_{\mathsf{v}}^{(l+1),Q}}
\end{aligned}
\tag{16}
$$

where $\mathbf{z}$ and $\mathbf{r}$ are the update and reset gates; σ is the sigmoid function; $\odot$ is element-wise multiplication; and g is a nonlinear activation function (such as ReLU).

A.1. GQGNN for inductive text classification (TextQGNN)

We follow (Zhang et al., 2020b) to build a graph $\mathcal{G}$ for each textual document by representing unique words as nodes and co-occurrences between words as edges. We then employ our proposed GQGNN to update node representations. To obtain the graph embedding $\mathbf{e}_{\mathcal{G}}^{Q}$, we follow (VAnonymous, 2021) to define a readout function as:

$$\mathbf{e}_{\mathsf{v}}^{Q} = \sigma\left(\mathbf{w}^{\mathsf{T}}\textsc{Vec}\left(\mathbf{h}_{\mathsf{v}}^{(L),Q}\right)+\mathsf{b}\right)\odot \mathsf{g}\left(\mathbf{W}^{Q}\otimes\mathbf{h}_{\mathsf{v}}^{(L),Q}\right) \tag{17}$$

$$\mathbf{e}_{\mathcal{G}}^{Q} = \sum_{\mathsf{v}\in\mathcal{V}}\mathbf{e}_{\mathsf{v}}^{Q}\odot \mathsf{max\_pooling}\left\{\mathbf{e}_{\mathsf{v}}^{Q}\right\}_{\mathsf{v}\in\mathcal{V}} \tag{18}$$

where $\sigma\left(\mathbf{w}^{\mathsf{T}}\textsc{Vec}\left(\mathbf{h}_{\mathsf{v}}^{(L),Q}\right)+\mathsf{b}\right)$ acts as soft attention mechanisms over nodes. Finally, we also follow Equation 13 to perform the task. We name our GQGNN-based text classification model as **TextQGNN**.

A.2. Experimental setup and results

Datasets We follow (Yao et al., 2019; Zhang et al., 2020b) to use four benchmarks – R8, R52, Ohsumed, and MR.

Training protocol We also follow (Zhang et al., 2020b) to use random vectors or pre-trained Glove (Pennington et al., 2014) with the dimension size of 300 to initialize the Euclidean feature vectors. Then, for each Euclidean feature vector, we simply split it into four parts to create four corresponding components of a Quaternion feature vector with the dimension size of 75. We follow (Yao et al., 2019; Zhang et al., 2020b) to construct 2-layer steps. We set the quaternion hidden size to 96 and vary the learning rate in $\left\{1e^{-4}, 5e^{-4}, 1e^{-3}, 5e^{-3}\right\}$. We use the Adam optimizer to train the model up to 150 epochs to evaluate our trained model.

Evaluation protocol For each dataset, we report the mean accuracy and standard deviation over 10 times, wherein for each time, we randomly sample 10% text from the training set to construct the validation set for hyper-parameter turning.

Table 4: Text classification accuracies (%) on the test sets. Some baseline results are taken from (Yao et al., 2019). We report the new results of TextING for the model which obtains the highest accuracy on the validation set for a fair comparison.

Dataset	MR	R8	R52	Ohsumed
Bi-LSTM	77.68 ± 0.86	96.31 ± 0.33	90.54 ± 0.91	49.27 ± 1.07
fastText	75.14 ± 0.20	96.13 ± 0.21	92.81 ± 0.09	57.70 ± 0.49
TextGCN	76.74 ± 0.20	97.07 ± 0.10	93.56 ± 0.18	68.36 ± 0.56
TextING	78.86 ± 0.26	96.90 ± 0.23	93.34 ± 0.24	69.72 ± 0.30
TextQGNN	78.93 ± 0.29	97.02 ± 0.28	94.45 ± 0.35	69.93 ± 0.31

Experimental results Table 4 presents the text classification accuracies of our TextQGNN and the baselines. In general, our TextQGNN works better than the baselines on the benchmark datasets.