MONASH University

REPRESENTATION LEARNING

FOR GRAPH-STRUCTURED DATA

by

Dai Quoc Nguyen

A thesis submitted for the degree of

DOCTOR OF PHILOSOPHY

Faculty of Information Technology
Monash University
Melbourne, Australia

December 2020

ABSTRACT

A graph is a connected network of nodes and edges. This type of graph-structured data is a fundamental mathematical representation and ubiquitous. They found applications in virtually all aspects of our daily lives from pandemic outburst response, information retrieval to circuit design, to name a few. In machine learning and data science, learning and inference from graphs have been one of the most important research topics. However, as data grow unprecedentedly in volume and complexity in modern time, traditional learning methods for graph are mostly inadequate to model increasing complexity, to harness rich contextual information as well as to scale with large-scale graphs. The recent rise of deep learning, and in turn, of representation learning field has radically advanced machine learning research in general, and pushing the frontier of graph learning. In particular, the notion of graph representation learning has recently emerged as a new promising learning paradigm, which aims to learn a parametric mapping function that embeds nodes, subgraphs, or the entire graph into low-dimensional continuous vector spaces. The central challenge to this endeavor is to learn rich classes of complex functions to capture and preserve the graph structural information as much as possible and also be able to geometrically represent the structural information in the embedded space.

In the quest to this research challenge, the first goal in this thesis is to develop new effective learning and embedding methods for undirected graphs, that can exploit side, context and relational information as well as new representation space. Consequently, we propose a new graph neural network model which, by leveraging a transformer self-attention network, induces a powerful aggregation function to improve graph classification performance. We further present two new unsupervised embedding models to learn effective embeddings not only for existing nodes but also for new nodes. Lastly, to enrich the space in which graph representation operates, we introduce a novel form of quaternion graph neural

networks to move beyond the Euclidean space to the Quaternion space, hence reducing the model size and improving the embedding quality.

Our second research goal is to develop advanced embedding models that can better encode relationships and correlate information in knowledge graphs. To this end, we propose two new embedding models to capture global relationships and translation characteristics between entities and relations. We also present another advanced embedding model to enhance relation-aware correlations between head and tail entities within the Quaternion space. Furthermore, we consider two other applications of triple classification and search personalisation and introduce a new embedding model to memorise and encode potential dependencies among relations and entities.

In all cases, the thesis has focused on developing novel and advanced embedding models to address the challenges for the two most popular types of graphs: undirected graph and knowledge graph, with extensive experimental evaluation, benchmarking with current state-of-the-art methods, and post-analysis to demonstrate the merits of the proposed methodologies.

PAPERS DURING CANDIDATURE

This thesis is the combination of the following papers:

Peer-reviewed Journal Articles

- **Dai Quoc Nguyen**, Dat Quoc Nguyen, Tu Dinh Nguyen and Dinh Phung. **2019**. A Convolutional Neural Network-based Model for Knowledge Base Completion and Its Application to Search Personalisation. *Semantic Web*, 10(5):947-960, 2019. DOI: 10.3233/SW-180318. (SCIE, JCR IF 2019: 3.524).

Peer-reviewed Conference Papers

- **Dai Quoc Nguyen**, Tu Dinh Nguyen, Dat Quoc Nguyen and Dinh Phung. **2018**. A Novel Embedding Model for Knowledge Base Completion Based on Convolutional Neural Network. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2018)*, pages 327-333.

- **Dai Quoc Nguyen**, Thanh Vu, Tu Dinh Nguyen, Dat Quoc Nguyen and Dinh Phung. **2019**. A Capsule Network-based Embedding Model for Knowledge Graph Completion and Search Personalisation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2019)*, pages 2180-2189.

- **Dai Quoc Nguyen**, Tu Dinh Nguyen and Dinh Phung. **2020**. A Relational Memory-based Embedding Model for Triple Classification and Search Personalisation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020)*, pages 3429–3435.

- **Dai Quoc Nguyen**, Tu Dinh Nguyen and Dinh Phung. **2020**. 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). https://doi.org/10.1007/978-3-030-67664-3_22

- **Dai Quoc Nguyen**, Tu Dinh Nguyen, Dat Quoc Nguyen and Dinh Phung. **2020**. A Capsule Network-based Model for Learning Node Embeddings. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM 2020)*. https://doi.org/10.1145/3340531.3417455

Preprint papers

- **Dai Quoc Nguyen**, Tu Dinh Nguyen and Dinh Phung. Universal Graph Transformer Self-Attention Networks. *arXiv preprint arXiv:1909.11855, 2019*.
- **Dai Quoc Nguyen**, Tu Dinh Nguyen and Dinh Phung. Quaternion Graph Neural Networks. *arXiv preprint arXiv:2008.05089*.
- **Dai Quoc Nguyen**, Thanh Vu, Tu Dinh Nguyen and Dinh Phung. QuatRE: Relation-Aware Quaternions for Knowledge Graph Embeddings. *arXiv preprint arXiv:2009.12517*.

The extended abstracts of the following submissions, "Quaternion Graph Neural Networks" and "QuatRE: Relation-Aware Quaternions for Knowledge Graph Embeddings", have been accepted to the NeurIPS 2020 Workshop on Differential Geometry meets Deep Learning (DiffGeo4DL).

ORIGINALITY STATEMENT

Thesis including published works declaration. I hereby declare that this thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

This thesis includes six original papers published in peer-reviewed conferences and journals. The core themes of the thesis are knowledge graph embeddings and graph neural networks. The ideas, development and writing up of all the papers in the thesis were the principal responsibility of myself, the student, working within the Faculty of Information Technology under the supervision of Dinh Phung, Tu Dinh Nguyen, and Geoff Webb.

The inclusion of co-authors reflects the fact that the work came from active collaboration between researchers and acknowledges input into team-based research.

In the case of Chapters 3 and 4, my contribution to the work involved the following:

Thesis Chapter	**Publication Title**	**Status**	**Nature and % of student contribution**	**Co-author name(s) Nature and % of co-author's contribution**	**Co-author(s) Monash student Y/N***
3.2.2	A Self-Attention Network based Node Embedding Model	Accepted	Proposing research ideas, implementing models, conducting experiments, and writing papers. 60%.	1) Tu Dinh Nguyen. Discussing idea and writing paper. 20%. 2) Dinh Phung. Discussing idea and writing paper. 20%.	N
3.2.2	A Capsule Network-based Model for Learning Node Embeddings	Accepted	Proposing research ideas, implementing models, conducting experiments, and writing papers. 55%.	1) Tu Dinh Nguyen. Discussing idea and writing paper. 15%. 2) Dat Quoc Nguyen. Discussing idea and writing paper. 15%. 3) Dinh Phung. Discussing idea and writing paper. 15%.	N

4.2.1	A Novel Embedding Model for Knowledge Base Completion Based on Convolutional Neural Network	Accepted	Proposing research ideas, implementing models, conducting experiments, and writing papers. 55%.	1) Tu Dinh Nguyen. Discussing idea and writing paper. 15%. 2) Dat Quoc Nguyen. Discussing idea and writing paper. 15%. 3) Dinh Phung. Discussing idea and writing paper. 15%.	N
4.2.1	A Convolutional Neural Network-based Model for Knowledge Base Completion and Its Application to Search Personalisation	Accepted	Proposing research ideas, implementing models, conducting experiments, and writing papers. 55%.	1) Dat Quoc Nguyen. Discussing idea and writing paper. 15%. 2) Tu Dinh Nguyen. Discussing idea and writing paper. 15%. 3) Dinh Phung. Discussing idea and writing paper. 15%.	N
4.2.1	A Capsule Network-based Embedding Model for Knowledge Graph Completion and Search Personalisation	Accepted	Proposing research ideas, implementing models, conducting experiments, and writing papers. 50%.	1) Thanh Vu. Discussing idea, preparing data, and writing paper. 15%. 2) Tu Dinh Nguyen. Discussing idea and writing paper. 10%. 3) Dat Quoc Nguyen. Discussing idea and writing paper. 10%. 4) Dinh Phung. Discussing idea and writing paper. 10%.	N
4.2.3	A Relational Memory-based Embedding Model for Triple Classification and Search Personalisation	Accepted	Proposing research ideas, implementing models, conducting experiments, and writing papers. 60%.	1) Tu Dinh Nguyen. Discussing idea and writing paper. 20%. 2) Dinh Phung. Discussing idea and writing paper. 20%.	N

I have / have not renumbered sections of submitted or published papers in order to generate a consistent presentation within the thesis.

Student signature: **Date:**

I hereby certify that the above declaration correctly reflects the nature and extent of the student's and co-authors' contributions to this work. In instances where I am not the responsible author I have consulted with the responsible author to agree on the respective contributions of the authors.

Main Supervisor signature: **Date:**

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my principal supervisor, Dinh Phung, for his encouragement, valuable guidance, and helpful feedback throughout the last three years. I have been very fortunate to have had an opportunity to work with him.

I would like to give my honest appreciation to my co-supervisor, Tu Dinh Nguyen, for his great support throughout this long scientific journey. He always appears when I need help and responds to queries so helpfully and promptly.

I would like to thank my twin brother, Dat Quoc Nguyen, and Thanh Vu for their collaboration. I really enjoyed working with them to develop knowledge graph embedding models and Vietnamese natural language processing toolkits.

I am also thankful to my co-supervisor, Geoff Webb, and the panel members, Lan Du, Vincent Lee, and Mario Boley, for their services, supports and feedback during my candidature.

I would like to express my gratitude to my colleagues Nhan Dam, Quan Hoang, Van Nguyen, Mahmoud Mohammad, Tuan-Anh Bui, Trung Le, Viet Huynh, Khanh Nguyen, Hung Vu, Ethan Zhao, Thanh Nguyen-Duc, Tuan Nguyen, Son Dao, Tu Vu, Cuong Xuan Chu, and Thanh-Nam Pham.

I would like to thank Viet-Anh Pham, Hong Nguyen, Tu Dinh Nguyen, Van-Anh Dang, Long Duong, Trang Vu, Thanh Vu, Trang Pham, Thanh Nguyen, Binh-Minh Nguyen, Trung Thai, Thu-Huyen Tran, Viet Bui, Viet-Dung Nguyen, Vi Truong, Trung Le, and Khanh-Linh Le, for all the exciting conversations we had during lunches and dinners.

Many thanks go to Thao-My Nguyen, Linh Vo, Ha Nguyen, Viet Nguyen, Trang Vo, Thy Nguyen, Khai Nguyen, Tho Nguyen, Minh Nguyen, Nha Phan, My-Dan Pham, Allan Nguyen, Long Le, Phuc Pham, and Thuan-Anh Bui. I enjoyed all the fun moments we had while playing billiard and badminton.

This thesis would not have been possible without the support and love of my parents Bao Quoc Nguyen and En Thi Nguyen.

Dedicated to my parents ♡

Contents

List of Figures

List of Tables

Abbreviations

NLP	natural language processing
ML	machine learning
KG	knowledge graph
CNN	convolutional neural network
GCN	graph convolutional network
GNN	graph neural network
QGNN	quaternion graph neural network
WN	Wordnet
FB	Freebase
MRR	mean reciprocal rank
RQ	research question
w.r.t	with respect to

Chapter 1

Introduction

1.1 Motivations

Graph-structured data is ubiquitous with countless real-world applications. In general, a graph can be viewed as a network of nodes and edges, wherein nodes represent the individual entities, and edges encode relationships among those entities. One example is in an online forum, wherein a discussion thread can be constructed as a graph of nodes representing the users and edges indicating the interactions such as responses and reactions between users (Yanardag and Vishwanathan, 2015). Besides, the graph structures found in social networks, molecular networks, biological protein-protein networks, recommender systems could be extremely complicated. Hence, it is worth exploring prospective mechanisms to deal with the unprecedented growth in volumes and problem complexity of graph-structured data.

Early machine learning models have limitations to work on graph-structured data. A common approach is to extract structural information such as summary graph statistics, e.g., kernel functions (Gärtner et al., 2003). Another approach is to carefully design hand-engineered features to measure the local neighbourhood structures (Liben-Nowell and Kleinberg, 2007). These methods, however, suffer from two significant drawbacks. First, designing good features requires prior knowledge, hence needs domain experts, and is usually time-consuming. Second, hand-engineered features are inflexible and not straightforward

to adapt to the graphs evolving over time.

Graph representation learning has recently emerged as a new promising learning paradigm, which has been offering numerous successful applications such as semantic searching and ranking (Kasneci et al., 2008; Schuhmacher and Ponzetto, 2014; Xiong et al., 2017), question answering (Zhang et al., 2016; Hao et al., 2017), traffic prediction (Cui et al., 2018), learning physics engines (Sanchez-Gonzalez et al., 2018), and advertising and recommending items to users (Ying et al., 2018a; Wang et al., 2018). The key idea 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. The challenge is that the learned embedding function needs to capture and preserve the graph structural information as much as possible and also be able to geometrically represent the structural information in the embedded space. In this thesis, we develop novel and advanced embedding models to address six research questions for the two most popular types of graphs: **undirected graph** and **knowledge graph**.

1.1.1 Representation learning for undirected graph

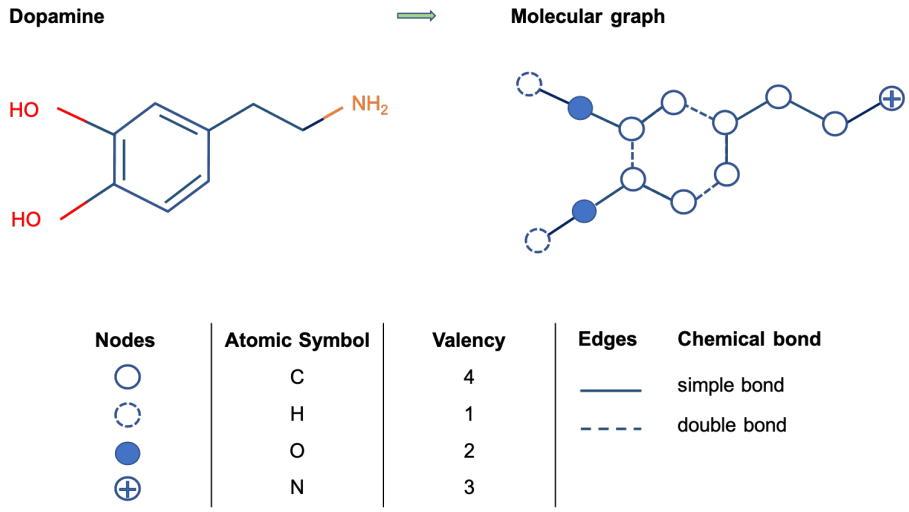

Figure 1.1: Illustration of a molecular graph. This figure is drawn based on (Nouleho Ilemo et al., 2019).

Undirected graphs contain bidirectional edges that do not have a direction. Undirected

graphs are predominant and have been utilised in many necessary fields, e.g., they can be used as a representation for molecules (Gärtner et al., 2003), as illustrated in Figure 1.1. Thus, learning node and graph vector representations is essential in both industry and academic applications (Cai et al., 2018; Chen et al., 2018a; Zhang et al., 2018b).

Recently, **graph neural networks** (GNNs) have become a central strand to learn low-dimensional continuous embeddings for nodes and graphs (Hamilton et al., 2017b; Wu et al., 2019b). Compared to early approaches such as graph kernel-based methods (Gärtner et al., 2003) and word embedding-based models (Perozzi et al., 2014), GNN-based approaches provide faster and practical training, higher accuracy, and state-of-the-art results on benchmark datasets for downstream tasks such as node and graph classifications (Kipf and Welling, 2017; Xu et al., 2019). Therefore, we raise three needed research questions and propose novel models to address these research questions (RQ).

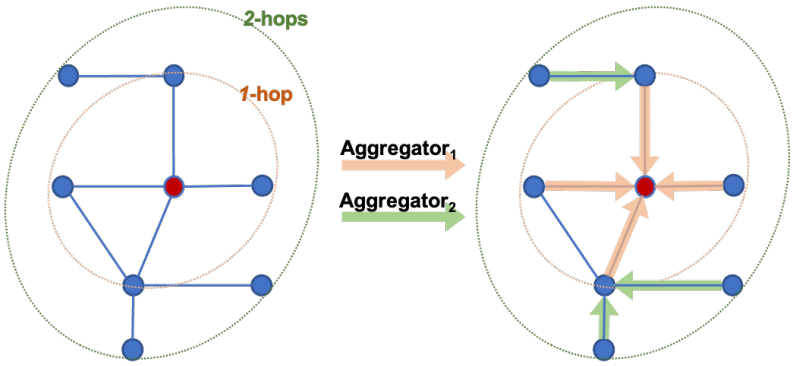

Figure 1.2: Illustration of an aggregation operation. This figure is drawn based on (Hamilton et al., 2017a).

• **RQ 1.** In general, GNNs use an aggregation function to update the vector representation of each node by transforming and aggregating the vector representations of its neighbours (Kipf and Welling, 2017; Hamilton et al., 2017a; Veličković et al., 2018), as illustrated in Figure 1.2. Then GNNs apply a graph-level pooling function (i.e., a readout operation such as simple sum pooling) to obtain graph embeddings (Gilmer et al., 2017; Zhang et al., 2018a; Ying et al., 2018b; Verma and Zhang, 2018; Xu et al., 2019). To further improve the GNN performance, our first research question is: "*How can we develop an advanced aggregation function to better update node representations from their*

neighbours?"

• **RQ 2.** Existing GNN models mainly focus on the *transductive* setting, where a model is trained using the entire input graph, i.e., the model requires all nodes with a fixed graph structure during training and lacks the flexibility in inferring embeddings for new nodes (Perozzi et al., 2014; Wang et al., 2016; Kipf and Welling, 2017). By contrast, a more important setup, but less mentioned, is the *inductive* setting, where only a part of the input graph is used to train the model, and then the learned model is used to infer embeddings for new nodes (Yang et al., 2016). Several attempts have additionally been made for the inductive settings (Duran and Niepert, 2017; Hamilton et al., 2017a). Working on the inductive setting is particularly more difficult than that on the transductive setting due to lacking the ability to generalise to the graph structure for new nodes. Hence, our second research question is: "*How can we develop an effective learning process to infer embeddings for new nodes?*"

• **RQ 3.** It is worth to note that most of the existing GNNs learn node and graph embeddings within the Euclidean vector space. However, for complex graphs such as protein interaction networks and social networks, the learned Euclidean embeddings have high distortion (Chami et al., 2019). It also has been noted in (Xu et al., 2019; Pei et al., 2020) 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, when increasing the number of hidden layers, the existing GNNs (Kipf and Welling, 2017; Hamilton et al., 2017a; Veličković et al., 2018; Xu et al., 2019) are not working very efficiently anymore since the number of parameters grows quickly. Our third research question is: "*How can we move beyond the Euclidean space to learn better graph representations and reduce the number of model parameters?*"

1.1.2 Knowledge graph embeddings

Knowledge graphs (KGs) can be viewed as *directed multi-relational networks* to represent directional relationships between entities in the form of triples *(head, relation, tail)* denoted

as *(h, r, t)*, e.g., (Patti, born_in, Miami), as illustrated in Figure 1.3. KGs are useful resources for many NLP and information retrieval applications such as semantic search and question answering (Wang et al., 2017). However, large knowledge graphs, such as YAGO (Suchanek et al., 2007) and Freebase (Bollacker et al., 2008), even at the scale of billions of triples, are still incomplete, i.e., missing a lot of valid triples (West et al., 2014). For example, in Freebase, 71% of the roughly 3 million people have no known place of birth, 94% have no known parents, and 99% have no known ethnicity (West et al., 2014). Therefore, existing research efforts have focused on inferring missing triples in KGs, i.e., predicting whether a triple not in KGs is likely to be valid or not (Bordes et al., 2011). Consequently, many **knowledge graph embedding** models have been proposed to learn vector representations for entities and relations and return a score for each triple, such that valid triples have higher scores than invalid ones (Bordes et al., 2013; Socher et al., 2013a), e.g., the score of the valid triple (Melbourne, city_Of, Australia) is higher than the score of the invalid one (Melbourne, city_Of, Germany). To this end, we also concern three important research questions and introduce novel KG embedding models to deal with these questions.

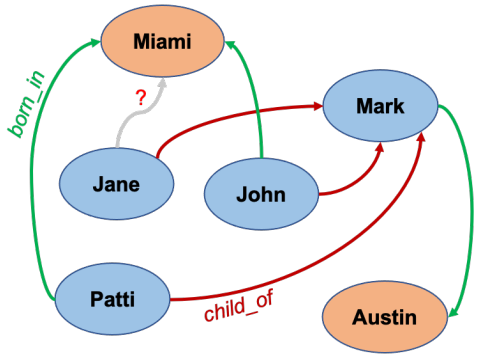

Figure 1.3: Illustration of an incomplete knowledge graph. This figure is drawn based on (Nguyen, 2020).

• **RQ 4.** Conventional embedding models often employ simple linear operators such as addition, subtraction, and multiplication. For example, TransE (Bordes et al., 2013), the most well-known embedding model for KGs, uses translations within a latent space to capture relationships between the head and tail entities, so that the embedding $\boldsymbol{v}_h$ of the

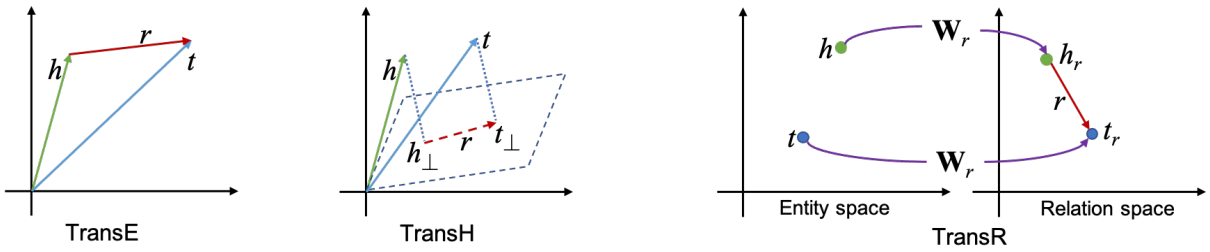

Figure 1.4: Illustration of translation-based models. This figure is drawn based on (Wang et al., 2014; Lin et al., 2015b).

head entity plus the embedding $\boldsymbol{v}_r$ of the relation is close to the embedding $\boldsymbol{v}_t$ of the tail entity, i.e., $\boldsymbol{v}_h + \boldsymbol{v}_r \approx \boldsymbol{v}_t$, where $\boldsymbol{v}_h$, $\boldsymbol{v}_r$, and $\boldsymbol{v}_t \in \mathbb{R}^n$, as illustrated in Figure 1.4. TransE leads to expand early embedding models such as TransH (Wang et al., 2014), TransR (Lin et al., 2015b), TransD (Ji et al., 2015), STransE (Nguyen et al., 2016), DISTMULT (Yang et al., 2015), and ComplEx (Trouillon et al., 2016). Since ConvE (Dettmers et al., 2018) has successfully leveraged convolutional neural networks (LeCun et al., 1998) to score the triples and obtain state-of-the-art results for knowledge graph completion, we see a potential strategy of developing deep neural networks for knowledge graph embeddings. Our fourth research question is: "*How can we develop deep KG embedding approaches to better model relationships among entities?*"

• **RQ 5.** Most of the existing models focus on embedding entities and relations within the Euclidean vector space (Bordes et al., 2013; Wang et al., 2014; Lin et al., 2015b; Yang et al., 2015; Dettmers et al., 2018; Nguyen et al., 2018). Moving beyond the Euclidean vector space, ComplEx (Trouillon et al., 2016) and RotatE (Sun et al., 2019) consider the complex vector space, MuRP (Balažević et al., 2019a) and AttH (Chami et al., 2020) leverage the hyperbolic space, and QuatE (Zhang et al., 2019) embeds entities and relations within the Quaternion space. However, these existing models have a limitation in capturing the correlations between the head and tail entities. Thus, our fifth research question is: "*How can we increase the correlations between the entities in KGs beyond the Euclidean space?*"

• **RQ 6.** The existing KG embedding models show promising performances mainly for

knowledge graph completion, where the goal is to infer a missing entity given a relation and another entity. But in other applications, less mentioned, such as triple classification (Socher et al., 2013a) that aims to predict whether a given triple is valid, and search personalisation (Vu et al., 2017) that aims to re-rank the relevant documents returned by a user-oriented search system given a query, these models do not effectively capture potential dependencies among entities and relations from existing triples to predict new triples. As a consequence, our last research question is: "*How can we develop an advanced KG embedding model for two other applications of triple classification and search personalisation?*"

1.2 Aims and Contributions

By investigating the research questions mentioned above, this thesis aims to:

- Develop new graph neural networks for undirected graphs.
- Develop advanced embedding models for knowledge graphs.

To accomplish these aims, we present a novel class of graph embedding models to address **RQ 1, 2**, and **3** in Chapter 3 and introduce our new KG embedding models to address **RQ 4, 5**, and **6** in Chapter 4. The main contributions of this thesis are highlighted as follows:

- **RQ 1: Graph transformer self-attention networks.** The transformer self-attention network (Vaswani et al., 2017; Dehghani et al., 2019) has been extensively used in research domains such as computer vision, image processing, and natural language processing. But it has not been actively used in GNNs where constructing an advanced aggregation function is essential. Our proposed model, named ***U2GNN***, is an effective GNN model which, by leveraging a transformer self-attention mechanism followed by a recurrent transition, induces a powerful aggregation function to learn graph representations. Our U2GNN achieves state-of-the-art accuracies on benchmark datasets for the graph classification task.

- **RQ 2: Learning node embeddings for new nodes.** We propose two new unsupervised embedding models – ***SANNE*** and ***Caps2NE*** – whose central ideas are to employ a transformer self-attention network (Vaswani et al., 2017) and a capsule network (Sabour et al., 2017) respectively. Our two models aim to produce effective embeddings not only for existing nodes but also for new nodes. The proposed SANNE and Caps2NE obtain state-of-the-art results on well-known benchmark datasets for the node classification task.

- **RQ 3: Quaternion graph neural networks.** We propose quaternion graph neural networks (***QGNN***) to generalise graph convolutional networks (Kipf and Welling, 2017) within the Quaternion space to learn quaternion embeddings for nodes and graphs. The Quaternion space, a hyper-complex vector space, provides powerful 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 learn better graph representations. 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.

- **RQ 4: Deep knowledge graph embedding models.** We propose a new embedding model, named ***ConvKB***, which advances state-of-the-art models by employing a convolutional neural network (LeCun et al., 1998). ConvKB aims to capture global relationships and translation characteristics among entities and relations in knowledge graphs. Besides, we introduce ***CapsE***, an extension of ConvKB, to explore a novel application of the capsule network (Sabour et al., 2017). Unlike the traditional modelling design of the capsule network where capsules are constructed by splitting feature maps, we use capsules to model the entries at the same dimension in the entity and relation embeddings. Both ConvKB and CapsE achieve better performance than previous KG embedding models on benchmark datasets for the knowledge graph completion task.

- **RQ 5: Relation-aware quaternions for knowledge graph embeddings.** We

propose an effective embedding model, named ***QuatRE***, to learn quaternion embeddings for entities and relations in knowledge graphs. QuatRE aims to enhance correlations between head and tail entities given a relation within the Quaternion space. QuatRE achieves this goal by further associating each relation with two relation-aware rotations, which are used to rotate quaternion embeddings of the head and tail entities, respectively. QuatRE produces state-of-the-art results also for the knowledge graph completion task.

- **RQ 6: Memory network for triple classification and search personalisation.** We introduce a new KG embedding model, named ***R-MeN***, that explores a transformer-based memory network (Santoro et al., 2018) to memorise and encode the potential dependencies among relations and entities. Our R-MeN considers each triple as a sequence of 3 input vectors that recurrently interact with memory using a transformer self-attention mechanism (Vaswani et al., 2017). R-MeN obtains better performance than previous models for triple classification and search personalisation.

1.3 Outline and Origins

The remaining parts of this thesis is outlined as follows:

- Chapter 2 presents necessary background and related work for graph representation learning and knowledge graph embeddings.

- Chapter 3 describes our proposed models for graph neural networks. This chapter is based on:

  - **Dai Quoc Nguyen**, Tu Dinh Nguyen and Dinh Phung. Universal Graph Transformer Self-Attention Networks. *arXiv preprint arXiv:1909.11855, 2019.*

  - **Dai Quoc Nguyen**, Tu Dinh Nguyen and Dinh Phung. **2020**. 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). https://doi.org/10.1007/978-3-030-67664-3_22

- **Dai Quoc Nguyen**, Tu Dinh Nguyen, Dat Quoc Nguyen and Dinh Phung. **2020**. A Capsule Network-based Model for Learning Node Embeddings. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM 2020)*. https://doi.org/10.1145/3340531.3417455

- **Dai Quoc Nguyen**, Tu Dinh Nguyen and Dinh Phung. Quaternion Graph Neural Networks. *arXiv preprint arXiv:2008.05089*.

- Chapter 4 details our proposed models for knowledge graph embeddings. This chapter is based on:

- **Dai Quoc Nguyen**, Tu Dinh Nguyen, Dat Quoc Nguyen and Dinh Phung. **2018**. A Novel Embedding Model for Knowledge Base Completion Based on Convolutional Neural Network. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2018)*, pages 327-333.

- **Dai Quoc Nguyen**, Dat Quoc Nguyen, Tu Dinh Nguyen and Dinh Phung. **2019**. A Convolutional Neural Network-based Model for Knowledge Base Completion and Its Application to Search Personalisation. *Semantic Web*, 10(5):947-960, 2019. DOI: 10.3233/SW-180318. (SCIE, JCR IF 2019: 3.524).

- **Dai Quoc Nguyen**, Thanh Vu, Tu Dinh Nguyen, Dat Quoc Nguyen and Dinh Phung. **2019**. A Capsule Network-based Embedding Model for Knowledge Graph Completion and Search Personalisation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2019)*, pages 2180-2189.

- **Dai Quoc Nguyen**, Thanh Vu, Tu Dinh Nguyen and Dinh Phung. QuatRE: Relation-Aware Quaternions for Knowledge Graph Embeddings. *arXiv preprint*

arXiv:2009.12517.

- **Dai Quoc Nguyen**, Tu Dinh Nguyen and Dinh Phung. **2020**. A Relational Memory-based Embedding Model for Triple Classification and Search Personalisation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020)*, pages 3429–3435.

- Chapter 5 provides concluding remarks by giving a summary of our work and future developments.

This thesis consists of **six** original papers published in peer-reviewed conferences and journals and **three** other preprint papers. Furthermore, the extended abstracts of the following **two** papers, "Quaternion Graph Neural Networks" and "QuatRE: Relation-Aware Quaternions for Knowledge Graph Embeddings", have also been accepted to the NeurIPS 2020 Workshop on Differential Geometry meets Deep Learning (DiffGeo4DL).

Chapter 2

Background

Contents

In this chapter, we review the literature related to this thesis. There are two main sections, wherein the first presents a brief overview of graph representation learning approaches, and the second provides a background of knowledge graph embedding methods.

2.1 Graph Representation Learning

Learning representations for graph-structured data is an important topic that has gained a substantial amount of attention recently (Hamilton et al., 2017b; Zhou et al., 2018). The common goal is to construct low-dimensional vectors for nodes, subgraphs, or the entire graph. Existing approaches can be categorised into three groups: (i) graph kernel-based methods, (ii) word embedding-based models, and (iii) graph neural networks.

Methods in the first group build vectors of frequencies of "atomic subgraphs" decom-

posed from a given graph. Typical types of atomic subgraphs include random walks (Gärtner et al., 2003; Kashima et al., 2003; Vishwanathan et al., 2010), shortest paths (Borgwardt and Kriegel, 2005), graphlets (Shervashidze et al., 2009), and Weisfeiler-Lehman subtree patterns (Shervashidze et al., 2011). In the second direction, several word embedding-based models (Perozzi et al., 2014; Tang et al., 2015; Grover and Leskovec, 2016) sample a large set of random walks, then treat each walk as a document whose words are nodes, and finally apply Word2Vec (Mikolov et al., 2013b) on such random walk set to learn node embeddings. Meanwhile, some other ones aim to obtain the embeddings of the atomic subgraphs (Yanardag and Vishwanathan, 2015), and the entire graphs (Narayanan et al., 2017; Ivanov and Burnaev, 2018) by aggregating node embeddings learned by Word2Vec or using Doc2Vec (Le and Mikolov, 2014). Recently, graph neural networks (GNNs) become an essential strand, forming the third direction to learn low-dimensional continuous representations for nodes and graphs, and achieve state-of-the-art performances for node and graph classification tasks (Scarselli et al., 2009; Hamilton et al., 2017b; Zhou et al., 2018; Wu et al., 2019b; Zhang et al., 2020).

In what follows, we first introduce the graph kernel-based methods in Section 2.1.1, then describe the word embedding-based models in Section 2.1.2, and finally focus on presenting graph neural networks in Section 2.1.3.

2.1.1 Graph kernel-based methods

In general, graph kernel-based approaches decompose graphs into "atomic subgraphs" to measure the similarities among graphs (Gärtner et al., 2003). Common types of atomic subgraphs consist of random walks (Gärtner et al., 2003; Kashima et al., 2003; Vishwanathan et al., 2010), shortest paths (Borgwardt and Kriegel, 2005), graphlets (Shervashidze et al., 2009), and Weisfeiler-Lehman subtree patterns (Shervashidze et al., 2011). Here we view each atomic substructure as a word term and each graph as a text document. Next, we represent a collection of graphs as a document-term matrix whose elements are the normalised frequency of terms in documents. We then derive a valid kernel, for example, a

linear one by taking inner product of every two documents, i.e., two graphs, and finally employ a kernel method such as Support Vector Machines (SVM) (Hofmann et al., 2008) to work on the graph classification problem.

In particular, graphlet kernel (Shervashidze et al., 2009) partitions graphs into graphlets (Pržulj, 2007) which are non-isomorphic induced subgraphs of size-k. The graphlet kernel between two graphs $\mathcal{G}$ and $\mathcal{G}'$ is defined as:

$$\mathcal{K}_{GK}\left(\mathcal{G}, \mathcal{G}\right) = \mathbf{f}_{\mathcal{G}}^{\mathsf{T}} \mathbf{f}_{\mathcal{G}'} \tag{2.1}$$

where $\mathbf{f}_{\mathcal{G}}$ (resp. $\mathbf{f}_{\mathcal{G}'}$) is the normalised vector of frequencies of graphlets occurring in $\mathcal{G}$ (resp. $\mathcal{G}'$).

Weisfeiler-Lehman (WL) kernel (Shervashidze et al., 2011) decomposes graphs into subtree patterns using a relabeling procedure. The main idea is to look at the label of each node and labels of its neighbours, sorted in a particular order, to form a new compressed label, and use that as the new label for such node at next iteration. This compressed label represents a subtree pattern rooted at such node. After repeatedly doing this process for several iterations, the WL kernel computes the frequencies of labels in both graphs across each iteration to construct the kernel matrix. This procedure is equivalent to counting the corresponding subtree patterns. The WL kernel between two labelled graphs $\mathcal{G}$ and $\mathcal{G}'$ is defined as:

$$\mathcal{K}_{WL}\left(\mathcal{G}, \mathcal{G}\right) = \mathbf{f}_{\mathcal{G}_{(0)}}^{\mathsf{T}} \mathbf{f}_{\mathcal{G}'_{(0)}} + \mathbf{f}_{\mathcal{G}_{(1)}}^{\mathsf{T}} \mathbf{f}_{\mathcal{G}'_{(1)}} + ... + \mathbf{f}_{\mathcal{G}_{(h)}}^{\mathsf{T}} \mathbf{f}_{\mathcal{G}'_{(h)}} \tag{2.2}$$

where h is the number of iterations, and $\mathbf{f}_{\mathcal{G}_{(i)}}$ (resp. $\mathbf{f}_{\mathcal{G}'_{(i)}}$) is the vector of frequencies of labels assigned to nodes in $\mathcal{G}$ (resp. $\mathcal{G}'$) at the i-th iteration.

Another method is to split graphs into shortest paths to form feature triples (Borgwardt and Kriegel, 2005). Let $(l(u); l(v); p_{(u,v)})$ denote a feature triple which represents the label $l(u)$ of the starting node u, the label $l(v)$ of the ending node v, and the shortest length $p_{(u,v)}$ between u and v. The shortest path kernel between two labelled graphs $\mathcal{G}$ and $\mathcal{G}'$ is defined as:

$$\mathcal{K}_{SP}\left(\mathcal{G}, \mathcal{G}\right) = \mathbf{f}_{\mathcal{G}}^{\mathsf{T}} \mathbf{f}_{\mathcal{G}'} \tag{2.3}$$

where $\mathbf{f}_{\mathcal{G}}$ (resp. $\mathbf{f}_{\mathcal{G}'}$) is the vector of frequencies of the triples $(l(u); l(v); p_{(u,v)})$ occurring in $\mathcal{G}$ (resp. $\mathcal{G}'$).

Lastly, random walk kernel (Gärtner et al., 2003; Kashima et al., 2003) decomposes graphs into random walks to produce label sequences and then calculates the frequencies of label sequences to construct the kernel matrix. It is worth to note that there are other graph kernel-based approaches, but we refer the readers to the surveys of graph kernels in (Nikolentzos et al., 2019; Kriege et al., 2019), rather than presenting here because they are outside of the scope of this thesis.

2.1.2 Word embedding-based models

There have been several ideas adopting the word embedding frameworks such as Word2Vec (Mikolov et al., 2013b) and Doc2Vec (Le and Mikolov, 2014) to learn the embeddings for nodes and graphs. In the following sections, we briefly describe both Word2Vec and Doc2Vec and then present word embedding-based models.

2.1.2.1 Word embeddings

Word embeddings are the distributed vector representations of words, where each word is mapped to a unique real-valued vector in such a way that similar words will have similar embedding vectors (Bengio et al., 2003; Collobert and Weston, 2008; Mikolov et al., 2013a,b; Pennington et al., 2014; Levy and Goldberg, 2014). Word embedding models have contributed numerous successfuly NLP applications such as sentiment analysis, question answering, topic models, machine translation, and sequence labeling (Socher et al., 2013b; Sutskever et al., 2014; Weston et al., 2015; Nguyen et al., 2015; Schnabel et al., 2015; Ma and Hovy, 2016).

Among those word embedding models, Word2Vec (Mikolov et al., 2013a,b) is the most well-known toolkit coming with two variants – Word2Vec Continuous Bag-of-Words (CBOW) and Word2Vec Skip-gram – as shown in Figure 2.1. Word2Vec CBOW aims to predict a target word from its context words, while Word2Vec Skip-gram uses a target

word to predict its context words.

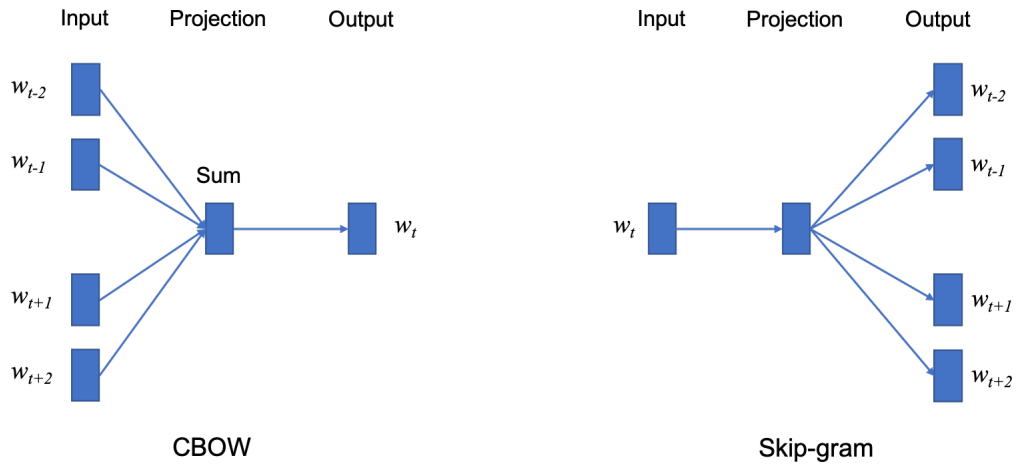

Figure 2.1: Illustration of Word2Vec. This figure is drawn based on (Mikolov et al., 2013a).

We denote $\boldsymbol{v}_w$ and $\tilde{\boldsymbol{v}}_w$ as the "target" and "context" vector representations of the word type w in the word vocabulary W, respectively. Given a word sequence $w_1, w_2, ..., w_M$, Word2Vec Skip-gram is to minimise the loss function as:

$$\mathcal{L}_{Skip-gram} = -\frac{1}{M}\sum_{t=1}^{M}\sum_{-k\leq j\leq k, j\neq 0}\log \mathrm{P}\left(w_{t+j}|w_t\right) \tag{2.4}$$

where k is the window size of a target word w_t, and w_{t+j} is one of the context words of w_t. In general, the probability $\mathrm{P}\left(c|w\right)$ of the context word c given the target word w in Equation 2.4 is computed using the softmax function as:

$$\mathrm{P}\left(c|w\right) = \frac{\exp(\tilde{\boldsymbol{v}}_c^{\mathsf{T}}\boldsymbol{v}_w)}{\sum_{c'\in V}\exp(\tilde{\boldsymbol{v}}_{c'}^{\mathsf{T}}\boldsymbol{v}_w)} \tag{2.5}$$

The cost of computing $\mathrm{P}\left(c|w\right)$ over every training instance is very expensive and impractical to large-scale datasets. One can resort to approximate Equation 2.5 using negative sampling (Mikolov et al., 2013b) as:

$$E = \log\sigma\left(\tilde{\boldsymbol{v}}_c^{\mathsf{T}}\boldsymbol{v}_w\right) + \sum_{i=1}^{K}\mathbb{E}_{c_i\sim \mathrm{P}_W}\left[\log\sigma\left(-\tilde{\boldsymbol{v}}_{c_i}^{\mathsf{T}}\boldsymbol{v}_w\right)\right] \tag{2.6}$$

where the sigmoid function $\sigma(u) = \frac{1}{1+e^{-u}}$, and $\{c_i\}_{i=1}^{K}$ are K randomly chosen context words which are sampled from an unigram distribution P_W raised to the 3/4 power.

The other model, Word2Vec CBOW, averages the vector representations of the context words into a vector $\tilde{\boldsymbol{v}}_{avg}$ to predict the target word. Word2Vec CBOW is to minimise the loss function as:

$$\begin{aligned}\mathcal{L}_{CBOW} &= -\frac{1}{M}\sum_{t=1}^{M}\log \mathrm{P}\left(w_t|w_{t-k},...,w_{t-1},w_{t+1},...,w_{t+k}\right)\\ &= -\frac{1}{M}\sum_{t=1}^{M}\log \mathrm{P}\left(w_t|avg\right)\\ &= -\frac{1}{M}\sum_{t=1}^{M}\log \frac{\exp(\boldsymbol{v}_{w_t}^{\mathsf{T}}\tilde{\boldsymbol{v}}_{avg})}{\sum_{w'\in V}\exp(\boldsymbol{v}_{w'}^{\mathsf{T}}\tilde{\boldsymbol{v}}_{avg})}\end{aligned} \tag{2.7}$$

Word2Vec models are then trained using stochastic gradient descent.

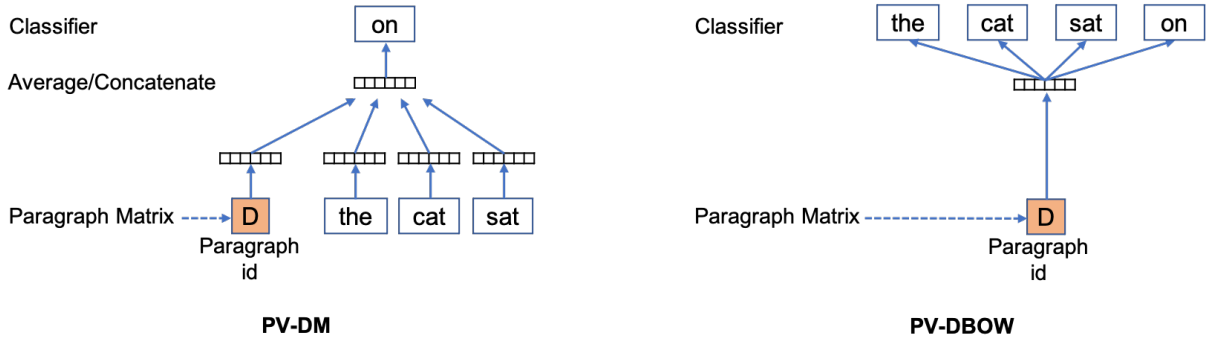

Figure 2.2: Illustration of Doc2Vec. This figure is drawn based on (Le and Mikolov, 2014).

It is worth to learn meaningful embeddings for variable-length pieces of texts such as sentences, paragraphs, and documents in some cases. A common way is taking a weighted average of word embeddings learned by Word2Vec in the text, but it loses information about word order. Doc2Vec (Le and Mikolov, 2014) is proposed to address such problem and learn a unique embedding vector for each document. Figure 2.2 illustrates two Doc2Vec variants, including Distributed Memory Model of Paragraph Vectors (PV-DM) and Distributed Bag of Words version of Paragraph Vector (PV-DBOW). PV-DV can either average or concatenate the document embedding and the word embeddings to predict the next word

in a context window, while PV-DBOW aims to use the document embedding to predict the words within a sampled window.

2.1.2.2 Learning node representations

DeepWalk (Perozzi et al., 2014) aims to generate random walks from a given graph by starting with a random node (uniformly sampled), and then repeat sampling (also uniformly) the next node from the neighbours of the last visited node in the walk until reaching the maximum length of the walk. In particular, let v_i denote the i-th node in the walk and v_0 is the root node, then v_i is sampled following the distribution:

$$\mathrm{P}(\mathsf{v}_i|\mathsf{v}_{i-1}) = \begin{cases} \frac{1}{\mathsf{Z}} \text{ if } (\mathsf{v}_{i-1}, \mathsf{v}_i) \in \mathcal{E} \\ 0 \text{ otherwise} \end{cases} \tag{2.8}$$

where Z is the constant and $\mathcal{E}$ denotes a set of edges. After that, each random walk can be seen as a sequence of (word) nodes, hence we then uses Word2Vec to learn node embeddings.

Node2Vec (Grover and Leskovec, 2016) utilises a biased random walk strategy of using interpolation between two sampling fashions: Breadth-First Sampling (BFS) and Depth-First Sampling (DFS), hence capturing the diversity of connectivity patterns observed in networks. Node2Vec computes the edge weights to form a 2^{nd} order random walk with two parameters p and q when sampling the next nodes as:

$$\pi_{\mathsf{v}_i,\mathsf{v}_{i+1}} = \alpha_{pq}(\mathsf{v}_{i-1}, \mathsf{v}_{i+1}) \cdot \tau_{\mathsf{v}_i,\mathsf{v}_{i+1}} \tag{2.9}$$

where $\mathsf{v}_{i+1} \in \mathcal{N}_{\mathsf{v}_i}$, $\tau_{\mathsf{v}_i,\mathsf{v}_{i+1}}$ is the static edge weight between nodes v_i and v_{i+1}, and:

$$\alpha_{pq}(\mathsf{v}_{i-1}, \mathsf{v}_{i+1}) = \begin{cases} \frac{1}{p} \text{ if } d_{\mathsf{v}_{i-1},\mathsf{v}_{i+1}} = 0 \\ 1 \text{ if } d_{\mathsf{v}_{i-1},\mathsf{v}_{i+1}} = 1 \\ \frac{1}{q} \text{ if } d_{\mathsf{v}_{i-1},\mathsf{v}_{i+1}} = 2 \end{cases} \tag{2.10}$$

where $d_{\mathsf{v}_{i-1},\mathsf{v}_{i+1}}$, valued in $\{0, 1, 2\}$, is the shortest path distance between v_{i-1} and v_{i+1}. Parameter p controls an immediate re-visitation to a node in the walk. Setting $p > \mathsf{max}(q, 1)$ is less likely to come back an already-visited node in the following two steps, while setting $p < \mathsf{min}(q, 1)$ is to lead the random walk to backtrack one step. Parameter q helps to explore neighbours in a BFS as well as DFS. If $q > 1$, the random walk is biased towards nodes which are close to the $(i-1)$-th node w.r.t BFS, while if $q < 1$, the walk is biased towards nodes which are far away from the $(i-1)$-th node w.r.t DFS. As a result, v_{i+1} is sampled following the distribution:

$$\mathrm{P}(\mathsf{v}_{i+1}|\mathsf{v}_i) = \begin{cases} \frac{\pi_{\mathsf{v}_i,\mathsf{v}_{i+1}}}{\mathsf{Z}} \text{ if } (\mathsf{v}_i, \mathsf{v}_{i+1}) \in \mathcal{E} \\ 0 \text{ otherwise} \end{cases} \tag{2.11}$$

LINE (Tang et al., 2015) can be seen as an extension of Word2Vec to induce edge weights between two nodes. The proposed model aims to minimise a loss function which extends Equation 2.4 as:

$$\mathcal{L}_{LINE} = -\sum_{\mathsf{v}\in\mathcal{V}} \sum_{\mathsf{v}'\in\mathcal{N}_\mathsf{v}} \tau_{\mathsf{v},\mathsf{v}'} \log \mathrm{P}\left(\mathsf{v}'|\mathsf{v}\right) \tag{2.12}$$

where $\mathcal{V}$ denotes a set of nodes; $\mathcal{N}_\mathsf{v}$ is the set of neighbours of v; and $\tau_{\mathsf{v},\mathsf{v}'}$ is the edge weight between v and v' and can be pre-defined through algorithms such as PageRank (Page et al., 1999).

DDRW (Li et al., 2016a) is a semi-supervised model using node labels during jointly training DeepWalk (Perozzi et al., 2014) with L2-regularised L2-loss Support Vector Classification (L2-SVC) (Fan et al., 2008). DDRW simultaneously minimises both the loss functions of DeepWalk and L2-SVC as:

$$\mathcal{L}_{DDRW} = \eta\mathcal{L}_{DeepWalk} + \mathcal{L}_{L2-SVC} \tag{2.13}$$

where η is a balancing parameter, $\mathcal{L}_{DeepWalk}$ is taken from the loss function of Word2Vec

(Mikolov et al., 2013a,b), and $\mathcal{L}_{L2-SVC}$ is defined as:

$$\mathcal{L}_{L2-SVC} = \frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} + C\sum_{i=1}^{|\mathcal{V}|}\left(\max\left(0, 1 - \mathsf{y_i}\mathbf{w}^{\mathsf{T}}\boldsymbol{v}_{\mathsf{v}_i}\right)\right)^2 \tag{2.14}$$

where $C > 0$ is a regularisation parameter, $\mathbf{w}$ is a weight vector to adjust the parameter C, $\mathsf{y_i}$ is the node label of node v_i, and $\boldsymbol{v}_{\mathsf{v}_i}$ denotes the node embedding of v_i.

2.1.2.3 Learning graph representations

Deep graph kernel (Yanardag and Vishwanathan, 2015) applies Word2Vec to learn the embeddings for atomic substructures such as the graphlets, the Weisfeiler-Lehman subtree patterns, and the shortest paths, and then derives the kernel between two graphs $\mathcal{G}$ and $\mathcal{G}'$ as:

$$\mathcal{K}_{SP}\left(\mathcal{G}, \mathcal{G}'\right) = \mathbf{f}_{\mathcal{G}}^{\mathsf{T}}\mathcal{M}\mathbf{f}_{\mathcal{G}'} \tag{2.15}$$

where $\mathbf{f}_{\mathcal{G}}$ (resp. $\mathbf{f}_{\mathcal{G}'}$) is the vector of frequencies of the atomic substructures occurring in $\mathcal{G}$ (resp. $\mathcal{G}'$), and $\mathcal{M}$ is the diagonal matrix where $\mathcal{M}_{ii}$ is computed as $\boldsymbol{v}_{s_i}^{\mathsf{T}}\boldsymbol{v}_{s_i}$ for the i-th atomic substructure s_i with its own embedding $\boldsymbol{v}_{s_i}$ produced by Word2Vec.

Anonymous walk embedding (Ivanov and Burnaev, 2018) maps each random walk into an "anonymous walk" where each state is recorded by its first occurrence index in the random walk, then views each anonymous walk as a word token, and utilises Doc2Vec to achieve the graph embeddings to compute the graph similarities to construct the kernel matrix. Graph2Vec (Narayanan et al., 2017) employs Doc2Vec on the Weisfeiler-Lehman subtree patterns to obtain the graph embeddings, which are then fed to SVM classifier training.

2.1.3 Graph neural networks

This section is used to summarise common GNN methods, wherein the material is partially based on (Wu et al., 2019b).

We represent each graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \{\mathbf{h}_{\mathsf{v}}\}_{\mathsf{v}\in\mathcal{V}})$, where $\mathcal{V}$ is a set of nodes, $\mathcal{E}$ is a set of

edges, and $\mathbf{h}_{\mathsf{v}}$ (i.e., $\mathbf{h}_{\mathsf{v}}^{(0)}$) represents the feature vector of node $\mathsf{v} \in \mathcal{V}$.

Node classification. We consider 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 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 $\{\mathsf{y}_m\}_{m=1}^{M} \subseteq \mathcal{Y}$, 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.

In general, GNNs use an aggregation function, which aims to update the vector representation of each node by recursively propagating the vector representations of its neighbours (Scarselli et al., 2009; Kipf and Welling, 2017; Hamilton et al., 2017a; Veličković et al., 2018). GNNs then utilise a readout pooling function to obtain the graph embeddings, which are fed to multiple fully-connected layers followed by a $\mathsf{softmax}$ layer to predict the graph labels. Mathematically, given a graph $\mathcal{G}$, we formulate GNNs as follows:

$$\mathbf{h}_{\mathsf{v}}^{(k+1)} = \textsc{Aggregation}\left(\left\{\mathbf{h}_{\mathsf{u}}^{(k)}\right\}_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}\cup\{\mathsf{v}\}}\right) \tag{2.16}$$

$$\mathbf{e}_{\mathcal{G}} = \textsc{ReadOut}\left(\left\{\left\{\mathbf{h}_{\mathsf{v}}^{(k)}\right\}_{k=0}^{K}\right\}_{\mathsf{v}\in\mathcal{V}}\right) \tag{2.17}$$

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

2.1.3.1 Aggregation functions

There have been many designs for the aggregation functions proposed in recent literature. In this section, we briefly summarise several common approaches to construct the aggregation functions.

Graph Convolutional Network (GCN) (Kipf and Welling, 2017) updates vector representation for a given node $\mathsf{v} \in \mathcal{V}$ from its neighbours, using multiple layers stacked on top

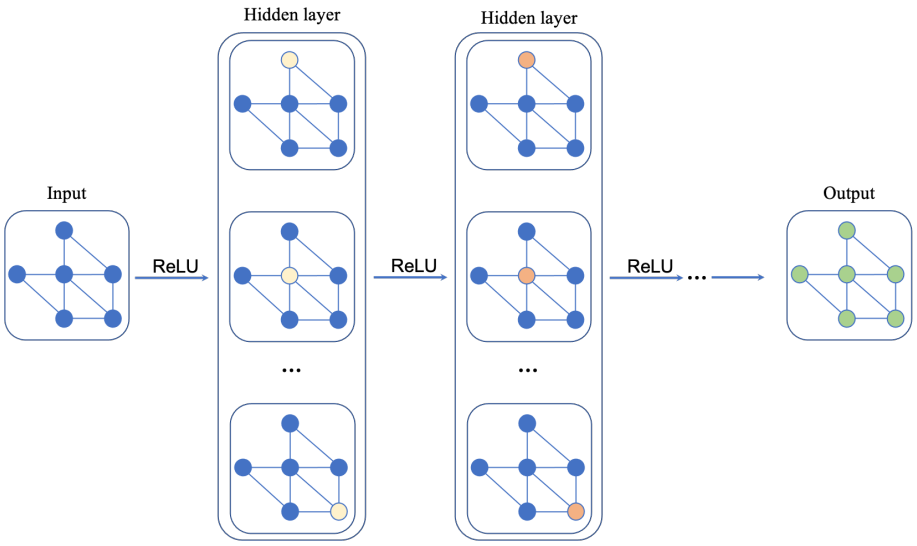

Figure 2.3: Illustration of GCN. This figure is drawn based on `https://tkipf.github.io/graph-convolutional-networks/`.

of each other as illustrated in Figure 2.3 as:

$$\mathbf{H}^{(k+1)} = \mathsf{g}\left(\mathbf{A}\mathbf{H}^{(k)}\mathbf{W}^{(k)}\right) \tag{2.18}$$

or in another form of:

$$\mathbf{h}_{\mathsf{v}}^{(k+1)} = \mathsf{g}\left(\sum_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}\cup\{\mathsf{v}\}} a_{\mathsf{v},\mathsf{u}}\mathbf{W}^{(k)}\mathbf{h}_{\mathsf{u}}^{(k)}\right), \forall \mathsf{v} \in \mathcal{V} \tag{2.19}$$

where k is the layer index; $a_{\mathsf{v},\mathsf{u}}$ is an edge constant between nodes v and u in the re-normalised 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 $\tilde{\mathbf{A}}$; $\mathbf{W}^{(k)}$ is a weight matrix; $\mathbf{h}_{\mathsf{u}}^{(0)}$ is a feature vector of node u; g is a nonlinear activation function such as ReLU; and $\mathcal{N}_{\mathsf{v}}$ is the set of neighbours of node v. After that, to perform the semi-supervised node classification task, at the last layer, g is replaced by a $\mathsf{softmax}$ layer to predict the node labels. The model parameters are then learned by minimizing the cross-entropy loss function.

Graph Attention Network (Veličković et al., 2018) extends GCN to compute edge

weights following the standard attention technique (Bahdanau et al., 2015) as:

$$\mathbf{h}_{\mathsf{v}}^{(k+1)} = \mathsf{g}\left(\sum_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}\cup\{\mathsf{v}\}} \tau_{\mathsf{v},\mathsf{u}}^{(k)}\mathbf{W}^{(k)}\mathbf{h}_{\mathsf{u}}^{(k)}\right), \forall \mathsf{v} \in \mathcal{V} \tag{2.20}$$

where g is the $\mathsf{sigmoid}$ function $\sigma(x) = \frac{1}{1+e^{-x}}$; and $\tau_{\mathsf{v},\mathsf{u}}^{(k)}$ is an edge weight between nodes v and u, which is computed as:

$$\tau_{\mathsf{v},\mathsf{u}}^{(k)} = \mathsf{softmax}\left(\mathsf{LeakyReLU}\left(\boldsymbol{a}^{(k)\mathsf{T}}\left[\mathbf{W}^{(k)}\mathbf{h}_{\mathsf{v}}^{(k)}; \mathbf{W}^{(k)}\mathbf{h}_{\mathsf{u}}^{(k)}\right]\right)\right) \tag{2.21}$$

where [;] denotes a vector concatenation. Besides, GAT 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.

While 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) uses a more powerful aggregation function based on a multi-layer perceptron (MLP) network of two fully-connected layers as:

$$\mathbf{h}_{\mathsf{v}}^{(k+1)} = \mathsf{MLP}^{(\mathsf{k})}\left(\sum_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}\cup\{\mathsf{v}\}} \mathbf{h}_{\mathsf{u}}^{(k)}\right), \forall \mathsf{v} \in \mathcal{V} \tag{2.22}$$

Node sampling procedures. Regarding a faster learning, GraphSAGE (Hamilton et al., 2017a) extends GCN to use a node-wise procedure of uniformly sampling a fixed number of neighbours for each node at each layer as illustrated in Figure 2.4 as:

$$\mathbf{h}_{\mathsf{v}}^{(k+1)} = \mathsf{g}\left(\mathbf{W}^{(k)}\left[\mathbf{h}_{\mathsf{v}}^{(k)}; \mathbf{h}_{\mathcal{N}_{\mathsf{v}}'}^{(k)}\right]\right), \forall \mathsf{v} \in \mathcal{V} \tag{2.23}$$

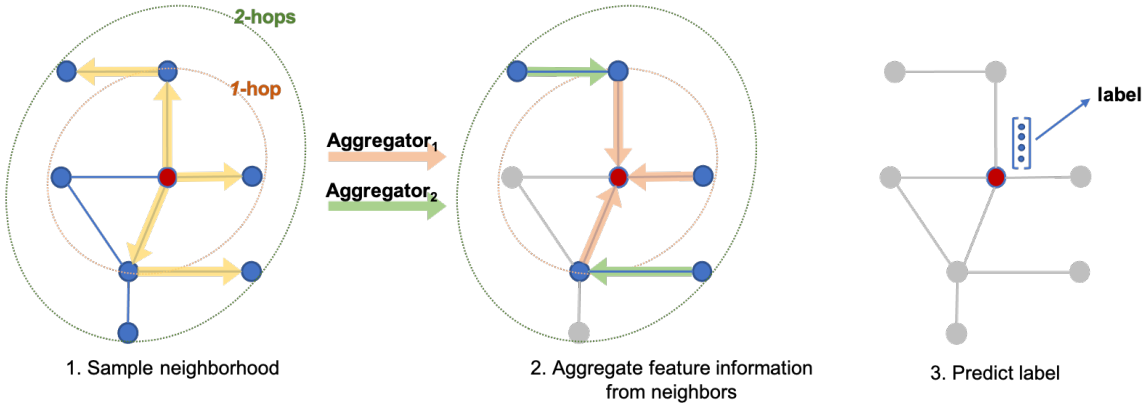

Figure 2.4: Illustration of GraphSAGE. This figure is drawn based on (Hamilton et al., 2017a).

where [;] denotes a vector concatenation, and $\mathbf{h}_{\mathcal{N}'_v}^{(k)}$ can be obtained using an element-wise max-pooling operation as:

$$\mathbf{h}_{\mathcal{N}'_v}^{(k)} = \max\left(\left\{g\left(\mathbf{W}_{pool}\mathbf{h}_u^{(k)} + \mathbf{b}\right)\right\}_{u\in\mathcal{N}'_v}\right) \tag{2.24}$$

where $\mathcal{N}'_v$ is defined as a fixed-size, uniformly sampled from $\mathcal{N}_v$ of v. Besides, $\mathcal{N}'_v$ is sampled differently through each layer.

Moreover, FastGCN (Chen et al., 2018b) utilises a layer-wise procedure of sampling $\mathcal{V}'$ (i.e., a fixed number of nodes) from $\mathcal{V}$ independently for each GCN layer as:

$$\mathbf{h}_v^{(k+1)} = g\left(\frac{|\mathcal{V}|}{|\mathcal{V}'^{(k)}|}\sum_{u\in\mathcal{N}_v\cup\{v\}} a_{v,u}\mathbf{W}^{(k)}\mathbf{h}_u^{(k)}\right), \forall v \in \mathcal{V}'^{(k)} \tag{2.25}$$

where $\mathcal{V}'^{(k)}$ is uniformly sampled from $\mathcal{V}$, or:

$$\mathbf{h}_v^{(k+1)} = g\left(\frac{1}{|\mathcal{V}'^{(k)}|}\sum_{u\in\mathcal{N}_v\cup\{v\}} \frac{a_{v,u}}{q_u}\mathbf{W}^{(k)}\mathbf{h}_u^{(k)}\right), \forall v \in \mathcal{V}'^{(k)} \tag{2.26}$$

where $\mathcal{V}'^{(k)}$ is sampled according to a probability distribution q for all the nodes in $\mathcal{V}$. In another work, Adapt (Huang et al., 2018) introduces an adaptive layer-wise procedure of

sampling nodes in the lower layers conditioned on the higher layers, and further adds a skip connection between the 2-hops layers to maintain the information over distant nodes.

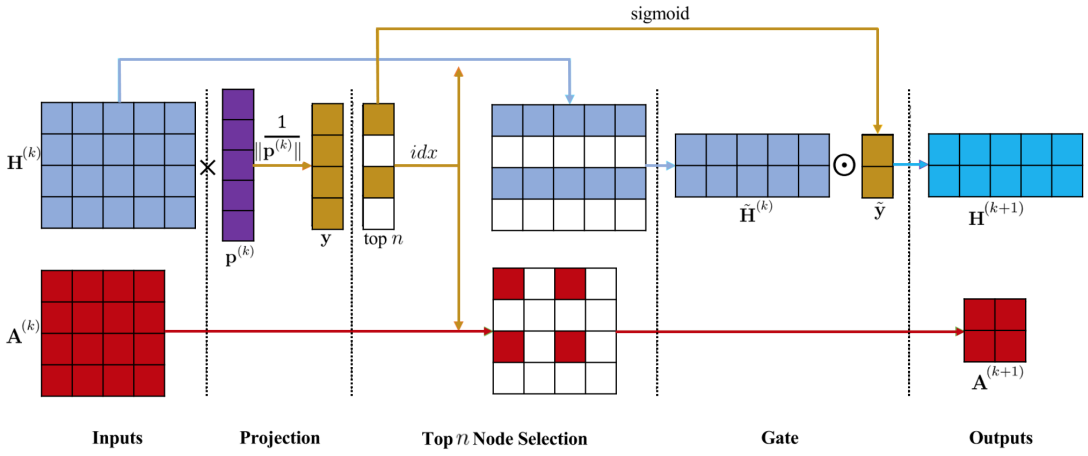

Figure 2.5: Illustration of Graph U-Net's node sampling procedure (gPool). This figure is drawn based on (Gao and Ji, 2019).

Furthermore, Graph U-Net (Gao and Ji, 2019) presents an advanced layer-wise sampling procedure, named gPool, to retain a portion of the input nodes as illustrated in Figure 2.5 as:

$$\begin{aligned}
\mathbf{y} &= \mathbf{H}^{(k)}\mathbf{p}^{(k)}/\|\mathbf{p}^{(k)}\| \\
idx &= \mathsf{rank}\left(\mathbf{k}, n\right) \\
\tilde{\mathbf{y}} &= \sigma\left(\mathbf{y}[idx]\right) \\
\tilde{\mathbf{H}}^{(k)} &= \mathbf{H}^{(k)}[idx, :] \\
\mathbf{H}^{(k+1)} &= \tilde{\mathbf{H}}^{(k)} \odot \tilde{\mathbf{y}} \\
\mathbf{A}^{(k+1)} &= \left(\mathbf{A}^{(k)}\mathbf{A}^{(k)}\right)[idx, idx]
\end{aligned} \tag{2.27}$$

where n is the number of retained nodes; idx is a list of indices of top n-largest values for n retained nodes; the sigmoid function $\sigma\left(x\right) = \frac{1}{1+e^{-x}}$; $\odot$ denotes the element-wise multiplication where the j-th row of $\mathbf{H}^{(k+1)}$ is the product of the j-th row of $\tilde{\mathbf{H}}^{(k)}$ and the j-th scalar value of $\tilde{\mathbf{y}}$; and $\mathbf{A}^{(k)}\mathbf{A}^{(k)}$ results in the 2^{nd} graph power to increase the graph connectivity. The output adjacency matrix $\mathbf{A}^{(k+1)}$ and the output matrix $\mathbf{H}^{(k+1)}$ of node

representations are new inputs for next GCN layers. Graph U-Net then utilises an unpooling procedure (gUnpool) followed by GCN layers to produce final node representations, as illustrated in Figure 2.6. Note that the gUnpool is used to reconstruct the original graph structure by fulfilling empty vectors for unselected nodes.

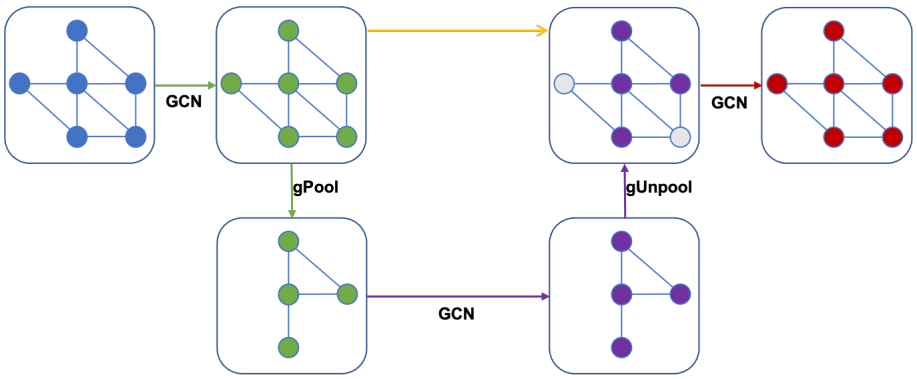

Figure 2.6: Illustration of Graph U-Net. This figure is drawn based on (Gao and Ji, 2019).

Other approaches. Jumping Knowledge Network (Xu et al., 2018) considers several ways to construct the final representation of node v over the different representations at the different layers as:

$$\mathbf{e}_{\mathsf{v}} = \phi\left(\left\{\mathbf{h}_{\mathsf{v}}^{(k)}\right\}_{k=1}^{K}\right) \tag{2.28}$$

where $\phi(.)$ can be a vector concatenation, a element-wise max-pooling operation, or a bi-directional LSTM (Hochreiter and Schmidhuber, 1997).

Deep Graph Infomax (Velickovic et al., 2019) leverages Deep InfoMax (Hjelm et al., 2018) to learn node representations by maximizing mutual information between patch representations and corresponding high-level summaries of graphs. Graph Transformer Network (Yun et al., 2019) identifies useful meta-paths (Wang et al., 2019b) to transform graph structures and applies GCN (Kipf and Welling, 2017) to learn the node embeddings for the node classification task on heterogeneous graphs.

The Graph Neural Network (Scarselli et al., 2009) updates the vector representations of nodes by recursively propagating their neighbours' vector representations, using a recurrent function until convergence, wherein the hidden states and the outputs of node v are

computed as:

$$\mathbf{h}_{\mathsf{v}}^{(t+1)} = \mathsf{f}\left(\boldsymbol{l}_{\mathsf{v}}, \{\boldsymbol{l}_{\mathsf{u}}\}_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}}, \{\boldsymbol{l}_{\mathsf{v},\mathsf{u}}\}_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}}, \{\mathbf{h}_{\mathsf{u}}^{(t)}\}_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}}\right) \;\; ; \quad \boldsymbol{o}_{\mathsf{v}}^{(t)} = \mathsf{g}\left(\boldsymbol{l}_{\mathsf{v}}, \mathbf{h}_{\mathsf{v}}^{(t)}\right) \tag{2.29}$$

where $\boldsymbol{l}_{\mathsf{v}}$ is the node label of v; $\boldsymbol{l}_{\mathsf{v},\mathsf{u}}$ is the edge label between v and u; f is a parametric function called "local transition function"; and g is the local output function.

Gated Graph Neural Network (GGNN) (Li et al., 2016b) adopts Gated Recurrent Units (GRUs) (Cho et al., 2014), unrolls the recurrence for a fixed number T of timesteps, and removes the need to constrain parameters to ensure convergence. The recurrent function in GGNN is given as:

$$\begin{aligned}
\mathbf{h}_{\mathsf{v}}^{(1)} &= \left[\boldsymbol{x}_{\mathsf{v}}^{\mathsf{T}}, \mathbf{0}\right]^{\mathsf{T}} \\
\mathbf{a}_{\mathsf{v}}^{(t+1)} &= \mathbf{A}_{\mathsf{v}:}^{\mathsf{T}}\left[\mathbf{h}_1^{(t)\mathsf{T}} ... \mathbf{h}_{|\mathcal{V}|}^{(t)\mathsf{T}}\right]^{\mathsf{T}} + \mathbf{b} \\
\mathbf{z}_{\mathsf{v}}^{(t+1)} &= \sigma\left(\mathbf{W}^{z}\mathbf{a}_{\mathsf{v}}^{(t+1)} + \mathbf{U}^{z}\mathbf{h}_{\mathsf{v}}^{(t)}\right) \\
\mathbf{r}_{\mathsf{v}}^{(t+1)} &= \sigma\left(\mathbf{W}^{r}\mathbf{a}_{\mathsf{v}}^{(t+1)} + \mathbf{U}^{r}\mathbf{h}_{\mathsf{v}}^{(t)}\right) \\
\widetilde{\mathbf{h}_{\mathsf{v}}^{(t+1)}} &= \tanh\left(\mathbf{W}\mathbf{a}_{\mathsf{v}}^{(t+1)} + \mathbf{U}\left(\mathbf{r}_{\mathsf{v}}^{(t+1)} \odot \mathbf{h}_{\mathsf{v}}^{(t)}\right)\right) \\
\mathbf{h}_{\mathsf{v}}^{(t+1)} &= \left(1 - \mathbf{z}_{\mathsf{v}}^{(t+1)}\right) \odot \mathbf{h}_{\mathsf{v}}^{(t)} + \mathbf{z}_{\mathsf{v}}^{(t+1)} \odot \widetilde{\mathbf{h}_{\mathsf{v}}^{(t+1)}}
\end{aligned} \tag{2.30}$$

where $\boldsymbol{x}_{\mathsf{v}}$ denotes the label annotations of node v.

Message Passing Neural Network (MPNN) (Gilmer et al., 2017) updates the hidden states $\mathbf{h}_{\mathsf{v}}^{(t+1)}$ for node v based on messages $\mathbf{m}_{\mathsf{v}}^{(t+1)}$ as:

$$\begin{aligned}
\mathbf{m}_{\mathsf{v}}^{(t+1)} &= \sum_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}} M_t\left(\mathbf{h}_{\mathsf{v}}^{(t)}, \mathbf{h}_{\mathsf{u}}^{(t)}, \mathbf{e}_{\mathsf{v},\mathsf{u}}\right) \\
\mathbf{h}_{\mathsf{v}}^{(t+1)} &= U_t\left(\mathbf{h}_{\mathsf{v}}^{(t)}, \mathbf{m}_{\mathsf{v}}^{(t+1)}\right)
\end{aligned} \tag{2.31}$$

where the message functions M_t, node update functions U_t are learnable differentiable functions. MPNN can generalise several existing networks such as Gated Graph Neural Network (Li et al., 2016b), Deep Tensor Neural Network (Schütt et al., 2017), and GCN

(Kipf and Welling, 2017).

Structural Deep Network Embedding (SDNE) (Wang et al., 2016) is proposed to preserve both local and global graph structures. SDNE constructs an unsupervised component using the auto-encoder architecture to preserve the global structure as:

$$\mathcal{L}_{\mathsf{unsup}} = \sum_{i=1}^{|\mathcal{V}|} \| \left(\hat{\boldsymbol{x}}_{\mathsf{v}} - \boldsymbol{x}_{\mathsf{v}}\right) \odot \boldsymbol{b}_i \|_2^2 \tag{2.32}$$

where $\hat{\boldsymbol{x}}_{\mathsf{v}}$ is the reconstructed vector representation of node v; $\odot$ is the Hadamard product; and $\boldsymbol{b}_i$ is a learnable weight vector, which is used to impose more penalty to the reconstruction error of the non-zero elements. SDNE uses a supervised component to constrain the similarities between node representations to capture the local structure as:

$$\mathcal{L}_{\mathsf{sup}} = \sum_{(\mathsf{v},\mathsf{u})} x_{ij} \| \boldsymbol{y}_{\mathsf{v}}^{(K)} - \boldsymbol{y}_{\mathsf{u}}^{(K)} \|_2^2 \tag{2.33}$$

SDNE jointly optimise both $\mathcal{L}_{\mathsf{unsup}}$ and $\mathcal{L}_{\mathsf{sup}}$ with L_2 regularisation $\mathcal{L}_{reg}$ into the following loss function:

$$\mathcal{L}_{\mathsf{SDNE}} = \mathcal{L}_{\mathsf{unsup}} + \alpha \mathcal{L}_{\mathsf{sup}} + \nu \mathcal{L}_{reg} \tag{2.34}$$

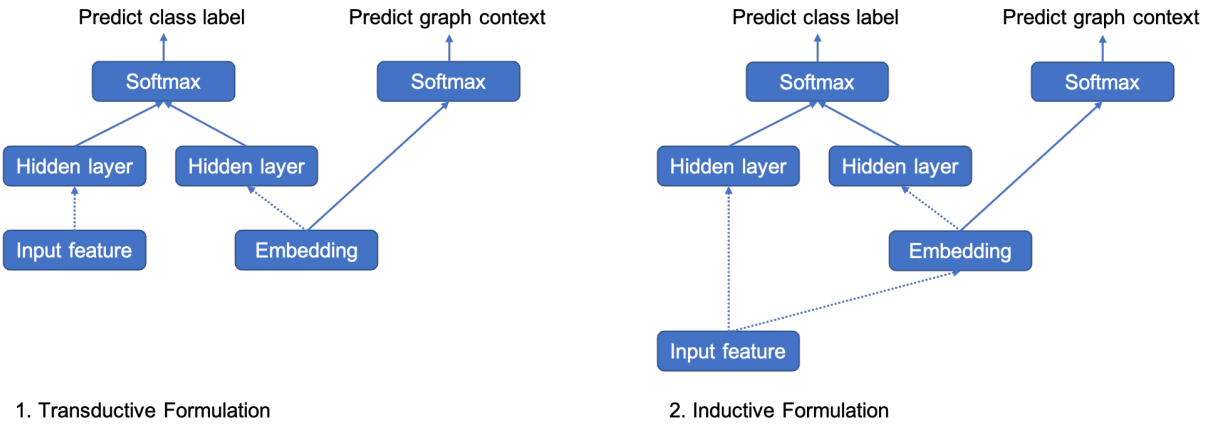

Figure 2.7: Illustration of Planetoid. This figure is drawn based on (Yang et al., 2016). Each dotted arrow represents a feed-forward network. Solid arrows denote direct connections.

Planetoid (Yang et al., 2016) introduces two training procedures including a transduc-

tive setting and an inductive setting, as shown in Figure 2.7. The transductive setting is used in most of the existing GNN approaches, where all nodes are present during training. By contrast, a more important setup, but less mentioned, is the inductive setting, wherein only a part of the input graph is used to train the model, i.e., a certain percentage of the nodes are removed and become unseen during training, hence they are new nodes during testing. The trained model is used to infer embeddings for these nodes. Planetoid defines the loss function for the transductive setting as:

$$\mathcal{L}_{Trans} = -\frac{1}{|\mathcal{V}|}\sum_{\mathsf{v}\in\mathcal{V}} \log \mathrm{P}\left(y_{\mathsf{v}}|\mathbf{h}_{\mathsf{v}}, \boldsymbol{v}_{\mathsf{v}}\right) - \lambda \mathbb{E}_{(\mathsf{v},\mathsf{v}',\gamma)} \log \sigma\left(\gamma \tilde{\boldsymbol{v}}_{\mathsf{v}'}^{\mathsf{T}} \boldsymbol{v}_{\mathsf{v}}\right) \tag{2.35}$$

And the loss function for the inductive setting is defined as:

$$\mathcal{L}_{Ind} = -\frac{1}{|\mathcal{V}|}\sum_{\mathsf{v}\in\mathcal{V}} \log \mathrm{P}\left(y_{\mathsf{v}}|\mathbf{h}_{\mathsf{v}}\right) - \lambda \mathbb{E}_{(\mathsf{v},\mathsf{v}',\gamma)} \log \sigma\left(\gamma \tilde{\boldsymbol{v}}_{\mathsf{v}'}^{\mathsf{T}} \mathsf{g}\left(\mathbf{W}\mathbf{h}_{\mathsf{v}} + \mathbf{b}\right)\right) \tag{2.36}$$

where $\mathrm{P}\left(y_{\mathsf{v}}|\mathbf{h}_{\mathsf{v}}, \boldsymbol{v}_{\mathsf{v}}\right) = \mathsf{softmax}\left(\mathbf{w}_{y_{\mathsf{v}}}^{\mathsf{T}}\left[\mathbf{h}_{\mathsf{v}}^{(M)}; \boldsymbol{v}_{\mathsf{v}}^{(N)}\right]\right)$; y_{v} is the label of v; λ is a constant weight; v' is a context node uniformly sampled for target node v; $\boldsymbol{v}_{\mathsf{v}}$ is the embedding of node v, and $\tilde{\boldsymbol{v}}_{\mathsf{v}'}$ is the context embedding of node v'; $\gamma = +1$ denotes $(\mathsf{v}, \mathsf{v}')$ is a positive pair, while $\gamma = -1$ denotes $(\mathsf{v}, \mathsf{v}')$ is a negative pair; g is a non-linear function (such as, ReLU); and after training, $\mathsf{g}\left(\mathbf{W}\mathbf{h}_{\mathsf{v}} + \mathbf{b}\right)$ is used to produce the embeddings for new nodes v in the inductive setting.

2.1.3.2 Graph-level readout poolings

In this section, we briefly describe several common graph-level poolings to obtain the graph embeddings.

Graph Isomorphism Network (GIN-0) (Xu et al., 2019), as mentioned in Equation 2.22, follows Jumping Knowledge Network (Xu et al., 2018) to employ a concatenation over the different representations at the different layers to obtain the final representation $\mathbf{e}_{\mathsf{v}}$ for node v as:

$$\mathbf{e}_{\mathsf{v}} = \left[\mathbf{h}_{\mathsf{v}}^{(0)}; \mathbf{h}_{\mathsf{v}}^{(1)}; \mathbf{h}_{\mathsf{v}}^{(2)}; ...; \mathbf{h}_{\mathsf{v}}^{(K)}\right], \forall \mathsf{v} \in \mathcal{V} \tag{2.37}$$

where K is the index of the last layer.[1] The graph-level readout function can be a simple sum pooling as GIN-0 leverages the sum pooling to obtain competitive accuracies. Thus, we can utilise the sum pooling to obtain the embedding $\mathbf{e}_{\mathcal{G}}$ of the entire graph $\mathcal{G}$ as:

$$\mathbf{e}_{\mathcal{G}} = \sum_{\mathsf{v} \in \mathcal{V}} \mathbf{e}_{\mathsf{v}} = \sum_{\mathsf{v} \in \mathcal{V}} \left[\mathbf{h}_{\mathsf{v}}^{(1)}; \mathbf{h}_{\mathsf{v}}^{(2)}; ...; \mathbf{h}_{\mathsf{v}}^{(K)} \right] \tag{2.38}$$

Deep Graph Convolutional Neural Network (DGCNN) (Zhang et al., 2018a) applies GCN (Kipf and Welling, 2017) to horizontally concatenate the vector representations of each node at the different layers to output a tensor $\mathbf{H}^{(1):(K)}$, which is then fed to a SortPooling layer to produce sorted node representations. These sorted representations are passed to a convolutional layer (LeCun et al., 1998) followed by a fully-connected layer then with a $\mathsf{softmax}$ layer to predict the graph label. In particular, SortPooling uses a row-wise sorting on $\mathbf{H}^{(K)}$ in a descending order to sort nodes. If two nodes have the same value, SortPooling continues comparing these nodes based on $\mathbf{H}^{(K-1)}$, $\mathbf{H}^{(K-2)}$, and so on until broken. After sorting, SortPooling utilises an unified function to unify graphs with different number of nodes to the same size for the convolutional layer.

Hierarchical pooling (HgPool) (Cangea et al., 2018) leverages the Graph U-Net's node sampling gPool (Gao and Ji, 2019) (as mentioned in Equation 2.28) to retain a portion of input nodes with the output matrix $\mathbf{H}^{(k)}$ at the k-th layer. HgPool then produces the graph embedding $\mathbf{e}_{\mathcal{G}}$ as:

$$\mathbf{e}_{\mathcal{G}} = \sum_{k=1}^{K} \mathbf{e}_{\mathcal{G}}^{(k)} = \sum_{k=1}^{K} \left[\frac{1}{|\mathcal{V}^{(k)}|} \sum_{\mathsf{v} \in \mathcal{V}^{(k)}} \mathbf{h}_{\mathsf{v}}^{(k)}; \mathsf{max} \left(\left\{ \mathbf{h}_{\mathsf{v}}^{(k)} \right\}_{\mathsf{v} \in \mathcal{V}^{(k)}} \right) \right] \tag{2.39}$$

where [;] denotes a vector concatenation; $\mathcal{V}^{(k)}$ is the set of remaining nodes at the k-th layer; and max denotes the element-wise max-pooling operation.

As shown in Figure 2.8, Self-Attention Graph Pooling (Lee et al., 2019) re-implements the general architectures from DGCNN (Zhang et al., 2018a) and HgPool (Cangea et al., 2018) with using the Graph U-Net's node sampling gPool (Gao and Ji, 2019) as the

[1] It is optional to include $\mathbf{h}_{\mathsf{v}}$ (i.e., $\mathbf{h}_{\mathsf{v}}^{(0)}$) into $\mathbf{e}_{\mathsf{v}}$.

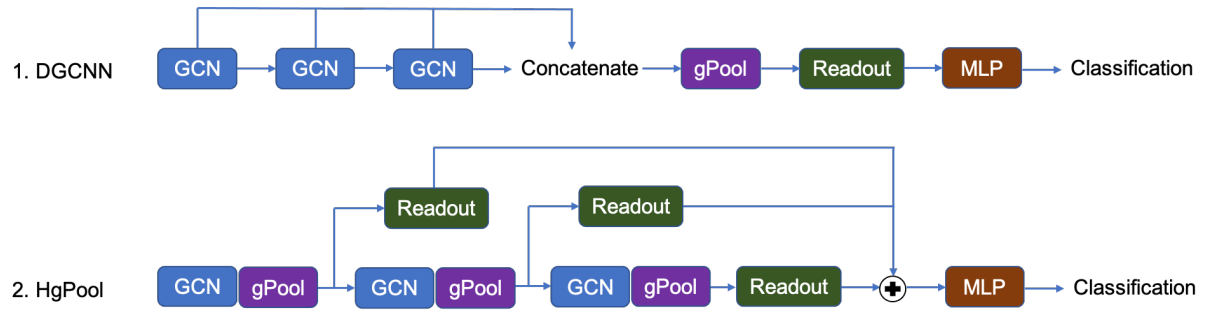

Figure 2.8: Illustration of model architectures re-implemented in SAGPool. This figure is drawn based on (Lee et al., 2019).

graph-level pooling layer.

Differentiable Pooling (DIFFPOOL) (Ying et al., 2018b) utilises a differentiable cluster assignment to map nodes at the k-th layer to a set of coarsened clusters at the $(k+1)$-th layer, where each cluster is then treated as an input node. Mathematically, DIFFPOOL constructs the two following functions as:

$$\mathbf{H}^{(k+1)} = \mathbf{S}^{(k)\mathsf{T}}\mathsf{GraphSAGE}^{(l)}_{\mathsf{embed}}\left(\mathbf{A}^{(k)}, \mathbf{H}^{(k)}\right) \tag{2.40}$$

$$\mathbf{A}^{(k+1)} = \mathbf{S}^{(k)\mathsf{T}}\mathbf{A}^{(k)}\mathbf{S}^{(k)} \tag{2.41}$$

where $\mathbf{S}^{(k)}$ denotes the learnable cluster assignment matrix to provide a soft assignment of each node at the k-th layer to a cluster at the $(k+1)$-th layer. $\mathbf{S}^{(k)}$ is defined as:

$$\mathbf{S}^{(k)} = \mathsf{softmax}\left(\mathsf{GraphSAGE}^{(l)}_{\mathsf{pool}}\left(\mathbf{A}^{(k)}, \mathbf{H}^{(k)}\right)\right) \tag{2.42}$$

DIFFPOOL assigns all nodes at the final layer to a single cluster to generate the final embedding $\mathbf{e}_{\mathcal{G}}$ of the entire graph $\mathcal{G}$.

Gated Graph Neural Network (Li et al., 2016b) defines a graph-level readout pooling function as:

$$\mathbf{e}_{\mathcal{G}} = \tanh\left(\sum_{v\in\mathcal{V}} \sigma\left(f_i\left(\left[\mathbf{h}_v^{(K)}; \boldsymbol{x}_v\right]\right)\right) \odot \tanh\left(f_j\left(\left[\mathbf{h}_v^{(K)}; \boldsymbol{x}_v\right]\right)\right)\right) \tag{2.43}$$

where $\mathbf{h}_{\mathsf{v}}^{(K)}$ is returned by Equation 2.30; $\sigma\left(f_i\left(\left[\mathbf{h}_{\mathsf{v}}^{(K)}; \boldsymbol{x}_{\mathsf{v}}\right]\right)\right)$ can be seen as a soft attention mechanism to decide which nodes are relevant to the graph-level task; f_i and f_j are neural networks with the input $\left[\mathbf{h}_{\mathsf{v}}^{(K)}; \boldsymbol{x}_{\mathsf{v}}\right]$. This pooling function is also utilised in Message Passing Neural Network (Gilmer et al., 2017).

PATCHY-SAN (Niepert et al., 2016) adapts a graph labeling procedure (such as Weisfeiler-Lehman (WL) relabeling process) to generate fixed-length sequences of nodes from a given graph. PATCHY-SAN then orders k-hop neighbours for each node in each generated sequence according to their graph labelings. After that, PATCHY-SAN selects a fixed number of ordered neighbours for each node and applies a convolutional neural network (LeCun et al., 1998) to predict the graph label.[2]

2.2 Knowledge Graph Embeddings

A knowledge graph $\mathcal{G}$ is a collection of valid factual triples in the form of *(head, relation, tail)* denoted as (h, r, t) such that $h, t \in \mathcal{E}$ and $r \in \mathcal{R}$ where $\mathcal{E}$ is a set of entities and $\mathcal{R}$ is a set of relations. In general, knowledge graph (KG) embedding models have been proposed to embed entities and relations to a low-dimensional vector space. On such space, one can define a score function f to score every triple (h, r, t) such that the valid triples obtain higher scores than the invalid triples.

The following sections introduce necessary background and a brief overview of existing KG embedding models.

2.2.1 Common loss and sampling functions

2.2.1.1 Loss functions

Let $\mathcal{G}$ and $\mathcal{G}'$ denote collections of valid and invalid triples respectively, wherein $\mathcal{G}'$ is generated by corrupting valid triples in $\mathcal{G}$. Let $\boldsymbol{\theta}$ be the model parameters. To $\boldsymbol{\theta}$, we can often minimise one of the following loss functions:

[2]Regarding other graph representation learning approaches, we refer the readers to the overview articles in (Hamilton et al., 2017b; Zhou et al., 2018; Zhang et al., 2018b; Wu et al., 2019b; Zhang et al., 2020).

- The margin-based loss function with L_2 regularisation on $\boldsymbol{\theta}$:

$$\mathcal{L}_{\mathsf{margin}} = \sum_{\substack{(h,r,t)\in\mathcal{G} \\ (h',r,t')\in\mathcal{G}'}} \max\left(0, \gamma - f\left(h,r,t\right) + f\left(h',r,t'\right)\right) + \lambda\|\boldsymbol{\theta}\|_2^2 \tag{2.44}$$

  where γ is the margin hyper-parameter; and λ is the regularisation rate.

- The logistic-based loss function (Trouillon et al., 2016) with L_2 regularisation on $\boldsymbol{\theta}$:

$$\mathcal{L}_{\mathsf{softplus}} = \sum_{(h,r,t)\in\{\mathcal{G}\cup\mathcal{G}'\}} \log\left(1 + \exp\left(-l_{(h,r,t)} \cdot f(h,r,t)\right)\right) + \lambda\|\boldsymbol{\theta}\|_2^2 \tag{2.45}$$

$$\text{in which, } l_{(h,r,t)} = \begin{cases} 1 & \text{for } (h,r,t) \in \mathcal{G} \\ -1 & \text{for } (h,r,t) \in \mathcal{G}' \end{cases} \tag{2.46}$$

- The negative log-likelihood-based loss function (Toutanova and Chen, 2015):

$$\mathcal{L}_{\mathsf{NLL}} = -\sum_{(h,r,t)\in\mathcal{G}} \log\left(\frac{\exp\left(f(h,r,t)\right)}{\sum_{t'\in\mathcal{E}\setminus\{t\}}\exp\left(f(h,r,t')\right)}\right) - \sum_{(h,r,t)\in\mathcal{G}} \log\left(\frac{\exp\left(f(h,r,t)\right)}{\sum_{h'\in\mathcal{E}\setminus\{h\}}\exp\left(f(h',r,t)\right)}\right) \tag{2.47}$$

- The binary cross-entropy loss function:

$$\mathcal{L}_{\mathsf{cross-entropy}} = -\sum_{(h,r,t)\in\{\mathcal{G}\cup\mathcal{G}'\}} \left(l_{(h,r,t)} \log\left(p_{(h,r,t)}\right) + \left(1 - l_{(h,r,t)}\right) \log\left(1 - p_{(h,r,t)}\right)\right) \tag{2.48}$$

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

  where $p_{(h,r,t)} = \sigma\left(f(h,r,t)\right)$, i.e., applying the sigmoid function $\sigma(.)$ to the score.

- The cross-entropy loss function (Sun et al., 2019) with negative sampling (Mikolov et al., 2013b):

$$\mathcal{L}_{\mathsf{neg}} = -\sum_{(h,r,t)\in\mathcal{G}} \left(\log\sigma\left(\gamma - f\left(h,r,t\right)\right) + \frac{1}{n}\sum_{i=1}^{K}\log\sigma\left(f\left(h'_i,r,t'_i\right) - \gamma\right)\right) \tag{2.49}$$

where n is the embedding dimension, and (h'_i, r, t'_i) is the i-th invalid triple.

Among these functions, the margin and logistic-based losses are commonly used in translation-based approaches and later KG embedding models.

2.2.1.2 Sampling procedures

Common sampling strategies to generate $\mathcal{G}'$ are summarised as follows:

- Uniform negative sampling: one can replace the head entity or the tail entity by a random entity uniformly sampled from $\mathcal{E}$, but not both at the same time.

- Bernoulli negative sampling (Wang et al., 2014): Given each relation r, let η_h denote the averaged number of head entities per tail entity whilst η_t denote the averaged number of tail entities per head entity. Given a valid triple (h, r, t) of relation r, we then generate a new head entity h' with probability $\frac{\eta_t}{\eta_h+\eta_t}$ to form an invalid triple (h', r, t) and a new tail entity t' with probability $\frac{\eta_h}{\eta_h+\eta_t}$ to form an invalid triple (h, r, t'). The Bernoulli negative sampling procedure is widely used in KG embedding models later on.

- Self-adversarial negative sampling (Sun et al., 2019): it is a recent method proposed to sample invalid triples from the following distribution:

$$\mathrm{p}\left((h'_j, r, t'_j)|\{(h_i, r_i, t_i)\}\right) = \frac{\exp \alpha f(h'_j, r, t'_j)}{\sum_i \exp \alpha f(h'_i, r, t'_i)}$$

  where α is the temperature of sampling.

Note that for a fair comparison, we should apply the same negative sampling procedure for the baseline models as well as the proposed models.

2.2.2 Applications and evaluation protocols

2.2.2.1 Knowledge graph completion

In the knowledge graph completion task (Bordes et al., 2013), the goal is to predict a missing entity given another entity and their relation, e.g., inferring a head entity h given (r, t) or inferring a tail entity t given (h, r). The results are calculated by ranking the scores produced by the score function f on triples in the test set.

In particular, following Bordes et al. (2013), for each valid test triple (h, r, t), we replace either h or t by each of other entities to create a set of corrupted triples. We use the "Filtered" setting protocol (Bordes et al., 2013), i.e., not including any corrupted triples that appear in the graph. We rank the valid test triple and corrupted triples in descending order based on their scores. We employ several metrics: mean rank (MR), mean reciprocal rank (MRR), and Hits@k (the proportion of the valid triples ranking in top k predictions). The final scores on the test set are reported for the model which obtains the highest Hits@10 on the validation set. Lower MR, higher MRR, and higher Hits@k indicate better performance.

2.2.2.2 Triple classification

The triple classification task is to predict whether a given triple (s, r, o) is valid or not (Socher et al., 2013a). Each relation r has a threshold θ_r determined by maximizing the micro-averaged classification accuracy on the validation set. If the score of a given triple (s, r, o) is above θ_r, then this triple is classified as a valid triple, otherwise it is classified as an invalid one. Note that this application was popular during 2013-2016, but nowadays it is not as widely used as the knowledge graph completion task.

2.2.2.3 Search personalisation

Personalised search systems utilise the historical interactions between the user and the systems, such as submitted queries and clicked documents to tailor returned results to the need of that user (Teevan et al., 2005, 2009). Widely used approaches to build effective search systems consist of two separated steps: (1) building the user profile from user's

historical interactions; and then (2) learning a ranking function to re-rank the search results based on user profile (Bennett et al., 2012; White et al., 2013; Harvey et al., 2013; Vu et al., 2015). Given a submitted *query* for a *user*, the goal is to re-rank the *documents* returned by a search system, so that the more the returned documents are relevant for that query, the higher their ranks are. In this case, apart from the user profile, dozens of other features have been proposed as the input of a learning-to-rank algorithm (Bennett et al., 2012; White et al., 2013).

Alternatively, we can follow Vu et al. (2017)'s prospective strategy of extending KG embedding models (e.g., TransE (Bordes et al., 2013)) to improve the ranking quality of the search personalisation systems, by viewing a relationship of the submitted query, the user and the returned document as a *(s, r, o)*-like triple *(query, user, document)*. Experimental results in (Vu et al., 2017) demonstrate that TransE outperforms the standard ranker as well as competitive search personalisation baselines (Teevan et al., 2011; Bennett et al., 2012; Vu et al., 2015). Therefore, we now can evaluate a KG embedding model via the search personalisation task as follows: (i) first train the model and use the trained model to compute a score for each *(query, user, document)* triple; (ii) then sort the scores in the descending order to obtain a new ranked list; (iii) finally employ two standard evaluation metrics: mean reciprocal rank (MRR) and Hits@1. For each metric, the higher value indicates better ranking performance.

2.2.3 Knowledge graph embedding approaches

This section is used to summarise existing KG embedding methods, wherein the material is partially based on (Nguyen, 2020). Table 2.1 illustrates the score functions $f(h, r, t)$ in some previous approaches.

2.2.3.1 Translation-based approaches

Early translation-based approaches exploit a translational characteristic so that the embedding of tail entity t should be close to the embedding of head entity h plus the embedding

Model	The score function $f(h, r, t)$
TransE	$-\|\boldsymbol{v}_h + \boldsymbol{v}_r - \boldsymbol{v}_t\|_p$ where $\boldsymbol{v}_h$, $\boldsymbol{v}_r$, and $\boldsymbol{v}_t \in \mathbb{R}^n$; $\|\boldsymbol{v}\|_p$ denotes the p-norm of vector $\boldsymbol{v}$
STransE	$-\|\mathbf{W}_{r,1}\boldsymbol{v}_h + \boldsymbol{v}_r - \mathbf{W}_{r,2}\boldsymbol{v}_t\|_p$ where $\mathbf{W}_{r,1}$ and $\mathbf{W}_{r,2} \in \mathbb{R}^{n\times n}$
ConvE	$\boldsymbol{v}_t^{\mathsf{T}}\mathsf{ReLU}\left(\mathbf{W}\mathsf{vec}\left(\mathsf{ReLU}\left([\widehat{\boldsymbol{v}}_h; \widehat{\boldsymbol{v}}_r] * \boldsymbol{\Omega}\right)\right)\right)$ where $*$ denotes a convolution operator; $\boldsymbol{\Omega}$ denotes a set of filters; $\mathsf{vec}(.)$ denotes a vectorisation; $[;]$ denotes a vector concatenation; and $\widehat{\boldsymbol{v}}$ denotes a 2D reshaping of $\boldsymbol{v}$
ConvKB	$\mathbf{w}^{\mathsf{T}}\mathsf{concat}\left(\mathsf{ReLU}\left([\boldsymbol{v}_h, \boldsymbol{v}_r, \boldsymbol{v}_t] * \boldsymbol{\Omega}\right)\right)$ where concat denotes a concatenation of feature maps; $[\boldsymbol{v}_h, \boldsymbol{v}_r, \boldsymbol{v}_t]$ denotes the 3-column embedding matrix of the input triple
TuckER	$\mathcal{W} \times_1 \boldsymbol{v}_h \times_2 \boldsymbol{v}_r \times_3 \boldsymbol{v}_t$ where $\times_d$ denotes the tensor product along the d-th mode
DistMult	$\langle \boldsymbol{v}_h, \boldsymbol{v}_r, \boldsymbol{v}_t \rangle = \sum_i^n \boldsymbol{v}_{h_i} \boldsymbol{v}_{r_i} \boldsymbol{v}_{t_i}$ where $\langle\rangle$ denotes a multiple-linear dot product
ComplEx	$\mathsf{Re}\left(\langle \boldsymbol{v}_h, \boldsymbol{v}_r, \boldsymbol{v}_t^* \rangle\right)$ where $\mathsf{Re}(c)$ denotes the real part of the complex c $\boldsymbol{v}_h$, $\boldsymbol{v}_r$, and $\boldsymbol{v}_t \in \mathbb{C}^n$; $\boldsymbol{v}^*$ denotes the conjugate of the complex vector $\boldsymbol{v}$
RotatE	$-\|\boldsymbol{v}_h \circ \boldsymbol{v}_r - \boldsymbol{v}_t\|_p$ where $\boldsymbol{v}_h$, $\boldsymbol{v}_r$, and $\boldsymbol{v}_t \in \mathbb{C}^n$; and $\circ$ denotes the element-wise product
QuatE	$(\boldsymbol{v}_h \otimes \boldsymbol{v}_r^{\triangleleft}) \bullet \boldsymbol{v}_t$ where $\boldsymbol{v}_h$, $\boldsymbol{v}_r$, and $\boldsymbol{v}_t \in \mathbb{H}^n$; $\bullet$ denotes a quaternion-inner product $\otimes$ denotes the Hamilton product; the superscript $^{\triangleleft}$ denotes the normalised embedding
QuatRE	$\left(\left(\boldsymbol{v}_h \otimes \mathbf{v}_{r,1}^{\triangleleft}\right) \otimes \boldsymbol{v}_r^{\triangleleft}\right) \bullet \left(\boldsymbol{v}_t \otimes \mathbf{v}_{r,2}^{\triangleleft}\right)$ where $\boldsymbol{v}_h$, $\boldsymbol{v}_r$, $\boldsymbol{v}_t$, $\mathbf{v}_{r,1}$, and $\mathbf{v}_{r,2} \in \mathbb{H}^n$

Table 2.1: Score functions. The table is adapted from (Nguyen, 2020).

of relation t. For example, TransE (Bordes et al., 2013) defines a score function:

$$f_{\mathsf{TransE}}(h, r, t) = -\|\boldsymbol{v}_h + \boldsymbol{v}_r - \boldsymbol{v}_t\|_p \tag{2.50}$$

where $\boldsymbol{v}_h$, $\boldsymbol{v}_r$, and $\boldsymbol{v}_t \in \mathbb{R}^n$ are vector embeddings of h, r and t respectively; and $\|\boldsymbol{v}\|_p$ denotes the p-norm of vector $\boldsymbol{v}$. The TransE is suitable for 1-to-1 relationships, but not well-adapted for Many-to-1, 1-to-Many, and Many-to-Many relationships. Therefore, some translation-based methods, as illustrated in Figure 1.4, have been proposed to deal with this issue.

The first method is TransH (Wang et al., 2014) that extends TransE to allow entities playing different roles given different relations. Each relation r is associated with a relation-specific hyperplane $\mathbf{w}_r$, and then the embeddings of h and t are projected to this hyperplane. TransH constructs a score function as:

$$f_{\mathsf{TransH}}\left(h, r, t\right) = -\|\boldsymbol{v}_{h\perp} + \boldsymbol{v}_r - \boldsymbol{v}_{t\perp}\|_p \tag{2.51}$$

where $\boldsymbol{v}_{h\perp} = \boldsymbol{v}_h - \mathbf{w}_r^\mathsf{T}\boldsymbol{v}_h\mathbf{w}_r$ and $\boldsymbol{v}_{t\perp} = \boldsymbol{v}_t - \mathbf{w}_r^\mathsf{T}\boldsymbol{v}_t\mathbf{w}_r$ are the projected embeddings of h and t on $\mathbf{w}_r$ respectively.

Later on, TransR (Lin et al., 2015b) extends TransH to associate each relation r with a projection matrix $\mathbf{W}_r$, which is used to project the entity embeddings into the vector space of relations as:

$$f_{\mathsf{TransR}}(h,r,t) = -\|\mathbf{W}_r\boldsymbol{v}_h + \boldsymbol{v}_r - \mathbf{W}_r\boldsymbol{v}_t\|_p \tag{2.52}$$

STransE (Nguyen et al., 2016) extends TransR to associate the head and tail entities with their own projection matrices respectively as:

$$f_{\mathsf{STransE}}(h,r,t) = -\|\mathbf{W}_{r,1}\boldsymbol{v}_h + \boldsymbol{v}_r - \mathbf{W}_{r,2}\boldsymbol{v}_t\|_p \tag{2.53}$$

2.2.3.2 Tensor-based approaches

Tensor-based approaches have produced potential performance for the knowledge graph completion task. Notably, DistMult (Yang et al., 2015) is also viewed as using a multiple-linear dot product to score the triples as:

$$f_{\mathsf{DistMult}}(h,r,t) = \langle \boldsymbol{v}_h, \boldsymbol{v}_r, \boldsymbol{v}_t \rangle = \sum_i^n \boldsymbol{v}_{h_i}\boldsymbol{v}_{r_i}\boldsymbol{v}_{t_i} \tag{2.54}$$

where $\langle\rangle$ denotes a multiple-linear dot product.

SimplE (Kazemi and Poole, 2018) considers two separate embeddings for each entity to be learned dependently. Then SimplE extends DistMult to define a score function as:

$$f_{\mathsf{SimplE}}(h,r,t) = \frac{1}{2}\left(\langle \boldsymbol{v}_{h,1}, \boldsymbol{v}_r, \boldsymbol{v}_{t,2}\rangle + \langle \boldsymbol{v}_{t,1}, \boldsymbol{v}_{r^{-1}}, \boldsymbol{v}_{h,2}\rangle\right) \tag{2.55}$$

where r^{-1} denotes the inverse relation of r to allow a new inverse triple (t, r^{-1}, h).

TuckER (Balažević et al., 2019b) is based on Tucker decomposition to factorise the

binary tensor of triples into a core tensor multiplied by a matrix along each mode as:

$$f_{\mathsf{TuckER}}(h,r,t) = \mathcal{W} \times_1 \boldsymbol{v}_h \times_2 \boldsymbol{v}_r \times_3 \boldsymbol{v}_t \tag{2.56}$$

where $\times_d$ denotes the tensor product along the d-th mode.

HolE (Nickel et al., 2016) employs a circular correlation product to define a score function as:

$$f_{\mathsf{HolE}}(h,r,t) = \sigma\left(\boldsymbol{v}_t^{\mathsf{T}}\left(\boldsymbol{v}_h \star \boldsymbol{v}_r\right)\right) \tag{2.57}$$

where σ denotes the sigmoid function, and $\star$ denotes the circular correlation product.

2.2.3.3 Neural network-based approaches

Neural Tensor Network (NTN) (Socher et al., 2013a) utilises a bilinear tensor layer across multiple dimensions to compute the score as:

$$f_{\mathsf{NTN}}(h,r,t) = \boldsymbol{v}_r^{\mathsf{T}}\mathsf{tanh}\left(\boldsymbol{v}_h^{\mathsf{T}}\mathbf{W}_r^{[1:k]}\boldsymbol{v}_t + \mathbf{V}_r\left[\boldsymbol{v}_h;\boldsymbol{v}_t\right] + \mathbf{b}_r\right) \tag{2.58}$$

where $\mathbf{W}_r^{[1:k]} \in \mathbb{R}^{n\times n\times k}$ is a tensor; the bilinear tensor product $\boldsymbol{v}_h^{\mathsf{T}}\mathbf{W}_r^{[1:k]}\boldsymbol{v}_t$ results in a vector in $\mathbb{R}^k$; $\mathbf{V}_r \in \mathbb{R}^{k\times 2n}$; and $[;]$ denotes a vector concatenation.

ER-MLP (Dong et al., 2014) concatenates the embeddings of h, r, and t into a vector, which is then fed to a multi-layer perceptron network with one-neuron output layer as:

$$f_{\mathsf{ER-MLP}}(h,r,t) = \sigma\left(\mathbf{w}^{\mathsf{T}}\mathsf{tanh}\left(\mathbf{W}[\boldsymbol{v}_h;\boldsymbol{v}_r;\boldsymbol{v}_t]\right)\right) \tag{2.59}$$

A more recent trend is to apply deep neural networks to calculate the triple scores (Dettmers et al., 2018; Schlichtkrull et al., 2018; Nguyen et al., 2018; Vashishth et al., 2020a). For example, ConvE (Dettmers et al., 2018) uses a convolution layer (LeCun et al., 1998) on a 2D input matrix of reshaping the embeddings of both the head entity and relation to produce feature maps that are then vectorised and computed with the

embedding of the tail entity to return the score. ConvE defines a score function as:

$$f_{\mathsf{ConvE}}\left(h,r,t\right)=\boldsymbol{v}_t^{\mathsf{T}}\mathsf{ReLU}\left(\mathbf{W}\mathsf{vec}\left(\mathsf{ReLU}\left(\left[\widehat{\boldsymbol{v}}_h;\widehat{\boldsymbol{v}}_r\right]*\boldsymbol{\Omega}\right)\right)\right) \tag{2.60}$$

where $*$ denotes a convolution operator; $\boldsymbol{\Omega}$ denotes a set of filters; $\mathsf{vec}(.)$ denotes a vectorisation; $[;]$ denotes a concatenation; and $\widehat{\boldsymbol{v}}$ denotes a 2D reshaping of $\boldsymbol{v}$.

Our proposed ConvKB (Nguyen et al., 2018), presented in Chapter 4, applies a convolutional layer on the 3-column embedding matrix of the input triple. It then concatenates the feature maps into a single vector, which is computed with a weight vector to produce the score. ConvKB proposes the following score function:

$$f_{\mathsf{ConvKB}}\left(h,r,t\right)=\mathbf{w}^{\mathsf{T}}\mathsf{concat}\left(\mathsf{ReLU}\left(\left[\boldsymbol{v}_h,\boldsymbol{v}_r,\boldsymbol{v}_t\right]*\boldsymbol{\Omega}\right)\right) \tag{2.61}$$

where $[\boldsymbol{v}_h,\boldsymbol{v}_r,\boldsymbol{v}_t]$ denotes the 3-column embedding matrix of the input triple; and concat denotes a vector concatenation of feature maps.

GNN-based approaches. Furthermore, graph neural networks (GNNs)-based KG embedding approaches have been proposed to encode features from the neighbourhood structures (Schlichtkrull et al., 2018; Shang et al., 2019). In general, these GNN-based models adopt an encoder-decoder architecture, wherein the encoder module leverages GNNs to update the vector representations of entities and relations, and then the decoder module utilises a score function (e.g., as employed in TransE, DistMult, and ConvE) to measure the triples. For example, R-GCN (Schlichtkrull et al., 2018) modifies GCNs (Kipf and Welling, 2017) to introduce a specific encoder to update only entity embeddings as:

$$\boldsymbol{v}_{\mathsf{e}}^{(k+1)}=\sigma\left(\sum_{r\in\mathcal{R}}\sum_{\mathsf{e}'\in\mathcal{N}_{\mathsf{e}}^r}\frac{1}{c_{\mathsf{e},r}}\mathbf{W}_r^{(k)}\boldsymbol{v}_{\mathsf{e}'}^{(k)}+\mathbf{W}^{(k)}\boldsymbol{v}_{\mathsf{e}}^{(k)}\right) \tag{2.62}$$

where $\mathcal{N}_{\mathsf{e}}^r$ denotes the set of entity neighbours of entity e via relation edge r; and $c_{\mathsf{e},r}$ is a problem-specific normalisation that can either be fixed to $|\mathcal{N}_{\mathsf{e}}^r|$ or learned as a model

parameter. R-GCN then uses DistMult as its decoder module.

SACN (Shang et al., 2019) proposes a weighted graph convolutional network (WGCN) as the encoder module as:

$$\boldsymbol{v}_{\mathsf{e}}^{(k+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{\mathsf{e}'\in\mathcal{N}_{\mathsf{e}}^{r}} \alpha_{r}^{(k)}\mathbf{W}^{(k)}\boldsymbol{v}_{\mathsf{e}'}^{(k)} + \mathbf{W}^{(k)}\boldsymbol{v}_{\mathsf{e}}^{(k)}\right) \tag{2.63}$$

where $\alpha_r^{(k)}$ is a learnable parameter weight for the relation edge r at the k-th layer. SACN then introduces Conv-TransE as the decoder module, which employs a convolutional layer on the 2-column matrix of vector representations of h and r and then concatenates the output feature maps into a single vector. This vector is fed to a single fully-connected layer to produce a vector, which is multiplied by the vector representation of t to return a score. Formally, SACN defines the decoder module, named Conv-TransE, as:

$$f_{\mathsf{Conv-TransE}}\left(h,r,t\right) = \boldsymbol{v}_{t}^{\mathsf{T}}\mathsf{ReLU}\left(\mathbf{W}\mathsf{concat}\left(\mathsf{ReLU}\left(\left[\boldsymbol{v}_{h},\boldsymbol{v}_{r}\right]*\boldsymbol{\Omega}\right)\right)\right) \tag{2.64}$$

where $[\boldsymbol{v}_h, \boldsymbol{v}_r]$ denotes the 2-column matrix of vector representations of h and r.

KB-GAT (Nathani et al., 2019) adapts GAT (Veličković et al., 2018) to construct the encoder module as:

$$\boldsymbol{v}_{\mathsf{e}}^{(k+1)} = \sigma\left(\sum_{r\in\mathcal{R}}\sum_{\mathsf{e}'\in\mathcal{N}_{\mathsf{e}}^{r}} \tau_{\mathsf{e},r,\mathsf{e}'}^{(k)}\mathbf{W}_{0}^{(k)}\left[\boldsymbol{v}_{\mathsf{e}}^{(k)};\boldsymbol{v}_{r}^{(k)};\boldsymbol{v}_{\mathsf{e}'}^{(k)}\right]\right) \tag{2.65}$$

$$\boldsymbol{v}_{r}^{(k+1)} = \mathbf{W}_{1}^{(k)}\boldsymbol{v}_{r}^{(k)} \tag{2.66}$$

where $\tau_{\mathsf{e},r,\mathsf{e}'}^{(k)}$ is an edge weight, which is computed as:

$$\tau_{\mathsf{e},r,\mathsf{e}'}^{(k)} = \mathsf{softmax}_{r,\mathsf{e}'}\left(\mathsf{LeakyReLU}\left(\boldsymbol{a}^{(k)\mathsf{T}}\mathbf{W}_{0}^{(k)}\left[\boldsymbol{v}_{\mathsf{e}}^{(k)};\boldsymbol{v}_{r}^{(k)};\boldsymbol{v}_{\mathsf{e}'}^{(k)}\right]\right)\right) \tag{2.67}$$

where [;] denotes a vector concatenation. Similar to GAT, KB-GAT also employs the multi-head attention technique (Vaswani et al., 2017), except the last layer using averaging. To

avoid losing the initial embedding information, the encoder module in KB-GAT returns the vector representation $\boldsymbol{v}_{\mathsf{e}}$ for entity e as:

$$\boldsymbol{v}_{\mathsf{e}} = \boldsymbol{v}_{\mathsf{e}}^{(K)} + \mathbf{W}_2 \boldsymbol{v}_{\mathsf{e}}^{(0)} \tag{2.68}$$

where K is the index of the last layer. After that, KB-GAT adopts our ConvKB (Nguyen et al., 2018) as the decoder module.

CompGCN (Vashishth et al., 2020b) extends GCNs to consider composition operations between entities and relations in the encoder module as follows:

$$\boldsymbol{v}_{\mathsf{e}}^{(k+1)} = \mathrm{g}\left(\sum_{(\mathsf{e}',r)\in\mathcal{N}_{\mathsf{e}}} \mathbf{W}_{\mathsf{dir}(r)}^{(k)} \phi\left(\boldsymbol{v}_{\mathsf{e}'}^{(k)}, \boldsymbol{v}_r^{(k)}\right)\right) \tag{2.69}$$

$$\boldsymbol{v}_r^{(k+1)} = \mathbf{W}^{(k)} \boldsymbol{v}_r^{(k)} \tag{2.70}$$

where $\mathcal{N}_{\mathsf{e}}$ is extended with inverse and self-looping relations; $\mathbf{W}_{\mathsf{dir}(r)}^{(k)}$ denotes direction specific weights as:

$$\mathbf{W}_{\mathsf{dir}(r)}^{(k)} = \begin{cases} \mathbf{W}_O^{(k)} & \text{for } r \in \mathcal{R} \\ \mathbf{W}_I^{(k)} & \text{for } r \in \mathcal{R}^{-1} \\ \mathbf{W}_S^{(k)} & r \text{ is a self-loop} \end{cases} \tag{2.71}$$

where $\mathcal{R}^{-1}$ denotes the set of inverse relation edges r^{-1} of r. CompGCN explores the composition functions (ϕ) inspired from TransE (Bordes et al., 2013), DistMult (Yang et al., 2015), and HolE (Nickel et al., 2016) as:

$$\phi\left(\mathbf{a}, \mathbf{b}\right) = \begin{cases} \mathbf{a} - \mathbf{b} & \text{if using subtraction} \\ \mathbf{a} \circ \mathbf{b} & \text{if using multiplication} \\ \mathbf{a} \star \mathbf{b} & \text{if using circular correlation} \end{cases} \tag{2.72}$$

where $\circ$ denotes the element-wise product; and $\star$ denotes the circular correlation product. CompGCN then applies ConvE (Dettmers et al., 2018) as the decoder module.

Compared to the vanilla GCNs in Equation 2.19, we see that R-GCN does not comprise

relation embeddings and CompGCN does not treat each relation as an individual node in the encoder module. These limitations also exist in other GNN-based models such as SACN (Shang et al., 2019) and KB-GAT (Nathani et al., 2019). Therefore, arguably these could lower the performance of the existing GNN-based models.

Transformer-based approaches. More recently, KG-BERT (Yao et al., 2019), CoKE (Wang et al., 2019a), K-BERT (Liu et al., 2020), and our proposed R-MeN (Nguyen et al., 2020c) are among the first models to leverage the transformer (Vaswani et al., 2017) for knowledge graph embeddings. Our R-MeN, presented in Chapter 4, utilises a transformer-based memory network (Santoro et al., 2018) to capture the triples for the triple classification task, while the latter work, CoKE, adapts the transformer encoder to model the triples for the knowledge graph completion task. Besides, KG-BERT and K-BERT employ BERT (Devlin et al., 2018) with textual mentions derived from a large external corpus.

2.2.3.4 Complex vector-based approaches

Several works have moved beyond the Euclidean vector space to the hyper-complex vector spaces. ComplEx (Trouillon et al., 2016) extends DistMult to use the multiple-linear dot product on the complex vector embeddings of entities and relations as:

$$f_{\mathsf{ComplEx}}\left(h, r, t\right) = \mathsf{Re}\left(\langle \boldsymbol{v}_h, \boldsymbol{v}_r, \boldsymbol{v}_t^* \rangle\right) \tag{2.73}$$

where $\mathsf{Re}(c)$ denotes the real part of the complex c; $\boldsymbol{v}_h$, $\boldsymbol{v}_r$, and $\boldsymbol{v}_t \in \mathbb{C}^n$; and $\boldsymbol{v}^*$ denotes the conjugate of the complex vector $\boldsymbol{v}$. In addition, RotatE (Sun et al., 2019) considers each relation as a rotation-based translation from the head entity to the tail entity within the complex vector space as:

$$f_{\mathsf{RotatE}}\left(h, r, t\right) = -\|\boldsymbol{v}_h \circ \boldsymbol{v}_r - \boldsymbol{v}_t\|_p \tag{2.74}$$

where $\boldsymbol{v}_h$, $\boldsymbol{v}_r$, and $\boldsymbol{v}_t \in \mathbb{C}^n$; and $\circ$ denotes the element-wise product.

QuatE (Zhang et al., 2019) is proposed to learn entity and relation embeddings within the Quaternion space. QuatE is considered as one of the recent state-of-the-art models as it outperforms up-to-date baselines for knowledge graph completion. In particular, QuatE uses a Hamilton product-based rotation followed by a quaternion-inner product to produce the triple score. Mathematically, QuatE computes the score of the triple *(h, r, t)* as:

$$f_{\mathsf{QuatE}}(h, r, t) = (\boldsymbol{v}_h \otimes \boldsymbol{v}_r^{\triangleleft}) \bullet \boldsymbol{v}_t \tag{2.75}$$

where $\otimes$ denotes the Hamilton product; $\bullet$ denotes the quaternion-inner product; and $\boldsymbol{v}_h$, $\boldsymbol{v}_r$, and $\boldsymbol{v}_t \in \mathbb{H}^n$, QuatE, however, has a limitation in capturing the correlations between the head and tail entities. Our QuatRE (Nguyen et al., 2020e), presented in Chapter 4, is proposed to overcome this limitation by integrating relation-aware rotations to increase the correlations between the entities.

2.2.3.5 Other approaches.

Some approaches utilises relation paths between entities to attain contextual information to improve the task performance. For example, CCP (Luo et al., 2015) generates sequences of entities and relations occurring in the pattern and then applies Word2Vec to learn embeddings for entities and relations; then these learned embeddings are used to initialise entity and relation embeddings in TransE. In addition, PTransE-ADD (Lin et al., 2015a) and TransE-COMP (Guu et al., 2015) consider multiple-step relation paths connecting two entities and define an additional function as:

$$f(h, \mathsf{path}, t) = -\|\boldsymbol{v}_h + \boldsymbol{v}_{r_1} + \boldsymbol{v}_{r_2} + ... + \boldsymbol{v}_{r_l} - \boldsymbol{v}_t\|_p \tag{2.76}$$

where path denotes a sequence of relations $\{r_1, r_2, ..., r_l\}$ between h and t. Besides, as mentioned in Section 2.2.3.3, CoKE (Wang et al., 2019a) is used to model the triples for the knowledge graph completion task, it also leverages the relation paths $(h, path, t)$ for the transformer encoder to answer path queries on KGs (Guu et al., 2015). Moreover, some

other approaches incorporate textual mentions derived from a large external corpus to further increase the performance (Toutanova et al., 2015; Wang and Li, 2016; García-Durán and Niepert, 2017; Yao et al., 2019; Liu et al., 2020).

2.3 Summary

In this chapter, we have briefly introduced the necessary background and related work for graph representation learning in Section 2.1 and knowledge graph embeddings in Section 2.2. These background and related work are the foundation for our new models proposed in Chapters 3 and 4. Figure 2.9 shows a taxonomy of our contributions in this thesis.

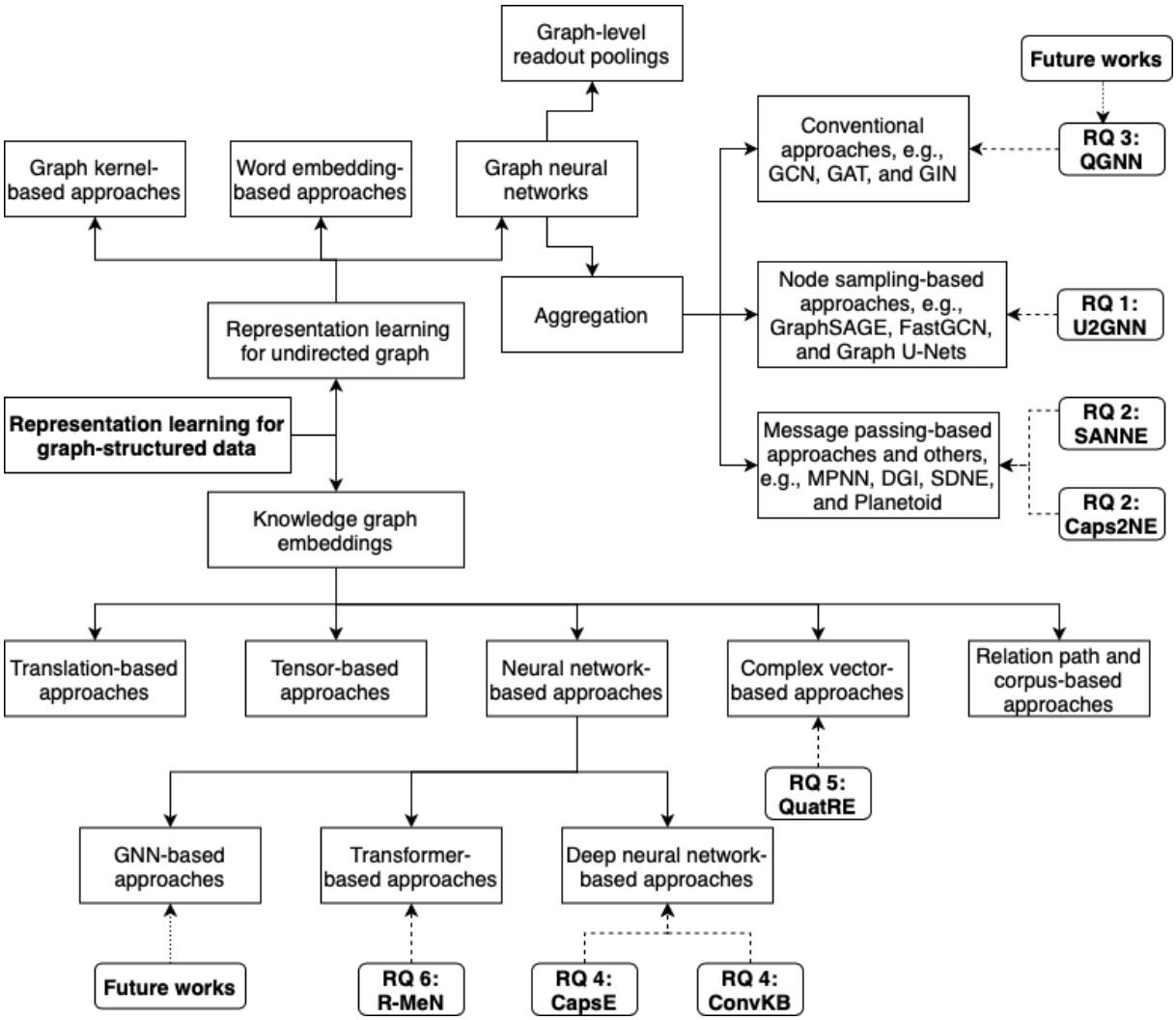

Figure 2.9: Contribution taxonomy.

Chapter 3

Graph Neural Networks

Contents

In this chapter, we briefly introduce our proposed models to learn node and graph embeddings in Section 3.1 and then present our research contributions in Section 3.2.

3.1 Introduction

Graph neural networks (GNNs) have become a leading research direction to learn low-dimensional continuous embedding vectors for nodes and graphs (Scarselli et al., 2009; Hamilton et al., 2017b; Wu et al., 2019b; Zhang et al., 2020). In general, GNNs utilise an aggregation function to update the vector representation of each node by aggregating those of its neighbours (Kipf and Welling, 2017; Hamilton et al., 2017a; Veličković et al., 2018). GNNs also apply a graph-level pooling function such as a simple sum pooling (Xu et al., 2019) to obtain graph embeddings. To further improve the GNN performance, we propose ***U2GNN*** that leverages the transformer self-attention network (Vaswani et al., 2017; Dehghani et al., 2019) to induce a more advanced aggregation function. The model

details are presented in Section 3.2.1.

Existing GNN models mainly focus on the *transductive* setting, where a model is trained using the entire input graph, i.e., all nodes with a fixed graph structure are present during training (Kipf and Welling, 2017). On the other hand, limited research has been conducted for an *inductive* setting (Yang et al., 2016), where embeddings are required for new nodes – a setting encountered commonly in practical applications of deep learning for graph networks. This significantly affects the performances of downstream tasks such as node classification. To that end, we introduce ***SANNE*** and ***Caps2NE*** – two unsupervised learning models – which aim to learn node embeddings in both the transductive and inductive settings. The details of our two models SANNE and Caps2NE are described in Section 3.2.2.

Note that most of the existing GNNs learn node and graph embeddings within the Euclidean vector space. However, for complex graphs such as protein interaction networks and social networks, the learned Euclidean embeddings have high distortion (Chami et al., 2019). Furthermore, when increasing the number of hidden layers, the existing GNNs (Kipf and Welling, 2017; Hamilton et al., 2017a; Veličković et al., 2018; Xu et al., 2019) are not working very efficiently anymore since the number of parameters grows quickly. To this end, we present a novel model, named ***QGNN***, to learn node and graph embeddings within the Quaternion space. The details of our QGNN are given in Section 3.2.3.

3.2 Research Contribution

3.2.1 Graph transformer self-attention networks

• **Dai Quoc Nguyen**, Tu Dinh Nguyen and Dinh Phung. Universal Graph Transformer Self-Attention Networks. *arXiv preprint arXiv:1909.11855, 2019.*

Contribution. The transformer self-attention network (Vaswani et al., 2017; Dehghani et al., 2019) has been widely applied as a novelty in research domains such as computer vision and NLP. Similarity, we also consider the successful use of this recent advanced

technique to a new domain, i.e., graph neural networks (GNNs), as a novel application. Moreover, as also discussed in (Xu et al., 2019), constructing an powerful aggregation mechanism is essential for GNNs. To this end, we present U2GNN (Nguyen et al., 2019b) to induce an advanced aggregation function, using the universal transformer network (Dehghani et al., 2019) consisting of a self-attention mechanism (Vaswani et al., 2017) followed by a recurrent transition (TRANS) with adding residual connections (He et al., 2016) and layer normalisation (LNORM) (Ba et al., 2016). In particular, given an input graph, we uniformly sample a set of neighbours for each node, which is fed to the U2GNN aggregation function of using multiple layers stacked on top of each other. We then apply a vector concatenation across the layers to obtain the final embeddings of nodes. After that, we sum all these final node embeddings to get the final embedding of the entire graph. We feed the graph embedding to a single fully-connected layer followed by a softmax layer to predict the graph label. The proposed U2GNN obtains state-of-the-art accuracies on well-known benchmark datasets for the graph classification task. The code is available at: `https://github.com/daiquocnguyen/Graph-Transformer`.

Universal Graph Transformer Self-Attention Networks

Dai Quoc Nguyen[1], Tu Dinh Nguyen[2], and Dinh Phung[1]

[1]Dept of Data Science and AI, Monash University, Australia
{dai.nguyen,dinh.phung}@monash.edu
[2]VinAI Research, Vietnam
v.tund21@vinai.io

Abstract. The transformer self-attention network has been extensively used in research domains such as computer vision, image processing, and natural language processing. The transformer, however, has not been actively used in graph neural networks, where constructing an advanced aggregation function is essential. To this end, we present an effective model, named U2GNN, which – by leveraging a transformer self-attention mechanism followed by a recurrent transition – induces an advanced aggregation function to learn graph representations. Experimental results show that U2GNN achieves state-of-the-art accuracies on well-known benchmark datasets for graph classification. Our code is available at: https://github.com/daiquocnguyen/Graph-Transformer.

Keywords: Graph neural networks · Graph classification · Transformer · Self-attention

1 Introduction

A graph is a connected network of nodes and edges. This type of graph-structured data is a fundamental mathematical representation and ubiquitous. They found applications in virtually all aspects of our daily lives from pandemic outburst response, information retrieval to circuit design, to name a few. In machine learning and data science, learning and inference from graphs have been one of the most important research topics. However, as data grow unprecedentedly in volume and complexity in modern time, traditional learning methods for graph are mostly inadequate to model increasing complexity, to harness rich contextual information as well as to scale with large-scale graphs. The recent rise of deep learning, and in turn, of representation learning field has radically advanced machine learning research in general, and pushing the frontier of graph learning. In particular, the notion of graph representation learning has recently emerged as a new promising learning paradigm, which aims to learn a parametric mapping function that embeds nodes, subgraphs, or the entire graph into low-dimensional continuous vector spaces [17,59]. The central challenge to this endeavor is to learn rich classes of complex functions to capture and preserve the graph structural

information as much as possible and also be able to geometrically represent the structural information in the embedded space.

Existing approaches can be categorised into three groups: (i) graph kernel-based methods, (ii) word embedding-based models, and (iii) graph neural networks. Methods in the first group build vectors of frequencies of "atomic subgraphs" decomposed from a given graph. Typical types of atomic subgraphs include random walks [13,22,45], shortest paths [3], graphlets [39], and Weisfeiler-Lehman subtree patterns [38]. In the second direction, several word embedding-based models [35,40,15] sample a large set of random walks, then treat each walk as a document whose words are nodes, and finally apply Word2Vec [31] on such random walk set to learn node embeddings. Meanwhile, some other ones aim to obtain the embeddings of the atomic subgraphs [52], and the entire graphs [32,20] by aggregating node embeddings learned by Word2Vec or using Doc2Vec [26].

Recently, graph neural networks (GNNs) become an essential strand, forming the third direction to learn low-dimensional continuous representations for nodes and graphs [36,17,59,48,55]. In general, GNNs use an aggregation function to update the vector representation of each node by transforming and aggregating the vector representations of its neighbours [24,16,43]. Then GNNs apply a graph-level pooling function (i.e., a readout operation such as simple sum pooling) to obtain graph embeddings [14,57,53,44,50]. GNN-based approaches provide faster and practical training, higher accuracy, and state-of-the-art results on benchmark datasets for downstream tasks such as graph classification [24,50].

To further improve the classification performance, it is worth developing an advanced aggregation function for GNNs to better update representations of nodes from their neighbours, as also discussed in [50]. Nowadays, there are novel applications of the transformer self-attention network [42,8] recognized, published, and used successfully in research domains such as computer vision, image processing, and natural language processing. Hence we consider the use of the transformer to a new domain such as GNNs as a novelty. Inspired by this self-attention network, we present U2GNN, an effective GNN model, which induces an advanced aggregation function leveraging a self-attention mechanism [42] followed by a recurrent transition, to update the vector representations of nodes from their neighbors. In particular, our U2GNN is different from related work as follows:

- The concurrent work – Hyper-SAGNN [58] – utilizes the transformer self-attention network for hypergraphs that have diverse and different structures, hence requiring a different model architecture. Besides, the *later* work, Graph-BERT [56], is an extension of our U2GNN for semi-supervised node classification.
- Graph Attention Network (GAT) [43] borrows the standard attention technique from [2] in using a single-layer feedforward neural network parametrized by a weight vector and then applying the non-linearity function followed by the softmax to compute the importance weights of neighbors of a given

node. Note that our U2GNN adopts a scaled dot-product attention mechanism which is more robust and efficient than the attention technique used in GAT.
- Regarding the model architecture, Graph Transformer Network [54] identifies useful meta-paths [46] to transform graph structures and applies GCN [24] to learn the node embeddings for the node classification task on heterogeneous graphs. Self-Attention Graph Pooling [27] re-implements the general architectures from Deep Graph Convolutional Neural Network [57] and Hierarchical Pooling [5] with using the Graph U-Net's node sampling [12] as the graph-level pooling layer.
- To this end, we note that U2GNN is entirely different from GAT [43], Graph Transformer Network [54], and Self-Attention Graph Pooling [27], except similar titles.

Contributions. Our main contributions in this paper are as follows:

- We propose U2GNN, an effective GNN model, leveraging the transformer self-attention network to construct an advanced aggregation function to learn the graph representations. To the best of our knowledge, our work is one of the firsts to explore transformer to graphs.
- Experimental results show that U2GNN obtains state-of-the-art accuracies on well-known benchmark datasets for the graph classification task.

2 Related work

Graph kernel-based methods

Graph kernel-based approaches decompose graphs into "atomic subgraphs" to measure the similarities among graphs [13]. Common types of atomic subgraphs consist of random walks [13,22,45], shortest paths [3], graphlets [39], and Weisfeiler-Lehman subtree patterns [38]. Here we view each atomic substructure as a word term and each graph as a text document. Next, we represent a collection of graphs as a document-term matrix whose elements are the normalised frequency of terms in documents. We then derive a valid kernel, for example, a linear one by taking inner product of every two documents, i.e., two graphs, and finally employ a kernel method such as Support Vector Machines (SVM) [19] to work on the graph classification problem. We refer the readers to the surveys of graph kernels in [34,25].

Word embedding-based models

There have been several methods adopting the word embedding frameworks such as Word2Vec [31] and Doc2Vec [26] to learn the embeddings for nodes and graphs to deal with the graph classification task. Deep graph kernel [52] applies Word2Vec to learn the embeddings for atomic substructures such as the graphlets, the Weisfeiler-Lehman subtree patterns, and the shortest paths, and

then derives the kernel between two graphs. Anonymous walk embedding [20] maps each random walk into an "anonymous walk" where each state is recorded by its first occurrence index in the random walk, then views each anonymous walk as a word token, and utilizes Doc2Vec to achieve the graph embeddings to compute the graph similarities to construct the kernel matrix. Graph2Vec [32] employs Doc2Vec on the Weisfeiler-Lehman subtree patterns to obtain the graph embeddings, which are then fed to SVM classifier training.

Graph neural networks

Recently, graph neural networks (GNNs) turn out to be a leading research direction to learn low-dimensional continuous embedding vectors for nodes and graphs [36,17,48,55]. In general, GNNs utilize an aggregation function to update the vector representation of each node by aggregating those of its neighbours [24,16,43]. GNNs also apply a graph-level pooling function such as a simple sum pooling [50] to obtain graph embeddings.

We represent each graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \{\mathbf{h}_\mathsf{v}\}_{\mathsf{v}\in\mathcal{V}})$, where $\mathcal{V}$ is a set of nodes, $\mathcal{E}$ is a set of edges, and $\mathbf{h}_\mathsf{v}$ (i.e., $\mathbf{h}_\mathsf{v}^{(0)}$) represents the feature vector of node $\mathsf{v} \in \mathcal{V}$. 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. Mathematically, given a graph $\mathcal{G}$, we formulate GNNs as follows:

$$\mathbf{h}_\mathsf{v}^{(k+1)} = \textsc{Aggregation}\left(\left\{\mathbf{h}_\mathsf{u}^{(k)}\right\}_{\mathsf{u}\in\mathcal{N}_\mathsf{v}\cup\{\mathsf{v}\}}\right) \tag{1}$$

$$\mathbf{e}_\mathcal{G} = \textsc{ReadOut}\left(\left\{\left\{\mathbf{h}_\mathsf{v}^{(k)}\right\}_{k=0}^{K}\right\}_{\mathsf{v}\in\mathcal{V}}\right) \tag{2}$$

where $\mathbf{h}_\mathsf{v}^{(k)}$ is the vector representation of node v at the k-th iteration/layer; $\mathcal{N}_\mathsf{v}$ is the set of neighbours 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. For example, Graph Convolutional Network (GCN) [24] updates vector representation for a given node $\mathsf{v} \in \mathcal{V}$ from its neighbours, using multiple layers stacked on top of each other as:

$$\mathbf{h}_\mathsf{v}^{(k+1)} = \mathsf{g}\left(\sum_{\mathsf{u}\in\mathcal{N}_\mathsf{v}\cup\{\mathsf{v}\}} a_{\mathsf{v},\mathsf{u}}\mathbf{W}^{(k)}\mathbf{h}_\mathsf{u}^{(k)}\right), \forall \mathsf{v} \in \mathcal{V} \tag{3}$$

where k is the layer index; $a_{\mathsf{v},\mathsf{u}}$ is an edge constant between nodes v and u in the re-normalised 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 $\tilde{\mathbf{A}}$; $\mathbf{W}^{(k)}$ is a weight matrix; $\mathbf{h}_\mathsf{u}^{(0)}$ is a feature vector of node u; g is a nonlinear activation function such as ReLU; and $\mathcal{N}_\mathsf{v}$ is the set of neighbours of node v.

GraphSAGE [16] extends GCN to use a node-wise procedure of uniformly sampling a fixed number of neighbours for each node at each layer as:

$$\mathbf{h}_{\mathsf{v}}^{(k+1)} = \mathsf{g}\left(\mathbf{W}^{(k)}\left[\mathbf{h}_{\mathsf{v}}^{(k)}; \mathbf{h}_{\mathcal{N}_{\mathsf{v}}'}^{(k)}\right]\right), \forall \mathsf{v} \in \mathcal{V} \tag{4}$$

where [;] denotes a vector concatenation, and $\mathbf{h}_{\mathcal{N}_{\mathsf{v}}'}^{(k)}$ can be obtained using an element-wise max-pooling operation as:

$$\mathbf{h}_{\mathcal{N}_{\mathsf{v}}'}^{(k)} = \mathsf{max}\left(\left\{\mathsf{g}\left(\mathbf{W}_{pool}\mathbf{h}_{\mathsf{u}}^{(k)} + \mathbf{b}\right)\right\}_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}'}\right) \tag{5}$$

where $\mathcal{N}_{\mathsf{v}}'$ is defined as a fixed-size, uniformly sampled from $\mathcal{N}_{\mathsf{v}}$ of v. Besides, $\mathcal{N}_{\mathsf{v}}'$ is sampled differently through each layer.

Graph Attention Network [43] extends GCN to compute edge weights following the standard attention technique [2] as:

$$\mathbf{h}_{\mathsf{v}}^{(k+1)} = \mathsf{g}\left(\sum_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}\cup\{\mathsf{v}\}} \tau_{\mathsf{v},\mathsf{u}}^{(k)}\mathbf{W}^{(k)}\mathbf{h}_{\mathsf{u}}^{(k)}\right), \forall \mathsf{v} \in \mathcal{V} \tag{6}$$

where g is the $\mathsf{sigmoid}$ function $\sigma(x) = \frac{1}{1+e^{-x}}$; and $\tau_{\mathsf{v},\mathsf{u}}^{(k)}$ is an edge weight between nodes v and u, which is computed as:

$$\tau_{\mathsf{v},\mathsf{u}}^{(k)} = \mathsf{softmax}\left(\mathsf{LeakyReLU}\left(\boldsymbol{a}^{(k)\mathsf{T}}\left[\mathbf{W}^{(k)}\mathbf{h}_{\mathsf{v}}^{(k)}; \mathbf{W}^{(k)}\mathbf{h}_{\mathsf{u}}^{(k)}\right]\right)\right) \tag{7}$$

where [;] denotes a vector concatenation. Besides, GAT employs the multi-head attention technique [42] to further stabilise the learning process, except the final (prediction) layer where GAT uses averaging.

While Simple Graph Convolution [47] is a simplified variant of GCN without using the non-linear activation function g, Graph Isomorphism Network [50] constructs an aggregation function based on a multi-layer perceptron (MLP) network of two fully-connected layers as:

$$\mathbf{h}_{\mathsf{v}}^{(k+1)} = \mathsf{MLP}^{(\mathsf{k})}\left(\sum_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}\cup\{\mathsf{v}\}} \mathbf{h}_{\mathsf{u}}^{(k)}\right), \forall \mathsf{v} \in \mathcal{V} \tag{8}$$

Following [51,50], we also employ a concatenation over the vector representations of node v at the different layers to construct a final vector representation $\mathbf{e}_{\mathsf{v}}$ for each node v as:

$$\mathbf{e}_{\mathsf{v}} = \left[\mathbf{h}_{\mathsf{v}}^{(1)}; \mathbf{h}_{\mathsf{v}}^{(2)}; ...; \mathbf{h}_{\mathsf{v}}^{(K)}\right], \forall \mathsf{v} \in \mathcal{V} \tag{9}$$

where K is the index of the last layer.

The graph-level readout function can be a simple sum pooling or a complicated pooling such as hierarchical pooling [5], and differentiable pooling [53].

As the sum pooling produces competitive results [50], we use the simple sum pooling to obtain the embedding $\mathbf{e}_{\mathcal{G}}$ of the entire graph $\mathcal{G}$ as:

$$\mathbf{e}_{\mathcal{G}} = \sum_{\mathsf{v} \in \mathcal{V}} \mathbf{e}_{\mathsf{v}} = \sum_{\mathsf{v} \in \mathcal{V}} \left[\mathbf{h}_{\mathsf{v}}^{(1)}; \mathbf{h}_{\mathsf{v}}^{(2)}; ...; \mathbf{h}_{\mathsf{v}}^{(K)} \right] \tag{10}$$

After that, we can follow [50] to feed the graph embeddings $\mathbf{e}_{\mathcal{G}}$ to a single fully-connected layer followed by a softmax layer to predict the graph labels.

3 U2GNN: Universal Graph Transformer Self-Attention Networks

The transformer self-attention network [42,8] has widely applied as a novelty in research domains such as computer vision and NLP. Similarity, we also consider the successful use of this recent advanced technique to a new domain, i.e., graph neural networks (GNNs), as a novel application. Moreover, as also discussed in [50], constructing an powerful aggregation mechanism is essential for GNNs. To this end, we induce an advanced aggregation function, using the universal transformer network [8] consisting of a self-attention mechanism [42] followed by a recurrent transition (TRANS) with adding residual connections [18] and layer normalization (LNORM) [1], as illustrated in Figure 1.

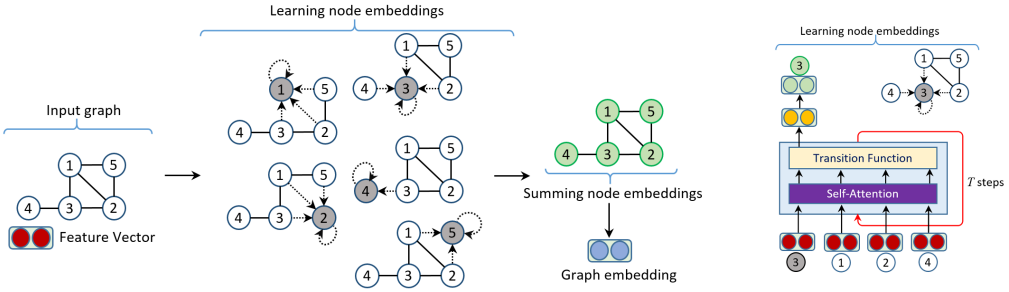

Fig. 1: Illustration of our U2GNN.

The residual connections [18] are used to add useful information learned in the lower layers to the higher layers, and more importantly, to allow gradients to directly pass through the layers to avoid vanishing gradient or exploding gradient problems. The layer normalization (LNORM) [1] is used to normalize the inputs across the feature dimensions to stabilize the network to enable smoother gradients and faster training. The residual connections and the layer normalization are commonly used in many architectures and thus are omitted in the paper for simplicity.

Formally, given an input graph $\mathcal{G}$, we uniformly sample a set $\mathcal{N}_{\mathsf{v}}$ of neighbors for each $\mathsf{v} \in \mathcal{V}$ and then input $\mathcal{N}_{\mathsf{v}} \cup \{\mathsf{v}\}$ to the U2GNN learning process. Note that

we sample a different $\mathcal{N}_{\mathsf{v}}$ for node v at each training batch. We also construct multiple layers stacked on top of each other in our U2GNN. Regarding the k-th layer, given a node $\mathsf{v} \in \mathcal{V}$, at each step t, we induce a transformer self-attention-based function to aggregate the vector representations for all nodes $\mathsf{u} \in \mathcal{N}_{\mathsf{v}} \cup \{\mathsf{v}\}$ as:

$$\mathbf{h}_{t,\mathsf{u}}^{(k)} = \textsc{Transformer-Aggregation}\left(\mathbf{h}_{t-1,\mathsf{u}}^{(k)}\right) \tag{11}$$

In particular,

$$\mathbf{x}_{t,\mathsf{u}}^{(k)} = \textsc{LNorm}\left(\mathbf{h}_{t-1,\mathsf{u}}^{(k)} + \text{ATT}\left(\mathbf{h}_{t-1,\mathsf{u}}^{(k)}\right)\right) \tag{12}$$

then,

$$\mathbf{h}_{t,\mathsf{u}}^{(k)} = \textsc{LNorm}\left(\mathbf{x}_{t,\mathsf{u}}^{(k)} + \textsc{Trans}\left(\mathbf{x}_{t,\mathsf{u}}^{(k)}\right)\right) \tag{13}$$

where $\mathbf{h}_{t,\mathsf{u}}^{(k)} \in \mathbb{R}^d$; $\textsc{Trans}(.)$ and $\text{ATT}(.)$ denote a MLP network (i.e., two fully-connected layers) and a self-attention network respectively:

$$\textsc{Trans}\left(\mathbf{x}_{t,\mathsf{u}}^{(k)}\right) = \mathbf{W}_2^{(k)}\mathsf{ReLU}\left(\mathbf{W}_1^{(k)}\mathbf{x}_{t,\mathsf{u}}^{(k)} + \mathbf{b}_1^{(k)}\right) + \mathbf{b}_2^{(k)} \tag{14}$$

where $\mathbf{W}_1^{(k)} \in \mathbb{R}^{s \times d}$ and $\mathbf{W}_2^{(k)} \in \mathbb{R}^{d \times s}$ are weight matrices, and $\mathbf{b}_1^{(k)}$ and $\mathbf{b}_2^{(k)}$ are bias parameters, and:

$$\text{ATT}\left(\mathbf{h}_{t-1,\mathsf{u}}^{(k)}\right) = \sum_{\mathsf{u}' \in \mathcal{N}_{\mathsf{v}} \cup \{\mathsf{v}\}} \alpha_{\mathsf{u},\mathsf{u}'}^{(k)}\left(\boldsymbol{V}^{(k)}\mathbf{h}_{t-1,\mathsf{u}'}^{(k)}\right) \tag{15}$$

where $\boldsymbol{V}^{(k)} \in \mathbb{R}^{d \times d}$ is a value-projection weight matrix; $\alpha_{\mathsf{u},\mathsf{u}'}$ is an attention weight, which is computed using the $\mathsf{softmax}$ function over scaled dot products between nodes u and u':

$$\alpha_{\mathsf{u},\mathsf{u}'}^{(k)} = \mathsf{softmax}\left(\frac{\left(\boldsymbol{Q}^{(k)}\mathbf{h}_{t-1,\mathsf{u}}^{(k)}\right)^{\mathsf{T}}\left(\boldsymbol{K}^{(k)}\mathbf{h}_{t-1,\mathsf{u}'}^{(k)}\right)}{\sqrt{d}}\right) \tag{16}$$

where $\boldsymbol{Q}^{(k)} \in \mathbb{R}^{d \times d}$ and $\boldsymbol{K}^{(k)} \in \mathbb{R}^{d \times d}$ are query-projection and key-projection matrices, respectively.

After T steps, we feed $\mathbf{h}_{T,\mathsf{v}}^{(k)} \in \mathbb{R}^d$ to the next $(k+1)$-th layer as:

$$\mathbf{h}_{0,\mathsf{v}}^{(k+1)} = \mathbf{h}_{T,\mathsf{v}}^{(k)}, \forall \mathsf{v} \in \mathcal{V} \tag{17}$$

Note that $\mathbf{h}_{0,\mathsf{v}}^{(0)} = \mathbf{h}_{\mathsf{v}}^{(0)} \in \mathbb{R}^d$ is the feature vector of node v.

We apply the vector concatenation across the layers to obtain the vector representations $\mathbf{e}_{\mathsf{v}}$ of nodes v following Equation 9 as:

$$\mathbf{e}_{\mathsf{v}} = \left[\mathbf{h}_{0,\mathsf{v}}^{(1)}; \mathbf{h}_{0,\mathsf{v}}^{(2)}; ...; \mathbf{h}_{0,\mathsf{v}}^{(K)}\right], \forall \mathsf{v} \in \mathcal{V} \tag{18}$$

where K is the number of layers. We use $\mathbf{e}_{\mathsf{v}}$ as the final embedding of node $\mathsf{v} \in \mathcal{V}$ and then sum all the final embeddings of nodes in $\mathcal{G}$ to get the final embedding

Algorithm 1: The U2GNN learning process.

Input: $\mathcal{G} = \left(\mathcal{V}, \mathcal{E}, \{\mathbf{h}_{\mathsf{v}}^{(0)}\}_{\mathsf{v}\in\mathcal{V}}\right)$ with its label y
for $k = 0, 1, ..., K-1$ **do**
  for $\mathsf{v} \in \mathcal{V}$ **do**
    SAMPLE $\mathcal{N}_{\mathsf{v}}$ for v
    for $t = 1, 2, ..., T$ **do**
      $\forall \mathsf{u} \in \mathcal{N}_{\mathsf{v}} \cup \{\mathsf{v}\}$
      $\mathbf{x}_{t,\mathsf{u}}^{(k)} \leftarrow \text{LNORM}\left(\mathbf{h}_{t-1,\mathsf{u}}^{(k)} + \text{ATT}\left(\mathbf{h}_{t-1,\mathsf{u}}^{(k)}\right)\right)$
      $\mathbf{h}_{t,\mathsf{u}}^{(k)} \leftarrow \text{LNORM}\left(\mathbf{x}_{t,\mathsf{u}}^{(k)} + \text{TRANS}\left(\mathbf{x}_{t,\mathsf{u}}^{(k)}\right)\right)$
    $\mathbf{h}_{0,\mathsf{v}}^{(k+1)} \leftarrow \mathbf{h}_{T,\mathsf{v}}^{(k)} \in \mathbb{R}^d$
$\mathbf{e}_{\mathsf{v}} \leftarrow \left[\mathbf{h}_{0,\mathsf{v}}^{(1)}; \mathbf{h}_{0,\mathsf{v}}^{(2)}; ...; \mathbf{h}_{0,\mathsf{v}}^{(K)}\right], \forall \mathsf{v} \in \mathcal{V}$ (w.r.t Equation 18)
$\mathbf{e}_{\mathcal{G}} \leftarrow \sum_{\mathsf{v}\in\mathcal{V}} \mathbf{e}_{\mathsf{v}}$
$\mathsf{y} \leftarrow \mathsf{softmax}\left(\mathbf{W}\mathbf{e}_{\mathcal{G}} + \mathbf{b}\right)$

$\mathbf{e}_{\mathcal{G}}$ of the entire graph $\mathcal{G}$. We feed $\mathbf{e}_{\mathcal{G}}$ to a single fully-connected layer followed by a $\mathsf{softmax}$ layer to predict the graph label as:

$$\hat{\mathbf{y}}_{\mathcal{G}} = \mathsf{softmax}\left(\mathbf{W}\mathbf{e}_{\mathcal{G}} + \mathbf{b}\right) \tag{19}$$

Finally, we learn the model parameters by minimizing the cross-entropy loss function. We briefly present the supervised learning process of our U2GNN in Algorithm 1.

3.1 Discussion

We discuss some findings in our proposed U2GNN as follows:

– If we set T to 1, $\alpha_{\mathsf{u},\mathsf{u}'}^{(k)}$ to 1, $\boldsymbol{V}^{(k)}$ to the identity matrix in Equation 15, and do not use both the residual connections and the layer normalization, we simplify our U2GNN aggregation function (from Equations 17 and 13) as:

$$\begin{aligned}
\mathbf{h}_{1,\mathsf{v}}^{(k+1)} &= \text{TRANS}\left(\text{ATT}\left(\mathbf{h}_{0,\mathsf{v}}^{(k+1)}\right)\right) \\
&= \text{TRANS}\left(\sum_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}\cup\{\mathsf{v}\}} \mathbf{h}_{0,\mathsf{u}}^{(k+1)}\right) \\
&= \text{TRANS}\left(\sum_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}\cup\{\mathsf{v}\}} \mathbf{h}_{1,\mathsf{u}}^{(k)}\right)
\end{aligned} \tag{20}$$

where TRANS(.) denotes the MLP network of two fully-connected layers (as defined in Equation 14). Thus, this implies that our U2GNN can be simplified (w.r.t Equation 20) to be equivalent to Graph Isomorphism Network

(GIN-0) [50] (w.r.t Equation 8) – one of the recent state-of-the-art GNNs. Experimental results presented in Section 5.3 show that U2GNN outperforms GIN-0 on benchmark datasets for the graph classification task.
– We would probably construct a complex architecture using a complicated graph-level pooling such as hierarchical pooling [5] followed by multiple fully-connected layers [6,30] to predict the graph labels. However, we refrained from doing that. Our key purpose is to introduce a single, unified, and effective model that can work well and produce competitive performances on the benchmark datasets. For that purpose, we followed [50] to use the simple sum pooling followed by a single fully-connected layer in our U2GNN.
– As established empirically, our results shown in Section 5.3 imply that the U2GNN self-attention-based aggregation function is a powerful computation process compared to other existing functions.

4 Experimental datasets

Table 1: Statistics of the experimental benchmark datasets. **#G** denotes the numbers of graphs. **#Cls** denotes the number of class labels. **Avg#N** denotes the average number of nodes per graph. **Avg#E** denotes the average number of neighbors per node. d is the dimension of feature vectors. Note that d is also equal to the node embedding size at each U2GNN layer.

Dataset	**#G**	**#Cls**	**Avg#N**	**Avg#E**	d
COLLAB	5,000	3	74.5	65.9	–
IMDB-M	1,500	3	13.0	10.1	–
IMDB-B	1,000	2	19.8	9.8	–
DD	1,178	2	284.3	5.0	82
PROTEINS	1,113	2	39.1	3.7	3
PTC	344	2	25.6	2.0	19
MUTAG	188	2	17.9	2.2	7

We use seven well-known datasets consisting of three social network datasets (COLLAB, IMDB-B, and IMDB-M) and four bioinformatics datasets (DD, MUTAG, PROTEINS, and PTC). The social network datasets do not have available node features; thus, we follow [33,57] to use node degrees as features.

– **Social networks datasets:** COLLAB is a scientific dataset, where each graph represents a collaboration network of a corresponding researcher with other researchers from each of 3 physics fields; each graph is labeled to a physics field that the researcher belongs to. IMDB-B and IMDB-M are movie

collaboration datasets, where each graph is derived from actor/actress and genre information of different movies on IMDB; nodes correspond to actors/actresses, and each edge represents a co-appearance of two actors/actresses in the same movie; each graph is assigned to a genre.
– **Bioinformatics datasets:** DD [9] is a collection of 1,178 protein network structures with 82 discrete node labels, where each graph is classified into enzyme or non-enzyme class. MUTAG [7] is a collection of 188 nitro compound networks with 7 discrete node labels, where classes indicate a mutagenic effect on a bacterium. PROTEINS comprises 1,113 graphs obtained from [4] to present secondary structure elements (SSEs). PTC [41] consists of 344 chemical compound networks with 19 discrete node labels where classes show carcinogenicity for male and female rats.

Table 1 reports the statistics of these datasets.

5 Experimental results

5.1 Training protocol

We vary the number K of U2GNN layers in {1, 2, 3}, the number of steps T in {1, 2, 3, 4}, the number of neighbors ($|\mathcal{N}_v| = N$) sampled for each node in {4, 8, 16}, and the dimension s of Trans(.) (in Equation 14) in {128, 256, 512, 1024} (in Equation 14). We set the batch size to 4. We apply the Adam optimizer [23] to train our U2GNN and select the Adam initial learning rate $lr \in \{5e^{-5}, 1e^{-4}, 5e^{-4}, 1e^{-3}\}$. We run up to 50 epochs to evaluate our U2GNN.

5.2 Evaluation protocol

We follow [50,49,29,37,6] 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 U2GNN with up-to-date strong baselines. We report the baseline results taken from the original papers or published in [20,44,49,11,6,37,50].

5.3 Main results

Table 2 presents the experimental results of U2GNN and other strong baseline models for the benchmark datasets. In general, our U2GNN gains competitive accuracies on the social network datasets. Especially, U2GNN produces state-of-the-art accuracies of 77.04% and 53.60% on IMDB-B and IMDB-M respectively, which outperform those of other existing models.

On the bioinformatics datasets, U2GNN obtains the highest accuracies of 80.23%, 78.53%, and 69.63% on DD, PROTEINS, and PTC, respectively. Moreover, U2GNN achieves a competitive accuracy compared with those of the baseline models on MUTAG. Additionally, there are no significant differences between our U2GNN and the baselines on MUTAG as this dataset only consists of 188 graphs, which explains the high variance in the results.

Table 2: Graph classification results (% accuracy). The best scores are in **bold**.

Model	COLLAB	IMDB-B	IMDB-M	DD	PROTEINS	MUTAG	PTC
GK [39]	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 [38]	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
PSCN [33]	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 [24]	79.00 ± 1.80	74.00 ± 3.40	51.90 ± 3.80	–	76.00 ± 3.20	85.60 ± 5.80	64.20 ± 4.30
GFN [6]	**81.50 ± 2.42**	73.00 ± 4.35	51.80 ± 5.16	78.78 ± 3.49	76.46 ± 4.06	90.84 ± 7.22	–
GraphSAGE [16]	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 [43]	75.80 ± 1.60	70.50 ± 2.30	47.80 ± 3.10	–	74.70 ± 2.20	89.40 ± 6.10	66.70 ± 5.10
DGCNN [57]	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
SAGPool [27]	–	–	–	76.45 ± 0.97	71.86 ± 0.97	–	–
PPGN [29]	81.38 ± 1.42	73.00 ± 5.77	50.46 ± 3.59	–	77.20 ± 4.73	90.55 ± 8.70	66.17 ± 6.54
CapsGNN [49]	79.62 ± 0.91	73.10 ± 4.83	50.27 ± 2.65	75.38 ± 4.17	76.28 ± 3.63	86.67 ± 6.88	–
DSGC [37]	79.20 ± 1.60	73.20 ± 4.90	48.50 ± 4.80	77.40 ± 6.40	74.20 ± 3.80	86.70 ± 7.60	–
GCAPS [44]	77.71 ± 2.51	71.69 ± 3.40	48.50 ± 4.10	77.62 ± 4.99	76.40 ± 4.17	–	66.01 ± 5.91
IEGN [30]	77.92 ± 1.70	71.27 ± 4.50	48.55 ± 3.90	–	75.19 ± 4.30	84.61 ± 10.0	59.47 ± 7.30
GIN-0 [50]	80.20 ± 1.90	75.10 ± 5.10	52.30 ± 2.80	–	76.20 ± 2.80	89.40 ± 5.60	64.60 ± 7.00
U2GNN	77.84 ± 1.48	**77.04 ± 3.45**	**53.60 ± 3.53**	**80.23 ± 1.48**	**78.53 ± 4.07**	89.97 ± 3.65	**69.63 ± 3.60**

It is worth noting that the superior performance of our method over up-to-date baseline models such as GIN-0 indicates that the self-attention-based aggregation function in U2GNN is an advanced computation process to improve the classification performance of supervised GNNs.

Visualization. To qualitatively demonstrate the effectiveness of our U2GNN, we use t-SNE [28] to visualize the node embeddings learned by GIN-0 and our U2GNN on PTC where the node labels are available. Compared to GIN-0, Figure 2 shows that our U2GNN produces a higher quality of learned node embeddings wherein the nodes are well-clustered according to the node labels.

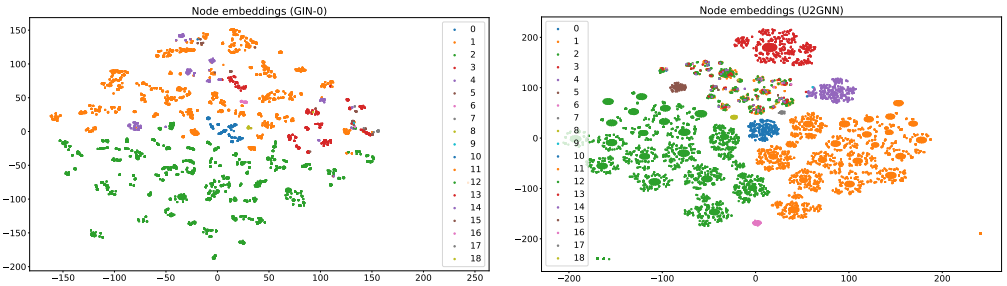

Fig. 2: A t-SNE visualization of the node embeddings learned by GIN-0 and our U2GNN on the PTC dataset.

6 Conclusion

We introduce an effective graph neural network model, named U2GNN, one of the firsts to explore transformer to graphs. In particular, U2GNN develops an advanced aggregation function leveraging the transformer self-attention network to improve the graph classification performance. We evaluate our U2GNN using the same data splits, the same 10-fold cross-validation scheme, and the standard evaluation settings on the well-known benchmark datasets. Experimental results demonstrate that U2GNN outperforms up-to-date models and produces state-of-the-art accuracies on these datasets.[1]

References

1. Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. arXiv:1607.06450 (2016)
2. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. International Conference on Learning Representations (ICLR) (2015)
3. Borgwardt, K.M., Kriegel, H.P.: Shortest-Path Kernels on Graphs. In: ICDM. pp. 74–81 (2005)
4. Borgwardt, K.M., Ong, C.S., Schönauer, S., Vishwanathan, S., Smola, A.J., Kriegel, H.P.: Protein function prediction via graph kernels. Bioinformatics **21**(suppl_1), i47–i56 (2005)
5. Cangea, C., Veličković, P., Jovanović, N., Kipf, T., Liò, P.: Towards sparse hierarchical graph classifiers. arXiv preprint arXiv:1811.01287 (2018)
6. Chen, T., Bian, S., Sun, Y.: Are powerful graph neural nets necessary? a dissection on graph classification. arXiv:1905.04579 (2019)
7. Debnath, A.K., Lopez de Compadre, R.L., Debnath, G., Shusterman, A.J., Hansch, C.: Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. Journal of medicinal chemistry **34**(2), 786–797 (1991)
8. Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J., Kaiser, Ł.: Universal Transformers. International Conference on Learning Representations (ICLR) (2019)
9. Dobson, P.D., Doig, A.J.: Distinguishing enzyme structures from non-enzymes without alignments. Journal of molecular biology **330**(4), 771–783 (2003)
10. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: LIBLINEAR: A Library for Large Linear Classification. Journal of Machine Learning Research **9**, 1871–1874 (2008)
11. Fan, X., Gong, M., Xie, Y., Jiang, F., Li, H.: Structured self-attention architecture for graph-level representation learning. Pattern Recognition **100** (2019)
12. Gao, H., Ji, S.: Graph u-nets. In: International Conference on Machine Learning. pp. 2083–2092 (2019)
13. Gärtner, T., Flach, P., Wrobel, S.: On graph kernels: Hardness results and efficient alternatives. In: Learning Theory and Kernel Machines. pp. 129–143 (2003)

[1] We propose a new unsupervised learning in Appendix A to train GNNs and hope that future GNN works can consider the unsupervised learning beside the supervised one.

14. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural Message Passing for Quantum Chemistry. In: International Conference on Machine Learning. pp. 1263–1272 (2017)
15. Grover, A., Leskovec, J.: Node2Vec: Scalable Feature Learning for Networks. In: The ACM SIGKDD Conference on Knowledge Discovery and Data Mining. pp. 855–864 (2016)
16. Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in Neural Information Processing Systems. pp. 1024–1034 (2017)
17. Hamilton, W.L., Ying, R., Leskovec, J.: Representation learning on graphs: Methods and applications. arXiv:1709.05584 (2017)
18. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778 (2016)
19. Hofmann, T., Schölkopf, B., Smola, A.J.: Kernel methods in machine learning. The annals of statistics pp. 1171–1220 (2008)
20. Ivanov, S., Burnaev, E.: Anonymous walk embeddings. In: International Conference on Machine Learning. pp. 2191–2200 (2018)
21. Jean, S., Cho, K., Memisevic, R., Bengio, Y.: On using very large target vocabulary for neural machine translation. In: ACL. pp. 1–10 (2015)
22. Kashima, H., Tsuda, K., Inokuchi, A.: Marginalized kernels between labeled graphs. In: International Conference on Machine Learning. pp. 321–328 (2003)
23. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. International Conference on Learning Representations (ICLR) (2015)
24. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (ICLR) (2017)
25. Kriege, N.M., Johansson, F.D., Morris, C.: A survey on graph kernels. arXiv:1903.11835 (2019)
26. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: International Conference on Machine Learning. pp. 1188–1196 (2014)
27. Lee, J., Lee, I., Kang, J.: Self-attention graph pooling. In: International Conference on Machine Learning (2019)
28. Maaten, L.v.d., Hinton, G.: Visualizing data using t-sne. Journal of machine learning research **9**, 2579–2605 (2008)
29. Maron, H., Ben-Hamu, H., Serviansky, H., Lipman, Y.: Provably powerful graph networks. In: Advances in Neural Information Processing Systems. pp. 2153–2164 (2019)
30. Maron, H., Ben-Hamu, H., Shamir, N., Lipman, Y.: Invariant and equivariant graph networks. International Conference on Learning Representations (ICLR) (2019)
31. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems. pp. 3111–3119 (2013)
32. Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y., Jaiswal, S.: graph2vec: Learning distributed representations of graphs. arXiv:1707.05005 (2017)
33. Niepert, M., Ahmed, M., Kutzkov, K.: Learning Convolutional Neural Networks for Graphs. In: International Conference on Machine Learning. pp. 2014–2023 (2016)
34. Nikolentzos, G., Siglidis, G., Vazirgiannis, M.: Graph kernels: A survey. arXiv:1904.12218 (2019)

35. Perozzi, B., Al-Rfou, R., Skiena, S.: DeepWalk: Online Learning of Social Representations. In: The ACM SIGKDD Conference on Knowledge Discovery and Data Mining. pp. 701–710 (2014)
36. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Transactions on Neural Networks **20**(1), 61–80 (2009)
37. Seo, Y., Loukas, A., Peraudin, N.: Discriminative structural graph classification. arXiv:1905.13422 (2019)
38. Shervashidze, N., Schweitzer, P., Leeuwen, E.J.v., Mehlhorn, K., Borgwardt, K.M.: Weisfeiler-lehman graph kernels. Journal of Machine Learning Research **12**(Sep), 2539–2561 (2011)
39. Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K., Borgwardt, K.: Efficient Graphlet Kernels for Large Graph Comparison. In: AISTATS. pp. 488–495 (2009)
40. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: LINE: Large-scale Information Network Embedding. In: WWW. pp. 1067–1077 (2015)
41. Toivonen, H., Srinivasan, A., King, R.D., Kramer, S., Helma, C.: Statistical evaluation of the predictive toxicology challenge 2000–2001. Bioinformatics **19**(10), 1183–1193 (2003)
42. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems. pp. 5998–6008 (2017)
43. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph Attention Networks. International Conference on Learning Representations (ICLR) (2018)
44. Verma, S., Zhang, Z.L.: Graph capsule convolutional neural networks. The Joint ICML and IJCAI Workshop on Computational Biology (2018)
45. Vishwanathan, S.V.N., Schraudolph, N.N., Kondor, R., Borgwardt, K.M.: Graph kernels. Journal of Machine Learning Research **11**(Apr), 1201–1242 (2010)
46. Wang, X., Ji, H., Shi, C., Wang, B., Ye, Y., Cui, P., Yu, P.S.: Heterogeneous graph attention network. In: The World Wide Web Conference. pp. 2022–2032 (2019)
47. Wu, F., Souza, A., Zhang, T., Fifty, C., Yu, T., Weinberger, K.: Simplifying graph convolutional networks. In: International Conference on Machine Learning. pp. 6861–6871 (2019)
48. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. arXiv:1901.00596 (2019)
49. Xinyi, Z., Chen, L.: Capsule Graph Neural Network. International Conference on Learning Representations (ICLR) (2019)
50. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How Powerful Are Graph Neural Networks? International Conference on Learning Representations (ICLR) (2019)
51. Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.i., Jegelka, S.: Representation learning on graphs with jumping knowledge networks. In: International Conference on Machine Learning. pp. 5453–5462 (2018)
52. Yanardag, P., Vishwanathan, S.: Deep graph kernels. In: The ACM SIGKDD Conference on Knowledge Discovery and Data Mining. pp. 1365–1374 (2015)
53. Ying, R., You, J., Morris, C., Ren, X., Hamilton, W.L., Leskovec, J.: Hierarchical graph representation learning with differentiable pooling. In: NeurIPS. pp. 4805–4815 (2018)
54. Yun, S., Jeong, M., Kim, R., Kang, J., Kim, H.J.: Graph transformer networks. In: Advances in Neural Information Processing Systems. pp. 11960–11970 (2019)
55. Zhang, D., Yin, J., Zhu, X., Zhang, C.: Network representation learning: A survey. IEEE Transactions on Big Data **6**, 3–28 (2020)

56. Zhang, J., Zhang, H., Xia, C., Sun, L.: Graph-bert: Only attention is needed for learning graph representations. arXiv preprint arXiv:2001.05140 (2020)
57. Zhang, M., Cui, Z., Neumann, M., Chen, Y.: An End-to-End Deep Learning Architecture for Graph Classification. In: AAAI (2018)
58. Zhang, R., Zou, Y., Ma, J.: Hyper-SAGNN: a self-attention based graph neural network for hypergraphs. In: International Conference on Learning Representations (ICLR) (2020)
59. Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., Sun, M.: Graph neural networks: A review of methods and applications. arXiv:1812.08434 (2018)

A Unsupervised Graph Neural Networks

The unsupervised learning is essential in both industry and academic applications, where expanding unsupervised GNN models is more suitable due to the limited availability of class labels. Therefore, we introduce a new unsupervised learning to train GNNs for the graph classification task.

A.1 Learning process

Most of the recent approaches have focused on the supervised learning where they use the graph labels during the training process [49,50,6,30,37]. In a situation where no graph labels are available during training, some works (such as DGK [52], Graph2Vec [32], and AWE [20]) have considered the unsupervised learning, where *they can have access to all nodes from the entire dataset (i.e., additionally using all nodes in the test set during training)*. But they produce lower classification accuracies compared to the supervised approaches.

Algorithm 2: The unsupervised learning.

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \{\mathbf{h}_v\}_{v \in \mathcal{V}})$
for $k = 0, ..., K-1$ **do**
  for $v \in \mathcal{V}$ **do**
    $\mathbf{h}_v^{(k+1)} = \textsc{Aggregation}\left(\left\{\mathbf{h}_u^{(k)}\right\}_{u \in \mathcal{N}_v \cup \{v\}}\right)$
$\mathbf{e}_v \leftarrow \left[\mathbf{h}_v^{(1)}; \mathbf{h}_v^{(2)}; ...; \mathbf{h}_v^{(K)}\right], \forall v \in \mathcal{V}$ (with respect to Equation 9)
$\mathbf{o}_v \leftarrow \mathbf{e}_v$ (with respect to Equation 21)
$\mathbf{e}_{\mathcal{G}} \leftarrow \sum_{v \in \mathcal{V}} \mathbf{o}_v$

To this end, we propose a new unsupervised learning to train GNNs for the graph classification task. We can see $\mathbf{e}_v$ in Equation 9 as a vector representation encoded for the substructure around node v. The goal of our unsupervised learning is to guide GNNs to recognize and distinguish the sub-graph structural information within each graph, leading to improve the classification accuracies

of unsupervised models. To achieve this goal, we consider a final embedding $\mathbf{o}_{\mathsf{v}}$ for each node v, and make the similarity between $\mathbf{e}_{\mathsf{v}}$ and $\mathbf{o}_{\mathsf{v}}$ higher than that between $\mathbf{e}_{\mathsf{v}}$ and the final embeddings of the other nodes, by minimizing the sampled $\mathsf{softmax}$ loss function [21] applied to node v as:

$$\mathcal{L}_{\mathsf{U2GNN}}(\mathsf{v}) = -\log \frac{\exp(\mathbf{o}_{\mathsf{v}}^{\mathsf{T}}\mathbf{e}_{\mathsf{v}})}{\sum_{\mathsf{v}' \in \mathcal{V}'} \exp(\mathbf{o}_{\mathsf{v}'}^{\mathsf{T}}\mathbf{e}_{\mathsf{v}})} \tag{21}$$

where $\mathcal{V}'$ is a subset sampled from $\{\cup \mathcal{V}_m\}_{m=1}^{M}$. Node embeddings $\mathbf{o}_{\mathsf{v}}$ are learned implicitly as model parameters. After that, we sum all the final embeddings $\mathbf{o}_{\mathsf{v}}$ of nodes v in $\mathcal{G}$ to obtain the graph embedding $\mathbf{e}_{\mathcal{G}}$. We then use the logistic regression classifier [10] with setting the termination criterion to 0.001 to evaluate our model. We outline the general process of our unsupervised learning in Algorithm 2.

A.2 Training protocol

We follow some unsupervised approaches such as DGK [52] and AWE [20] to train our unsupervised U2GNN on all nodes from the entire dataset (i.e., consisting of all nodes from the test set during training) for a fair comparison. The hyper-parameters are varied as same as in Section 5.1.

We also train our GCN baseline following our unsupervised learning. We set the batch size to 4 and vary the number of GCN layers in {1, 2, 3} and the hidden layer size in {32, 64, 128, 256}. We also use the Adam optimizer [23] to train this unsupervised GCN up to 50 epochs.

A.3 Evaluation protocol

We utilizes the logistic regression classifier [10] with using the 10-fold cross-validation scheme to evaluate our models. We compare our unsupervised GCN (denoted as uGCN) and U2GNN with the baselines: Deep Graph Kernel (DGK) [52] and Anonymous Walk Embedding (AWE) [20].

A.4 Experimental results

Table 3 presents the experimental results in the unsupervised learning. We encourage a re-evaluation to examine the existing GNNs from the supervised learning to our new unsupervised learning to see negative results. For example, regarding the supervised learning, as shown in Table 2, our supervised U2GNN outperforms GCN on the bioinformatics datasets. However, regarding the unsupervised learning, as shown in Table 3, our uGCN works better than our unsupervised U2GNN on PROTEINS, MUTAG, and PTC. These three datasets are much sparse, and node neighbors have a similar effect on each other. Hence, without using the graph labels during training, U2GNN does not increase the similar effects on node neighbors, leading to be outperformed by our uGCN on these datasets.

Table 3: Graph classification results (% accuracy) in the unsupervised learning. The best scores are in bold. uGCN denotes our unsupervised GCN. Note that we do not make any direct comparison between the *unsupervised* approaches and the *supervised* ones because of the difference in the training data.

Model	COLLAB	IMDB-B	IMDB-M	DD	PROTEINS	MUTAG	PTC
DGK [52]	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 [20]	73.93 ± 1.94	74.45 ± 5.83	51.54 ± 3.61	71.51 ± 4.02	–	87.87 ± 9.76	–
uGCN	93.28 ± 0.99	94.50 ± 2.79	81.66 ± 3.16	94.31 ± 1.71	**89.09 ± 3.25**	**95.36 ± 2.64**	**92.67 ± 4.60**
U2GNN	**95.62 ± 0.92**	**96.41 ± 1.94**	**89.20 ± 2.52**	**95.67 ± 1.89**	80.01 ± 3.21	88.47 ± 7.13	91.81 ± 6.61

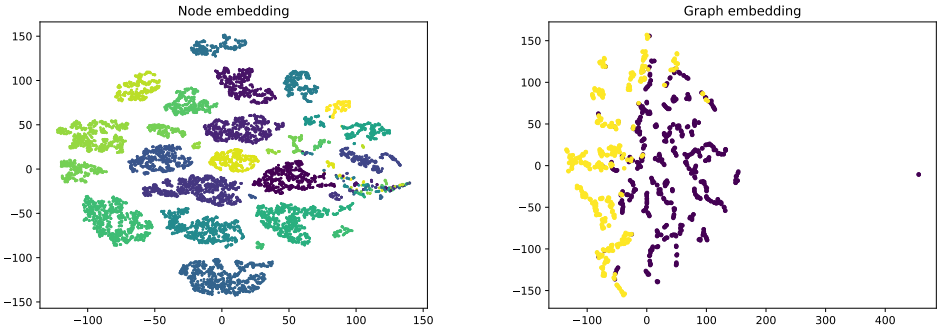

Fig. 3: A visualization of the node and graph embeddings learned by our unsupervised U2GNN on the DD dataset.

In general, both our unsupervised U2GNN and uGCN obtain the state-of-the-art accuracies on the benchmark datasets. The significant gains demonstrate a notable impact of our unsupervised learning. It aims to guide GNNs to identify the sub-graph structural information for every node; hence, the models can memorize the structural differences among graphs to produce the plausible node and graph embeddings as visualized in Figure 3, leading to improve the unsupervised performance. We hope that future GNN works can consider the unsupervised learning beside the supervised one.

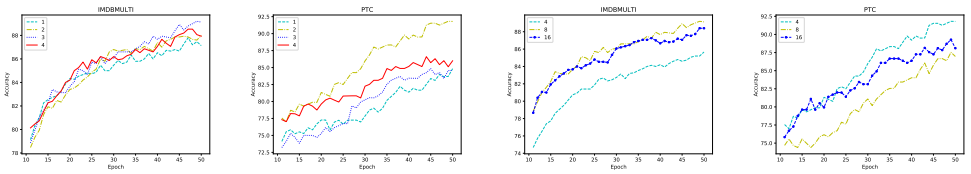

Fig. 4: Effects of the number of timesteps (T) and the number of neighbors sampled for each node ($N = |\mathcal{N}_v|$) in the unsupervised learning.

Hyper-parameter analysis. We investigate the effects of the number of timesteps (T) and the number of neighbors sampled for each node ($N = |\mathcal{N}_{\mathrm{v}}|$) in Figure 4. We see similar findings for both the supervised and unsupervised training settings. In general, we find that higher T can help on most of the datasets as we may use more steps T to encode the graph structures better. Furthermore, the social network datasets are denser than the bioinformatics ones; hence we should use more sampled neighbors (i.e., using higher N) on the social network datasets rather than on the bioinformatics ones.

3.2.2 ECML-PKDD 2020 & CIKM 2020 - Learning node embeddings for new nodes

• **Dai Quoc Nguyen**, Tu Dinh Nguyen and Dinh Phung. **2020**. 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)*. https://doi.org/10.1007/978-3-030-67664-3_22

• **Dai Quoc Nguyen**, Tu Dinh Nguyen, Dat Quoc Nguyen and Dinh Phung. **2020**. A Capsule Network-based Model for Learning Node Embeddings. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM 2020)*. https://doi.org/10.1145/3340531.3417455

Contribution. We present SANNE (Nguyen et al., 2020d) – a new unsupervised learning model that adapts a transformer self-attention network (Vaswani et al., 2017) to learn node embeddings. SANNE uses random walks (generated for every node) as inputs for a stack of attention layers. Each attention layer consists of a self-attention sub-layer followed by a feed-forward sub-layer, wherein the self-attention sub-layer is constructed using query, key and value projection matrices to compute pairwise similarities among nodes. SANNE then samples a set of neighbours for each node in the random walk and uses output vector representations from the last attention layer to infer embeddings for these neighbours. As a consequence, our SANNE produces the effective node embeddings for both the transductive and inductive settings. The code is available at: https://github.com/daiquocnguyen/Walk-Transformer.

Inspired by the advanced capsule networks (Sabour et al., 2017), we also propose Caps2NE (Nguyen et al., 2020a) – another new unsupervised embedding model that utilises a capsule network to learn node embeddings. Caps2NE aims to capture k-hops context neighbours to predict a target node from generated random walks. In particular, Caps2NE consists of two capsule layers with connections from the first to the second layer, but no connections within layers. The first layer constructs capsules to encapsulate

context neighbours. Then a routing process is used to aggregate the feature information from capsules in the first layer to only one capsule in the second layer. After that, the second layer produces a continuous vector which is used to infer an embedding for the target node. As a result, our Caps2NE can generate high-level features to infer effective node embeddings for the transductive and inductive settings. The code is available at: `https://github.com/daiquocnguyen/Caps2NE`.

A Self-Attention Network based Node Embedding Model

Dai Quoc Nguyen[1]*, Tu Dinh Nguyen[2], and Dinh Phung[1]

[1]Monash University, Australia
{dai.nguyen,dinh.phung}@monash.edu
[2]nguyendinhtu@gmail.com

Abstract. Despite several signs of progress have been made recently, limited research has been conducted for an inductive setting where embeddings are required for newly unseen nodes – a setting encountered commonly in practical applications of deep learning for graph networks. This significantly affects the performances of downstream tasks such as node classification, link prediction or community extraction. To this end, we propose SANNE – a novel unsupervised embedding model – whose central idea is to employ a transformer self-attention network to iteratively aggregate vector representations of nodes in random walks. Our SANNE aims to produce plausible embeddings not only for present nodes, but also for newly unseen nodes. Experimental results show that the proposed SANNE obtains state-of-the-art results for the node classification task on well-known benchmark datasets.

Keywords: Node Embeddings · Transformer · Self-Attention Network · Node Classification.

1 Introduction

Graph-structured data appears in plenty of fields in our real-world from social networks, citation networks, knowledge graphs and recommender systems to telecommunication networks, biological networks [11, 3–5]. In graph-structured data, nodes represent individual entities, and edges represent relationships and interactions among those entities. For example, in citation networks, each document is treated as a node, and a citation link between two documents is treated as an edge.

Learning node embeddings is one of the most important and active research topics in representation learning for graph-structured data. There have been many models proposed to embed each node into a continuous vector as summarized in [29]. These vectors can be further used in downstream tasks such as node classification, i.e., using learned node embeddings to train a classifier to predict node labels. Existing models mainly focus on the *transductive* setting where

* The final authenticated version is available online at https://doi.org/10.1007/978-3-030-67664-3_22.

a model is trained using the entire input graph, i.e., the model requires all nodes with a fixed graph structure during training and lacks the flexibility in inferring embeddings for unseen/new nodes, e.g., DeepWalk [22], LINE [24], Node2Vec [8], SDNE [27] and GCN [15]. By contrast, a more important setup, but less mentioned, is the *inductive* setting wherein only a part of the input graph is used to train the model, and then the learned model is used to infer embeddings for new nodes [28]. Several attempts have additionally been made for the inductive settings such as EP-B [6], GraphSAGE [10] and GAT [26]. Working on the inductive setting is particularly more difficult than that on the transductive setting due to lacking the ability to generalize to the graph structure for new nodes.

One of the most convenient ways to learn node embeddings is to adopt the idea of a word embedding model by viewing each node as a word and each graph as a text collection of random walks to train a Word2Vec model [19], e.g., DeepWalk, LINE and Node2Vec. Although these Word2Vec-based approaches allow the current node to be directly connected with k-hops neighbors via random walks, they ignore feature information of nodes. Besides, recent research has raised attention in developing graph neural networks (GNNs) for the node classification task, e.g., GNN-based models such as GCN, GraphSAGE and GAT. These GNN-based models iteratively update vector representations of nodes over their k-hops neighbors using multiple layers stacked on top of each other. Thus, it is difficult for the GNN-based models to infer plausible embeddings for new nodes when their k-hops neighbors are also unseen during training.

The transformer self-attention network [25] has been shown to be very powerful in many NLP tasks such as machine translation and language modeling. Inspired by this attention technique, we present SANNE – an unsupervised learning model that adapts a transformer self-attention network to learn node embeddings. SANNE uses random walks (generated for every node) as inputs for a stack of attention layers. Each attention layer consists of a self-attention sub-layer followed by a feed-forward sub-layer, wherein the self-attention sub-layer is constructed using query, key and value projection matrices to compute pairwise similarities among nodes. Hence SANNE allows a current node at each time step to directly attend its k-hops neighbors in the input random walks. SANNE then samples a set of (1-hop) neighbors for each node in the random walk and uses output vector representations from the last attention layer to infer embeddings for these neighbors. As a consequence, our proposed SANNE produces the plausible node embeddings for both the transductive and inductive settings.

In short, our main contributions are as follows:

- Our SANNE induces a transformer self-attention network not only to work in the transductive setting advantageously, but also to infer the plausible embeddings for new nodes in the inductive setting effectively.
- The experimental results show that our unsupervised SANNE obtains better results than up-to-date unsupervised and supervised embedding models on three benchmark datasets Cora, Citeseer and Pubmed for the transductive

and inductive settings. In particular, SANNE achieves relative error reductions of more than 14% over GCN and GAT in the inductive setting.

2 Related work

DeepWalk [22] generates unbiased random walks starting from each node, considers each random walk as a sequence of nodes, and employs Word2Vec [19] to learn node embeddings. Node2Vec [8] extends DeepWalk by introducing a biased random walk strategy that explores diverse neighborhoods and balances between exploration and exploitation from a given node. LINE [24] closely follows Word2Vec, but introduces node importance, for which each node has a different weight to each of its neighbors, wherein weights can be pre-defined through algorithms such as PageRank [21]. DDRW [17] jointly trains a DeepWalk model with a Support Vector Classification [7] in a supervised manner.

SDNE [27], an autoencoder-based supervised model, is proposed to preserve both local and global graph structures. EP-B [6] is introduced to explore the embeddings of node attributes (such as words) with node neighborhoods to infer the embeddings of unseen nodes. Graph Convolutional Network (GCN) [15], a semi-supervised model, utilizes a variant of convolutional neural networks (CNNs) which makes use of layer-wise propagation to aggregate node features (such as profile information and text attributes) from the neighbors of a given node. GraphSAGE [10] extends GCN in using node features and neighborhood structures to generalize to unseen nodes. Another extension of GCN is Graph Attention Network (GAT) [26] that uses a similar idea with LINE [24] in assigning different weights to different neighbors of a given node, but learns these weights by exploring an attention mechanism technique [2]. These GNN-based approaches construct multiple layers stacked on top of each other to indirectly attend k-hops neighbors; thus, it is not straightforward for these approaches to infer the plausible embeddings for new nodes especially when their neighbors are also not present during training.

3 The proposed SANNE

Let us define a graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, in which $\mathcal{V}$ is a set of nodes and $\mathcal{E}$ is a set of edges, i.e., $\mathcal{E} \subseteq \{(\mathsf{u}, \mathsf{v}) | \mathsf{u}, \mathsf{v} \in \mathcal{V}\}$. Each node $\mathsf{v} \in \mathcal{V}$ is associated with a feature vector $\mathbf{x}_\mathsf{v} \in \mathbb{R}^d$ representing node features. In this section, we detail the learning process of our proposed SANNE to learn a node embedding $\mathbf{o}_\mathsf{v}$ for each node $\mathsf{v} \in \mathcal{V}$. We then use the learned node embeddings to classify nodes into classes.

SANNE architecture. Particularly, we follow DeepWalk [22] to uniformly sample random walks of length N for every node in $\mathcal{V}$. For example, Figure 1 shows a graph consisting of 6 nodes where we generate a random walk of length $N = 4$ for node 1, e.g., $\{1, 2, 3, 4\}$; and then this random walk is used as an input for the SANNE learning process.

Given a random walk w of N nodes $\{\mathsf{v}_{w,i}\}_{i=1}^{N}$, we obtain an input sequence of vector representations $\{\mathbf{u}^0_{\mathsf{v}_{w,i}}\}_{i=1}^{N}$: $\mathbf{u}^0_{\mathsf{v}_{w,i}} = \mathbf{x}_{\mathsf{v}_{w,i}}$. We construct a stack of K

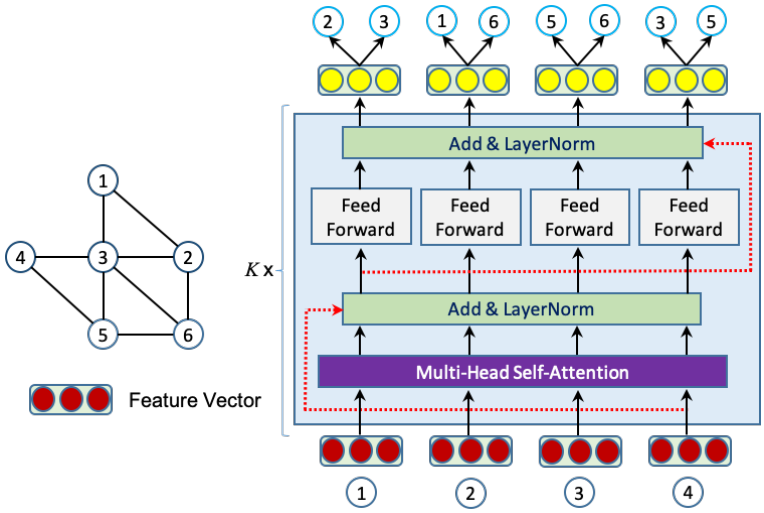

Fig. 1. Illustration of our SANNE learning process with $d = 3$, $N = 4$ and $M = 2$.

attention layers [25], in which each of them has the same structure consisting of a multi-head self-attention sub-layer followed by a feed-forward sub-layer together with additionally using a residual connection [12] and layer normalization [1] around each of these sub-layers. At the k-th attention layer, we take an input sequence $\{\mathbf{u}_{v_{w,i}}^{(k-1)}\}_{i=1}^{N}$ and produce an output sequence $\{\mathbf{u}_{v_{w,1}}^{(k)}\}_{i=1}^{N}$, $\mathbf{u}_{v_{w,1}}^{(k)} \in \mathbb{R}^d$ as:

$$\mathbf{u}_{v_{w,i}}^{(k)} = \textsc{Walk-Transformer}\left(\mathbf{u}_{v_{w,i}}^{(k-1)}\right)$$

$$\text{In particular,} \quad \mathbf{u}_{v_{w,i}}^{(k)} = \textsc{LayerNorm}\left(\mathbf{y}_{v_{w,i}}^{(k)} + \text{FF}\left(\mathbf{y}_{v_{w,i}}^{(k)}\right)\right)$$

$$\text{with} \quad \mathbf{y}_{v_{w,i}}^{(k)} = \textsc{LayerNorm}\left(\mathbf{u}_{v_{w,i}}^{(k-1)} + \textsc{Att}\left(\mathbf{u}_{v_{w,i}}^{(k-1)}\right)\right)$$

where $\text{FF}(.)$ and $\textsc{Att}(.)$ denote a two-layer feed-forward network and a multi-head self-attention network respectively:

$$\text{FF}\left(\mathbf{y}_{v_{w,i}}^{(k)}\right) = \mathbf{W}_2^{(k)}\text{ReLU}\left(\mathbf{W}_1^{(k)}\mathbf{y}_{v_{w,i}}^{(k)} + \mathbf{b}_1^{(k)}\right) + \mathbf{b}_2^{(k)}$$

where $\mathbf{W}_1^{(k)}$ and $\mathbf{W}_2^{(k)}$ are weight matrices, and $\mathbf{b}_1^{(k)}$ and $\mathbf{b}_2^{(k)}$ are bias parameters. And:

$$\textsc{Att}\left(\mathbf{u}_{v_{w,i}}^{(k-1)}\right) = \mathbf{W}^{(k)}\left[\mathbf{h}_{v_{w,i}}^{(k),1}; \mathbf{h}_{v_{w,i}}^{(k),2}; ...; \mathbf{h}_{v_{w,i}}^{(k),H}\right]$$

where $\mathbf{W}^{(k)} \in \mathbb{R}^{d \times Hs}$ is a weight matrix, H is the number of attention heads, and $[;]$ denotes a vector concatenation. Regarding the h-th attention head, $\mathbf{h}_{v_{w,i}}^{(k),h} \in \mathbb{R}^s$ is calculated by a weighted sum as:

$$\mathbf{h}_{v_{w,i}}^{(k),h} = \sum_{j=1}^{N} \alpha_{i,j,h}^{(k)}\left(\mathbf{W}_h^{(k),V}\mathbf{u}_{v_{w,j}}^{(k-1)}\right)$$

where $\mathbf{W}_h^{(k),V} \in \mathbb{R}^{s\times d}$ is a value projection matrix, and $\alpha_{i,j,h}^{(k)}$ is an attention weight. $\alpha_{i,j,h}^{(k)}$ is computed using the $\mathsf{softmax}$ function over scaled dot products between i-th and j-th nodes in the walk w as:

$$\alpha_{i,j,h}^{(k)} = \mathsf{softmax}\left(\frac{\left(\mathbf{W}_h^{(k),Q}\mathbf{u}_{\mathsf{v}_{w,i}}^{(k-1)}\right)^{\mathsf{T}}\left(\mathbf{W}_h^{(k),K}\mathbf{u}_{\mathsf{v}_{w,j}}^{(k-1)}\right)}{\sqrt{k}}\right)$$

where $\mathbf{W}_h^{(k),Q}$ and $\mathbf{W}_h^{(k),K} \in \mathbb{R}^{s\times d}$ are query and key projection matrices respectively.

We randomly sample a fixed-size set of M neighbors for each node in the random walk w. We then use the output vector representations $\mathbf{u}_{\mathsf{v}_{w,i}}^{(K)}$ from the K-th last layer to infer embeddings $\mathbf{o}$ for sampled neighbors of $\mathsf{v}_{w,i}$. Figure 1 illustrates our proposed SANNE where we set the length N of random walks to 4, the dimension size d of feature vectors to 3, and the number M of sampling neighbors to 2. We also sample different sets of neighbors for the same input node at each training step.

Algorithm 1: The SANNE learning process.

Input: A network graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.
for $\mathsf{v} \in \mathcal{V}$ **do**
  SAMPLE T random walks of length N rooted by v.
for *each* random walk w **do**
  for k = *1, 2, ..., K* **do**
    $\forall \mathsf{v} \in w$
    $\mathbf{y}_{\mathsf{v}}^{(k)} \leftarrow \textsc{LayerNorm}\left(\mathbf{u}_{\mathsf{v}}^{(k-1)} + \textsc{Att}\left(\mathbf{u}_{\mathsf{v}}^{(k-1)}\right)\right)$
    $\mathbf{u}_{\mathsf{v}}^{(k)} \leftarrow \textsc{LayerNorm}\left(\mathbf{y}_{\mathsf{v}}^{(k)} + \text{FF}\left(\mathbf{y}_{\mathsf{v}}^{(k)}\right)\right)$
  for $\mathsf{v} \in w$ **do**
    SAMPLE a set C_{v} of M neighbors of node v.
    $\mathbf{o}_{\mathsf{v}'} \leftarrow \mathbf{u}_{\mathsf{v}}^{(K)}, \forall \mathsf{v}' \in \mathsf{C}_{\mathsf{v}}$

Training SANNE: We learn our model's parameters including the weight matrices and node embeddings by minimizing the sampled softmax loss function [13] applied to the random walk w as:

$$\mathcal{L}_{\mathsf{SANNE}}(w) = -\sum_{i=1}^{N}\sum_{\mathsf{v}'\in\mathsf{C}_{\mathsf{v}_{w,i}}} \log\frac{\exp(\mathbf{o}_{\mathsf{v}'}^{\mathsf{T}}\mathbf{u}_{\mathsf{v}_{w,i}}^{(K)})}{\sum_{\mathsf{u}\in\mathcal{V}'}\exp(\mathbf{o}_{\mathsf{u}}^{\mathsf{T}}\mathbf{u}_{\mathsf{v}_{w,i}}^{(K)})}$$

where C_{v} is the fixed-size set of M neighbors randomly sampled for node v, $\mathcal{V}'$ is a subset sampled from $\mathcal{V}$, and $\mathbf{o}_{\mathsf{v}} \in \mathbb{R}^d$ is the node embedding of node $\mathsf{v}, \forall \mathsf{v} \in \mathcal{V}$. Node embeddings $\mathbf{o}_{\mathsf{v}}$ are learned implicitly as model parameters.

We briefly describe the learning process of our proposed SANNE model in Algorithm 1. Here, the learned node embeddings $\mathbf{o}_{\mathsf{v}}$ are used as the final representations of nodes $\mathsf{v} \in \mathcal{V}$. We explicitly aggregate node representations from both left-to-right and right-to-left sides in the walk for each node in predicting its neighbors. This allows SANNE to infer the plausible node embeddings even in the inductive setting.

Algorithm 2: The embedding inference for new nodes.

Input: A network graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a trained model $\text{SANNE}_{trained}$ for $\mathcal{G}$, a set $\mathcal{V}_{new}$ of new nodes.
for $v \in \mathcal{V}_{new}$ **do**
  SAMPLE Z random walks $\{w_i\}_{i=1}^{Z}$ of length N rooted by v.
  for $i \in \{1,\ 2,\ ...,\ Z\}$ **do**
    $\mathbf{u}_{v,i}^{(K)} \leftarrow \text{SANNE}_{trained}\left(w_i\right)[0]$
  $\mathbf{o}_v \leftarrow \text{AVERAGE}\left(\{\mathbf{u}_{v,i}^{(K)}\}_{i=1}^{Z}\right)$

Inferring embeddings for new nodes in the inductive setting: After training our SANNE on a given graph, we show in Algorithm 2 our method to infer an embedding for a new node v adding to this given graph. We randomly sample Z random walks of length N starting from v. We use each of these walks as an input for our trained model and then collect the first vector representation (at the index 0 corresponding to node v) from the output sequence at the K-th last layer. Thus, we obtain Z vectors and then average them into a final embedding for the new node v.

4 Experiments

Our SANNE is evaluated for the node classification task as follows: (i) We train our model to obtain node embeddings. (ii) We use these node embeddings to learn a logistic regression classifier to predict node labels. (iii) We evaluate the classification performance on benchmark datasets and then analyze the effects of hyper-parameters.

4.1 Datasets and data splits

Datasets We use three well-known benchmark datasets CORA, CITESEER [23] and PUBMED [20] which are citation networks. For each dataset, each node represents a document, and each edge represents a citation link between two documents. Each node is assigned a class label representing the main topic of the document. Besides, each node is also associated with a feature vector of a bag-of-words. Table 1 reports the statistics of these three datasets.

Table 1. Statistics of the experimental datasets. |Vocab.| denotes the vocabulary size. Avg.W denotes the average number of words per node.

Dataset	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	#Classes	|Vocab.|	Avg.W
CORA	2,708	5,429	7	1,433	18
CITESEER	3,327	4,732	6	3,703	31
PUBMED	19,717	44,338	3	500	50

Data splits We follow the same settings used in [6] for a fair comparison. For each dataset, we uniformly sample 20 random nodes for each class as training data, 1000 different random nodes as a validation set, and 1000 different random nodes as a test set. We repeat 10 times to have 10 training sets, 10 validation sets, and 10 test sets respectively, and finally report the mean and standard deviation of the accuracy results over 10 data splits.

4.2 Training protocol

Feature vectors initialized by Doc2Vec For each dataset, each node represents a document associated with an existing feature vector of a bag-of-words. Thus, we train a PV-DBOW Doc2Vec model [16] to produce new 128-dimensional embeddings $\mathbf{x}_\mathsf{v}$ which are considered as new feature vectors for nodes v. Using this initialization is convenient and efficient for our proposed SANNE compared to using the feature vectors of bag-of-words.

Positional embeddings We hypothesize that the relative positions among nodes in the random walks are useful to provide meaningful information about the graph structure. Hence we add to each position i in the random walks a pre-defined positional embedding $\mathbf{t}_i \in \mathbb{R}^d, i \in \{1, 2, ..., N\}$ using the sinusoidal functions [25], so that we can use $\mathbf{u}^0_{\mathsf{v}_{w,i}} = \mathbf{x}_{\mathsf{v}_{w,i}} + \mathbf{t}_i$ where $\mathsf{t}_{i,2j} = \sin(i/10000^{2j/d})$ and $\mathsf{t}_{i,2j+1} = \cos(i/10000^{2j/d})$. From preliminary experiments, adding the positional embeddings produces better performances on CORA and PUBMED; thus, we keep to use the positional embeddings on these two datasets.

Transductive setting *This setting is used in most of the existing approaches where we use the entire input graph, i.e., all nodes are present during training.* We fix the dimension size d of feature vectors and node embeddings to 128 ($d = 128$ with respect to the Doc2Vec-based new feature vectors), the batch size to 64, the number M of sampling neighbors to 4 ($M = 4$) and the number of samples in the sampled loss function to 512 ($|\mathcal{V}'| = 512$). We also sample T random walks of a fixed length $N = 8$ starting from each node, wherein T is empirically varied in {16, 32, 64, 128}. We vary the hidden size of the feed-forward sub-layers in {1024, 2048}, the number K of attention layers in {2, 4, 8} and the number H of attention heads in {4, 8, 16}. The dimension size s of attention heads is set to satisfy that $Hs = d$.

Inductive setting We use the same inductive setting as used in [28, 6]. *Specifically, for each of 10 data splits, we first remove all 1000 nodes in the test set from the original graph before the training phase, so that these nodes are becoming unseen/new in the testing/evaluating phase.* We then apply the standard training process on the resulting graph. From preliminary experiments, we set the number T of random walks sampled for each node on CORA and PUBMED to 128 ($T = 128$), and on CITESEER to 16 ($T = 16$). Besides, we adopt the same value sets of other hyper-parameters for tuning as used in the transductive setting to train our SANNE in this inductive setting. After training, we infer the embedding for each unseen/new node v in the test set as described in Algorithm 2 with setting $Z = 8$.

Training SANNE to learn node embeddings For each of the 10 data splits, to learn our model parameters in the transductive and inductive settings, we use the Adam optimizer [14] to train our model and select the initial learning rate in $\{1e^{-5}, 5e^{-5}, 1e^{-4}\}$. We run up to 50 epochs and evaluate the model for every epoch to choose the best model on the validation set.

4.3 Evaluation protocol

We also follow the same setup used in [6] for the node classification task. For each of the 10 data splits, we use the learned node embeddings as feature inputs to learn a L2-regularized logistic regression classifier [7] on the training set. We monitor the classification accuracy on the validation set for every training epoch, and take the model that produces the highest accuracy on the validation set to compute the accuracy on the test set. We finally report the mean and standard deviation of the accuracies across 10 test sets in the 10 data splits.

Baseline models: We compare our unsupervised SANNE with previous unsupervised models including DeepWalk (DW), Doc2Vec and EP-B; and previous supervised models consisting of Planetoid, GCN and GAT. Moreover, as reported in [9], GraphSAGE obtained low accuracies on CORA, PUBMED and CITESEER, thus we do not include GraphSAGE as a strong baseline.

The results of DeepWalk (DW), DeepWalk+BoW (DW+BoW), Planetoid, GCN and EP-B are taken from [6].[1] Note that DeepWalk+BoW denotes a concatenation between node embeddings learned by DeepWalk and the bag-of-words feature vectors. Regarding the inductive setting for DeepWalk, [6] computed the embeddings for new nodes by averaging the embeddings of their neighbors. In addition, we provide our new results for Doc2Vec and GAT using our experimental setting.

[1] As compared to our experimental results for Doc2Vec and GAT, showing the statistically significant differences for DeepWalk, Planetoid, GCN and EP-B against our SANNE in Table 2 is justifiable.

4.4 Main results

Table 2 reports the experimental results in the transductive and inductive settings where the best scores are in bold, while the second-best in underline. As discussed in [6], the experimental setup used for GCN and GAT [15, 26] is not fair enough to show the effectiveness of existing models when the models are evaluated only using the fixed training, validation and test sets split by [28], thus we do not rely on the GCN and GAT results reported in the original papers. Here, we do include the accuracy results of GCN and GAT using the same settings used in [6].

Table 2. Experimental results on the CORA, PUBMED and CITESEER test sets in the transductive and inductive settings across the 10 data splits. The best score is in bold, while the second-best score is in underline. "Unsup" denotes a group of unsupervised models. "Sup" denotes a group of supervised models using node labels from the training set during training. "Semi" denotes a group of semi-supervised models also using node labels from the training set together with node feature vectors from the entire dataset during training. $*$ denotes the statistically significant differences against our SANNE at $p < 0.05$ (using the two-tailed *paired t-test*). Numeric subscripts denote the relative error reductions over the baselines. *Note that the inductive setting [28, 6] is used to evaluate the models when we do not access nodes in the test set during training. This inductive setting was missed in the original GCN and GAT papers which relied on the semi-supervised training process* for CORA, PUBMED and CITESEER. Regarding the inductive setting on Cora and Citeseer, many neighbors of test nodes also belong to the test set, thus these neighbors are unseen during training and then become new nodes in the testing/evaluating phase.

Transductive		**Cora**	**Pubmed**	**Citeseer**
Unsup	DW [22]	$71.11 \pm 2.70^{*}_{33.6}$	$73.49 \pm 3.00^{*}_{23.3}$	$47.60 \pm 2.34^{*}_{43.0}$
	DW+BoW	$76.15 \pm 2.06^{*}_{19.6}$	$77.82 \pm 2.19^{*}_{8.3}$	$61.87 \pm 2.30^{*}_{21.8}$
	Doc2Vec [16]	$64.90 \pm 3.07^{*}_{45.4}$	$76.12 \pm 1.62^{*}_{14.9}$	$64.58 \pm 1.84^{*}_{15.8}$
	EP-B [6]	$\underline{78.05 \pm 1.49}^{*}_{12.6}$	$\underline{79.56 \pm 2.10}_{0.5}$	$\mathbf{71.01 \pm 1.35}_{-2.9}$
	Our **SANNE**	**80.83 ± 1.94**	**79.67 ± 1.28**	$\underline{70.18 \pm 2.12}$
Semi	GCN [15]	$79.59 \pm 2.02_{6.1}$	$77.32 \pm 2.66^{*}_{10.4}$	$69.21 \pm 1.25_{3.1}$
	GAT [26]	$81.72 \pm 2.93_{-4.8}$	$79.56 \pm 1.99_{0.5}$	$70.80 \pm 0.92_{-2.1}$
	Planetoid [28]	$71.90 \pm 5.33^{*}_{31.7}$	$74.49 \pm 4.95^{*}_{20.3}$	$58.58 \pm 6.35^{*}_{28.0}$
Inductive		**Cora**	**Pubmed**	**Citeseer**
Unsup	DW+BoW	$68.35 \pm 1.70^{*}_{26.5}$	$74.87 \pm 1.23^{*}_{20.6}$	$59.47 \pm 2.48^{*}_{23.1}$
	EP-B [6]	$\underline{73.09 \pm 1.75}^{*}_{13.6}$	$\underline{79.94 \pm 2.30}_{0.5}$	$\underline{68.61 \pm 1.69}_{0.7}$
	Our **SANNE**	**76.75 ± 2.45**	**80.04 ± 1.67**	**68.82 ± 3.21**
Sup	GCN [15]	$67.76 \pm 2.11^{*}_{27.9}$	$73.47 \pm 2.48^{*}_{24.8}$	$63.40 \pm 0.98^{*}_{14.8}$
	GAT [26]	$69.37 \pm 3.81^{*}_{24.1}$	$71.29 \pm 3.56^{*}_{30.5}$	$59.55 \pm 4.21^{*}_{22.9}$
	Planetoid [28]	$64.80 \pm 3.70^{*}_{33.9}$	$75.73 \pm 4.21^{*}_{17.8}$	$61.97 \pm 3.82^{*}_{18.0}$

Regarding the transductive setting, SANNE obtains the highest scores on CORA and PUBMED, and the second-highest score on CITESEER in the group of unsupervised models. In particular, SANNE works better than EP-B on CORA, while both models produce similar scores on PUBMED. Besides, SANNE

produces high competitive results compared to the up-to-date semi-supervised models GCN and GAT. Especially, SANNE outperforms GCN with relative error reductions of 6.1% and 10.4% on CORA and PUBMED respectively. Furthermore, it is noteworthy that there is no statistically significant difference between SANNE and GAT at $p < 0.05$ (using the two-tailed paired t-test)on these datasets.

EP-B is more appropriate than other models for CITESEER in the transductive setting because (i) EP-B simultaneously learns word embeddings from the texts within nodes, which are then used to reconstruct the embeddings of nodes from their neighbors; (ii) CITESEER is quite sparse; thus word embeddings can be useful in learning the node embeddings. But we emphasize that using a significant test, there is *no* difference between EP-B and our proposed SANNE on CITESEER; hence the results are comparable.

More importantly, regarding the inductive setting, SANNE obtains the highest scores on three benchmark datasets, hence these show the effectiveness of SANNE in inferring the plausible embeddings for new nodes. Especially, SANNE outperforms both GCN and GAT in this setting, e.g., SANNE achieves absolute improvements of 8.9%, 6.6% and 5.4% over GCN, and 7.3%, 8.7% and 9.3% over GAT, on CORA, PUBMED and CITESEER, respectively (with relative error reductions of more than 14% over GCN and GAT).

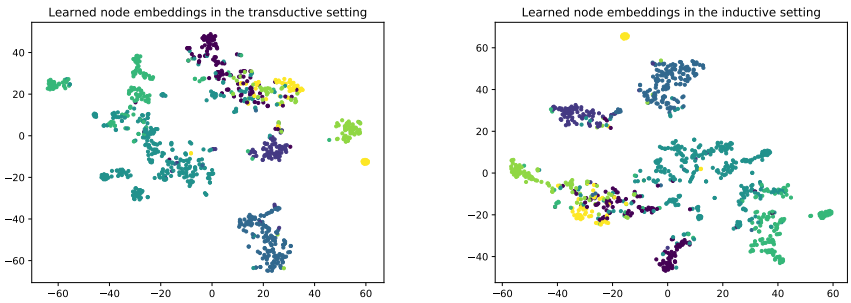

Fig. 2. A visualization of the learned node embeddings in the transductive and inductive settings on CORA.

Compared to the transductive setting, the inductive setting is particularly difficult due to requiring the ability to align newly observed nodes to the present nodes. As shown in Table 2, there is a significant decrease for GCN and GAT from the transductive setting to the inductive setting on all three datasets, while by contrast, our SANNE produces reasonable accuracies for both settings. To qualitatively demonstrate this advantage of SANNE, we use t-SNE [18] to visualize the learned node embeddings on one data split of the CORA dataset in Figure 2. We see a similarity in the node embeddings (according to their labels) between two settings, verifying the plausibility of the node embeddings learned by our SANNE in the inductive setting.

4.5 Effects of hyper-parameters

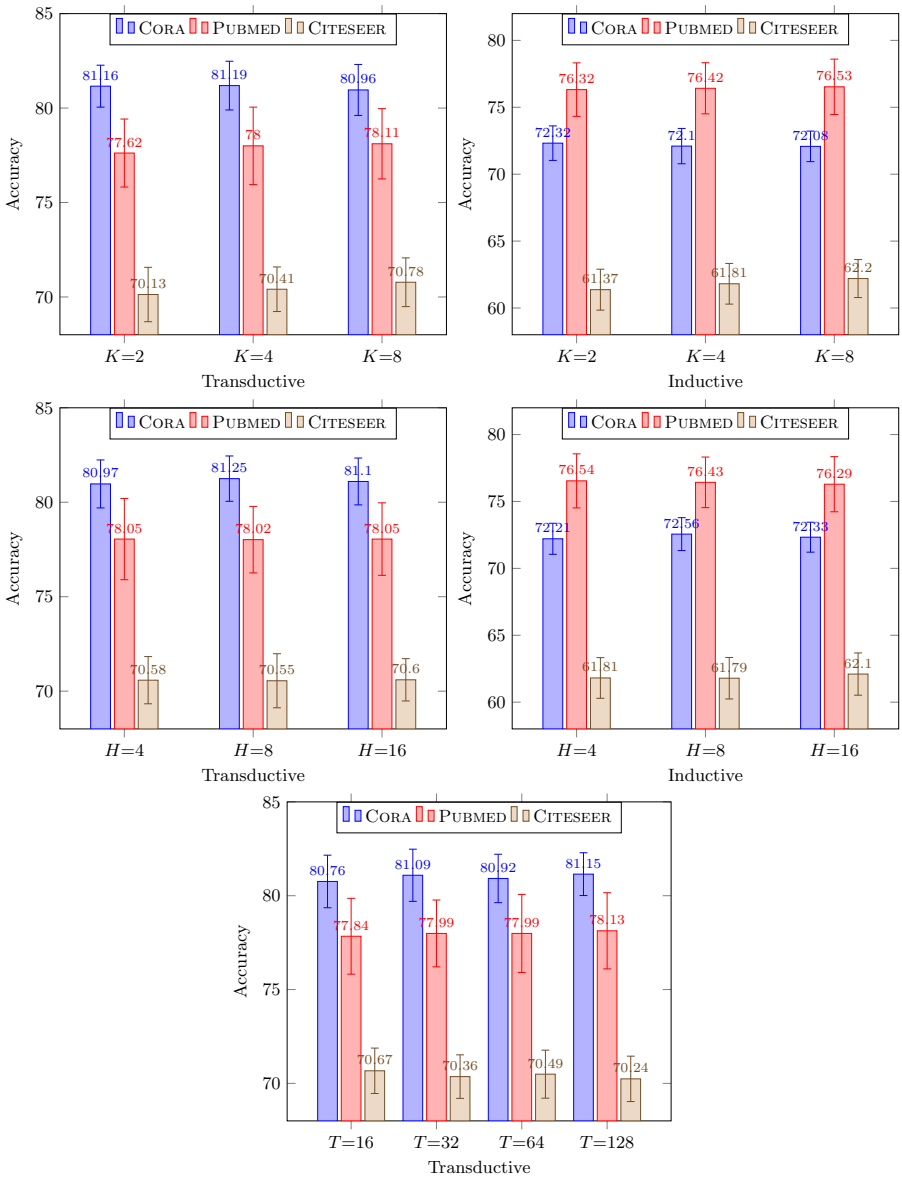

Fig. 3. Effects of the number T of random walks, the number K of attention layers and the number H of attention heads on the validation sets. We fixed the same value for one hyper-parameter and tune other hyper-parameters for all 10 data splits of each dataset.

We investigate the effects of hyper-parameters on the Cora, Pubmed, and Citeseer validation sets of 10 data splits in Figure 3, when we use the same value for one hyper-parameter and then tune other hyper-parameters for all 10

data splits of each dataset. Regarding the transductive setting, we see that the high accuracies can be generally obtained when using $T = 128$ on CORA and PUBMED, and $T = 16$ on CITESEER. This is probably because CITESEER are more sparse than CORA and PUBMED, especially the average number of neighbors per node on CORA and PUBMED are 2.0 and 2.2 respectively, while it is just 1.4 on CITESEER. This is also the reason why we set $T = 128$ on CORA and PUBMED, and $T = 16$ on CITESEER during training in the inductive setting. Besides, regarding the number K of attention layers for both the transductive and inductive settings, using a small K produces better results on CORA. At the same time, there is an accuracy increase on PUBMED and CITESEER along with increasing K. Regarding the number H of attention heads, we achieve higher accuracies when using $H = 8$ on CORA in both the settings. Besides, there is not much difference in varying H on PUBMED and CITESEER in the transductive setting. But in the inductive setting, using $H = 4$ gives high scores on PUBMED, while the high scores on CITESEER are obtained by setting $H = 16$.

4.6 Ablation analysis

Table 3. Ablation results on the validation sets in the transductive setting. (i) Without using the feed-forward sub-layer: $\mathbf{u}_v^{(k)} = \text{LAYERNORM}\left(\mathbf{u}_v^{(k-1)} + \text{ATT}\left(\mathbf{u}_v^{(k-1)}\right)\right)$ (ii) Without using the multi-head self-attention sub-layer: $\mathbf{u}_v^{(k)} = \text{LAYERNORM}\left(\mathbf{u}_v^{(k-1)} + \text{FF}\left(\mathbf{u}_v^{(k-1)}\right)\right)$. $*$ denotes the statistically significant differences at $p < 0.05$ (using the two-tailed paired t-test).

Transductive	**Cora**	**Pubmed**	**Citeseer**
Our **SANNE**	81.32 ± 1.20	78.28 ± 1.24	70.77 ± 1.18
(i) w/o FF	80.77 ± 1.34	77.90 ± 1.76	70.36 ± 1.32
(ii) w/o ATT	77.87 ± 1.09*	74.52 ± 2.66*	65.68 ± 1.31*

We compute and report our ablation results on the validation sets in the transductive setting over two factors in Table 3. There is a decrease in the accuracy results when not using the feed-forward sub-layer, but we do not see a significant difference between with and without using this sub-layer (at $p < 0.05$ using the two-tailed paired t-test). More importantly, without the multi-head self-attention sub-layer, the results degrade by more than 3.2% on all three datasets, showing the merit of this self-attention sub-layer in learning the plausible node embeddings. Note that similar findings also occur in the inductive setting.

5 Conclusion

We introduce a novel unsupervised embedding model SANNE to leverage from the random walks to induce the transformer self-attention network to learn node

embeddings. SANNE aims to infer plausible embeddings not only for present nodes but also for new nodes. Experimental results show that our SANNE obtains the state-of-the-art results on CORA, PUBMED, and CITESEER in both the transductive and inductive settings. Our code is available at: `https://github.com/daiquocnguyen/Walk-Transformer`.[2]

Acknowledgements

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

References

1. Ba, J.L., Kiros, J.R., Hinton, G.E.: Layer normalization. arXiv:1607.06450 (2016)
2. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. International Conference on Learning Representations (ICLR) (2015)
3. Battaglia, P.W., Hamrick, J.B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al.: Relational inductive biases, deep learning, and graph networks. arXiv:1806.01261 (2018)
4. Cai, H., Zheng, V.W., Chang, K.: A comprehensive survey of graph embedding: problems, techniques and applications. IEEE Transactions on Knowledge and Data Engineering **30**(9), 1616–1637 (2018)
5. Chen, H., Perozzi, B., Al-Rfou, R., Skiena, S.: A tutorial on network embeddings. arXiv:1808.02590 (2018)
6. Duran, A.G., Niepert, M.: Learning graph representations with embedding propagation. In: Advances in Neural Information Processing Systems. pp. 5119–5130 (2017)
7. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: LIBLINEAR: A Library for Large Linear Classification. Journal of Machine Learning Research **9**, 1871–1874 (2008)
8. Grover, A., Leskovec, J.: Node2Vec: Scalable Feature Learning for Networks. In: The ACM SIGKDD Conference on Knowledge Discovery and Data Mining. pp. 855–864 (2016)
9. Guo, J., Xu, L., Chen, E.: SPINE: Structural Identity Preserved Inductive Network Embedding. arXiv:1802.03984 (2018)
10. Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in Neural Information Processing Systems. pp. 1024–1034 (2017)
11. Hamilton, W.L., Ying, R., Leskovec, J.: Representation learning on graphs: Methods and applications. arXiv:1709.05584 (2017)
12. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778 (2016)

[2] Our accuracy results are obtained using the implementation based on Tensorflow 1.6, and now this implementation is out-of-date. We have released the SANNE implementation based on Pytorch 1.5 for future works.

13. Jean, S., Cho, K., Memisevic, R., Bengio, Y.: On using very large target vocabulary for neural machine translation. In: ACL. pp. 1–10 (2015)
14. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. International Conference on Learning Representations (ICLR) (2015)
15. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (ICLR) (2017)
16. Le, Q., Mikolov, T.: Distributed representations of sentences and documents. In: International Conference on Machine Learning. pp. 1188–1196 (2014)
17. Li, J., Zhu, J., Zhang, B.: Discriminative deep random walk for network classification. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 1004–1013 (2016)
18. Maaten, L.v.d., Hinton, G.: Visualizing data using t-sne. Journal of machine learning research **9**, 2579–2605 (2008)
19. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems. pp. 3111–3119 (2013)
20. Namata, G.M., London, B., Getoor, L., Huang, B.: Query-driven active surveying for collective classification. In: Workshop on Mining and Learning with Graphs (2012)
21. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. Tech. rep., Stanford InfoLab (1999)
22. Perozzi, B., Al-Rfou, R., Skiena, S.: DeepWalk: Online Learning of Social Representations. In: The ACM SIGKDD Conference on Knowledge Discovery and Data Mining. pp. 701–710 (2014)
23. Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., Eliassi-Rad, T.: Collective classification in network data. AI magazine **29**(3), 93 (2008)
24. Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: LINE: Large-scale Information Network Embedding. In: WWW. pp. 1067–1077 (2015)
25. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems. pp. 5998–6008 (2017)
26. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph Attention Networks. International Conference on Learning Representations (ICLR) (2018)
27. Wang, D., Cui, P., Zhu, W.: Structural deep network embedding. In: The ACM SIGKDD Conference on Knowledge Discovery and Data Mining. pp. 1225–1234 (2016)
28. Yang, Z., Cohen, W.W., Salakhutdinov, R.: Revisiting semi-supervised learning with graph embeddings. In: International Conference on Machine Learning. pp. 40–48 (2016)
29. Zhang, D., Yin, J., Zhu, X., Zhang, C.: Network representation learning: A survey. IEEE Transactions on Big Data **6**, 3–28 (2020)

A Capsule Network-based Model for Learning Node Embeddings

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

Tu Dinh Nguyen
nguyendinhtu@gmail.com

Dat Quoc Nguyen
VinAI Research, Vietnam
v.datnq9@vinai.io

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

ABSTRACT

In this paper, we focus on learning low-dimensional embeddings for nodes in graph-structured data. To achieve this, we propose Caps2NE – a new unsupervised embedding model leveraging a network of two capsule layers. Caps2NE induces a routing process to aggregate feature vectors of context neighbors of a given target node at the first capsule layer, then feed these features into the second capsule layer to infer a plausible embedding for the target node. Experimental results show that our proposed Caps2NE obtains state-of-the-art performances on benchmark datasets for the node classification task. Our code is available at: https://github.com/daiquocnguyen/Caps2NE.

CCS CONCEPTS

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

ACM Reference Format:
Dai Quoc Nguyen, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Phung. 2020. A Capsule Network-based Model for Learning Node Embeddings. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3340531.3417455

1 INTRODUCTION

Numerous real-world and scientific data are represented in forms of graphs, e.g. data from knowledge graphs, recommender systems, social and citation networks as well as telecommunication and biological networks [1, 4]. Recent years have witnessed many successful downstream applications of utilizing the graph-structured data such as for improving information extraction and text classification systems [13], traffic learning and forecasting [5] and for advertising and recommending relevant items to users [22, 24]. This is largely boosted by a surge of methodologies that learn embedding representations to encode graph structures [3].

One of the most important tasks in learning graph representations is to learn low-dimensional embeddings for nodes in the graph-structured data [26]. These embedding vectors can then be used in a downstream task such as node classification, i.e., using the learned node embeddings as feature inputs to train a classifier to predict node labels [10].

A simple and effective approach is to treat each node as a word token and each graph as a text collection; hence we can apply a word embedding model such as Word2Vec [15] to learn node embeddings such as DeepWalk [18] and Node2Vec [8]. Recent work has developed deep neural networks (DNN) for the node classification task, e.g., GCN [13], GraphSAGE [10] and GAT [21]. We see that the DNN-based approaches are showing state-of-the-art performances, but not well-efficient to exploit the structural dependencies among nodes.

In this paper, inspired by the advanced capsule networks [19], we present Caps2NE – a new unsupervised embedding model that adapts capsule network to learn node embeddings. Caps2NE aims to capture k-hops context neighbors to predict a target node. In particular, Caps2NE consists of two capsule layers with connections from the first to the second layer, but no connections within layers. The first layer constructs capsules to encapsulate context neighbors. Then a routing process is used to aggregate the feature information from capsules in the first layer to only one capsule in the second layer. After that, the second layer produces a continuous vector which is used to infer an embedding for the target node. Note that encapsulating the context neighbors into the corresponding capsules aims to preserve node properties more efficiently. And the routing process aims to generate high-level features to infer plausible node embeddings effectively.

Our main contributions are as follows:

- We investigate the advanced use of capsule networks for the graph-structured data and propose a new embedding model Caps2NE to learn node embeddings.
- We evaluate the performance of the proposed Caps2NE on benchmark datasets for the node classification task.
- The experimental results show that that our Caps2NE produces state-of-the-art accuracy results on these datasets.

2 THE PROPOSED CAPS2NE

This section presents our Caps2NE model. In particular, we detail how to sample data from an input graph, then how to construct Caps2NE to learn node embeddings.

Definition 1. A network graph $\mathcal{G}$ is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, in which $\mathcal{V}$ is a set of nodes, $\mathcal{E} \subseteq \{(v, v')|v, v' \in \mathcal{V}\}$ is a set of edges, and each node $v \in \mathcal{V}$ is associated with a feature vector $\boldsymbol{x}_v \in \mathbb{R}^d$. We aim to learn a node embedding $\mathbf{o}_v$ for each node v.

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3417455

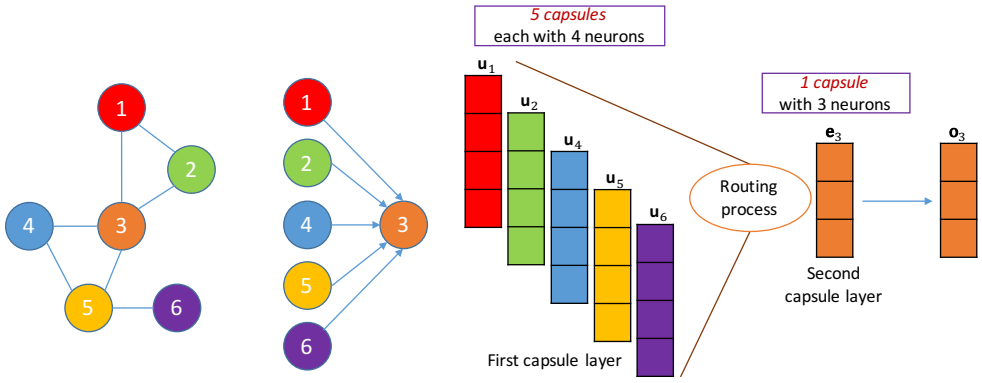

Figure 1: Processes in our Caps2NE with $q = 6, d = 4, k = 3$ for an illustration purpose. Note that in this illustration, we use numbered subscripts to denote nodes themselves, not indexes of nodes or capsules. The indexes of capsules are fixed from 1 to $(q-1)$, not depending on the indexes of the context neighbors. With v be the target node 3, we have $C_v = \{v_1 = 1, v_2 = 2, v_3 = 4, v_4 = 5, v_5 = 6\}$.

Sampling input pairs. We follow Perozzi et al. [18] to uniformly sample a number T of random walks of length q for every node in $\mathcal{V}$. From each random walk, we randomly sample a target node v, treat $(q-1)$ remaining nodes as the context neighbors of node v, and construct an input pair of (C_v, v), where we denote C_v be the list of context neighbors v_i of the target node v (here, $i \in \{1, 2, ..., q-1\}$ and $|C_v| = q-1$).

Figure 1 shows an example of a graph consisting of 6 nodes. If we sample a random walk of length $q = 6$ for node 1 such as $\{1, 2, 3, 4, 5, 6\}$ and select node 3 as the target node v, then the remaining nodes $\{1, 2, 4, 5, 6\}$ are treated as the context neighbors of node 3, i.e., $C_v = \{v_1 = 1, v_2 = 2, v_3 = 4, v_4 = 5, v_5 = 6\}$.

Definition 2. A *capsule* is a group of neurons. A capsule *layer* is a group of capsules without connections among capsules in the same layer [19]. Two continuous capsule layer is connected using a *routing* process.

Constructing Caps2NE. We build our Caps2NE with two capsule layers. In the first layer, we construct $(q-1)$ capsules, where the feature vector of each context neighbor v_i is encapsulated by the i-th corresponding capsule (with $i \in \{1, 2, ..., q-1\}$). In the second layer, we construct one capsule to produce a vector representation which is then used to infer an embedding for the target node v.

The first capsule layer consists of $(q-1)$ capsules, in which the i-th capsule use a non-linear squashing function to transform the feature vector $\boldsymbol{x}_{v_i}$ of the context neighbor v_i into $\mathbf{u}_{v_i}^{(i)}$ as:

$$\mathbf{u}_{v_i}^{(i)} = \text{squash}\left(\boldsymbol{x}_{v_i}\right) = \frac{\|\boldsymbol{x}_{v_i}\|^2}{1 + \|\boldsymbol{x}_{v_i}\|^2} \frac{\boldsymbol{x}_{v_i}}{\|\boldsymbol{x}_{v_i}\|} \tag{1}$$

The squashing function ensures that the orientation of each feature vector is unchanged while its length is scaled down to below 1.

Vectors $\mathbf{u}_{v_i}^{(i)}$ are then linearly transformed using weight matrices $\mathbf{W}_i \in \mathbb{R}^{k \times d}$ to produce vectors $\hat{\mathbf{u}}_{v_i}^{(i)} \in \mathbb{R}^k$. These vectors $\hat{\mathbf{u}}_{v_i}^{(i)}$ are weighted to sum up to return a vector $\mathbf{s}_v \in \mathbb{R}^k$ for the capsule in the second layer (recall that the second layer consists of only one capsule). This capsule then performs the non-linear squashing function to produce a vector $\mathbf{e}_v \in \mathbb{R}^k$. Formally, we have:

$$\mathbf{e}_v = \text{squash}\left(\mathbf{s}_v\right) \;\; ; \;\; \mathbf{s}_v = \sum_i c_i \hat{\mathbf{u}}_{v_i}^{(i)} \;\; ; \;\; \hat{\mathbf{u}}_{v_i}^{(i)} = \mathbf{W}_i \mathbf{u}_{v_i}^{(i)} \tag{2}$$

where c_i are coupling coefficients determined by the routing process as presented in Algorithm 1. Here, c_i aims to weight $\mathbf{u}_{v_i}^{(i)}$ of the i-th capsule in the first layer.

As we use one capsule in the second layer, we make two differences in our routing process in Algorithm 1: (i) we apply softmax in a direction from all capsules in the previous layer to each of capsules in the next layer, (ii) thus, we propose a new update rule $(b_i \leftarrow \hat{\mathbf{u}}_{v_i}^{(i)} \cdot \mathbf{e}_v)$ instead of employing $(b_i \leftarrow b_i + \hat{\mathbf{u}}_{v_i}^{(i)} \cdot \mathbf{e}_v)$ originally used by Sabour et al. [19].

Algorithm 1: The Caps2NE routing process.

```
for i = 1, 2, ..., q-1 do
    b_i ← 0
for iteration = 1, 2, ..., m do
    c ← softmax(b)
    s_v ← Σ_i c_i û^(i)_{v_i}
    e_v ← squash(s_v)
    for i = 1, 2, ..., q-1 do
        b_i ← û^(i)_{v_i} · e_v
```

Learning model parameters. The vector representation $\mathbf{e}_v$ is then used to infer the final embedding $\mathbf{o}_v \in \mathbb{R}^k$ of the target node v, as shown in Equation 3. We learn all model parameters (including the node embeddings $\mathbf{o}_v$) by minimizing the sampled softmax loss function [11] applied to the target node v as:

$$\mathcal{L}_{\text{Caps2NE}}(v) = -\log \frac{\exp(\mathbf{o}_v^\top \mathbf{e}_v)}{\sum_{v' \in \mathcal{V}'} \exp(\mathbf{o}_{v'}^\top \mathbf{e}_v)} \tag{3}$$

where $\mathcal{V}'$ is a subset sampled from $\mathcal{V}$.

Algorithm 2: The Caps2NE learning process.

Input: A network graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
for $\mathsf{v} \in \mathcal{V}$ **do**
  SAMPLE T random walks of length q starting at v
for *each* random walk **do**
  SAMPLE a node v as a target node
  $\mathsf{C_v} \leftarrow$ Remaining nodes
  for *i = 1, 2, ..., q-1* **do**
    $\mathbf{u}_{\mathsf{v}_i}^{(i)} \leftarrow \text{squash}\left(\boldsymbol{x}_{\mathsf{v}_i}\right) \quad \forall \mathsf{v}_i \in \mathsf{C_v}$
  $\mathbf{e}_{\mathsf{v}} \leftarrow \text{ROUTING}\left(\left\{\mathbf{u}_{\mathsf{v}_i}^{(i)}\right\}_{i=1}^{q-1}\right)$
  $\mathbf{o}_{\mathsf{v}} \leftarrow \mathbf{e}_{\mathsf{v}}$

We briefly represent the general learning process of our proposed Caps2NE model in Algorithm 2 whose main steps **3**, **7–9** and **10** are previously detailed in parts "*Sampling input pairs*", "*Constructing Caps2NE*" and "*Learning model parameters*", respectively.

We illustrate our model in Figure 1 where the length q of random walks, the dimension size d of the feature vectors and the dimension size k of output node embeddings are equal to 6, 4 and 3, respectively. Thus, the first capsule layer has 5 capsules, each with 4 neurons, and the second capsule layer has 1 capsule with 3 neurons. For the target node 3 in the illustration, the vector output of the capsule in the second layer is used to infer the embedding of node 3. Our Caps2NE aims to aggregate feature information from the context neighbors (i.e., k-hops neighbors) to infer the target node 3; hence this helps our proposed model to infer the structural dependencies among nodes to produce the plausible node embeddings effectively.

Algorithm 3: The inference process for new nodes.

Input: A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a trained model $\text{Caps2NE}_{trained}$, a set $\mathcal{V}_{test}$ of new nodes.
for $v \in \mathcal{V}_{test}$ **do**
  SAMPLE Z pairs $\{p_j\}_{j=1}^{Z}$ of (C_v, v)
  for $j \in \{1, 2, \ldots, Z\}$ **do**
    $\mathbf{e}_{(v,j)} \leftarrow \text{Caps2NE}_{trained}\left(p_j\right)$
  $\mathbf{o}_v \leftarrow \text{AVERAGE}\left(\{\mathbf{e}_{(v,j)}\}_{j=1}^{Z}\right)$

Inferring embeddings for new nodes in the inductive setting. Algorithm 3 shows how we infer an embedding for a new node v adding to an existing graph. After training our model, we generate random walks of length q to extract Z pairs of (C_v, v). We use each of these pairs as an input for our trained model and then collect the output vector $\mathbf{e}$ from the second capsule layer. Thus, we obtain Z vectors associated with node v and then average them into an embedding representation of v.

3 EXPERIMENTAL RESULTS ON PPI, POS, AND BLOGCATALOG

3.1 Datasets and data splits

PPI [2] is a subgraph of the Protein-Protein Interaction network for Homo Sapiens, and its node labels represent biological states. POS [14] is a co-occurrence network of words from the Wikipedia dump, and its node labels represent the part-of-speech tags. BLOGCATALOG [25] is a social network of relationships of the bloggers listed on the BlogCatalog website, and its node labels represent bloggers' interests. PPI, POS and BLOGCATALOG are given without node features, in which each node is assigned with one or more class labels. These datasets are used for the multi-label node classification task. Table 1 presents the statistics of these benchmark datasets.

Table 1: Statistics of the experimental datasets.

Dataset	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	#Classes
PPI	3,890	76,584	50
POS	4,777	184,812	40
BLOGCATALOG	10,312	333,983	39

A certain fraction γ of nodes is provided to train a classifier which is then used to predict the labels of the remaining nodes.

3.2 Training protocol

We only use the transductive setting for these three datasets. We uniformly sample 64 random walks ($T = 64$) of length 10 ($q = 10$) for each node in the graph. In each random walk, we rotationally select each node in the walk as a target node and 9 remaining nodes as its context nodes. We also run up to 50 training epochs and use the batch size to 128, the embedding size $k = 128$ and $|\mathcal{V}'| = 256$ in Equation 3. We vary the Adam initial learning rate $lr \in \{1e^{-5}, 5e^{-5}, 1e^{-4}\}$. Nodes are given without pre-computed features, hence we set the size d of feature vectors $\boldsymbol{x}_{\mathsf{v}_i}$ to 128 ($d = 128$), and these vectors are randomly initialized uniformly, and updated during training.

3.3 Evaluation protocol

We follow the same experimental setup used for the multi-label node classification task from Perozzi et al. [18] and Duran and Niepert [6] where we uniformly sample a fraction γ of nodes at random as training set for learning a one-vs-rest logistic regression classifier. The learned node embeddings after each Caps2NE training epoch are used as input feature vectors for this logistic regression classifier. We use default parameters for learning this classifier from Perozzi et al. [18]. The classifier is then used to categorize the remaining nodes. We monitor the Micro-F1 and Macro-F1 scores of the classifier after each Caps2NE training epoch, for which the best model is chosen by using 10-fold cross-validation for each fraction value. We repeat this manner 10 times for each fraction value, and then compute the averaged Micro-F1 and Macro-F1 scores. We show final scores w.r.t. each value $\gamma \in \{10\%, 50\%, 90\%\}$. The baseline results are taken from Duran and Niepert [6].

Table 2: Multi-label classification results on PPI, POS and BlogCatalog.

Method (Micro-F1)	POS			PPI			BlogCatalog		
	γ = 10%	γ = 50%	γ = 90%	γ = 10%	γ = 50%	γ = 90%	γ = 10%	γ = 50%	γ = 90%
DeepWalk	45.02	49.10	49.33	17.14	**23.52**	25.02	34.48	38.11	38.34
LINE	45.22	**51.64**	52.28	16.55	23.01	**25.28**	34.83	38.99	38.77
Node2Vec	44.66	48.73	49.73	17.00	23.31	24.75	**35.54**	39.31	40.03
EP-B	**46.97**	49.52	50.05	17.82	23.30	24.74	35.05	**39.44**	40.41
Our Caps2NE	46.01	50.93	**53.92**	**18.52**	23.15	25.08	34.31	38.35	**40.79**
Method (Macro-F1)	**POS**			**PPI**			**BlogCatalog**		
	γ = 10%	γ = 50%	γ = 90%	γ = 10%	γ = 50%	γ = 90%	γ = 10%	γ = 50%	γ = 90%
DeepWalk	8.20	10.84	12.23	13.01	18.73	20.01	18.16	22.65	22.86
LINE	8.49	12.43	12.40	12,79	18.06	**20.59**	18.13	22.56	23.00
Node2Vec	8.32	11.07	12.11	13.32	18.57	19.66	**19.08**	23.97	24.82
EP-B	8.85	10.45	12.17	13.80	18.96	20.36	**19.08**	**25.11**	25.97
Our Caps2NE	**9.71**	**13.16**	**14.11**	**15.20**	**19.63**	20.27	18.40	24.80	**26.63**

3.4 Overall results

We show in Table 2 the Micro-F1 and Macro-F1 scores on test sets in the transductive setting. Especially, on POS, Caps2NE produces a new state-of-the-art Macro-F1 score for each of the three fraction values γ, the highest Micro-F1 score when $\gamma = 90\%$ and the second highest Micro-F1 scores when $\gamma \in \{10\%, 50\%\}$. Caps2NE obtains new highest F1 scores on PPI and BlogCatalog when $\gamma = 10\%$ and $\gamma = 90\%$, respectively. On PPI, Caps2NE also achieves the highest Macro-F1 score when $\gamma = 50\%$ and the second highest Micro-F1 score when $\gamma = 90\%$. On BlogCatalog, Caps2NE also achieves the second highest Macro-F1 scores when $\gamma \in \{10\%, 50\%\}$.

In short, from Table 2, Caps2NE obtains top performances on these three datasets: producing the highest scores in 9 over 18 comparison groups (3 datasets $\times$ 3 values of the fraction $\gamma \times 2$ metrics), the second highest scores in 5/18 groups and competitive scores in the remaining 4 groups.

4 EXPERIMENTAL RESULTS ON CORA, CITESEER, AND PUBMED

4.1 Datasets and data splits

Cora, Citeseer [20] and Pubmed [16] are citation networks where each node represents a document (here, each node is associated with a class labeling the main topic of the document), and each edge represents a citation link between two documents. Each node is also associated with a feature vector of a bag-of-words, i.e. the feature vectors $\boldsymbol{x}_{v_i}$ in the first capsule layer (Equation 1) are pre-computed based on bag-of-words features and fixed during training. Table 3 presents the statistics of these three benchmark datasets.

Duran and Niepert [6] show that the experimental setup used in [13, 21] is not fair to show the effectiveness of existing models when these models are evaluated using the fixed & pre-split training, validation and test sets from the Planetoid model [23]. Therefore, for a fair comparison, we follow the same experimental setup used in [6, 17]. In particular, for each dataset, we uniformly sample 20 random nodes for each class as training data, 1000 different random nodes as a validation set and 1000 different random nodes as a test set. We then repeat this manner 10 times to produce 10 data splits of training-validation-test sets.

Table 3: Statistics of the experimental datasets. d is the dimension size of the feature vectors.

Dataset	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	#Classes	d
Cora	2,708	5,429	7	1,433
Citeseer	3,327	4,732	6	3,703
Pubmed	19,717	44,338	3	500

4.2 Training protocol

Transductive setting. We set the embedding size k to 128 (k = 128) and the number of samples in the sampled softmax loss function to 256 ($|\mathcal{V}'| = 256$ in Equation 3). We also set the batch size to 64 for both Cora and Citeseer and to 128 for Pubmed. We use a fixed walk length $q = 10$ for uniformly sampling T random walks starting from each node. We may get slightly better results when we rotationally selecting each node in the random walk as a target node. But we aim to save training time due to the limitation of computation resources, thus we only select target nodes at indexes of $\{3, 4, 5, 6\}$. We optimize the loss function using the Adam optimizer [12] and select the initial learning rate $lr \in \{1e^{-5}, 5e^{-5}, 1e^{-4}\}$. We vary the number T of random walks $T \in \{8, 16, 32, 64\}$ and the number m of iterations in the routing process (Algorithm 1) $m \in \{1, 3, 5, 7\}$. We run up to 50 epochs and evaluate the model for each epoch to choose the best model on the validation set. We use the same values of hyper-parameters above for all data splits.

Inductive setting. We use the same inductive setting as used in [6, 23] where *we firstly remove all nodes in the test set from the original graph before training phase, thus these nodes are unseen/new in the testing/evaluating phase.* We then apply the standard training process on the remaining of the graph. Here, we use the same set of hyper-parameters tuned for the transductive setting to train

Caps2NE in the inductive setting. After training, we infer the embedding for each node v in the test set as in Algorithm 3 using a fixed value $Z = 10$.

4.3 Evaluation protocol

We also follow the same setup used in Duran and Niepert [6] use to evaluate our Caps2NE. For each of 10 data splits, the learned node embeddings after each Caps2NE training epoch are used as input features for learning a L2-regularized logistic regression classifier [7] on the training set.We monitor the node classification accuracy on the validation set for every Caps2NE training epoch and then choose the model that produces the highest accuracy on the validation set to compute the accuracy on the test set. We finally report the average of the accuracies across 10 test sets from the 10 data splits. We compare Caps2NE with strong baseline models BoW (Bag-of-Words), DeepWalk, DeepWalk+BoW, EP-B [6], Planetoid, GCN and GAT. As reported in [9], GraphSAGE obtained low accuracies on Cora, Pubmed and Citeseer, thus we do not include GraphSAGE as a strong baseline.

4.4 Overall results

Transductive setting. Table 4 reports the experimental results of our proposed Caps2NE and other baselines. BoW is evaluated by directly using the bag-of-words feature vectors for learning the classifier. DeepWalk+BoW concatenates the learned embedding of a node from DeepWalk with the BoW feature vector of the node. As discussed in Duran and Niepert [6], the experimental setup used to evaluate GCN and GAT is not fair for existing models when they are evaluated using the fixed & pre-split training, validation and test sets from Yang et al. [23]. Thus we report results, and also fine-tune and re-evaluate GAT, using the same experimental setup used in Duran and Niepert [6]. The results of other baselines (e.g., BoW, DeepWalk+BoW, EP-B, Planetoid and GCN) are taken from Duran and Niepert [6].

Table 4: Accuracies on the Cora, Citeseer and Pubmed test sets in the transductive setting. "Unsup" denotes unsupervised graph embedding models, where the best score is in bold while the second best score is in underline. "Semi" denotes a group of semi-supervised models using node labels from the training set together with feature vectors of nodes from the entire dataset during training.

	Model	Cora	Citeseer	Pubmed
Unsup	BoW	58.63	58.07	70.49
	DeepWalk	71.11	47.60	73.49
	DeepWalk+BoW	76.15	61.87	77.82
	EP-B	<u>78.05</u>	<u>71.01</u>	**79.56**
	Our **Caps2NE**	**80.53**	**71.34**	<u>78.45</u>
Semi	GAT	81.72	70.80	79.56
	GCN	79.59	69.21	77.32
	Planetoid	71.90	58.58	74.49

Caps2NE obtains the highest scores on Cora and Citeseer and the second highest score on Pubmed against other unsupervised baseline models. In addition, we also compare our unsupervised Caps2NE to the semi-supervised models GCN, Planetoid and GAT, for which Caps2NE works better than GCN and Planetoid on these three datasets, and outperforms GAT on Citeseer.

Table 5: Accuracies on the Cora, Citeseer and Pubmed test sets in the inductive setting. "Unsup" denotes unsupervised graph embedding models, where the best score is in bold while the second best score is in underline. "Sup" denotes a group of supervised models using node labels from the training set during training.

	Model	Cora	Citeseer	Pubmed
Unsup	DeepWalk+BoW	68.35	59.47	74.87
	EP-B	<u>73.09</u>	<u>68.61</u>	**79.94**
	Our **Caps2NE**	**76.54**	**69.84**	<u>78.98</u>
Sup	GAT	69.37	59.55	71.29
	GCN	67.76	63.40	73.47
	Planetoid	64.80	61.97	75.73

Inductive setting: Table 5 reports the experimental results of our Caps2NE and other baselines in the inductive setting. Note that the inductive setting is used to evaluate the models when we do not access nodes in the test set during training. This inductive setting was missed in the original GCN and GAT papers which relied on the semi-supervised training process. Regarding Cora and Citeseer in the inductive setting, many neighbors of test nodes also belong to the test set, thus these neighbors are unseen during training and then become new nodes in the testing/evaluating phase. Table 4 also shows that under the inductive setting, Caps2NE produces new state-of-the-art scores of 76.54% and 69.84% on Cora and Citeseer respectively, and also obtains the second highest score of 78.98% on Pubmed. As previously discussed in the last paragraph in the "The proposed Caps2NE" section, we re-emphasize that our unsupervised Caps2NE model notably outperforms the supervised models GCN and GAT for this inductive setting. In particular, Caps2NE achieves 4+% absolute higher accuracies than both GCN and GAT on the three datasets, clearly showing the effectiveness of Caps2NE to infer embeddings for unseen nodes.

Discussion. EP-B is the best model on Pubmed: (i) EP-B simultaneously learns word embeddings on texts from all nodes. Then the embeddings of words from each node are averaged into a new feature vector which is then used to reconstruct the node embedding. (ii) On Pubmed, neighbors of unseen nodes in the test set are frequently present in the training set. Therefore, these are reasons why on Pubmed, EP-B obtains higher performance than Caps2NE and other models (but, note that we only make use of the bag-of-words feature vectors).

4.5 Ablation analysis on the routing update

The routing process presented in Algorithm 1 can be considered as an attention mechanism to compute the coupling coefficient c_i which is used to weight the output of the i-th capsule in the first layer. Sabour et al. [19] use ($b_i \leftarrow b_i + \hat{\mathbf{u}}^{(i)}_{\mathrm{v}_i} \cdot \mathbf{e}_{\mathrm{v}}$) for the image classification task, but this might not be well-suited for graph-structured data because of the high order variant among different

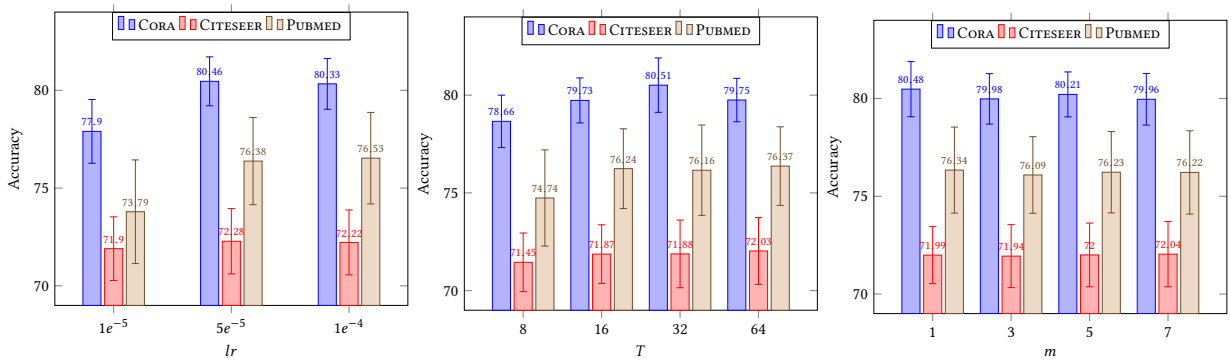

Figure 2: Effects of the Adam initial learning rate lr (left figure), the number T of random walks sampled for each node (central figure), and the number m of iterations in the routing process (right figure) on the validation sets in the transductive setting.

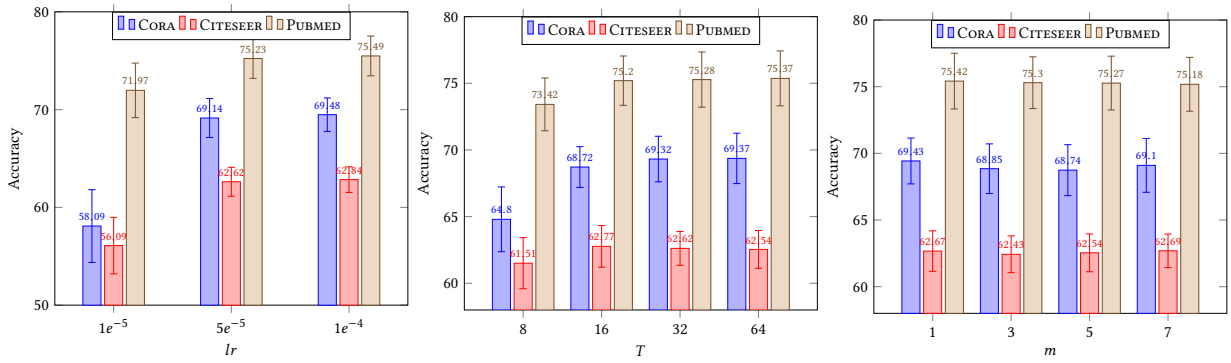

Figure 3: Effects of the Adam initial learning rate lr (left figure), the number T of random walks sampled for each node (central figure), and the number m of iterations in the routing process (right figure) on the validation sets in the inductive setting.

Table 6: Accuracy results on the Cora validation sets w.r.t each data split and each value $m > 1$ of routing iterations for the transductive and inductive settings. Regarding Algorithm 1 when $m > 1$, "Ours" denotes our update rule ($b_i \leftarrow \hat{\mathbf{u}}_{v_i}^{(i)} \cdot \mathbf{e}_v$), while "Sab." denotes the update rule ($b_i \leftarrow b_i + \hat{\mathbf{u}}_{v_i}^{(i)} \cdot \mathbf{e}_v$) originally used by Sabour et al. [19].

Split	Transductive						Inductive					
	m=3		m=5		m=7		m=3		m=5		m=7	
	Ours	Sab.	Ours	Sab.	Ours	Sab.	Ours	Sab.	Ours	Sab.	Ours	Sab.
1st	80.1	80.1	80.2	79.6	79.7	79.3	70.2	70.3	70.2	69.2	70.6	68.3
2nd	79.4	79.6	79.7	78.9	79.7	78.6	66.0	65.9	65.7	64.4	65.6	64.3
3rd	78.5	78.5	78.6	78.6	78.5	78.4	68.2	67.6	68.3	68.4	69.2	67.6
4th	81.3	80.8	81.1	80.1	81.1	79.3	66.5	66.3	66.5	65.4	66.4	65.9
5th	81.9	81.6	81.7	81.5	81.7	80.9	69.4	68.7	69.9	68.5	69.5	68.1
6th	78.6	79.0	78.8	78.7	78.7	78.0	66.7	67.1	66.7	66.2	67.5	65.3
7th	80.1	80.2	80.5	80.0	79.9	79.4	70.4	70.1	70.4	69.9	70.4	68.8
8th	81.8	82.1	82.1	81.5	82.3	81.2	69.6	69.0	68.7	67.8	69.7	67.5
9th	79.3	79.4	79.7	78.1	78.6	77.8	71.2	70.8	71.5	71.7	72.2	70.1
10th	78.8	79.3	79.7	78.9	79.4	78.7	70.3	69.7	69.5	68.8	69.9	68.3
Overall	79.98	**80.06**	**80.21**	79.59	**79.96**	79.16	**68.85**	68.55	**68.74**	68.03	**69.10**	67.42

nodes. Therefore, we propose to use the new update rule ($b_i \leftarrow \hat{\mathbf{u}}_{v_i}^{(i)} \cdot \mathbf{e}_v$) as this new rule generally helps obtain a higher performance for each setup. Table 6 shows a comparison between the accuracy results of these two update rules on the Cora validation sets w.r.t each data split and the number m ($m > 1$) of routing iterations.

4.6 Effects of hyper-parameters

Figures 2 and 3 presents effects of the Adam initial learning rate lr, the number T of random walks sampled for each node and the number m of iterations in the routing process on the validation sets in the transductive and inductive settings respectively. In these experiments, for the 10 data splits of each dataset, we apply the same value of one hyper-parameter and then tune other hyper-parameters.

We find that in general using $lr = 1e^{-4}$ produces the top scores on the validation sets to both transductive and inductive settings. We also find that we generally obtain high accuracies with a high value of T at either 32 or 64. However, there is an exception in the inductive setting, where using $T = 16$ produces the highest accuracy on Citeseer. A possible reason might come from the fact that Citeseer is more sparse than Cora and Pubmed: the average number of neighbors per node on Citeseer is 1.4 which is substantially smaller than 2.0 on Cora and 2.2 on Pubmed.

Furthermore, using $m = 1$ usually obtains the top performances in both the settings. But we also note that the best configurations

of hyper-parameters over 10 data splits are not always relied on using $m = 1$.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we present a new unsupervised embedding model Caps2NE based on the capsule network to learn node embeddings from the graph-structured data. Our proposed Caps2NE aims to effectively use context neighbors in random walks to infer plausible embeddings for target nodes. Experimental results show that Caps2NE obtains state-of-the-art performances on benchmark datasets for the node classification task.

ACKNOWLEDGEMENT

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

REFERENCES

[1] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. 2018. Relational inductive biases, deep learning, and graph networks. *arXiv:1806.01261* (2018).

[2] Bobby-Joe Breitkreutz, Chris Stark, Teresa Reguly, Lorrie Boucher, Ashton Breitkreutz, Michael Livstone, Rose Oughtred, Daniel Lackner, JÃijrg BÃďhler, Valerie Wood, Kara Dolinski, and Mike Tyers. 2008. The BioGRID interaction database: 2008 update. 36 (2008), D637–40.

[3] Hongyun Cai, Vincent W Zheng, and Kevin Chang. 2018. A comprehensive survey of graph embedding: problems, techniques and applications. *IEEE Transactions on Knowledge and Data Engineering* 30, 9 (2018), 1616–1637.

[4] Haochen Chen, Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2018. A Tutorial on Network Embeddings. *arXiv:1808.02590* (2018).

[5] Zhiyong Cui, Kristian Henrickson, Ruimin Ke, and Yinhai Wang. 2018. High-Order Graph Convolutional Recurrent Neural Network: A Deep Learning Framework for Network-Scale Traffic Learning and Forecasting. arXiv:arXiv:1802.07007

[6] Alberto Garcia Duran and Mathias Niepert. 2017. Learning Graph Representations with Embedding Propagation. In *Advances in Neural Information Processing Systems*. 5119–5130.

[7] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research* 9 (2008), 1871–1874.

[8] Aditya Grover and Jure Leskovec. 2016. Node2Vec: Scalable Feature Learning for Networks. In *The ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 855–864.

[9] Junliang Guo, Linli Xu, and Enhong Chen. 2018. SPINE: Structural Identity Preserved Inductive Network Embedding. *arXiv:1802.03984* (2018).

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

[11] Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015. On Using Very Large Target Vocabulary for Neural Machine Translation. In *ACL*. 1–10.

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

[13] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations (ICLR)*.

[14] Matt Mahoney. 2011. Large text compression benchmark. http://www.mattmahoney.net/text/text.html

[15] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *Advances in Neural Information Processing Systems*. 3111–3119.

[16] Galileo Mark Namata, Ben London, Lise Getoor, and Bert Huang. 2012. Query-driven Active Surveying for Collective Classification. In *Workshop on Mining and Learning with Graphs*.

[17] Dai Quoc Nguyen, Tu Dinh Nguyen, and Dinh Phung. 2020. A Self-Attention Network based Node Embedding Model. In *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*.

[18] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *The ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 701–710.

[19] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. 2017. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*. 3859–3869.

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

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

[22] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. 2018. Billion-scale Commodity Embedding for E-commerce Recommendation in Alibaba. In *The ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 839–848.

[23] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. 2016. Revisiting Semi-supervised Learning with Graph Embeddings. In *International Conference on Machine Learning*. 40–48.

[24] 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 *The ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 974–983.

[25] R. Zafarani and H. Liu. 2009. Social Computing Data Repository at ASU. http://socialcomputing.asu.edu

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

3.2.3 Quaternion graph neural networks

• **Dai Quoc Nguyen**, Tu Dinh Nguyen and Dinh Phung. Quaternion Graph Neural Networks. *arXiv preprint arXiv:2008.05089*.

Contribution. We propose our quaternion graph neural networks (QGNN) (Nguyen et al., 2020b) to generalise GCNs (Kipf and Welling, 2017) 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 significantly. Besides, any slight change in any of the quaternion input components results in an entirely different output (Parcollet et al., 2019), 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. QGNN obtains superior accuracies compared to GCN and up-to-date baseline models for the tasks of graph classification, semi-supervised node classification, and text (node) classification.

Furthermore, an extended abstract of the submitted paper has been accepted to the NeurIPS 2020 Workshop on Differential Geometry meets Deep Learning (DiffGeo4DL). The code is available at: `https://github.com/daiquocnguyen/QGNN`.

Quaternion Graph Neural Networks

Dai Quoc Nguyen[1] **Tu Dinh Nguyen**[2] **Dinh Phung**[1]

[1]Monash University, Australia, {dai.nguyen,dinh.phung}@monash.edu
[2]nguyendinhtu@gmail.com

Abstract

Recently, graph neural networks (GNNs) become a principal research direction to learn low-dimensional continuous embeddings for nodes and graphs to predict node and graph labels, respectively. Most of the existing GNNs learn node and graph embeddings within the Euclidean vector space. However, for complex graphs such as protein interaction networks and social networks, the learned Euclidean embeddings have high distortion. Furthermore, when increasing the number of hidden layers, the existing GNNs are not working very efficiently anymore since the number of parameters grows quickly. 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 learn 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.

1 INTRODUCTION

Graph representation learning is one of the most important topics for graph-structured data [Hamilton et al., 2017b, Zhou et al., 2018, Wu et al., 2019b, Zhang et al., 2020], 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 [Scarselli et al., 2009, Hamilton et al., 2017b, Wu et al., 2019b, Zhang et al., 2020]. GNNs use an AGGREGATION function [Kipf and Welling, 2017, Hamilton et al., 2017a, Veličković et al., 2018, Nguyen et al., 2019] over neighbors of each node to update its vector representation iteratively, and then apply a graph-level READOUT pooling function to obtain graph embeddings [Gilmer et al., 2017, Zhang et al., 2018, Ying et al., 2018, Verma and Zhang, 2018, Xu et al., 2019]. We note that the GNNs have been producing state-of-the-art accuracy performances for node and graph classification tasks.

It is noted that most of the existing GNNs learn embeddings for nodes and graphs within the Euclidean vector space. However, for complex graphs such as protein interaction networks and social networks, the learned Euclidean embeddings have high distortion [Chami et al., 2019]. It also has been noted in [Xu et al., 2019, Pei et al., 2020] 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, when increasing the number of hidden layers, the existing GNNs [Kipf and Welling, 2017, Hamilton et al., 2017a, Veličković et al., 2018, Xu et al., 2019] are not working very efficiently anymore since the number of parameters grows quickly. 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 learn better graph representations and reduce the number of model parameters?

To this end, we present a novel approach to learn node and graph embeddings within the Quaternion space. Some quaternion-based methods have been applied in image classification [Gaudet and Maida, 2018, Zhu et al., 2018], speech recognition [Parcollet et al., 2018, 2019b] and knowledge

graph [Zhang et al., 2019]. The closely related work is applying quaternion networks for natural language processing [Tay et al., 2019]. However, to the best of our knowledge, our work is the first to investigate quaternion embeddings for general graphs with diverse and different structures, hence requiring a different solution approach.

In general, our strategy can be flexibly applied to existing AGGREGATION functions such as in Graph Convolutional Network (GCN) [Kipf and Welling, 2017]. Note that GCN is one of the simplest yet state-of-the-art GNNs for the semi-supervised node classification task. As noted in [Xu et al., 2019, Chen et al., 2019], GCN also outperforms GraphSAGE [Hamilton et al., 2017a] and GAT [Veličković et al., 2018], and produces competitive accuracies compared to other up-to-date GNN models [Xu et al., 2019, Du et al., 2019] for the graph classification task. Therefore, *we propose our quaternion graph neural networks (QGNN) to generalize GCNs within the Quaternion space*. Note that the same procedure can be applied for other GNNs such as Simple Graph Convolution [Wu et al., 2019a] or Graph Isomorphism Network [Xu et al., 2019].

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 significantly. Besides, any slight change in any of the quaternion input components results in an entirely different output [Parcollet et al., 2019a], leading to a different performance. This enhances capturing potential relations within each hidden quaternion layer and between the different hidden layers to increase the embedding quality.

Graphs are ubiquitous in science, engineering, and real-life applications. They are fundamental to, for example, the study of disease outbreak, human dynamics, biological networks, social networks, information retrieval, to name a few. Our work provides basic building blocks for such studies, hence applicable and relevant to many research problems beyond computer science and machine learning. As the work represents a fundamental research problem in representing graphs, we believe it does not put anyone at disadvantages. Our proposed QGNN has demonstrated to achieve superior performances through extensive experimental evaluations, making state-of-the-art performances on a wide range of benchmark datasets for three downstream tasks.

Contributions. In summary, our main contributions can be highlighted as follows:

- We propose to learn the node and graph embeddings within the Quaternion space and introduce our quaternion graph neural networks (QGNN) to generalize GCNs within the Quaternion space. QGNN can reduce the model size and improve the embedding quality.
- We evaluate the effectiveness of our proposed QGNN on a variety of well-known benchmark datasets for the tasks of graph classification, semi-supervised node classification, and text (node) classification.
- Experimental results show that QGNN obtains superior accuracies compared to GCN and up-to-date baseline models, especially producing state-of-the-art accuracies on these benchmark datasets.

2 RELATED WORK

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

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 $\{\mathsf{y}_m\}_{m=1}^M \subseteq \mathcal{Y}$, 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 [Kipf and Welling, 2017, Hamilton et al., 2017a, Veličković et al., 2018]. After that, GNNs use a READOUT pooling function to obtain the vector representation of the entire graph [Gilmer et al., 2017, Zhang et al., 2018, Ying et al., 2018, Verma and Zhang, 2018, Xu et al., 2019]. Mathematically, given a graph $\mathcal{G}$, we formulate GNNs as follows:

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

$$\mathbf{e}_\mathcal{G} = \text{READOUT}\left(\left\{\left\{\mathbf{h}_\mathsf{v}^{(l)}\right\}_{l=0}^{L}\right\}_{\mathsf{v}\in\mathcal{V}}\right) \quad (2)$$

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) [Kipf and Welling, 2017] as:

$$\mathbf{h}_\mathsf{v}^{(l+1)} = \mathrm{g}\left(\sum_{\mathsf{u}\in\mathcal{N}_\mathsf{v}\cup\{\mathsf{v}\}} a_{\mathsf{v},\mathsf{u}}\mathbf{W}^{(l)}\mathbf{h}_\mathsf{u}^{(l)}\right), \forall \mathsf{v} \in \mathcal{V} \quad (3)$$

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.

While 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] uses a more powerful AGGREGATION function based on a multi-layer perceptron (MLP) network of two fully-connected layers as:

$$\mathbf{h}_{\mathsf{v}}^{(l+1)} = \mathsf{MLP}^{(l)}\left(\sum_{\mathsf{u}\in\mathscr{N}_{\mathsf{v}}\cup\{\mathsf{v}\}}\mathbf{h}_{\mathsf{u}}^{(l)}\right), \forall \mathsf{v}\in\mathscr{V} \tag{4}$$

Following [Xu et al., 2018, 2019], we also employ a concatenation over the vector representations of node v at the different layers to construct the node embedding $\mathbf{e}_{\mathsf{v}}$ as:

$$\mathbf{e}_{\mathsf{v}} = \left[\mathbf{h}_{\mathsf{v}}^{(1)};\mathbf{h}_{\mathsf{v}}^{(2)};...;\mathbf{h}_{\mathsf{v}}^{(L)}\right], \forall \mathsf{v}\in\mathscr{V} \tag{5}$$

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 [Zhang et al., 2018], hierarchical pooling [Cangea et al., 2018], and differentiable pooling [Ying et al., 2018]. As the sum pooling often produces competitive performances [Xu et al., 2019], we utilize the sum pooling to obtain the embedding $\mathbf{e}_{\mathscr{G}}$ of the entire graph $\mathscr{G}$ as:

$$\mathbf{e}_{\mathscr{G}} = \sum_{\mathsf{v}\in\mathscr{V}}\mathbf{e}_{\mathsf{v}} = \sum_{\mathsf{v}\in\mathscr{V}}\left[\mathbf{h}_{\mathsf{v}}^{(1)};\mathbf{h}_{\mathsf{v}}^{(2)};...;\mathbf{h}_{\mathsf{v}}^{(L)}\right] \tag{6}$$

3 QUATERNION GRAPH NEURAL NETWORKS

3.1 QUATERNION BACKGROUND

The hyper-complex vector space has recently been considered on the Quaternion space [Hamilton, 1844] 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 [Zhu et al., 2018, Gaudet and Maida, 2018], speech recognition [Parcollet et al., 2018, 2019b], knowledge graph [Zhang et al., 2019], and natural language processing [Tay et al., 2019].

We provide key notations and operations related to the Quaternion space required for our later development. Additional details can further be found in [Parcollet et al., 2019a].

A quaternion $q\in\mathbb{H}$ is a hyper-complex number consisting of one real and three separate imaginary components [Hamilton, 1844] defined as:

$$q = q_r + q_i\mathbf{i} + q_j\mathbf{j} + q_k\mathbf{k} \tag{7}$$

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 $q\in\mathbb{H}^n$ is defined as:

$$q = q_r + q_i\mathbf{i} + q_j\mathbf{j} + q_k\mathbf{k} \tag{8}$$

where $q_r, q_i, q_j, q_k \in \mathbb{R}^n$. The operations for the Quaternion algebra are defined as follows:

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

$$q + p = (q_r + p_r) + (q_i + p_i)\mathbf{i} + (q_j + p_j)\mathbf{j} + (q_k + p_k)\mathbf{k} \tag{9}$$

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

$$\|q\| = \sqrt{q_r^2 + q_i^2 + q_j^2 + q_k^2} \tag{10}$$

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

$$q^{\triangleleft} = \frac{q}{\|q\|} \tag{11}$$

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

$$\lambda q = \lambda q_r + \lambda q_i\mathbf{i} + \lambda q_j\mathbf{j} + \lambda q_k\mathbf{k} \tag{12}$$

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

$$q^* = q_r - q_i\mathbf{i} - q_j\mathbf{j} - q_k\mathbf{k} \tag{13}$$

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

$$\begin{aligned} q\otimes p \;=\; & (q_r p_r - q_i p_i - q_j p_j - q_k p_k) \\ + \; & (q_i p_r + q_r p_i - q_k p_j + q_j p_k)\mathbf{i} \\ + \; & (q_j p_r + q_k p_i + q_r p_j - q_i p_k)\mathbf{j} \\ + \; & (q_k p_r - q_j p_i + q_i p_j + q_r p_k)\mathbf{k} \end{aligned} \tag{14}$$

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

$$q\otimes p = \begin{bmatrix}1\\ \mathbf{i}\\ \mathbf{j}\\ \mathbf{k}\end{bmatrix}^{\top}\begin{bmatrix} q_r & -q_i & -q_j & -q_k \\ q_i & q_r & -q_k & q_j \\ q_j & q_k & q_r & -q_i \\ q_k & -q_j & q_i & q_r \end{bmatrix}\begin{bmatrix} p_r \\ p_i \\ p_j \\ p_k\end{bmatrix} \tag{15}$$

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 q and p as:

$$[q;p] = [q_r;p_r] + [q_i;p_i]\,\mathbf{i} + \left[q_j;p_j\right]\mathbf{j} + [q_k;p_k]\,\mathbf{k} \tag{16}$$

3.2 THE PROPOSED QGNN

Note that we use the superscript $^{\mathcal{Q}}$ to denote the Quaternion space in this section, e.g., $\mathbf{h}_{\mathsf{v}}^{(0),\mathcal{Q}}$ is the quaternion feature vector of node v.

It is worth to note that the existing GNNs [Kipf and Welling, 2017, Hamilton et al., 2017a, Veličković et al., 2018, Xu et al., 2019] are not working very efficiently anymore since the number of parameters grows quickly when increasing the number of hidden layers. Furthermore, as mentioned in [Xu et al., 2019, Pei et al., 2020], 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 [Chami et al., 2019] that, for complex graphs with scale-free and hierarchical structures such as protein interaction networks and social networks, the learned Euclidean embeddings have high distortion. 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.

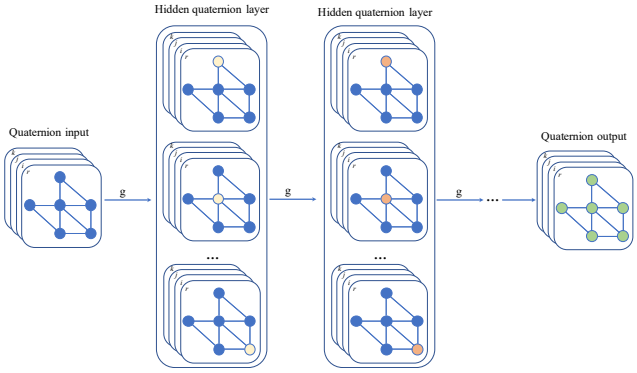

Figure 1: Illustration of our QGNN.

Specifically, as illustrated in Figure 1, the AGGREGATION function in our proposed QGNN is defined as:[1]

$$\mathbf{h}_{\mathsf{v}}^{(l+1),\mathcal{Q}} = \mathrm{g}\left(\sum_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}\cup\{\mathsf{v}\}} a_{\mathsf{v},\mathsf{u}}\mathbf{W}^{(l),\mathcal{Q}} \otimes \mathbf{h}_{\mathsf{u}}^{(l),\mathcal{Q}}\right), \forall \mathsf{v} \in \mathscr{V} \quad (17)$$

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

$$\mathrm{g}(q) = \mathrm{g}(q_r) + \mathrm{g}(q_i)\mathbf{i} + \mathrm{g}(q_j)\mathbf{j} + \mathrm{g}(q_k)\mathbf{k} \quad (18)$$

[1] In practice, similar to GCN, we also implement QGNN efficiently using sparse matrix multiplications.

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

$$\mathbf{h}_{\mathsf{u}}^{(l),\mathcal{Q}} = \mathbf{h}_{\mathsf{u},r}^{(l)} + \mathbf{h}_{\mathsf{u},i}^{(l)}\mathbf{i} + \mathbf{h}_{\mathsf{u},j}^{(l)}\mathbf{j} + \mathbf{h}_{\mathsf{u},k}^{(l)}\mathbf{k} \quad (19)$$

$$\mathbf{W}^{(l),\mathcal{Q}} = \mathbf{W}_r^{(l)} + \mathbf{W}_i^{(l)}\mathbf{i} + \mathbf{W}_j^{(l)}\mathbf{j} + \mathbf{W}_k^{(l)}\mathbf{k} \quad (20)$$

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 $\mathbf{W}^{(l),\mathcal{Q}}$ and $\mathbf{h}_{\mathsf{u}}^{(l),\mathcal{Q}}$ derived from Equation 15 as:

$$\mathbf{W}^{(l),\mathcal{Q}} \otimes \mathbf{h}_{\mathsf{u}}^{(l),\mathcal{Q}} = \begin{bmatrix}1\\ \mathbf{i}\\ \mathbf{j}\\ \mathbf{k}\end{bmatrix}^{\top} \begin{bmatrix}\mathbf{W}_r^{(l)} & -\mathbf{W}_i^{(l)} & -\mathbf{W}_j^{(l)} & -\mathbf{W}_k^{(l)}\\ \mathbf{W}_i^{(l)} & \mathbf{W}_r^{(l)} & -\mathbf{W}_k^{(l)} & \mathbf{W}_j^{(l)}\\ \mathbf{W}_j^{(l)} & \mathbf{W}_k^{(l)} & \mathbf{W}_r^{(l)} & -\mathbf{W}_i^{(l)}\\ \mathbf{W}_k^{(l)} & -\mathbf{W}_j^{(l)} & \mathbf{W}_i^{(l)} & \mathbf{W}_r^{(l)}\end{bmatrix} \begin{bmatrix}\mathbf{h}_{\mathsf{u},r}^{(l)}\\ \mathbf{h}_{\mathsf{u},i}^{(l)}\\ \mathbf{h}_{\mathsf{u},j}^{(l)}\\ \mathbf{h}_{\mathsf{u},k}^{(l)}\end{bmatrix} \quad (21)$$

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 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]. This parameter reduction can also be applied for other existing AGGREGATION functions such as in GIN-0 (Equation 4) or Simple Graph Convolution [Wu et al., 2019a].

Note that the quaternion components of $\mathbf{W}^{(l),\mathcal{Q}}$ 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 $\mathbf{h}_{\mathsf{u}}^{(l),\mathcal{Q}}$, we get an entirely different output [Parcollet et al., 2019a], 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. Our qualitative and experimental results verify that this learning helps our QGNN to outperform GCN using similar architectures.

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

$$\mathbf{x}_{\mathsf{v}}^{(L)} = \mathrm{VEC}\left(\mathbf{h}_{\mathsf{v}}^{(L),\mathcal{Q}}\right) = \left[\mathbf{h}_{\mathsf{v},r}^{(L)}; \mathbf{h}_{\mathsf{v},i}^{(L)}; \mathbf{h}_{\mathsf{v},j}^{(L)}; \mathbf{h}_{\mathsf{v},k}^{(L)}\right] \quad (22)$$

where 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 prediction layer followed by a softmax layer as follows:

$$\hat{\mathbf{y}}_{\mathsf{v}} = \mathsf{softmax}\left(\sum_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}\cup\{\mathsf{v}\}} a_{\mathsf{v},\mathsf{u}}\mathbf{W}_1\mathbf{x}_{\mathsf{u}}^{(L)}\right), \forall \mathsf{v}\in\mathcal{V} \quad (23)$$

QGNN for graph classification. Following [Xu et al., 2019] as shown in Equations 5 and 6, we obtain the quaternion embedding $\mathbf{e}_{\mathcal{G}}^{\mathcal{Q}}$ of the entire graph $\mathcal{G}$ as:

$$\mathbf{e}_{\mathcal{G}}^{\mathcal{Q}} = \sum_{\mathsf{v}\in\mathcal{V}} \mathbf{e}_{\mathsf{v}}^{\mathcal{Q}} = \sum_{\mathsf{v}\in\mathcal{V}} \left[\mathbf{h}_{\mathsf{v}}^{(1),\mathcal{Q}}; \mathbf{h}_{\mathsf{v}}^{(2),\mathcal{Q}}; ...; \mathbf{h}_{\mathsf{v}}^{(L),\mathcal{Q}}\right] \quad (24)$$

$$\mathbf{x}_{\mathcal{G}} = \textsc{Vec}\left(\mathbf{e}_{\mathcal{G}}^{\mathcal{Q}}\right) \quad (25)$$

To perform the graph classification task, we also use the VEC(.) to vectorize $\mathbf{e}_{\mathcal{G}}^{\mathcal{Q}}$ to obtain the final graph embedding $\mathbf{x}_{\mathcal{G}}$, which is fed to a single fully-connected layer followed by the softmax layer to predict the graph label as:

$$\hat{\mathbf{y}}_{\mathcal{G}} = \mathsf{softmax}\left(\mathbf{W}_2\mathbf{x}_{\mathcal{G}} + \mathbf{b}\right) \quad (26)$$

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

Parameter initialization. Parcollet et al. [2019b] and Zhang et al. [2019] used a specialized scheme to initialize the parameters in the quaternion weight matrices, while Zhu et al. [2018] and Tay et al. [2019] applied the Glorot initialization [Glorot and Bengio, 2010] that is also used in previous GNN works such as GCN [Kipf and Welling, 2017]; hence we use the Glorot initialization for a fair comparison with the previous works.

The quaternion feature vectors $\mathbf{h}_{\mathsf{v}}^{(0),\mathcal{Q}}$. As shown in Equation 17, we consider how to initialize the quaternion feature vectors $\mathbf{h}_{\mathsf{v}}^{(0),\mathcal{Q}}$ 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. 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 $\mathbf{h}_{\mathsf{v}}^{(0),\mathcal{Q}}$ as:

$$\mathbf{h}_{\mathsf{v},r}^{(0)} = \mathbf{h}_{\mathsf{v},i}^{(0)} = \mathbf{h}_{\mathsf{v},j}^{(0)} = \mathbf{h}_{\mathsf{v},k}^{(0)} = \mathbf{h}_{\mathsf{v}} \quad (27)$$

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

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.

4 EXPERIMENTAL SETUP AND RESULTS

We conduct experiments to evaluate our proposed QGNN on the tasks of graph classification, semi-supervised node classification, and text (node) classification. Table 1 reports the statistics of benchmark datasets used for the three tasks.

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).

Table 1: 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

4.1 GRAPH CLASSIFICATION

Datasets. We use well-known datasets that are three social network datasets (consisting of COLLAB, IMDB-B and

Table 2: Graph classification accuracies (%). The best scores are in **bold**.

Model	COLLAB	IMDB-B	IMDB-M	DD	PROTEINS	MUTAG	PTC
GK [2009]	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 [2011]	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 [2015]	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 [2018]	73.93 ± 1.94	74.45 ± 5.83	51.54 ± 3.61	71.51 ± 4.02	–	87.87 ± 9.76	–
PSCN [2016]	72.60 ± 2.15	71.00 ± 2.29	45.23 ± 2.84	77.12 ± 2.41	75.89 ± 2.76	$\mathbf{92.63 \pm 4.21}$	62.29 ± 5.68
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
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	–
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]	$\mathbf{81.50 \pm 2.42}$	73.00 ± 4.35	51.80 ± 5.16	78.78 ± 3.49	76.46 ± 4.06	90.84 ± 7.22	–
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
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
QGNN	81.36 ± 1.31	$\mathbf{77.56 \pm 2.45}$	$\mathbf{53.78 \pm 3.83}$	$\mathbf{79.92 \pm 3.54}$	$\mathbf{78.47 \pm 3.30}$	92.59 ± 3.59	$\mathbf{69.92 \pm 2.59}$

IMDB-M) [Yanardag and Vishwanathan, 2015] and four bioinformatics datasets (consisting of DD, MUTAG, PROTEINS, and PTC). The social network datasets do not have available node features; thus, we follow [Niepert et al., 2016, Zhang et al., 2018] to use node degrees as features on these datasets.

Evaluation protocol. We follow [Xu et al., 2019, Xinyi and Chen, 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 published in the original papers or reported in [Ivanov and Burnaev, 2018, Verma and Zhang, 2018, Xinyi and Chen, 2019, Chen et al., 2019, Seo et al., 2019, Xu et al., 2019].

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 [Kingma and Ba, 2015] 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.

Experimental results. Table 2 presents the graph classification 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. In particular, QGNN outperforms all baseline models on IMDB-B, 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, compared to GCN, the obtained results demonstrate the effectiveness of QGNN to generalize GCN within the Quaternion space for the graph classification task.

4.2 NODE CLASSIFICATION

Datasets. We use three benchmark datasets consisting of CORA, CITESEER [Sen et al., 2008] and PUBMED [Namata et al., 2012] 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 [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 average accuracy on the test sets across the 10 data splits.

Training protocol. The architecture used by 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 1-layer QGNN followed by a prediction layer and then a softmax layer (referring to our QGNN for node classification as mentioned in Equation 23). 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. Similarity, we also provide the accuracy results of Hyperbolic Graph Convolutional Neural Networks (HGCN) [Chami et al., 2019] following these evaluation and training protocols.

Table 3: Node classification accuracies (%).

Dataset	GAT	GCN	HGCN	QGNN
CORA	86.37	85.77	86.09 ± 1.34	$\mathbf{87.48 \pm 1.08}$
CITESEER	74.32	73.68	74.84 ± 2.03	$\mathbf{76.03 \pm 1.89}$
PUBMED	87.62	**88.13**	87.13 ± 0.89	87.65 ± 0.47

Experimental results. Table 3 presents the node classification accuracies, where the results of GCN and GAT are also taken from [Pei et al., 2020].[2] As mentioned in the evaluation protocol, we use exactly the same experimental settings used in [Pei et al., 2020], 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, GAT and HGCN on CORA and CITESEER, and produces competitive results on PUBMED.

4.3 TEXT (NODE) CLASSIFICATION

Datasets. Yao et al. [2019] 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 [Yao et al., 2019] 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 [Yao et al., 2019] to report the mean and standard deviation over 10 runs.

Training protocol. Yao et al. [2019] also used 2-layer GCN to perform the text (node) classification with the hidden size of 200. Similarity, we utilize 1-layer QGNN followed by a prediction layer and then a softmax layer, using the quaternion hidden size of 50. We also use the same hyperparameter settings; those are 0.02 Adam learning rate and 200 training epochs.

[2] Pei et al. [2020] did not report the standard deviation.

Table 4: Text (node) classification accuracies (%).

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

Experimental results. Table 4 presents the text (node) classification accuracies for GCN and our QGNN, wherein the results of GCN are also taken from [Yao et al., 2019]. As mentioned above, we also use exactly the same experimental settings used in [Yao et al., 2019]. In general, QGNN works better than GCN on four datasets, except MR.

4.4 COMPARED WITH GCN

Table 5: Number of learnable parameters for GCN and our QGNN on IMDB-B when using the hidden size of 256 in GCN.

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

Model size. We report the total number of learnable parameters for the GCN and QGNN using the similar architectures on IMDB-B in Table 5, wherein the GCN's hidden size is set to 256; accordingly, the QGNN's hidden size is 64. It is worth to note that GCN and QGNN have the same complexity.

As mentioned in Equation 27, we set the feature vectors $\mathbf{h}_v$ to the four components of the quaternion feature vectors $\mathbf{h}_v^{(0),Q}$ 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.

Qualitative result. To qualitatively demonstrate the effectiveness of our QGNN, we use t-SNE [Maaten and Hinton, 2008] to visualize the node representations learned at the first layer of GCN and QGNN on CORA in Figure 2, where colors denote the class labels. We concatenate the four components of the quaternion node embeddings to have the

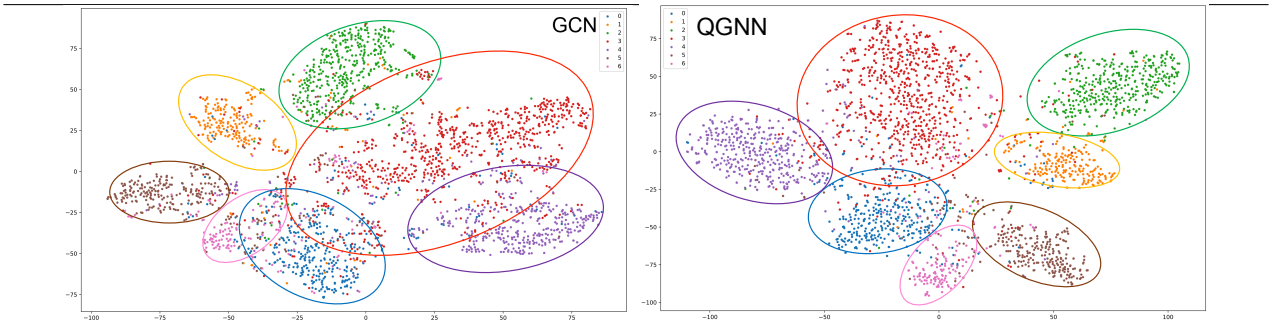

Figure 2: Visualization of node representations on CORA.

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.

Discussion. Our QGNN generalizes GCN within the Quaternion space. Note that we use the similar network architectures and the same optimal hyper-parameters directly taken from [Pei et al., 2020] and [Yao et al., 2019] for the tasks of node classification and text (node) classification respectively, i.e., we do not need to tune our QGNN's hyper-parameters for these two tasks and our QGNN still works better than GCN. In general, the performance gains of QGNN over GCN on the three tasks indicate that QGNN strengthens the node and graph embeddings, as visualized in Figure 2.

5 CONCLUSION

In this paper, we aim to increase the graph representation quality and reduce the number of model parameters. 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.

REFERENCES

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 *Advances in Neural Information Processing Systems*, pages 4869–4880, 2019.

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

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.

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.

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.

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.

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

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

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

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

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

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

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

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.

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

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning Convolutional Neural Networks for Graphs. In *International Conference on Machine Learning*, 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 *International Conference on Learning Representations (ICLR)*, 2019b.

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.

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.

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 Perraudin. Discriminative structural graph classification. *arXiv:1905.13422*, 2019.

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

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.

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.

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

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

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning*, 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. *International Conference on Learning Representations (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 *International Conference on Machine Learning*, pages 5453–5462, 2018.

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

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *SIGKDD*, pages 1365–1374, 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, 2020.

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 *Advances in Neural Information Processing Systems*, pages 2731–2741, 2019.

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.

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

Chapter 4

Knowledge Graph Embeddings

Contents

In this chapter, Section 4.1 presents an introduction and motivation for knowledge graph embeddings, and the details of our research contributions are described in Section 4.2.

4.1 Introduction

Knowledge graphs (KGs) are constructed to represent the genuine relationships among 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 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., 2013a).

Early embedding models such as TransE (Bordes et al., 2013), TransH (Wang et al., 2014), TransR (Lin et al., 2015b), TransD (Ji et al., 2015), DISTMULT (Yang et al., 2015) and ComplEx (Trouillon et al., 2016) often employ simple linear operators such as addition, subtraction and multiplication. Since ConvE (Dettmers et al., 2018) has successfully adapted convolutional neural networks (LeCun et al., 1998) to produce the triple scores and obtain state-of-the-art results for knowledge graph completion, we realise a potential strategy of advancing deep neural networks for knowledge graph embeddings. To this end, we present two new KG embedding models, named ***ConvKB*** and ***CapsE***, whose architectures are based on convolutional neural networks and capsule networks (Sabour et al., 2017), respectively. The details of these two models are given in Section 4.2.1.

While most of the existing embedding models have worked in the Euclidean vector space, several works have moved beyond that to the complex vector space such as ComplEx (Trouillon et al., 2016) and RotatE (Sun et al., 2019), the hyperbolic space such as MuRP (Balažević et al., 2019a) and AttH (Chami et al., 2020), and the Quaternion space as in QuatE (Zhang et al., 2019). These models, however, encounter a common problem of struggling to strengthen the relation-aware correlations between the head and tail entities. Our key contribution is to overcome this limitation by integrating relation-aware rotations to increase the correlations between the entities, resulting in another new embedding model, named ***QuatRE***. The details of our QuatRE are described in Section 4.2.2.

It is worth noting that the knowledge graph completion task is commonly used to evaluate the KG embedding models, wherein the goal is 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). But, for other applications of triple classification (Socher et al., 2013a) and search personalisation (Vu et al., 2017), the existing models often suffer from a limitation of generalisation capability in which they just simply memorise valid triples to predict new ones. Hence, we present ***R-MeN*** – a new KG embedding model to capture and encode the potential dependencies among relations and entities for these two applications. The details of our proposed R-MeN are presented in Section 4.2.3.

4.2 Research Contribution

4.2.1 NAACL-HLT 2018 & SWJ 2019 & NAACL-HLT 2019 - Deep knowledge graph embedding models

• **Dai Quoc Nguyen**, Tu Dinh Nguyen, Dat Quoc Nguyen and Dinh Phung. **2018**. A Novel Embedding Model for Knowledge Base Completion Based on Convolutional Neural Network. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2018)*, pages 327-333.

• **Dai Quoc Nguyen**, Dat Quoc Nguyen, Tu Dinh Nguyen and Dinh Phung. **2019**. A Convolutional Neural Network-based Model for Knowledge Base Completion and Its Application to Search Personalisation. *Semantic Web*, 10(5):947-960, 2019. DOI: 10.3233/SW-180318. (SCIE, JCR IF: 3.524).

• **Dai Quoc Nguyen**, Thanh Vu, Tu Dinh Nguyen, Dat Quoc Nguyen and Dinh Phung. **2019**. A Capsule Network-based Embedding Model for Knowledge Graph Completion and Search Personalisation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2019)*, pages 2180-2189.

Contribution. Our proposed ConvKB (Nguyen et al., 2018) utilises a convolutional layer on the 3-column embedding matrix of the input triple to produce feature maps. These feature maps are then concatenated into a single vector, which is computed with a weight vector to return the triple score. Our ConvKB works better than previous embedding models on benchmark datasets for the knowledge graph completion task. Moreover, in the journal paper (Nguyen et al., 2019a), we extend and demonstrate the effectiveness of the ConvKB for triple classification and search personalisation tasks, where our model outperforms up-to-date baselines and produces state-of-the-art performance. The code is available at: `https://github.com/daiquocnguyen/ConvKB`.

Instead of using the vector concatenation over the feature maps in ConvKB, our

proposed CapsE (Nguyen et al., 2019c) leverages a capsule network (Sabour et al., 2017) to reconstruct the feature maps into corresponding capsules, which are then routed to another capsule to produce a continuous vector. The length of this vector is used to compute the triple score. Our CapsE obtains better performance than up-to-date embedding models for the tasks of knowledge graph completion and search personalisation. The code is available at: `https://github.com/daiquocnguyen/CapsE`.

A Novel Embedding Model for Knowledge Base Completion Based on Convolutional Neural Network

Dai Quoc Nguyen[1], **Tu Dinh Nguyen**[1], **Dat Quoc Nguyen**[2], **Dinh Phung**[1]
[1] Deakin University, Geelong, Australia, Centre for Pattern Recognition and Data Analytics
{dai.nguyen,tu.nguyen,dinh.phung}@deakin.edu.au
[2] The University of Melbourne, Australia
dqnguyen@unimelb.edu.au

Abstract

In this paper, we propose a novel embedding model, named ConvKB, for knowledge base completion. Our model ConvKB advances state-of-the-art models by employing a convolutional neural network, so that it can capture global relationships and transitional characteristics between entities and relations in knowledge bases. In ConvKB, each triple *(head entity, relation, tail entity)* is represented as a 3-column matrix where each column vector represents a triple element. This 3-column matrix is then fed to a convolution layer where multiple filters are operated on the matrix to generate different feature maps. These feature maps are then concatenated into a single feature vector representing the input triple. The feature vector is multiplied with a weight vector via a dot product to return a score. This score is then used to predict whether the triple is valid or not. Experiments show that ConvKB achieves better link prediction performance than previous state-of-the-art embedding models on two benchmark datasets WN18RR and FB15k-237.

1 Introduction

Large-scale knowledge bases (KBs), such as YAGO (Suchanek et al., 2007), Freebase (Bollacker et al., 2008) and DBpedia (Lehmann et al., 2015), are usually databases of triples representing the relationships between entities in the form of fact *(head entity, relation, tail entity)* denoted as *(h, r, t)*, e.g., *(Melbourne, cityOf, Australia)*. These KBs are useful resources in many applications such as semantic searching and ranking (Kasneci et al., 2008; Schuhmacher and Ponzetto, 2014; Xiong et al., 2017), question answering (Zhang et al., 2016; Hao et al., 2017) and machine reading (Yang and Mitchell, 2017). However, the KBs are still incomplete, i.e., missing a lot of valid triples (Socher et al., 2013; West et al., 2014). Therefore, much research work has been devoted towards *knowledge base completion* or *link prediction* to predict whether a triple *(h, r, t)* is valid or not (Bordes et al., 2011).

Many embedding models have proposed to learn vector or matrix representations for entities and relations, obtaining state-of-the-art (SOTA) link prediction results (Nickel et al., 2016a). In these embedding models, valid triples obtain lower implausibility scores than invalid triples. Let us take the well-known embedding model TransE (Bordes et al., 2013) as an example. In TransE, entities and relations are represented by k-dimensional vector embeddings. TransE employs a transitional characteristic to model relationships between entities, in which it assumes that if *(h, r, t)* is a valid fact, the embedding of head entity h plus the embedding of relation r should be close to the embedding of tail entity t, i.e. $\boldsymbol{v}_h + \boldsymbol{v}_r \approx \boldsymbol{v}_t$ (here, $\boldsymbol{v}_h$, $\boldsymbol{v}_r$ and $\boldsymbol{v}_t$ are embeddings of h, r and t respectively). That is, a TransE score $\|\boldsymbol{v}_h + \boldsymbol{v}_r - \boldsymbol{v}_t\|_p$ of the valid triple *(h, r, t)* should be close to 0 and smaller than a score $\|\boldsymbol{v}_{h'} + \boldsymbol{v}_{r'} - \boldsymbol{v}_{t'}\|_p$ of an invalid triple *(h', r', t')*. The transitional characteristic in TransE also implies the global relationships among same dimensional entries of $\boldsymbol{v}_h$, $\boldsymbol{v}_r$ and $\boldsymbol{v}_t$.

Other transition-based models extend TransE to additionally use projection vectors or matrices to translate head and tail embeddings into the relation vector space, such as: TransH (Wang et al., 2014), TransR (Lin et al., 2015b), TransD (Ji et al., 2015), STransE (Nguyen et al., 2016b) and TranSparse (Ji et al., 2016). Furthermore, DISTMULT (Yang et al., 2015) and ComplEx (Trouillon et al., 2016) use a tri-linear dot product to compute the score for each triple. Recent research

has shown that using relation paths between entities in the KBs could help to get contextual information for improving KB completion performance (Lin et al., 2015a; Luo et al., 2015; Guu et al., 2015; Toutanova et al., 2016; Nguyen et al., 2016a). See other embedding models for KB completion in Nguyen (2017).

Recently, convolutional neural networks (CNNs), originally designed for computer vision (LeCun et al., 1998), have significantly received research attention in natural language processing (Collobert et al., 2011; Kim, 2014). CNN learns non-linear features to capture complex relationships with a remarkably less number of parameters compared to fully connected neural networks. Inspired from the success in computer vision, Dettmers et al. (2018) proposed ConvE—the first model applying CNN for the KB completion task. In ConvE, only $\boldsymbol{v}_h$ and $\boldsymbol{v}_r$ are reshaped and then concatenated into an input matrix which is fed to the convolution layer. Different filters of the same 3×3 shape are operated over the input matrix to output feature map tensors. These feature map tensors are then vectorized and mapped into a vector via a linear transformation. Then this vector is computed with $\boldsymbol{v}_t$ via a dot product to return a score for *(h, r, t)*. See a formal definition of the ConvE score function in Table 1. It is worth noting that ConvE focuses on the local relationships among different dimensional entries in each of $\boldsymbol{v}_h$ or $\boldsymbol{v}_r$, i.e., *ConvE does not observe the global relationships among same dimensional entries of an embedding triple* ($\boldsymbol{v}_h$, $\boldsymbol{v}_r$, $\boldsymbol{v}_t$), so that ConvE ignores the transitional characteristic in transition-based models, which is one of the most useful intuitions for the task.

In this paper, we present ConvKB—an embedding model which proposes a novel use of CNN for the KB completion task. In ConvKB, each entity or relation is associated with an unique k-dimensional embedding. Let $\boldsymbol{v}_h$, $\boldsymbol{v}_r$ and $\boldsymbol{v}_t$ denote k-dimensional embeddings of h, r and t, respectively. For each triple *(h, r, t)*, the corresponding triple of k-dimensional embeddings ($\boldsymbol{v}_h$, $\boldsymbol{v}_r$, $\boldsymbol{v}_t$) is represented as a $k \times 3$ input matrix. This input matrix is fed to the convolution layer where different filters of the same 1×3 shape are used to extract the global relationships among same dimensional entries of the embedding triple. That is, these filters are repeatedly operated over every row of the input matrix to produce different feature maps. The feature maps are concatenated into a single feature vector which is then computed with a weight vector via a dot product to produce a score for the triple *(h, r, t)*. This score is used to infer whether the triple *(h, r, t)* is valid or not.

Model	The score function $f(h, r, t)$
TransE	$\|\boldsymbol{v}_h + \boldsymbol{v}_r - \boldsymbol{v}_t\|_p$
DISTMULT	$\langle \boldsymbol{v}_h, \boldsymbol{v}_r, \boldsymbol{v}_t \rangle$
ComplEx	$\mathsf{Re}\left(\langle \boldsymbol{v}_h, \boldsymbol{v}_r, \overline{\boldsymbol{v}}_t \rangle\right)$
ConvE	$g\left(\mathsf{vec}\left(g\left(\mathsf{concat}\left(\widehat{\boldsymbol{v}}_h, \widehat{\boldsymbol{v}}_r\right) * \boldsymbol{\Omega}\right)\right) \boldsymbol{W}\right) \cdot \boldsymbol{v}_t$
ConvKB	$\mathsf{concat}\left(g\left(\left[\boldsymbol{v}_h, \boldsymbol{v}_r, \boldsymbol{v}_t\right] * \boldsymbol{\Omega}\right)\right) \cdot \mathbf{w}$

Table 1: The score functions in previous SOTA models and in our ConvKB model. $\|\boldsymbol{v}\|_p$ denotes the p-norm of $\boldsymbol{v}$. $\langle \boldsymbol{v}_h, \boldsymbol{v}_r, \boldsymbol{v}_t \rangle = \sum_i \boldsymbol{v}_{h_i} \boldsymbol{v}_{r_i} \boldsymbol{v}_{t_i}$ denotes a tri-linear dot product. g denotes a non-linear function. $*$ denotes a convolution operator. $\cdot$ denotes a dot product. concat denotes a concatenation operator. $\widehat{\boldsymbol{v}}$ denotes a 2D reshaping of $\boldsymbol{v}$. $\boldsymbol{\Omega}$ denotes a set of filters.

Our contributions in this paper are as follows:

- We introduce ConvKB—a novel embedding model of entities and relationships for knowledge base completion. ConvKB models the relationships among same dimensional entries of the embeddings. This implies that ConvKB generalizes transitional characteristics in transition-based embedding models.
- We evaluate ConvKB on two benchmark datasets: WN18RR (Dettmers et al., 2018) and FB15k-237 (Toutanova and Chen, 2015). Experimental results show that ConvKB obtains better link prediction performance than previous SOTA embedding models. In particular, ConvKB obtains the best mean rank and the highest Hits@10 on WN18RR, and produces the highest mean reciprocal rank and highest Hits@10 on FB15k-237.

2 Proposed ConvKB model

A knowledge base $\mathcal{G}$ is a collection of valid factual triples in the form of *(head entity, relation, tail entity)* denoted as (h, r, t) such that $h, t \in \mathcal{E}$ and $r \in \mathcal{R}$ where $\mathcal{E}$ is a set of entities and $\mathcal{R}$ is a set of relations. Embedding models aim to define a *score function* f giving an implausibility score for each triple (h, r, t) such that valid triples receive lower scores than invalid triples. Table 1 presents score functions in previous SOTA models.

We denote the dimensionality of embeddings by k such that each embedding triple ($\boldsymbol{v}_h$, $\boldsymbol{v}_r$, $\boldsymbol{v}_t$) are

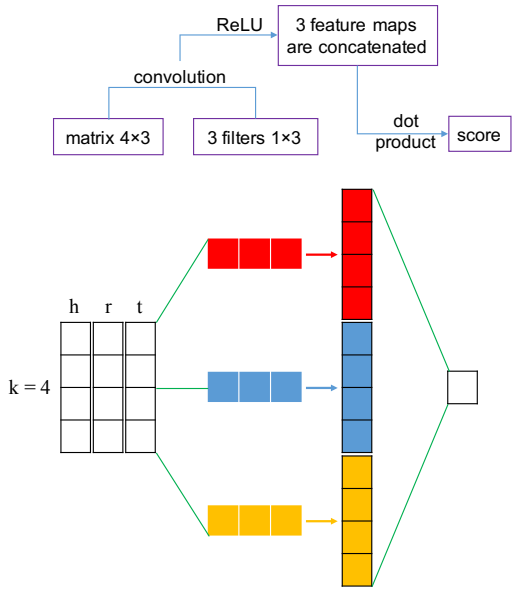

Figure 1: Process involved in ConvKB (with the embedding size $k = 4$, the number of filters $\tau = 3$ and the activation function $g =$ ReLU for illustration purpose).

viewed as a matrix $\boldsymbol{A} = [\boldsymbol{v}_h, \boldsymbol{v}_r, \boldsymbol{v}_t] \in \mathbb{R}^{k\times 3}$. And $\boldsymbol{A}_{i,:} \in \mathbb{R}^{1\times 3}$ denotes the i-th row of $\boldsymbol{A}$. Suppose that we use a filter $\boldsymbol{\omega} \in \mathbb{R}^{1\times 3}$ operated on the convolution layer. $\boldsymbol{\omega}$ is not only aimed to examine the global relationships between same dimensional entries of the embedding triple $(\boldsymbol{v}_h, \boldsymbol{v}_r, \boldsymbol{v}_t)$, but also to generalize the transitional characteristics in the transition-based models. $\boldsymbol{\omega}$ is repeatedly operated over every row of $\boldsymbol{A}$ to finally generate a feature map $\boldsymbol{v} = [v_1, v_2, ..., v_k] \in \mathbb{R}^k$ as:

$$v_i = g\left(\boldsymbol{\omega} \cdot \boldsymbol{A}_{i,:} + b\right)$$

where $b \in \mathbb{R}$ is a bias term and g is some activation function such as ReLU.

Our ConvKB uses different filters $\in \mathbb{R}^{1\times 3}$ to generate different feature maps. Let $\boldsymbol{\Omega}$ and τ denote the set of filters and the number of filters, respectively, i.e. $\tau = |\boldsymbol{\Omega}|$, resulting in τ feature maps. These τ feature maps are concatenated into a single vector $\in \mathbb{R}^{\tau k\times 1}$ which is then computed with a weight vector $\mathbf{w} \in \mathbb{R}^{\tau k\times 1}$ via a dot product to give a score for the triple (h, r, t). Figure 1 illustrates the computation process in ConvKB.

Formally, we define the ConvKB score function f as follows:

$$f(h, r, t) = \mathsf{concat}\left(g\left([\boldsymbol{v}_h, \boldsymbol{v}_r, \boldsymbol{v}_t] * \boldsymbol{\Omega}\right)\right) \cdot \mathbf{w}$$

where $\boldsymbol{\Omega}$ and $\mathbf{w}$ are shared parameters, independent of h, r and t; $*$ denotes a convolution operator; and concat denotes a concatenation operator.

Dataset	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{R}\rvert$	#Triples in train/valid/test		
WN18RR	40,943	11	86,835	3,034	3,134
FB15k-237	14,541	237	272,115	17,535	20,466

Table 2: Statistics of the experimental datasets.

If we only use one filter $\boldsymbol{\omega}$ (i.e. using $\tau = 1$) with a fixed bias term $b = 0$ and the activation function $g(x) = |x|$ or $g(x) = x^2$, and fix $\boldsymbol{\omega} = [1, 1, -1]$ and $\mathbf{w} = \mathbf{1}$ during training, ConvKB reduces to the plain TransE model (Bordes et al., 2013). So our ConvKB model can be viewed as an extension of TransE to further model global relationships.

We use the Adam optimizer (Kingma and Ba, 2014) to train ConvKB by minimizing the loss function $\mathcal{L}$ (Trouillon et al., 2016) with L_2 regularization on the weight vector $\mathbf{w}$ of the model:

$$\mathcal{L} = \sum_{(h,r,t)\in\{\mathcal{G}\cup\mathcal{G}'\}} \log\left(1 + \exp\left(l_{(h,r,t)} \cdot f\left(h, r, t\right)\right)\right) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

$$\text{in which, } l_{(h,r,t)} = \begin{cases} 1 \text{ for } (h, r, t) \in \mathcal{G} \\ -1 \text{ for } (h, r, t) \in \mathcal{G}' \end{cases}$$

here $\mathcal{G}'$ is a collection of invalid triples generated by corrupting valid triples in $\mathcal{G}$.

3 Experiments

3.1 Datasets

We evaluate ConvKB on two benchmark datasets: WN18RR (Dettmers et al., 2018) and FB15k-237 (Toutanova and Chen, 2015). WN18RR and FB15k-237 are correspondingly subsets of two common datasets WN18 and FB15k (Bordes et al., 2013). As noted by Toutanova and Chen (2015), WN18 and FB15k are easy because they contain many reversible relations. So knowing relations are reversible allows us to easily predict the majority of test triples, e.g. state-of-the-art results on both WN18 and FB15k are obtained by using a simple reversal rule as shown in Dettmers et al. (2018). Therefore, WN18RR and FB15k-237 are created to not suffer from this reversible relation problem in WN18 and FB15k, for which the knowledge base completion task is more realistic. Table 2 presents the statistics of WN18RR and FB15k-237.

Method	WN18RR			FB15k-237		
	MR	MRR	H@10	MR	MRR	H@10
IRN (Shen et al., 2017)	–	–	–	211	–	46.4
KBGAN (Cai and Wang, 2018)	–	0.213	48.1	–	0.278	45.8
DISTMULT (Yang et al., 2015) [⋆]	5110	0.43	49	254	0.241	41.9
ComplEx (Trouillon et al., 2016) [⋆]	5261	0.44	51	339	0.247	42.8
ConvE (Dettmers et al., 2018)	5277	**0.46**	48	246	0.316	49.1
TransE (Bordes et al., 2013) (our results)	3384	0.226	50.1	347	0.294	46.5
Our ConvKB model	**2554**	0.248	**52.5**	257	**0.396**	**51.7**
KB_{LRN} (García-Durán and Niepert, 2017)	–	–	–	**209**	0.309	49.3
R-GCN+ (Schlichtkrull et al., 2017)	–	–	–	–	0.249	41.7
Neural LP (Yang et al., 2017)	–	–	–	–	0.240	36.2
Node+LinkFeat (Toutanova and Chen, 2015)	–	–	–	–	0.293	46.2

Table 3: Experimental results on WN18RR and FB15k-237 test sets. MRR and H@10 denote the mean reciprocal rank and Hits@10 (in %), respectively. [⋆]: Results are taken from Dettmers et al. (2018) where Hits@10 and MRR are rounded to 2 decimal places on WN18RR. The last 4 rows report results of models that exploit information about relation paths (KB_{LRN}, R-GCN+ and Neural LP) or textual mentions derived from a large external corpus (Node+LinkFeat). The best score is in **bold**, while the second best score is in underline.

3.2 Evaluation protocol

In the KB completion or link prediction task (Bordes et al., 2013), the purpose is to predict a missing entity given a relation and another entity, i.e, inferring h given (r, t) or inferring t given (h, r). The results are calculated based on ranking the scores produced by the score function f on test triples.

Following Bordes et al. (2013), for each valid test triple (h, r, t), we replace either h or t by each of other entities in $\mathcal{E}$ to create a set of corrupted triples. We use the "**Filtered**" setting protocol (Bordes et al., 2013), i.e., not taking any corrupted triples that appear in the KB into accounts. We rank the valid test triple and corrupted triples in ascending order of their scores. We employ three common evaluation metrics: mean rank (MR), mean reciprocal rank (MRR), and Hits@10 (i.e., the proportion of the valid test triples ranking in top 10 predictions). Lower MR, higher MRR or higher Hits@10 indicate better performance.

3.3 Training protocol

We use the common Bernoulli trick (Wang et al., 2014; Lin et al., 2015b) to generate the head or tail entities when sampling invalid triples. We also use entity and relation embeddings produced by TransE to *initialize* entity and relation embeddings in ConvKB. We employ a TransE implementation available at: https://github.com/datquocnguyen/STransE. We train TransE using a grid search of hyper-parameters: the dimensionality of embeddings $k \in \{50, 100\}$, SGD learning rate $\in \{1e^{-4}, 5e^{-4}, 1e^{-3}, 5e^{-3}\}$, l_1-norm or l_2-norm, and margin $\gamma \in \{1, 3, 5, 7\}$. The highest Hits@10 scores on the validation set are when using l_1-norm, learning rate at $5e^{-4}$, γ = 5 and k = 50 for WN18RR, and using l_1-norm, learning rate at $5e^{-4}$, γ = 1 and k = 100 for FB15k-237.

To learn our model parameters including entity and relation embeddings, filters $\boldsymbol{\omega}$ and the weight vector $\mathbf{w}$, we use Adam (Kingma and Ba, 2014) and select its initial learning rate $\in \{5e^{-6}, 1e^{-5}, 5e^{-5}, 1e^{-4}, 5e^{-4}\}$. We use ReLU as the activation function g. We fix the batch size at 256 and set the L_2-regularizer λ at 0.001 in our objective function. The filters $\boldsymbol{\omega}$ are initialized by a truncated normal distribution or by $[0.1, 0.1, -0.1]$. We select the number of filters $\tau \in \{50, 100, 200, 400, 500\}$. We run ConvKB up to 200 epochs and use outputs from the last epoch for evaluation. The highest Hits@10 scores on the validation set are obtained when using k = 50, τ = 500, the truncated normal distribution for filter initialization, and the initial learning rate at $1e^{-4}$ on WN18RR; and k = 100, τ = 50, $[0.1, 0.1, -0.1]$ for filter initialization, and the initial learning rate at $5e^{-6}$ on FB15k-237.

3.4 Main experimental results

Table 3 compares the experimental results of our ConvKB model with previous published results, using the same experimental setup. Table 3 shows that ConvKB obtains the best MR and highest Hits@10 scores on WN18RR and also the highest

MRR and Hits@10 scores on FB15k-237.

ConvKB does better than the closely related model TransE on both experimental datasets, especially on FB15k-237 where ConvKB gains significant improvements of $347 - 257 = 90$ in MR (which is about 26% relative improvement) and $0.396 - 0.294 = 0.102$ in MRR (which is 34+% relative improvement), and also obtains $51.7 - 46.5 = 5.2\%$ absolute improvement in Hits@10. Previous work shows that TransE obtains very competitive results (Lin et al., 2015a; Nickel et al., 2016b; Trouillon et al., 2016; Nguyen et al., 2016a). However, when comparing the CNN-based embedding model ConvE with other models, Dettmers et al. (2018) did not experiment with TransE. We reconfirm previous findings that TransE in fact is a strong baseline model, e.g., TransE obtains better MR and Hits@10 than ConvE on WN18RR.

ConvKB obtains better scores than ConvE on both datasets (except MRR on WN18RR and MR on FB15k-237), thus showing the usefulness of taking transitional characteristics into accounts. In particular, on FB15k-237, ConvKB achieves improvements of $0.394 - 0.316 = 0.078$ in MRR (which is about 25% relative improvement) and $51.7 - 49.1 = 2.6\%$ in Hits@10, while both ConvKB and ConvE produce similar MR scores. ConvKB also obtains 25% relatively higher MRR score than the relation path-based model KB_{LRN} on FB15k-237. In addition, ConvKB gives better Hits@10 than KB_{LRN}, however, KB_{LRN} gives better MR than ConvKB. We plan to extend ConvKB with relation path information to obtain better link prediction performance in future work.

4 Conclusion

In this paper, we propose a novel embedding model ConvKB for the knowledge base completion task. ConvKB applies the convolutional neural network to explore the global relationships among same dimensional entries of the entity and relation embeddings, so that ConvKB generalizes the transitional characteristics in the transition-based embedding models. Experimental results show that our model ConvKB outperforms other state-of-the-art models on two benchmark datasets WN18RR and FB15k-237. Our code is available at: https://github.com/daiquocnguyen/ConvKB.

We also plan to extend ConvKB for a new application where we could formulate data in the form of triples. For example, inspired from the work by Vu et al. (2017) for search personalization, we can also apply ConvKB to model *user*-oriented relationships between submitted *queries* and *documents* returned by search engines, i.e. modeling triple representations (query, user, document).

Acknowledgments

This research was partially supported by the Australian Research Council (ARC) Discovery Grant Project DP160103934.

References

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 1247–1250.

Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *Advances in Neural Information Processing Systems 26*, pages 2787–2795.

Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. 2011. Learning Structured Embeddings of Knowledge Bases. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages 301–306.

Liwei Cai and William Yang Wang. 2018. KBGAN: Adversarial Learning for Knowledge Graph Embeddings. In *Proceedings of The 16th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, page to appear.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2018. Convolutional 2D Knowledge Graph Embeddings. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, page to appear.

Alberto García-Durán and Mathias Niepert. 2017. KBLRN: End-to-end learning of knowledge base representations with latent, relational, and numerical features. *arXiv preprint abs/1709.04676*.

Kelvin Guu, John Miller, and Percy Liang. 2015. Traversing Knowledge Graphs in Vector Space. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 318–327.

Yanchao Hao, Yuanzhe Zhang, Kang Liu, Shizhu He, Zhanyi Liu, Hua Wu, and Jun Zhao. 2017. An end-to-end model for question answering over knowledge base with cross-attention combining global knowledge. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 221–231.

Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Knowledge Graph Embedding via Dynamic Mapping Matrix. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 687–696.

Guoliang Ji, Kang Liu, Shizhu He, and Jun Zhao. 2016. Knowledge Graph Completion with Adaptive Sparse Transfer Matrix. In *Proceedings of the Thirtieth Conference on Artificial Intelligence*, pages 985–991.

Gjergji Kasneci, Fabian M Suchanek, Georgiana Ifrim, Maya Ramanath, and Gerhard Weikum. 2008. Naga: Searching and ranking knowledge. In *Proceedings of the 24th IEEE International Conference on Data Engineering*, pages 953–962.

Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1746–1751.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278–2324.

Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. 2015. DBpedia–a large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6:167–195.

Yankai Lin, Zhiyuan Liu, Huanbo Luan, Maosong Sun, Siwei Rao, and Song Liu. 2015a. Modeling Relation Paths for Representation Learning of Knowledge Bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 705–714.

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015b. Learning Entity and Relation Embeddings for Knowledge Graph Completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence Learning*, pages 2181–2187.

Yuanfei Luo, Quan Wang, Bin Wang, and Li Guo. 2015. Context-Dependent Knowledge Graph Embedding. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1656–1661.

Dat Quoc Nguyen. 2017. An overview of embedding models of entities and relationships for knowledge base completion. *arXiv preprint*, arXiv:1703.08098.

Dat Quoc Nguyen, Kairit Sirts, Lizhen Qu, and Mark Johnson. 2016a. Neighborhood Mixture Model for Knowledge Base Completion. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 40–50.

Dat Quoc Nguyen, Kairit Sirts, Lizhen Qu, and Mark Johnson. 2016b. STransE: a novel embedding model of entities and relationships in knowledge bases. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 460–466.

Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. 2016a. A Review of Relational Machine Learning for Knowledge Graphs. *Proceedings of the IEEE*, 104(1):11–33.

Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. 2016b. Holographic Embeddings of Knowledge Graphs. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 1955–1961.

Michael Schlichtkrull, Thomas Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2017. Modeling relational data with graph convolutional networks. *arXiv preprint arXiv:1703.06103*.

Michael Schuhmacher and Simone Paolo Ponzetto. 2014. Knowledge-based graph document modeling. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, pages 543–552.

Yelong Shen, Po-Sen Huang, Ming-Wei Chang, and Jianfeng Gao. 2017. Traversing knowledge graph in vector space without symbolic space guidance. *arXiv preprint arXiv:1611.04642v4*.

Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. 2013. Reasoning With Neural Tensor Networks for Knowledge Base Completion. In *Advances in Neural Information Processing Systems 26*, pages 926–934.

Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: A core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web*, pages 697–706.

Kristina Toutanova and Danqi Chen. 2015. Observed Versus Latent Features for Knowledge Base and Text Inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pages 57–66.

Kristina Toutanova, Xi Victoria Lin, Wen tau Yih, Hoifung Poon, and Chris Quirk. 2016. Compositional Learning of Embeddings for Relation Paths in Knowledge Bases and Text. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1434–1444.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex Embeddings for Simple Link Prediction. In *Proceedings of the 33nd International Conference on Machine Learning*, pages 2071–2080.

Thanh Vu, Dat Quoc Nguyen, Mark Johnson, Dawei Song, and Alistair Willis. 2017. Search Personalization with Embeddings. In *Proceedings of the 39th European Conference on Information Retrieval*, pages 598–604.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge Graph Embedding by Translating on Hyperplanes. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 1112–1119.

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

Chenyan Xiong, Russell Power, and Jamie Callan. 2017. Explicit semantic ranking for academic search via knowledge graph embedding. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1271–1279.

Bishan Yang and Tom Mitchell. 2017. Leveraging knowledge bases in lstms for improving machine reading. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1436–1446.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *Proceedings of the International Conference on Learning Representations*.

Fan Yang, Zhilin Yang, and William W Cohen. 2017. Differentiable Learning of Logical Rules for Knowledge Base Reasoning. pages 2316–2325.

Yuanzhe Zhang, Kang Liu, Shizhu He, Guoliang Ji, Zhanyi Liu, Hua Wu, and Jun Zhao. 2016. Question answering over knowledge base with neural attention combining global knowledge information. *arXiv preprint arXiv:1606.00979*.

Semantic Web 0 (0)
IOS Press

A Convolutional Neural Network-based Model for Knowledge Base Completion and Its Application to Search Personalization

Dai Quoc Nguyen [a,*], Dat Quoc Nguyen [b], Tu Dinh Nguyen [a] and Dinh Phung [a]
[a] *Monash University, Australia*
E-mails: dai.nguyen@monash.edu, tu.dinh.nguyen@monash.edu, dinh.phung@monash.edu
[b] *The University of Melbourne, Australia*
E-mail: dqnguyen@unimelb.edu.au

Editors: Dagmar Gromann, Technical University Dresden, Germany; Luis Espinosa Anke, Cardiff University, UK; Thierry Declerck, DFKI GmbH, Germany
Solicited reviews: Sergio Oramas, Music Technology Group, Pompeu Fabra University, Spain; Two anonymous reviewers

Abstract. In this paper, we propose a novel embedding model, named ConvKB, for knowledge base completion. Our model ConvKB advances state-of-the-art models by employing a convolutional neural network, so that it can capture global relationships and transitional characteristics between entities and relations in knowledge bases. In ConvKB, each triple *(head entity, relation, tail entity)* is represented as a 3-column matrix where each column vector represents a triple element. This 3-column matrix is then fed to a convolution layer where multiple filters are operated on the matrix to generate different feature maps. These feature maps are then concatenated into a single feature vector representing the input triple. The feature vector is multiplied with a weight vector via a dot product to return a score. This score is then used to predict whether the triple is valid or not. Experiments show that ConvKB obtains better link prediction and triple classification results than previous state-of-the-art models on benchmark datasets WN18RR, FB15k-237, WN11 and FB13. We further apply our ConvKB to a search personalization problem which aims to tailor the search results to each specific user based on the user's personal interests and preferences. In particular, we model the potential relationship between the submitted query, the user and the search result (i.e., document) as a triple *(query, user, document)* on which the ConvKB is able to work. Experimental results on query logs from a commercial web search engine show that ConvKB achieves better performances than the standard ranker as well as strong search personalization baselines.

Keywords: Knowledge base completion, Convolutional neural network, ConvKB, Link prediction, Triple classification, Search personalization

1. Introduction

Large-scale knowledge bases (KBs), such as YAGO [41], Freebase [3] and DBpedia [25], are usually databases of triples representing the relationships between entities in the form of fact *(head entity, relation, tail entity)* denoted as *(h, r, t)*, e.g., *(Melbourne, cityOf, Australia)*. These KBs are useful resources in many applications such as semantic searching and ranking [21, 38, 60], question answering [17, 66] and machine reading [61]. However, the KBs are still incomplete, i.e., missing a lot of valid triples [40, 55]. Therefore, much research work has been devoted towards *knowledge base completion* or *link prediction* to predict whether a triple *(h, r, t)* is valid or not [4].

*Corresponding author. E-mail: dai.nguyen@monash.edu. The final publication is available at IOS Press through http://dx.doi.org/10.3233/SW-180318.

Many embedding models have proposed to learn vector or matrix representations for entities and relations, obtaining state-of-the-art (SOTA) link prediction results [35]. In these embedding models, valid triples obtain lower implausibility scores than invalid triples. Let us take the well-known embedding model TransE [5] as an example. In TransE, entities and relations are represented by k-dimensional vector embeddings. TransE employs a transitional characteristic to model relationships between entities, in which it assumes that if *(h, r, t)* is a valid fact, the embedding of head entity h plus the embedding of relation r should be close to the embedding of tail entity t, i.e. $\boldsymbol{v}_h + \boldsymbol{v}_r \approx \boldsymbol{v}_t$ (here, $\boldsymbol{v}_h$, $\boldsymbol{v}_r$ and $\boldsymbol{v}_t$ are embeddings of h, r and t respectively). That is, a TransE score $\|\boldsymbol{v}_h + \boldsymbol{v}_r - \boldsymbol{v}_t\|_p$ of the valid triple *(h, r, t)* should be close to 0 and smaller than a score $\|\boldsymbol{v}_{h'} + \boldsymbol{v}_{r'} - \boldsymbol{v}_{t'}\|_p$ of an invalid triple *(h', r', t')*. The transitional characteristic in TransE also implies the global relationships among same dimensional entries of $\boldsymbol{v}_h$, $\boldsymbol{v}_r$ and $\boldsymbol{v}_t$. Other transition-based models extend TransE to use additional projection vectors or matrices to translate head and tail embeddings into the relation vector space, such as: TransH [54], TransR [27], TransD [19], STransE [33] and TranSparse [20]. Furthermore, DISTMULT [62] and ComplEx [48] use a tri-linear dot product to compute the score for each triple. Recent research has shown that using relation paths between entities in the KBs could help to get contextual information for improving KB completion performance [16, 26, 29, 32, 47]. See other embedding models for KB completion in Nguyen [31].

Recently, convolutional neural networks (CNNs), originally designed for computer vision [24], have significantly received research attention in natural language processing [9, 22]. CNN learns non-linear features to capture complex relationships with a remarkably less number of parameters compared to fully connected neural networks. Inspired from the success in computer vision, Dettmers et al. [10] proposed ConvE—the first model applying CNN for KB completion. In ConvE, only $\boldsymbol{v}_h$ and $\boldsymbol{v}_r$ are reshaped and then concatenated into an input matrix which is fed to the convolution layer. Different filters of the same 3×3 shape are operated over the input matrix to output feature map tensors. These feature map tensors are then vectorized and mapped into a vector via a linear transformation. Then this vector is computed with $\boldsymbol{v}_t$ via a dot product to return a score for *(h, r, t)*. See a formal definition of the ConvE score function in Table 1. It is worth noting that ConvE focuses on the local relationships among different dimensional entries in each of $\boldsymbol{v}_h$ or $\boldsymbol{v}_r$, i.e., *ConvE does not observe the global relationships among same dimensional entries of an embedding triple ($\boldsymbol{v}_h$, $\boldsymbol{v}_r$, $\boldsymbol{v}_t$)*, so that ConvE ignores the transitional characteristic in transition-based models, which is one of the most useful intuitions for KB completion.

In this paper, we present ConvKB—a novel embedding model which proposes a novel use of CNN for the KB completion task. In ConvKB, each entity or relation is associated with an unique k-dimensional embedding. Let $\boldsymbol{v}_h$, $\boldsymbol{v}_r$ and $\boldsymbol{v}_t$ denote k-dimensional embeddings of h, r and t, respectively. For each triple *(h, r, t)*, the corresponding triple of k-dimensional embeddings ($\boldsymbol{v}_h$, $\boldsymbol{v}_r$, $\boldsymbol{v}_t$) is represented as a $k \times 3$ input matrix. This input matrix is fed to the convolution layer where different filters of the same 1×3 shape are used to extract the global relationships among same dimensional entries of the embedding triple. That is, these filters are repeatedly operated over every row of the input matrix to produce different feature maps. The feature maps are concatenated into a single feature vector which is then computed with a weight vector via a dot product to produce a score for the triple *(h, r, t)*. This score is used to infer whether the triple *(h, r, t)* is valid or not.

Our contributions in this paper are as follows:

- We introduce ConvKB—a novel embedding model of entities and relationships for knowledge base completion. ConvKB models the relationships among same dimensional entries of the embeddings. This implies that ConvKB generalizes transitional characteristics in transition-based embedding models.
- We evaluate ConvKB on two benchmark datasets WN18RR [10] and FB15k-237 [45], and show that ConvKB obtains better link prediction performance than previous SOTA embedding models. In particular, ConvKB obtains the best mean rank and the highest Hits@10 on WN18RR, and produces the highest mean reciprocal rank and highest Hits@10 on FB15k-237.
- We also evaluate ConvKB for triple classification on two benchmark datasets WN11 and FB13 [40]. The goal is to classify whether a given triple is valid or not. ConvKB also does better than previous SOTA models, obtaining the best and second best accuracies on WN11 and FB13, respectively.
- We adapt ConvKB to search personalization where the search results for a query from a user are driven toward the personal needs of that user by exploiting historical interactions (e.g., submitted queries, clicked documents) between the user

and the system [1, 18, 42, 43, 56]. Our general aim is to re-rank the documents returned by the search system in such a way that the more relevant documents are ranked higher. More specially, we train our ConvKB to measure a score for each triple *(query, user, document)*, and to reward higher plausibility scores for more relevant documents. We then verify this application of ConvKB on the query logs of a commercial web search engine. Experimental results show that ConvKB significantly improves the ranking quality over the strong baselines.

The paper is organized as follows. We provide related work in Section 2. We then describe our proposed model ConvKB in Section 3. We evaluate and compare ConvKB with previous models on the link prediction and triple classification tasks in Section 4. The application of ConvKB to search personalization is introduced in Section 5. The conclusion is finally presented in Section 6.

2. Related work

TransH [54] extends TransE to allow entities playing different roles in different relations. Each relation r is associated with a relation-specific hyperplane $\mathbf{w}_r$, and then the embeddings of h and t are projected to this hyperplane. The TransH score function is defined as:

$$f_{\text{TransH}}(h,r,t) = \|\boldsymbol{v}_{h\perp} + \boldsymbol{v}_r - \boldsymbol{v}_{t\perp}\|_p$$

where $\boldsymbol{v}_{h\perp} = \boldsymbol{v}_h - \mathbf{w}_r^{\mathsf{T}}\boldsymbol{v}_h\mathbf{w}_r$ and $\boldsymbol{v}_{t\perp} = \boldsymbol{v}_t - \mathbf{w}_r^{\mathsf{T}}\boldsymbol{v}_t\mathbf{w}_r$ are the projected embeddings of h and t on $\mathbf{w}_r$ respectively.

TransR [27] extends TransH to perform projections where each relation r is associated with a projection matrix $\mathbf{W}_r$ which is used to map entity embeddings into the vector space of relations:

$$f_{\text{TransR}}(h,r,t) = \|\mathbf{W}_r\boldsymbol{v}_h + \boldsymbol{v}_r - \mathbf{W}_r\boldsymbol{v}_t\|_p$$

Both TransH and TransR use only one vector or matrix to perform projections, ignoring the fact that head and tail entities have different properties to each relation. Therefore, head and tail entities should be associated with their own projection vectors or matrices as presented in direct extensions of TransH and TransR such as TransD [19], STransE [33], lppTransD [65], TransR-FT [12], TranSparse [20] and ITransF[59]. The transitional characteristics in these transition-based models can be intuitively defined as: if (h,r,t) is a valid fact, the *projected* embedding of h plus the embedding of r is close to the *projected* embedding of t. This reflects global relationships among same dimensional entries of *projected* entity embeddings with relation embedding. In our ConvKB model, each filter with the shape of 1×3 is responsible for mapping head, tail and relation embeddings to the relation vector space. So ConvKB can generalize the transitional characteristics in the transition-based models.

Table 1
The score functions in previous models and in our ConvKB model. $\|\boldsymbol{v}\|_p$ denotes the p-norm of $\boldsymbol{v}$. $\langle \boldsymbol{v}_h, \boldsymbol{v}_r, \boldsymbol{v}_t\rangle = \sum_i \boldsymbol{v}_{h_i}\boldsymbol{v}_{r_i}\boldsymbol{v}_{t_i}$ denotes a tri-linear dot product. $\bar{\boldsymbol{v}} = \mathsf{Re}(\boldsymbol{v}) - \imath\mathsf{Im}(\boldsymbol{v})$ with $\mathsf{Re}(\boldsymbol{v})$ and $\mathsf{Im}(\boldsymbol{v})$ corresponding to the real and imaginary parts of the complex-valued vector $\boldsymbol{v}$, and $\imath$ denoting the square root of -1. In addition, g denotes a non-linear function. $*$ denotes a convolution operator. $\cdot$ denotes a dot product. concat denotes a concatenation operator. $\widehat{\boldsymbol{v}}$ denotes a 2D reshaping of $\boldsymbol{v}$. $\boldsymbol{\Omega}$ denotes a set of filters.

Model	The score function $f(h,r,t)$
TransE	$\|\boldsymbol{v}_h + \boldsymbol{v}_r - \boldsymbol{v}_t\|_p$
DISTMULT	$\langle \boldsymbol{v}_h, \boldsymbol{v}_r, \boldsymbol{v}_t\rangle$
ComplEx	$\mathsf{Re}\left(\langle \boldsymbol{v}_h, \boldsymbol{v}_r, \bar{\boldsymbol{v}}_t\rangle\right)$
ConvE	$g\left(\mathsf{vec}\left(g\left(\mathsf{concat}\left(\widehat{\boldsymbol{v}}_h, \widehat{\boldsymbol{v}}_r\right) * \boldsymbol{\Omega}\right)\right)\boldsymbol{W}\right) \cdot \boldsymbol{v}_t$
Our ConvKB	$\mathsf{concat}\left(g\left(\left[\boldsymbol{v}_h, \boldsymbol{v}_r, \boldsymbol{v}_t\right] * \boldsymbol{\Omega}\right)\right) \cdot \mathbf{w}$

DISTMULT [62] and ComplEx [48] use a tri-linear dot product to compute the score for each triple. See formal definitions of DISTMULT and ComplEx in Table 1. In addition, NTN [40] uses a bilinear tensor operator into a neural network to compute the triple score. Recent approaches also show that using relation paths between entities in the KBs could help to get contextual information for improving the KB completion performance [16, 26, 32, 47]. For example, PTransE-ADD [26] and TransE-COMP [16] represent each path by summing up the embeddings of all relations in the path, while Bilinear-COMP [16] and PRUNED-PATHS [47] represent each relation in the path by a matrix and directly use matrix multiplication to modeling relation path. Some approaches also incorporate textual mentions derived from a large external corpus for improving the performance [14, 45, 46, 53]. See other methods for learning from KBs in [31, 35].

3. Proposed ConvKB model

A knowledge base $\mathcal{G}$ is a collection of valid factual triples in the form of *(head entity, relation, tail entity)* denoted as (h,r,t) such that $h,t \in \mathcal{E}$ and $r \in \mathcal{R}$ where $\mathcal{E}$ is a set of entities and $\mathcal{R}$ is a set of relations. Embedding models aim to define a *score function* f giving an implausibility score for each triple (h,r,t)

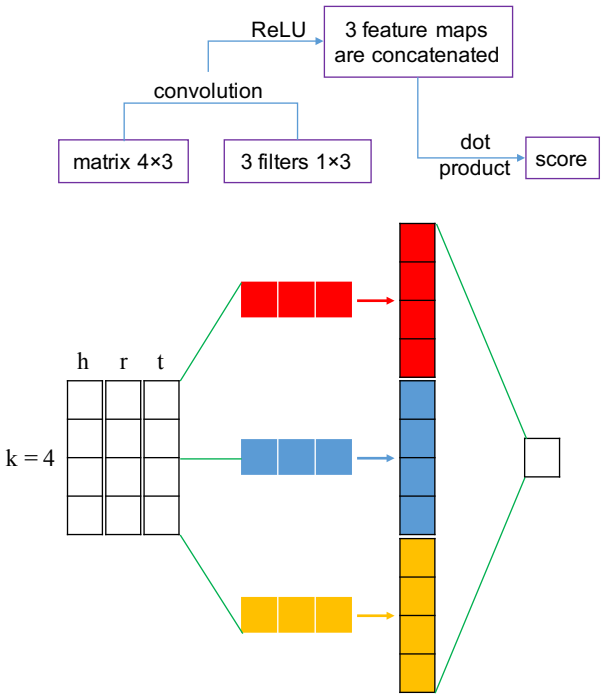

Figure 1. Process involved in the proposed ConvKB (with $k = 4$ and $\tau = 3$ for illustration purpose).

such that valid triples receive lower scores than invalid triples. Table 1 presents score functions in previous SOTA models.

We denote the dimensionality of embeddings by k such that each embedding triple ($\boldsymbol{v}_h$, $\boldsymbol{v}_r$, $\boldsymbol{v}_t$) are viewed as a matrix $\boldsymbol{A} = [\boldsymbol{v}_h, \boldsymbol{v}_r, \boldsymbol{v}_t] \in \mathbb{R}^{k\times 3}$. And $\boldsymbol{A}_{i,:} \in \mathbb{R}^{1\times 3}$ denotes the i-th row of $\boldsymbol{A}$. Suppose that we use a filter $\boldsymbol{\omega} \in \mathbb{R}^{1\times 3}$ operated on the convolution layer. $\boldsymbol{\omega}$ is not only aimed to examine the global relationships between same dimensional entries of the embedding triple ($\boldsymbol{v}_h$, $\boldsymbol{v}_r$, $\boldsymbol{v}_t$), but also to generalize the transitional characteristics in the transition-based models. $\boldsymbol{\omega}$ is repeatedly operated over every row of $\boldsymbol{A}$ to finally generate a feature map $\boldsymbol{v} = [v_1, v_2, ..., v_k] \in \mathbb{R}^k$ as:

$$v_i = g\left(\boldsymbol{\omega} \cdot \boldsymbol{A}_{i,:} + b\right)$$

where $b \in \mathbb{R}$ is a bias term and g is a non-linear activation function such as ReLU.

Our ConvKB uses different filters $\in \mathbb{R}^{1\times 3}$ to generate different feature maps. Let $\boldsymbol{\Omega}$ and τ denote the set of filters and the number of filters, respectively, i.e. $\tau = |\boldsymbol{\Omega}|$, resulting in τ feature maps. These τ feature maps are concatenated into a single vector $\in \mathbb{R}^{\tau k\times 1}$ which is then computed with a weight vector $\mathbf{w} \in \mathbb{R}^{\tau k\times 1}$ via a dot product to give a score for the triple (h, r, t). Figure 1 illustrates the computation process in ConvKB.

Formally, we define the ConvKB score function f as follows:

$$f(h, r, t) = \text{concat}\left(g\left([\boldsymbol{v}_h, \boldsymbol{v}_r, \boldsymbol{v}_t] * \boldsymbol{\Omega}\right)\right) \cdot \mathbf{w}$$

where $\boldsymbol{\Omega}$ and $\mathbf{w}$ are shared parameters, independent of h, r and t; $*$ denotes a convolution operator; and concat denotes a concatenation operator.

If we only use one filter $\boldsymbol{\omega}$ (i.e. using $\tau = 1$) with a fixed bias term $b = 0$ and the activation function $g(x) = |x|$ or $g(x) = x^2$, and fix $\boldsymbol{\omega} = [1, 1, -1]$ and $\mathbf{w} = \mathbf{1}$ during training, ConvKB reduces to the plain TransE model [5]. So our ConvKB model can be viewed as an extension of TransE to further model global relationships.

We use the Adam optimizer [23] to train ConvKB by minimizing the loss function $\mathcal{L}$ [48] with L_2 regularization on the weight vector $\mathbf{w}$ of the model:

$$\mathcal{L} = \sum_{(h,r,t)\in\{\mathcal{G}\cup\mathcal{G}'\}} \log\left(1 + \exp\left(l_{(h,r,t)} \cdot f\left(h, r, t\right)\right)\right) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

$$\text{in which, } l_{(h,r,t)} = \begin{cases} 1 \text{ for } (h, r, t) \in \mathcal{G} \\ -1 \text{ for } (h, r, t) \in \mathcal{G}' \end{cases}$$

Input: knowledge base $\mathcal{G}$, set of entities $\mathcal{E}$, set of relations $\mathcal{R}$, set of filters $\mathbf{\Omega}$, weight vector $\mathbf{w}$, batch size *batch_size*, L_2-regularizer λ, boolean *init_filter_normal*, pre-trained entity and relation k-dimensional embeddings produced by TransE [5].

// Initialize variables.

for $e \in \mathcal{E}$ **do**
  $\boldsymbol{v}_e \leftarrow$ **embeddings_by_TransE**(e)

for $r \in \mathcal{R}$ **do**
  $\boldsymbol{v}_r \leftarrow$ **embeddings_by_TransE**(r)

// Using a truncated normal distribution with $init\_filter\_normal = True$ while using $[0.1, 0.1, -0.1]$ with $init\_filter\_normal = False$.

for $\omega \in \mathbf{\Omega}$ **do**
  Initialize$(\omega, init\_filter\_normal)$

$\mathbf{w} \leftarrow$ **uniform** $\left(-\frac{\sqrt{6}}{\sqrt{k \times |\mathbf{\Omega}|+1}}, \frac{\sqrt{6}}{\sqrt{k \times |\mathbf{\Omega}|+1}}\right)$

// Optimization step.

for *epoch = 1, 2, ..., 200* **do**
  for *i = 1, 2, ...,* $\frac{|\mathcal{G}|}{batch\_size} + 1$ **do**
    Sample a vaild_batch of *batch_size* triples (h, r, t) from $\mathcal{G}$.
    invalid_batch $= \{\}$
    for $(h, r, t) \in$ vaild_batch **do**
      invalid_batch $\leftarrow$ invalid_batch $\cup$ **sample_invalid_triple**(h, r, t)
    batch $\leftarrow$ vaild_batch $\cup$ invalid_batch
    for $(h, r, t) \in$ batch **do**
      Compute $f(h, r, t) = \text{concat}\left(g\left([\boldsymbol{v}_h, \boldsymbol{v}_r, \boldsymbol{v}_t] * \mathbf{\Omega}\right)\right) \cdot \mathbf{w}$
      if (h, r, t) *is valid* **then**
        $l_{(h,r,t)} = 1$
      else
        $l_{(h,r,t)} = -1$
    Compute gradient $\nabla \mathcal{L}_{batch}$ w.r.t batch: $\nabla \sum_{(h,r,t) \in \text{batch}} \log\left(1 + \exp\left(l_{(h,r,t)} \cdot f(h, r, t)\right)\right) + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$
    Update entity and relation embeddings, weight vector $\mathbf{w}$ and filters w.r.t $\nabla \mathcal{L}_{batch}$.

Algorithm 1: Parameter optimization for ConvKB in the KB completion.

here $\mathcal{G}'$ is a collection of invalid triples generated by corrupting valid triples in $\mathcal{G}$. We use the common Bernoulli trick [27, 54] to generate the head or tail entities for invalid triples. For each relation r, let η_h denote the averaged number of head entities per tail entity whilst η_t denote the averaged number of tail entities per head entity. Given a valid triple (h, r, t) of relation r, we then generate a new head entity h' with probability $\frac{\eta_t}{\eta_t + \eta_h}$ to form an invalid triple (h', r, t) and a new tail entity t' with probability $\frac{\eta_h}{\eta_t + \eta_h}$ to form an invalid triple (h, r, t'). Algorithm 1 details the learning process of our ConvKB model.

4. KB completion evaluation

We evaluate ConvKB on two KB completion tasks: the link prediction task [5] and the triple classification task [40]. We use benchmark datasets WN18RR [10] and FB15k-237 [45] for link prediction, and datasets FB13 and WN11 [40] for triple classification. WN18RR and FB15k-237 are subsets of two common datasets WN18 and FB15k [5], respectively. As noted by Toutanova and Chen [45], WN18 and FB15k are easy because they contain many reversible relations. So knowing relations are reversible allows us to easily

Table 2
Statistics of the experimental datasets. In both WN11 and FB13, each validation and test set also contains the same number of incorrect triples as the number of correct triples. It is to note that the FB13 test set is filtered to only contain 7 relations taken from 13 relations appearing in the FB13 training set.

Dataset	$\mid \mathcal{E} \mid$	$\mid \mathcal{R} \mid$	#Triples in train/valid/test		
FB15k-237	14,541	237	272,115	17,535	20,466
WN18RR	40,943	11	86,835	3,034	3,134
FB13	75,043	13	316,232	5,908	23,733
WN11	38,696	11	112,581	2,609	10,544

predict the majority of test triples, e.g. state-of-the-art results on both WN18 and FB15k are obtained by using a simple reversal rule as shown in Dettmers et al. [10]. Therefore, WN18RR and FB15k-237 are created to not suffer from this reversible relation problem in WN18 and FB15k, for which the knowledge base completion task is more realistic. It is also worth noting that when constructing datasets FB13 and WN11, Socher et al. [40] filtered out triples from the test set if either or both of their head and tail entities also appear in the training set in a different relation type or order. Table 2 gives statistics of the experimental datasets.

4.1. Link prediction

4.1.1. Task description

In the KB completion or link prediction task [5], the purpose is to predict a missing entity given a relation and another entity, i.e, inferring h given (r,t) or inferring t given (h,r). The results are calculated based on ranking the scores produced by the score function f on test triples. Following Bordes et al. [5], for each valid test triple (h,r,t), we replace either h or t by each of other entities in $\mathcal{E}$ to create a set of corrupted triples. We use the "**Filtered**" setting protocol [5], i.e., not taking any corrupted triples that appear in the KB into accounts. We rank the valid test triple and corrupted triples in ascending order of their scores.

We employ three common evaluation metrics: mean rank (MR), mean reciprocal rank (MRR), and Hits@10 (i.e., the proportion of the valid test triples ranking in top 10 predictions). Lower MR, higher MRR or higher Hits@10 indicate better performance. We report the final scores on the test set for the model obtaining the highest Hits@10 score on the validation set.[1]

4.1.2. Training protocol

We use the common Bernoulli trick [27, 54] to generate the head or tail entities when sampling invalid triples. We also use entity and relation embeddings produced by TransE to *initialize* entity and relation embeddings in ConvKB.[2] We train TransE using a grid search of hyper-parameters: the dimensionality of embeddings $k \in \{50, 100\}$, SGD learning rate $\in \{1e^{-4}, 5e^{-4}, 1e^{-3}, 5e^{-3}\}$, l_1-norm or l_2-norm, and margin $\gamma \in \{1, 3, 5, 7\}$. The highest Hits@10 scores on the validation set are when using l_1-norm, learning rate at $5e^{-4}$, γ = 5 and k = 50 for WN18RR, and using l_1-norm, learning rate at $5e^{-4}$, γ = 1 and k = 100 for FB15k-237.

To learn our model parameters including entity and relation embeddings, filters $\boldsymbol{\omega}$ and the weight vector $\mathbf{w}$, we use Adam [23] and select its initial learning rate $\in \{5e^{-6}, 1e^{-5}, 5e^{-5}, 1e^{-4}, 5e^{-4}\}$. We use ReLU as the activation function g. We fix the batch size at 256 and set the L_2-regularizer λ at 0.001 in our objective function. The filters $\boldsymbol{\omega}$ are initialized by a truncated normal distribution or by $[0.1, 0.1, -0.1]$. We select the number of filters $\tau \in \{50, 100, 200, 400, 500\}$. We run ConvKB up to 200 epochs and use outputs from the last epoch for evaluation. The highest Hits@10 scores on the validation set are obtained when using k = 50, $\tau = 500$, the truncated normal distribution for filter initialization, and the initial learning rate at $1e^{-4}$ on WN18RR; and k = 100, $\tau = 50$, $[0.1, 0.1, -0.1]$ for filter initialization, and the initial learning rate at $5e^{-6}$ on FB15k-237.

4.1.3. Main experimental results

Table 3 compares the experimental results of our ConvKB model with previous published results, using the same experimental setup. Table 3 shows that ConvKB obtains the best MR and highest Hits@10 scores on WN18RR and also the highest MRR and Hits@10 scores on FB15k-237.

ConvKB does better than the closely related model TransE on both experimental datasets, especially on FB15k-237 where ConvKB gains significant improvements of $347 - 257 = 90$ in MR (which is about 26% relative improvement) and $0.396 - 0.294 = 0.102$ in MRR (which is 34+% relative improvement), and also obtains $51.7\% - 46.5\% = 5.2\%$ absolute

[1] Some previous works also reported Hits@1. However, the formulas of MRR and Hits@1 show a strong correlation between these two scores. So using Hits@1 does not really reveal any additional information for this task.

[2] We employ a TransE implementation available at: https://github.com/datquocnguyen/STransE

Table 3

Link prediction results on WN18RR and FB15k-237 test sets. MRR and H@10 denote the mean reciprocal rank and Hits@10 (in %), respectively. [⋆]: Results are taken from Dettmers et al. [10] where Hits@10 and MRR are rounded to 2 decimal places on WN18RR. The last 4 rows report results of models that exploit information about relation paths (KB_{LRN}, R-GCN+ and Neural LP) or textual mentions derived from a large external corpus (Node+LinkFeat). The best score is in **bold**, while the second best score is in underline.

Method	WN18RR			FB15k-237		
	MR	MRR	H@10	MR	MRR	H@10
IRN [39]	–	–	–	211	–	46.4
KBGAN [7]	–	0.213	48.1	–	0.278	45.8
DISTMULT [62] [⋆]	5110	0.43	49	254	0.241	41.9
ComplEx [48] [⋆]	5261	0.44	51	339	0.247	42.8
ConvE [10]	5277	**0.46**	48	246	0.316	49.1
TransE [5] (our results)	3384	0.226	50.1	347	0.294	46.5
Our ConvKB model	**2554**	0.248	**52.5**	257	**0.396**	**51.7**
KB_{LRN} [14]	–	–	–	**209**	0.309	49.3
R-GCN+ [37]	–	–	–	–	0.249	41.7
Neural LP [63]	–	–	–	–	0.240	36.2
Node+LinkFeat [45]	–	–	–	–	0.293	46.2

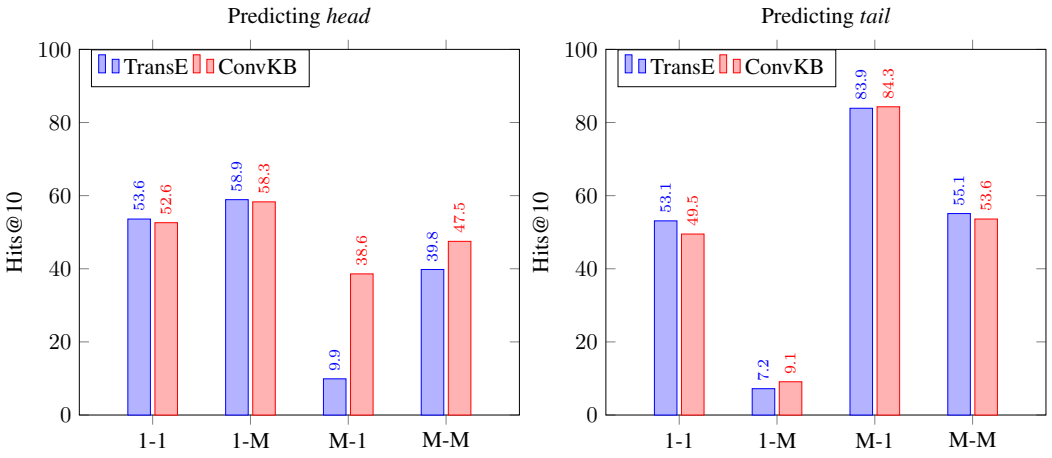

Figure 2. Hits@10 (in %) on the FB15k-237 test set w.r.t each relation category.

improvement in Hits@10. Previous work shows that TransE obtains very competitive results [26, 32, 36, 48]. However, when comparing the CNN-based embedding model ConvE with other embedding models, Dettmers et al. [10] did not experiment with TransE. We reconfirm previous findings that TransE in fact is a strong baseline model, e.g., TransE obtains better MR and Hits@10 than ConvE on WN18RR.

ConvKB obtains better scores than ConvE on both datasets (except MRR on WN18RR and MR on FB15k-237), thus showing the usefulness of taking transitional characteristics into accounts. In particular, on FB15k-237, ConvKB achieves improvements of $0.394 - 0.316 = 0.078$ in MRR (which is about 25% relative improvement) and $51.7\% - 49.1\% = 2.6\%$ in Hits@10, while both ConvKB and ConvE produce similar MR scores. ConvKB also obtains 25% relatively higher MRR score than the relation path-based model KB_{LRN} on FB15k-237. In addition, ConvKB gives better Hits@10 than KB_{LRN}, however, KB_{LRN} gives better MR than ConvKB. We plan to extend ConvKB with relation path information to obtain better link prediction performance in future work.

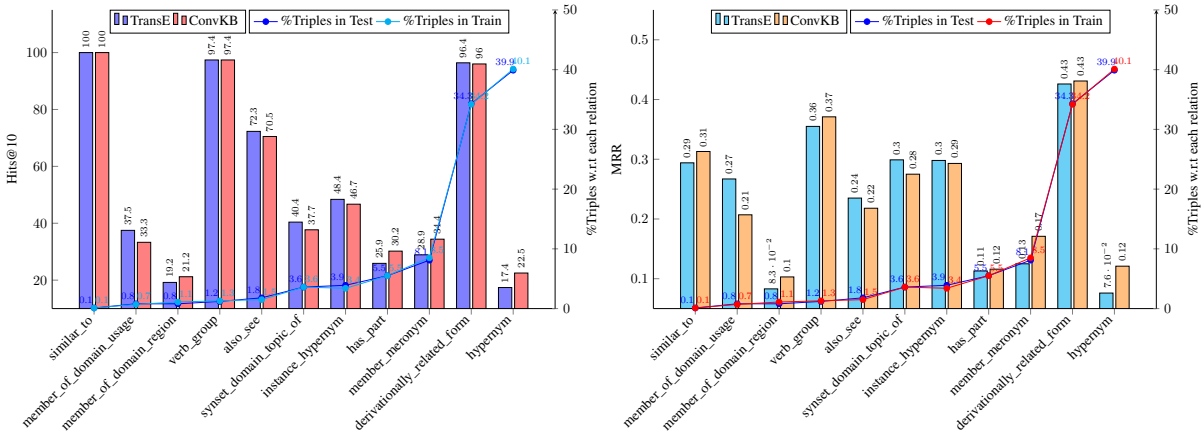

Figure 3. Hits@10 and MRR on the WN18RR test set w.r.t each relation. The right y-axis is the percentage of triples corresponding to relations.

Following Bordes et al. [5], we explore the Hits@10 results on the FB15k-237 test set corresponding to the relation categories. For each relation r, we calculate the averaged number η_h of heads per tail and the averaged number η_t of tails per head. If η_h <1.5 and η_t <1.5, r is classified as one-to-one (**1-1**). If η_h <1.5 and $\eta_t \geqslant$1.5, r is classified as one-to-many (**1-M**). If $\eta_h \geqslant$1.5 and η_t <1.5, r is classified as many-to-one (**M-1**). If $\eta_h \geqslant$1.5 and $\eta_t \geqslant$1.5, r is classified as many-to-many (**M-M**). We find that 17, 26, 81 and 113 relations are classified as 1-1, 1-M, M-1 and M-M, respectively. And 0.9%, 6.3%, 20.5% and 72.3% of the FB15k-237 test triples have their relations classified as 1-1, 1-M, M-1 and M-M, respectively.

Figure 2 shows the Hits@10 results for separately predicting head and tail entities on the FB15k-237 test set with respect to (w.r.t.) each relation category. We find that ConvKB is outperformed by TransE in 1-1 as 1-1 relations are relatively rare. We also find that both TransE and ConvKB are easier to predict entities on the relational "side 1" triples (i.e., predicting head entities in 1-1 and 1-M, and predicting tail entities in 1-1 and M-1). However, TransE is not good at predicting head entities in M-1 and M-M where TransE obtains the Hits@10 scores of 9.9% and 39.8%, while ConvKB is better in achieving the Hits@10 scores of 38.6% and 47.5%, respectively. A reason is probably that ConvKB could bring a generalization of projecting the embedding triples into the vector space of relations rather than TransE. Hence, this helps ConvKB to better modeling M-1 and M-M relations.

For a more concrete example, Figure 3 presents Hits@10 and MRR scores on WN18RR w.r.t. each relation type. *member_meronym* and *hypernym* are 1-M and M-1 relations, respectively. We find that TransE encounters a difficulty when dealing with these relation types. E.g., for 1,251 triples containing the relation *hypernym* from 3,134 test triples in the WN18RR test set, TransE only obtains the Hits@10 and MRR scores of 17.4% and 0.076 respectively, while ConvKB performs better than TransE and gets the Hits@10 and MRR scores of 22.5% and 0.121 respectively. In summary, figures 2 and 3 show that ConvKB are better at modeling 1-M, M-1 and M-M relations than TransE.

4.2. Triple classification

4.2.1. Task description

The triple classification task aims to predict whether a given triple (h, r, t) is valid or not [40]. Each relation r is associated with a threshold θ_r. For an unseen test triple (h, r, t), if its score is below θ_r then it will be classified as valid, otherwise invalid. Following Socher et al. [40], the relation-specific threshold θ_r is obtained by maximizing the micro-averaged classification accuracy on the validation set.

4.2.2. Training protocol

Similar to the training protocol in Section 4.1.2, we sample invalid triples using the common Bernoulli trick and also train TransE to produce entity and relation embeddings for initializing embeddings in ConvKB. The best accuracies obtained by TransE on the validation set are when using l_1-norm, learning rate at 0.01, γ = 7 and k = 50 for WN11, and using l_2-norm, learning rate at 0.01, γ = 1 and k = 100 for FB13. We then use a grid search to choose the hyper-parameter for ConvKB. We monitor the accuracy after each training epoch, and obtain the best accuracies on validation

Table 4
Accuracy results (in %) on the WN11 and FB13 test sets. The last 4 rows report accuracies of the models that use relation paths or incorporate with a large external corpus. The best score is in **bold** while the second best score is in underline. "Avg." denotes the averaged accuracy over two datasets.

Method	WN11	FB13	Avg.
NTN [40]	70.6	87.2	78.9
TransH [54]	78.8	83.3	81.1
TransR [27]	85.9	82.5	84.2
TransD [19]	86.4	**89.1**	87.8
TransR-FT [12]	86.6	82.9	84.8
TranSparse-S [20]	86.4	88.2	87.3
TranSparse-US [20]	86.8	87.5	87.2
ManifoldE [57]	87.5	87.2	87.4
TransG [58]	87.4	87.3	87.4
lppTransD [65]	86.2	88.6	87.4
TransE [5] (our results)	86.5	87.5	87.0
Our ConvKB model	**87.6**	88.8	**88.2**
TransE-NMM [32]	86.8	88.6	87.7
TEKE_H [53]	84.8	84.2	84.5
Bilinear-COMP [16]	77.6	86.1	81.9
TransE-COMP [16]	80.3	87.6	84.0

set when using k = 50, τ = 200, the truncated normal distribution for filter initialization, and the initial learning rate at $5e^{-4}$ on WN11; and k = 100, τ = 200, also the truncated normal distribution for filter initialization, and the initial learning rate at $5e^{-5}$ on FB13.

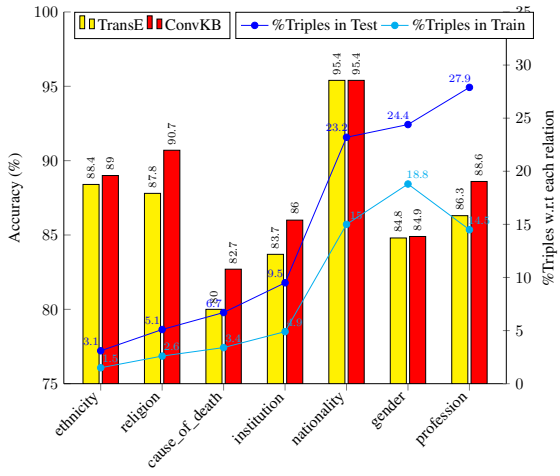

Figure 4. Accuracy results on the FB13 test set w.r.t each relation. The right y-axis is the number of triples corresponding to relations.

4.2.3. Main results

Table 4 presents the accuracy results of our ConvKB model and previous published results on the WN11 and FB13 datasets. On WN11, ConvKB obtains an accuracy of 87.6% which outperforms all other models. On FB13, ConvKB gains a second highest accuracy of 88.8% which is 0.3% outperformed by TransD. Compared to TransE, ConvKB absolutely improves by 1.1% on WN11 and 1.3% on FB13. Overall, ConvKB yields the best performance averaged over these two benchmark datasets. This also indicates the generalization of ConvKB over different datasets.

Regarding to TransE, Table 4 demonstrates that we obtain very competitive accuracies of 86.5% and 87.5% on WN11 and FB13 respectively. On WN11, TransE is comparable with TransD, TransR-FT and TranSparse-S while it scores better than lppTransD and TransE-COMP. On FB13, TransE performs slightly better than ManifoldE and TransG while it achieves similar scores in comparison with TransE-COMP and TranSparse-US. Note that TransR, TranSparse-S/US and TransD also perform the embedding initialization using TransE outputs (but these models do not report their TransE accuracy results). Hence, these models might get better results when using our TransE results as shown in this paper.

Figure 4 visualizes the accuracy results of different relations on FB13 for TransE and ConvKB. Relations *institution* and *profession* can be categorized as M-M where ConvKB is about 2.3% absolute higher accuracy than TransE, while remaining relations can be categorized as M-1. In short, Figure 4 shows that ConvKB performs equal to or better than TransE for all 7 relations in the FB13 test set.

5. Application for search personalization

Search personalization, an important feature of commercial search engines, has been recently attracted much attention from both academia [6, 8, 11, 28, 30, 49, 51, 64] and industry (e.g., Bing, Google, Airbnb [15]). Unlike classical searching methods, personalized search systems utilize the historical interactions such as submitted queries and clicked documents between a user and the systems to tailor returned search results to the needs of that user [1, 18, 42, 43, 56]. That historical information can be used to build the *user* profile, which is crucial to effective personalization [42, 43, 50, 52].

Table 5
Basic statistics of the dataset [51].

# users	106
#distinct queries	6,632
#SAT clicks	8,052
#distinct documents	33,591

Given a *user*, a submitted *query* and the *documents* returned by a search system for that query, our approach is to re-rank the returned documents so that the more relevant documents should be ranked higher. Following Vu et al. [51], we represent the relationship between the submitted query, the user and the returned document as a (h, r, t)-like triple *(query, user, document)*. The triple captures how much interest a user puts on a document given a query. Therefore, we can also evaluate the effectiveness of our ConvKB model for the search personalization task.

We evaluate ConvKB using the search results returned by a commercial search engine. We use the same dataset of query logs of 106 anonymous users from Vu et al. [51]. A log entity consists of a user identifier, a query, top-10 returned documents ranked by the search engine and clicked documents along with the user's dwell time. Vu et al. [51] employed the SAT criteria [13] to identify whether or not a clicked document is relevant from the query logs (i.e., a SAT click). They then assigned a *relevant* label to a returned document if it is a SAT click and also assigned *irrelevant* labels to the remaining top-10 documents. The rank position of the *relevant* labeled documents is used as the ground truth to evaluate the search performance before and after re-ranking. As a result, the dataset contains 8,052 valid triples *(query, user, relevant document)* in which 5,658, 1,184 and 1,210 valid triples are used for training, validation and test, respectively. Table 5 presents the dataset statistics.

5.1. Evaluation protocol

Our ConvKB model is used to re-rank the original list of top-10 documents returned by the commercial search engine as follows: (1) We train ConvKB and use the trained model to calculate a score for each triple $(question, user, document)$. (2) We then sort the scores in the ascending order to achieve a new ranked list. To evaluate the performance, we use two common metrics in document ranking: MRR and Hits@1.[3]

5.2. Training protocol

5.2.1. Query and document embedding initialization

We initialize query and document embeddings for ConvKB and the baseline TransE, then fix query and document embeddings (i.e. not updating these embeddings) during training.

To initialize document embeddings, we follow Vu et al. [51] to train a LDA topic model [2] with 200 topics only on the *relevant* documents (i.e., SAT clicks) extracted from the query logs. We then use the trained LDA model to infer the probability distribution over topics for each document. We use the topic proportion vector of each document as its document embedding (i.e. $k = 200$). In particular, the z^{th} element ($z = 1, 2, ..., k$) of the vector embedding for document d is: $\boldsymbol{v}_{d,z} = \mathrm{P}(z \mid d)$ where $\mathrm{P}(z \mid d)$ is the probability of the topic z given the document d.

We also represent each query by a probability distribution vector over topics. Let $\mathcal{D}_q = \{d_1, d_2, ..., d_n\}$ be the set of top n ranked documents returned for a query q (here, $n = 10$). The z^{th} element of the vector embedding for query q is defined as in [51]: $\boldsymbol{v}_{q,z} = \sum_{i=1}^{n} \lambda_i \mathrm{P}(z \mid d_i)$, where $\lambda_i = \frac{\delta^{i-1}}{\sum_{j=1}^{n} \delta^{j-1}}$ is the exponential decay function of i which is the rank of d_i in D_q. And δ is the decay hyper-parameter ($0 < \delta < 1$).

5.2.2. Hyper-parameter tuning

Similar to the training protocol presented in Section 4.1.2, we run model up to 200 epochs and perform a grid search to choose optimal hyper-parameters on the validation set. Following Vu et al. [51], we use $\delta = 0.8$. We also monitor the MRR score after each training epoch and obtain the highest MRR score on the validation set when using the *margin* at 5, l_1-norm and learning rate at $5e^{-3}$ for TransE; and using $\tau = 500$, the truncated normal distribution for filter initialization, and the initial learning rate at $5e^{-4}$ for ConvKB.

5.3. Results

Table 6 presents the experimental results of ConvKB, TransE and the previous published results of other strong baselines, in which ConvKB obtains highest MRR and Hits@1 scores. In particular, ConvKB

[3] We re-rank the list of top-10 documents returned by the search engine, so all models obtain the same Hits@10 scores.

Table 6
Experimental results on the test set. ⋆ denotes the results reported in Vu et al. [51]. **SE**: The original rank is returned by the search engine. **CI**: This baseline use a personalized navigation method based on previously clicking returned documents [44]. **SP**: A search personalization method makes use of the short-term profiles [1, 50]. The subscripts denote the relative improvement over the baseline **TransE**.

Model	**MRR**	**Hits@1** (%)
SE [⋆]	0.559	38.5
CI [44] [⋆]	0.597	41.6
SP [1, 50] [⋆]	0.631	45.2
TransE [5] [⋆]	0.645	48.1
STransE [33] [⋆]	0.656	50.1
TransE (our results)	0.669	50.9
Our **ConvKB** model	$0.750_{+12.1\%}$	$59.9_{+17.7\%}$

does significantly better than TransE with relative improvements at 12.1% for MRR and 17.7% for Hits@1. It is probably because our model not only can capture richer relational characteristics within the triple but also generalize the transitional relationships between embeddings of user queries and relevant documents for user profiles. We also obtain higher TransE results than those reported in Vu et al. [51]. The reason is that for each valid triple, rather than using only one invalid triple as in [51], we take into account its all invalid triples to train TransE (each valid or invalid triple contains a relevant- or irrelevant-labelled document, respectively).

6. Conclusion

In this paper, we propose a novel embedding model ConvKB for the knowledge base completion task. ConvKB applies the convolutional neural network to explore the global relationships among same dimensional entries of the entity and relation embeddings, so that ConvKB generalizes the transitional characteristics in the transition-based embedding models. Experimental results show that our ConvKB model outperforms other state-of-the-art models on two benchmark datasets WN18RR and FB15k-237 for the link prediction task, and on two other benchmark datasets WN11 and FB13 for the triple classification task. ConvKB obtains the best mean rank and the highest Hits@10 on WN18RR and obtains the highest mean reciprocal rank and Hits@10 on FB15k-237. In addition, ConvKB produces the best accuracy on WN11 and the second best accuracy on FB13. Moreover, we show the effectiveness of ConvKB for search personalization, in which ConvKB outperforms the strong baselines on the query logs of a commercial web search engine.

In the future work, we plan to extend ConvKB with relation path information to achieve better performance. We will also adapt ConvKB to other personalization tasks where we can model each task as a triple relationship, e.g. in personalized query suggestion or auto-completion.

Our ConvKB implementation is available at: https://github.com/daiquocnguyen/ConvKB.

Bibliographic note

This paper extends our paper [34] published in Proceedings of NAACL-HLT 2018 (Volume 2: Short Papers). We first add a significantly improved analysis to the link prediction task. Then we conduct new extensive empirical study on the triple classification and search personalization tasks.

Acknowledgement

This research was partially supported by the Australian Research Council (ARC) DP150100031 and DP160103934.

References

[1] P. N. Bennett, R. W. White, W. Chu, S. T. Dumais, P. Bailey, F. Borisyuk, and X. Cui. Modeling the impact of short- and long-term behavior on search personalization. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, pages 185–194, Portland, Oregon, USA, 2012. ACM. URL http://dx.doi.org/10.1145/2348283.2348312.

[2] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 601–608. MIT Press, 2002. URL http://papers.nips.cc/paper/2070-latent-dirichlet-allocation.pdf.

[3] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1247–1250, Vancouver, Canada, 2008. ACM. ISBN 978-1-60558-102-6. URL https://doi.org/10.1145/1376616.1376746.

[4] A. Bordes, J. Weston, R. Collobert, and Y. Bengio. Learning Structured Embeddings of Knowledge Bases. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, AAAI'11, pages 301–306, San Francisco, California, 2011. AAAI Press. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3659.

[5] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2787–2795. Curran Associates, Inc., 2013.

[6] F. Cai, S. Wang, and M. de Rijke. Behavior-based personalization in web search. *Journal of the Association for Information Science and Technology*, 68(4):855–868, 2017. URL https://doi.org/10.1002/asi.23735.

[7] L. Cai and W. Y. Wang. KBGAN: Adversarial learning for knowledge graph embeddings. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1470–1480, New Orleans, Louisiana, 2018. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N18-1133.

[8] Z. Cheng, S. Jialie, and S. C. Hoi. On effective personalized music retrieval by exploring online user behaviors. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, pages 125–134, Pisa, Italy, 2016. ACM. URL http://doi.org/10.1145/2911451.2911491.

[9] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011. URL www.jmlr.org/papers/volume12/collobert11a/collobert11a.pdf.

[10] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel. Convolutional 2D knowledge graph embeddings. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, AAAI'18, pages 1811–1818. AAAI Press, 2018. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17366.

[11] Z. Dou, R. Song, and J.-R. Wen. A large-scale evaluation and analysis of personalized search strategies. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 581–590, Banff, Alberta, Canada, 2007. International World Wide Web Conferences Steering Committee. URL http://doi.org/10.1145/1242572.1242651.

[12] J. Feng, M. Huang, M. Wang, M. Zhou, Y. Hao, and X. Zhu. Knowledge graph embedding by flexible translation. In *Proceedings of the Fifteenth International Conference on Principles of Knowledge Representation and Reasoning*, KR'16, pages 557–560, Cape Town, South Africa, 2016. AAAI Press. URL https://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12887.

[13] S. Fox, K. Karnawat, M. Mydland, S. Dumais, and T. White. Evaluating Implicit Measures to Improve Web Search. *ACM Transactions on Information Systems*, 23(2):147–168, 2005. URL http://doi.org/10.1145/1059981.1059982.

[14] A. García-Durán and M. Niepert. KBLRN: End-to-End Learning of Knowledge Base Representations with Latent, Relational, and Numerical Features. *arXiv preprint abs/1709.04676*, 2017. URL https://arxiv.org/abs/1709.04676.

[15] M. Grbovic. Search ranking and personalization at Airbnb. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, pages 339–340, Como, Italy, 2017. ACM. URL http://doi.org/10.1145/3109859.3109920.

[16] K. Guu, J. Miller, and P. Liang. Traversing knowledge graphs in vector space. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 318–327, Lisbon, Portugal, 2015. Association for Computational Linguistics. URL http://aclweb.org/anthology/D15-1038.

[17] Y. Hao, Y. Zhang, K. Liu, S. He, Z. Liu, H. Wu, and J. Zhao. An end-to-end model for question answering over knowledge base with cross-attention combining global knowledge. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 221–231, Vancouver, Canada, 2017. Association for Computational Linguistics. URL http://aclweb.org/anthology/P17-1021.

[18] M. Harvey, F. Crestani, and M. J. Carman. Building user profiles from topic models for personalised search. In *Proceedings of the 22Nd ACM International Conference on Information & Knowledge Management*, CIKM '13, pages 2309–2314, San Francisco, California, USA, 2013. ACM. URL http://doi.org/10.1145/2505515.2505642.

[19] G. Ji, S. He, L. Xu, K. Liu, and J. Zhao. Knowledge graph embedding via dynamic mapping matrix. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 687–696, Beijing, China, 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P15-1067.

[20] G. Ji, K. Liu, S. He, and J. Zhao. Knowledge graph completion with adaptive sparse transfer matrix. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 985–991, Phoenix, Arizona, 2016. AAAI Press. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/11982.

[21] G. Kasneci, F. M. Suchanek, G. Ifrim, M. Ramanath, and G. Weikum. Naga: Searching and ranking knowledge. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 953–962, Washington, DC, USA, 2008. IEEE Computer Society. URL https://doi.org/10.1109/ICDE.2008.4497504.

[22] Y. Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D14-1181.

[23] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. URL https://arxiv.org/abs/1412.6980.

[24] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. URL http://doi.org/10.1109/5.726791.

[25] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer, et al. DBpedia – a large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, 6:167–195, 2015. URL https://doi.org/10.3233/SW-140134.

[26] Y. Lin, Z. Liu, H. Luan, M. Sun, S. Rao, and S. Liu. Modeling relation paths for representation learning of knowledge bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 705–714, Lisbon, Portugal, 2015. Association for Computational Linguistics. URL http://aclweb.org/anthology/D15-1082.

[27] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 2181–2187, Austin, Texas, 2015. AAAI Press. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9571.

[28] X. Liu. Modeling users' dynamic preference for personalized recommendation. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 1785–1791, Buenos Aires, Argentina, 2015. AAAI Press. URL https://www.ijcai.org/Proceedings/15/Papers/254.pdf.

[29] Y. Luo, Q. Wang, B. Wang, and L. Guo. Context-dependent knowledge graph embedding. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1656–1661, Lisbon, Portugal, 2015. Association for Computational Linguistics. URL http://aclweb.org/anthology/D15-1191.

[30] N. Nanas, M. Vavalis, and A. De Roeck. A network-based model for high-dimensional information filtering. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pages 202–209, Geneva, Switzerland, 2010. ACM. URL http://doi.org/10.1145/1835449.1835485.

[31] D. Q. Nguyen. An overview of embedding models of entities and relationships for knowledge base completion. *arXiv preprint*, arXiv:1703.08098, 2017. URL https://arxiv.org/abs/1703.08098.

[32] D. Q. Nguyen, K. Sirts, L. Qu, and M. Johnson. Neighborhood mixture model for knowledge base completion. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 40–50, Berlin, Germany, 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/K16-1005.

[33] D. Q. Nguyen, K. Sirts, L. Qu, and M. Johnson. Stranse: a novel embedding model of entities and relationships in knowledge bases. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 460–466, San Diego, California, 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N16-1054.

[34] D. Q. Nguyen, T. D. Nguyen, D. Q. Nguyen, and D. Phung. A novel embedding model for knowledge base completion based on convolutional neural network. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 327–333, New Orleans, Louisiana, 2018. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N18-2053.

[35] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A Review of Relational Machine Learning for Knowledge Graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016. URL https://doi.org/10.1109/JPROC.2015.2483592.

[36] M. Nickel, L. Rosasco, and T. Poggio. Holographic embeddings of knowledge graphs. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 1955–1961, Phoenix, Arizona, 2016. AAAI Press. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12484.

[37] M. Schlichtkrull, T. Kipf, P. Bloem, R. v. d. Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. *arXiv preprint arXiv:1703.06103*, 2017. URL https://arxiv.org/abs/1703.06103.

[38] M. Schuhmacher and S. P. Ponzetto. Knowledge-based graph document modeling. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, pages 543–552, New York, New York, USA, 2014. ACM. URL http://doi.org/10.1145/2556195.2556250.

[39] Y. Shen, P. Huang, M. Chang, and J. Gao. Traversing knowledge graph in vector space without symbolic space guidance. *arXiv preprint arXiv:1611.04642v4*, 2017. URL https://arxiv.org/abs/1611.04642v4.

[40] R. Socher, D. Chen, C. D. Manning, and A. Ng. Reasoning with neural tensor networks for knowledge base completion. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 926–934. Curran Associates, Inc., 2013.

[41] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 697–706, Banff, Alberta, Canada, 2007. International World Wide Web Conferences Steering Committee. URL http://doi.org/10.1145/1242572.1242667.

[42] J. Teevan, S. T. Dumais, and E. Horvitz. Personalizing search via automated analysis of interests and activities. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 449–456, Salvador, Brazil, 2005. ACM. URL http://doi.org/10.1145/1076034.1076111.

[43] J. Teevan, M. R. Morris, and S. Bush. Discovering and using groups to improve personalized search. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, WSDM '09, pages 15–24, Barcelona, Spain, 2009. ACM. URL http://doi.org/10.1145/1498759.1498786.

[44] J. Teevan, D. J. Liebling, and G. Ravichandran Geetha. Understanding and predicting personal navigation. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, WSDM '11, pages 85–94, Hong Kong, China, 2011. ACM. URL http://doi.org/10.1145/1935826.1935848.

[45] K. Toutanova and D. Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pages 57–66, Beijing, China, 2015. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W15-4007.

[46] K. Toutanova, D. Chen, P. Pantel, H. Poon, P. Choudhury, and M. Gamon. Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509, Lisbon, Portugal, 2015. Association for Computational Linguistics. URL http://aclweb.org/anthology/D15-1174.

[47] K. Toutanova, V. Lin, W.-t. Yih, H. Poon, and C. Quirk. Compositional learning of embeddings for relation paths in knowledge base and text. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1434–1444, Berlin, Germany, 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P16-1136.

[48] T. Trouillon, J. Welbl, S. Riedel, E. Gaussier, and G. Bouchard. Complex embeddings for simple link prediction. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 2071–2080, New York, NY, USA, 2016. JMLR.org. URL http://proceedings.mlr.press/v48/trouillon16.pdf.

[49] T. Vu, D. Song, A. Willis, S. N. Tran, and J. Li. Improving search personalisation with dynamic group formation. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '14, pages 951–954, Gold Coast, Queensland, Australia, 2014. ACM. URL http://doi.org/10.1145/2600428.2609482.

[50] T. Vu, A. Willis, S. N. Tran, and D. Song. Temporal latent topic user profiles for search personalisation. In *Proceedings of the European Conference on Information Retrieval*, pages 605–616, Vienna, Austria, 2015. Springer International Publishing. URL https://doi.org/10.1007/978-3-319-16354-3_67.

[51] T. Vu, D. Q. Nguyen, M. Johnson, D. Song, and A. Willis. Search Personalization with Embeddings. In *Proceedings of the European Conference on Information Retrieval*, pages 598–604, Aberdeen, Scotland, 2017. Springer International Publishing. URL https://doi.org/10.1007/978-3-319-56608-5_54.

[52] T. Vu, A. Willis, U. Kruschwitz, and D. Song. Personalised query suggestion for intranet search with temporal user profiling. In *Proceedings of the 2017 Conference on Conference Human Information Interaction and Retrieval*, CHIIR '17, pages 265–268, Oslo, Norway, 2017. ACM. URL http://doi.org/10.1145/3020165.3022129.

[53] Z. Wang and J. Li. Text-enhanced representation learning for knowledge graph. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 1293–1299, New York, New York, USA, 2016. AAAI Press. URL https://www.ijcai.org/Proceedings/16/Papers/187.pdf.

[54] Z. Wang, J. Zhang, J. Feng, and Z. Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, pages 1112–1119, Quebec City, Quebec, Canada, 2014. AAAI Press. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8531.

[55] R. West, E. Gabrilovich, K. Murphy, S. Sun, R. Gupta, and D. Lin. Knowledge base completion via search-based question answering. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 515–526, Seoul, Korea, 2014. International World Wide Web Conferences Steering Committee. URL http://doi.org/10.1145/2566486.2568032.

[56] R. W. White, W. Chu, A. Hassan, X. He, Y. Song, and H. Wang. Enhancing personalized search by mining and modeling task behavior. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13, pages 1411–1420, Rio de Janeiro, Brazil, 2013. International World Wide Web Conferences Steering Committee. URL http://doi.org/10.1145/2488388.2488511.

[57] H. Xiao, M. Huang, and X. Zhu. From one point to a manifold: Knowledge graph embedding for precise link prediction. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 1315–1321, New York, New York, USA, 2016. AAAI Press. URL https://www.ijcai.org/Proceedings/16/Papers/190.pdf.

[58] H. Xiao, M. Huang, and X. Zhu. Transg : A generative model for knowledge graph embedding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2316–2325, Berlin, Germany, 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P16-1219.

[59] Q. Xie, X. Ma, Z. Dai, and E. Hovy. An interpretable knowledge transfer model for knowledge base completion. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 950–962, Vancouver, Canada, 2017. Association for Computational Linguistics. URL http://aclweb.org/anthology/P17-1088.

[60] C. Xiong, R. Power, and J. Callan. Explicit semantic ranking for academic search via knowledge graph embedding. In *Proceedings of the 26th International Conference on World Wide Web*, WWW '17, pages 1271–1279, Perth, Australia, 2017. International World Wide Web Conferences Steering Committee. URL https://doi.org/10.1145/3038912.3052558.

[61] B. Yang and T. Mitchell. Leveraging knowledge bases in lstms for improving machine reading. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1436–1446, Vancouver, Canada, 2017. Association for Computational Linguistics. URL http://aclweb.org/anthology/P17-1132.

[62] B. Yang, W.-t. Yih, X. He, J. Gao, and L. Deng. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *Proceedings of the International Conference on Learning Representations*, 2015. URL https://arxiv.org/abs/1412.6575.

[63] F. Yang, Z. Yang, and W. W. Cohen. Differentiable learning of logical rules for knowledge base reasoning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2319–2328. Curran Associates, Inc., 2017.

[64] L. Yang, Q. Guo, Y. Song, S. Meng, M. Shokouhi, K. McDonald, and W. B. Croft. Modeling User Interests for Zero-Query Ranking. In *Proceedings of the European Conference on Information Retrieval*, pages 171–184. Springer International Publishing, 2016. URL https://doi.org/10.1007/978-3-319-30671-1_13.

[65] H.-G. Yoon, H.-J. Song, S.-B. Park, and S.-Y. Park. A translation-based knowledge graph embedding preserving logical property of relations. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 907–916, San Diego, California, 2016. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/N16-1105.

[66] Y. Zhang, K. Liu, S. He, G. Ji, Z. Liu, H. Wu, and J. Zhao. Question Answering over Knowledge Base with Neural Attention Combining Global Knowledge Information. *arXiv preprint arXiv:1606.00979*, 2016. URL https://arxiv.org/abs/1606.00979.

A Capsule Network-based Embedding Model for Knowledge Graph Completion and Search Personalization

Dai Quoc Nguyen[1], **Thanh Vu**[2], **Tu Dinh Nguyen**[1], **Dat Quoc Nguyen**[3], **Dinh Phung**[1]
[1]Monash University, Australia; [3]The University of Melbourne, Australia
[2]The Australian e-Health Research Centre, CSIRO, Australia
[1]{dai.nguyen,tu.dinh.nguyen,dinh.phung}@monash.edu
[2]thanh.vu@csiro.au; [3]dqnguyen@unimelb.edu.au

Abstract

In this paper, we introduce an embedding model, named CapsE, exploring a capsule network to model relationship triples *(subject, relation, object)*. Our CapsE represents each triple as a 3-column matrix where each column vector represents the embedding of an element in the triple. This 3-column matrix is then fed to a convolution layer where multiple filters are operated to generate different feature maps. These feature maps are reconstructed into corresponding capsules which are then routed to another capsule to produce a continuous vector. The length of this vector is used to measure the plausibility score of the triple. Our proposed CapsE obtains better performance than previous state-of-the-art embedding models for knowledge graph completion on two benchmark datasets WN18RR and FB15k-237, and outperforms strong search personalization baselines on SEARCH17.

1 Introduction

Knowledge graphs (KGs) containing relationship triples *(subject, relation, object)*, denoted as *(s, r, o)*, are the useful resources for many NLP and especially information retrieval applications such as semantic search and question answering (Wang et al., 2017). However, large knowledge graphs, even containing billions of triples, are still incomplete, i.e., missing a lot of valid triples (West et al., 2014). Therefore, much research efforts have focused on the knowledge graph completion task which aims to predict missing triples in KGs, i.e., predicting whether a triple not in KGs is likely to be valid or not (Bordes et al., 2011, 2013; Socher et al., 2013). To this end, many embedding models have been proposed to learn vector representations for entities (i.e., *subject*/head entity and *object*/tail entity) and relations in KGs, and obtained state-of-the-art results as summarized by Nickel et al. (2016a) and Nguyen (2017). These embedding models score triples *(s, r, o)*, such that valid triples have higher plausibility scores than invalid ones (Bordes et al., 2011, 2013; Socher et al., 2013). For example, in the context of KGs, the score for *(Melbourne, cityOf, Australia)* is higher than the score for *(Melbourne, cityOf, United Kingdom)*.

Triple modeling is applied not only to the KG completion, but also for other tasks which can be formulated as a triple-based prediction problem. An example is in search personalization, one would aim to tailor search results to each specific user based on the user's personal interests and preferences (Teevan et al., 2005, 2009; Bennett et al., 2012; Harvey et al., 2013; Vu et al., 2015, 2017). Here the triples can be formulated as *(submitted query, user profile, returned document)* and used to re-rank documents returned to a user given an input query, by employing an existing KG embedding method such as TransE (Bordes et al., 2013), as proposed by Vu et al. (2017). Previous studies have shown the effectiveness of modeling triple for either KG completion or search personalization. However, there has been no single study investigating the performance on both tasks.

Conventional embedding models, such as TransE (Bordes et al., 2013), DISTMULT (Yang et al., 2015) and ComplEx (Trouillon et al., 2016), use addition, subtraction or simple multiplication operators, thus only capture the linear relationships between entities. Recent research has raised interest in applying deep neural networks to triple-based prediction problems. For example, Nguyen et al. (2018) proposed ConvKB—a convolutional neural network (CNN)-based model for KG completion and achieved state-of-the-art results. Most of KG embedding models are constructed to modeling entries at the same dimension of the given triple, where presumably each dimension captures some relation-specific attribute of entities. To the

best of our knowledge, however, none of the existing models has a "deep" architecture for modeling the entries in a triple at the same dimension.

Sabour et al. (2017) introduced capsule networks (CapsNet) that employ capsules (i.e., *each capsule is a group of neurons*) to capture entities in images and then uses a routing process to specify connections from capsules in a layer to those in the next layer. Hence CapsNet could encode the intrinsic spatial relationship between a part and a whole constituting viewpoint invariant knowledge that automatically generalizes to novel viewpoints. Each capsule accounts for capturing variations of an object or object part in the image, which can be efficiently visualized. Our high-level hypothesis is that embedding entries at the same dimension of the triple also have these variations, although it is not straightforward to be visually examined.

To that end, we introduce CapsE to explore a novel application of CapsNet on triple-based data for two problems: KG completion and search personalization. Different from the traditional modeling design of CapsNet where capsules are constructed by splitting feature maps, we use capsules to model the entries at the same dimension in the entity and relation embeddings. In our CapsE, $\boldsymbol{v}_s$, $\boldsymbol{v}_r$ and $\boldsymbol{v}_o$ are unique k-dimensional embeddings of s, r and o, respectively. The embedding triple $[\boldsymbol{v}_s, \boldsymbol{v}_r, \boldsymbol{v}_o]$ of *(s, r, o)* is fed to the convolution layer where multiple filters of the same 1×3 shape are repeatedly operated over every row of the matrix to produce k-dimensional feature maps. Entries at the same dimension from all feature maps are then encapsulated into a capsule. Thus, *each capsule can encode many characteristics in the embedding triple to represent the entries at the corresponding dimension*. These capsules are then routed to another capsule which outputs a continuous vector whose length is used as a score for the triple. Finally, this score is used to predict whether the triple *(s, r, o)* is valid or not.

In summary, our main contributions from this paper are as follows:

• We propose an embedding model CapsE using the capsule network (Sabour et al., 2017) for modeling relationship triples. To our best of knowledge, our work is the first consideration of exploring the capsule network to knowledge graph completion and search personalization.

• We evaluate our CapsE for knowledge graph completion on two benchmark datasets WN18RR (Dettmers et al., 2018) and FB15k-237 (Toutanova and Chen, 2015). CapsE obtains the best mean rank on WN18RR and the highest mean reciprocal rank and highest Hits@10 on FB15k-237.

• We restate the prospective strategy of expanding the triple embedding models to improve the ranking quality of the search personalization systems. We adapt our model to search personalization and evaluate on SEARCH17 (Vu et al., 2017) – a dataset of the web search query logs. Experimental results show that our CapsE achieves the new state-of-the-art results with significant improvements over strong baselines.

2 The proposed CapsE

Let $\mathcal{G}$ be a collection of valid factual triples in the form of *(subject, relation, object)* denoted as *(s, r, o)*. Embedding models aim to define a *score function* giving a score for each triple, such that valid triples receive higher scores than invalid triples.

We denote $\boldsymbol{v}_s$, $\boldsymbol{v}_r$ and $\boldsymbol{v}_o$ as the k-dimensional embeddings of s, r and o, respectively. In our proposed CapsE, we follow Nguyen et al. (2018) to view each embedding triple $[\boldsymbol{v}_s, \boldsymbol{v}_r, \boldsymbol{v}_o]$ as a matrix $\boldsymbol{A} = [\boldsymbol{v}_s, \boldsymbol{v}_r, \boldsymbol{v}_o] \in \mathbb{R}^{k \times 3}$, and denote $\boldsymbol{A}_{i,:} \in \mathbb{R}^{1 \times 3}$ as the i-th row of $\boldsymbol{A}$. We use a filter $\boldsymbol{\omega} \in \mathbb{R}^{1 \times 3}$ operated on the convolution layer. This filter $\boldsymbol{\omega}$ is repeatedly operated over every row of $\boldsymbol{A}$ to generate a feature map $\mathbf{q} = [\mathsf{q}_1, \mathsf{q}_2, ..., \mathsf{q}_k] \in \mathbb{R}^k$, in which $\mathsf{q}_i = g\left(\boldsymbol{\omega} \cdot \boldsymbol{A}_{i,:} + b\right)$ where $\cdot$ denotes a dot product, $b \in \mathbb{R}$ is a bias term and g is a non-linear activation function such as ReLU. Our model uses multiple filters $\in \mathbb{R}^{1 \times 3}$ to generate feature maps. We denote $\boldsymbol{\Omega}$ as the set of filters and $\mathsf{N} = \mid \boldsymbol{\Omega} \mid$ as the number of filters, thus we have N k-dimensional feature maps, for which each feature map can capture one single characteristic among entries at the same dimension.

We build our CapsE with two single capsule layers for a simplified architecture. In the first layer, we construct k capsules, wherein entries at the same dimension from all feature maps are encapsulated into a corresponding capsule. Therefore, each capsule can capture many characteristics among the entries at the corresponding dimension in the embedding triple. These characteristics are generalized into one capsule in the second layer which produces a vector output whose length is used as the score for the triple.

The first capsule layer consists of k capsules, for which each capsule $i \in \{1, 2, ..., k\}$ has a vector

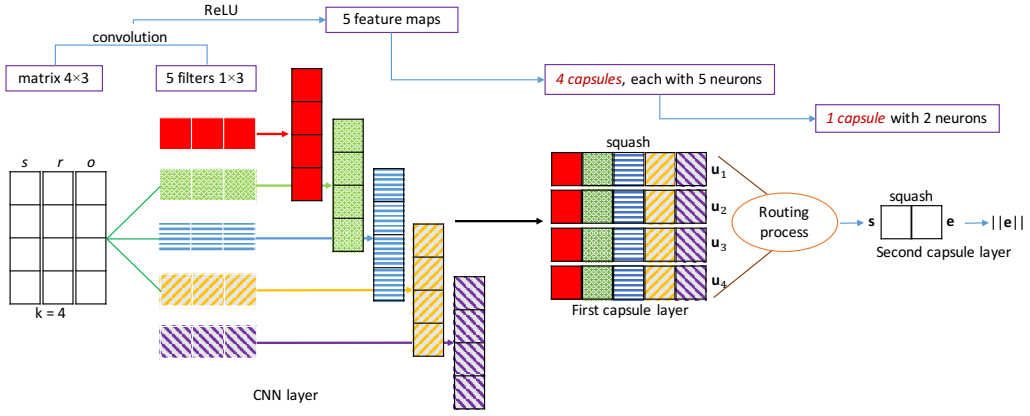

Figure 1: An example illustration of our CapsE with $k = 4$, $\mathsf{N} = 5$, and $d = 2$.

output $\mathbf{u}_i \in \mathbb{R}^{\mathsf{N}\times 1}$. Vector outputs $\mathbf{u}_i$ are multiplied by weight matrices $\mathbf{W}_i \in \mathbb{R}^{d\times \mathsf{N}}$ to produce vectors $\hat{\mathbf{u}}_i \in \mathbb{R}^{d\times 1}$ which are summed to produce a vector input $\mathbf{s} \in \mathbb{R}^{d\times 1}$ to the capsule in the second layer. The capsule then performs the non-linear squashing function to produce a vector output $\mathbf{e} \in \mathbb{R}^{d\times 1}$:

$$\mathbf{e} = \mathsf{squash}\left(\mathbf{s}\right) \quad ; \quad \mathbf{s} = \sum_i c_i \hat{\mathbf{u}}_i \quad ; \quad \hat{\mathbf{u}}_i = \mathbf{W}_i \mathbf{u}_i$$

where $\mathsf{squash}\left(\mathbf{s}\right) = \frac{\|\mathbf{s}\|^2}{1+\|\mathbf{s}\|^2}\frac{\mathbf{s}}{\|\mathbf{s}\|}$, and c_i are coupling coefficients determined by the routing process as presented in Algorithm 1. Because there is one capsule in the second layer, we make only one difference in the routing process proposed by Sabour et al. (2017), for which we apply the softmax in a direction from all capsules in the previous layer to each of capsules in the next layer.[1]

for *all capsule* $i \in$ *the first layer* **do**
  $b_i \leftarrow 0$
for iteration = *1, 2, ..., m* **do**
  $\mathbf{c} \leftarrow \mathsf{softmax}\left(\mathbf{b}\right)$
  $\mathbf{s} \leftarrow \sum_i c_i \hat{\mathbf{u}}_i$
  $\mathbf{e} = \mathsf{squash}\left(\mathbf{s}\right)$
  for *all capsule* $i \in$ *the first layer* **do**
    $b_i \leftarrow b_i + \hat{\mathbf{u}}_i \cdot \mathbf{e}$

Algorithm 1: The routing process is extended from Sabour et al. (2017).

[1]The softmax in the original routing process proposed by Sabour et al. (2017) is applied in another direction from each of capsules in the previous layer to all capsules in the next layer.

We illustrate our proposed model in Figure 1 where embedding size: $k = 4$, the number of filters: $\mathsf{N} = 5$, the number of neurons within the capsules in the first layer is equal to N, and the number of neurons within the capsule in the second layer: $d = 2$. The length of the vector output $\mathbf{e}$ is used as the score for the input triple.

Formally, we define the score function f for the triple (s, r, o) as follows:

$$f\left(s, r, o\right) = \|\mathsf{capsnet}\left(g\left(\left[\boldsymbol{v}_s, \boldsymbol{v}_r, \boldsymbol{v}_o\right] * \boldsymbol{\Omega}\right)\right)\|$$

where the set of filters $\boldsymbol{\Omega}$ is shared parameters in the convolution layer; $*$ denotes a convolution operator; and capsnet denotes a capsule network operator. We use the Adam optimizer (Kingma and Ba, 2014) to train CapsE by minimizing the loss function (Trouillon et al., 2016; Nguyen et al., 2018) as follows:

$$\mathcal{L} = \sum_{(s,r,o)\in\{\mathcal{G}\cup\mathcal{G}'\}} \log\left(1 + \exp\left(-t_{(s,r,o)} \cdot f\left(s, r, o\right)\right)\right)$$

$$\text{in which, } t_{(s,r,o)} = \begin{cases} 1 \text{ for } (s, r, o) \in \mathcal{G} \\ -1 \text{ for } (s, r, o) \in \mathcal{G}' \end{cases}$$

here $\mathcal{G}$ and $\mathcal{G}'$ are collections of valid and invalid triples, respectively. $\mathcal{G}'$ is generated by corrupting valid triples in $\mathcal{G}$.

3 Knowledge graph completion evaluation

In the knowledge graph completion task (Bordes et al., 2013), the goal is to predict a missing entity given a relation and another entity, i.e, inferring a head entity s given (r, o) or inferring a tail entity o given (s, r). The results are calculated based on ranking the scores produced by the score function f on test triples.

3.1 Experimental setup

Datasets: We use two recent benchmark datasets WN18RR (Dettmers et al., 2018) and FB15k-237 (Toutanova and Chen, 2015). These two datasets are created to avoid reversible relation problems, thus the prediction task becomes more realistic and hence more challenging (Toutanova and Chen, 2015). Table 1 presents the statistics of WN18RR and FB15k-237.

Dataset	#E	#R	#Triples in train/valid/test		
WN18RR	40,943	11	86,835	3,034	3,134
FB15k-237	14,541	237	272,115	17,535	20,466

Table 1: Statistics of the experimental datasets. #E is the number of entities. #R is the number of relations.

Evaluation protocol: Following Bordes et al. (2013), for each valid test triple (s, r, o), we replace either s or o by each of all other entities to create a set of corrupted triples. We use the "Filtered" setting protocol (Bordes et al., 2013), i.e., not taking any corrupted triples that appear in the KG into accounts. We rank the valid test triple and corrupted triples in descending order of their scores. We employ evaluation metrics: mean rank (MR), mean reciprocal rank (MRR) and Hits@10 (i.e., the proportion of the valid test triples ranking in top 10 predictions). Lower MR, higher MRR or higher Hits@10 indicate better performance. Final scores on the test set are reported for the model obtaining the highest Hits@10 on the validation set.

Training protocol: We use the common Bernoulli strategy (Wang et al., 2014; Lin et al., 2015b) when sampling invalid triples. For WN18RR, Pinter and Eisenstein (2018)[2] found a strong evidence to support the necessity of a WordNet-related semantic setup, in which they averaged pre-trained word embeddings for word surface forms within the WordNet to create synset embeddings, and then used these synset embeddings to initialize entity embeddings for training their TransE association model. We follow this evidence in using the pre-trained 100-dimensional Glove word embeddings (Pennington et al., 2014) to train a TransE model on WN18RR.

[2]Pinter and Eisenstein (2018) considered WN18RR and evaluated their M3GM model only for 7 relations as they employed the inverse rule model (Dettmers et al., 2018) for 4 remaining symmetric relations. Regarding a fair comparison to other models, we use the M3GM implementation released by Pinter and Eisenstein (2018) to re-train and re-evaluate the M3GM model for all 11 relations. We thank Pinter and Eisenstein (2018) for their assistance running their code.

We employ the TransE and ConvKB implementations provided by Nguyen et al. (2016b) and Nguyen et al. (2018). For ConvKB, we use a new process of training up to 100 epochs and monitor the Hits@10 score after every 10 training epochs to choose optimal hyper-parameters with the Adam initial learning rate in $\{1e^{-5}, 5e^{-5}, 1e^{-4}\}$ and the number of filters N in $\{50, 100, 200, 400\}$. We obtain the highest Hits@10 scores on the validation set when using N= 400 and the initial learning rate $5e^{-5}$ on WN18RR; and N= 100 and the initial learning rate $1e^{-5}$ on FB15k-237.

Like in ConvKB, we use the same pre-trained entity and relation embeddings produced by TransE to initialize entity and relation embeddings in our CapsE for both WN18RR and FB15k-237 ($k = 100$). We set the batch size to 128, the number of neurons within the capsule in the second capsule layer to 10 ($d = 10$), and the number of iterations in the routing algorithm m in $\{1, 3, 5, 7\}$. We run CapsE up to 50 epochs and monitor the Hits@10 score after each 10 training epochs to choose optimal hyper-parameters. The highest Hits@10 scores for our CapsE on the validation set are obtained when using $m = 1$, $\mathsf{N} = 400$ and the initial learning rate at $1e^{-5}$ on WN18RR; and $m = 1$, $\mathsf{N} = 50$ and the initial learning rate at $1e^{-4}$ on FB15k-237.

3.2 Main experimental results

Table 2 compares the experimental results of our CapsE with previous state-of-the-art published results, using the same evaluation protocol. Our CapsE performs better than its closely related CNN-based model ConvKB on both experimental datasets (except Hits@10 on WN18RR and MR on FB15k-237), especially on FB15k-237 where our CapsE gains significant improvements of $0.523 - 0.418 = 0.105$ in MRR (which is about 25.1% relative improvement), and $59.3\% - 53.2\% = 6.1\%$ absolute improvement in Hits@10. Table 2 also shows that our CapsE obtains the best MR score on WN18RR and the highest MRR and Hits@10 scores on FB15k-237.

Following Bordes et al. (2013), for each relation r in FB15k-237, we calculate the averaged number η_s of head entities per tail entity and the averaged number η_o of tail entities per head entity. If η_s <1.5 and η_o <1.5, r is categorized one-to-one (1-1). If η_s <1.5 and η_o ≥1.5, r is categorized one-to-many (1-M). If η_s ≥1.5 and η_o <1.5, r is

Method	WN18RR			FB15k-237		
	MR	MRR	H@10	MR	MRR	H@10
DISTMULT (Yang et al., 2015)	5110	0.425	49.1	254	0.241	41.9
ComplEx (Trouillon et al., 2016)	5261	**0.444**	50.7	339	0.247	42.8
ConvE (Dettmers et al., 2018)	4187	0.433	51.5	**244**	0.325	50.1
KBGAN (Cai and Wang, 2018)	–	0.213	48.1	–	0.278	45.8
M3GM (Pinter and Eisenstein, 2018)	1864	0.311	53.3	–	–	–
TransE (Bordes et al., 2013)	743⋆	0.245⋆	56.0⋆	347	0.294	46.5
ConvKB (Nguyen et al., 2018)	763⋆	0.253⋆	**56.7**⋆	254⋆	0.418⋆	53.2⋆
Our **CapsE**	**719**	0.415	56.0	303	**0.523**	**59.3**

Table 2: Experimental results on the WN18RR and FB15k-237 test sets. Hits@10 (H@10) is reported in %. Results of DISTMULT, ComplEx and ConvE are taken from Dettmers et al. (2018). Results of TransE on FB15k-237 are taken from Nguyen et al. (2018). Our CapsE Hits@1 scores are 33.7% on WN18RR and 48.9% on FB15k-237. Formulas of MRR and Hits@1 show a strong correlation, so using Hits@1 does not really reveal any additional information for this task. The best score is in bold, while the second best score is in underline. ⋆ denotes *our new results* for TransE and ConvKB, which are better than those published by Nguyen et al. (2018).

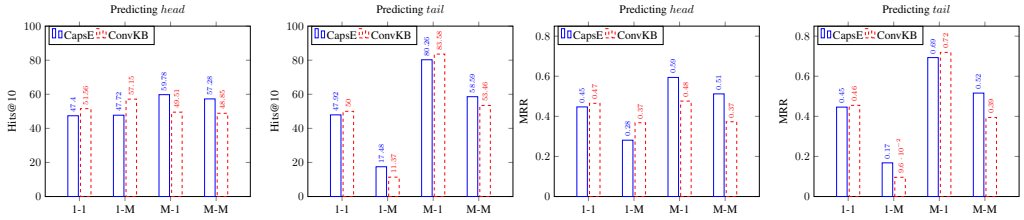

Figure 2: Hits@10 (in %) and MRR on the FB15k-237 test set w.r.t each relation category.

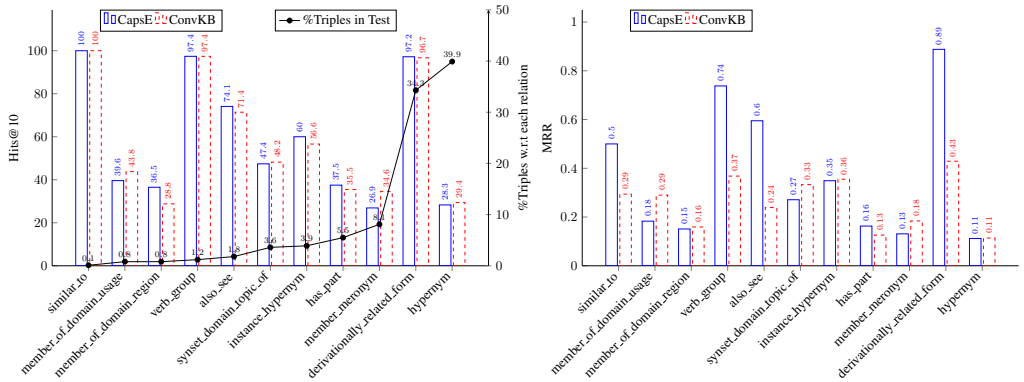

Figure 3: Hits@10 and MRR on the WN18RR test set w.r.t each relation. The right y-axis is the percentage of triples corresponding to relations.

categorized many-to-one (M-1). If $\eta_s \geq 1.5$ and $\eta_o \geq 1.5$, r is categorized many-to-many (M-M). As a result, 17, 26, 81 and 113 relations are labelled 1-1, 1-M, M-1 and M-M, respectively. And 0.9%, 6.3%, 20.5% and 72.3% of the test triples in FB15k-237 contain 1-1, 1-M, M-1 and M-M relations, respectively.

Figure 2 shows the Hits@10 and MRR results for predicting head and tail entities w.r.t each relation category on FB15k-237. CapsE works better than ConvKB in predicting entities on the "side M" of triples (e.g., predicting *head* entities in M-1 and M-M; and predicting *tail* entities in 1-M and M-M), while ConvKB performs better than CapsE in predicting entities on the "side 1" of triples (i.e., predicting *head* entities in 1-1 and 1-M; and predicting *tail* entities in 1-1 and M-1).

Figure 3 shows the Hits@10 and MRR scores w.r.t each relation on WN18RR. $also\_see$, $similar\_to$, $verb\_group$ and $derivationally\_related\_form$ are symmetric relations which can be considered as M-M relations. Our CapsE also performs better than ConvKB on these 4 M-M relations. Thus, results

m	10	20	30	40	50
1	**48.37**	**52.60**	**53.14**	**53.33**	**53.21**
3	47.78	52.34	52.93	52.99	52.86
5	47.03	52.25	45.80	45.99	45.76
7	40.46	45.36	45.79	45.85	45.93

Table 3: Hits@10 on the WN18RR validation set with $\mathsf{N} = 50$ and the initial learning rate at $1e^{-5}$ w.r.t each number of iterations in the routing algorithm m and each 10 training epochs.

shown in Figures 2 and 3 are consistent. These also imply that our CapsE would be a potential candidate for applications which contain many M-M relations such as search personalization.

We see that the length and orientation of each capsule in the first layer can also help to model the important entries in the corresponding dimension, thus CapsE can work well on the "side M" of triples where entities often appear less frequently than others appearing in the "side 1" of triples. Additionally, existing models such as DISTMULT, ComplEx and ConvE can perform well for entities with high frequency, but may not for rare entities with low frequency. These are reasons why our CapsE can be considered as the best one on FB15k-237 and it outperforms most existing models on WN18RR.

Effects of routing iterations: We study how the number of routing iterations affect the performance. Table 3 shows the Hits@10 scores on the WN18RR validation set for a comparison w.r.t each number value of the routing iterations and epochs with the number of filters $\mathsf{N} = 50$ and the Adam initial learning rate at $1e^{-5}$. We see that the best performance for each setup over each 10 epochs is obtained by setting the number m of routing iterations to 1. This indicates the opposite side for knowledge graphs compared to images. In the image classification task, setting the number m of iterations in the routing process higher than 1 helps to capture the relative positions of entities in an image (e.g., eyes, nose and mouth) properly. In contrast, this property from images may be only right for the 1-1 relations, but not for the 1-M, M-1 and M-M relations in the KGs because of the high variant of each relation type (e.g., symmetric relations) among different entities.

4 Search personalization application

Given a *user*, a submitted *query* and the *documents* returned by a search system for that query, our approach is to re-rank the returned documents so that the more relevant documents should be ranked higher. Following Vu et al. (2017), we represent the relationship between the submitted query, the user and the returned document as a *(s, r, o)*-like triple *(query, user, document)*. The triple captures how much interest a user puts on a document given a query. Thus, we can evaluate the effectiveness of our CapsE for the search personalization task.

4.1 Experimental setup

Dataset: We use the SEARCH17 dataset (Vu et al., 2017) of query logs of 106 users collected by a large-scale web search engine. A log entity consists of a user identifier, a query, top-10 ranked documents returned by the search engine and clicked documents along with the user's dwell time. Vu et al. (2017) constructed short-term (session-based) user profiles and used the profiles to personalize the returned results. They then employed the SAT criteria (Fox et al., 2005) to identify whether a returned document is relevant from the query logs as either a clicked document with a dwell time of at least 30 seconds or the last clicked document in a search session (i.e., a SAT click). After that, they assigned a $relevant$ label to a returned document if it is a SAT click and also assigned $irrelevant$ labels to the remaining top-10 documents. The rank position of the $relevant$ labeled documents is used as the ground truth to evaluate the search performance before and after re-ranking.

The dataset was uniformly split into the training, validation and test sets. This split is for the purpose of using historical data in the training set to predict new data in the test set (Vu et al., 2017). The training, validation and test sets consist of 5,658, 1,184 and 1,210 relevant (i.e., valid) triples; and 40,239, 7,882 and 8,540 irrelevant (i.e., invalid) triples, respectively.

Evaluation protocol: Our CapsE is used to re-rank the original list of documents returned by a search engine as follows: (i) We train our model and employ the trained model to calculate the score for each (s, r, o) triple. (ii) We then sort the scores in the descending order to obtain a new ranked list. To evaluate the performance of our proposed model, we use two standard evaluation metrics: mean reciprocal rank (MRR) and Hits@1.[3] For each metric, the higher value indi-

[3]We re-rank the list of top-10 documents returned by the

cates better ranking performance.

We compare CapsE with the following baselines using the same experimental setup: (**1**) SE: The original rank is returned by the search engine. (**2**) CI (Teevan et al., 2011): This baseline uses a personalized navigation method based on previously clicking returned documents. (**3**) SP (Bennett et al., 2012; Vu et al., 2015): A search personalization method makes use of the session-based user profiles. (**4**) Following Vu et al. (2017), we use TransE as a strong baseline model for the search personalization task. Previous work shows that the well-known embedding model TransE, despite its simplicity, obtains very competitive results for the knowledge graph completion (Lin et al., 2015a; Nickel et al., 2016b; Trouillon et al., 2016; Nguyen et al., 2016a, 2018). (**5**) The CNN-based model ConvKB is the most closely related model to our CapsE.

Embedding initialization: We follow Vu et al. (2017) to initialize user profile, query and document embeddings for the baselines TransE and ConvKB, and our CapsE.

We train a LDA topic model (Blei et al., 2003) with 200 topics only on the *relevant* documents (i.e., SAT clicks) extracted from the query logs. We then use the trained LDA model to infer the probability distribution over topics for every returned document. We use the topic proportion vector of each document as its document embedding (i.e. $k = 200$). In particular, the z^{th} element ($z = 1, 2, ..., k$) of the vector embedding for document d is: $\boldsymbol{v}_{d,z} = \mathrm{P}(z \mid d)$ where $\mathrm{P}(z \mid d)$ is the probability of the topic z given the document d.

We also represent each query by a probability distribution vector over topics. Let $\mathcal{D}_q = \{d_1, d_2, ..., d_n\}$ be the set of top n ranked documents returned for a query q (here, $n = 10$). The z^{th} element of the vector embedding for query q is defined as in (Vu et al., 2017): $\boldsymbol{v}_{q,z} = \sum_{i=1}^{n} \lambda_i \mathrm{P}(z \mid d_i)$, where $\lambda_i = \frac{\delta^{i-1}}{\sum_{j=1}^{n} \delta^{j-1}}$ is the exponential decay function of i which is the rank of d_i in D_q. And δ is the decay hyper-parameter ($0 < \delta < 1$). Following Vu et al. (2017), we use $\delta = 0.8$. Note that if we learn query and document embeddings during training, the models will overfit to the data and will not work for new queries and documents. Thus, after the initialization process, we fix (i.e., not updating) query and document embeddings during training for TransE, ConvKB and CapsE.

In addition, as mentioned by Bennett et al. (2012), the more recently clicked document expresses more about the user current search interest. Hence, we make use of the user clicked documents in the training set with the temporal weighting scheme proposed by Vu et al. (2015) to initialize user profile embeddings for the three embedding models.

Hyper-parameter tuning: For our CapsE model, we set batch size to 128, and also the number of neurons within the capsule in the second capsule layer to 10 ($d = 10$). The number of iterations in the routing algorithm is set to 1 ($m = 1$). For the training model, we use the Adam optimizer with the initial learning rate $\in \{5e^{-6}, 1e^{-5}, 5e^{-5}, 1e^{-4}, 5e^{-4}\}$. We also use ReLU as the activation function g. We select the number of filters $\mathsf{N} \in \{50, 100, 200, 400, 500\}$. We run the model up to 200 epochs and perform a grid search to choose optimal hyper-parameters on the validation set. We monitor the MRR score after each training epoch and obtain the highest MRR score on the validation set when using $\mathsf{N} = 400$ and the initial learning rate at $5e^{-5}$.

We employ the TransE and ConvKB implementations provided by Nguyen et al. (2016b) and Nguyen et al. (2018) and then follow their training protocols to tune hyper-parameters for TransE and ConvKB, respectively. We also monitor the MRR score after each training epoch and attain the highest MRR score on the validation set when using margin = 5, l_1-norm and SGD learning rate at $5e^{-3}$ for TransE; and $\mathsf{N} = 500$ and the Adam initial learning rate at $5e^{-4}$ for ConvKB.

4.2 Main results

Table 4 presents the experimental results of the baselines and our model. Embedding models TranE, ConvKB and CapsE produce better ranking performances than traditional learning-to-rank search personalization models CI and SP. This indicates a prospective strategy of expanding the triple embedding models to improve the ranking quality of the search personalization systems. In particular, our MRR and Hits@1 scores are higher than those of TransE (with relative improvements of 14.5% and 22% over TransE, respectively). Specifically, our CapsE achieves the highest performances in both MRR and Hits@1 (our improvements over all five baselines are statistically

search engine, so Hits@10 scores are same for all models.

Method	MRR	H@1
SE [⋆]	0.559	38.5
CI [⋆]	0.597	41.6
SP [⋆]	0.631	45.2
TransE [⋆]	0.645	48.1
TransE (ours)	0.669	50.9
ConvKB	$0.750_{+12.1\%}$	$59.9_{+17.7\%}$
Our CapsE	$\mathbf{0.766}_{+14.5\%}$	$\mathbf{62.1}_{+22.0\%}$

Table 4: Experimental results on the test set. [⋆] denotes the results reported in (Vu et al., 2017). Hits@1 (H@1) is reported in %. In information retrieval, Hits@1 is also referred to as P@1. The subscripts denote the relative improvement over our TransE results.

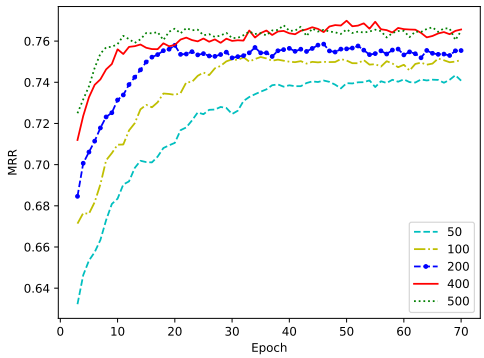

Figure 4: Learning curves on the validation set with the initial learning rate at $5e^{-5}$.

significant with $p < 0.05$ using the *paired t-test*).

To illustrate our training progress, we plot performances of CapsE on the validation set over epochs in Figure 4. We observe that the performance is improved with the increase in the number of filters since capsules can encode more useful properties for a large embedding size.

5 Related work

Other transition-based models extend TransE to additionally use projection vectors or matrices to translate embeddings of s and o into the vector space of r, such as: TransH (Wang et al., 2014), TransR (Lin et al., 2015b), TransD (Ji et al., 2015) and STransE (Nguyen et al., 2016b). Furthermore, DISTMULT (Yang et al., 2015) and ComplEx (Trouillon et al., 2016) use a tri-linear dot product to compute the score for each triple. Moreover, ConvKB (Nguyen et al., 2018) applies convolutional neural network, in which feature maps are concatenated into a single feature vector which is then computed with a weight vector via a dot product to produce the score for the input triple. ConvKB is the most closely related model to our CapsE. See an overview of embedding models for KG completion in (Nguyen, 2017).

For search tasks, unlike classical methods, personalized search systems utilize the historical interactions between the user and the search system, such as submitted queries and clicked documents to tailor returned results to the need of that user (Teevan et al., 2005, 2009). That historical information can be used to build the *user profile*, which is crucial to an effective search personalization system. Widely used approaches consist of two separated steps: (1) building the user profile from the interactions between the user and the search system; and then (2) learning a ranking function to *re-rank* the search results using the user profile (Bennett et al., 2012; White et al., 2013; Harvey et al., 2013; Vu et al., 2015). The general goal is to re-rank the documents returned by the search system in such a way that the more relevant documents are ranked higher. In this case, apart from the user profile, dozens of other features have been proposed as the input of a learning-to-rank algorithm (Bennett et al., 2012; White et al., 2013). Alternatively, Vu et al. (2017) modeled the potential *user*-oriented relationship between the submitted query and the returned document by applying TransE to reward higher scores for more relevant documents (e.g., clicked documents). They achieved better performances than the standard ranker as well as competitive search personalization baselines (Teevan et al., 2011; Bennett et al., 2012; Vu et al., 2015).

6 Conclusion

We propose CapsE—a novel embedding model using the capsule network to model relationship triples for knowledge graph completion and search personalization. Experimental results show that our CapsE outperforms other state-of-the-art models on two benchmark datasets WN18RR and FB15k-237 for the knowledge graph completion. We then show the effectiveness of our CapsE for the search personalization, in which CapsE outperforms the competitive baselines on the dataset SEARCH17 of the web search query logs. In addition, our CapsE is capable to effectively model many-to-many relationships. Our code is available at: https://github.com/daiquocnguyen/CapsE.

Acknowledgements

This research was partially supported by the ARC Discovery Projects DP150100031 and DP160103934. The authors thank Yuval Pinter for assisting us in running his code.

References

Paul N. Bennett, Ryen W. White, Wei Chu, Susan T. Dumais, Peter Bailey, Fedor Borisyuk, and Xiaoyuan Cui. 2012. Modeling the impact of short- and long-term behavior on search personalization. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 185–194.

David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022.

Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *Advances in Neural Information Processing Systems 26*, pages 2787–2795.

Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. 2011. Learning Structured Embeddings of Knowledge Bases. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages 301–306.

Liwei Cai and William Yang Wang. 2018. KBGAN: Adversarial Learning for Knowledge Graph Embeddings. In *Proceedings of The 16th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, page to appear.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2018. Convolutional 2D Knowledge Graph Embeddings. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 1811–1818.

Steve Fox, Kuldeep Karnawat, Mark Mydland, Susan Dumais, and Thomas White. 2005. Evaluating implicit measures to improve web search. *ACM Transactions on Information Systems*, 23(2):147–168.

Morgan Harvey, Fabio Crestani, and Mark J. Carman. 2013. Building user profiles from topic models for personalised search. In *Proceedings of the ACM International Conference on Information and Knowledge Management*, pages 2309–2314.

Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Knowledge Graph Embedding via Dynamic Mapping Matrix. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 687–696.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Yankai Lin, Zhiyuan Liu, Huanbo Luan, Maosong Sun, Siwei Rao, and Song Liu. 2015a. Modeling Relation Paths for Representation Learning of Knowledge Bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 705–714.

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015b. Learning Entity and Relation Embeddings for Knowledge Graph Completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence Learning*, pages 2181–2187.

Dai Quoc Nguyen, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Phung. 2018. A Novel Embedding Model for Knowledge Base Completion Based on Convolutional Neural Network. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 327–333.

Dat Quoc Nguyen. 2017. An overview of embedding models of entities and relationships for knowledge base completion. *arXiv preprint*, arXiv:1703.08098.

Dat Quoc Nguyen, Kairit Sirts, Lizhen Qu, and Mark Johnson. 2016a. Neighborhood Mixture Model for Knowledge Base Completion. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 40–50.

Dat Quoc Nguyen, Kairit Sirts, Lizhen Qu, and Mark Johnson. 2016b. STransE: a novel embedding model of entities and relationships in knowledge bases. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 460–466.

Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. 2016a. A Review of Relational Machine Learning for Knowledge Graphs. *Proceedings of the IEEE*, 104(1):11–33.

Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. 2016b. Holographic Embeddings of Knowledge Graphs. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 1955–1961.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543.

Yuval Pinter and Jacob Eisenstein. 2018. Predicting Semantic Relations using Global Graph Properties. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1741–1751.

Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. 2017. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pages 3859–3869.

Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. 2013. Reasoning With Neural Tensor Networks for Knowledge Base Completion. In *Advances in Neural Information Processing Systems 26*, pages 926–934.

Jaime Teevan, Susan T. Dumais, and Eric Horvitz. 2005. Personalizing search via automated analysis of interests and activities. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 449–456.

Jaime Teevan, Daniel J. Liebling, and Gayathri Ravichandran Geetha. 2011. Understanding and predicting personal navigation. In *Proceedings of the ACM International Conference on Web Search and Data Mining*, pages 85–94.

Jaime Teevan, Meredith Ringel Morris, and Steve Bush. 2009. Discovering and using groups to improve personalized search. In *Proceedings of the ACM International Conference on Web Search and Data Mining*, pages 15–24.

Kristina Toutanova and Danqi Chen. 2015. Observed Versus Latent Features for Knowledge Base and Text Inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pages 57–66.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex Embeddings for Simple Link Prediction. In *Proceedings of the 33nd International Conference on Machine Learning*, pages 2071–2080.

Thanh Vu, Dat Quoc Nguyen, Mark Johnson, Dawei Song, and Alistair Willis. 2017. Search personalization with embeddings. In *Proceedings of the European Conference on Information Retrieval*, pages 598–604.

Thanh Vu, Alistair Willis, Son Ngoc Tran, and Dawei Song. 2015. Temporal latent topic user profiles for search personalisation. In *Proceedings of the European Conference on Information Retrieval*, pages 605–616.

Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. 2017. Knowledge Graph Embedding: A Survey of Approaches and Applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge Graph Embedding by Translating on Hyperplanes. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 1112–1119.

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

Ryen W. White, Wei Chu, Ahmed Hassan, Xiaodong He, Yang Song, and Hongning Wang. 2013. Enhancing personalized search by mining and modeling task behavior. In *Proceedings of the World Wide Web conference*, pages 1411–1420.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *Proceedings of the International Conference on Learning Representations*.

4.2.2 Relation-aware quaternions for knowledge graph embeddings

• **Dai Quoc Nguyen**, Thanh Vu, Tu Dinh Nguyen and Dinh Phung. QuatRE: Relation-Aware Quaternions for Knowledge Graph Embeddings. *arXiv preprint arXiv:2009.12517*.

Contribution. We propose an effective embedding model, named QuatRE (Nguyen et al., 2020e), to learn the quaternion embeddings for entities and relations. QuatRE further utilises two relation-aware rotations for the head and tail embeddings through the Hamilton product, respectively. As a result, QuatRE strengthens the correlations between the head and tail entities. Our QuatRE obtains state-of-the-art performances on well-known benchmark datasets for the knowledge graph completion task; thus, it can act as a new strong baseline for future works.

Furthermore, an extended abstract of the submitted paper – entitled "QuatRE: Relation-Aware Quaternions for Knowledge Graph Embeddings" – has been accepted to the NeurIPS 2020 Workshop on Differential Geometry meets Deep Learning (DiffGeo4DL). The code is available at: `https://github.com/daiquocnguyen/QuatRE`.

QuatRE: Relation-Aware Quaternions for Knowledge Graph Embeddings

Dai Quoc Nguyen[1], **Thanh Vu**[2], **Tu Dinh Nguyen**[3], **Dinh Phung**[1]
[1]Dept of Data Science and AI, Monash University, Australia
[2]The Australian e-Health Research Centre, CSIRO, Australia
[3]VinAI Research, Vietnam
[1]{dai.nguyen,dinh.phung}@monash.edu
[2]thanh.vu@csiro.au; [3]v.tund21@vinai.io

Abstract

We propose an effective embedding model to learn quaternion embeddings for entities and relations in knowledge graphs. Our model aims to enhance correlations between head and tail entities given a relation within the Quaternion space with Hamilton product. The model achieves this goal by further associating each relation with two relation-aware rotations, which are used to rotate quaternion embeddings of the head and tail entities, respectively. Experimental results show that our proposed model produces state-of-the-art performances on well-known benchmark datasets for knowledge graph completion.

1 Introduction

Knowledge graphs (KGs) are constructed to represent relationships between entities in the form of triples *(head, relation, tail)* denoted as *(h, r, t)*. A typical problem in KGs is the lack of many valid triples (West et al., 2014); therefore, research approaches have been proposed to predict whether a new triple missed in KGs is likely valid (Bordes et al., 2011, 2013; Socher et al., 2013). These approaches often utilize embedding models to compute a score for each triple, such that valid triples have higher scores than invalid ones. 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).

Most of the existing models focus on embedding entities and relations within the real-valued vector space (Bordes et al., 2013; Wang et al., 2014; Lin et al., 2015; Yang et al., 2015; Dettmers et al., 2018; Nguyen et al., 2018). Moving beyond the real-valued vector space, ComplEx (Trouillon et al., 2016) and RotatE (Sun et al., 2019) consider the complex vector space, while MuRP (Balažević et al., 2019) and AttH (Chami et al., 2020) leverage the hyperbolic space.

Recently the use of hyper-complex vector space has been considered on the Quaternion space $\mathbb{H}$ consisting of a real and three separate imaginary axes (Zhu et al., 2018; Gaudet and Maida, 2018; Parcollet et al., 2018, 2019; Tay et al., 2019). QuatE (Zhang et al., 2019) – one of the recent state-of-the-art models – is proposed to embed entities and relations in KGs within the Quaternion space. However, QuatE is not completely effective at capturing the correlations between the head and tail entities. For example, given a relation "has positive test'", QuatE does not capture fully the correlations between the attributes (e.g., age, gender, and medical record) of the head entity (e.g., "Donald Trump") and the attributes (e.g., transmission rate and clinical characteristics) of the tail entity (e.g., "COVID-19"). Some early translation-based models such as TransR (Lin et al., 2015) and STransE(Nguyen et al., 2016) can partially address this issue by associating each relation with translation matrices, but growing model parameters significantly.

Addressing these problems, we propose an effective embedding model, named QuatRE, to learn the quaternion embeddings for entities and relations. QuatRE further utilizes two relation-aware rotations for the head and tail embeddings through the Hamilton product, respectively. As a result, QuatRE strengthens the correlations between the head and tail entities. Experimental results demonstrate that our QuatRE obtains state-of-the-art performances on well-known benchmark datasets for the knowledge graph completion task; thus, it can act as a new strong baseline for future works.

2 The approach

2.1 Quaternion background

We provide key notations and operations related to quaternion space required for our model. Additional details can further be found in the supplementary material.

A quaternion $q \in \mathbb{H}$ is a hyper-complex number consisting of a real and three separate imaginary components (Hamilton, 1844) defined as: $q = q_\mathsf{r} + q_\mathsf{i}\mathbf{i} + q_\mathsf{j}\mathbf{j} + q_\mathsf{k}\mathbf{k}$, where $q_\mathsf{r}, q_\mathsf{i}, q_\mathsf{j}, q_\mathsf{k} \in \mathbb{R}$, and $\mathbf{i}, \mathbf{j}, \mathbf{k}$ are imaginary units that $\mathbf{ijk} = \mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$, leads to noncommutative multiplication rules as $\mathbf{ij} = \mathbf{k}, \mathbf{ji} = -\mathbf{k}, \mathbf{jk} = \mathbf{i}, \mathbf{kj} = -\mathbf{i}, \mathbf{ki} = \mathbf{j}$, and $\mathbf{ik} = -\mathbf{j}$. Correspondingly, a n-dimensional quaternion vector $\boldsymbol{q} \in \mathbb{H}^n$ is defined as: $\boldsymbol{q} = \boldsymbol{q}_\mathsf{r} + \boldsymbol{q}_\mathsf{i}\mathbf{i} + \boldsymbol{q}_\mathsf{j}\mathbf{j} + \boldsymbol{q}_\mathsf{k}\mathbf{k}$, where $\boldsymbol{q}_\mathsf{r}, \boldsymbol{q}_\mathsf{i}, \boldsymbol{q}_\mathsf{j}, \boldsymbol{q}_\mathsf{k} \in \mathbb{R}^n$.

Norm. The normalized quaternion vector $\boldsymbol{q}^\triangleleft$ of $\boldsymbol{q} \in \mathbb{H}^n$ is computed as:

$$\boldsymbol{q}^\triangleleft = \frac{\boldsymbol{q}_\mathsf{r} + \boldsymbol{q}_\mathsf{i}\mathbf{i} + \boldsymbol{q}_\mathsf{j}\mathbf{j} + \boldsymbol{q}_\mathsf{k}\mathbf{k}}{\sqrt{\boldsymbol{q}_\mathsf{r}^2 + \boldsymbol{q}_\mathsf{i}^2 + \boldsymbol{q}_\mathsf{j}^2 + \boldsymbol{q}_\mathsf{k}^2}} \tag{1}$$

Hamilton product. The Hamilton product of two vectors $\boldsymbol{q}$ and $\boldsymbol{p} \in \mathbb{H}^n$ is computed as:

$$\begin{aligned} \boldsymbol{q} \otimes \boldsymbol{p} \;=\; & (\boldsymbol{q}_\mathsf{r} \circ \boldsymbol{p}_\mathsf{r} - \boldsymbol{q}_\mathsf{i} \circ \boldsymbol{p}_\mathsf{i} - \boldsymbol{q}_\mathsf{j} \circ \boldsymbol{p}_\mathsf{j} - \boldsymbol{q}_\mathsf{k} \circ \boldsymbol{p}_\mathsf{k}) \\ + \; & (\boldsymbol{q}_\mathsf{i} \circ \boldsymbol{p}_\mathsf{r} + \boldsymbol{q}_\mathsf{r} \circ \boldsymbol{p}_\mathsf{i} - \boldsymbol{q}_\mathsf{k} \circ \boldsymbol{p}_\mathsf{j} + \boldsymbol{q}_\mathsf{j} \circ \boldsymbol{p}_\mathsf{k})\mathbf{i} \\ + \; & (\boldsymbol{q}_\mathsf{j} \circ \boldsymbol{p}_\mathsf{r} + \boldsymbol{q}_\mathsf{k} \circ \boldsymbol{p}_\mathsf{i} + \boldsymbol{q}_\mathsf{r} \circ \boldsymbol{p}_\mathsf{j} - \boldsymbol{q}_\mathsf{i} \circ \boldsymbol{p}_\mathsf{k})\mathbf{j} \\ + \; & (\boldsymbol{q}_\mathsf{k} \circ \boldsymbol{p}_\mathsf{r} - \boldsymbol{q}_\mathsf{j} \circ \boldsymbol{p}_\mathsf{i} + \boldsymbol{q}_\mathsf{i} \circ \boldsymbol{p}_\mathsf{j} + \boldsymbol{q}_\mathsf{r} \circ \boldsymbol{p}_\mathsf{k})\mathbf{k} \end{aligned} \tag{2}$$

where $\circ$ denotes the element-wise product. 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, which is computed as:

$$\boldsymbol{q} \bullet \boldsymbol{p} = \boldsymbol{q}_\mathsf{r}^\mathsf{T}\boldsymbol{p}_\mathsf{r} + \boldsymbol{q}_\mathsf{i}^\mathsf{T}\boldsymbol{p}_\mathsf{i} + \boldsymbol{q}_\mathsf{j}^\mathsf{T}\boldsymbol{p}_\mathsf{j} + \boldsymbol{q}_\mathsf{k}^\mathsf{T}\boldsymbol{p}_\mathsf{k} \tag{3}$$

QuatE: QuatE (Zhang et al., 2019) computes the score of the triple *(h, r, t)* as: $(\boldsymbol{v}_h \otimes \boldsymbol{v}_r^\triangleleft) \bullet \boldsymbol{v}_t$, where $\boldsymbol{v}_h$, $\boldsymbol{v}_r$, and $\boldsymbol{v}_t \in \mathbb{H}^n$. However, only using $\boldsymbol{v}_h$, $\boldsymbol{v}_r$, $\boldsymbol{v}_t$ in QuatE to obtain the triple score is not completely effective at modeling the correlations between the head and tail entities. Our key contribution is to overcome the limitation of QuatE by integrating relation-aware rotations to increase the correlations between the entities.

2.2 The proposed QuatRE

A knowledge graph (KG) $\mathcal{G}$ is a collection of valid factual triples in the form of *(head, relation, tail)* denoted as (h, r, t) such that $h, t \in \mathcal{E}$ and $r \in \mathcal{R}$ where $\mathcal{E}$ is a set of entities and $\mathcal{R}$ is a set of relations. KG embedding models aim to embed entities and relations to a low-dimensional vector space to define a score function f. This function is to give a score for each triple (h, r, t), such that the valid triples obtain higher scores than the invalid triples.

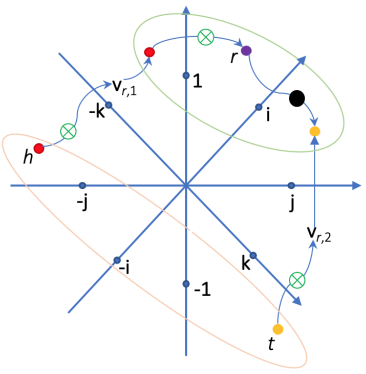

Figure 1: An illustration of our proposed QuatRE. Using relation-aware rotations enhances correlations between head and tail entities.

Given a triple (h, r, t), QuatRE also represents the embeddings of entities and relations within the Quaternion space. QuatRE further associates each relation r with two quaternion vectors $\mathbf{v}_{r,1}$ and $\mathbf{v}_{r,2} \in \mathbb{H}^n$. QuatRE then uses the Hamilton product to rotate $\boldsymbol{v}_h$ and $\boldsymbol{v}_t$ by the normalized vectors $\mathbf{v}_{r,1}^\triangleleft$ and $\mathbf{v}_{r,2}^\triangleleft$ respectively as:

$$\boldsymbol{v}_{h,r,1} = \boldsymbol{v}_h \otimes \mathbf{v}_{r,1}^\triangleleft \tag{4}$$

$$\boldsymbol{v}_{t,r,2} = \boldsymbol{v}_t \otimes \mathbf{v}_{r,2}^\triangleleft \tag{5}$$

After that, QuatRE also utilizes a Hamilton product-based rotation for $\boldsymbol{v}_{h,r,1}$ by the normalized quaternion embedding $\boldsymbol{v}_r^\triangleleft$, then followed by a quaternion-inner product with $\boldsymbol{v}_{t,r,2}$ to produce the triple score. The quaternion components of input vectors are shared during computing the Hamilton product, as shown in Equation 2. Therefore, QuatRE uses two rotations in Equations 4 and 5 for $\boldsymbol{v}_h$ and $\boldsymbol{v}_t$ to increase the correlations between the head h and tail t entities given the relation r, as illustrated in Figure 1.

Formally, we define the QuatRE score function f for the triple (h, r, t) as:

$$\begin{aligned} f(h, r, t) &= (\boldsymbol{v}_{h,r,1} \otimes \boldsymbol{v}_r^\triangleleft) \bullet \boldsymbol{v}_{t,r,2} \\ &= \left((\boldsymbol{v}_h \otimes \mathbf{v}_{r,1}^\triangleleft) \otimes \boldsymbol{v}_r^\triangleleft\right) \bullet \left(\boldsymbol{v}_t \otimes \mathbf{v}_{r,2}^\triangleleft\right) \end{aligned} \tag{6}$$

Proposition. If we fix the real components of both $\mathbf{v}_{r,1}$ and $\mathbf{v}_{r,2}$ to $\mathbf{1}$, and fix the imaginary components of both $\mathbf{v}_{r,1}$ and $\mathbf{v}_{r,2}$ to $\mathbf{0}$, our QuatRE

Method	WN18RR					FB15k-237				
	MR	MRR	H@10	H@3	H@1	MR	MRR	H@10	H@3	H@1
TransE (2013)	3384	0.226	50.1	–	–	357	0.294	46.5	–	–
DistMult (2015)	5110	0.430	49.0	44.0	39.0	254	0.241	41.9	26.3	15.5
ConvE (2018)	5277	0.460	48.0	43.0	39.0	246	0.316	49.1	35.0	23.9
ConvKB (2018)	2741	0.220	50.8	–	–	196	0.302	48.3	–	–
NKGE (2018)	4170	0.450	52.6	46.5	42.1	237	0.330	51.0	36.5	24.1
InteractE (2020)	5202	0.463	52.8	–	43.0	172	0.354	53.5	–	26.3
AutoSF (2020)	–	0.490	56.7	–	**45.1**	–	0.360	55.2	–	26.7
ComplEx (2016)	5261	0.440	51.0	46.0	41.0	339	0.247	42.8	27.5	15.8
RotatE (2019)	3277	0.470	56.5	48.8	42.2	185	0.297	48.0	32.8	20.5
MuRP (2019)	–	0.481	56.6	49.5	44.0	–	0.335	51.8	36.7	24.3
AttH (2020)	–	0.486	57.3	49.9	44.3	–	0.348	54.0	38.4	25.2
RotH (2020)	–	**0.496**	58.6	51.4	44.9	–	0.344	53.5	38.0	24.6
QuatE (2019)	2314	0.488	58.2	50.8	43.8	**87**	0.348	55.0	38.2	24.8
QuatRE	**1986**	0.493	**59.2**	**51.9**	43.9	88	**0.367**	**56.3**	**40.4**	**26.9**
GC-OTE (2020)	–	0.491	58.3	51.1	44.2	–	0.361	55.0	39.6	26.7
ReInceptionE (2020)	1894	0.483	58.2	–	–	173	0.349	52.8	–	–
RotatE$_{Adv}$ (2019)	3340	0.476	57.1	49.2	42.8	177	0.338	53.3	37.5	24.1
R-GCN+ (2018)	–	–	–	–	–	–	0.249	41.7	26.4	15.1

Table 1: Experimental results on WN18RR and FB15k-237. Hits@k (H@k) is reported in %. The best scores are in bold, while the second best scores are in underline. The results of TransE are taken from (Nguyen et al., 2018). The results of DistMult and ComplEx are taken from (Dettmers et al., 2018). We note that GC-OTE and RotatE$_{Adv}$ apply a self-adversarial negative sampling, which is different from the common negative sampling used in the previous baselines, QuatE and our QuatRE. Furthermore, GC-OTE, ReInceptionE, and R-GCN+ integrate information about relation paths. For a fair comparison, we do not compare our QuatRE with these models.

is simplified to QuatE. Hence QuatRE is viewed as an extension of QuatE. Furthermore, given the same embedding dimension, QuatE and our QuatRE have comparable numbers of parameters. Besides, an advantage of QuatRE is to change the common use of translation matrices in translation-based models such as TransR (Lin et al., 2015) and STransE(Nguyen et al., 2016), hence reducing computation significantly.

Learning process. We employ the Adagrad optimizer (Duchi et al., 2011) to train our proposed QuatRE by minimizing the following loss function (Trouillon et al., 2016) with the regularization on model parameters $\boldsymbol{\theta}$ as:

$$\mathcal{L} = \sum_{(h,r,t)\in\{\mathcal{G}\cup\mathcal{G}'\}} \log\left(1+\exp\left(-l_{(h,r,t)}\cdot f(h,r,t)\right)\right) + \lambda\|\boldsymbol{\theta}\|_2^2 \quad (7)$$

$$\text{in which, } l_{(h,r,t)} = \begin{cases} 1 & \text{for } (h,r,t)\in\mathcal{G} \\ -1 & \text{for } (h,r,t)\in\mathcal{G}' \end{cases}$$

where we use l_2-norm with the regularization rate λ; and $\mathcal{G}$ and $\mathcal{G}'$ are collections of valid and invalid triples, respectively. $\mathcal{G}'$ is generated by corrupting valid triples in $\mathcal{G}$.

3 Experimental results

We evaluate our QuatRE on two benchmark datasets WN18RR (Dettmers et al., 2018) and FB15k-237 (Toutanova and Chen, 2015) for the knowledge graph completion task (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). We present the evaluation protocol and the training protocol in the supplementary material.

Main results. We report the experimental results on the datasets in Table 1. In general, QuatRE outperforms up-to-date baselines for all metrics except the second-best MRR and the Hits@1 on WN18RR and the second-best MR on FB15k-237. Especially when comparing with QuatE, on WN18RR, QuatRE gains significant improvements of $2314-1986=328$ in MR (which is about 14% relative improvement), and 1.0% and 1.1% absolute improvements in Hits@10 and Hits@3 respectively. Besides, on FB15k-237, QuatRE achieves improvements of $0.367-0.348=0.019$ in MRR (which is 5.5% relative improvement) and obtains absolute gains of 1.3%, 2.2%, and 2.1% in Hits@10, Hits@3, and Hits@1 respectively.

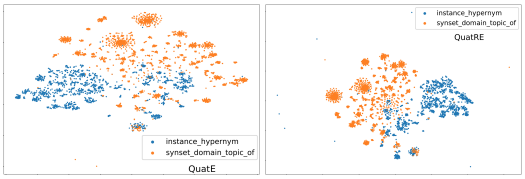

Figure 2: Visualization of the learned entity embeddings on WN18RR.

Correlation analysis. To qualitatively demonstrate the correlations between the entities, we use t-SNE (Maaten and Hinton, 2008) to visualize the learned quaternion embeddings of the entities on WN18RR for QuatE and QuatRE. We select all entities associated with two relations consisting of "instance_hypernym" and "synset_domain_topic_of". We then vectorize each quaternion embedding using a vector concatenation across the four components; hence, we obtain a real-valued vector representation for applying t-SNE. The visualization in Figure 2 shows that the entity distribution in our QuatRE is denser than that in QuatE; hence this implies that QuatRE strengthens the correlations between the entities.

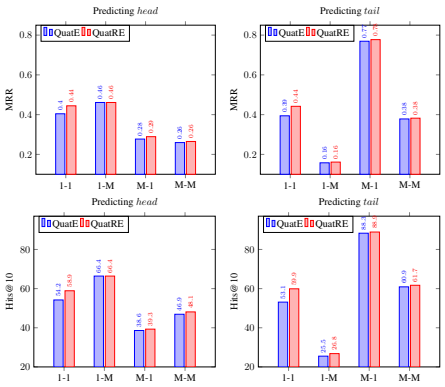

Figure 3: MRR and Hits@10 on FB15k-237 for QuatE and our QuatRE with respect to each relation category.

Relation analysis. Following Bordes et al. (2013), for each relation r, we calculate the averaged number η_h of head entities per tail entity and the averaged number η_t of tail entities per head entity. If η_h <1.5 and η_t <1.5, r is categorized one-to-one (1-1). If η_h <1.5 and η_t ≥1.5, r is categorized one-to-many (1-M). If η_h ≥1.5 and η_t <1.5, r is categorized many-to-one (M-1). If η_h ≥1.5 and η_t ≥1.5, r is categorized many-to-many (M-M). Figure 3 shows the MRR and H@10 scores for predicting the head entities and then the tail entities

Relation	QuatE	QuatRE
hypernym	0.173	**0.190**
derivationally_related_form	**0.953**	0.943
instance_hypernym	0.364	**0.380**
also_see	0.629	**0.633**
member_meronym	0.232	**0.237**
synset_domain_topic_of	0.468	**0.495**
has_part	**0.233**	0.226
member_of_domain_usage	0.441	**0.470**
member_of_domain_region	0.193	**0.364**
verb_group	**0.924**	0.867
similar_to	**1.000**	**1.000**

Table 2: MRR score on the WN18RR test set with respect to each relation.

with respect to each relation category on FB15k-237, wherein our QuatRE outperforms QuatE on these relation categories. Furthermore, we report the MRR scores for each relation on WN18RR in Table 2, which shows the effectiveness of QuatRE in modeling different types of relations.

Model	WN18RR		FB15k-237	
	MRR	H@10	MRR	H@10
$\left(\left(\boldsymbol{v}_h \otimes \mathbf{v}^{\mathrm{q}}_{r,1}\right) \otimes \boldsymbol{v}^{\mathrm{q}}_r\right) \bullet \left(\boldsymbol{v}_t \otimes \mathbf{v}^{\mathrm{q}}_{r,2}\right)$	0.493	59.2	0.367	56.3
(i) $\left(\left(\boldsymbol{v}_h \otimes \mathbf{v}^{\mathrm{q}}_{r,1}\right) \otimes \boldsymbol{v}^{\mathrm{q}}_r\right) \bullet \boldsymbol{v}_t$	0.491	58.9	0.364	56.0
(ii) $\left(\boldsymbol{v}_h \otimes \boldsymbol{v}^{\mathrm{q}}_r\right) \bullet \left(\boldsymbol{v}_t \otimes \mathbf{v}^{\mathrm{q}}_{r,2}\right)$	0.491	58.8	0.364	56.1
QuatE: $\left(\boldsymbol{v}_h \otimes \boldsymbol{v}^{\mathrm{q}}_r\right) \bullet \boldsymbol{v}_t$	0.488	58.2	0.348	55.0

Table 3: Ablation results for two different variants of our QuatRE. (i) With only using $\mathbf{v}_{r,1}$. (ii) With only using $\mathbf{v}_{r,2}$.

Ablation analysis. We report our ablation results for two variants of our QuatRE in Table 3. In particular, the results degrade on both datasets when only using either of $\mathbf{v}_{r,1}$ and $\mathbf{v}_{r,2}$. However, these two variants of QuatRE still outperforms QuatE, hence clearly showing the advantage of further using the relation-aware rotations in our QuatRE to enhance the correlations in knowledge graphs.

4 Conclusion

In this paper, we propose QuatRE – an advantageous knowledge graph embedding model – to learn the embeddings of entities and relations within the Quaternion space with the Hamilton product. QuatRE further utilizes two relation-aware rotations to strengthen the correlations between the head and tail entities. Experimental results demonstrate that QuatRE outperforms up-to-date embedding models and produces state-of-the-art performances on well-known benchmark datasets for the knowledge graph completion task.

References

Ivana Balažević, Carl Allen, and Timothy Hospedales. 2019. Multi-relational poincaré graph embeddings. In *Advances in Neural Information Processing Systems*, pages 4465–4475.

Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *Advances in Neural Information Processing Systems 26*, pages 2787–2795.

Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. 2011. Learning Structured Embeddings of Knowledge Bases. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages 301–306.

Ines Chami, Adva Wolf, Da-Cheng Juan, Frederic Sala, Sujith Ravi, and Christopher Ré. 2020. Low-dimensional hyperbolic knowledge graph embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6901–6914.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2018. Convolutional 2D Knowledge Graph Embeddings. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 1811–1818.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159.

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

William Rowan Hamilton. 1844. 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.

Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Knowledge Graph Embedding via Dynamic Mapping Matrix. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 687–696.

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning Entity and Relation Embeddings for Knowledge Graph Completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence Learning*, pages 2181–2187.

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

Dai Quoc Nguyen, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Phung. 2018. A Novel Embedding Model for Knowledge Base Completion Based on Convolutional Neural Network. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 327–333.

Dat Quoc Nguyen. 2017. An Overview of Embedding Models of Entities and Relationships for Knowledge Base Completion. *arXiv preprint*, arXiv:1703.08098.

Dat Quoc Nguyen, Kairit Sirts, Lizhen Qu, and Mark Johnson. 2016. STransE: a novel embedding model of entities and relationships in knowledge bases. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 460–466.

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

Titouan Parcollet, Ying Zhang, Mohamed Morchid, Chiheb Trabelsi, Georges Linarès, Renato De Mori, and Yoshua Bengio. 2018. Quaternion convolutional neural networks for end-to-end automatic speech recognition. In *The 19th Annual Conference of the International Speech Communication Association (Interspeech)*, pages 22–26.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035.

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

Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. 2013. Reasoning With Neural Tensor Networks for Knowledge Base Completion. In *Advances in Neural Information Processing Systems 26*, pages 926–934.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. 2019. Rotate: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations*.

Yun Tang, Jing Huang, Guangtao Wang, Xiaodong He, and Bowen Zhou. 2020. Orthogonal relation transforms with graph context modeling for knowledge graph embedding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2713–2722. Association for Computational Linguistics.

Yi Tay, Aston Zhang, Anh Tuan Luu, Jinfeng Rao, Shuai Zhang, Shuohang Wang, Jie Fu, and Siu Cheung Hui. 2019. 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.

Kristina Toutanova and Danqi Chen. 2015. Observed Versus Latent Features for Knowledge Base and Text Inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pages 57–66.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex Embeddings for Simple Link Prediction. In *Proceedings of the 33nd International Conference on Machine Learning*, pages 2071–2080.

Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, Nilesh Agrawal, and Partha Talukdar. 2020. Interacte: Improving convolution-based knowledge graph embeddings by increasing feature interactions. In *International Conference on Learning Representations*.

Kai Wang, Yu Liu, Xiujuan Xu, and Dan Lin. 2018. Knowledge graph embedding with entity neighbors and deep memory network. *arXiv preprint arXiv:1808.03752*.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge Graph Embedding by Translating on Hyperplanes. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 1112–1119.

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

Zhiwen Xie, Guangyou Zhou, Jin Liu, and Jimmy Xiangji Huang. 2020. ReInceptionE: Relation-aware inception network with joint local-global structural information for knowledge graph embedding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5929–5939.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *Proceedings of the International Conference on Learning Representations*.

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

Y. Zhang, Q. Yao, W. Dai, and L. Chen. 2020. Autosf: Searching scoring functions for knowledge graph embedding. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 433–444.

Xuanyu Zhu, Yi Xu, Hongteng Xu, and Changjian Chen. 2018. Quaternion convolutional neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 631–647.

A Related work

Existing embedding models (Bordes et al., 2013; Wang et al., 2014) have been proposed to learn the vector representations of entities and relations for the knowledge graph completion task, where the goal is to score valid triples higher than invalid triples. As an example, Table 4 illustrates the score functions $f(h, r, t)$ in previous state-of-the-art models as well as our proposed model.

Early translation-based approaches exploit a translational characteristic so that the embedding of tail entity t should be close to the embedding of head entity h plus the embedding of relation t. For example, TransE (Bordes et al., 2013) defines a score function: $f(h, r, t) = -\|\boldsymbol{v}_h + \boldsymbol{v}_r - \boldsymbol{v}_t\|_p$, where $\boldsymbol{v}_h$, $\boldsymbol{v}_r$, and $\boldsymbol{v}_t \in \mathbb{R}^n$ are vector embeddings of h, r and t respectively; and $\|\boldsymbol{v}\|_p$ denotes the p-norm of vector $\boldsymbol{v}$. As a result, TransE is suitable for 1-to-1 relationships, but not well-adapted for Many-to-1, 1-to-Many, and Many-to-Many relationships. To this end, some translation-based methods have been proposed to deal with this issue such as TransH (Wang et al., 2014), TransR (Lin et al., 2015), TransD (Ji et al., 2015), and STransE (Nguyen et al., 2016). Notably, DistMult (Yang et al., 2015) employs a multiple-linear dot product to score the triples as: $f(h, r, t) = \sum_i^n \boldsymbol{v}_{h_i} \boldsymbol{v}_{r_i} \boldsymbol{v}_{t_i}$.

One of the recent trends is to apply deep neural networks to measure the triples (Dettmers et al., 2018; Schlichtkrull et al., 2018; Nguyen et al., 2018; Vashishth et al., 2020). For example, ConvE (Dettmers et al., 2018) uses a convolution layer on a 2D input matrix of reshaping the embeddings of both the head entity and relation to produce feature maps that are then vectorized and computed with the embedding of the tail entity to return the score. While most of the existing models have worked in the real-valued vector space, several works have

Model	The score function $f(h, r, t)$
TransE	$-\|\boldsymbol{v}_h + \boldsymbol{v}_r - \boldsymbol{v}_t\|_p$ where $\boldsymbol{v}_h$, $\boldsymbol{v}_r$, and $\boldsymbol{v}_t \in \mathbb{R}^n$; $\|\boldsymbol{v}\|_p$ denotes the p-norm of vector $\boldsymbol{v}$
ConvE	$\boldsymbol{v}_t^{\mathsf{T}} g\left(\mathbf{W}\mathsf{vec}\left(g\left(\mathsf{concat}\left(\widehat{\boldsymbol{v}}_h, \widehat{\boldsymbol{v}}_r\right) * \boldsymbol{\Omega}\right)\right)\right)$ where $*$ denotes a convolution operator $\boldsymbol{\Omega}$ denotes a set of filters; concat denotes a concatenation operator g denotes a non-linear function; $\widehat{\boldsymbol{v}}$ denotes a 2D reshaping of $\boldsymbol{v}$
ConvKB	$\mathbf{w}^{\mathsf{T}}\mathsf{concat}\left(g\left(\left[\boldsymbol{v}_h, \boldsymbol{v}_r, \boldsymbol{v}_t\right] * \boldsymbol{\Omega}\right)\right)$
DistMult	$\langle \boldsymbol{v}_h, \boldsymbol{v}_r, \boldsymbol{v}_t \rangle = \sum_i^n \boldsymbol{v}_{h_i} \boldsymbol{v}_{r_i} \boldsymbol{v}_{t_i}$ where $\langle\rangle$ denotes a multiple-linear dot product
ComplEx	$\mathsf{Re}\left(\langle \boldsymbol{v}_h, \boldsymbol{v}_r, \boldsymbol{v}_t^* \rangle\right)$ where $\mathsf{Re}(c)$ denotes the real part of the complex c $\boldsymbol{v}_h$, $\boldsymbol{v}_r$, and $\boldsymbol{v}_t \in \mathbb{C}^n$; $\boldsymbol{v}^*$ denotes the conjugate of the complex vector $\boldsymbol{v}$
RotatE	$-\|\boldsymbol{v}_h \circ \boldsymbol{v}_r - \boldsymbol{v}_t\|_p$ where $\boldsymbol{v}_h$, $\boldsymbol{v}_r$, and $\boldsymbol{v}_t \in \mathbb{C}^n$; and $\circ$ denotes the element-wise product
QuatE	$\left(\boldsymbol{v}_h \otimes \boldsymbol{v}_r^{\triangleleft}\right) \bullet \boldsymbol{v}_t$ where $\boldsymbol{v}_h$, $\boldsymbol{v}_r$, and $\boldsymbol{v}_t \in \mathbb{H}^n$; $\bullet$ denotes a quaternion-inner product $\otimes$ denotes the Hamilton product; the superscript $^{\triangleleft}$ denotes the normalized embedding
Our **QuatRE**	$\left(\left(\boldsymbol{v}_h \otimes \mathbf{v}_{r,1}^{\triangleleft}\right) \otimes \boldsymbol{v}_r^{\triangleleft}\right) \bullet \left(\boldsymbol{v}_t \otimes \mathbf{v}_{r,2}^{\triangleleft}\right)$ where $\boldsymbol{v}_h$, $\boldsymbol{v}_r$, $\boldsymbol{v}_t$, $\mathbf{v}_{r,1}$, and $\mathbf{v}_{r,2} \in \mathbb{H}^n$

Table 4: The score functions in previous models. The table is adapted from (Nguyen, 2017).

moved beyond the real-valued vector space to the complex vector space such as ComplEx (Trouillon et al., 2016) and RotatE (Sun et al., 2019). ComplEx extends DistMult to use the multiple-linear dot product on the complex vector embeddings of entities and relations. Besides, RotatE considers a rotation-based translation within the complex vector space.

Recently the use of hyper-complex vector space has considered on the Quaternion space consisting of a real and three separate imaginary axes. It provides highly expressive computations through the Hamilton product compared to the real-valued and complex vector spaces. Zhu et al. (2018) and Gaudet and Maida (2018) embed the greyscale and each of RGB channels of the image to the real and three separate imaginary axes of the Quaternion space and achieve better accuracies compared real-valued convolutional neural networks with same structures for image classification tasks. The Quaternion space has also been successfully applied to speech recognition (Parcollet et al., 2018, 2019), and machine translation (Tay et al., 2019). Regarding knowledge graph embeddings, Zhang et al. (2019) has recently proposed QuatE, which aims to learn entity and relation embeddings within the Quaternion space with the Hamilton product. QuatE, however, has a limitation in capturing the correlations between the head and tail entities. Our key contribution is to overcome this limitation by integrating relation-aware rotations to increase the correlations between the entities as illustrated in Figure 1.

B Quaternion background

For completeness, we briefly provide a background in quaternion, which has also similarly described in recent works (Zhu et al., 2018; Parcollet et al., 2019; Zhang et al., 2019; Tay et al., 2019). A quaternion $q \in \mathbb{H}$ is a hyper-complex number consisting of a real and three separate imaginary components (Hamilton, 1844) defined as:

$$q = q_{\mathsf{r}} + q_{\mathsf{i}}\mathbf{i} + q_{\mathsf{j}}\mathbf{j} + q_{\mathsf{k}}\mathbf{k} \tag{8}$$

where $q_{\mathsf{r}}, q_{\mathsf{i}}, q_{\mathsf{j}}, q_{\mathsf{k}} \in \mathbb{R}$, and $\mathbf{i}, \mathbf{j}, \mathbf{k}$ are imaginary units that $\mathbf{ijk} = \mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$, leads to non-commutative multiplication rules as $\mathbf{ij} = \mathbf{k}, \mathbf{ji} = -\mathbf{k}, \mathbf{jk} = \mathbf{i}, \mathbf{kj} = -\mathbf{i}, \mathbf{ki} = \mathbf{j}$, and $\mathbf{ik} = -\mathbf{j}$. Correspondingly, a n-dimensional quaternion vector $\boldsymbol{q} \in \mathbb{H}^n$ is defined as:

$$\boldsymbol{q} = \boldsymbol{q}_{\mathsf{r}} + \boldsymbol{q}_{\mathsf{i}}\mathbf{i} + \boldsymbol{q}_{\mathsf{j}}\mathbf{j} + \boldsymbol{q}_{\mathsf{k}}\mathbf{k} \tag{9}$$

where $\boldsymbol{q}_{\mathsf{r}}, \boldsymbol{q}_{\mathsf{i}}, \boldsymbol{q}_{\mathsf{j}}, \boldsymbol{q}_{\mathsf{k}} \in \mathbb{R}^n$. The operations for the Quaternion algebra are defined as follows:

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

$$q^* = q_{\mathsf{r}} - q_{\mathsf{i}}\mathbf{i} - q_{\mathsf{j}}\mathbf{j} - q_{\mathsf{k}}\mathbf{k} \tag{10}$$

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

$$q+p = (q_{\mathsf{r}}+p_{\mathsf{r}})+(q_{\mathsf{i}}+p_{\mathsf{i}})\mathbf{i}+(q_{\mathsf{j}}+p_{\mathsf{j}})\mathbf{j}+(q_{\mathsf{k}}+p_{\mathsf{k}})\mathbf{k} \tag{11}$$

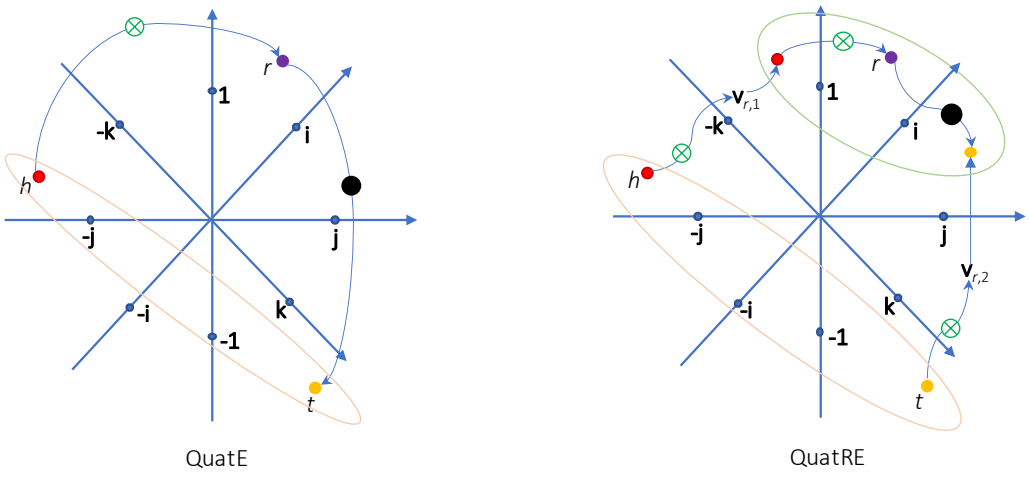

Figure 4: An illustration of QuatE versus our proposed QuatRE.

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

$$\lambda q = \lambda q_{\mathsf{r}} + \lambda q_{\mathsf{i}}\mathbf{i} + \lambda q_{\mathsf{j}}\mathbf{j} + \lambda q_{\mathsf{k}}\mathbf{k} \tag{12}$$

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

$$\|q\| = \sqrt{q_{\mathsf{r}}^2 + q_{\mathsf{i}}^2 + q_{\mathsf{j}}^2 + q_{\mathsf{k}}^2} \tag{13}$$

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

$$q^\triangleleft = \frac{q}{\|q\|} \tag{14}$$

And the normalized quaternion vector $\boldsymbol{q}^\triangleleft$ of $\boldsymbol{q} \in \mathbb{H}^n$ is computed as:

$$\boldsymbol{q}^\triangleleft = \frac{\boldsymbol{q}_{\mathsf{r}} + \boldsymbol{q}_{\mathsf{i}}\mathbf{i} + \boldsymbol{q}_{\mathsf{j}}\mathbf{j} + \boldsymbol{q}_{\mathsf{k}}\mathbf{k}}{\sqrt{\boldsymbol{q}_{\mathsf{r}}^2 + \boldsymbol{q}_{\mathsf{i}}^2 + \boldsymbol{q}_{\mathsf{j}}^2 + \boldsymbol{q}_{\mathsf{k}}^2}} \tag{15}$$

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

$$\begin{aligned} q \otimes p \;=\; & (q_{\mathsf{r}}p_{\mathsf{r}} - q_{\mathsf{i}}p_{\mathsf{i}} - q_{\mathsf{j}}p_{\mathsf{j}} - q_{\mathsf{k}}p_{\mathsf{k}}) \\ + \; & (q_{\mathsf{i}}p_{\mathsf{r}} + q_{\mathsf{r}}p_{\mathsf{i}} - q_{\mathsf{k}}p_{\mathsf{j}} + q_{\mathsf{j}}p_{\mathsf{k}})\mathbf{i} \\ + \; & (q_{\mathsf{j}}p_{\mathsf{r}} + q_{\mathsf{k}}p_{\mathsf{i}} + q_{\mathsf{r}}p_{\mathsf{j}} - q_{\mathsf{i}}p_{\mathsf{k}})\mathbf{j} \\ + \; & (q_{\mathsf{k}}p_{\mathsf{r}} - q_{\mathsf{j}}p_{\mathsf{i}} + q_{\mathsf{i}}p_{\mathsf{j}} + q_{\mathsf{r}}p_{\mathsf{k}})\mathbf{k} \end{aligned} \tag{16}$$

The Hamilton product of two quaternion vectors $\boldsymbol{q}$ and $\boldsymbol{p} \in \mathbb{H}^n$ is computed as:

$$\begin{aligned} \boldsymbol{q} \otimes \boldsymbol{p} \;=\; & (\boldsymbol{q}_{\mathsf{r}} \circ \boldsymbol{p}_{\mathsf{r}} - \boldsymbol{q}_{\mathsf{i}} \circ \boldsymbol{p}_{\mathsf{i}} - \boldsymbol{q}_{\mathsf{j}} \circ \boldsymbol{p}_{\mathsf{j}} - \boldsymbol{q}_{\mathsf{k}} \circ \boldsymbol{p}_{\mathsf{k}}) \\ + \; & (\boldsymbol{q}_{\mathsf{i}} \circ \boldsymbol{p}_{\mathsf{r}} + \boldsymbol{q}_{\mathsf{r}} \circ \boldsymbol{p}_{\mathsf{i}} - \boldsymbol{q}_{\mathsf{k}} \circ \boldsymbol{p}_{\mathsf{j}} + \boldsymbol{q}_{\mathsf{j}} \circ \boldsymbol{p}_{\mathsf{k}})\mathbf{i} \\ + \; & (\boldsymbol{q}_{\mathsf{j}} \circ \boldsymbol{p}_{\mathsf{r}} + \boldsymbol{q}_{\mathsf{k}} \circ \boldsymbol{p}_{\mathsf{i}} + \boldsymbol{q}_{\mathsf{r}} \circ \boldsymbol{p}_{\mathsf{j}} - \boldsymbol{q}_{\mathsf{i}} \circ \boldsymbol{p}_{\mathsf{k}})\mathbf{j} \\ + \; & (\boldsymbol{q}_{\mathsf{k}} \circ \boldsymbol{p}_{\mathsf{r}} - \boldsymbol{q}_{\mathsf{j}} \circ \boldsymbol{p}_{\mathsf{i}} + \boldsymbol{q}_{\mathsf{i}} \circ \boldsymbol{p}_{\mathsf{j}} + \boldsymbol{q}_{\mathsf{r}} \circ \boldsymbol{p}_{\mathsf{k}})\mathbf{k} \end{aligned} \tag{17}$$

where $\circ$ denotes the element-wise product. 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, which is computed as:

$$\boldsymbol{q} \bullet \boldsymbol{p} = \boldsymbol{q}_{\mathsf{r}}^\mathsf{T}\boldsymbol{p}_{\mathsf{r}} + \boldsymbol{q}_{\mathsf{i}}^\mathsf{T}\boldsymbol{p}_{\mathsf{i}} + \boldsymbol{q}_{\mathsf{j}}^\mathsf{T}\boldsymbol{p}_{\mathsf{j}} + \boldsymbol{q}_{\mathsf{k}}^\mathsf{T}\boldsymbol{p}_{\mathsf{k}} \tag{18}$$

C Experimental setup

In the knowledge graph completion task (Bordes et al., 2013), the goal is to predict a missing entity given a relation with another entity, for example, inferring a head entity h given (r, t) or inferring a tail entity t given (h, r). The results are calculated by ranking the scores produced by the score function f on triples in the test set.

C.1 Datasets

We evaluate our proposed QuatRE on benchmark datasets: WN18RR (Dettmers et al., 2018) and FB15k-237 (Toutanova and Chen, 2015), which are derived to eliminate the reversible relation problem to create more realistic and challenging prediction tasks.

C.2 Evaluation protocol

Following Bordes et al. (2013), for each valid test triple (h, r, t), we replace either h or t by each of other entities to create a set of corrupted triples. We use the "Filtered" setting protocol (Bordes et al., 2013), i.e., not including any corrupted triples that appear in the KG. We rank the valid test triple

and corrupted triples in descending order of their scores. We employ evaluation metrics: mean rank (MR), mean reciprocal rank (MRR), and Hits@k (the proportion of the valid triples ranking in top k predictions). The final scores on the test set are reported for the model which obtains the highest Hits@10 on the validation set. Lower MR, higher MRR, and higher Hits@k indicate better performance.

C.3 Training protocol

We implement our QuatRE based on Pytorch (Paszke et al., 2019) and test on a single GPU. We set 100 batches for all datasets. We then vary the learning rate α in {0.02, 0.05, 0.1}, the number s of negative triples sampled per training triple in {1, 5, 10}, the embedding dimension n in {128, 256, 384}, and the regularization rate λ in $\{0.05, 0.1, 0.2, 0.5\}$. We train our QuatRE up to 8,000 epochs on WN18RR and 2,000 epochs on FB15k-237. We monitor the Hits@10 score after each 400 epochs on WN18RR and each 200 epochs on FB15k-237. We select the hyper-parameters using grid search and early stopping on the validation set with Hits@10. We present the statistics of the datasets in Table 5 and the optimal hyper-parameters on the validation set for each dataset in Table 6.

Dataset	$\mid \mathcal{E} \mid$	$\mid \mathcal{R} \mid$	#Triples in train/valid/test		
WN18RR	40,943	11	86,835	3,034	3,134
FB15k-237	14,541	237	272,115	17,535	20,466

Table 5: Statistics of the experimental datasets.

Dataset	α	n	λ	s
WN18RR	0.1	256	0.5	5
FB15k-237	0.1	384	0.5	10

Table 6: The optimal hyper-parameters on the validation sets.

4.2.3 ACL 2020 - Memory network for triple classification and search personalisation

• **Dai Quoc Nguyen**, Tu Dinh Nguyen and Dinh Phung. **2020**. A Relational Memory-based Embedding Model for Triple Classification and Search Personalisation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL 2020)*, pages 3429–3435.

Contribution. We leverage a transformer-based memory network (Santoro et al., 2018) to propose R-MeN (Nguyen et al., 2020c) to infer a valid fact of new triples. R-MeN transforms each triple along with adding positional embeddings into a sequence of three input vectors. R-MeN then uses a transformer self-attention mechanism (Vaswani et al., 2017) to guide the memory to interact with each input vector to produce an encoded vector. As a result, R-MeN feeds these three encoded vectors to a convolutional neural network (CNN)-based decoder to return the triple score. R-MeN obtains better performance than up-to-date embedding models for the tasks of triple classification and search personalisation. The code is available at: `https://github.com/daiquocnguyen/R-MeN`.

A Relational Memory-based Embedding Model for Triple Classification and Search Personalization

Dai Quoc Nguyen[1], **Tu Dinh Nguyen**[2], **Dinh Phung**[1]
[1]Monash University, Australia
[2]Trusting Social
[1]{dai.nguyen,dinh.phung}@monash.edu
[2]tu@trustingsocial.com

Abstract

Knowledge graph embedding methods often suffer from a limitation of memorizing valid triples to predict new ones for triple classification and search personalization problems. To this end, we introduce a novel embedding model, named R-MeN, that explores a relational memory network to encode potential dependencies in relationship triples. R-MeN considers each triple as a sequence of 3 input vectors that recurrently interact with a memory using a transformer self-attention mechanism. Thus R-MeN encodes new information from interactions between the memory and each input vector to return a corresponding vector. Consequently, R-MeN feeds these 3 returned vectors to a convolutional neural network-based decoder to produce a scalar score for the triple. Experimental results show that our proposed R-MeN obtains state-of-the-art results on SEARCH17 for the search personalization task, and on WN11 and FB13 for the triple classification task.

1 Introduction

Knowledge graphs (KGs) – representing the genuine relationships among entities in the form of triples *(subject, relation, object)* denoted as *(s, r, o)* – are often insufficient for knowledge presentation due to the lack of many valid triples (West et al., 2014). Therefore, research work has been focusing on inferring whether a new triple missed in KGs is likely valid or not (Bordes et al., 2011, 2013; Socher et al., 2013). As summarized in (Nickel et al., 2016; Nguyen, 2017), KG embedding models aim to compute a score for each triple, such that valid triples have higher scores than invalid ones.

Early embedding models such as TransE (Bordes et al., 2013), TransH (Wang et al., 2014), TransR (Lin et al., 2015), TransD (Ji et al., 2015), DISTMULT (Yang et al., 2015) and ComplEx (Trouillon et al., 2016) often employ simple linear operators such as addition, subtraction and multiplication. Recent embedding models such as ConvE (Dettmers et al., 2018) and CapsE (Nguyen et al., 2019b) successfully apply deep neural networks to score the triples.

Existing embedding models are showing promising performances mainly for knowledge graph completion, where the goal is to infer a missing entity given a relation and another entity. But in real applications, less mentioned, such as triple classification (Socher et al., 2013) that aims to predict whether a given triple is valid, and search personalization (Vu et al., 2017) that aims to re-rank the relevant documents returned by a user-oriented search system given a query, these models do not effectively capture potential dependencies among entities and relations from existing triples to predict new triples.

To this end, we leverage the relational memory network (Santoro et al., 2018) to propose R-MeN to infer a valid fact of new triples. In particular, R-MeN transforms each triple along with adding positional embeddings into a sequence of 3 input vectors. R-MeN then uses a transformer self-attention mechanism (Vaswani et al., 2017) to guide the memory to interact with each input vector to produce an encoded vector. As a result, R-MeN feeds these 3 encoded vectors to a convolutional neural network (CNN)-based decoder to return a score for the triple. In summary, our main contributions are as follows:

- We present R-MeN – a novel KG embedding model to memorize and encode the potential dependencies among relations and entities for two real applications of triple classification and search personalization.
- Experimental results show that R-MeN obtains better performance than up-to-date embedding models, in which R-MeN produces new state-of-the-art results on SEARCH17

for the search personalization task, and a new highest accuracy on WN11 and the second-highest accuracy on FB13 for the triple classification task.

2 The proposed R-MeN

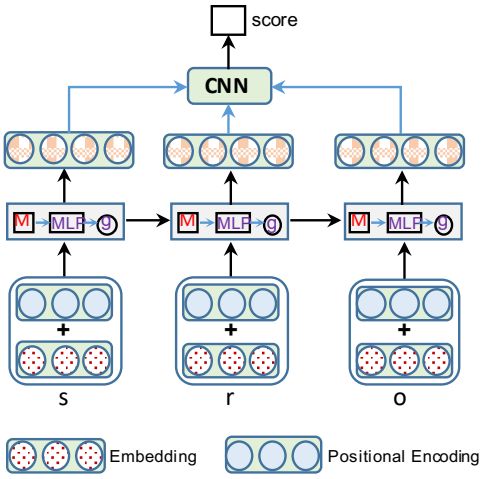

Figure 1: Processes in our proposed R-MeN for an illustration purpose. "M" denotes a memory. "MLP" denotes a multi-layer perceptron. "g" denotes a memory gating. "CNN" denotes a convolutional neural network-based decoder.

Let $\mathcal{G}$ be a KG database of valid triples in the form of *(subject, relation, object)* denoted as *(s, r, o)*. KG embedding models aim to compute a score for each triple, such that valid triples obtain higher scores than invalid triples.

We denote $\mathbf{v}_s$, $\mathbf{v}_r$ and $\mathbf{v}_o \in \mathbb{R}^d$ as the embeddings of s, r and o, respectively. Besides, we hypothesize that relative positions among s, r and o are useful to reason instinct relationships; hence we add to each position a positional embedding. Given a triple *(s, r, o)*, we obtain a sequence of 3 vectors $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ as:

$$\begin{aligned}\mathbf{x}_1 &= \mathbf{W}(\mathbf{v}_s + \mathbf{p}_1) + \mathbf{b}\\ \mathbf{x}_2 &= \mathbf{W}(\mathbf{v}_r + \mathbf{p}_2) + \mathbf{b}\\ \mathbf{x}_3 &= \mathbf{W}(\mathbf{v}_o + \mathbf{p}_3) + \mathbf{b}\end{aligned}$$

where $\mathbf{W} \in \mathbb{R}^{k \times d}$ is a weight matrix, and $\mathbf{p}_1, \mathbf{p}_2$ and $\mathbf{p}_3 \in \mathbb{R}^d$ are positional embeddings, and k is the memory size.

We assume we have a memory M consisting of N rows wherein each row is a memory slot. We use $M^{(t)}$ to denote the memory at timestep t, and $M^{(t)}_{i,:} \in \mathbb{R}^k$ to denote the i-th memory slot at timestep t. We follow Santoro et al. (2018) to take $\mathbf{x}_t$ to update $M^{(t)}_{i,:}$ using the multi-head self-attention mechanism (Vaswani et al., 2017) as:

$$\begin{aligned}\hat{M}^{(t+1)}_{i,:} &= [\hat{M}^{(t+1),1}_{i,:} \oplus \hat{M}^{(t+1),2}_{i,:} \oplus \\ &\quad \ldots \oplus \hat{M}^{(t+1),H}_{i,:}]\\ \text{with } \hat{M}^{(t+1),h}_{i,:} &= \alpha_{i,N+1,h}\left(\mathbf{W}^{h,V}\mathbf{x}_t\right)\\ &\quad + \sum_{j=1}^{N}\alpha_{i,j,h}\left(\mathbf{W}^{h,V}M^{(t)}_{j,:}\right)\end{aligned}$$

where H is the number of attention heads, and $\oplus$ denotes a vector concatenation operation. Regarding the h-th head, $\mathbf{W}^{h,V} \in \mathbb{R}^{n \times k}$ is a value-projection matrix, in which n is the head size and $k = nH$. Note that $\{\alpha_{i,j,h}\}_{j=1}^{N}$ and $\alpha_{i,N+1,h}$ are attention weights, which are computed using the softmax function over scaled dot products as:

$$\begin{aligned}\alpha_{i,j,h} &= \frac{\exp(\beta_{i,j,h})}{\sum_{m=1}^{N+1}\exp(\beta_{i,m,h})}\\ \alpha_{i,N+1,h} &= \frac{\exp(\beta_{i,N+1,h})}{\sum_{m=1}^{N+1}\exp(\beta_{i,m,h})}\\ \text{with } \beta_{i,j,h} &= \frac{\left(\mathbf{W}^{h,Q}M^{(t)}_{i,:}\right)^{\mathsf{T}}\left(\mathbf{W}^{h,K}M^{(t)}_{j,:}\right)}{\sqrt{n}}\\ \beta_{i,N+1,h} &= \frac{\left(\mathbf{W}^{h,Q}M^{(t)}_{i,:}\right)^{\mathsf{T}}\left(\mathbf{W}^{h,K}\mathbf{x}_t\right)}{\sqrt{n}}\end{aligned}$$

where $\mathbf{W}^{h,Q} \in \mathbb{R}^{n \times k}$ and $\mathbf{W}^{h,K} \in \mathbb{R}^{n \times k}$ are query-projection and key-projection matrices, respectively. As following Santoro et al. (2018), we feed a residual connection between $\mathbf{x}_t$ and $\hat{M}^{(t+1)}_{i,:}$ to a multi-layer perceptron followed by a memory gating to produce an encoded vector $\mathbf{y}_t \in \mathbb{R}^k$ for timestep t and the next memory slot $M^{(t+1)}_{i,:}$ for timestep $(t+1)$.

As a result, we obtain a sequence of 3 encoded vectors $\{\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3\}$ for the triple (s, r, o). We then use a CNN-based decoder to compute a score for the triple as:

$$f(s, r, o) = \mathsf{max}\left(\mathsf{ReLU}\left([\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3] * \mathbf{\Omega}\right)\right)^{\mathsf{T}}\mathbf{w}$$

where we view $[\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3]$ as a matrix in $\mathbb{R}^{k \times 3}$; $\mathbf{\Omega}$ denotes a set of filters in $\mathbb{R}^{m \times 3}$, in which m is the window size of filters; $\mathbf{w} \in \mathbb{R}^{|\mathbf{\Omega}|}$ is a weight vector; $*$ denotes a convolution operator; and max denotes a max-pooling operator. Note that we use the max-pooling operator – instead of the vector

concatenation of all feature maps used in ConvKB (Nguyen et al., 2018) – to capture the most important feature from each feature map, and to reduce the number of weight parameters.

We illustrate our proposed R-MeN as shown in Figure 1. In addition, we employ the Adam optimizer (Kingma and Ba, 2014) to train R-MeN by minimizing the following loss function (Trouillon et al., 2016; Nguyen et al., 2018):

$$\mathcal{L} = \sum_{(s,r,o)\in\{\mathcal{G}\cup\mathcal{G}'\}} \log\left(1+\exp\left(-t_{(s,r,o)}\cdot f\left(s,r,o\right)\right)\right)$$

$$\text{in which, } t_{(s,r,o)} = \begin{cases} 1 \text{ for } (s,r,o)\in\mathcal{G} \\ -1 \text{ for } (s,r,o)\in\mathcal{G}' \end{cases}$$

where $\mathcal{G}$ and $\mathcal{G}'$ are collections of valid and invalid triples, respectively. $\mathcal{G}'$ is generated by corrupting valid triples in $\mathcal{G}$.

3 Experimental setup

3.1 Task description and evaluation

3.1.1 Triple classification

The triple classification task is to predict whether a given triple (s, r, o) is valid or not (Socher et al., 2013). Following Socher et al. (2013), we use two benchmark datasets WN11 and FB13, in which each validation or test set consists of the same number of valid and invalid triples. It is to note in the test set that Socher et al. (2013) did not include triples that either or both of their subject and object entities also appear in a different relation type or order in the training set, to avoid reversible relation problems. Table 1 gives statistics of the experimental datasets.

Dataset	$\#\mathcal{E}$	$\#\mathcal{R}$	#Triples in train/valid/test		
FB13	75,043	13	316,232	11,816	47,466
WN11	38,696	11	112,581	5,218	21,088

Table 1: Statistics of the experimental datasets. $\#\mathcal{E}$ is the number of entities. $\#\mathcal{R}$ is the number of relations.

Each relation r has a threshold θ_r computed by maximizing the micro-averaged classification accuracy on the validation set. If the score of a given triple (s, r, o) is above θ_r, then this triple is classified as a valid triple, otherwise, it is classified as an invalid one.

3.1.2 Search personalization

In search personalization, given a submitted *query* for a *user*, we aim to re-rank the *documents* returned by a search system, so that the more the returned documents are relevant for that query, the higher their ranks are. We follow (Vu et al., 2017; Nguyen et al., 2019a,b) to view a relationship of the submitted query, the user and the returned document as a *(s, r, o)*-like triple *(query, user, document)*. Therefore, we can adapt our R-MeN for the search personalization task.

We evaluate our R-MeN on the benchmark dataset SEARCH17 (Vu et al., 2017) as follows: (i) We train our model and use the trained model to compute a score for each *(query, user, document)* triple. (ii) We sort the scores in the descending order to obtain a new ranked list. (iii) We employ two standard evaluation metrics: mean reciprocal rank (MRR) and Hits@1. For each metric, the higher value indicates better ranking performance.

3.2 Training protocol

3.2.1 Triple classification

We use the common Bernoulli strategy (Wang et al., 2014; Lin et al., 2015) when sampling invalid triples. For WN11, we follow Guu et al. (2015) to initialize entity and relation embeddings in our R-MeN by averaging word vectors in the relations and entities, i.e., $\mathbf{v}_{american\_arborvitae} = \frac{1}{2}\left(\mathbf{v}_{american} + \mathbf{v}_{arborvitae}\right)$, in which these word vectors are taken from the Glove 50-dimensional pre-trained embeddings (Pennington et al., 2014) (i.e., d = 50). For FB13, we use entity and relation embeddings produced by TransE to initialize entity and relation embeddings in our R-MeN, for which we obtain the best result for TransE on the FB13 validation set when using l_2-norm, learning rate at 0.01, margin γ = 2 and d = 50.

Furthermore, on WN11, we provide our new fine-tuned result for TransE using our experimental setting, wherein we use the same initialization taken from the Glove 50-dimensional pre-trained embeddings to initialize entity and relation embeddings in TransE. We get the best score for TransE on the WN11 validation set when using l_1-norm, learning rate at 0.01, margin γ = 6 and d = 50.

In preliminary experiments, we see the highest accuracies on the validation sets for both datasets when using a single memory slot (i.e., $N = 1$); and this is consistent with utilizing the single memory slot in language modeling (Santoro et al., 2018). Therefore, we set $N = 1$ to use the single memory slot for the triple classification task. Also from preliminary experiments, we select the batch size $bs = 16$ for WN11 and $bs = 256$ for FB13, and

set the window size m of filters to 1 (i.e., $m = 1$).

Regarding other hyper-parameters, we vary the number of attention heads H in $\{1, 2, 3\}$, the head size n in $\{128, 256, 512, 1024\}$, the number of MLP layers l in $\{2, 3, 4\}$, and the number of filters $F = |\mathbf{\Omega}|$ in $\{128, 256, 512, 1024\}$. The memory size k is set to be $nH = k$. To learn our model parameters, we train our model using the Adam initial learning rate lr in $\{1e^{-6}, 5e^{-6}, 1e^{-5}, 5e^{-5}, 1e^{-4}, 5e^{-4}\}$. We run up to 30 epochs and use a grid search to select the optimal hyper-parameters. We monitor the accuracy after each training epoch to compute the relation-specific threshold θ_r to get the optimal hyper-parameters (w.r.t the highest accuracy) on the validation set, and to report the final accuracy on the test set.

3.2.2 Search personalization

We use the same initialization of user profile, query and document embeddings used by Nguyen et al. (2019b) on SEARCH17 to initialize the corresponding embeddings in our R-MeN respectively. From the preliminary experiments, we set $N = 1$, $bs = 16$ and $m = 1$. Other hyper-parameters are varied as same as used in the triple classification task. We monitor the MRR score after each training epoch to obtain the highest MRR score on the validation set to report the final scores on the test set.

4 Main results

4.1 Triple classification

Table 2 reports the accuracy results of our R-MeN model and previously published results on WN11 and FB13. R-MeN sets a new state-of-the-art accuracy of 90.5% that significantly outperforms other models on WN11. R-MeN also achieves a second highest accuracy of 88.9% on FB13. Overall, R-MeN yields the best performance averaged over these two datasets.

Regarding TransE, we obtain the second-best accuracy of 89.2% on WN11 and a competitive accuracy of 88.1% on FB13. Figure 2 shows the accuracy results for TransE and our R-MeN w.r.t each relation. In particular, on WN11, the accuracy for the one-to-one relation "similar_to" significantly increases from 50.0% for TransE to 78.6% for R-MeN. On FB13, R-MeN improves the accuracies over TransE for the many-to-many relations "institution" and "profession".

Method	WN11	FB13	Avg.
NTN (Socher et al., 2013)	86.2	87.2	86.7
TransH (Wang et al., 2014)	78.8	83.3	81.1
TransR (Lin et al., 2015)	85.9	82.5	84.2
TransD (Ji et al., 2015)	86.4	**89.1**	87.8
TransR-FT (Feng et al., 2016)	86.6	82.9	84.8
TranSparse-S (Ji et al., 2016)	86.4	88.2	87.3
TranSparse-US (Ji et al., 2016)	86.8	87.5	87.2
ManifoldE (Xiao et al., 2016a)	87.5	87.2	87.4
TransG (Xiao et al., 2016b)	87.4	87.3	87.4
lppTransD (Yoon et al., 2016)	86.2	88.6	87.4
ConvKB (Nguyen et al., 2019a)	87.6	88.8	88.2
TransE (Bordes et al., 2013) (ours)	89.2	88.1	88.7
Our R-MeN model	**90.5**	88.9	**89.7**
TransE-NMM (Nguyen et al., 2016)	86.8	88.6	87.7
TEKE_H (Wang and Li, 2016)	84.8	84.2	84.5
Bilinear-COMP (Guu et al., 2015)	87.6	86.1	86.9
TransE-COMP (Guu et al., 2015)	84.9	87.6	86.3

Table 2: Accuracy results (in %) on the WN11 and FB13 test sets. The last 4 rows report accuracies of the models that use relation paths or incorporate with a large external corpus. The best score is in bold while the second best score is in underline. "Avg." denotes the averaged accuracy over two datasets.

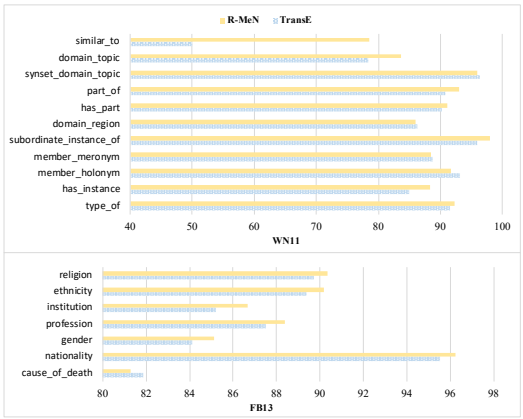

Figure 2: Accuracies for R-MeN and TransE w.r.t each relation on WN11 and FB13.

4.2 Search personalization

Table 3 presents the experimental results on SEARCH17, where R-MeN outperforms up-to-date embedding models and obtains the new highest performances for both MRR and Hits@1 metrics. We restate the prospective strategy proposed by Vu et al. (2017) in utilizing the KG embedding methods to improve the ranking quality of the personalized search systems.

Method	MRR	H@1
SE (Original rank)	0.559	38.5
CI (Teevan et al., 2011)	0.597	41.6
SP (Vu et al., 2015)	0.631	45.2
TransE (Bordes et al., 2013)	0.669	50.9
ConvKB (Nguyen et al., 2019a)	0.750	59.9
CapsE (Nguyen et al., 2019b)	0.766	62.1
Our **R-MeN**	**0.778**	**63.6**

Table 3: Experimental results on the SEARCH17 test set. Hits@1 (H@1) is reported in %. Our improvements over all baselines are statistically significant with $p < 0.05$ using the paired t-test.

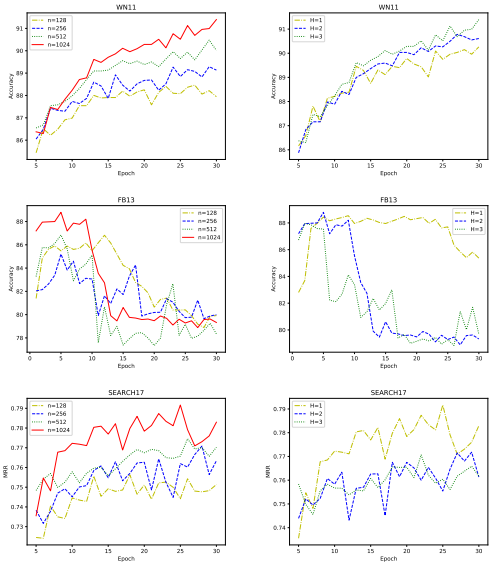

Figure 3: Effects of the head size n and the number H of attention heads on the validation sets.

4.3 Effects of hyper-parameters

Next, we present in Figure 3 the effects of hyper-parameters consisting of the head size n, and the number H of attention heads. Using large head sizes (e.g., $n = 1024$) can produce better performances on all 3 datasets. Additionally, using multiple heads gives better results on WN11 and FB13, while using a single head (i.e., $H = 1$) works best on SEARCH17 because each query usually has a single intention.

4.4 Ablation analysis

For the last experiment, we compute and report our ablation results over 2 factors in Table 4. In particular, the scores degrade on FB13 and SEARCH17 when not using the positional embeddings. More importantly, the results degrade on all 3 datasets without using the relational memory network. These show that using the positional embeddings can explore the relative positions among s, r and o; besides, using the relational memory network helps to memorize and encode the potential dependencies among relations and entities.

Model	WN11	FB13	SEARCH17
Our **R-MeN**	**91.3**	**88.8**	**0.792**
(a) w/o Pos	91.3	88.7	0.787
(b) w/o M	89.6	88.4	0.771

Table 4: Ablation results on the validation sets. (i) Without using the positional embeddings. (ii) Without using the relational memory network, thus we define $f(s, r, o) = \mathsf{max}\left(\mathsf{ReLU}\left([\mathbf{v}_s, \mathbf{v}_r, \mathbf{v}_o] * \mathbf{\Omega}\right)\right)^{\mathsf{T}} \mathbf{w}$.

5 Conclusion

We propose a new KG embedding model, named R-MeN, where we integrate transformer self-attention mechanism-based memory interactions with a CNN decoder to capture the potential dependencies in the KG triples effectively. Experimental results show that our proposed R-MeN obtains the new state-of-the-art performances for both the triple classification and search personalization tasks. In future work, we plan to extend R-MeN for multi-hop knowledge graph reasoning. Our code is available at: https://github.com/daiquocnguyen/R-MeN.

Acknowledgements

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

References

Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. 2013. Translating Embeddings for Modeling Multi-relational Data. In *Advances in Neural Information Processing Systems 26*, pages 2787–2795.

Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. 2011. Learning Structured Embeddings of Knowledge Bases. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages 301–306.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. 2018. Convolutional 2D Knowledge Graph Embeddings. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 1811–1818.

Jun Feng, Minlie Huang, Mingdong Wang, Mantong Zhou, Yu Hao, and Xiaoyan Zhu. 2016. Knowledge Graph Embedding by Flexible Translation. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference*, pages 557–560.

Kelvin Guu, John Miller, and Percy Liang. 2015. Traversing Knowledge Graphs in Vector Space. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 318–327.

Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Knowledge Graph Embedding via Dynamic Mapping Matrix. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 687–696.

Guoliang Ji, Kang Liu, Shizhu He, and Jun Zhao. 2016. Knowledge Graph Completion with Adaptive Sparse Transfer Matrix. In *Proceedings of the Thirtieth Conference on Artificial Intelligence*, pages 985–991.

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

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning Entity and Relation Embeddings for Knowledge Graph Completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence Learning*, pages 2181–2187.

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

Dai Quoc Nguyen, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Phung. 2018. A Novel Embedding Model for Knowledge Base Completion Based on Convolutional Neural Network. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 327–333.

Dai Quoc Nguyen, Thanh Vu, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Phung. 2019b. A Capsule Network-based Embedding Model for Knowledge Graph Completion and Search Personalization. In *Proceedings of the 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 2180–2189.

Dat Quoc Nguyen. 2017. An Overview of Embedding Models of Entities and Relationships for Knowledge Base Completion. *arXiv preprint*, arXiv:1703.08098.

Dat Quoc Nguyen, Kairit Sirts, Lizhen Qu, and Mark Johnson. 2016. Neighborhood Mixture Model for Knowledge Base Completion. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 40–50.

Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. 2016. A Review of Relational Machine Learning for Knowledge Graphs. *Proceedings of the IEEE*, 104(1):11–33.

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

Adam Santoro, Ryan Faulkner, David Raposo, Jack Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy Lillicrap. 2018. Relational Recurrent Neural Networks. In *Advances in Neural Information Processing Systems*, pages 7299–7310.

Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. 2013. Reasoning With Neural Tensor Networks for Knowledge Base Completion. In *Advances in Neural Information Processing Systems 26*, pages 926–934.

Jaime Teevan, Daniel J. Liebling, and Gayathri Ravichandran Geetha. 2011. Understanding and Predicting Personal Navigation. In *Proceedings of the ACM International Conference on Web Search and Data Mining*, pages 85–94.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex Embeddings for Simple Link Prediction. In *Proceedings of the 33nd International Conference on Machine Learning*, pages 2071–2080.

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

Thanh Vu, Dat Quoc Nguyen, Mark Johnson, Dawei Song, and Alistair Willis. 2017. Search Personalization with Embeddings. In *Proceedings of the European Conference on Information Retrieval*, pages 598–604.

Thanh Vu, Alistair Willis, Son Ngoc Tran, and Dawei Song. 2015. Temporal Latent Topic User Profiles for Search Personalisation. In *Proceedings of the European Conference on Information Retrieval*, pages 605–616.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge Graph Embedding by Translating on Hyperplanes. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 1112–1119.

Zhigang Wang and Juan-Zi Li. 2016. Text-Enhanced Representation Learning for Knowledge Graph. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 1293–1299.

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

Han Xiao, Minlie Huang, and Xiaoyan Zhu. 2016a. From One Point to A Manifold: Knowledge Graph Embedding for Precise Link Prediction. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 1315–1321.

Han Xiao, Minlie Huang, and Xiaoyan Zhu. 2016b. TransG : A Generative Model for Knowledge Graph Embedding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2316–2325.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2015. Embedding Entities and Relations for Learning and Inference in Knowledge Bases. In *Proceedings of the International Conference on Learning Representations*.

Hee-Geun Yoon, Hyun-Je Song, Seong-Bae Park, and Se-Young Park. 2016. A Translation-Based Knowledge Graph Embedding Preserving Logical Property of Relations. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 907–916.

Chapter 5

Conclusion

This thesis has set out to develop new embedding models to learn representations for graph-structured data, focusing on undirected graphs and knowledge graphs. In this chapter, we summarise our solutions and contributions for the research questions we have articulated in Chapter 1. We then discuss possible directions for future work.

5.1 Contributions

In Chapter 3, we addressed the research questions on developing new graph neural networks (GNNs) for undirected graphs:

RQ 1: *How can we develop an advanced aggregation function to better update node representations from their neighbours?*

We introduced U2GNN (Nguyen et al., 2019b) – an advantageous variant of GNNs – which induces a powerful aggregation function leveraging the transformer self-attention network (Vaswani et al., 2017; Dehghani et al., 2019) to improve the graph classification performance. We demonstrated that U2GNN outperforms up-to-date models and produces state-of-the-art accuracies on well-known benchmark datasets.

Another research question we addressed focused on inferring embeddings for new nodes:

RQ 2: *How can we develop an effective learning process to infer embeddings for new nodes?*

We presented two new unsupervised models SANNE (Nguyen et al., 2020d) and Caps2NE (Nguyen et al., 2020a) that generate random walks and then leverage the transformer self-attention network (Vaswani et al., 2017) and the capsule network (Sabour et al., 2017) respectively, to learn node embeddings. Both models aim to infer effective embeddings not only for existing nodes but also for new nodes. We showed that both SANNE and Caps2NE obtain state-of-the-art accuracy results for the node classification task.

Besides, in Chapter 3, we addressed another research question of moving beyond the Euclidean space to increase representation capability:

RQ 3: *How can we move beyond the Euclidean space to learn better graph representations and reduce the number of model parameters?*

We proposed to learn node and graph embeddings in the Quaternion space as this space provides highly expressive computations through the Hamilton product compared to the Euclidean and complex vector spaces. We studied our strategy by introducing our quaternion graph neural networks (Nguyen et al., 2020b) to generalise GCNs (Kipf and Welling, 2017). Our proposed model obtains state-of-the-art accuracies on a range of well-known benchmark datasets for three downstream tasks of graph classification, semi-supervised node classification, and text (node) classification.

In Chapter 4, we addressed the following research question for knowledge graphs:

RQ 4: *How can we develop deep KG embedding approaches to better model relationships among entities?*

We introduced two new embedding models, ConvKB (Nguyen et al., 2018, 2019a) and CapsE (Nguyen et al., 2019c), to capture global relationships and translation characteristics between entities and relations. The former leverages a convolutional layer to output the feature maps, which are then concatenated and computed with a weight vector to calculate the triple score. The latter extends ConvKB in using a capsule network (Sabour et al., 2017) to reconstruct the feature maps into the corresponding capsules, which are then routed to another capsule whose length is used to return the triple score. We showed that our models perform better than previous embedding models for the knowledge graph

completion task.

Furthermore, we focused on another research question on enhancing the relation-aware correlations between the entities:

RQ 5: *How can we increase the correlations between the entities in KGs beyond the Euclidean space?*

We presented QuatRE (Nguyen et al., 2020e) to learn the embeddings of entities and relations within the Quaternion space with the Hamilton product. In particular, QuatRE utilises two relation-aware rotations to strengthen the correlations between the head and tail entities. Experimental results demonstrated that our proposed QuatRE outperforms up-to-date embedding models and produces state-of-the-art performances for the knowledge graph completion task.

Also, in Chapter 4, we addressed the last research question regarding two other tasks of triple classification and search personalisation:

RQ 6: *How can we develop an advanced KG embedding model for two other applications of triple classification and search personalisation?*

We proposed a new KG embedding model, named R-MeN (Nguyen et al., 2020c), to effectively capture and encode the potential dependencies among relations and entities. To this end, R-MeN integrates a transformer-based memory network with a CNN-based decoder to compute the triple score. We found that our proposed R-MeN obtains new state-of-the-art performances for both the triple classification and search personalisation tasks.

5.2 Future work

This thesis broadly covers both graph neural networks and knowledge graph embeddings, which have been received significant attention from the scientific community because of their limitless potentials in many real-world applications. In what follows, we highlight some promising directions for future work.

Regarding GNNs, as pointed out in Section 3.2.3, to the best of our knowledge, our

proposed QGNN is the first work that takes advantages of quaternion embedding space to GNNs, and proves the effectiveness in the downstream tasks of graph classification, semi-supervised node classification, and text (node) classification. It would also be valuable to develop QGNN-based methods for open graph benchmark (Hu et al., 2020). Furthermore, it is worth noting that most of the existing applications using GCN (Kipf and Welling, 2017) can also adopt our framework. Thus, we plan to further investigate QGNN for other applications in computer vision, natural language processing, and healthcare.

Regarding KG embeddings, as mentioned in Section 2.2.3.3, beside conventional embedding models such as TransE (Bordes et al., 2013), DistMult (Yang et al., 2015), ComplEx (Trouillon et al., 2016), and ConvE (Dettmers et al., 2018), recent approaches have adapted GNNs for knowledge graph completion (Schlichtkrull et al., 2018; Shang et al., 2019; Nathani et al., 2019; Vashishth et al., 2020b). In general, vanilla GNNs are modified and utilised as an encoder module to update vector representations for entities and relations before feeding to a decoder module that adopts a score function (e.g., as employed in TransE, DistMult, and ConvE) to return the triple scores. For example, R-GCN (Schlichtkrull et al., 2018) modifies GCNs (Kipf and Welling, 2017) to build a specific encoder to update only entity embeddings. CompGCN (Vashishth et al., 2020b) customises GCNs to adapt composition operations between entities and relations in the encoder module. These GNN-based embedding models, however, are still outperformed by other conventional models such as TuckER (Balažević et al., 2019b) on challenging and difficult benchmark datasets. We note that none of them treats each relation as an individual node to consider coherences among entities and relations in the encoder module; hence this could lower their performance. Therefore, a potential direction for KG embeddings is to develop a GNN-based embedding model by integrating node coherences among entities and relations.

Bibliography

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv:1607.06450*, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations (ICLR)*, 2015.

Ivana Balažević, Carl Allen, and Timothy Hospedales. Multi-relational poincaré graph embeddings. In *Advances in Neural Information Processing Systems*, pages 4465–4475, 2019a.

Ivana Balažević, Carl Allen, and Timothy M Hospedales. Tucker: Tensor factorization for knowledge graph completion. In *Empirical Methods in Natural Language Processing*, pages 5185–5194, 2019b.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of Machine Learning Research*, pages 1137–1155, 2003.

Paul N. Bennett, Ryen W. White, Wei Chu, Susan T. Dumais, Peter Bailey, Fedor Borisyuk, and Xiaoyuan Cui. Modeling the impact of short- and long-term behavior on search personalization. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 185–194, 2012.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 1247–1250, 2008.

Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning Structured Embeddings of Knowledge Bases. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages 301–306, 2011.

Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating Embeddings for Modeling Multi-relational Data. In *Advances in Neural Information Processing Systems 26*, pages 2787–2795, 2013.

Karsten M. Borgwardt and Hans-Peter Kriegel. Shortest-Path Kernels on Graphs. In *ICDM*, pages 74–81, 2005.

Hongyun Cai, Vincent W Zheng, and Kevin Chang. A comprehensive survey of graph embedding: problems, techniques and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

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 *Advances in Neural Information Processing Systems*, pages 4869–4880, 2019.

Ines Chami, Adva Wolf, Da-Cheng Juan, Frederic Sala, Sujith Ravi, and Christopher Ré. Low-dimensional hyperbolic knowledge graph embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6901–6914, 2020.

Haochen Chen, Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. A tutorial on network embeddings. *arXiv:1808.02590*, 2018a.

Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations*, 2018b.

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 *Proceedings of the 2014*

Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1724–1734, 2014.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *International Conference on Machine Learning*, 2008.

Zhiyong Cui, Kristian Henrickson, Ruimin Ke, and Yinhai Wang. High-order graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting, 2018.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal Transformers. *International Conference on Learning Representations (ICLR)*, 2019.

Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2D Knowledge Graph Embeddings. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, pages 1811–1818, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Xin Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 601–610, 2014.

Alberto Garcia Duran and Mathias Niepert. Learning graph representations with embedding propagation. In *Advances in Neural Information Processing Systems*, pages 5119–5130, 2017.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

Hongyang Gao and Shuiwang Ji. Graph u-nets. In *International Conference on Machine Learning*, pages 2083–2092, 2019.

Alberto García-Durán and Mathias Niepert. KBLRN: End-to-end learning of knowledge base representations with latent, relational, and numerical features. *arXiv preprint abs/1709.04676*, 2017.

Thomas Gärtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning Theory and Kernel Machines*, pages 129–143, 2003.

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.

Aditya Grover and Jure Leskovec. Node2Vec: Scalable Feature Learning for Networks. In *The ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 855–864, 2016.

Kelvin Guu, John Miller, and Percy Liang. Traversing Knowledge Graphs in Vector Space. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 318–327, 2015.

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

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

Yanchao Hao, Yuanzhe Zhang, Kang Liu, Shizhu He, Zhanyi Liu, Hua Wu, and Jun Zhao. An end-to-end model for question answering over knowledge base with cross-attention combining global knowledge. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 221–231, 2017.

Morgan Harvey, Fabio Crestani, and Mark J. Carman. Building user profiles from topic models for personalised search. In *Proceedings of the ACM International Conference on Information and Knowledge Management*, pages 2309–2314, 2013.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. *arXiv preprint arXiv:1808.06670*, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, pages 1735–1780, 1997.

Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The annals of statistics*, pages 1171–1220, 2008.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. In *Advances in neural information processing systems*, pages 4558–4567, 2018.

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

Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. Knowledge Graph Embedding via Dynamic Mapping Matrix. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 687–696, 2015.

Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graphs. In *International Conference on Machine Learning*, pages 321–328, 2003.

Gjergji Kasneci, Fabian M Suchanek, Georgiana Ifrim, Maya Ramanath, and Gerhard Weikum. Naga: Searching and ranking knowledge. In *Proceedings of the 24th IEEE International Conference on Data Engineering*, pages 953–962, 2008.

Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. In *Advances in neural information processing systems*, pages 4284–4295, 2018.

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

Nils M Kriege, Fredrik D Johansson, and Christopher Morris. A survey on graph kernels. *arXiv:1903.11835*, 2019.

Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196, 2014.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278–2324, 1998.

Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International Conference on Machine Learning*, 2019.

Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems*, pages 2177–2185, 2014.

Juzheng Li, Jun Zhu, and Bo Zhang. Discriminative deep random walk for network classification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1004–1013, 2016a.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated Graph Sequence Neural Networks. *International Conference on Learning Representations (ICLR)*, 2016b.

David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.

Yankai Lin, Zhiyuan Liu, Huanbo Luan, Maosong Sun, Siwei Rao, and Song Liu. Modeling Relation Paths for Representation Learning of Knowledge Bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 705–714, 2015a.

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning Entity and Relation Embeddings for Knowledge Graph Completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence Learning*, pages 2181–2187, 2015b.

Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Qi Ju, Haotang Deng, and Ping Wang. K-bert: Enabling language representation with knowledge graph. In *AAAI*, pages 2901–2908, 2020.

Yuanfei Luo, Quan Wang, Bin Wang, and Li Guo. Context-Dependent Knowledge Graph Embedding. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1656–1661, 2015.

Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064–1074, 2016.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013b.

Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. *arXiv:1707.05005*, 2017.

Deepak Nathani, Jatin Chauhan, Charu Sharma, and Manohar Kaul. Learning attention-based embeddings for relation prediction in knowledge graphs. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4710–4723, 2019.

Dai Quoc Nguyen, Tu Dinh Nguyen, Dat Quoc Nguyen, and Dinh Phung. A Novel Embedding Model for Knowledge Base Completion Based on Convolutional Neural Network. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 327–333, 2018.

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. URL `https://doi.org/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, Dat Quoc Nguyen, and Dinh Phung. A Capsule Network-based Embedding Model for Knowledge Graph Completion and Search Personalization. In *Proceedings of the 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 2180–2189, 2019c.

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*, CIKM '20, page 3313–3316, New York, NY, USA, 2020a. Association for Computing Machinery. ISBN 9781450368599. doi: 10.1145/3340531.3417455. URL `https://doi.org/10.1145/3340531.3417455`.

Dai Quoc Nguyen, Tu Dinh Nguyen, and Dinh Phung. Quaternion graph neural networks. *NeurIPS 2020 Workshop on Differential Geometry meets Deep Learning. arXiv preprint arXiv:2008.05089*, 2020b.

Dai Quoc Nguyen, Tu Dinh Nguyen, and Dinh Phung. A Relational Memory-based Embedding Model for Triple Classification and Search Personalization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 3429—-3435, 2020c.

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)*, 2020d. doi: 10.1007/978-3-030-67664-3_22. URL `https://doi.org/10.1007/978-3-030-67664-3_22`.

Dai Quoc Nguyen, Thanh Vu, Tu Dinh Nguyen, and Dinh Phung. Quatre: Relation-aware quaternions for knowledge graph embeddings. *NeurIPS 2020 Workshop on Differential Geometry meets Deep Learning. arXiv preprint arXiv:2009.12517*, 2020e.

Dat Quoc Nguyen. A survey of embedding models of entities and relationships for knowl-

edge graph completion. In *Proceedings of the 14th Workshop on Graph-based Methods for Natural Language Processing*, pages 1–14, 2020.

Dat Quoc Nguyen, Richard Billingsley, Lan Du, and Mark Johnson. Improving Topic Models with Latent Feature Word Representations. *Transactions of the Association for Computational Linguistics*, 3:299–313, 2015.

Dat Quoc Nguyen, Kairit Sirts, Lizhen Qu, and Mark Johnson. STransE: a novel embedding model of entities and relationships in knowledge bases. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 460–466, 2016.

Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. Holographic Embeddings of Knowledge Graphs. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 1955–1961, 2016.

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

Giannis Nikolentzos, Giannis Siglidis, and Michalis Vazirgiannis. Graph kernels: A survey. *arXiv:1904.12218*, 2019.

Stefi Nouleho Ilemo, Dominique Barth, Olivier David, Franck Quessette, Marc-Antoine Weisser, and Dimitri Watel. Improving graphs of cycles approach to structural similarity of molecules. *PLOS ONE*, 14(12):1–25, 12 2019. doi: 10.1371/journal.pone.0226680.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

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

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.

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 (EMNLP 2014)*, pages 1532–1543, 2014.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online Learning of Social Representations. In *The ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 701–710, 2014.

Nataša Pržulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):e177–e183, 2007.

Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pages 3859–3869, 2017.

Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin Riedmiller, Raia Hadsell, and Peter Battaglia. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning*, pages 4470–4479, 2018.

Adam Santoro, Ryan Faulkner, David Raposo, Jack Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy Lillicrap. Relational Recurrent Neural Networks. In *Advances in Neural Information Processing Systems*, pages 7299–7310, 2018.

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.

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.

Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 298–307, 2015.

Michael Schuhmacher and Simone Paolo Ponzetto. Knowledge-based graph document modeling. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, pages 543–552, 2014.

Kristof T Schütt, Farhad Arbabzadah, Stefan Chmiela, Klaus R Müller, and Alexandre Tkatchenko. Quantum-chemical insights from deep tensor neural networks. *Nature communications*, 8(1):1–8, 2017.

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 *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3060–3067, 2019.

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

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.

Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning With Neural Tensor Networks for Knowledge Base Completion. In *Advances in Neural Information Processing Systems 26*, pages 926–934, 2013a.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, 2013b.

Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A core of semantic knowledge. In *Proceedings of the 16th International Conference on World Wide Web*, pages 697–706, 2007.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations*, 2019.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, pages 3104–3112, 2014.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: Large-scale Information Network Embedding. In *WWW*, pages 1067–1077, 2015.

Jaime Teevan, Susan T. Dumais, and Eric Horvitz. Personalizing search via automated analysis of interests and activities. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 449–456, 2005.

Jaime Teevan, Meredith Ringel Morris, and Steve Bush. Discovering and using groups to improve personalized search. In *Proceedings of the ACM International Conference on Web Search and Data Mining*, pages 15–24, 2009.

Jaime Teevan, Daniel J. Liebling, and Gayathri Ravichandran Geetha. Understanding and Predicting Personal Navigation. In *Proceedings of the ACM International Conference on Web Search and Data Mining*, pages 85–94, 2011.

Kristina Toutanova and Danqi Chen. Observed Versus Latent Features for Knowledge Base and Text Inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pages 57–66, 2015.

Kristina Toutanova, Danqi Chen, Patrick Pantel, Hoifung Poon, Pallavi Choudhury, and Michael Gamon. Representing Text for Joint Embedding of Text and Knowledge Bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509, 2015.

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex Embeddings for Simple Link Prediction. In *Proceedings of the 33nd International Conference on Machine Learning*, pages 2071–2080, 2016.

Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, Nilesh Agrawal, and Partha Talukdar. Interacte: Improving convolution-based knowledge graph embeddings by increasing feature interactions. In *International Conference on Learning Representations*, 2020a.

Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. In *International Conference on Learning Representations*, 2020b.

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

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

Petar Velickovic, William Fedus, William L Hamilton, Pietro Lio, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *International Conference on Learning Representations (ICLR)*, 2019.

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

S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11(Apr):1201–1242, 2010.

Thanh Vu, Alistair Willis, Son Ngoc Tran, and Dawei Song. Temporal Latent Topic User Profiles for Search Personalisation. In *Proceedings of the European Conference on Information Retrieval*, pages 605–616, 2015.

Thanh Vu, Dat Quoc Nguyen, Mark Johnson, Dawei Song, and Alistair Willis. Search Personalization with Embeddings. In *Proceedings of the European Conference on Information Retrieval*, pages 598–604, 2017.

Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *The ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1225–1234, 2016.

Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *The ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 839–848, 2018.

Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge Graph Embedding: A Survey of Approaches and Applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

Quan Wang, Pingping Huang, Haifeng Wang, Songtai Dai, Wenbin Jiang, Jing Liu, Yajuan Lyu, Yong Zhu, and Hua Wu. Coke: Contextualized knowledge graph embedding. *arXiv preprint arXiv:1911.02168*, 2019a.

Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The World Wide Web Conference*, pages 2022–2032, 2019b.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge Graph Embedding by Translating on Hyperplanes. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 1112–1119, 2014.

Zhigang Wang and Juan-Zi Li. Text-Enhanced Representation Learning for Knowledge Graph. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 1293–1299, 2016.

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.

Jason Weston, Antoine Bordes, Sumit Chopra, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *CoRR*, abs/1502.05698, 2015.

Ryen W. White, Wei Chu, Ahmed Hassan, Xiaodong He, Yang Song, and Hongning Wang. Enhancing personalized search by mining and modeling task behavior. In *Proceedings of the World Wide Web conference*, pages 1411–1420, 2013.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International Conference on Machine Learning*, 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.

Chenyan Xiong, Russell Power, and Jamie Callan. Explicit semantic ranking for academic search via knowledge graph embedding. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1271–1279, 2017.

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

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

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *The ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 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 *Proceedings of the International Conference on Learning Representations*, 2015.

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

Liang Yao, Chengsheng Mao, and Yuan Luo. Kg-bert: Bert for knowledge graph completion. *arXiv preprint arXiv:1909.03193*, 2019.

Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *The ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 974–983, 2018a.

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, 2018b.

Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. In *Advances in Neural Information Processing Systems*, pages 11960–11970, 2019.

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

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

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

Yuanzhe Zhang, Kang Liu, Shizhu He, Guoliang Ji, Zhanyi Liu, Hua Wu, and Jun Zhao. Question answering over knowledge base with neural attention combining global knowledge information. *arXiv preprint arXiv:1606.00979*, 2016.

Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *arXiv:1812.04202*, 2018b.

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.