Leveraging Basis Alignment to create a Generalized Multi-Relational Graph Convolution Network in the Federated Setting

by

Nicholas Ramirez

S.B. in Electrical Engineering and Computer Science at Massachusetts Institute of Technology (2022)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2023

Authored By: Nicholas Ramirez
Department of Electrical Engineering and Computer Science
May 12, 2023

Certified by: Lalana Kagal
Principal Research Scientist
Thesis Supervisor

Accepted by: Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Leveraging Basis Alignment to create a Generalized Multi-Relational Graph Convolution Network in the Federated Setting

by

Nicholas Ramirez

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 2023, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Knowledge graphs have seen a significant rise in popularity and usage in recent years with many real-world applications taking advantage of their ability to model interlinked data easily. In general, many institutions maintain their own knowledge graphs, however these graphs tend to suffer from incompleteness. This is due to two main reasons: knowledge is naturally distributed across institutions and institutions are unable to share sensitive data. With this in mind, federated learning appears to be a promising solution to this problem as it enables clients to develop a shared global model without sharing any data. This thesis aims to solve the knowledge graph completion problem by introducing a federated learning protocol for the state-of-the-art Knowledge Embedding Based Graph Convolutional Network (KE-GCN) [51]. KE-GCN was chosen for it's unification of multiple graph convolutional networks and it's ability to provide as much flexibility as possible for clients. As a result, my federated protocol, Fed-KE-GCN, is focused on data privacy and flexibility. In addition to Fed-KE-GCN, this thesis empirically shows that a common approach for differential privacy for deep learning, Differentially Private Stochastic Gradient Descent (DP-SGD) [2], is not viable in this domain due to the nature of graph data and the internal framework of Graph Convolutional Networks.

Thesis Supervisor: Lalana Kagal
Title: Principal Research Scientist

Acknowledgments

I would like to acknowledge and give my deepest gratitude to my supervisor Lalana for her great guidance, support, and advice throughout this process. This work would not have been possible without her mentorship and expertise on the subject. In addition, I would like to express a sincere thanks to Vaikkunth for his great advice and help throughout the year. Vaik provided me help and suggestions at any time I needed despite his demanding schedule from starting his own company. I am very grateful for the invaluable assistance from Lalana and Vaik.

In addition, I want to thank the 6.390 Fall 2022 and 6.101 Spring 2023 staff for granting me the opportunity to be a teaching assistant. The overall experience was very enriching, and one of the main highlights of my year.

I would also like to extend my appreciation to two teachers who are directly responsible for where I am at today. Sharon Veenhoff and Matthew Judge from BAHS played a crucial role in mentoring me during high school. I would not be at MIT, let alone completing my thesis, without their influence and guidance.

Furthermore, I am thankful for my colleagues, friends, and Baker 1W for their support. They gave me much needed motivation throughout college and especially during my work on this thesis.

Last, but not least, I would like to thank my Mom, Dad, siblings - Dante, Amanda, and Vanessa -, and my family for their much needed support and love. Their encouragement was a main driving force for me, and I was only able to continue and complete this thesis because of them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Knowledge graphs have become an increasingly used data structure to represent human knowledge with many real-world applications such as recommendation systems, question answering, and dialogue systems taking advantage [28]. A knowledge graph is a directed graph that consists of nodes, which are entities such as a person or place, and directed edges, which signify a relationship between the nodes. Knowledge graphs are usually represented as triples (head, relation, tail). Both the head and tail components of a triple are nodes, and the relation component is a directed edge from head to tail. This structure provides a efficient way to store and model a significant amount of interlinked data. However, knowledge graphs tend to suffer from incompleteness. For example, one of the largest knowledge graphs commonly used in research, Freebase, has around 71% of person entities lacking a birthplace [12]. As a result, current literature focuses on knowledge graph completion by inferring missing facts. Specifically, most literature focuses on link prediction or node classification. The link prediction task attempts to predict the existence of a link (or relationship) between two entities. For node classification, the task aims to classify entities in order to infer characteristics. Both these tasks are typically accomplished by creating low-dimensional representations of the knowledge graph (knowledge graph embeddings) or utilizing machine learning models.

The problem of knowledge graph incompleteness inherently stems from knowledge being naturally distributed among different institutions, and in a lot of cases, insti-

tutions are unable to share their data. For example, sensitive data, such as medical or financial data, is highly private and regulated. Even beyond regulated data, institutions aren't incentivized to reveal their data directly to other companies due to the increased privacy risk. Federated learning [48] is a distributed machine learning paradigm that has seen recent success within this private data dilemma. As a result, some recent research efforts have taken a federated learning approach to knowledge graph completion as a way to collaboratively learn over knowledge graphs distributed across institutions without requiring the actual data to be exchanged. However, most recent literature is either not compatible with multiple embedding choices or lacks a flexible implementation that can enable clients to utilize the most desirable model for their application.

My research addresses the knowledge graph incompleteness problem through the creation of a federated system with a specific focus on maintaining generalizability. Specifically, I have implemented Fed-KE-GCN, which is a federated protocol that takes advantage of basis decomposition and optimal transport to enable the generalizable KE-GCN model to work in a decentralized setting. I chose KE-GCN as my core model because of its state-of-the-art performance and generalizability, or in other words, its ability to be compatible and encompass different applications. The KE-GCN model can fully recover the most popular Graph Convolutional Networks as well as utilize many different embeddings. The resultant system enables clients and the server to easily utilize the most suitable model for their respective problems. Furthermore, my federated learning system enables clients to use any uniform embedding approach as long as all clients utilize the same embedding. In addition to building a federated protocol for KE-GCN, I empirically show that a common approach for implementing differential privacy in machine learning models, DP-SGD, is unusable for ensuring privacy due to the nature of Graph Convolutional Networks.

Chapter 2

Background

This section provides an extensive foundation of information and context necessary for understanding the landscape of the knowledge completion problem. I start by defining knowledge graphs, their properties, and the knowledge completion problem itself. Next, I describe general approaches to the knowledge competition problem. I also formally define federated learning and differential privacy both of which my work utilizes. Lastly, I discuss state-of-the-art approaches and models used in current literature.

2.1 Knowledge Graphs

As mentioned in Chapter 1, knowledge graphs are directed graphs used to model interlinked data. At the highest level, knowledge graphs contain nodes, edges, and labels; knowledge graphs can be represented using Resource Description Framework (RDF) triples. These triples are in the form (head, relation, tail) where the head and tail items are nodes and the relation item is the directed edge from head to tail. The nodes in a knowledge graph represent entities such as real-world objects, events, or concepts. Labels refer to general information about the individual knowledge graph components. For example, these labels can be general categories that nodes in the graph can belong to. Figure 2-1 shows an example of a knowledge graph, and an example of a triple from this figure would be (Da Vinci, is a, Person). Here, there are

no specified labels, but an example would be May 12 having the label "date".

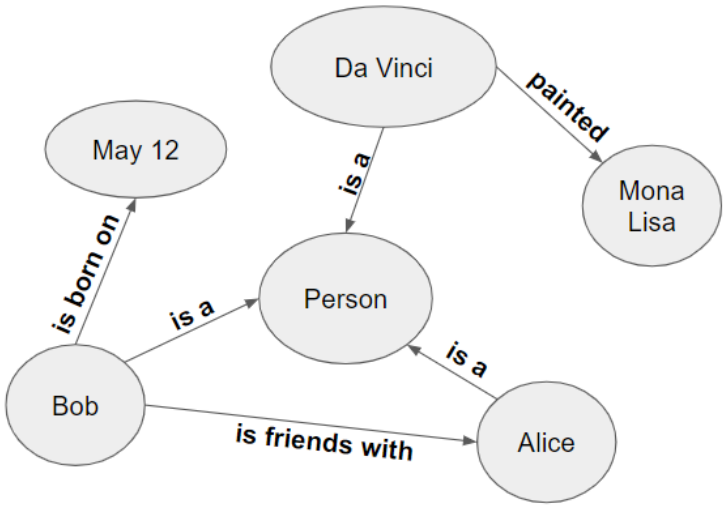

Figure 2-1: Knowledge graph example.

Formally, we can define a knowledge graph $\mathcal{G}$ that has $\mathcal{E}$ as the entity set and $\mathcal{R}$ as the set of relations within the graph. $\mathcal{G}$ consists of triples (h, r, t) such that r $\in \mathcal{R}$ and h,t $\in \mathcal{E}$. In this case, (h, r, t) is the short form for (head, relation, tail).

2.2 Knowledge Graph Completion

In this section, I define the two most common learning problems on knowledge graphs: link prediction and node classification.

Given a triple (h,r,t) where $h, t \in \mathcal{E}$ and $r \in \mathcal{R}$, the link prediction task aims to find the missing triple in the knowledge graph. In addition, the link prediction task can be broken down into two subtasks: entity ranking and relation predicting. For entity ranking, the goal is to find the entity that correctly fulfills the triple. Specifically, the subtask aims to find the correct entity that fulfills (h, r, ?) or (?, r, t). The other subtask, relation prediction, aims to find the correct relation that fulfills (h, ?, t). Figure 2-2 depicts a simple example of link prediction on the example knowledge graph from Figure 2-1.

In order to complete the link prediction task, the majority of literature aims to rank triples in a way that valid triples are ranked higher relative to triples not in the

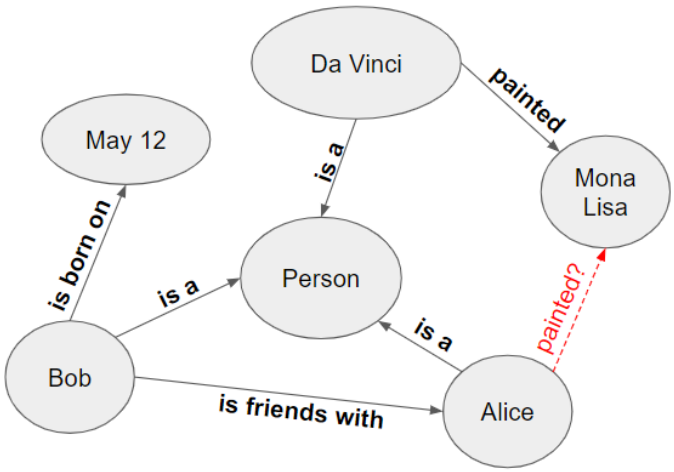

Figure 2-2: Link prediction task example

graph (called negative triples) [44, 32]. The procedure to create negative triples is to take every triple in the testing set and replace that triple's relation with every other relation in the relation set. As a result, the evaluation for link prediction is based on where the valid triple ranks among the negative samples. In order to evaluate link prediction, the two standard metrics used are mean reciprocal rank (MRR) and hits@N. Hits@N is simply the ratio of the valid test triples that are ranked in the top N triples. MRR is defined below where T is the set of test triples.

$$MRR = \frac{1}{N} * \sum_{x \in T} \frac{1}{rank(x)} \tag{2.1}$$

The entity classification task is a semi-supervised task aimed at classifying entities in order to assign categorical properties. Since this semi-supervised task is a classification problem, the most common evaluation metric is accuracy.

2.2.1 Transductive and Inductive

There are two settings for both knowledge graph completion tasks: transductive and inductive. Both settings only differ at the inference stage of the task. In the transductive setting, the model's training and inference graph are the same. In other words, the model is trained on the same graph it will make ultimately make predictions

for. A consequence of the transductive setting is the model has "seen" all entities and relations. The inductive setting removes this requirement that the graph must be same for both training and predicting. Specifically, the inference stage is done on an unseen graph (differs from training) where relation and entity sets may not be the same. In general, the majority of literature focuses on the transductive setting as noted in [15].

2.3 Knowledge Graphs Embeddings

In this section, I describe one of the most important concepts used in many approaches to the knowledge graph completion problem: the creation of a knowledge graph embedding. I also discuss the three common ways in literature these embeddings are created. First, I start by defining an embedding below.

Embeddings are one of the most used tools in machine learning and data science. In general, embeddings don't have a formal ontology since every embedding will rely on the type of model training them and the data itself. However, I will define an embedding as a mapping of discrete data to a low-dimensional vector of continuous numbers. These low-dimensional vectors should contain important semantics from the discrete data. In the context of machine learning, these embeddings are typically learned. A common example that highlights how semantics might be contained within an embedding is that the more similar two data points are, the closer they are in the embedding space (in terms of distance). Ultimately, embeddings create meaningful representations of the data that are memory efficient and usable by various techniques such as machine learning models.

Since knowledge graphs are naturally sparse with very large adjacency matrices, statistical relation learning (SRL) has been a common approach to the knowledge graph completion task, especially the creation and use of low-dimensional embeddings [26]. In the context of knowledge graphs, the general intuition for using embeddings comes from distributional semantics where the meaning of something can be derived from its context. As a result, the embeddings will model entities and relations us-

ing the contextual information found in the topological structure of the graph and ultimately extrapolate patterns. The learned embedding space can then be used to easily infer information and knowledge. One of the main structural patterns that can be modeled is the inherent relational properties that exist in a knowledge graph: symmetry, antisymmetry, inverse, and transitive. I formally define these properties below [38]:

Symmetric:

Definition: A relation r is symmetric if $\forall x, y : (x, r, y) \rightarrow (y, r, x)$

Example: x=Alice and y=Bob and r="married to". (x,r,y)= Alice is married to Bob $\rightarrow$ (y,r,x) = Bob is married to Alice.

Antisymmetric:

Definition: A relation r is antisymmetric if $\forall x, y : (x, r, y) \rightarrow \neg(y, r, x)$

Example: x=Alice and y=Bob and r="teacher of". (x,r,y)= Alice is a teacher of Bob $\rightarrow$ $\neg$(y,r,x) = Bob is not a teacher of Alice.

Inverse:

Definition: A relation r_1 is inverse to r_2 if $\forall x, y : (x, r_2, y) \rightarrow (y, r_1, x)$

Example: x=Alice and y=Bob and r_1 ="advisor to" and r_2 ="advisee of". (x,r_2,y)= Alice is an advisee of Bob $\rightarrow$ (y,r_1,x)= Bob is an advisor to Alice.

Composition (Transitive):

Definition: A relation r_1 is composed of relation r_2 and r_3 if $\forall x, y, z : (x, r_2, y) \wedge (y, r_3, z) \rightarrow (x, r_1, z)$

Example: x=Charlie, y=Bob, z=Alice and r_1 ="grandfather", r_2 ="father", r_3 ="father". (x,r_2,y)= Charlie is a father to Bob and (y,r_3,z)= Bob is a father to Alice $\rightarrow$ (x,r_1,z)= Charlie is a grandfather to Alice.

The common methodology for creating an embedding is defining a score function, $f_s(h, r, t)$, for the triples and creating an embedding space that maximizes this score function for true triples over false triples. Specifically, a given triple (h,r,t) in $\mathcal{G}$ should

have a larger score value than false triples (h′, r, t) or (h,r,t′). Thus, the score function represents the plausibility that the triple is true. One of the most common measures of these scoring functions is its expressivity which is the degree the embedding can capture the structural patterns defined above. There are three general categories of learned embeddings used in recent literature: translational, semantic, and Graph Convolutional Networks.

2.3.1 Translational Embeddings

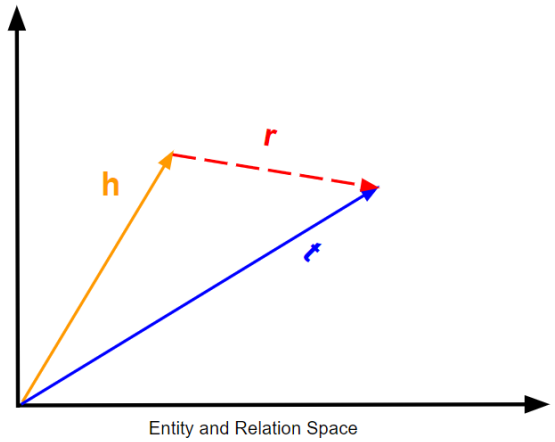

Figure 2-3: Translational embedding example.

Translational embeddings are categorized as embeddings created from a score function that utilizes distance within the embedding space to determine similarity. Specifically, these models aim to learn a low-dimensional embedding of the entities in the graph by utilizing some form of translation between pairs of the entities themselves. The intuition for these embeddings is best depicted in a simple example from one of the most commonly used baselines TransE [4]. The score function for TransE is $f_s(h, r, t) = -||h + r - t||$. In training, the score function is minimized over true triples in the graph. As a result, the score function will ultimately model the value of these vectors such that $h + r \approx t$ if the triple (h,r,t) exists in the graph (it is true). In Figure 2-3, I depict the result of a single triple with the (head, relation, tail) as h, r, and t respectively. In this case, t can be viewed as a translation of vector h by vector

r. All translational embeddings utilize this notion of modeling the distance between entities and relations as a translation in the embedding space.

2.3.2 Semantic Matching

Semantic matching embeddings are categorized as embeddings created from a score function that utilizes the similarity of the low-dimensional representation features. A couple of intuitive examples for calculating the similarity between low-dimensional features include cosine similarity, inner product, and a bilinear scoring function.

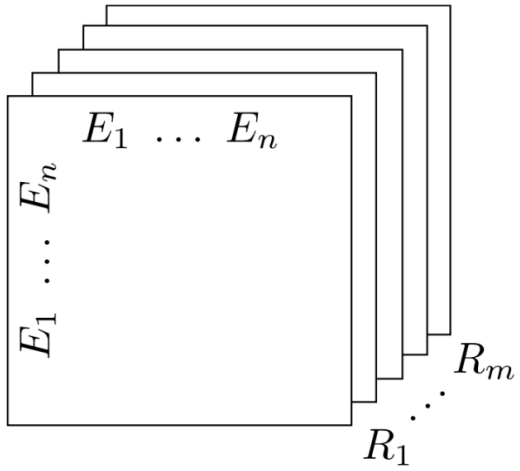

Figure 2-4: Tensor representation with $E_i \in \mathcal{E}$ and $R_i \in \mathcal{R}$ [8].

To provide a more general framework, most of the semantic matching models are built on tensor factorization since knowledge graphs can naturally be represented with tensors. Tensors are multidimensional arrays and can be categorized into first-order, second-order, and higher-order. These categorizations respectively refer to a vector, matrix, and any array with an order greater than two. Tensor factorization is representing a tensor as a series of basic operations on other tensors [19, 29]. In the context of knowledge graphs, I represent the knowledge graph as a ($|\mathcal{E}|$ by $|\mathcal{E}|$ by $|\mathcal{R}|$) tensor as seen in Figure 2-4. In this tensor, the value at an index (i,j,k) is 1 if (E_i, R_k, E_j) is in the knowledge graph, otherwise, it is 0. With this representation, the score function can be bilinear as shown in Figure 2-5 with the model DistMult

[47]. Ultimately, the scoring function can be intuitively viewed as calculating the similarity between h and t dependent on r.

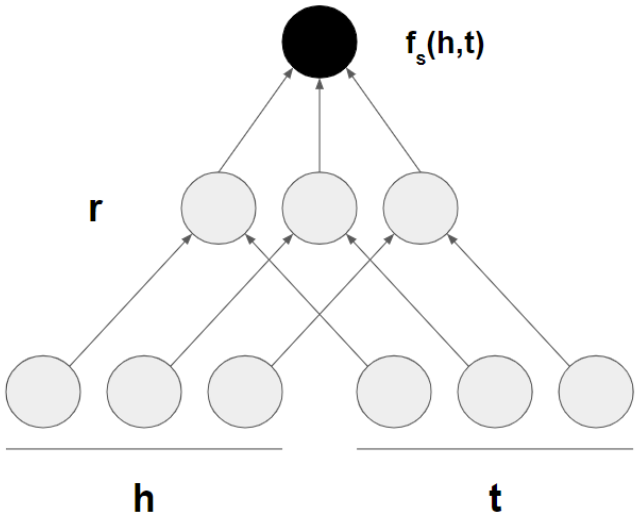

Figure 2-5: Simplified DistMult with a bilinear operation.

2.3.3 Graph Convolutional Networks

The last category of learned embeddings is embeddings derived from Graph Convolutional Networks (GCNs). Recently, these graph models have seen a lot of success [46, 52, 55] on non-interlinked data. As a result, GCNs have been adapted to work on interlinked data and has seen state-of-the-art performance [49]. The general framework for GCNs is built on an iterative aggregation and update scheme that aims to build a continuous representation for a given node [18]. The aggregation stage, also called message passing, will ultimately build this continuous representation that contains the individual node's information as well as all of the local information from the nodes surrounding it. Specifically for every iteration, the model aggregates all nodes' embeddings directly connected to a given node, and updates that given node's embedding. In the first iteration, only the immediately surrounding node information is shared. However, after the first iteration, a node's information is passed to nodes it is not directly connected to. As a result, the graph's local information is propagated throughout the individual node embeddings (up to a distance determined

by the number of iterations), and the graph's topological structure is encapsulated. The total number of layers in the model determines how many iterations, also called hops, will take place. Specifically, the aggregation and update scheme for a generic GCN is depicted for a single layer as:

$$m_v^{l+1} = \sum_{u \in N(v)} h_u^l \tag{2.2}$$

$$h_v^{l+1} = \sigma(W^l(m_v^{l+1} + h_v^l)) \tag{2.3}$$

where h_v^l is the embedding of node v at layer l, N(v) is the set of directly connected nodes to node v, m_v^l is the aggregation of node v's neighbors, W^l is the learned parameters of the model, and σ is an activation function. In the case of GCNs in the domain of knowledge graphs, I will utilize the reformulation of this formula presented in [51].

Specifically, the scoring function is best introduced within the context of the aggregation and update scheme. Here, a scoring function is defined as the inner product between two node representations for the generic GCN $f_s(h_u, h_v) = h_u^T h_v$. Plugging this into the scheme, the new aggregation equation becomes:

$$m_v^{l+1} = \sum_{u \in N(v)} \frac{\partial f_s(h_u^l, h_v^l)}{\partial h_v^l} = \frac{\partial(\sum_{u \in N(v)} f_s(h_u^l, h_v^l))}{\partial h_v^l} \tag{2.4}$$

This framework provides a broad and clear view of the structure of the GCN model. For link prediction, GCNs typically use an encoder and decoder setup as shown in Figure 2-6. The encoder section is the actual GCN model as described in the above framework. In order to evaluate the latent representation created by the aggregation of the neighboring nodes (and the relations to those nodes), the decoding section is simply a score function from the common embedding techniques. For node classification, only the GCN model with a chosen score function is sufficient since the evaluation is simply the labels for the nodes themselves.

Graph Convolutional Networks follow a similar pattern to other deep-learning

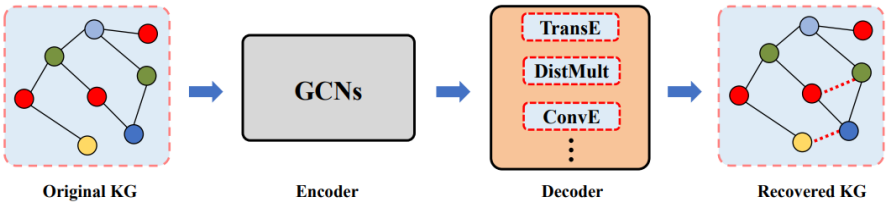

Figure 2-6: GCN encoder-decoder setup [56].

techniques with a training and inference stage. During training, the model's parameters for each layer (W^l) are updated using the respective loss function and backpropagation. Once the model is done training, the model can be used for inference.

2.4 Federated Learning

Federated learning is a privacy-preserving paradigm that aims to collaboratively train a shared machine learning model over multiple clients' data without requiring them to exchange their local data [20]. Formally, I define a federated learning system as a decentralized environment where N clients wish to collaboratively train a model on a single server S without having to expose their data to other clients. This is typically done by having each client train a local model and having the server aggregate these local model parameters. The aggregation of the parameters can be done directly or through the aggregation of the local model's gradients over time. This ultimately enables clients to leverage other clients' data for their own local tasks.

One of the most significant challenges for federated learning is the use of non independent and identically distributed (non-IID) data. With non-IID data, the data between clients can be extremely varied or come from different distributions. This is apparent since clients can be diverse institutions that utilize different methods, and ultimately collect their data in different ways. As a result, the fundamental centralized learning assumption that the data used for training and testing is IID isn't assumed in the federated setting [57]. Furthermore, there is an inherent heterogeneity in knowledge graph data and graph data for that matter. Specifically, knowledge graphs can be statistically or structurally heterogeneous. In the statistical context,

this could simply be a significant difference in the number of nodes and edges. In the structural context, this could be a different surrounding subgraph for a given entity in separate datasets [22]. As a result, a model's accuracy and convergence in this field tend to be significantly affected.

2.5 Differential Privacy

Differential privacy is a rigorous mathematical definition that quantifies an algorithm's degree of privacy protection. I formally define differential privacy such that a random algorithm mechanism $\mathcal{A}$ satisfies (ϵ)-differential privacy for a non-negative number ϵ iff for all neighboring datasets $\mathcal{D}$ and $\mathcal{D}'$ differing by at most one element and all subsets $\mathcal{R}$ of $\mathcal{A}$'s range [14]:

$$\frac{P[\mathcal{A}(\mathcal{D}) \in \mathcal{R}]}{P[\mathcal{A}(\mathcal{D}') \in \mathcal{R}]} \leq e^{\epsilon} \tag{2.5}$$

where ϵ represents the privacy budget and a smaller value means better privacy protection at the cost of accuracy.

In this paper, I will utilize a natural relaxation of differential privacy called Rényi Differential Privacy [24]. I formally define this relaxation as a random algorithm $\mathcal{A}$ is (α, ϵ)-RDP for $\alpha > 1, \epsilon > 0$ if for all neighboring datasets $\mathcal{X}$ and $\mathcal{X}'$ differing by at most one element, $D_\alpha(\mathcal{A}(\mathcal{X})||\mathcal{A}(\mathcal{X}')) \leq \epsilon$ where $D_\alpha(P||Q)$ is the Rényi divergence of order α between probability distributions P and Q as shown as:

$$D_\alpha(P||Q) = \frac{1}{\alpha - 1} log \mathbb{E}_{x \sim Q}[\frac{P(x)}{Q(x)}]^\alpha \tag{2.6}$$

2.6 Related Work

In this section, I summarize state-of-the-art approaches in recent literature on the knowledge graph completion problem and differential privacy for GCNs. The recent literature can be categorized into four main categories: statistical relation learning, centralized machine learning, federated learning, and differential privacy.

2.6.1 Statistical Relation Learning

The creation of knowledge graph embeddings makes up a significant amount of previous work in this field, and these embeddings can achieve state-of-the-art performance if tuned correctly. Below are the four most common embeddings used in recent literature:

TransE [4] models relations as translations between entities in the low dimensional space. Specifically, optimizing the score function $f_s(h, r, t) = -||h + r - t||_x$ where $x \in \{1, 2\}$ enforces tail entities to be close to the sum of the head and relation vectors (h+r $\approx$ t). TransE is the standard baseline model due to its fundamental role in the literature for embeddings, its simplicity, and its great performance.

RotatE [38] models relations as a rotational translation between entities in this latent space, and as a result, it is categorized as a translational embedding. Figure 2-7 depicts this rotational translation with a small example. Formally, h,r,t $\in \mathbb{C}^d$ and the modulus $|r_i| = 1$ for all $r_i \in \mathcal{R}$. The scoring function is defined as $f_s(h, r, t) = -||h \circ r - t||$. RotatE is also a standard baseline and has seen state-of-the-art performance in certain contexts.

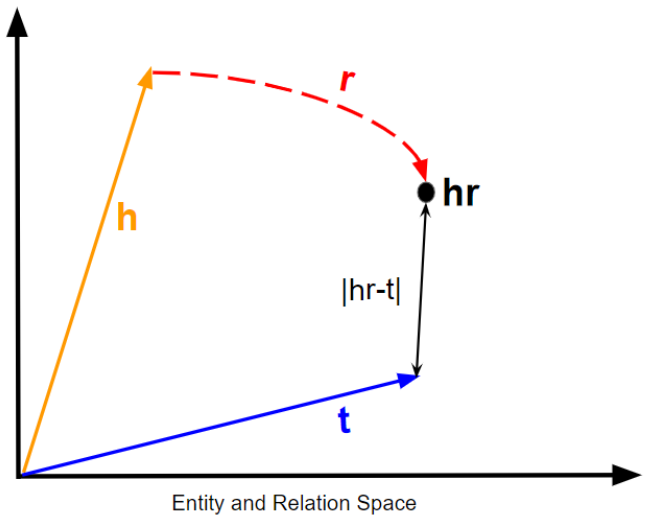

Figure 2-7: RotatE embedding example.

DistMult [47] is a bilinear model with entities represented as a vector h,t $\in \mathbb{R}^d$ and relations are modeled as diagonal matrices $W^r \in \mathbb{R}^{dxd}$. The scoring function

Table 2.1: Different Embeddings and their inherent ability to model relational structural properties.

Model	Score	Sym.	Antisym.	Inv.	Comp.
TransE	$-\|\|h+r-t\|\|$	X	✓	✓	✓
DistMult	$h^T W_r t$	✓	X	X	X
RotatE	$-\|\|h \circ r - t\|\|$	✓	✓	✓	✓
QuatE	$h \otimes r \cdot t$	✓	✓	✓	X

is defined as a bilinear product $f_s(h,r,t) = h^T W_r t$. The intuition is calculating the cosine similarity between $h \cdot r$ and t where $h \cdot r$ is defined as $\Sigma_i h_i \cdot r_i$.

QuatE [54] utilizes hypercomplex representations to model entities and relations. These quaternion embeddings consist of one real component and three imaginary components. Formally, h,r,t $\in \mathbb{H}^d$ and QuatE works by rotating the head entity by the relation ($h_r = h \otimes r$) where $\otimes$ represents the Hamilton Product. The scoring function is defined as $f_s(h,r,t) = h_r \cdot t$. QuatE can be viewed in two steps, rotating the head entity by the unit relation quaternion and then taking the inner product between the rotated head and tail. QuatE is an embedding derived from semantic matching as the central comparison of the scoring function is an inner product.

Each knowledge graph embedding has a different level of capability for capturing the possible relationship properties of the graph as noted in the background. These properties are symmetric, antisymmetric, inverse, and composition. Table 2.1 provides a summarized view of each embedding with its respective score function and whether it can express each relational property in its model.

2.6.2 Centralized Machine Learning

The second common approach to the knowledge graph completion problem is using machine learning models, specifically Graph Convolutional Networks. Below are three of the best-performing GCNs that utilize a relational parameter to work on knowledge graphs:

R-GCN [34] is one of the first successful GCNs that is adapted to interlinked

data by introducing relation-specific transformations to the message-passing formulations. To summarize this key aspect, the R-GCN adds an additional parameter in the message-passing framework that essentially models relations when aggregating neighbors. In particular, it is an extension of the vanilla GCN with a relation linear transformation added to the node representation update equation. The aggregation and update scheme is depicted below:

$$m_v^{l+1} = \sum_{(u,r)\in N_{in}(v)} W_r^l h_u^l + \sum_{(u,r)\in N_{out}(v)} W_r^l h_u^l \tag{2.7}$$

$$h_v^{l+1} = \sigma(m_v^{l+1} + W_0^l h_v^l) \tag{2.8}$$

W-GCN [35] is another top-performing graph convolutional network that extends the vanilla GCN to utilize relations by modeling them as learnable weights of edges. Therefore, the W-GCN essentially treats the complex knowledge graph with numerous relations as multiple subgraphs that consist of only a single relation type. The learnable weight enables the W-GCN to weigh the overall value of the subgraph when creating the node representations. The aggregation and update equations for the node embeddings are defined as:

$$m_v^{l+1} = \sum_{(u,r)\in N_{in}(v)} W^l(\alpha_r^l h_u^l) + \sum_{(u,r)\in N_{out}(v)} W^l(\alpha_r^l h_u^l) \tag{2.9}$$

$$h_v^{l+1} = \sigma(m_v^{l+1} + W^l h_v^l) \tag{2.10}$$

where α_r^l is the relation-specific learned parameter.

Comp-GCN [42] introduced learning a low dimensional vector for relations in addition to nodes. Specifically, Comp-GCN has two aggregation and update schemes. The first scheme is to update the node representation and is similar to R-GCN where it contains relation-specific linear transformations within this entity update. The second scheme introduced in this model is a continuous relation representation which the model will update with every iteration. However, this second scheme differs since

it is only a linear transformation update rather than an aggregation of the neighbors. These schemes also utilize composition operations such as element-wise subtraction, multiplication, or circular correlation as defined by ϕ_{in} and ϕ_{out} in the equations below:

Entity Update:

$$m_v^{l+1} = \sum_{(u,r)\in N_{in}(v)} W_r^l \phi_{in}(h_u^l, h_r^l) + \sum_{(u,r)\in N_{out}(v)} W_r^l \phi_{out}(h_u^l, h_r^l) \tag{2.11}$$

$$h_v^{l+1} = \sigma(m_v^{l+1} + W_0^l h_v^l) \tag{2.12}$$

Relation Update:

$$h_r^{l+1} = W_{rel}^l h_r^l \tag{2.13}$$

KE-GCN [51] is the model used in my proposed federated protocol. The KE-GCN model builds upon the work of Comp-GCN by expanding the relation representation to be updated in a similar manner to the node representation. Specifically, the relation representation updates itself with the aggregation of neighbor entity representations to gather the semantics of the relations. This is different from Comp-GCN where the relations are updated by linear transformations. In addition to building upon Comp-GCN, KE-GCN is defined as a generalizable model since it can fully recover both Comp-GCN, W-GCN, and R-GCN. The overall framework, notion of fully recovering a model, and how to fully recover these models will be explained in depth within the methods chapter.

2.6.3 Federated Learning

The total literature on federated learning for knowledge graphs is relatively small, but all related works commonly aim to bring these knowledge graph embeddings into a federated setting. I categorize the literature into two sections, one is aggregation over the embeddings themselves and the other is an aggregation with a model. FedE [6], and FedR [53] are both protocols that securely aggregate a client's learned embedding.

These learned embeddings are the most common SRL embeddings such as TransE. For FedE, the secure aggregation is strictly over the entity embeddings. For FedR, the paper argues FedE contains privacy leakage where an attacker can reconstruct the client's knowledge graph from the aggregated entity embeddings. To overcome this, the model only aggregates the relation embeddings as reconstructing a triple from just the known relation has a lot more possibilities.

The second category of related works is focused on the design of an aggregation protocol with some use of a larger deep-learning model. Differentially Private Federated Knowledge Graphs Embedding (FKGE) [27] is a privacy-preserving embedding that utilizes PATE-GAN [50] to create differentially private embeddings. Specifically, PATE-GAN is used to combine pairs of client embeddings such that each client's embeddings are impossible to discriminate from each other. However, this complex setup requires a significant amount of time and resources to train a PATE-GAN model between pairs of clients every single time there is an update. The second approach in this category is MaKEr (Meta-Learning Based Knowledge Extrapolation) [5] which is a Graph Convolutional Network in the federated setting that focused on inductive tasks. Specifically, the entire goal of the model is to extrapolate as much information from a knowledge graph such that it can perform on unseen entities and relations.

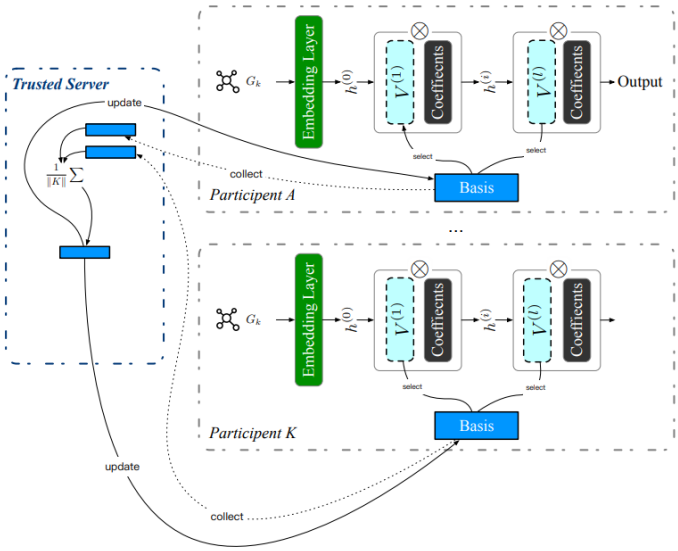

Figure 2-8: General architecture for Fed-Align [22].

My work takes on an approach outlined by Fed-Align [22] which falls under this second category of an aggregation protocol for a deep-learning model. The paper introduces Fed-Align which takes advantage of the internal basis decomposition for parameters within R-GCNs [39]. For the R-GCN update equation 2.7, basis decomposition is a technique of learning a common basis for all relations instead of a unique W_r for all relations. At a high level, the use of basis decomposition minimizes the total number of parameters to learn which is useful as GCNs can become massive as the number of relations grows. In addition, basis decomposition provides regularization over the parameters which prevents overfitting to a specific relation. In addition to basis decomposition, the paper utilizes optimal transport to enforce convergence between clients. Both basis decomposition and optimal transport will be discussed in more in-depth within the methods section, but the general architecture of a Fed-Align is depicted in Figure 2-8.

2.6.4 Differential Privacy with GCNs

Most of the current literature surrounding the use of differential privacy in GCNs utilizes a complex approach in order to create differentially private models that protect against privacy leakage on specific aspects of the graph. In particular, I define two distinct types of differential privacy for knowledge graphs that protect against a set of characteristics from leaking in the graph: node-level and edge-level differential privacy. These types protect node information and edge information in the graph respectively.

One of the most successful approaches to creating differentially private machine learning models is Differentially-Private Stochastic Gradient Descent (DP-SGD) [2]. The algorithm is a direct adaptation to Stochastic Gradient Descent to create differentially private outputs from a machine learning model by applying a calculated amount of random noise to the gradients. As a result, the parameters and outputs of the model are guaranteed to be differentially private.

The first paper for GCNs with differential privacy is the model GAP [33] which decouples the GCN into three individual segments: the encoder, aggregation, and

classification segments. By decoupling the GCN into three segments, noise can be added to each segment individually in order to guarantee different levels of differential privacy. In particular, GAP guarantees both node and edge privacy if noise is added to all three segments. In addition, the decoupling enables a more careful analysis and bounds on the amount of noise necessary since the aggregation module can be precomputed and stored for use.

The second paper introduces a differential privacy approach for the graph classification task [25]. Specifically, the paper applies DP-SGD within a Graph Neural Network for the specific task of graph classification. In this environment, the aim of the model is to protect entire graphs rather than individual nodes or edges. For this task, the dataset is a collection of graphs and the goal is to classify each graph into a category. Using the differential privacy definition, the two datasets $\mathbb{D}$ and $\mathbb{D}'$ differ by one element and in this case, the one element is an entire graph.

The last paper on differential privacy with Graph Neural Networks focuses on ensuring node-level privacy [10]. The paper carefully bounds the total number of node occurrences used during training by sampling subgraphs for batches. This enables a more careful bound on the total amount of noise necessary to ensure node-level differential privacy since message-passing will pass the added noise to its surrounding neighborhood. With these subgraphs, the paper applies DP-SGD directly over these batches.

Chapter 3

Method

The main contribution of my thesis is a generalizable KE-GCN system that performs in a federated setting to enable clients to flexibly utilize the most relevant model for the given task. Being able to provide a model that is generalizable is significant as the diverse nature of knowledge graphs as datasets cause no single model to be considered universally optimal. This is shown in the literature as performance varies significantly for models depending on the dataset. As a result, the core model for my federated protocol is KE-GCN due to its state-of-the-art performance in the centralized setting and its ability to fully recover both the R-GCN, W-GCN, and Comp-GCN. This allows a significant amount of flexibility for clients to choose the best-performing model for their given dataset and task. For my work, I implemented a federated learning approach due to the nature of knowledge graph incompleteness stemming from unsharable sensitive data among institutions and knowledge graph information naturally being distributed among many institutions. My federated approach enables an effective way to utilize distributed data while maintaining the privacy of sensitive data. In addition to a federated learning protocol, my work also focuses on privacy of the data for the centralized setting. Specifically, I implemented a differential private KE-GCN model by utilizing the popular DP-SGD as the optimizer for my model. As a result, my research can be split into 3 main categories: the core KE-GCN model, the federated learning protocol (Fed-KE-GCN), and a differentially private KE-GCN (DP-KE-GCN).

3.1 KE-GCN Model

As mentioned, my core model I used for my work is KE-GCN due to its generalizability. In this section, I describe the internal framework of the model, formally define generalizability, and outline how KE-GCN is generalizable due to its ability to recover other models.

3.1.1 Framework

As briefly mentioned in the related work chapter, the KE-GCN model [51] is a graph convolutional network that follows a similar aggregation and update method as depicted in the Comp-GCN model. The KE-GCN scheme has the same node update equations with a relation-specific transformation to capture relation information within the node embeddings. However, the KE-GCN model expands beyond the Comp-GCN by utilizing the aggregation and update method for relation representations in order to fully take advantage of the connection between entities and relations within the graph. I formally define the node representation aggregation and update equations for KE-GCN below:

$$m_v^{l+1} = \sum_{(u,r)\in N_{in}(v)} W_r^l \frac{\partial f_{in}(h_u^l, h_r^l, h_v^l)}{\partial h_v^l} + \sum_{(u,r)\in N_{out}(v)} W_r^l \frac{\partial f_{out}(h_v^l, h_r^l, h_u^l)}{\partial h_v^l} \tag{3.1}$$

$$h_v^{l+1} = \sigma_e(m_v^{l+1} + W_0^l h_v^l) \tag{3.2}$$

where h_v^l is the embedding of entity v at layer l. N_{in} is the set of tuples of nodes and relations that have an incoming directed edge towards node v. N_{out} is the set of tuples of nodes and relations that have an outgoing directed edge away from node v. f_{in} and f_{out} are the scoring functions for incoming and outgoing neighbors. Lastly, σ_e is the activation function for entity updates. The relation update rule for KE-GCN can be formally defined as:

$$m_r^{l+1} = \sum_{(u,v)\in N(r)} \frac{\partial f_r(h_u^l, h_r^l, h_v^l)}{\partial h_r^l} \tag{3.3}$$

$$h_r^{l+1} = \sigma_r(W_{rel}^l(m_r^{l+1} + h_r^l)) \tag{3.4}$$

where h_r^l is the embedding of relation r at layer l. N_r is the set of entity neighbors that fulfill (h, r, t). Lastly, σ_r is the activation function for relation updates.

In addition to this framework, the KE-GCN model applies a normalization on the aggregation such that m_v^{l+1} in Equation 3.2 becomes $\alpha m_v^{l+1}/(|N_{in}(v)| + |N_{out}(v)|)$ and m_r^{l+1} in Equation 3.4 becomes $\alpha m_r^{l+1}/|N(r)|$. Here, α is a hyperparameter, and the goal of this normalization is to prevent exploding and vanishing gradients.

Ultimately, the framework theoretically enables the maximum amount of information to be learned from the knowledge graph. This framework encapsulates the entities, relations, graph structure, and context surrounding both entities and relations due to the aggregation for both node and relation representations.

3.1.2 Generalizability

Since the main focus of this work is enabling a generalized approach to work in a decentralized setting, I formally define generalizability as the degree to which an approach can be applied to a broader context. In this case, it is the ability of a model to be compatible with (or encompass) different applications. The generalizability of the KE-GCN is manifested through its ability to recover multiple models by simply setting key parts within the framework of the model to specific values. In other words, it is the direct ability to capture multiple frameworks within one approach. This is a significant and valuable characteristic for the decentralized setting with knowledge graphs since the datasets typically vary to a great degree. As a direct result, there is a large variance in the performance of models where typical baselines can outperform proposed models. By utilizing a model that captures and unifies multiple models, my work enables clients to optimize the federated protocol for their given task and data.

As demonstrated in the KE-GCN paper [51], I will show how R-GCN, W-GCN, and Comp-GCN are fully restricted by the generalizable model and how it is fully recoverable within the unified representation framework. In order to fully recover the

R-GCN model with the KE-GCN model, the entire relation representation must be removed by setting $h_r^l = 0$ within Equation 3.4. In addition, I set both $f_{in}(h_u^l, h_r^l, h_v^l)$ and $f_{out}(h_u^l, h_r^l, h_v^l)$ equal to $(h_u^l)^T h_v^l$ for Equation 3.1.

In order to fully recover the W-GCN model with KE-GCN, I show that three values need to be set. In a similar fashion as the case of recovering R-GCN, I remove the entire relation representation by setting $h_r^l = 0$ in Equation 3.4. For Equation 3.1, I set $f_{in}(h_u^l, h_r^l, h_v^l) = f_{out}(h_u^l, h_r^l, h_v^l) = (h_u^l)^T h_v^l$. Finally, I set the learned parameters for KE-GCN in Equation 3.1 to $W_r^l = W^l \alpha_r^l$

Lastly, to fully recover Comp-GCN with the KE-GCN model, I set $f_{in}(h_u^l, h_r^l, h_v^l) = \phi_{in}(h_u^l, h_r^l)^T h_v^l$ and $f_{out}(h_u^l, h_r^l, h_v^l) = \phi_{out}(h_u^l, h_r^l)^T h_v^l$ in the node aggregation Equation 3.1. Next, I set $f_r = 0$ in the relation aggregation equation 3.3. Lastly, the activation function σ_r for the relation update Equation 3.4 is simply the identity function. In addition to fully recovering Comp-GCN, another way that the KE-GCN is more general than previous models is based on the fact that the scoring function f_{in} and f_{out} is not restricted to any operators. As mentioned in the related works, Comp-GCN restricted ϕ_{in} and ϕ_{out} to be composition operators. For example, TransH [45], TransD [16], MLP [13], and NTN [37] are all possible scoring functions that are compatible with KE-GCN, but not Comp-GCN.

3.2 Federated Learning Protocol

This section builds upon the previous section by describing my federated learning protocol for enabling the KE-GCN to work in the decentralized setting. Within this section, I explain my use of basis decomposition and optimal transport for the protocol. At the end of this section, I outline the Fed-KE-GCN protocol step by step.

3.2.1 Basis Decomposition

As briefly mentioned in the background section, basis decomposition is a type of regularization used to prevent overparameterization and overfitting within R-GCN. Specifically, basis decomposition for a relational graph convolutional network is a

technique where you utilize a shared basis and different coefficients for relations. The KE-GCN paper [51] mentions a different approach to preventing overparameterization by setting the linear transformation matrix $W_r^l = W^l$ as all relation parameters will be non-relation-specific. However, my work builds upon the R-GCN concept for basis decomposition into the KE-GCN model by replacing each unique W_r, the parameters learned for every relation in the knowledge graph, to the shared basis and coefficient setup. As a result, my implementation results in a reduction in the total number of parameters as well as enforcing implicit regularization. The regularization is caused by the sharing of a subset of the parameters for each relation. In particular, the use of basis decomposition prevents the model from overfitting to a single specific relation as all relation parameters will share the same basis.

My work takes advantage of basis decomposition to extend the model into the federated setting as shown in [22]. Specifically, for every layer in the model, I implemented that layer's relation parameter matrix as a linear combination of a shared basis and coefficient such that:

$$W_r^l = \sum_{b=1}^{B} a_{rb}^l V_b^l \tag{3.5}$$

where l is the model's layer and $V_b^l \in \mathbb{R}^{d^{(l+1)} x d^l}$ is the basis transformations with coefficients a_{rb}^l.

By representing the model's relation parameter matrices as a linear combination of basis transformations, my federated protocol aligns every corresponding basis for each client's model. This will be discussed more in Fed-KE-GCN protocol, but at a high level, the protocol aggregates the gradients for each basis in each layer. This is sent to the server which applies the gradients and sends its own server gradient to all clients. As a result, the aggregation does not directly rely on the client's nodes or edges since the protocol is aggregating the gradients of the parameters for each local model. Thus, basis decomposition provides a stronger privacy guarantee than aggregating embeddings as seen in FedE and FedR. In addition, the implementation of basis alignment provides a natural unifying aggregation of a GCNs numerous parameters.

3.2.2 Optimal Transport

Using basis alignment by itself is not enough for the protocol to properly perform in a federated setting as convergence is not guaranteed due to the nature of the graph data. In order to understand why, I first use definitions from Li et al.[21] that detail the circumstances necessary to guarantee that federated learning on heterogeneous data will converge so it becomes apparent why Fed-KE-GCN requires more than basis alignment for the federated setting. The first definition is based on the notion that clients will train their model on a local objective. In order to provide flexibility with training, the paper defines the need for the local objective to be solved inexactly. Specifically, the paper defines this idea of γ-inexact solution below:

Definition 3.2.1 (γ_k^t-inexact solution) *For a function $h_k(w; w_t) = F_k(w) + \frac{\mu}{2}||w - w_t||^2$ and $\gamma \in [0, 1]$, let w^* be a γ_k^t-inexact solution of $min_w h_k(w; w_t)$ if $||\nabla h_k(w^*; w_t)|| \leq \gamma_k^t ||\nabla h_k(w_t; w_t)||$ where $\nabla h_k(w; w_t) = \nabla F_k(w) + \mu(w - w_t)$.*

To summarize this definition, it is defining a solution w^* to the local objective $F_k(w)$ as an inexact solution if the gradients are only off by a magnitude γ_k^t. This definition is mostly used to analyze the amount of local computation completed by clients during every iteration. Specifically, γ_k^t can be viewed as a measure of the amount of computation done by client k at time t for the local objective. The second definition describes the dissimilarity between clients in the federated setting.

Definition 3.2.2 (B-local dissimilarity) *Denote f(w) as the global objective and $F_k(w)$ the local objective for k clients. The local objectives are B-locally dissimilar at w if $\mathbb{E}_k[||\nabla F_k(w)||^2] \leq ||\nabla f(w)||^2 B^2$. Furthermore, denote $B(w) = \sqrt{\frac{\mathbb{E}_k[||\nabla F_k(w)||^2]}{||\nabla f(w)||^2}}$ for $^2||\nabla f(w)|| \neq 0$.*

B-local dissimilarity definition can be seen as a generalization of the IID assumption by adding bounded dissimilarity to allow for statistical heterogeneity. As a result, definition 3.2.2 is used to understand the convergence relationship between the local and global objectives. In particular, an optimal weight w that satisfies the local objective can also minimize the global one if and only if f(w) and $F_k(w)$ are similar enough which is directly measured by the above definition.

With these definitions and the assumption that there exists a bound on the dissimilarity, a federated algorithm is guaranteed to converge as proved in Li et al.[21]. For an intuitive explanation, I can utilize the B-local dissimilarity to provide a bound on the dissimilarity between the global and local objectives such that these two objectives are considered similar enough. The γ_k^t-inexact solution is used to prove that clients will ultimately work towards minimizing the local objective across multiple rounds. If all clients are working towards this local objective and the global objective is similar enough to these local objectives, then the clients will converge towards some global solution. These definitions are most likely designed for batched data where the separability of the data is clear. In this domain, the graph data spread around clients is usually separated in a stochastic way, and as a result, doesn't provide a complete and exclusive set. Thus, the expected value for the weights between all clients isn't equal to some stationary weight as desired. In a similar fashion, the significant variances between local objectives will cause large variance for any practical B-local dissimilarity usage. Therefore, convergence for graph modeling data is not guaranteed in the federated setting from the usual standpoint [22].

As a result, I implemented optimal transport (OT) [43] within the loss function of KE-GCN as a solution to the convergence issue. To start, the weights of a federated model can simply be viewed as a sample drawn from a distribution. This perspective is applied to the Fed-KE-GCN's weights within the protocol. First, I assume there exists a solution w^* or stationary weight for the general minimization problem. If w^* is drawn from some distribution and the global weight of the model is a true estimation of the stationary w^*, then it is expected that the distance between the local and global weights will converge to 0. In order to achieve this 0 distance convergence between local and global weights, optimal transport is a common solution [22]. The most intuitive and common explanation of optimal transport is Earth Mover's Distance (EMD) [40] in computer vision. In this case, OT distance is the minimum amount of mass that needs to be moved in order to turn one pile into another. Here, pile is a distribution defined for a given metric space. Furthermore, this concept of the minimum amount of mass to move is also determined by a cost matrix. Optimal

transport considers the total amount to move as well as how expensive it is to move that mass. In a simple example, a value at a given index i,j in the cost matrix is simply the distance between two points i and j. This simple example is the EMD problem or also called Wasserstein-1 distance [7].

I formally define EMD following the Fed-Align paper[22] by starting with U(r,c), where r and c are probability vectors with dimensions n and m, as the set of positive nxm matrices where the rows sum to r and columns sum to c as depicted by:

$$U(r, c) = \{P \in \mathbb{R}_{+}^{nxm} | P1_m = r, P^T 1_n = c\} \tag{3.6}$$

With this set, I can define two multinomial random variables X and Y that take in values 1,2,...n and 1,2,...m with distribution r and c. The given cost matrix M for X and Y variables describes the cost to move X to Y. Specifically at index i,j in matrix M, this is detailing the cost to move X=i to Y=j. In addition to the cost matrix, any matrix P from the set U(r,c) can be identified with a joint probability for X,Y such that p(X=i, Y=j) = p_{ij}. With all this in mind, I can now define EMD as an optimization problem:

$$\texttt{EMD(r,c)} := \texttt{min}_{P \in U(r,c)} \langle P, M \rangle \tag{3.7}$$

In the case of Fed-KE-GCN, r can be viewed as the basis weights on client A's device and c as the basis weights of client B. The EMD will be calculating the total distance between these basis weights for every index according to a specified cost matrix. This will be explained in more depth in the Fed-KE-GCN protocol section.

To reiterate, I've implemented optimal transport for two significant reasons. The first and most important is that using optimal transport provides a solution to the innate issue of divergence for graph-related data in the federated setting. By viewing these weights as a sample drawn from a distribution, I utilize the existing solution of optimal transport to help ensure that the local weights of clients won't diverge from the global and optimal weights. In addition to providing better convergence, using optimal transport to calculate distance seems to be a more natural and suitable ap-

proach to the knowledge graph problem compared to other common distance metrics such as Euclidean distance and Kullback-Leibler Divergence [36] as OT distance is symmetric for two distributions. In my case, calculating the distance between two client's basis should be symmetrical if the desired goal is for each client's basis to approach the same value.

In order to solve the optimal transport optimization problem within a reasonable time, I use sinkhorn distance which is a faster approximation for computing the optimal transport problem.

3.2.3 Sinkhorn Algorithm

The Sinkhorn Algorithm is a faster procedure for solving optimal transport problems [9]. In this case, I utilize Sinkhorn distance in my protocol which is an additional term added to the original EMD equation. This term is a negative regularization term that at a high level represents the need to maximize entropy.

$$\texttt{Sinkhorn(r,c)} := \texttt{min}_{P \in U(r,c)} \langle P, M \rangle - \frac{1}{\lambda} h(P) \tag{3.8}$$

where $h(P) = -\sum_{i,j}^{d} p_{ij} log(p_{ij})$ and λ is a positive real number. h(P) is very similar to an information entropy term, so the addition of this term can be viewed as encouraging higher entropy transport matrices which translates to the transport matrix being more uniformly distributed. The solution to the usual optimal transport equation can be solved with linear programming or any optimizer. This takes $\mathcal{O}(d^3 log(d))$. With sinkhorn distances, the procedure instead scales the rows and columns of P until a valid approximation of the transport matrix is found. This is computed in $\mathcal{O}(d^2)$ [9].

3.2.4 Fed-KE-GCN Protocol

After describing the design choices for basis decomposition and optimal transport, the overall Fed-KE-GCN Protocol can be described. To start, I will define the following conditions necessary to use the protocol. First, all clients must utilize the same model, and in this case, the same model refers not only to the use of KE-GCN, but

the internal choice of representation as well as the same dimensions, hidden layers, and activation functions. Secondly, the protocol requires all clients to utilize the same scoring function for KE-GCN. It is important to note that this still provides great flexibility for the server and clients to determine which model (that is recoverable from KE-GCN) and what scoring function to use for the best results for a given task.

In the protocol, all models utilize basis decomposition such that their parameters are the linear combination of some shared basis between all relation matrices and unique coefficients. In each iteration, a global server chooses N clients and has every client locally update and train their model with their own data. The gradients calculated over these local epochs are collected and sent to the server. Once the server receives gradients from all the clients, the server aggregates by summing all these gradients together (for every individual basis). Then the model returns the aggregated gradient to all clients for the clients to use. This overall procedure is outlined in Algorithm 1.

Every client will locally train their model for a total number of local epochs for every round of the protocol. Let's define the local loss function which every client will optimize:

$$F_k(w) = f_k(w) + \frac{\mu}{||N||} \sum_{j \neq k}^{N} \sum_{l=1}^{L} OT(V_k^l, V_j^l) \tag{3.9}$$

where μ is a hyperparameter (usually set to 1), $f_k(w)$ is the normal loss function (cross entropy for classification for example), and the summation is the application of optimal transport for that client's basis and the basis of every other client that is in the same layer. To be more clear, for client 1, I sum the optimal transport distance between that client's layer one basis and all other client's layer one basis. This is done for each layer and the final loss is the total distance and the actual local loss from the task. From a less mechanical perspective during training, a client model's basis will attempt to converge with the corresponding basis from all other clients as the model attempts to minimize the distance from the loss function. In addition, the normal loss term $f_k(w)$ will lead to the expected performance convergence towards the potential global (in this case the stationary) optimum that is desired for machine

learning.

Algorithm 1 Fed-KE-GCN Protocol

Input: $E_{Global}, E_{Local}, \mu, \lambda,$ `optimizer, Score fn, basis number, N`

Server:

```
epoch ← 0
while epoch ≤ E_Global do
    Wait while collecting gradients ∇w_k^T from client k
    if ||∇w_k^T|| = N then
        ∇w ← (1/N) Σ_{k=1}^{N} ∇w_k^T
    end if
    Send ∇w to every client
    epoch ← epoch + 1
end while
```

Client:

```
∇w_k^T ← 0
Apply aggregated gradient ∇w from Server
for epoch=1 to E_local do
    Calculate local loss F_k from Equation 3.9
    ∇w_k ← Calculated gradient from the optimizer
    ∇w_k^T ← ∇w_k^T + ∇w_k
    Apply gradient ∇w_k
end for
Send w_k^T (local gradients) to Server
```

Output: $w \leftarrow \frac{1}{||N||} \sum_{k=1}^{N} w_k$

3.3 DP-KE-GCN

In addition to the federated protocol, my work implemented DP-SGD for the KE-GCN to create a differential private graph neural network. To do so, I utilized TensorFlow's Privacy modules [1] to wrap the optimizer. This optimizer wrapper modifies the original stochastic gradient descent to be differentially private according to the DP-SGD paper [2]. The wrapper utilizes 3 hyperparameters: a norm clipping value, noise multiplier, and micro-batch size. During training, the wrapper's implementation will clip the gradients such the maximum L2 norm of each gradient computed per micro-batch is smaller or equal to this value. By clipping the gradients, I directly bound the possible influence a single training data point can have on the machine

learning model. In addition to clipping the gradients, the wrapper will need to add random noise to the clipped gradients to achieve privacy for the data. The amount of noise is determined by the noise multiplier hyperparameter which is defined as the ratio of the standard deviation to the clipping norm. The last factor is the overall size of a single example which is defined by the micro-batch hyperparameter. For a given batch, the batch is split into micro-batches which is the smallest unit used for clipping the gradient. In my case, I define the smallest unit as a node, so the micro-batch is size 1.

Chapter 4

Evaluation

In this chapter, I provide an assessment of my work through multiple experiments. First, I describe the general setup used for all experiments such as the datasets, general framework, and testing infrastructure. The rest of the sections reveal the experimental results of my work while providing only the necessary amount of discussion required for understanding. The main analysis and discussion on the results is in the final chapter. I categorize my experiments in 3 main sections: basis decomposition in the centralized setting, Fed-KE-GCN, and DP-KE-GCN.

4.1 General Experimental Setup

In this section, I describe the experimental setup I used for basis decomposition in the centralized setting, Fed-KE-GCN, and DP-KE-GCN. For the computation infrastructure, all my experiments were run using the MIT Supercloud system [30], specifically with Nvidia Volta V100 accelerators and Intel Xeon Gold 6248 processors.

For all my experiments, I utilized four graph datasets: AIFB [3], Wordnet [11], Freebase [41], and Amsterdam Museum [31]. AIFB is a knowledge graph that models the organizational structure of the research community at the University of Karlsruhe. Within this dataset, entities can have additional metadata attached that pertains to whether the entity is a Person, Publication, Event, Organization, Topic, Project or Other. This is important as I take advantage of these meta tags to split the data

Table 4.1: Logistical Information about each dataset.

Dataset	**# Entities**	**# Relations**	**# Edges**	**# Labeled**	**Classes**
AIFB	8,295	45	29,043	176	4
WN18	40,943	18	151,442	31,943	24
FB15K	14,904	1,341	592,213	13,445	50
AM	1,666,764	133	5,988,321	1,000	11

for federated learning. The Wordnet (WN18) dataset is a knowledge graph derived from Wordnet 3.0 [23] in the form (synset, relation, synset) where a synset is a set of synonyms. In the WN18 knowledge graph, each entity is a synset and each relation is the lexical relation between two synsets. Some examples of lexical relations between two synsets are hypernym, hyponym, and meronym. A triple example from WN18 would be (dog, hyponym, pit bull) where the word dog is a more general word for pit bull. The next dataset I used in my experiments is Freebase (FB15k) which is a knowledge graph that represents real-world objects. This can be viewed as general facts about real-world things like people, events, and abstract concepts. Lastly, the Amsterdam Museum (AM) dataset is a knowledge graph that models cultural objects related to the history of Amsterdam as described by the Amsterdam Museum. For my experiments, I only use this dataset in the centralized setting due to its size. Logistical information about each dataset, such as the number of entities, is depicted in Table 4.1.

For all experiments, I first split the datasets into train, valid, and test sets. To do so, I follow the same method mentioned in the KE-GCN paper [51] by randomly splitting the data with a ratio of 10%, 10%, and 80% for train, valid, and test respectively.

For all federated setting experiments, I set the total number of clients to 3 as I ran into excessive memory usage with more clients. I will further discuss this excessive memory usage within the discussion section. For splitting the AIFB data between clients for the federated setting, I first randomly shuffle the labeled nodes in the training data and evenly split it between all clients. For each client's test set, I simply

store the entire AIFB test set with no alterations. For the remaining unlabeled data, I randomly select 6 types (from Person, Publication, Event, Organization, Topic, Project, and Other). For each type selected, I sample nodes such that $||X_{sampled}|| \sim U(0, ||X_{type}||)$ from the complete dataset for each client. With all the unlabeled nodes sampled with this method and the evenly split training nodes, I only add an edge between two nodes for the client's base if an edge exists in the full knowledge graph.

In order to split the FB15k and WN18 dataset in the federated setting, I follow a similar pattern used for AIFB. For each dataset, I first randomly shuffle the training data, and then I evenly split it among the clients. Next, I store the entire test set to all clients without making any changes. The difference in setup occurs for the remaining unlabeled nodes as I don't have metadata about the types of the nodes. Since there are no given distinct types, I split the unlabeled nodes into 7 categories such that the sum of the nodes in all these categories add up to the total number of unlabeled nodes. In addition, nodes are assigned to these categories in an uneven way in order to provide a more realistic environment for the federated setting. With these 7 categories, I follow the same procedure as AIFB where I select 6 categories from the 7 total. For each category selected, I sample nodes such that $||X_{sampled}|| \sim U(0, ||X_{category}||)$ from the complete dataset for each client. With all the unlabeled nodes sampled with this method and the evenly split training nodes, I add an edge for all edges in the full knowledge graph if both nodes exist in the client's base.

4.2 Centralized Setting Results

The first experiment I ran was testing my basis decomposition implementation with the KE-GCN model in a centralized setting. To do so, I first ran an unchanged KE-GCN in the centralized setting for the node classification task on the AM, WN18, AIFB and FB15K datasets. The results for AM and WN18 are taken directly from KE-GCN's paper as noted in the table [51]. After performing a standard hyperparameter search, I set the model's hidden dimension size to 32, number of layers to 2, optimizer to Adam [17], activation function to ReLU, α to 0.5, learning rate to 0.01,

and the scoring function to TransE. In addition, all centralized experiments were ran in with full batch training with 500 epochs. After running the regular KE-GCN model, I ran my implementation of KE-GCN model with basis decomposition with the same parameters as above and basis size set to 50 (determined by a search over 10, 30, and 50). Lastly, I utilized the KE-GCN model's generalizability to run R-GCN with basis decomposition in the centralized setting as well. All of these results are depicted in Table 4.2. These results show that R-GCN with basis decomposition performed worse than KE-GCN with basis decomposition. This is most likely due to the additionally learned relation embedding that KE-GCN uses since the original KE-GCN outperforms R-GCN on these datasets. For example, the original R-GCN implementation got 89.3% and 55.1% accuracy on the AM and WN18 datasets respectively [51].

Model	AM	WN18	AIFB	FB15K
KE-GCN	91.2 ± 0.2*	57.8 ± 0.5*	97.2	75.5
R-GCN with Basis Decomp	87.8	54.1	94.3	67.8
KE-GCN with Basis Decomp	88.9	56.5	94.5	71.8

Table 4.2: Node classification across multiple datasets. The * indicates it was taken from the KE-GCN paper directly.

4.3 Fed-KE-GCN Results

My second set of experiments are aimed to test the Fed-KE-GCN protocol, which utilizes basis decomposition and optimal transport. These experiments can be broken down into two tasks: node classification and link prediction.

Across all node classification experiments, I utilized full batch training, 5 local epochs, 20 global epochs, and the same set of KE-GCN parameters as used in the centralized setting. Specifically, I set the model's hidden dimension size to 32, number of layers to 2, Adam optimizer [17], ReLU activation function, and α (weight of entity convolution update) to 0.5. However, the only change is that the best learning rate was 0.1. With these standard parameters set, I ran every node classification experiment with three different scoring functions: TransE, DistMult, and RotatE. In

addition, I conducted these node classification experiments in two settings: Single and Fed. Both settings use 3 total clients and each client's data is derived from the split data procedure detailed in the evaluation setup. The Single setting involves running the experiment without sharing gradients with the server and without optimal transport in the client's loss function. In other words, each client is only training locally on their data with no external information or help from other clients. The Fed setting is the full federated protocol with optimal transport and a server aggregating client gradients. To determine the best-performing basis size, I ran each node classification experiment with basis sizes 10, 30, and 50. Each of the following tables show the best-performing results from these parameters. For the first node classification experiment, I ran the Fed-KE-GCN protocol on the AIFB dataset with a basis size of 10. These results are shown in Table 4.3.

As shown in the table, the Fed setting performed better than the Single setting by a significant margin. My hypothesis for this leap of performance is a combination of the dataset properties and the potential to get stuck at a local optimum in the Single setting. The AIFB dataset as noted in Table 4.1 is small and only contains 4 classes. In this case, the stochastic approach I used to split AIFB for each client doesn't guarantee a final connected graph or all classes to appear in the training set. As a result, the training set can be unbalanced or lack significant neighborhood information for the GCN to take advantage of. This can ultimately lead clients to get stuck at a local optimum. A simple example would be a client always guessing a single class as it is appears the most in the training set. With the Fed setting, the introduction of optimal transport in the loss function introduces a new component that makes it harder to be stuck at a local optimum as all other client's basis will be changing. This will force the gradient for the stuck client to change even if it is at a local optimum for it's own training data.

The next experiment for the node classification task was on the WN18 dataset. The best performing basis size was 30 for the TransE and RotatE scoring functions, however, DistMult's best performing basis was size 10. The accuracies for Fed-KE-GCN on the WN18 dataset are shown in Table 4.4.

KGE	Setting	Client 1	Client 2	Client 3
		Accuracy	Accuracy	Accuracy
TransE	Single	44.444	52.778	69.444
	Fed	44.444	75.000	77.778
DistMult	Single	41.667	47.222	61.111
	Fed	47.222	69.444	63.889
RotatE	Single	55.556	55.556	63.889
	Fed	55.556	58.333	63.889

Table 4.3: Node Classification Evaluation on AIFB dataset.

KGE	Setting	Client 1	Client 2	Client 3
		Accuracy	Accuracy	Accuracy
TransE	Single	25.776	18.286	13.739
	Fed	25.204	17.875	14.142
DistMult	Single	25.878	19.753	14.357
	Fed	26.339	20.822	13.739
RotatE	Single	22.364	15.821	11.9
	Fed	22.485	16.881	13.273

Table 4.4: Node Classification Evaluation on WN18 dataset.

The last node classification experiment for my Fed-KE-GCN protocol is on the FB15k dataset. Table 4.5 shows my results with a basis size of 30 for all scoring functions.

KGE	Setting	Client 1	Client 2	Client 3
		Accuracy	Accuracy	Accuracy
TransE	Single	39.850	39.300	41.240
	Fed	40.790	39.440	44.780
DistMult	Single	37.320	36.430	44.570
	Fed	36.720	38.040	30.070
RotatE	Single	41.240	39.690	44.570
	Fed	41.440	39.780	44.730

Table 4.5: Node Classification Evaluation on FB15k dataset.

For the link prediction task, my experiments ran into unsolvable memory issues which is one of the most severe limitations of my protocol. As mentioned before, these memory issues will be discussed in the next chapter.

4.4 DP-KE-GCN Results

The last set of experiments I ran was the implementation of DP-SGD with the KE-GCN model. The differentially private KE-GCN was tested with node classification on FB15k. These experiments used the same parameters as KE-GCN in the centralized setting. As mentioned in the method, there are three new hyperparameters: gradient L2 norm clipping value, noise multiplier, and micro-batch. For this experiment, all runs used a micro-batch size of 1 as this should provide the best possible accuracy at the expense of longer computation time. In addition, I set δ to $\frac{1}{10{,}000}$ for all runs as the standard is to set this external risk factor to 1 divided by the amount of data points. I ran a standard hyperparameter search over the gradient L2 norm clip with values (0.0001, 0.00001, 0.000001), noise multiplier with values (0.001, 0.1, 0.25, 0.35, 0.5, 0.65, 0.8, 1.0, 2.0), and learning rate of values (0.001, 0.01, 0.05, 0.1, 0.15, 0.25). In order to properly analyze this experiment, let me define a subset of these hyperparameters as an unchanged set of norm clipping values and learning rate, and the entire range of noise multiplier values from 0.001 to 2.0. This ultimately varies epsilon across the changing noise multiplier, but enables us to analyze the accuracy change. In general, these subsets should follow an expected trend where a decrease in privacy (larger epsilon value) leads to increased utility (higher accuracy).

For my results, I chose the best-performing subset which is the best norm clipping value and learning rate combination. In this case, it was a norm value of 0.0001 and a learning rate of 0.25. These results are shown in Table 4.6, and it is evident that the more noise added leads to a smaller epsilon value as expected. In order to better see the expected privacy-utility trend, I plotted the different calculated epsilon values over the x-axis, and test accuracy from the node classification on FB15k over the y-axis. Figure 4-1 shows this plot and depicts the expected privacy-utility trend. Specifically, as epsilon increases over the x-axis, the overall utility of the model approaches its non-differential private performance.

Noise Mult.	Epsilon	Test Acc
0.1	12546	73.16
0.25	2046	65.78
0.35	1066	58.3
0.5	542	46.22
0.65	341	38.1
0.8	241	30.6
1.0	171	24.18
2.0	59	16.36

Table 4.6: DP-KE-GCN results on FB15k with norm clipping value of 0.0001 and learning rate of 0.25.

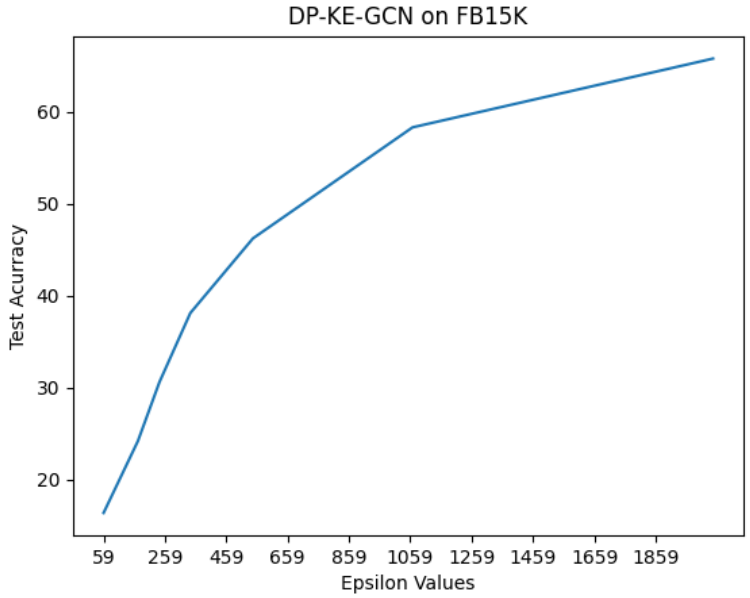

Figure 4-1: General privacy-utility trend for node classification on FB15k.

Chapter 5

Discussion and Conclusion

This section provides specific analysis, interpretation, and discussion of the main experimental results along with a clear outline of the limitations. The analysis is structured into the two approaches: basis decomposition with optimal transport and DP-SGD. In addition to diving into potential causes for the results, I provide future work and possible solutions to overcoming these limitations faced. Lastly, I give final remarks about my work and the broader problem of knowledge graph completeness.

5.1 Basis Decomposition and Optimal Transport

Both the centralized setting and Fed-KE-GCN results reveal two main insights. Firstly, basis decomposition with KE-GCN leads to marginal increases in performance under certain conditions. Secondly, KE-GCN's computational load with basis decomposition and optimal transport is a serious limitation.

To understand the first insight, I will start by analyzing the centralized setting results first. Ultimately, basis decomposition is a form of regularization that limits the total number of parameters. The regularization occurs at the relation level as the learned basis is shared between all relation weight matrices and aims to prevent overfitting to any specific relation. Like any regularizer for machine learning, the prevention of overfitting can also simply decrease overall performance for multiple reasons such as the regularization has too much of an effect. In the case of the cen-

tralized version in Table 4.2, the KE-GCN model with basis decomposition performs slightly worse by 3 to 4 percent than the normal KE-GCN results. This is most likely due to the overall regularization having too large of an effect and underfitting the data. Furthermore, this hypothesis is also backed up by the fact that the KE-GCN model uses full batch training and a significant amount of epochs which will attempt to fit as much to the data as possible. In addition, the KE-GCN with basis decomposition performs marginally better than the R-GCN with basis decomposition which is a good indication that utilizing basis decomposition with KE-GCN might perform better in different environments. Beyond the centralized setting results, the federated learning results also provide more insight into the marginal increases in performance that basis decomposition and optimal transport can offer. All node classification tasks reveal a general trend of very slight (less than 1%) increases in performance, however, it is inconsistent. There are also many cases where performance is slightly decreased, (and even large decreases in performance with DistMult as the embedding) from the single setting where clients only train locally. Thus, it appears the potential for basis decomposition and optimal transport is not the best solution for extending KE-GCN into the federated setting due to its inconsistency and only marginal increases in performance.

The second insight from the results is also the most important. There is a serious computational requirement in order to run KE-GCN with both basis decomposition and optimal transport that leads to numerous memory issues. These memory issues prevented me from running more than 3 clients and completing the link prediction task. My hypothesis for the cause of this limitation is the extensive amount of tensors, information, and computation that is needed for each client. Specifically, I will break down all the necessary components with my protocol and how TensorFlow ultimately optimizes and computes these components. For Fed-KE-GCN, the model utilizes full batch training as mentioned before so every client will have some potentially non-negligible amount of training data stored. Since optimal transport is computed between a client's basis for a given layer and every corresponding basis for all clients, this requires storing every basis for every layer for every client. These basis need to

be stored locally such that the client can access it directly in the loss function. In addition, the basis is a decent-sized tensor where its dimension is dependent on the hidden dimension size for the layer and the total basis size. In my protocol, I used a hidden dimension size of 32, 2 layers, and a basis size of 30. So each client will be using their own (32,32, 30) total tensor size for each basis as well as storing all other clients' basis. In addition to storage, these optimal transport calculation requires making a cost matrix that details the cost to move a value between every index of two tensors. So the cost matrix will compare every index of a (32,32) tensor with another (32,32) tensor creating a (1024, 1024) cost matrix. The total number of cost matrices grows with every basis as the cost matrix needs to be computed individually for each pair. In addition to this cost matrix, there are multiple other calculations occurring within the sinkhorn algorithm in order to get the sinkhorn distance. As a result, it's easy to see that the total required memory is significant and grows rather quickly considering that other memory-intensive tasks such as computing and storing gradients also need to be done. The question arises about potentially computing the cost matrix and then immediately deleting it from memory in order to save as much space for memory as possible, however, the current implementation of KE-GCN is in TensorFlow. At a low level, TensorFlow creates a static computation graph where it initializes all the necessary tensors and information needed to train the model. This is done to for efficiency reasons as TensorFlow can do many behind-the-scenes optimizations so the model can run fast. As a result, this static computation graph immediately requires a substantial amount of memory in order to account for all the above storage requirements. There are two immediate solutions to see if this is the main memory-causing issue. The first is to convert the model to PyTorch which uses a dynamic computation graph so this immediate need to account for all memory isn't necessary as the computation graph is created along the process and as needed. In addition, TensorFlow has a way to utilize a dynamic computation graph with eager execution. However, the current version of TensorFlow with the model is outdated and requires converting to a more updated version in order to fully utilize the eager execution method. Beyond infrastructure changes to the model itself, a potential so-

lution is to distribute the memory burden across multiple nodes in a computer cluster. Specifically, the federated setting conveniently enables the possibility of running individual clients on a node and utilizing message passing between nodes in the cluster to communicate new basis values for example.

5.2 DP-KE-GCN

The experimental results of DP-SGD in the KE-GCN model correctly show the expected privacy-utility trend. However, the epsilon values are too large to be considered usable for real-world tasks as the privacy guarantee is too small for real data. The smallest epsilon I was able to create with my work was 59 as the utility of the model becomes too low. In most applications, a good epsilon value is in the single digits. There are several potential reasons why achieving differential privacy in the KE-GCN model with DP-SGD led to unsatisfactory results.

The first potential reason is centered on the mechanics of DP-SGD and the nature of graph data. DP-SGD takes advantage of a single "example" to ensure the model is differential private and maximizes utility. This single example can be thought of as a single training data point (i.e. an image) for most machine learning applications. With a single example, DP-SGD limits the total effect that example can have on the parameters by clipping its gradient using the norm clipping value. By bounding the total effect the data point can have, the algorithm can add random noise proportional to this max bound and ultimately ensure a certain level of privacy. For graph data and specifically Graph Convolutional Networks, there is no clear concept of a single example in training. In my work, the nodes are considered single examples, however, these nodes are connected and the KE-GCN model uses neighborhood information when training so the limit of a single node isn't bounded. As a result, the ability to clip the gradient accurately and precisely to maximize privacy and utility is worsened.

The second potential reason for the DP-KE-GCN results is based on the iterative aggregation and update scheme with KE-GCN. As DP-SGD adds noise to the gradients based on the calculations for a single example, this noise is propagated to

the model's parameters as desired. During training, the model's parameters are used to calculate node representations by aggregating all neighboring nodes through hops. This leads to the injected noise being used for multiple calculations due to the numerous amount of hops. The total amount of noise added is not calculated or tuned for this framework. As a result, the noise's impact on accuracy during inference can be a lot more substantial for the KE-GCN model than other machine learning models.

Both of the above potential reasons can be solved by a well-defined single example within graph data. Specifically, a well-defined single example would enable a much more precise clipping of the gradient, and more importantly, a more precise amount of noise added to limit the effect on utility while also ensuring the correct level of privacy. An example can be seen in the graph classification task where the data used is a set of entire graphs [25]. In this case, the single example is an entire graph which enables DP-SGD to be directly applied to the model.

The last potential reason for large epsilon values for DP-KE-GCN is the combination of full batch training and having a lot of epochs. By definition, full batch training uses the entire training set to update the model for each epoch. For my model, this means all training nodes are utilized during training for every epoch out of the total 500 epochs. This has a significant effect on the privacy guarantees for the model. I will explain using an intuitive approach as well as provide examples with data. From an intuitive perspective, the goal of differential privacy is to ensure uncertainty about whether an individual from the data was used during training from just the output of the model. In full batch training, this individual is used every time for a total of 500 times which means its impact on the output of the model will most likely be significant. In other models that utilize mini-batch training, uncertainty about whether the individual is used is generated from the fact that an individual may not be sampled for that batch of training. This induces randomness as well as directly making the gradients learned an approximation of the true gradient which leads to more uncertainty within the model's output. Table 5.1 shows that my model could achieve immediately lower epsilons values if mini-batch training was used for batch sizes that are $\frac{1}{4}$ the size of the total training set. The lowest epsilon is drastically

Full batch ϵ	**Mini-batch ϵ**
12546	6400
2046	483
1066	232
542	104
341	61
241	41
171	27
59	9

Table 5.1: A table comparing the calculated epsilons from the DP-KE-GCN results and potential epsilon values if a batch size of 1/4 was used.

reduced from 59 to 9. It is important to note that these values are still high, but the use of mini-batches can contribute to a better potential solution. Furthermore, creating batches for graph data is a nontrivial task. A potential approach is to generate normalized cuts such that each subgraph is a single batch and is roughly the same size. In addition, these normalized cuts would guarantee every node is included in exactly one subgraph which is desired for mini-batch training. However, this approach could lead to lost neighborhood information as edges must be removed.

5.3 Final Remarks

The knowledge graph completion problem remains a relevant and prominent issue where effective solutions can greatly enhance many real world applications. The significance of this problem comes from the utility knowledge graphs offer with their ability to model human knowledge and interlinked data. Given the sensitive and distributed nature of knowledge graphs in different institutions, my work focuses on privacy by exploring a federated learning and differential privacy approach.

My work introduces Fed-KE-GCN, which is a federated learning protocol with the generalizable KE-GCN model. By using a generalizable model, my protocol empowers clients to use the most appropriate model and scoring function for their specific task. Furthermore, the experiments for Fed-KE-GCN provide great insight into the practical limitations surrounding the use of basis decomposition and optimal

transport for a federated setting.

In addition, my work also empirically shows that the common differential privacy approach of DP-SGD for machine learning models is ineffective due to the nature of graph data, the framework for GCNs, and the use of full batch training in this case. As a result, my work highlights the need for a more advanced solution in order to implement an effective differentially private Graph Convolutional Network.

Lastly, my work lays out potential solutions and future work that can be completed to overcome the limitations of basis decomposition, optimal transport, and DP-SGD with the KE-GCN. In conclusion, my work contributes to the broader knowledge completion problem by introducing an approach that highlights the need for flexibility as well as privacy.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, oct 2016.

[3] Stephan Bloehdorn and York Sure. Kernel methods for mining instance data in ontologies. In Karl Aberer, Key-Sun Choi, Natasha Noy, Dean Allemang, Kyung-Il Lee, Lyndon Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors, *The Semantic Web*, pages 58–71, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[4] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[5] Mingyang Chen, Wen Zhang, Zhen Yao, Xiangnan Chen, Mengxiao Ding, Fei Huang, and Huajun Chen. Meta-learning based knowledge extrapolation for knowledge graphs in the federated setting, 2022.

[6] Mingyang Chen, Wen Zhang, Zonggang Yuan, Yantao Jia, and Huajun Chen. Fede: Embedding knowledge graphs in federated setting. *CoRR*, abs/2010.12882, 2020.

[7] Yongxin Chen, Tryphon T. Georgiou, Lipeng Ning, and Allen Tannenbaum. Matricial wasserstein-1 distance, 2017.

[8] Shivani Choudhary, Tarun Luthra, Ashima Mittal, and Rajat Singh. A survey of knowledge graph embedding and their applications, 2021.

[9] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transportation distances, 2013.

[10] Ameya Daigavane, Gagan Madan, Aditya Sinha, Abhradeep Guha Thakurta, Gaurav Aggarwal, and Prateek Jain. Node-level differentially private graph neural networks, 2022.

[11] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), Apr. 2018.

[12] 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*, KDD '14, page 601–610, New York, NY, USA, 2014. Association for Computing Machinery.

[13] 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*, KDD '14, page 601–610, New York, NY, USA, 2014. Association for Computing Machinery.

[14] Cynthia Dwork. Differential privacy: A survey of results. In Manindra Agrawal, Dingzhu Du, Zhenhua Duan, and Angsheng Li, editors, *Theory and Applications of Models of Computation*, pages 1–19, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[15] Mikhail Galkin, Max Berrendorf, and Charles Tapley Hoyt. An open challenge for inductive link prediction on knowledge graphs, 2022.

[16] 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, Beijing, China, July 2015. Association for Computational Linguistics.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[18] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[19] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

[20] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Yuan Li, Xu Liu, and Bingsheng He. A survey on federated learning systems: Vision, hype and reality for data privacy and protection. *IEEE Transactions on Knowledge and Data Engineering*, 35(4):3347–3366, apr 2023.

[21] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks, 2020.

[22] Yilun Lin, Chaochao Chen, Cen Chen, and Li Wang. Improving federated relational data modeling via basis alignment and weight penalty. *CoRR*, abs/2011.11369, 2020.

[23] George A. Miller. WordNet: A lexical database for English. In *Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, February 23-26, 1992*, 1992.

[24] Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*. IEEE, aug 2017.

[25] Tamara T. Mueller, Johannes C. Paetzold, Chinmay Prabhakar, Dmitrii Usynin, Daniel Rueckert, and Georgios Kaissis. Differentially private graph classification with gnns, 2022.

[26] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, jan 2016.

[27] Hao Peng, Haoran Li, Yangqiu Song, Vincent W. Zheng, and Jianxin Li. Federated knowledge graphs embedding. *CoRR*, abs/2105.07615, 2021.

[28] B. Wang Q. Wang, Z. Mao and L. Gu. Knowledge graph embedding: A survey of approaches and applications. *IEEE Trans. Knowl. Data Eng.*, 29(12):2724–2743, 2017.

[29] Stephan Rabanser, Oleksandr Shchur, and Stephan Günnemann. Introduction to tensor decompositions and their applications in machine learning, 2017.

[30] Albert Reuther, Jeremy Kepner, Chansup Byun, Siddharth Samsi, William Arcand, David Bestor, Bill Bergeron, Vijay Gadepally, Michael Houle, Matthew Hubbell, Michael Jones, Anna Klein, Lauren Milechin, Julia Mullen, Andrew Prout, Antonio Rosa, Charles Yee, and Peter Michaleas. Interactive supercomputing on 40,000 cores for machine learning and data analysis. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2018.

[31] Petar Ristoski, Gerben Klaas Dirk de Vries, and Heiko Paulheim. A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web. In *The Semantic Web – ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part II*, page 186–194, Berlin, Heidelberg, 2016. Springer-Verlag.

[32] Andrea Rossi, Donatella Firmani, Antonio Matinata, Paolo Merialdo, and Denilson Barbosa. Knowledge graph embedding for link prediction: A comparative analysis. *CoRR*, abs/2002.00819, 2020.

[33] Sina Sajadmanesh, Ali Shahin Shamsabadi, Aurélien Bellet, and Daniel Gatica-Perez. Gap: Differentially private graph neural networks with aggregation perturbation, 2022.

[34] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks, 2017.

[35] Chao Shang, Yun Tang, Jing Huang, Jinbo Bi, Xiaodong He, and Bowen Zhou. End-to-end structure-aware convolutional networks for knowledge base completion. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):3060–3067, Jul. 2019.

[36] Jonathon Shlens. Notes on kullback-leibler divergence and likelihood, 2014.

[37] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[38] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. *CoRR*, abs/1902.10197, 2019.

[39] Thiviyan Thanapalasingam, Lucas van Berkel, Peter Bloem, and Paul Groth. Relational graph convolutional networks: a closer look. *PeerJ Computer Science*, 8:e1073, nov 2022.

[40] Carlo Tomasi. The earth mover's distance, multi-dimensional scaling, and color-based image retrieval. 1997.

[41] 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, Beijing, China, July 2015. Association for Computational Linguistics.

[42] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha P. Talukdar. Composition-based multi-relational graph convolutional networks. *CoRR*, abs/1911.03082, 2019.

[43] Cédric Villani. *Optimal transport – Old and new*, volume 338, pages xxii+973. 01 2008.

[44] Meihong Wang, Linling Qiu, and Xiaoli Wang. A survey on knowledge graph embeddings for link prediction. *Symmetry*, 13(3), 2021.

[45] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. *Proceedings of the AAAI Conference on Artificial Intelligence*, 28(1), Jun. 2014.

[46] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, jan 2021.

[47] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[48] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.*, 10(2), jan 2019.

[49] Zi Ye, Yogan Jaya Kumar, Goh Ong Sing, Fengyan Song, and Junsong Wang. A comprehensive survey of graph neural networks for knowledge graphs. *IEEE Access*, 10:75729–75741, 2022.

[50] Jinsung Yoon, James Jordon, and Mihaela van der Schaar. PATE-GAN: Generating synthetic data with differential privacy guarantees. In *International Conference on Learning Representations*, 2019.

[51] Donghan Yu, Yiming Yang, Ruohong Zhang, and Yuexin Wu. Generalized multi-relational graph convolution network. *CoRR*, abs/2006.07331, 2020.

[52] He Zhang, Bang Wu, Xingliang Yuan, Shirui Pan, Hanghang Tong, and Jian Pei. Trustworthy graph neural networks: Aspects, methods and trends, 2022.

[53] Kai Zhang, Yu Wang, Hongyi Wang, Lifu Huang, Carl Yang, Xun Chen, and Lichao Sun. Efficient federated learning on knowledge graphs via privacy-preserving relation embedding aggregation, 2022.

[54] Shuai Zhang, Yi Tay, Lina Yao, and Qi Liu. Quaternion knowledge graph embedding. *CoRR*, abs/1904.10281, 2019.

[55] Zaixi Zhang, Qi Liu, Zhenya Huang, Hao Wang, Chengqiang Lu, Chuanren Liu, and Enhong Chen. Graphmi: Extracting private graph data from graph neural networks, 2021.

[56] Zhanqiu Zhang, Jie Wang, Jieping Ye, and Feng Wu. Rethinking graph convolutional networks in knowledge graph completion. In *Proceedings of the ACM Web Conference 2022*, WWW '22, page 798–807, New York, NY, USA, 2022. Association for Computing Machinery.

[57] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *CoRR*, abs/1806.00582, 2018.