CS224w Project Report

Toward Embedding a Web-scale Knowledge Graph

Daniel Lee[1] **and William Caruso**[1]
{daniel.lee, wcaruso} @stanford.edu

Abstract

Knowledge graphs have been growing in popularity as more advanced graph neural networks improve the quality of embeddings. These embeddings are used to encode information about relations between entities and are most commonly evaluated on link-prediction tasks. The quality of these embeddings is dependent not only on the graph neural network (GNN) architecture used, but also the structure of the underlying knowledge graph. The research community has settled on standard datasets such as FB15K-237, WN18RR, NELL-995, etc. which all feature a distinct graph structure. We introduce a new real-world dataset, DKG (Diffbot Knowledge Graph), scraped from a web-scale knowledge graph with 10 billion entities and 1 trillion relations aggregated by scraping the web. We analyze and compare the link-prediction performance of various embeddings (TransE and RotatE) on several subgraphs of DKG to better understand the relation between graph structure and embedding quality. Additionally, we demonstrate improved link-prediction performance using a state-of-the-art knowledge graph GAT (Graph Attention Network). We hope that this work will spur further research efforts toward generating useful embeddings for large, real-world knowledge graphs.

1 Introduction

Knowledge Graphs (KGs) represent a collection of facts as a directed graph where nodes are entities and edges are relations between entities. Each fact is represented as a triple *(Head, Relation, Tail)* where *Head* and *Tail* are entities and *Relation* is a valid relation between them. For example, *(Tesla Inc, CEO, Elon Musk)* is a triple in the Diffbot Knowledge Graph (DKG). Knowledge graphs are considered heterogeneous graphs because entities and relations can have different types. They have a wide variety of applications across domains such as question answering, logical querying, and semantic understanding.

Graph embeddings are low-dimensional feature representations of entities and relations in the knowledge graph. Embeddings are used to encode heterogenous, factual information in a knowledge graph and can be extremely useful for some of the downstream tasks mentioned above. The most common method to evaluate the quality of knowledge graph embeddings is link-prediction, the task of filling in missing relations between entities. Today, much research has focused on developing new graph neural network (GNN) architectures and training procedures to produce high quality embeddings for standard knowledge graph datasets such as FB15K-237, WN18RR, NELL-995, etc. However, there has been much less effort toward understanding the relation between graph properties and embedding quality.

Today's standard knowledge graph datasets are constructed by hand and curated by humans - their structure reflects this. In contrast, the World Wide Web is a rich source of many types of information and is often referred to as the largest database in the world. However, the failure of the Semantic Web, where human authors are responsible for annotating their own Webpages with metadata, has proven that structuring such a massive (web-scale) amount of knowledge into a more useful form is a super-human task. In recent years, scientists have leveraged advances in natural language processing to help automatically organize knowledge on the Web into structured knowledge-graphs. Their autonomous creation allows them to be generated frequently, so the information they hold is always up-to-date. However, these graphs constructed by web-scraping bots, are large-scale, often contain noisy data, and have very different structure from standard knowledge graph datasets.

Knowledge graphs constructed on data pulled from the Web already power many services used by millions of people around the world everyday, including search and recommendation engines. Often these graphs are incomplete and only contain partial information in the form of missing relations between entities. As a result, knowledge completion tasks (analogous to link prediction) have become more popular in recent years, but rely on high quality feature embeddings.

[1] Equal Contribution

We believe that the research community should focus on improving embedding quality for these web-scale, real-world knowledge graphs as good embeddings will empower further work to be done with these networks. In this project, we collaborated with Diffbot, a Stanford founded startup, which provides a real-world web-scale knowledge graph assembled by scraping data from the Web. We first scraped a large subgraph ($\sim$ 600k entities) from the total graph which has over 10 billion entities. We pruned our subgraph down into several smaller subgraphs. For each pruned graph, we use the TransE and RotatE models to generate a set of initial embeddings and evaluate the results using a link prediction task. We compare the embedding quality with subgraph structure. Then, building upon recent work, we use a graph attention network (GAT) to further enrich the embedding quality. We compare our results across various size subgraphs and finally make recommendations for further work on improving the structure of the Diffbot Knowledge Graph.

In this paper, we will discuss our three main contributions

1. Curation and characterization of a new real-world web-scale dataset, Diffbot Knowledge Graph (DKG)
2. Comparison of embedding (TransE & RotatE) quality for various subgraphs of DKG
3. Attention-based embeddings for DKG with very high link-prediction performance

We hope that this work will spur further research efforts toward generating useful embeddings for large, real-world knowledge graphs.

2 Related Work

First we discuss two simple embedding frameworks, TransE and RotatE. Then we describe Graph Attention Networks (GATs) and focus on a recent GAT designed to model multi-relational data in knowledge graphs.

2.1 TransE Embeddings [Bordes et al., 2013]

The TransE model represents relationships as translations in the embedding space such that for all triples (h, r, t), it optimizes over objective $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$ where embeddings $\mathbf{h}, \mathbf{r}, \mathbf{t} \in \mathbb{R}^d$ with d embedding dimension. In other words, the score is

$$-||\mathbf{h} + \mathbf{r} - \mathbf{t}|| \tag{1}$$

2.2 RotatE Embeddings [Sun et al., 2019]

In contrast, the RotatE model optimizes the score

$$-||\mathbf{h} \circ \mathbf{r} - \mathbf{t}||^2 \tag{2}$$

where $\circ$ is the Hadamard product. Intuitively, RotatE defines relations as rotations from the head to tail entity in complex vector space. This method is capable of modelling a wider range of relation patterns such as symmetry, inversion, and composition.

2.3 Graph Attention Networks [Veličković et al., 2017]

Real world networks are often very large and complex leading to unwanted noise in the inherent structure. As a solution, attention is used to allow the model to focus task-relevant parts of the graph. Attention mechanisms for knowledge graphs usually involve learning a set of attention weights as an addendum to the node/edge embedding problem where we additionally learn a second function, $f' : V \times V \rightarrow [0, 1]$ to assign "attention weights" to the target elements [Lee et al., 2018]. The goal of attention-based embeddings is to assign more weight to more relevant inputs, where relevancy is defined according to a learned attention mechanism. For node i, attention weight α_{ij} is the importance assigned to node j

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) \tag{3}$$

where $e_{i,j}$ is the attention coefficient between node i and j. This attention coefficient is the result of a learned similarity function acting on the embeddings for node i and node j.

Graph Attention Networks (GATs) extends Graph Convolutional Networks (GCN) by introducing these attention weights between neighboring nodes as the primary aggregation mechanism [Veličković et al., 2017]. This manifests as a single self-attention layer in which nodes attend to their neighborhoods' features. Stacking these layers allows the network to model more complex interactions between embeddings for distant, yet similar nodes. Velickovic extends this basic attention layer by using multi-head attention, where multiple independent attention mechanisms are used and their features are concatenated at each intermediate stage, and are averaged together in the final stage. Intuitively, we can think of each head in this multi-head attention mechanism as focusing on a different representation subspace, so each head learns a different metric for similarity between features.

2.4 KBAT: Attention Based Embeddings for Knowledge Graphs [Nathani et al., 2019]

Recently, Nathani et al. proposes KBAT, an extension of GAT to heterogeneous graphs, such as knowledge graphs. He does this by parameterizing the attention mechanism with relation embeddings. Consider Algorithm 1, where you can see the attention mechanism acts on the concatenation of head, tail, and relation embeddings (rather than just head and tail as in standard GAT).

Algorithm 1 Single-head KBAT layer (Nathani 2019)

Input: $H \in \mathbb{R}^{|\mathrm{N}|\mathrm{xF}}$, $G \in \mathbb{R}^{|\mathrm{E}|\mathrm{xP}}$
Returns: $H' \in \mathbb{R}^{|\mathrm{N}|\mathrm{xF}'}$, $G' \in \mathbb{R}^{|\mathrm{E}|\mathrm{xP}'}$
Parameters: F', P'
Learnable Parameters: $W_1 \in \mathbb{R}^{\mathrm{F}'\mathrm{x}(2\mathrm{F}+\mathrm{P})}$, $W_2 \in \mathbb{R}^{1\mathrm{xF}'}$, $W_G \in \mathbb{R}^{\mathrm{PxP}'}$

1: $t_{ijk} = [H_i || H_j || G_k]$ ▷ Triple t_{ijk} for features of (N_i, N_j, E_k) where E_k connects (N_i, N_j)
2: $c_{ijk} = W_1 t_{ijk}$ ▷ Triple intermediate feature $c_{ijk} \in \mathbb{R}^{\mathrm{F}'}$
3: $e_{ijk} = \mathrm{LeakyReLU}(W_2 c_{ijk})$
4: $\alpha_{ijk} = \begin{cases} \mathrm{softmax}_{jk}(e_{ijk}), & \text{if } j \in \mathrm{Neighborhood}(N_i) \\ 0, & \text{else} \end{cases}$
5: $H'_i = \sum_{j=1}^{N} \sum_{k \in E_{ij}} \alpha_{ijk} c_{ijk}$ ▷ $k \in E_{ij}$ for all edges connecting (N_i, N_j)
6: $G' = G W_G$ ▷ Linear transformation on edge embeddings
7: **Return** H', G'

Nathani also adds meta-relations to n-hop neighbors to capture longer-range structure while using few graph attention layers. This helps preserve local information which is often diluted in very deep GNNs. He trains this model with the TransE translational objective, and initializes the entity and relation embeddings, H and G, with trained TransE embeddings. He justified this TransE initialization because his model used a similar translational objective. His results significantly outperform SOTA results on standard datasets, so we explore whether KBAT demonstrates similar remarkable performance on DKG.

2.5 Link Prediction with ConvKB [Nguyen et al., 2018]

ConvKB is a state-of-the-art embedding model for link prediction. Triples are represented in a three column matrix where the first column comprises of triple heads, the second column is triple relations, and the third column is triple tails. This matrix is processed by a convolutional neural network (CNN) which maps each triple to a score, where higher scores represent higher likelihood of that triple being valid. Nguyen et. al. demonstrated SOTA performance on link prediction for standard KG datasets, so Nathani et. al. used ConvKB as the decoder, after training embeddings with the translational objective.

3 Dataset

3.1 Diffbot

Diffbot is a startup founded in 2008 at Stanford University which offers public APIs for gathering structured data off the Web. They maintain an "automated knowledge graph" by crawling the web and using its automatic Web

page extraction to build a large database of structured web data. Michael Tung, the CEO of Diffbot, gave us access to Diffbot's knowledge graph API to explore techniques for generating useful embeddings.

3.2 Structure

The Diffbot Knowledge Graph™ is automatically generated using Diffbot's proprietary algorithms that scrape around 50 billion public web pages. Their algorithms are able to parse and aggregate information from general unstructured web text sources. The graph contains about 10 billion entities, each with a unique `entityID`, and is adding 150 million entities monthly. Each entity is of a certain type, and contains associated metadata attributes. The primary entity types are **Person, Corporation, Organization, Educational Institution, Local Business, Skill, Place, Administrative Area, Product, etc**. Since LinkedIn is one of Diffbot's deeply crawled sites, it is very likely that members of CS224w staff and students are present in the graph. (We were able to find ourselves!)

Entity types have specific attributes - for example, an **Organization** entity's attributes include *name, foundingDate, founders, ceo, etc.*. Some attributes are strings (*name*), datetimes (*foundingDate*), etc. but others are entities (*ceo*) or lists of entities (*founders*). If entity **H** has entity **T** as attribute R, then we say the triple (**H**, R, **T**) is in the graph. Although we don't use non-entity attributes, we keep this data in the dataset for future use.

3.3 Data Collection

In order to work with such a large graph, we first needed to extract a subgraph for offline experimentation. The Diffbot API allows querying an `entityID` and returns a JSON payload containing the attributes of the corresponding entity.

We performed a multi-threaded Breadth-First-Search of Diffbot's Knowledge Graph. For each visited entity, we create an object keeping track of the type, name, and triples this entity is involved in, as well as a short text description of the entity and Diffbot's source of information (such as Wikipedia or LinkedIn). We define a relation type by the path in which we traverse the JSON to arrive at an entity. For example the triple (Apple Inc, /location/city, Cupertino) is inferred from

```
{"Apple Inc": {
  "location": {
    ...,
    "city": Cupertino
  },
  ...
}}
```

We seeded the BFS with President Donald Trump's `entityId` and parallelized the search across 100 threads. We collected a graph G_{560k} of about 560,000 entities and 6 million relations. In this sub graph, we captured about 15 entity types and 30 relation types.

3.4 Preprocessing

Knowledge graphs created from Web scraping often contain inaccurate data, duplicate entities, and redundant relations. Diffbot is focused this year on making their knowledge graph more accurate through a variety of data verification and fusion techniques. However, we observed that our subgraph contained redundant entities and relations. In addition, our subgraph of around 560,000 entities was still very big to work with for initial experimentation, so we implemented a pruning algorithm to capture an even smaller, more densely connected subgraph to start with.

3.4.1 Redundant Entities

When scraping the Web, it is common to come across the same entity across different sites. Diffbot tries to fuse this knowledge together to minimize the amount of duplicate entities present in the graph. However, in our subgraph we observed that several thousand names belonged to more than one entity. In total, 31,195 entities were found to have a name clash with another entity. It is possible that some of these entities represent different people/places/things with

the same name, but the overwhelming majority appeared to be duplicates. For example, the CEO of Mastercard, Ajay Banga, appeared in our subgraph twice, with two different IDs: (Ajay Banga — Pl0EhRdyiN46hgtuJSExnrA) and (Ajay Banga — PpU7ZWl4HN9il9e4ZYvIuQQ).

We decided to remove duplicates with the same name in order to clean the graph and allow each entity in the graph to represent a single entity in the real world. We had to be careful not to remove distinct entities that just happened to have the same name (e.g. there are many different people named James Smith). Thus, we only removed entities if the set of their triples had an intersection-over-union (IOU) above a certain threshold, or if one's triple set was a subset of the other. In total we removed 17,060 redundant entities from DKG. After removing, we were left with G_{544k}.

Algorithm 2 Prune to Dense Subgraph. Iteratively sort and remove lowest-degree entities from the graph.

Input: G
Returns: G
Parameters: max_size, min_remove

while $|G| >$ max_size: **do**
    G_{rev} = reverseRelations(G) ▷ reverseRelations maps (H, R, T) → (T, R, H)
    outDeg = getOutDeg(G) ▷ {entity: outDeg}
    inDeg = getOutDeg(G_{rev}) ▷ {entity: inDeg}
    totalDeg = inDeg + outDeg ▷ entity: total degree
    lowDeg = getKsmallest(totalDeg, min_remove) ▷ gets the degree of the min_remove smallest entity
    G = trimGraph(G, lowDeg, totalDeg) ▷ removes all entities with degree ≤ lowDeg
Return G

3.4.2 Selecting Subgraphs

Generating embeddings for Diffbot's entire graph (billions of entities and trillions of edges) is outside the scope of this project. This is a challenging engineering task that requires distributing embeddings across a cluster of GPUs. PyTorch-BigGraph is a recent open-sourced distributed system for training embeddings for such a large graph, but it is not conducive toward model exploration and we leave this to future work.

Instead, we focus on achieving high quality embeddings for a variety of subgraphs, hoping that our results on these subgraphs may generalize to the full graph or at least give us insight toward how we may approach this problem. We selected a set of subgraphs that we could fit in memory for various embedding methods and that fulfilled various desirable properties for embedding.

First, we pruned G_{544k} randomly, resulting in subgraphs of size 300k, 100k, 30k and 15k. The randomly pruned graphs exhibited a large loss in relations proportional to the number of entities removed. As a result, the graphs were sparse, had few relation types, and low average clustering coefficients. The degree distributions plotted on a log-log scale can be seen in Figure 1. Because the network properties and connectivity degrades significantly as nodes are removed, we can tell that the DKG is not robust - random removal results in the loss of some entities, such as *United States of America*, which are integral to the graph connectivity. Thus, we need a different method for selecting desirable subgraphs.

We sought to select connected subgraphs with high average degree. In order to prune the original graph to smaller, dense subgraphs, we used the following dense pruning algorithm, as described in Algorithm 2, to produce subgraphs of size 300k, 100k, 30k, and 15k. Additionally, the 15k subgraph ⊂ 30k subgraph ⊂ 100k subgraph ⊂ 300k subgraph. The resulting subgraphs were highly connected, contained more relation types and exhibited a high average degree than the randomly-selected subgraphs. The degree distributions plotted on a log-log scale can also be seen in Figure 1. We list the counts of entity types in each subgraph in Table 1 to show which types of entities were removed in our dense pruning. Notice that we remove a very large number of Corporations - there were many insignificant Corporations with very few relations in DKG.

	Dense 15k	Dense 30k	Dense 100k	Dense 300k	544k
AdministrativeArea	158	361	2,000	14,923	18,977
Corporation	133	673	23,041	148,806	349,692
DegreeEntity	5	5	5	5	5
EducationalInstitution	1	12	380	2,463	4,545
EmploymentCategory	42	47	47	47	48
LocalBusiness	2	19	875	6,957	9,957
Miscellaneous	518	822	1,319	1,963	4,095
Organization	138	446	5,207	31,145	53,640
Person	13,538	26,828	65,842	91,757	99,969
Place	1	1	16	84	160
Role	0	1	4	6	13
Skill	473	794	1,273	1,853	3,649

Table 1: Counts of entity types for each densely pruned subgraph.

	Random 15k	Random 30k	Random 100k	Random 300k
# entities	15,009	30,009	100,009	300,009
# relations	2,984	10,290	148,735	1,129,028
# relation types	21	25	29	30
Average degree	0.3976	0.6858	2.9744	8.1315
Avg. clustering coefficient	0.0011712	0.0024635	0.0830886	0.1668803

	Dense 15k	Dense 30k	Dense 100k	Dense 300k
# entities	15,009	30,009	100,009	300,009
# relations	635,764	1,070,817	2,043,741	3,525,519
# relation types	28	30	30	31
Average degree	84.7177	71.3664	40.8711	20.5055
Avg. clustering coefficient	0.0034036	0.0055966	0.0439629	0.1657300
Max PageRank	0.032380	0.040049	0.082616	0.131864

Table 2: Basic statistics for random and densely pruned subgraphs of size 15k, 30k, 100k, and 300k.

3.5 Analysis & Statistics

Below we report basic analysis and statistics for all of our pruned subgraphs. Basic characteristics of the pruned subgraphs, including entity and relation counts, relation type counts and average degree, can be found in Table 2.

3.5.1 Entity & Relation Type Connectivity

The clustering coefficient, when applied to a single node, is a measure of how complete the neighborhood of a node is. We computed the average clustering coefficient for the pruned graphs and documented them in Table 2. Note that, when calculating clustering, we assume the graph is homogeneous (i.e. relations don't have types).

3.5.2 Degree distribution

Figure 1 shows the total degree distribution for all subgraphs plotted on a log-log scale. Each subgraph follows a similar distribution as the parent network from which it was derived from. As expected, the randomly-pruned subgraphs have near-monotonically decreasing degree distributions. However, the densely-pruned subgraphs exhibit few entities on the left side of the distribution with lower degree. This is an artifact of the pruning algorithm - during pruning when we are nearing the target number of entities, removing the last few entities (with low degree k) causes the degree of several other entities to fall below degree k. These few low-degree entities are accentuated because the plot is on a log-log scale. Notice how, as we prune to smaller and smaller subgraphs, the minimum degree increases, corresponding to an increase in average degree.

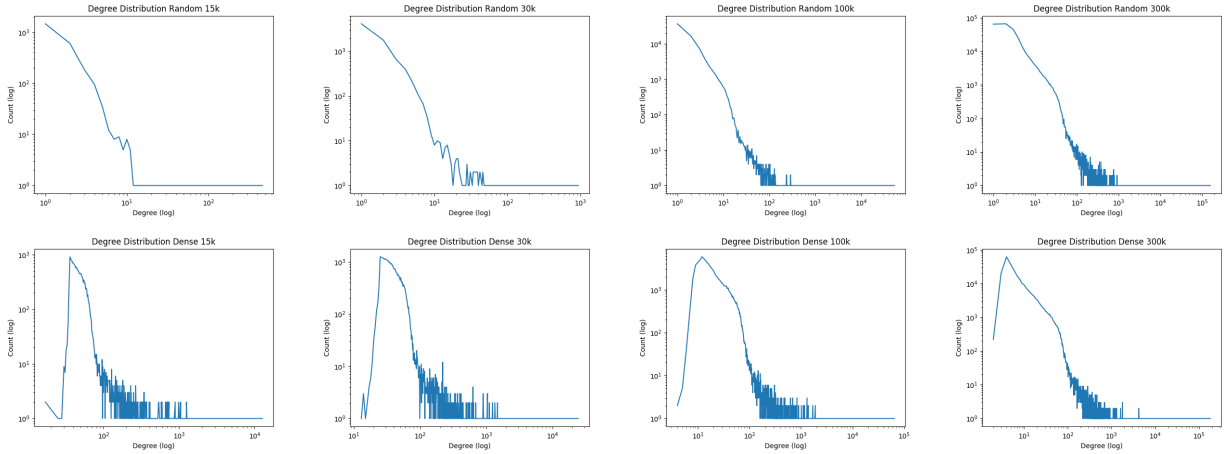

Figure 1: Degree distribution on a log-log scale for randomly and densely pruned subgraphs of size 15k, 30k, 100k, and 300k.

Entity Name	Entity Type	PageRank Score (Dense 30k)
United States of America	AdministrativeArea	0.039232682417297056
Executive	EmploymentCategory	0.009916846725975165
management	Skill	0.008147224456748725

Table 3: Top 3 Entities by PageRank Scores

3.5.3 PageRank

The distribution of PageRank scores for entities in our pruned subgraph are plotted in Figure 2. Notice how there are a few entities with very high pagerank scores. The entities with the highest PageRank scores are listed in Table 3. For example, we see that United States has the highest PageRank score, because this is the tail of */locations/country* for most entities in the graph. Similarly, other high PageRank entities are common attributes of **Person** entities.

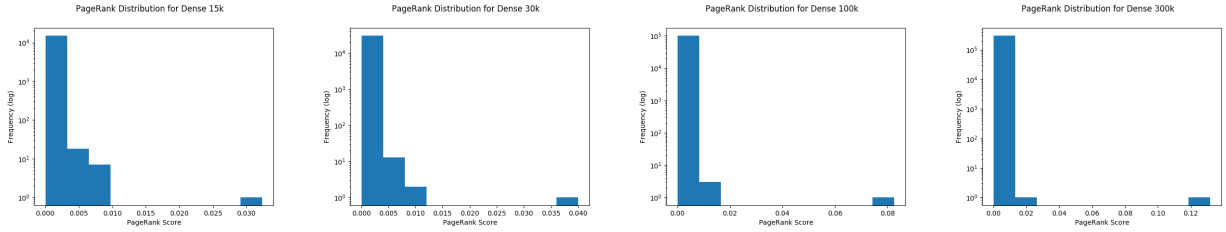

Figure 2: Distribution of PageRank scores for entities in the densely pruned subgraphs.

4 Embedding Methods

We train TransE and RotatE embeddings for each of the dense 15k, 30k, 100k, and 300k subgraphs. We evaluate their performance on the link prediction task using raw TransE (eq 1) and RotatE (Eq 2) scores respectively.

Then, we train a ConvKB [Nguyen et al., 2018] model for the same link prediction task. We use the trained TransE and RotatE embeddings and evaluate only on 15k and 30k graphs because the larger graphs do not fit in memory with this model. We see a significant increase in performance using the ConvKB scoring mechanism as opposed to the original TransE and RotatE scores.

Finally, we train a KBAT [Nathani et al., 2019] model paired with a ConvKB model. As described in 2.4 we initialize KBAT with the trained TransE embeddings. We also initialize KBAT with trained RotatE embeddings (this is outside the scope of Nathani's work) and compare the results. We see that this initialization performs very

	Dense 15k	Dense 30k	Dense 100k	Dense 300k
Hits@10 (TransE)	0.3713	0.3493	0.3909	0.4431
Hits@3 (TransE)	0.2260	0.2136	0.2572	0.3324
Hits@1 (TransE)	0.1257	0.1173	0.1182	0.1733
MRR (TransE)	0.2052	0.1923	0.2112	0.2712
MR (TransE)	695.8	1389.7	4246.5	9156.6
Hits@10 (RotatE)	0.3628	0.3354	0.3917	0.4457
Hits@3 (RotatE)	0.2116	0.1914	0.2633	0.3414
Hits@1 (RotatE)	0.1096	0.0996	0.1646	0.2343
MRR (RotatE)	0.1909	0.1745	0.2387	0.3075
MR (RotatE)	634.7	1331.4	2497.1	4505.2

Table 4: TransE and RotatE link-prediction performance for densely pruned subgraphs

similarly. We did not try a random initialization for KBAT because Nathani claimed this has very poor performance.

Note, for each embedding method and for each graph, we ran a separate hyperparameter search. We spent significant effort automating and parallelizing this search across several GPUs.

4.1 TransE and RotatE

Open-KE is an open-source framework for knowledge embedding implemented with PyTorch [Han et al., 2018]. The GitHub repository has 1.4k stars and the code has been optimized to run on the GPU and tested thoroughly. The library implements some underlying operations such as data preprocessing and negative sampling as well as TransE and RotatE embeddings. We modified the Open-KE implementation to support a parallelized hyperparameter search as well as Tensorboard logging and checkpointing.

For TransE, the hyperparameter search yielded a configuration of embedding dimension 200, margin 5.0, batch size 4000, negative ratio 64, and Adam optimizer. The optimal learning rate was 1e-3 for 15k, 30k, and 100k graphs, and the learning rate was 1e-4 for 300k.

For RotatE, the hyperparameter search yielded a configuration of entity embedding dimension 400, relation embedding dimension 200, margin 6.0, batch size 4000, negative ratio 64, and Adam optimizer. The optimal learning rate was 1e-3 for 15k and 30k graphs, and the learning rate was 1e-4 for 100k and 300k graphs. We trained until convergence.

4.2 ConvKB & KBAT

We built off Nathani's implementation of ConvKB and KBAT. We make several modifications. For example, we don't use his meta-relations between n-hop neighbors. This is because our graphs have significantly denser relations than the graphs on which Nathani experimented. The number of n-hop neighbors grows at order n with respect to the graph density so adding this exorbitant number of meta-relations would be infeasible. Additionally, whereas Nathani only initializes KBAT with trained TransE embeddings, we also try initializing KBAT with trained RotatE embeddings. RotatE embeddings did not gain prominence until after Nathani released his results in 2019, so Nathani did not try such a RotatE initialization. When we discussed with him, he believed TransE initialization may perform better because they share the translational objective, but we thought it would be interesting to try RotatE initialization as well. We ran an optimal hyperparameter search over KBAT and ConvKB. On the 15k graph, we found the best performance when the KBAT had an initial learning rate of 0.0001, a weight decay of 0.0001 a batch size of 86835 and a valid/invalid ratio of 2. The parameters were the same for the 30k graph with the exception of a weight decay of 0.001. For both graphs, ConvKB performed best with batch size of 128, a LeakyRelu alpha of 0.2, and a valid/invalid ratio of 40.

	Dense 15K (ConvKB only)	Dense 15k (KBAT + ConvKB)	Dense 30k (ConvKB only)	Dense 30k (KBAT + ConKB)
Hits@10 (TransE)	0.5929	0.7343	0.4989	0.6612
Hits@3 (TransE)	0.4855	0.6672	0.3940	0.5794
Hits@1 (TransE)	0.3796	0.5935	0.2901	0.4951
MRR (TransE)	0.4529	0.6429	0.3626	0.5525
MR (TransE)	419.1079	362.1288	806.4607	828.2424
Hits@10 (RotatE)	0.5940	0.7301	0.4926	-
Hits@3 (RotatE)	0.4878	0.6644	0.3828	-
Hits@1 (RotatE)	0.3829	0.5931	0.2842	-
MRR (RotatE)	0.4557	0.6410	0.3555	-
MR (RotatE)	412.1633	363.9889	804.5306	-

Table 5: Results from KBAT and ConvKB experiments. MRR is mean reciprocal rank and MR is mean rank. Blank column because RotatE initialization for KBAT has 2x embedding dimension, so doesn't fit in memory.

5 Results

We display our embedding results in Table 4 and Table 5. Table 4 contains the best link-prediction results using TransE and RotatE embeddings and raw scoring (translational & rotational distance) on 15k, 30k, 100k, and 300k subgraphs. Table 5 displays link prediction results using ConvKB as a learned scoring mechanism and using KBAT to enhance the TransE and RotatE initialized embeddings.

5.1 Comparing Performance of TransE vs. RotatE

TransE and RotatE embeddings perform about the same on the link prediction task, with a $\delta < 0.01$ for most configurations. This is likely due to the relational patterns in DKG - we notice there are few symmetric, inverted, or composite relations. Within the ~ 30 relation types, we could not identify any that exhibited these relation patterns adversarial to the TransE objective. So it makes sense that TransE and RotatE perform similarly with both raw scoring and ConvKB scoring, even though the RotatE objective and larger entity embedding dimension allow more model expressivity.

It is more surprising that KBAT initialized with TransE embeddings performs similarly to KBAT initialized with RotatE embeddings (see column 2 of Table 5). ConvKB scoring (without KBAT) also performs similarly on both TransE and RotatE embeddings (see column 1 of Table 5). Notable differences are: KBAT shares the translational objective with TransE but not with RotatE, RotatE entity embeddings are higher-dimensional than TransE ones, and RotatE can model more complex relation patterns. Additionally, TransE raw scores perform slightly better, by about 1 percentage point, than RotatE raw scores (see column 1 of Table 4). The similarity between these initializations suggests that KBAT may be robust to the training objective of initialized embeddings, as long as the embedding initialization provides sufficient information about graph structure.

5.2 Comparing Performance between Dense Subgraph Sizes

Our initial motivation for considering various subgraph sizes was to find a graph whose embedding model could fit in memory. However, we are also very interested if there is a correlation between graph size/structure and embedding quality for each of the models we examine.

5.2.1 Raw Scoring

For raw scoring, we compare performance across all subgraph sizes. We observe that, for both TransE and RotatE, the graphs' link prediction performance in increasing order is 30k, 15k, 100k, 300k. This is interesting because the average degree decreases with increasing graph size and the average clustering coefficient increases with increasing graph size. Additionally, we must remember that the 15k subgraph $\subset$ 30k subgraph $\subset$ 100k subgraph $\subset$ 300k subgraph. This means sets of triples are also subsets of each other. Thus, since we generally achieve higher link prediction accuracies in larger subgraphs, larger subgraphs likely contains many triples that are classified correctly, but do not exist in

the smaller subgraphs. We hypothesize that, in each pruning step where we remove the lowest-degree entities, these entities have similar structural roles in the graph and their triples are easy to predict.

5.2.2 ConvKB and KBAT

For both ConvKB and KBAT, we achieve significantly higher performance (about 10 percentage points) on 15k subgraph than the 30k subgraph. This corresponds to slightly higher performance of 15k over 30k using raw scoring. We weren't able to run these more complex models on the larger subgraph sizes, so it is hard to generalize these findings. But we believe that these more expressive models amplify differences in performances across subgraph sizes. So we predict that, if we were to evaluate KBAT on the 300k subgraph, we would achieve very high performance.

5.3 Comparing Performance with ConvKB and KBAT

We observe that using the ConvKB scoring as opposed to raw scoring provides a significant boost in link prediction performance. The boost is about 20 percentage points (comparing column 1 of Table 4 and column 1 of Table 5) in the case of the 15k subgraph and about 10 percentage points in the case of the 30k subgraph (comparing column 2 of Table 4 and column 3 of Table 5). This is because the CNN in ConvKB considers global relationships and transitional characteristics of the TransE and RotatE embeddings whereas the raw scoring method only considers the distance of embeddings for a local triple. The higher boost for the smaller subgraph may be because this smaller subgraph is more densely connected so global information has greater significance.

We observe impressive link prediction results using KBAT with trained RotatE and TransE embeddings. We can view KBAT training using TransE or RotatE initialization as fine-tuning the embeddings leveraging global graph structure. This fine-tuning yields an impressive 20% increase in performance using ConvKB scoring. A Hits@1 of 60% for the 15k subgraph and 50% for 30k subgraph is extremely high compared to the corresponding SOTA on FB15K-237, WN18RR and NELL-995. It is lower than SOTA on Kinship, which has a Hits@1 of 86% [Nathani et al., 2019]. This suggests that the structure of DKG may be easier to learn than some of these standard datasets. This high performance on small subgraphs is also exciting because it is promising for high quality embeddings for even larger subgraphs of DKG. But only further experimentation will be able to confirm this hypothesis.

6 Conclusion

In this paper, we provided three novel contributions: **1.** Curation and characterization of a new real-world web-scale dataset; **2.** Comparison of embedding (TransE & RotatE) quality for various subgraphs of DKG; and **3.** Attention-based embeddings (KBAT) for DKG with very high link-prediction performance (we also showed that KBAT is robust to the training objective of initialized embeddings).

We hope that the introduction of DKG will help kickstart increased effort toward embedding real-world web-scale knowledge graphs. Improving embedding quality for web-scale knowledge graphs will benefit a multitude of downstream tasks and open doors for new, creative applications of graph embeddings. Future work should focus on scaling our approaches toward larger subgraphs of DKG and determining whether larger subgraphs continue to achieve these high levels of performance.

6.1 Acknowledgements

We'd like to thank Michael Tung (Diffbot CEO) for sharing Diffbot's dataset.
We'd also like to thank Rex Ying for sharing his ideas and providing guidance.

References

[Bordes et al., 2013] Bordes, A., Usunier, N., Garcia-Durán, A., Weston, J., and Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 2787–2795, USA. Curran Associates Inc.

[Han et al., 2018] Han, X., Cao, S., Xin, L., Lin, Y., Liu, Z., Sun, M., and Li, J. (2018). Openke: An open toolkit for knowledge embedding. In *Proceedings of EMNLP*.

[Lee et al., 2018] Lee, J. B., Rossi, R. A., Kim, S., Ahmed, N. K., and Koh, E. (2018). Attention models in graphs: A survey. *arXiv preprint arXiv:1807.07984*.

[Nathani et al., 2019] Nathani, D., Chauhan, J., Sharma, C., and Kaul, M. (2019). Learning attention-based embeddings for relation prediction in knowledge graphs. *arXiv preprint arXiv:1906.01195*.

[Nguyen et al., 2018] Nguyen, D. Q., Nguyen, T. D., Nguyen, D. Q., and Phung, D. (2018). A novel embedding model for knowledge base completion based on convolutional neural network. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 327–333, New Orleans, Louisiana. Association for Computational Linguistics.

[Sun et al., 2019] Sun, Z., Deng, Z.-H., Nie, J.-Y., and Tang, J. (2019). Rotate: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations*.

[Veličković et al., 2017] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.