Large Scale Link Prediction in Knowledge Graphs

Aakash Khochare [*1] **Sweta Sharma** [*1] **Tony Gracious** [*1]

Abstract

Knowledge Bases (KBs) such as FreeBase (Bollacker et al., 2008), Nell (Mitchell et al., 2015) and DBPedia (Bollacker et al., 2008) are becoming popular methods to condense knowledge into a structured form. Such KBs are however far from complete and hence there is an emphasis on developing techniques that infer new relations from the existing ones. There have been a lot of literature that tackles this problem by projecting the existing KB entries into a low dimensional vector using tensor factorization. Our project will be focus using the current state of the art knowledge embedding techniques for doing scalability study using distributed computing. In this work, we focused on particular two state of the art techniques Complex embedding (Trouillon et al., 2016) and Holographic Embedding (Nickel et al., 2016)

1. Introduction

The are existing knowledge bases like FreeBase, Nell and DBPedia where entities and relations are stored as triplets $< es, r, eo >$. Here, es is subject entity, r is the relation and eo is the object entity. These knowledge bases are incomplete and we want to find the relation between entities. So, we want to model it as querying problem to find the most probable r for a query $(es, ?, eo)$ containing subject and object.

The goal of this work is to learn vector representation for entities and relations for building a model. According to (Trouillon et al., 2016), the model should be able to learn relations that have following property and the embedding learning techniques should be linearly scalable in time and space to large datasets.

- Reflexive: Entities are related to itself. For example relation 'Similar To' is a reflexive relation as the entities can be similar to itself.
- Ir-reflexive: Entities cannot be related to itself. For example relation 'is inside', as entities cannot be inside itself.
- Symmetric: Relations where subject and object can be interchanged without affecting the relation. For example, Spouse of' where man and wife can be interchanged without affecting
- Anti-Symmetric: Relations where subject and object cannot be interchanged. For example, 'husband of' where we cannot exchange man with the wife and make the same sense as the original relation.
- Transitive: Relation where one entity1 is related to entity2 by relation r and entity2 is related to entity3 by r then entity1 is related to entity3 by r. For example relation 'inside of' is a transitive relation.

2. Methods

The current methods in literature use vector embeddings for entities and relations and assign a scoring function to each triplets in the KB. Vector embeddings are learned in such a way that scoring function should give more score for triplets in the KB compared to triplets not in KB which are generated through corrupting the known triplets. These works are mainly based on inner product similarities which can easily represent reflexive, ir-reflexive, symmetric and transitive relations but representing antisymmetric relations have been achieved using by increasing the model parameters (Nickel et al., 2011) or by modelling subject and object entities differently which resulted in poor performing models due to over-fitting.

2.1. Complex Embedding(Trouillon et al., 2016)

This embedding technique uses a complex vector embedding of size K for each entities and relations. Uses the following scoring function $\phi(es, r, eo)$. Since complex embedding uses hermitian inner-products which are not symmetric so it can represent antisymmetric relations

[1]Indian Institute of Science, Bangalore. Correspondence to: Aakash Khochare <aakashkhochare@gmail.com>, Sweta Sharma <sharma.sweta1990@gmail.com>, Tony Gracious <tonygracious@gmail.com>.

$$\begin{aligned}\phi(es, r, eo) &= Re(< es, r, \bar{eo} >)\\ &= Re(\sum_{k=1}^{K} es_k * r_k * \hat{eo}_k)\\ &=< Re(es) * Re(r) * Re(eo) > +\\ &< Im(es) * Re(r) * Im(eo) > +\\ &< Re(es) * Im(r) * Im(eo) > -\\ &< Im(es) * Im(r) * Re(eo) >\end{aligned} \quad (1)$$

It uses this scoring function to find the probability that a triplet is a valid triplet in KB using sigmoid of the scoring function $P(Y = 1) = \sigma(\phi(es, r, eo))$. The embeddings are learn in such a way the given a triplet we should be able to predict whether triplet is a valid triplet in KB ($Y_i = 1$) or negative sample ($Y_i = -1$) created by corrupting samples in the triplet. Given a KB and negative samples generated from it, the model uses following objective function for finding embedding vectors by minimizing the log loss in predicting whether a triplet is in KB

$$\min_{\forall\theta} \sum_{i=1}^{m} log(1 + exp(-y_i \phi(es_i, r_i, eo_i))) + \lambda ||\theta||^2 \quad (2)$$

Here, θ is embedding parameters for entities and relations and m is the number of samples both positive and negative samples.

2.2. Holographic Embedding

In Holographic embedding circular correlation of vectors is used to represent pairs of entities, i.e., we use the following compositional operator:

$$a \circ b = a \star b,$$

where $\star : \mathcal{R}^K \times \mathcal{R}^K -> \mathcal{R}^K$ denotes circular correlation:

$$[a \star b]_k = \sum a_i b_{(k+i) mod\ d}.$$

The probability of a relation triple is modelled as :

$$Pr(\phi_p(s, o) = 1|\Theta) = \sigma(r^T (es \star eo)).$$

where σ denotes the logistic function. The circular correlation can be computed as :

$$a \star b = \mathcal{F}^{-1}(\bar{\mathcal{F}(a)} \odot \mathcal{F}(b)).$$

The correlation operator is non-commutative and also contains a similarity component for $k = 0$. We learn the vector representations for relations and entities by minimising the regularized logistic loss :

$$min \sum_{i=1}^{m} log(1 + exp(-y_i \eta_i)) + \lambda ||\Theta||_2^2$$

where y_i = 1 if the relation triple is in the knowledge graph and -1 otherwise. To generate negative triples we use the Local Closed World Assumption by corrupting either the subject or the object of the relation triple.

3. Implementation and Results

3.1. Experimental Setup

All the experiments were performed on the Turing cluster. The batch size was fixed at 10,000 samples for both local and distributed versions. The experiments were performed on the Freebase 15k-237 dataset, which as roughly 15k entities, 237 relations and 272,115 triples. SGD was used for optimization along with AdaGrad, that had a starting rate of 0.05. All the vectors had a dimension of 100. Static LCWA sampling was performed, so all the negatives were pre-determined and loaded into the main memory.

3.2. Complex

3.2.1. Local

For local implementation, mini-batch stochastic optimization using adagrad optimizer. We converted each equation into matrix-matrix multiplications instead of the matrix-vector multiplications since matrix-matrix multiplications are better parallelized. The scoring function have 4 parts each part $< A, B, C >$ can implemented as matrix-matrix multiplication $A \odot B \odot C * \mathbb{1}$ where $A \in \mathbb{R}^{(m*K)}, B \in \mathbb{R}^{(m*K)}, C \in \mathbb{R}^{(m*K)}$ and $\mathbb{1} \in \mathbb{R}^{K*1}$ a all one vector. Here m, is the minibatch size and use these equation we can get scoring function $\Phi \in \mathbb{R}^{m*1}$.

$$\begin{aligned}\Phi(Es, R, Eo) &=< Re(Es) * Re(R) * Re(Eo) > +\\ &< Im(Es) * Re(R) * Im(Eo) > +\\ &< Re(Es) * Im(R) * Im(Eo) > -\\ &< Im(Es) * Im(R) * Re(Eo) >\end{aligned} \quad (3)$$

Here $Es \in \mathbb{C}^{(m*K)}, R \in \mathbb{C}^{(m*K)}, Eo \in \mathbb{C}^{(m*K)}$ where $\mathbb{C}$ is the complex domain. The loss function can be written as

$$log(\mathbb{1}_{m*1} + exp(-Y_{m*1} \odot \Phi(Es, R, Eo))) * \mathbb{1}_{m*1} + ||\theta||_F \quad (4)$$

here log and exp are applied elementwise for each element of a vector and $||.||_F$ is frobenious norm. For each batch in from training dataset, negative samples are generated by local close world assumption by corrupting subject and object entities and passed along the training data with label -1.

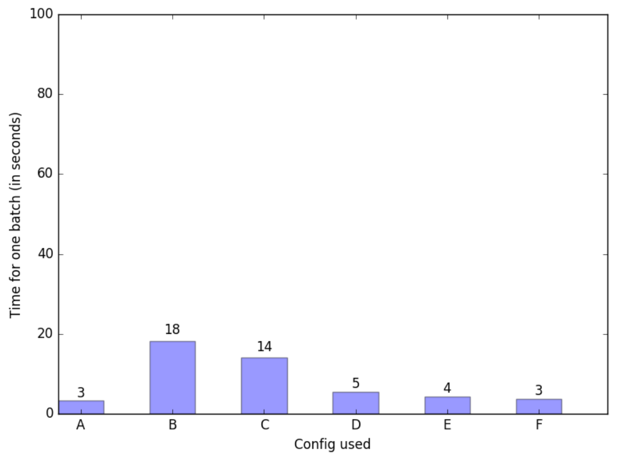

Figure 1. Timings for Configs used

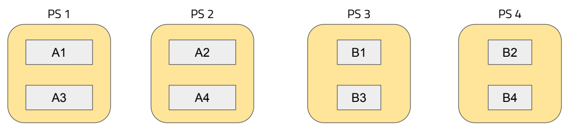

Figure 2. Imbalanced Parameter Servers

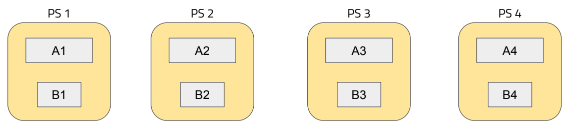

Figure 3. Balanced Parameter Servers

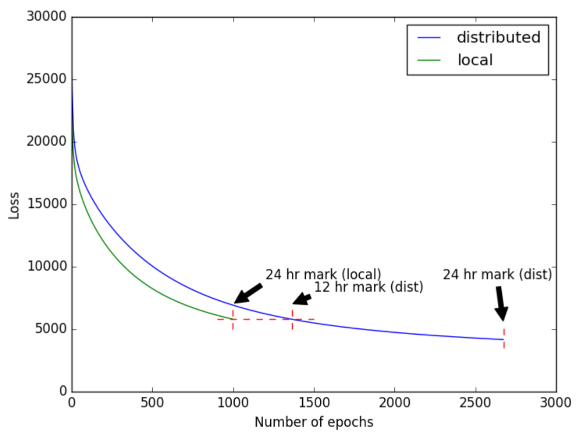

Figure 4. Loss for Local and Distributed

3.2.2. Parallel

For the parallel implementation, Synchronous Parallel SGD was used on different shards of the data. We performed parameter mixing after every batch. However, as we can see in Figure 1, config B, the per batch time actually increased over the local implementation. So, just adding the parallelism constructs weren't sufficient and more work had to be done.

First, we reduced the number of vectors that were pulled from the Parameter servers by uniquely identifying and extracting only the vectors that were needed for the computation in the batch. Config C, notes the run time for this optimization and as we can see, it gives us a benefit of 4 secs.

Next, we increased the number of parameter servers, from 1 to 2, 4 and 6 noted by Configs D, E and F. We can see that the run times improve significantly as the number of parameter servers increases from 1 to 2 but, incresing any further shows a diminishing return. However, balancing the load was an important aspect in using the parameter servers. Figures 2 and 3 show two different configurations with and without imbalance in the amount of network transfers that have to be handled by each parameter server. Table 1 shows the information about all the configurations that were used.

3.2.3. Results

We let both the Local and Distributed versions run for around 24 hrs. The values of Loss can be seen in Fig 4, it is worth noticing that the distributed version matches the loss of the local one in just 11.74 hrs. Table 2 shows the hits@n and mean reciprocal rank (MRR) metrics achieved after training for 24 hrs. We can see that the results don't really match that in the paper. The paper reports the Hits@10

Table 1. Configurations used in evaluation

Config	A	B	C	D	E	F
# of PS	N/A	1	1	2	4	6
# of Workers	N/A	4	4	4	4	4
Comments	Local	Parallel but not optimized for network calls	Parallel and optimized	Tests the effect of PS	Tests the effect of PS	Tests the effect of PS

to be 0.84. We attribute this to the static offline sampling that we performed.

3.3. Holographic Embedding

3.3.1. Local

For local implementation, mini batch stochastic optimization using Adagrad Optimizer was used. Here m, is the minibatch size and using these equations we can get scoring function $\Phi \in \mathbb{R}^{m*1}$.

$$\Phi(Es, R, Eo) = R \star (\mathcal{F}^{-1}(\mathcal{F}(\bar{E}s) \odot \mathcal{F}(Eo)) \tag{5}$$

Here $Es \in \mathbb{R}^{(m*K)}, R \in \mathbb{R}^{(m*K)}, Eo \in \mathbb{R}^{(m*K)}$ and $\star$ denotes row-wise dot-product. The loss function can be written as

$$log(\mathbb{1}_{m*1} + exp(-Y_{m*1} \odot \Phi(Es, R, Eo))) * \mathbb{1}_{m*1} + ||\theta||_F \tag{6}$$

here log and exp are applied elementwise for each element of a vector and $||.||_F$ is frobenious norm. For each batch in from training dataset, negative samples are generated by local close world assumption by corrupting subject and object entities and passed along the training data with label -1. The code was implemented in TensorFlow with offline sampling first. Initially the time taken for one epoch was approx 75 minutes. So we re-organised the tensorflow graph and brought it down to 30 minutes approx with offline sampling. However, the results obtained with offline sampling was not satisfactory as the loss on the validation set was not going down. So, we retried it with online sampling. But the time taken for online sampling per epoch increased to approx 60 minutes with no improvement in loss convergence. The time taken per epoch is large due to FFT calculation for each entity pair in the KB which is about O(KlogK) where K is the dimension of the entity vector.

3.3.2. Parallel

For the parallel implementation, Synchronous Parallel SGD was used on different shards of the data. We performed parameter mixing after every batch. We did experiments with one parameter server and two worker nodes. The time taken was approx 61 minutes for this configuration. With two parameter servers and two worker nodes the it was taking approx 52 minutes. The time taken per epoch improved without any improvement in convergence of loss.

4. Challenges in Large Scale Implemention

For learning the vector embedding we need to create negative samples for each positive samples in the KB. For online sampling, as the KB size increases the time for creating negative samples by corrupting triplets will increase. This will increase running time of each batch. For static offline sampling, the space required for storing this file increases with size of samples in the KB. We performed offline sampling which meant that the file size would be very large for Freebase 1M. Ideally a fast online sampler needs to be implemented. We did try to implement one in C++, however, it didn't workout well. Hence we were unable to test on a larger dataset.

References

Bollacker, Kurt, Evans, Colin, Paritosh, Praveen, Sturge, Tim, and Taylor, Jamie. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1247–1250. AcM, 2008.

Mitchell, Tom M, Cohen, William W, Hruschka Jr, Estevam R, Talukdar, Partha Pratim, Betteridge, Justin, Carlson, Andrew, Mishra, Bhavana Dalvi, Gardner, Matthew, Kisiel, Bryan, Krishnamurthy, Jayant, et al. Never ending learning. In *AAAI*, pp. 2302–2310, 2015.

Nickel, Maximilian, Tresp, Volker, and Kriegel, Hans-Peter. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 809–816, 2011.

Nickel, Maximilian, Rosasco, Lorenzo, Poggio, Tomaso A, et al. Holographic embeddings of knowledge graphs. 2016.

Table 2. Configurations used in evaluation

Config	**Hits@10**	**Hits@3**	**Hits@1**	**MRR**	**Time for 1000 epochs**
A	0.5376	0.3823	0.2444	0.3447	24.4 hrs
E	0.683	0.556	0.4076	0.5047	8.611 hrs

Trouillon, Théo, Welbl, Johannes, Riedel, Sebastian, Gaussier, Éric, and Bouchard, Guillaume. Complex embeddings for simple link prediction. In *International Conference on Machine Learning*, pp. 2071–2080, 2016.