ParHyTE: For Time-Aware Knowledge Graph Embeddings

Sruthi Gorantla (15190) [1]

Abstract

Stochastic Gradient Descent poses multiple challenges while training a model. It being inherently serial makes it difficult to parallelize without loss of computations. The main goal of this project is to exploit the sparsity while computing knowledge graph embeddings using translational embeddings. With theoretical guarantees, a parallelized version of SGD can still be used for TransE (Bordes et al., 2013) without losing the updates as shown in (Zhang et al., 2017). These guarantees can also be extended to a recent algorithm HyTE (Dasgupta et al., 2018), hyper plane based temporally aware knowledge graph embeddings. In this project, I show detailed analysis on how the sparsity of time information in *wiki_data* can be used to parallelize the computations of HyTE.

1. Introduction

1.1. Knowledge Graph

Knowledge graphs (**KG**) are structured graphs with varius entities as nodes and relations as edges. They are usually in the form of RDF triples *(h, r, t)*, where *h* represents a head entity, *t* a tail entity, and *r* the relation between them. Several KGs have sprung up in the recent information extraction, e.g., Freebase, WorkNet, YAGO etc., and have playes a pivotal role in supporting many applications, such as link prediction, question answering etc. Although these knowledge graphs are very large, i.e., usually containing thousands of relation types, millions of entities and billions of triples, they are still far from complete. As a result, *knowledge graph completion (KGC)* has been payed much attention to, which mainly aims to predict missing relations between entities under the supervision of the existing triples.

[1]Department of Computer Science and Automation, IISc Bangalore. Correspondence to: Sruthi Gorantla <gorantlas@iisc.ac.in>.

Course project for Machine Learning with Large Datasets, IISc, 2018.

1.2. Knowledge Graph Embeddings

The history of knowledge graph embeddings goes back to the basic *TransE* (Bordes et al., 2013) algorithm where, with just the triplets information, knowledge graph embeddings are found by minimizing the following loss function.

$$L = \sum_{x \in \mathcal{D}^+} \sum_{y \in \mathcal{D}^-} max(0, f(x) - f(y) + \gamma) \tag{1}$$

where

$$f(h, r, t) = ||e_h + e_r - r_t||_{l_1/l_2} \tag{2}$$

and for the set of positive samples observed in the knowledge graph $\mathcal{D}^+$, the negative samples are drawn randomly from the set:

$$\begin{aligned} \mathcal{D}^- = \{(h', r, t) | h' \in \mathcal{E}, (h', r, t) \notin \mathcal{D}^+\} \\ \cup \{(h, r, t') | t' \in \mathcal{E}, (h, r, t') \notin \mathcal{D}^+\} \end{aligned} \tag{3}$$

This loss function makes sure that the head entity, with respect to the relation, is close in space to the tail entity, but this is not true for a negative sample with a margin more than γ. These embeddings, however, do not take into account the validity of the relations. In datasets like *Wikidata12k*, each triple holds true only for certain period of time. This makes the KG more expressive and forces us to use this information while learning the embeddings. Recent work in this area, HyTE (Dasgupta et al., 2018), is an algorithm proposed to learn temporally-aware knowledge graph embeddings.

1.3. Distributed Computing

It is often the case that the number of such triplets that form the knowledge graph is very large. In fact, TransE takes several years to train on datasets like Freebase, which has millions of triples as the training set. This necessitates a paralleled computation in order to come up with the knowledge graph embeddings for the entire graph. However, it is challenging to parallel the translating embedding methods, since the training processes mainly employ stochastic gradient descent algorithm (**SGD**) or the variant of it. SGD in inherently sequential, as a dependence exists between each iteration. Parallelizing translating embedding methods straightforwardly will result in collisions between different processors. For instance, an entity embedding vector is updated by two processors at the same time, and the gradients calculated by these processors are different. In this

case, the diverse gradients are called *collisions*. To avoid collisions, we can use *locks*. But this will slow down the training process greatly as there are so many vectors. Nevertheless, in (Zhang et al., 2017), theoretical guarantees for the convergence of translational embeddings have been provided by making use of the structure of the knowledge graphs. As mentioned previously, the knowledge graphs are often far from complete, which make them sparse. In this project, I extend this sparsity in terms of temporal information in knowledge graphs and parallelize the recently proposed HyTE algorithm.

2. Related Work

Several embedding methods were proposed succeeding the TransE algorithm, that take into account extra information that is available in the knowledge graph itself. For example, TransH (Wang et al., 2014) projects the embeddings into the hyperplane corresponding to the relation in consideration. This way, the relations are not in the same embedding space and the entities also have meanings according to the relation. The objective function still remains same in TransH where as the score function changes.

$$f(h,r,t) = ||e_{h_\perp} + e_r - e_{t_\perp}||_{l_1/l_2} \tag{4}$$

where the entity embeddings are projected onto the relation hyperplane.

$$\begin{aligned} e_{h_\perp} &= e_h - w_r^\top e_h w_r \\ e_{t_\perp} &= e_t - w_r^\top e_t w_r \end{aligned} \tag{5}$$

where $w_r \in \mathbb{R}^{d_e}$ is the normal vector to the hyperplane related to the relation r. Similarly, TransR (Lin et al., 2015) employs rotation transformation to project the entities to a relation-specific space, i.e.,

$$f(h,r,t) = ||e_{h_r} + e_r - e_{t_r}||_{l_1/l_2} \tag{6}$$

where relation specific rotations are applied on the entities.

$$\begin{aligned} e_{h_r} &= M_r e_h \\ e_{t_r} &= M_r e_t \end{aligned} \tag{7}$$

Nevertheless, its is not easy to train translating embedding methods in parallel, since the main optimization algorithm SGD is born to run in sequence. The major obstacle to parallel SGD is the collisions between updates of different processors for the same parameter, to overcome which there are two main approaches.

The first strategy is to resolve the collisions according to specific data structure. For example, Hogwild! (Niu et al., 2011) is a lock-free scheme which works well for sparse data, which means that there is only a small part of parameters to update by each iteration of SGD. It has been proven that processors are unlikely to overwrite each others progress, and the method can achieve a nearly optimal rate of convergence. The second approach is to split the training data to reduce collisions. Downpour SGD divides the training data into a number of subsets, then the model replicas run independently on each of these subsets, and do not communicate with each other. Tensorflow, on the other hand splits the computation graph into a subgraph for every worker and communication takes place using Send/Receive node pairs.

Knowledge graphs specifically have the property of being very sparse. In (Zhang et al., 2017), it is shown that because of this sparsity, the SGD to train translational embeddings can be executed in parallel. In datasets like *Wikidata12k*, this sparsity can be proved for temporal information and hence HyTE, being a translational embedding method, also can use the parallel version of SGD for faster convergence.

3. Method

3.1. ParTransE

Firslty, knowledge graph is represented as $G = (E, R, T)$, where E is the set of entities with R the st of relations, and T is the set of triples (h, r, t), in which $h, t \in E$ and $r \in R$. The cardinalities of E, R,and T are n_e, n_r and n respectively. In this graph, nodes are entities, and edges are triples that connect nodes with distinguished relation. A hypergraph $H = (V, S)$ is now defined as follows:

Definition 3.1. *For a knowledge graph $G = (E, R, T)$, a 4-uniform Hypergraph $H = (V, S)$ is constructed where $V = \{E \cup R\}$ is the set of entities or relations, and S is the set of training samples s, where $s = \{(h, r, t, h') : h, t, h' \in E, r \in R\} \cup \{(h, r, t, t') : h, t, t' \in E, r \in R\}$*

For a hyperedge s in the hypergraph G,

$$\sigma(s) = \{s' : \exists r \in s \cap s', r \in R\} \tag{8}$$

denotes the set of hyperedges containing the same relations with hyperedge s,

$$\hat{\sigma} = max_{s \in S} |\sigma(s)| \tag{9}$$

denotes the maximal number of hyperedges containing same relations, where $|.|$ denotes the cardinality.

$$\rho(s) = \{s' : \exists e \in s \cap s', e \in E\} \tag{10}$$

denotes the set of hyperedges containing one or more same entities with hyperedge s,

$$\hat{\rho} = max_{s \in S} |\rho(s)| \tag{11}$$

denotes the maximal number of hyperedges containing same entities, where $|.|$ denotes the cardinality same as before.

Let us define the following events to prove that the number of collisions during the updates of the embeddings is very less.

- X_{smap} : event that p processors select p different samples.
- X_{rel} : event that there are collisions between relations, i.e., different processors update the same relation vector.
- X_{ent} : event that there are collisions between entities, i.e., different processors update the same entity vector.

Now the following have to be prove in order to be able to parallelize SGD with very less collisions.

1. $P(X_{samp} = 1) \approx 1$
2. $P(X_{rel} = 0) \approx 1$ and $P(X_{ent} = 0) \approx 1$

The theorems below show that these two conditions hold true when the knowledge graph is sparse, i.e., when $\hat{\sigma}$ and $\hat{\rho}$ are very small compared to the number of triplets n.

Theorem 3.1. *(Zhang et al., 2017) For a knowledge graph with n triples and training by p processors in parallel, when n is large and p is relatively small, the probability that p processors select p different samples is*

$$P(X_{samp} = 1) = \prod_{i=1}^{p-1}(1 - \frac{i}{n}) \approx 1 \tag{12}$$

Theorem 3.2. *(Zhang et al., 2017) For a knowledge graph with n triples and training with p processors in parallel, when $\frac{\hat{\sigma}}{n}$ is very small and $p < \frac{n}{\hat{\sigma}} + 1$, we have the probability of no relation in collision is*

$$P(X_{rel} = 0) = \prod_{i=1}^{p-1}(1 - \frac{i\hat{\sigma}}{n}) \approx 1 \tag{13}$$

Theorem 3.3. *(Zhang et al., 2017) For a knowledge graph with n triples and training with p processors in parallel, when $\frac{\hat{\rho}}{n}$ is very small and $p < \frac{n}{\hat{\rho}} + 1$, we have the probability of no relation in collision is*

$$P(X_{ent} = 0) = \prod_{i=1}^{p-1}(1 - \frac{i\hat{\rho}}{n}) \approx 1 \tag{14}$$

The above theoretical guarantees can as well be extended to TransH algorithm where the normal vectors to the hyperplanes also form the parameters and updates to them also can be shown to be sparse, in which case, the ParTransH will still achieve near optimal convergence.

3.2. HyTE

HyTE (Dasgupta et al., 2018) is a hyperplane-based method for learning temporally aware knowledge graph embeddings. This method exploits temporally scoped facts of KG to perform link prediction as well as prediction of time scopes for unannotated temporal facts. Given the knowledge graph G, based on the timestep $\tau = [\tau_s, \tau_e]$, G is divided into subgraphs $G_{\tau_1}, G_{\tau_2}, \ldots, G_{\tau_T}$ such that $G_{\tau_1} \cup G_{\tau_1} \cup \cdots \cup G_{\tau_T} = G$ The objective function still remains same as TransE but the score function changes as below:

$$\begin{aligned} e_{h_\tau} &= e_h - w_\tau^\top e_h w_\tau \\ e_{r_\tau} &= e_r - w_\tau^\top e_r w_\tau \\ e_{t_\tau} &= e_t - w_\tau^\top e_t w_\tau \end{aligned} \tag{15}$$

then the score function will be

$$f(h, r, t) = ||e_{h_\tau} + e_{r_\tau} - e_{t_\tau}||_{l_1/l_2} \tag{16}$$

3.3. ParHyTE

In ParHyTE, we define a 5-uniform hypergraph $H = (V, S)$ is now defined as follows:

Definition 3.2. *For a knowledge graph $G = (E, R, Q)$, a 5-uniform Hypergraph $H = (V, S)$ is constructed where $V = \{E \cup R \cup Q\}$ is the set of entities or relations or timelines, and S is the set of training samples s, where $s = \{(h, r, t, \tau, h') : h, t, h' \in E, r \in R, \tau \in Q\} \cup \{(h, r, t, \tau, t') : h, t, t' \in E, r \in R, \tau \in Q\}$*

For a hyperedge s in the hypergraph G,

$$\phi(s) = \{s' : \exists \tau \in s \cap s', \tau \in Q\} \tag{17}$$

denotes the set of hyperedges containing the same time window τ with hyperedge s,

$$\hat{\phi} = max_{s \in S}|\phi(s)| \tag{18}$$

denotes the set of hyperedges containing the same time window τ with hyperedge s. *Theorem 3.2* also applies in case of time windows when $\frac{\hat{\phi}}{n}$ is very small and $p < \frac{n}{\hat{\phi}} + 1$. Hence, we can say that, when the knowledge graph is sparse enough, we can parallelize the HyTE algorithm while losing minimal number of updates.

4. Dataset

For this project, I use the temporally rich subgraph, **Wikidata12k**, extracted from the Wiki dataset. It has 12.5k entities, 24 relations and 40k triples. Each triple has the starting year and the ending year. I split the triples into groups using the timestep parameter τ, i.e., for all the valid time windows, if the period between starting and ending

year of the triple overlaps with the time window, it belongs to that time window. For the sake of simplicity, each training example now is represented as a quadruple (h, r, t, τ_s) where τ_s is the starting year of the current time window. Depending on the timestep τ, the number of training examples also vary. The lower τ, the more number of training examples n and hence, lesser the number of collisions and hence faster convergence. At the same time, lower τ also means more number of parameters to train, which in contrast needs more training time. Hence, the parameter τ will be a hyperparamter to be tuned. Table 1 shows the number of training examples in case of few values of τ.

τ	n_e, n_r	T	#train (n)	#valid	#test
2000	12.5k,24	1	32k	4k	4k
100	12.5k,24	20	63k	6k	5.5k
25	12.5k,24	80	148k	11k	9.5k
10	12.5k,24	200	321k	22k	18k

Table 1. Wikidata size w.r.t. to τ.

5. Experiments

Figure 1 shows the convergence of loss function in case of ParTransE. It is clear that high speedup is achieved even with 5 workers. Figure 2 shows the convergence of loss function in case of ParHyTE. It converges slower than TransE.

Figure 3 shows the time taken to train the embeddings. Figure 5 contrasts the mean of head and tail hits10 for both the algorithms while figure 4 shows the mean ranks. All the results give empirical proof that parallelization doesn't affect the performance of the algorithm while converging faster.

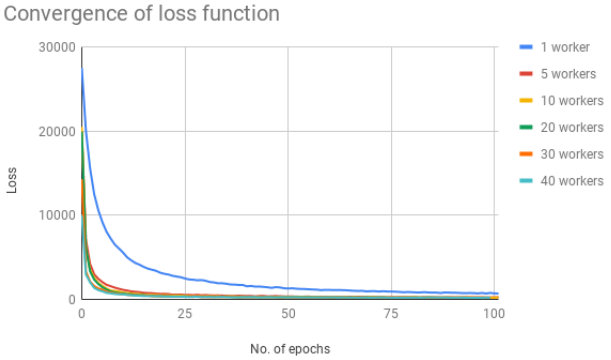

Figure 1. Convergence of loss function for multiple levels of parallelization for ParTransE.

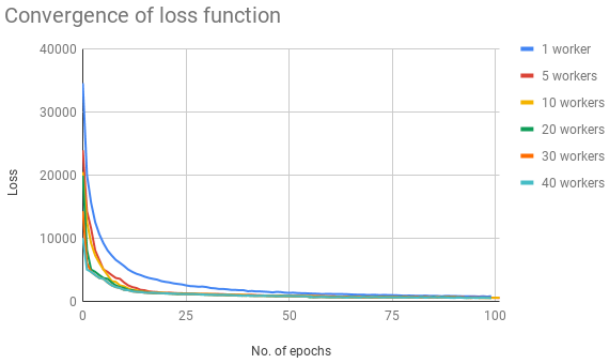

Figure 2. Convergence of loss function for multiple levels of parallelization for ParHyTE.

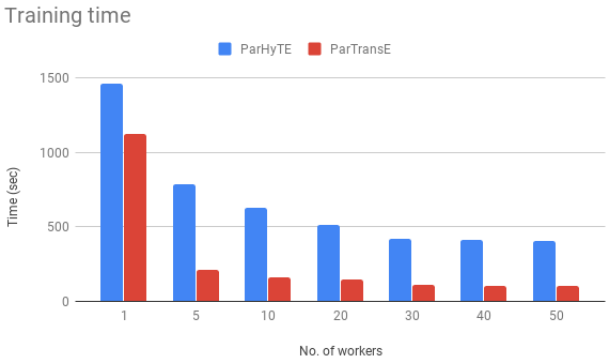

Figure 3. Training time in seconds for multiple levels of parallelization for ParHyTE and ParTransE.

6. Discussion

It is clear from the results that a speedup can be achieved by using ParHyTE while not compromising on the performance. However, the speedup achieved in TransE, **10**, is much higher than the speedup achieved in case of HyTE, **3.5**. This might be because of more number of parameters in HyTE and hence less sparsity of the training data.

7. Future Work

As future work, we can extend the same idea to many other dimensions of the training data. In this case, extra information can be used to learn the knowledge graph embeddings while still not compromising on the performance.

Model	Mean Rank		Hits@10		Training Time (min)	Speedup ($\frac{t_{serial}}{t_{parallel}}$)
	Head	Tail	Head	Tail		
TransE	338	303	0.28	0.41	~ 20	-
HyTE	574	224	0.24	0.41	~ 25	-
ParTransE	328	279	0.29	0.42	~ 2	10
ParHyTE	587	213	0.23	0.40	~ 7	3.5

Table 2. Comparative results on serial and parallel versions of TransE and HyTE. For Par*, the number of workers used is 50. And for ParHyTE, $\tau = 25$.

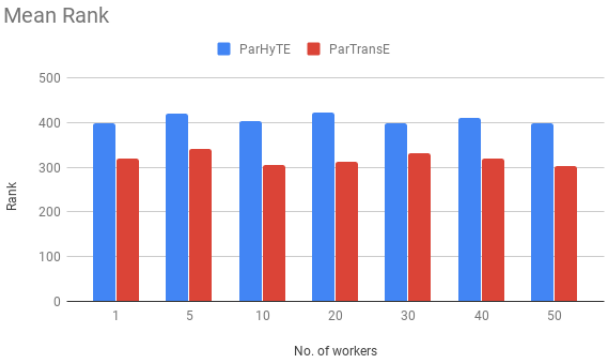

Figure 4. Mean Rank for multiple levels of parallelization for ParHyTE and ParTransE.

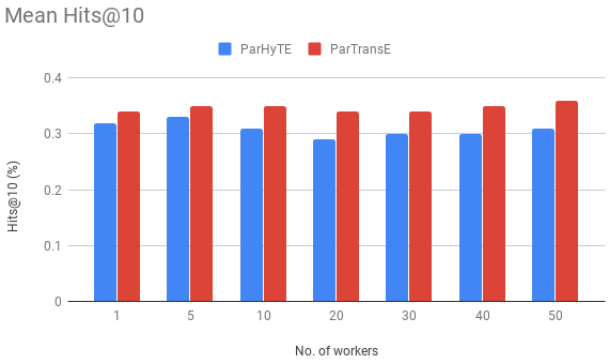

Figure 5. Means hits@10 for multiple levels of parallelization for ParHyTE and ParTransE.

References

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. Translating embeddings for modeling multi-relational data. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 26*, pp. 2787–2795. Curran Associates, Inc., 2013.

Dasgupta, S. S., Ray, S. N., and Talukdar, P. Hyte: Hyperplane-based temporally aware knowledge graph embedding. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2001–2011. Association for Computational Linguistics, 2018. URL http://aclweb.org/anthology/D18-1225.

Lin, Y., Liu, Z., Sun, M., Liu, Y., and Zhu, X. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, 2015.

Niu, F., Recht, B., Ré, C., and Wright, S. J. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011.

Wang, Z., Zhang, J., Feng, J., and Chen, Z. Knowledge graph embedding by translating on hyperplanes, 2014. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8531.

Zhang, D., Li, M., Jia, Y., and Wang, Y. Efficient parallel translating embedding for knowledge graph. In *IEEE/WIC/ACM International Conference on Web Intelligence 2017 (WI 2017)*, 2017.