Dynamic Strategies for High Performance Training of Knowledge Graph Embeddings

Anwesh Panda
Department of Computational and Data
Indian Institute of Science
Bangalore, India
anweshpanda@alum.iisc.ac.in

Sathish Vadhiyar
Department of Computational and Data
Indian Institute of Science
Bangalore, India
vss@iisc.ac.in

ABSTRACT

Knowledge graph embeddings (KGEs) are the low dimensional representations of entities and relations between the entities. They can be used for various downstream tasks such as triple classification, link prediction, knowledge base completion, etc. Training these embeddings for a large dataset takes a huge amount of time. This work proposes strategies to make the training of KGEs faster in a distributed memory parallel environment. The first strategy is to choose between either an all-gather or an all-reduce operation based on the sparsity of the gradient matrix. The second strategy focuses on selecting those gradient vectors which significantly contribute to the reduction in the loss. The third strategy employs gradient quantization to reduce the number of bits to be communicated. The fourth strategy proposes to split the knowledge graph triples based on relations so that inter-node communication for the gradient matrix corresponding to the relation embedding matrix is eliminated. The fifth and last strategy is to select the negative triple which the model finds difficult to classify.

All the strategies are combined and this allows us to train the ComplEx Knowledge Graph Embedding (KGE) model on the FB250K dataset in 6 hours with 16 nodes when compared to 11.5 hours taken to train on the same number of nodes without applying any of the above optimizations. This reduction in training time is also accompanied by a significant improvement in Mean Reciprocal Rank (MRR) and Triple Classification Accuracy (TCA).

CCS CONCEPTS

• **Computing methodologies** → **Parallel algorithms**; **Machine learning approaches**.

KEYWORDS

Knowledge graph embeddings, communication minimization, gradient quantization, selection of gradient vectors.

ACM Reference Format:
Anwesh Panda and Sathish Vadhiyar. 2022. Dynamic Strategies for High Performance Training of Knowledge Graph Embeddings. In *51st International Conference on Parallel Processing (ICPP '22), August 29-September 1, 2022, Bordeaux, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3545008.3545075

ICPP '22, August 29-September 1, 2022, Bordeaux, France

ACM ISBN 978-1-4503-9733-9/22/08…$15.00
https://doi.org/10.1145/3545008.3545075

1 INTRODUCTION

Knowledge Graphs (KG) are multi-relational graphs consisting of entities and relations. An instance of a KG is represented as a triple. Each triple consists of two entities and a relation between those two entities. They are popularly known as {head, relation, tail} or {subject, predicate, object}. An example can be {New Delhi, capital of, India} where {New Delhi} and {India} are entities and {capital of} is the relationship between them. Knowledge graph embeddings are vectorized representations of these entities and relations, and has various applications including triple classification, link prediction, knowledge base completion, etc.

With the increase in the size and sparsity of graphs, the time and memory requirements for training of Knowledge Graph Embeddings (KGE) increase. Hence there arises a demand for distributed training for exploring larger problems. There are two main approaches to distributed training. One is the parameter server [14] approach in which some nodes, known as servers, are used to store the model parameters. Data is distributed uniformly among the other nodes, known as worker nodes. In an iteration, the workers pull the updated model parameters and use the local data to compute gradients which are sent back to the servers to update the model parameters. The main drawback of this approach is the communication bottleneck to the server. To overcome this issue we can use more than one server. But this creates an all-to-all communication pattern that is not efficient.

To overcome the disadvantage of the above-mentioned approach, synchronous all-reduce and all-gather methods are proposed for smaller models. In this approach, data is uniformly distributed among the workers and the same model is replicated in every worker. Each worker uses its part of the data to compute gradients which are aggregated using either an all-reduce or an all-gather operation to update the model in each worker. The workers need to be synchronized at the end of each iteration. This method is well implemented in a framework named Horovod [17], which along with the deep learning library Tensorflow [1] is used in our work.

The above-mentioned approach has communication and synchronization bottlenecks. Another aspect of KGE training is the quality of the negative samples. Sampling more number of negative samples up to a certain threshold gives both convergence and generalization benefits, but requires more training time. Also with the increase in the number of nodes, the number of batches trained simultaneously and the effective batch size increases which not

only makes the training difficult but also brings down the generalization ability of the model. In this work, we address the above issues and try to reduce the training time while making a better generalization of the model.

In this work, we propose five methods to solve the above-mentioned challenges. The first method involves dynamically choosing between either an all-reduce or an all-gather method based on the number of non-zero gradient rows of the gradient matrix. The second method relies on the idea that the 2-norm of a gradient vector can be considered as a metric for the effective reduction in final loss. With this notion, we drop the gradient vectors with lower values of 2-norm by defining a Bernoulli random variable. The third method is to quantize the gradient values to 1-bit by using the sign of the gradient values and maximum of the absolute values in a gradient vector. Hence instead of communicating 32 bits for a gradient value, we communicate only 1 bit reducing the communication volume by a factor of 32. The fourth strategy is all about partitioning the triples among the nodes so that no two nodes have triples having the same relation. This helps us eliminate the inter-node communication for the relation gradient matrix. The fifth technique relies on the fact that KGE models can be efficiently trained with lesser but effective negative samples. We identify that the negative samples which the model finds difficult to classify are the most effective ones and use only them for training. This helps us reduce the number of triples for training which reduces the total training time and avoids the class imbalance problem due to the presence of more negative triples than positive ones. Experiments were conducted with the ComplEX[18] KGE on two datasets, namely, FB15K and FB250K datasets[7]. Our strategies put together help in training the model on the FB250K dataset in 6 hours on 16 nodes when compared to 11.5 hours it takes to train it on the same number of nodes without applying any of these optimizations. This performance improvement is also accompanied by an increase in accuracy parameters, namely, Mean Reciprocal Rank (MRR) and Triple Classification Accuracy (TCA). We also show huge performance improvement on the FB15K dataset in terms of training time, MRR and TCA.

The primary contribution of this work is to explore the combined effects of multiple important strategies for performance and accuracy of the KGE models. Strategies including dynamic selection of allgather and allreduce communication and relation partition are novel contributions of the work.

The paper is organized as follows. Section 2 deals with the works done related to the parallelization of deep learning models and KGE training. We also list the works undertaken to improve the convergence and reduce the adverse effects of large batch training. Section 3 introduces briefly the ComplEx [18] KGE model, details of the datasets and experimental setup including the hyper-parameters. This section then gives the baseline methodology and the corresponding results. Section 4 explains the proposed methods in detail with the supporting experiments. Section 5 presents the results and gives analysis. Section 6 gives conclusions and future work.

2 RELATED WORK

There have been some efforts by Zhang et al. [23] and Niu et al. [16] to train the KGE using shared memory parallelism by employing lock-free updates in a multi-threaded environment. Gupta and Vadhiyar [7] proposed three optimizations, namely, avoiding zero gradient row communication, variable margin approach, and dist-Adam optimizer to train TransE [19] embeddings efficiently using 32 nodes in a distributed memory paradigm. In our case, with the ComplEx KGE model[18], the number of gradient rows with absolute zero values is less. Hence we introduce a method to eliminate the insignificant gradient rows. PyTorch Big Graph [13] tried to split the graph into buckets and train the non-overlapping parts simultaneously without involving any communication between them. But, with their proposed techniques, the communication of entity embedding is reduced but not eliminated. We introduce a relation partition in which we eliminate the communication for the relation gradient matrix and also take advantage of gradient quantization.

A lot of work has been done on gradient sparsification and quantization to reduce the communication volume for training deep learning models. Aji et al. [2] proposed to communicate the gradient values whose magnitudes are greater than a certain threshold value. He also suggested storing the gradient values that are not communicated as residuals which are to be added with the gradient values in the next iteration. Wangni et al. [20] formulated the sparsification as an optimization problem to control the variance of the sparse gradient vector compared to its dense counterpart. Lin et al. [15] proposed to scale the locally accumulated gradient residuals with the momentum term in the case of momentum stochastic gradient descent. In our case, the size of each gradient vector is smaller (up to 200 dimensions), and communicating only a small percentage of the values can affect convergence. Also, in the other works, the indices of the data will have to be communicated, requiring large volume of communication.

The quantization method aims to reduce the number of bits to be communicated for a specific gradient value. Wen et al. [21] proposed terngrad which required three numbers to represent the gradient value and can be communicated using 2 bits. Alistarh et al [3] also proposed a 2-bit scheme called *q-sgd* in which they suggested quantizing the gradient vector by splitting it into multiple blocks. While q-sgd used the 2-norm of a vector along with its sign to represent a gradient value, terngrad suggested using the infinity norm. Wu et al. [22] suggested accumulating the error in the form of residual for the above-mentioned two methods and use it as feedback in the next iteration. Karimireddy et al. [9] suggested a 1-bit quantization method in which the sign of the gradient value is used along with the average of the gradient vector. The work also suggested accumulating the quantization error at every iteration and use it as feedback in the next iteration. We performed a comparison between various 1-bit and 2-bit quantization schemes and tried different techniques to assign the quantized values for a given gradient vector. Based on these comparisons, we choose the 1-bit quantization scheme with the sign of the gradient values multiplied with the maximum of the absolute value of the gradient vector as the quantized value.

Another approach is to factorize the gradient matrix to convert it to smaller ones. But factorization of a single matrix gives rise to at least two matrices which demand all-reduction of both the matrices. Cho et al. [5] proposed a method to use the same random matrix in all the processors and factoring the original matrix into the random matrix and one other matrix so that only one matrix needs to be all-reduced. Then the random matrix and the all-reduced matrix

can be used to reconstruct the approximate gradient matrix. As each row of the gradient matrix corresponds to different entities or relations in our case, reconstruction of the factored matrix does not seem intuitive and shows poor convergence in practice.

Keskar et al. [10] addressed the loss of generalization issue of large batch training, which can lead to convergence to a sharp minima. Goyal et al. [6] proposed strategies like linear scaling and progressively increasing the learning rate to overcome the large batch training issue. For linear scaling, the learning rate is multiplied by the number of processors. However, in our experiments with large number of processors, we observed that such linear scaling resulted in reduced validation accuracy. This is due to the high learning rate caused by the divergence from the minima. Hence, we fixed an upper bound for the learning rate. This method successfully solves the issue of large batch training to a large extent.

Kotnis et al. [12] studied the effect of negative sampling and proposed the nearest neighbor negative sampling which shows a good improvement in performance. The basic idea is to take the sample which is difficult to classify for the model. For a given head and relation, the work takes the tail entity which is closest to the positive tail entity in the embedding space. However, the approach is computationally expensive. In our approach, we take k negative samples randomly for a given positive sample and use the one which the model finds difficult to classify by using the scores assigned by the model.

3 BACKGROUND AND ANALYSIS

3.1 KGE models

Knowledge Graphs (KGs) are multi-relational graphs consisting of entities and relations. An instance of a KG is represented as a triple. Each triple consists of two entities and a relation between those two entities. They are popularly known as {head, relation, tail} or {subject, predicate, object}. An example can be New Delhi, capital of, India where New Delhi and India are entities and capital of is the relationship between them. Knowledge graph embeddings are vectorized representations of these entities and relations and KGEs can be used for triple classification, link prediction, knowledge base completion, etc.

KGE models define a scoring function for a correct triple and an incorrect triple. In this work we use complex model [18], since complex models are important classes of models used in many applications and generalize other models including real models [7]. Strategies developed with these models are generally applicable to the other models. In our work, out of the five strategies, all the strategies, except the negative sample selection strategy, are applicable to the other KGE models.

In the complex model, instead of using real embedding vectors, complex embedding vectors are used. The score function is defined as the real part of the element-wise product of head embedding, relation embedding, and conjugate of tail embedding.

$$\begin{aligned}\phi(h,r,t) &= Re(< E_h, E_r, \overline{E}_t >) \\ &=< Re(E_r), Re(E_h), Re(E_t) > \\ &+ < Re(E_r), Im(E_h), Im(E_t) > \\ &+ < Im(E_r), Re(E_h), Im(E_t) > \\ &- < Im(E_r), Im(E_h), Re(E_t) >\end{aligned} \tag{1}$$

The loss function is defined as

$$\min_{\theta} \sum_{h,r,t} log(1 + exp(-Y_{h,r,t}\phi_{h,r,t})) + \lambda||\theta^2||$$

where $Y_{h,r,t} = 1$ for a correct triple and -1 for an incorrect or false or negative triple. The negative triples are obtained by randomly replacing either head or tail entity of a correct triple with some other entity.

3.2 Evaluation Metrics

We follow standard evaluation techniques as followed by ComplEx [18], namely, filtered-MRR (filtered Mean Reciprocal Rank) and triple classification accuracy. For calculating raw MRR, first, we replace the head with every other entity in the dataset and find the rank of the original triple based on the score. We repeat the same process by replacing the tail. We take the reciprocal of the rank and take the average of both. We do the same for all the test triples and take an average. In the case of filtered MRR, we skip the triples which are already present in the dataset. The triple classification accuracy is obtained by evaluating if a given test triple is correct or not. We use the well-known library OpenKE [8] for modeling and evaluation.

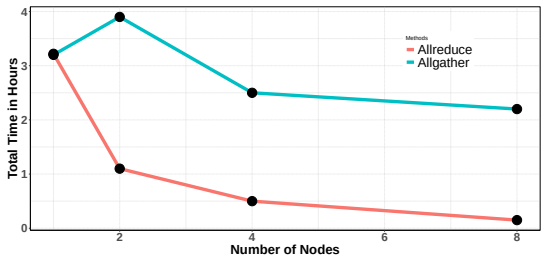
3.3 Experimental Setup

We performed experiments on two datasets i.e. FB15K [4] and FB250K [7]. Both these datasets are created by skimming the original Freebase dataset which consists of around 2 billion triples. FB15K consists of 14951 entities and 1345 relations and approximately 600K triples. FB250K has around 16 million facts, 240K entities, and 9280 relations. Although the FB15K dataset is a smaller one, it is used as a benchmark dataset in almost all the KGE models. Hence, we use it to show the efficacy of our techniques in terms of generalization and accuracy. FB250K dataset is used to show the scalability of our approaches. We use the same train, test, and validation splits which was created by the original creator of the datasets. All our results were obtained as average over five runs.

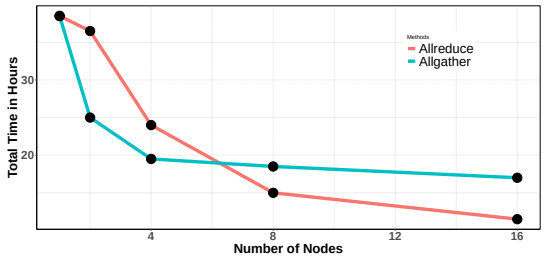
We performed our experiments with equal number of batches per worker with batch-size of 10000 and using Adam optimizer [11]. The experiments were performed on a CrayXC40 supercomputing system in our Institute. We experimented with maximum of 16 nodes where each node has 2 CPU sockets with 12 cores and 128GB RAM. Hence the calculation in a node is multi-processed by 24 cores. We start with an initial learning rate of 0.001 scaled by a scaling factor which we define for different methods and with a tolerance of 15, reduce it by a factor of 0.1 until a defined minimum learning rate. This means that if we do not see any improvement in validation accuracy until 15 epochs, we decrease the learning rate. We also specify the number of negative samples for each method.

3.4 Baseline Method

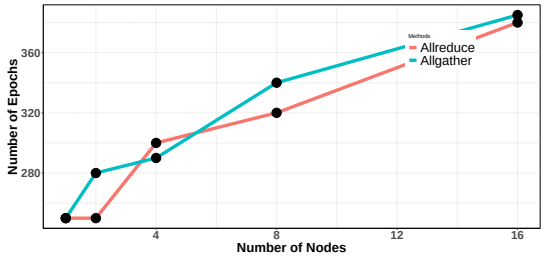
We consider ComplEx [18] KGE model parallelized with the distributed library Horovod [17] with both sparse and dense updates of the gradient matrix. The dense updates involve performing an all-reduce operation which involves communicating the whole gradient matrix including the zeros so that values at each row and column get aligned to perform the addition operation. On the other hand, sparse updates involve the all-gather operation where only



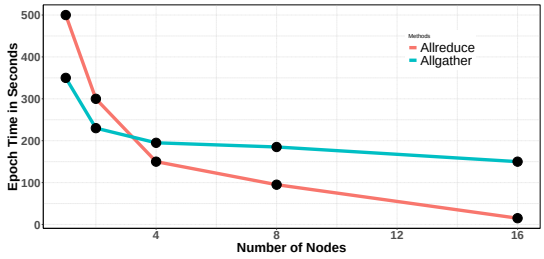
(a) TT on FB15K



(b) TT on FB250K



(c) N on FB250K



(d) epoch time on FB250K

Figure 1: Baseline Results

the non-zero rows of a gradient matrix are communicated and gathered by all the processors. We first used the linear scaling rule to scale the learning rate which is to simply multiply it by the number of nodes. But it resulted in highly unstable training because of the higher learning rate for the number of nodes greater than four. Hence we take the maximum scaling factor as 4. Formally, $lr = lr \times min(4, number\ of\ nodes)$. The number of negative samples for FB250K is 1 and for FB15K is 10 per one positive triple.

The baseline results are shown in Tables 1 and 2 and Figure 1. For FB15K, all-reduce always results in smaller execution times than all-gather primarily because it is a small dataset and the gradient embedding matrix is small because of which the sparsity is less.

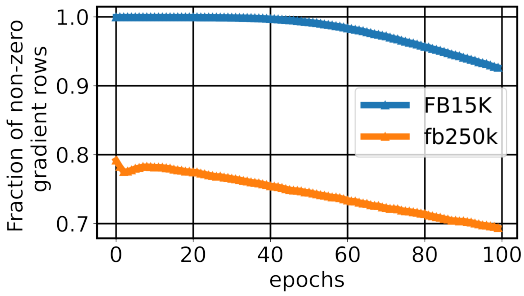


Figure 2: Number of non-zero gradient rows

Figure 1b shows the total training time for the FB250K dataset. Initially, training time for all gather seems to be less than all-reduce until the number of nodes equals 4 which can be explained by the lesser epoch time as shown in Figure 1d. However, after 4 nodes the situation reverses, and all-reduce performs better.

In Figure 1c, we can see the increase in the number of epochs with the increase in the number of nodes. With the increase in the number of nodes, the time taken for a full epoch decreases but the number of epochs required for convergence increases due to an increase in the effective batch size. In Figure 1b, we can see the saturation of total time after the number of nodes equals 4. This is because with the increase in the number of nodes, reduction in time per epoch is less whereas the number of epochs for convergence increases giving rise to higher total time. This lesser reduction in epoch time is due to the increase in all-gather volume with an increase in the number of nodes creating a communication bottleneck, as explained in the following section.

4 OPTIMIZATIONS

This section discusses our proposed methods to improve the baseline results. The objective is to reduce the total training time by reducing communication and improving the convergence rate without compromising accuracy.

4.1 Dynamic Selection of AllReduce and AllGather Communications

We find two methods for accumulating the gradients for all the workers, namely, all-reduce and all-gather operations. For an all-reduce operation, every worker must have the full gradient matrix communicated between them so that elements at a given index can be added across all the workers. This is beneficial when we have a dense matrix. But with more sparsity, it may not be beneficial to work with the full gradient matrix but to gather the rows of the matrix among all the workers. In this work, we try to dynamically choose between either an all-reduce or an all-gather operation based on the sparsity. As shown in Figure 1d related to the time taken for one epoch for all-gather and all-reduce operation for varying number of nodes, for a smaller number of nodes, an all-gather operation gives better performance while for higher number of nodes all-reduce gives better performance.

Figure 2 shows that with an increase in the number of epochs as we go along training the model, the number of non-zero gradient

	all-reduce				all-gather			
No of nodes	TT	N	TCA	MRR	TT	N	TCA	MRR
1	3.26	301	90.7	0.59	3.26	301	90.7	0.59
2	1.27	257	90.2	0.57	3.52	358	90.6	0.59
4	0.78	300	90.3	0.58	2.48	349	90.3	0.58
8	0.54	381	90.3	0.58	2.34	314	90.1	0.56

Table 1: Baseline result on FB15K dataset. TT refers to total training time in hours, N refers to number of epochs, MRR refers to Mean Reciprocal Rank, TCA refers to Triple Classification Accuracy

	all-reduce				all-gather			
No of nodes	TT	N	TCA	MRR	TT	N	TCA	MRR
1	37.2	250	89.6	0.28	37.2	250	89.6	0.28
2	35.3	252	89.6	0.28	26.3	283	89.9	0.28
4	24.04	302	89.6	0.28	19.6	298	89.7	0.28
8	14.3	323	89.5	0.29	17.53	339	89.1	0.28
16	11.3	379	88.5	0.28	16.1	386	88.5	0.28

Table 2: Baseline result on FB250K dataset

rows decreases which motivates us to point towards an all-gather operation. Hence we start the first epoch with an all-reduce operation. At every k^{th} epoch, we perform an all-gather operation and compare the time taken for both. If the time taken for all-gather operation is less, we switch to all-gather for the rest of the training. Else, we continue to perform all-reduce. We set k to be 10 in our case.

4.2 Selecting the Gradient Vectors

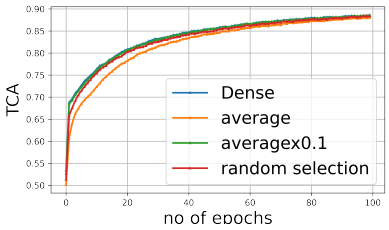
The above-proposed optimization heavily relies on the fact that a certain number of gradient vectors needs to be completely zero which is not the case for most of the datasets. It also takes a lot of training to reach that state. This motivates us to introduce sparsity by selecting the gradient vectors which contribute more towards the decrease in the overall loss. The change in loss heavily depends on the change in parameter vector which in turn relies on the magnitude of gradient vectors. Hence we use the 2-norm of the gradient vectors to decide the contribution of each gradient vector.

Now the task is to decide a certain threshold below which we can drop a gradient vector. One method is to set the average of the 2-norms of the gradient vectors as a threshold. This simply means we will drop all the gradient vectors whose 2-norm will be lesser than the average value. By this process, we may miss the vectors which are just below the threshold but have the potential to cause a significant impact. Hence, we design this task with the help of the Bernoulli random variable for each gradient vector. If the variable takes 0 values then we drop that gradient vector. Formally, if X_i is the random variable corresponding to the i^{th} gradient vector

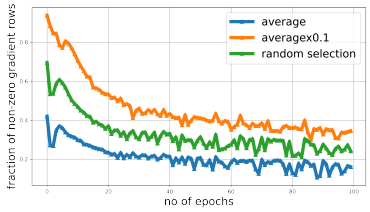
$P(X_i = 1) = min(1, \frac{||X_i||_2}{C})$

where C = average of the 2-norms of all gradient vectors

Figure 3a shows the triple classification accuracy along with sparsity for 3 scenarios. The average scenario is when we take the average as threshold and averagex0.1 is when we multiply 0.1 with the average value and take it as a threshold. The second method is because we observed that only taking the average results in



(a) Comparing Based on TCA



(b) Comparing based on sparsity

Figure 3: Different thresholds for random selection

skipping of many gradient vectors. By multiplying the average by 0.1, we attempt to reduce this skipping by a factor of 10. The third method is random selection which is as described earlier. From Figure 3a we can see that the convergence curve for the random selection method almost overlaps with its dense counterpart and also introduces a good amount of sparsity as shown in Figure 3b, which results in reduced communications.

4.3 Gradient Quantization

Gradient quantization involves reducing the number of bits to represent the gradient values. In this work, we focus on 2-bit and 1-bit quantization to represent a 32-bit scalar gradient value.

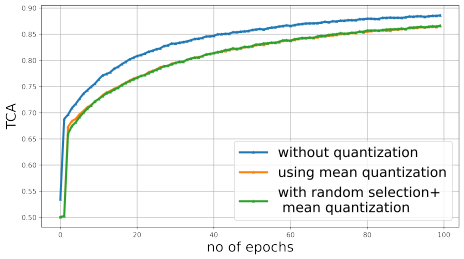
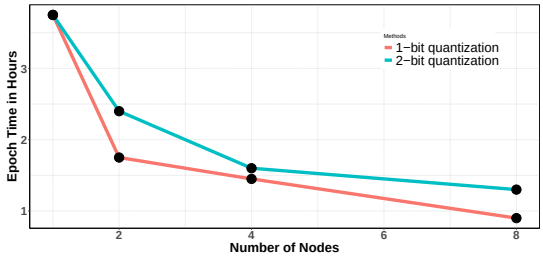


Figure 4: 2-bit quantization with random selection on FB15K dataset

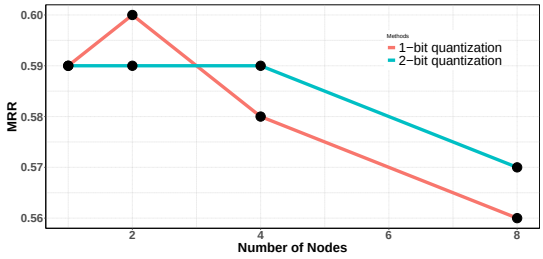
Our 2-bit quantization scheme is motivated by TernGrad [21] with some minor changes. Instead of using the maximum of the absolute values of the gradient vector, we use the mean of the absolute values as we found experimentally that it gives better results. Our 2-bit quantization method can be summarized as follows.
$quant(v) = sign(v) * mean(|v|) * P$
where v is the original gradient vector.
sign(v) represents the sign of the elements of vector v
P is a binary random vector with
$P(P_i = 1) = min(1, \frac{|P_i|}{mean(|v|)})$
Fig. 4 shows the effect when we merge random selection of gradient vectors with 2-bit quantization on the FB15K dataset. We can see that the performance of the 2-bit quantization scheme is not affected by the addition of random selection.

In the 1-bit quantization scheme, we eliminate the zeros which we introduce in the 2-bit quantization scheme. We explored different methods including average, max, negmax, posmax, negavg, posavg. For example, negmax is when we represent negative values of the gradient vector by the maximum of all the negative values. We found experimentally that the max method, where we take the maximum of the absolute values of all the elements, outperform the other methods in terms of accuracy. The 1-bit method can be summarized as follows.
$quant(v) = sign(v) * max(|v|)$

The results for 2-bit and 1-bit gradient quantization with random selection are given in Figure 5 for the FB15K dataset. Note that for the FB15K dataset we do not include dynamic allreduce-allgather along with random selection as all-reduce seems to always perform the best for this dataset due to the smaller gradient matrices. Figure 5a shows the comparison of total training time between 2-bit and 1-bit gradient quantizations embedded with random selection. The 1-bit quantization strategy seems to be more beneficial in terms of the total training time while the accuracy metrics of MRR, as shown in Figure 5b remain almost the same for both the cases. Hence for our further experiments, we use 1-bit gradient quantization scheme with the sign of the gradient value multiplied with the maximum of absolute values of the gradient vector. We found that after the introduction of the gradient quantization, the fraction of Allreduce communications reduced by about 60%.



(a) Comparison of total training time



(b) Comparison of MRR

Figure 5: Comparison between 1-bit and 2-bit quantization on FB15K

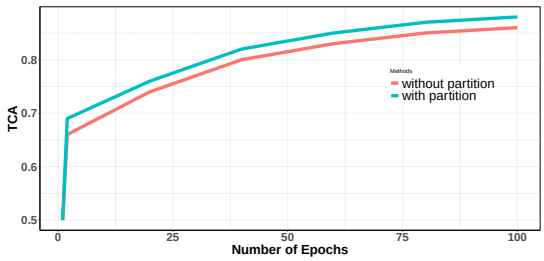
S.N.	head	relation	tail
1	1	1	2
2	2	1	10
3	3	2	5
4	6	3	9
5	7	3	8

Table 3: Triples

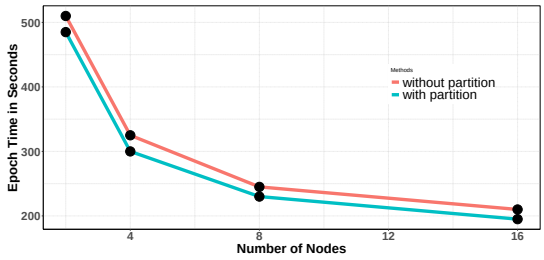
4.4 Relation Partition

The primary operations in the KGE algorithms is to collect the gradient vectors in each processor by either an all-gather or an all-reduce operation to obtain the global gradient matrix in each processor and update the model. There are two gradient matrices in each processor, namely, entity gradient matrix and relation gradient matrix. We propose to arrange the triples in such a way that the communication required to accumulate the gradients is minimized. There are some efforts [13] to partition entities to minimize communication overhead. Instead of partitioning the entities, we focus on partitioning the relations. In our approach, we eliminate the need of accumulating the gradient matrices for relation embeddings in each processor, thus eliminating the communication overhead. This is done by assigning triples corresponding to non-overlapping relations to different processors. Let us consider the below scenario where we have 5 triples (Table 3) to be distributed across two processors.

For this case, if we assign the first and second triples to processor-1 and the rest to processor-2, the relations in the two processors do not overlap. Hence we do not need to communicate the gradient matrices corresponding to the relation embeddings.



(a) Comparing TCA on FB15K



(b) Comparing epoch time on FB250K

Figure 6: Comparing with and without relation partition

Hence, we first sort the triples based on the relations. Then we construct an array representing the number of triples corresponding to each relation. Then we perform a prefix sum over the array. For p processors, we have to perform p-splits so that the number of triples in each processor remains balanced. This can be done by performing a binary search over the prefix array to find the range of relations for each split which are then scattered across the processors.

The relation gradient matrix is smaller when compared to the entity gradient matrix and hence the timing benefits obtained with this method may not be significant. But the benefit of this method is realized with gradient quantization. Since we do not need to communicate the relation gradient matrix, we also do not need to quantize the values and hence the values can be used with full precision. This gives rise to improvement in convergence and accuracy for the gradient quantization methods.

We had earlier found that random selection with 1-bit gradient quantization is effective in reducing the training time. Here, we explore the impact of relation partition on the method. Figure 6a shows the improvement in convergence with the addition of our relation partition method to the scheme of 1-bit gradient quantization plus random selection. In Figure 6b, although initially the reduction in time was not significant, with the increase in the number of nodes, large reductions in times were obtained.

4.5 Negative Sample Selection

For ComplEx[18] KGE, the number of negative triples dictates the accuracy and convergence. From the original ComplEx paper[18], it can be seen that the accuracy (MRR) and convergence improve till a certain number of negative triples per one positive triple. Beyond this number of negative triples, both the accuracy and convergence show degradation. However, a large number of negative triples increases the computation cost. The objective is to use those negative triples which will be effective in increasing accuracy and convergence. These are typically the negative triples which the model finds difficult to classify.

Sample Ratio	TT(in hours)	N	MRR	TCA
1 out of 1	0.41	423	0.523	89.3
1 out of 5	0.66	240	0.59	90.53
1 out of 10	0.775	229	0.61	90.7
1 out of 20	0.97	210	0.629	90.74
1 out of 30	1.06	187	0.63	90.8
5 out of 5	1.29	390	0.585	90.5
10 out of 10	2.1	344	0.592	90.5

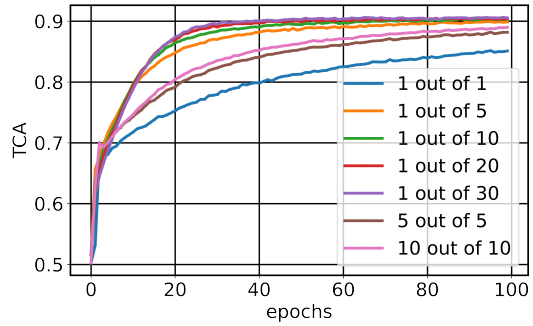
Table 4: Sample selection with 1-bit gradient quantization on 2 nodes

Formally, from the loss function it can be seen that the model tends to give high positive scores for the positive triples and high negative scores for a negative triples. So the negative triple which has the least negative score is the one that needs to be considered. In our method, we sample n negative samples for a given positive triple. Then we perform a forward pass to find out the scores for all the n negative triples. Then we find the triple with the least negative score and use it for training. Another point to note here is that a forward pass is far less expensive than the backward pass where the gradients need to be computed. When there are multiple negative samples for one positive sample, there may be a chance of a class imbalance problem which is successfully avoided in our case.

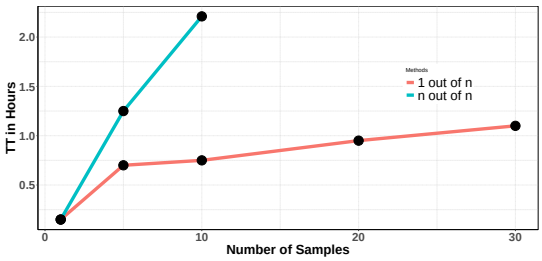
Table 4 shows the training details for different negative sample selection on the FB15K dataset based on which the plots in Figure 7 are drawn. Figure 7a shows the comparison between different sample selection ratios. In the figures, *m out of n* means that n negative samples are drawn out of which m are used for training. From the plot, it is visible that even for 1 out of 5, the convergence is better than that of 10 out of 10. This may be due to the avoidance of the class imbalance problem caused by the use of more than one negative sample per positive sample. Hence, we decide to sample one negative sample and use the best one for training. From Figure 7c, we observe that with the increase in n, MRR improves but not in a proportional manner and tends to saturate after a certain number. From Figure 7b, we can see that the training time increases due to more sample scores required for the forward pass. But the increase in time is far less than that of sampling and training n out of n samples as gradient computation of $n-1$ negative samples per positive sample is avoided. Also from Figure 7d, the number of epochs required for convergence is lesser for 1 out of n sample selection and it further decreases with the increase in n.

5 EXPERIMENTS AND RESULTS

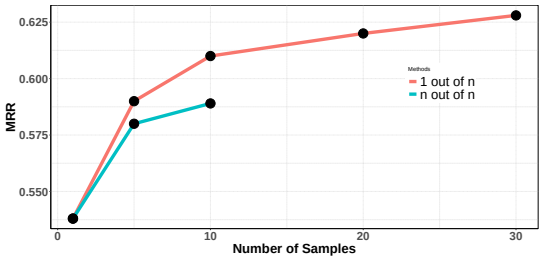
We presented results for the various methods for both FB15K and FB250K datasets. Note that we have used the same experimental setup as given in Section 3. The additional point here is to note that the ratio for the Sample Selection strategy is 1:10 in the case of



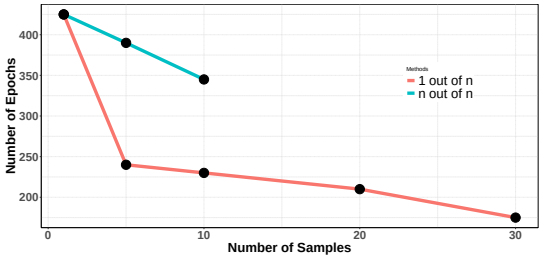
(a) Comparison of Convergence



(b) Comparison of Total Training Time



(c) Comparison of MRR



(d) Comparison of Number of Epochs Required for Convergence

Figure 7: Comparison between two sampling schemes on FB15K dataset

FB15K dataset and 1:5 for FB250K dataset. As seen in Table 4, for the FB15K dataset, the ratio of 1:10 gave near-maximum accuracy with the least training time. Hence this ratio was chosen for the FB15K dataset. We chose a ratio of 1:5, i.e., smaller number of negative samples for the FB250K dataset due to the larger size of the data and the resulting larger training time involved. In general, the values for the ratio can be tuned as a hyper-parameter based on the size of the data and the training time required. For the results obtained without the Sample Selection strategy, the number of negative samples per positive sample is 10 in the case of the FB15K and 1 in the case of the FB250K dataset. Table 5 gives the different terms and parameters considered.

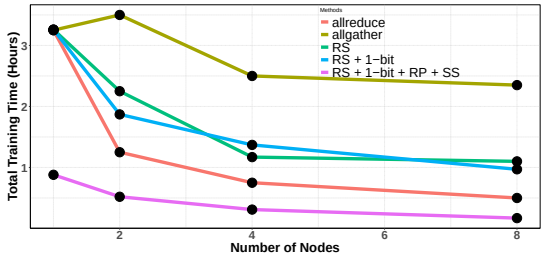
N	Number of epochs
TT	Total training time in hours
MRR	Mean Reciprocal Rank
TCA	Triple Classification Accuracy
RS	Random selection of gradient vectors
DRS	Dynamic all-gather reduce along with RS
RS+1-bit	RS along with 1-bit gradient quantization
DRS+1-bit	DRS along with 1-bit gradient quantization
RS+1-bit+RP+SS	RS+1-bit along with Relation partition and Sample Selection
DRS+1-bit+RP+SS	DRS+1-bit along with Relation partition and Sample Selection

Table 5: Terms and Parameters

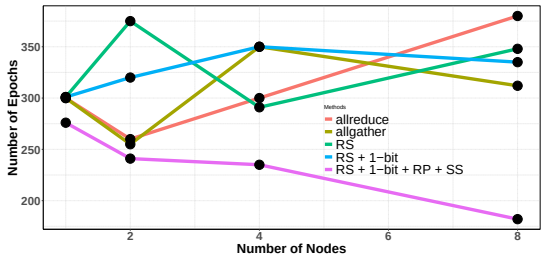
5.1 Results for FB15K Dataset

For the FB15K dataset, we have not included the dynamic gather-reduce optimization as all-reduce seems to be always faster in the case of a small dataset. We use the FB15K dataset primarily for the analysis of accuracy. Figure 8 shows the results.

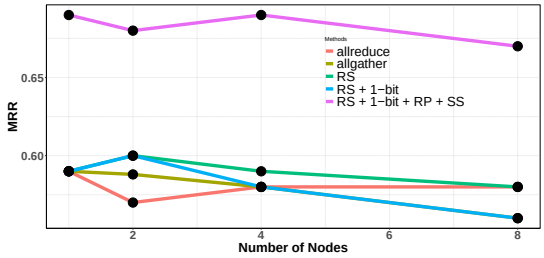
From Figure 8c, one can see that the MRR remains almost the same in the random selection (RS) method when compared to the baseline, but with the introduction of 1-bit quantization with a higher number of nodes (in this case 8), it degrades to 0.56. In all-reduce it is 0.58 causing 3.4% reduction in MRR. The results are obtained for 10 negative samples per positive sample in the case of RS and RS+1-bit. In the case of RS+1-bit+RP+SS, we keep the ratio of sample selection as 1 to 10 (10 sampled and best one is chosen for training) and obtain huge benefits with both relation partition and sample selection methods, which drastically reduces the training time (Figure 8a) while increasing MRR (Figure 8c). When compared to the all-reduce baseline, the maximum reduction for training time is 73% for a single node execution whereas the minimum reduction is 59% for two-node execution. Similarly, when compared to the all-gather baseline, the maximum reduction for training time is 92.7% for an eight-node execution whereas the minimum reduction is 73% for single-node execution. The training time is even lesser than all-reduce for all the nodes. We did not observe any specific pattern for the number of epochs with the increase in the number of nodes. From Figure 8b, it can be observed that the number of epochs required for RS+1-bit+RP+SS is always lesser than that for the other methods. We also obtain an improvement in MRR when compared to the baseline with a minimum increment of 15% over the all-reduce baseline for an eight-node execution and a maximum increment of 19% over the all-gather baseline for an eight-node execution.



(a) Comparison of Total Training time(in hours)

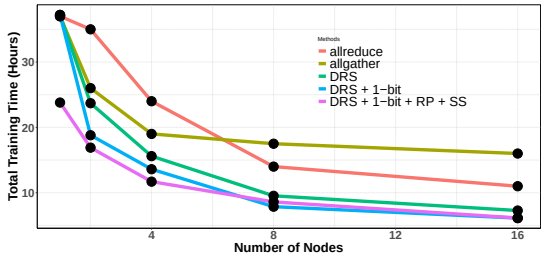


(b) Comparison No of epochs for convergence

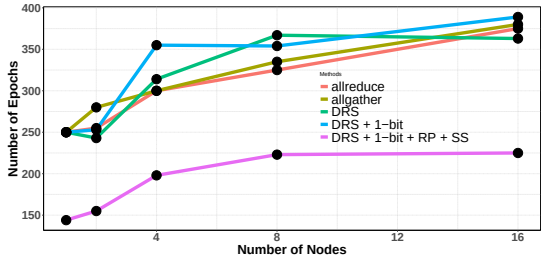


(c) Comparison of MRR

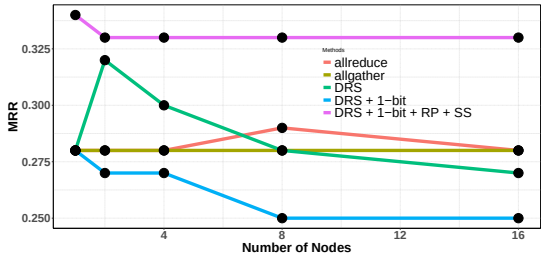
Figure 8: Comparison of Different Methods on FB15K dataset



(a) Comparison of Total Training time(in hours)



(b) Comparison No of epochs for convergence



(c) Comparison of MRR

Figure 9: Comparison of Different Methods on FB250K dataset

5.2 Results for FB250K Dataset

Figure 9 shows the results for theFB250K dataset. Figure 9a shows that all the three methods, namely, DRS, (DRS+1-bit), (DRS+1-bit+RP+SS), result in smaller training times than the baseline. For smaller number of nodes (DRS+1-bit+RP+SS) (with a sample selection ratio 1 to 5) is the clear winner whereas for larger number of nodes its performance is similar to (DRS+1-bit) and is the lowest among all the methods in terms of the training time. When compared to the all-reduce baseline, the maximum reduction for training time is 52% for a two-node execution whereas the minimum reduction is 36% for a single node execution. Similarly, when compared to the all-gather baseline, the maximum reduction for training time is 61% for an eight-node execution whereas the minimum reduction is 35% for a two-node execution. From Figure 9b, it can be noticed that the total number of iterations increases with the increase in the number of nodes which is due to the increase in effective batch size. This is one of the main reasons why we do not get a strong scaling. Overall the number of epochs required is far lesser in the case of (DRS+1-bit+RP+SS) due to the introduction of relation partition and sample selection leading to better convergence. The maximum reduction in the number of epochs was 45% and is obtained for a 2-node execution and the minimum reduction of 31% is obtained for a four-node execution when compared to the all-reduce baseline. In Figure 9c, it can be seen that the methods DRS and (DRS+1-bit) perform either similarly or worse in terms of MRR when compared to the baseline. With the increase in the number of nodes, their performance degrades even further. The reduction of MRR is a maximum of 10.35% for (DRS+1-bit) with eight and sixteen node executions. But with the introduction of relation partition and sample selection, MRR increases and remains fairly constant with the increase in the number of nodes. The maximum increment in MRR is 21% for a single node execution when compared to both the all-reduce and all-gather baselines and the minimum increment is 13% for an eight-node execution when compared to the all-reduce baseline.

5.3 Summary of Results and Discussions

From the results, it is clear that the combined method (DRS+1-bit+RP+SS) outperforms all the others both in terms of the training times and accuracy. The time taken for an epoch is reduced by introducing Random Selection and 1-bit Gradient Quantization

which in turn reduce the communication volume causing the reduction in communication time. Relation Partition also contributes to this reduction as it eliminates communication of relation gradient matrix and hence the communication time. The Sample Selection method improves convergence, reducing the number of iterations for the convergence. This reduction in the number of iterations is accompanied by an increase in the epoch time required for an extra forward pass which in the worst case may cancel out the benefits obtained. But, the Sample Selection method helps in the increase of MRR. The convergence and accuracy benefits are also obtained from the Relation Partition method which allows us to use full precision values for the relation gradient matrix. On an average, we obtain a 44.95% reduction in the total training time and 17.5% increase in MRR in the case of the FB250K dataset and a 65.2% reduction in total training time and 17.7% increase in MRR in the case of the FB15K dataset.

6 CONCLUSIONS AND FUTURE WORK

We have presented five methods to reduce the training time and increase the MRR and TCA. We have presented the results for the FB250K dataset up to 16 nodes and shown scalability of our methods. We have also shown results for the FB15K dataset on the convergence and accuracy for the ComplEx [18] KGE model. Among our methods, dynamic all-reduce all-gather decides dynamically between all-reduce and all-gather operation and picks the one which will give the maximum timing benefit. Random Selection and Gradient Quantization reduce the communication volume and hence reduces the communication time overhead. Relation partition not only contributes to the reduction in time by eliminating the communication of relation gradient matrix but also contributes to the convergence in the case of gradient quantization. The Sample Selection method not only increases the MRR and TCA but also brings down the number of iterations for convergence with a small overhead of an extra forward pass. On an average, we obtain a 44.95% reduction in total training time and 17.5% increment in MRR in the case of the FB250K dataset and a 65.2% reduction in total training time and 17.7% increment in MRR in the case of the FB15K dataset. As future work, we would like to explore our methods with other KGE models on different datasets.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 265–283.

[2] Alham Fikri Aji and Kenneth Heafield. 2017. Sparse communication for distributed gradient descent. *arXiv preprint arXiv:1704.05021* (2017).

[3] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2016. QSGD: Communication-efficient SGD via gradient quantization and encoding. *arXiv preprint arXiv:1610.02132* (2016).

[4] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Neural Information Processing Systems (NIPS)*. 1–9.

[5] Minsik Cho, Vinod Muthusamy, Brad Nemanich, and Ruchir Puri. 2019. GradZip: Gradient Compression using Alternating Matrix Factorization for Large-scale Deep Learning.

[6] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. 2017. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677* (2017).

[7] Udit Gupta and Sathish Vadhiyar. 2019. Fast and Accurate Learning of Knowledge Graph Embeddings at Scale. In *2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. IEEE, 173–182.

[8] Xu Han, Shulin Cao, Xin Lv, Yankai Lin, Zhiyuan Liu, Maosong Sun, and Juanzi Li. 2018. Openke: An open toolkit for knowledge embedding. In *Proceedings of the 2018 conference on empirical methods in natural language processing: system demonstrations*. 139–144.

[9] Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian Stich, and Martin Jaggi. 2019. Error feedback fixes signsgd and other gradient compression schemes. In *International Conference on Machine Learning*. PMLR, 3252–3261.

[10] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. 2016. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836* (2016).

[11] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[12] Bhushan Kotnis and Vivi Nastase. 2017. Analysis of the impact of negative sampling on link prediction in knowledge graphs. *arXiv preprint arXiv:1708.06816* (2017).

[13] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. 2019. Pytorch-biggraph: A large-scale graph embedding system. *arXiv preprint arXiv:1903.12287* (2019).

[14] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*. 583–598.

[15] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. 2017. Deep gradient compression: Reducing the communication bandwidth for distributed training. *arXiv preprint arXiv:1712.01887* (2017).

[16] Xiao-Fan Niu and Wu-Jun Li. 2017. ParaGraphE: a library for parallel knowledge graph embedding. *arXiv preprint arXiv:1703.05614* (2017).

[17] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* (2018).

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

[19] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 28.

[20] Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. 2017. Gradient sparsification for communication-efficient distributed optimization. *arXiv preprint arXiv:1710.09854* (2017).

[21] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2017. TernGrad: Ternary gradients to reduce communication in distributed deep learning. *arXiv preprint arXiv:1705.07878* (2017).

[22] Jiaxiang Wu, Weidong Huang, Junzhou Huang, and Tong Zhang. 2018. Error compensated quantized SGD and its applications to large-scale distributed optimization. In *International Conference on Machine Learning*. PMLR, 5325–5333.

[23] Denghui Zhang, Manling Li, Yantao Jia, Yuanzhuo Wang, and Xueqi Cheng. 2017. Efficient parallel translating embedding for knowledge graphs. In *Proceedings of the International Conference on Web Intelligence*. 460–468.