DS222: Machine Learning with Large Datasets

ParConvE: Parrallel Convolutional 2D Knowledge Graph Embeddings for Link Prediction at Scale

Instructor

Prof. Partha Pratim Talukdar

Candidate

Nilesh Agrawal(15007)
Nihar Sahoo(14739)

December 12, 2018

Problem Statement

Problem Statement:

- To generate latent vector representations for large scale knowledge graphs by implementing KGC method using scalable architecture to generate embeddings using distributed deep learning.

- Incorporate textual/literal information while learning embedding as to generate robust graph embeddings, which increases efficiency of link prediction and other downstream prediction tasks.

- Compare the performance of evaluation metrics: **MRR, MR, Hits@k** and running time for different data parallel distributed strategies and compare their **speedup** w.r.t serial implementation.

Problem Statement (Contd...)

- We have implemented three KGE methods - ParConvE, ParDistMult and ParLiteralE and compared them for evaluation metrics and speedup using :
  - Parameter server strategy (Sync, SSP, Async)
  - Ring All-Reduce architecture
- As told in earlier presentation to use some large dataset, so we have used DB100K dataset in this work, which was released in Jun, 2018, for which speedup results are pretty good.

Related Work

- A knowledge graph $G = \{(se, r, oe)\}$ is a set of triples.
- The link prediction problem can be formalised as a point-wise learning to rank each triple, where the objective is learning a scoring function $f_r(se, oe) : E * R * E \Rightarrow \mathbb{R}$.

Related Work:

- TransE: Translating Embeddings for Modeling Multi-relational Data - Bordes et. al.
- DistMult: Embedding Entities and Relations for Learning and Inference in Knowledge Bases - Yang et. al.
- ConvE: Convolutional 2D Knowledge Graph Embeddings - Dettmers et. al.
- ParTransX: Efficient Parallel Translating Embedding For Knowledge Graphs - Zhang et. al.

Method

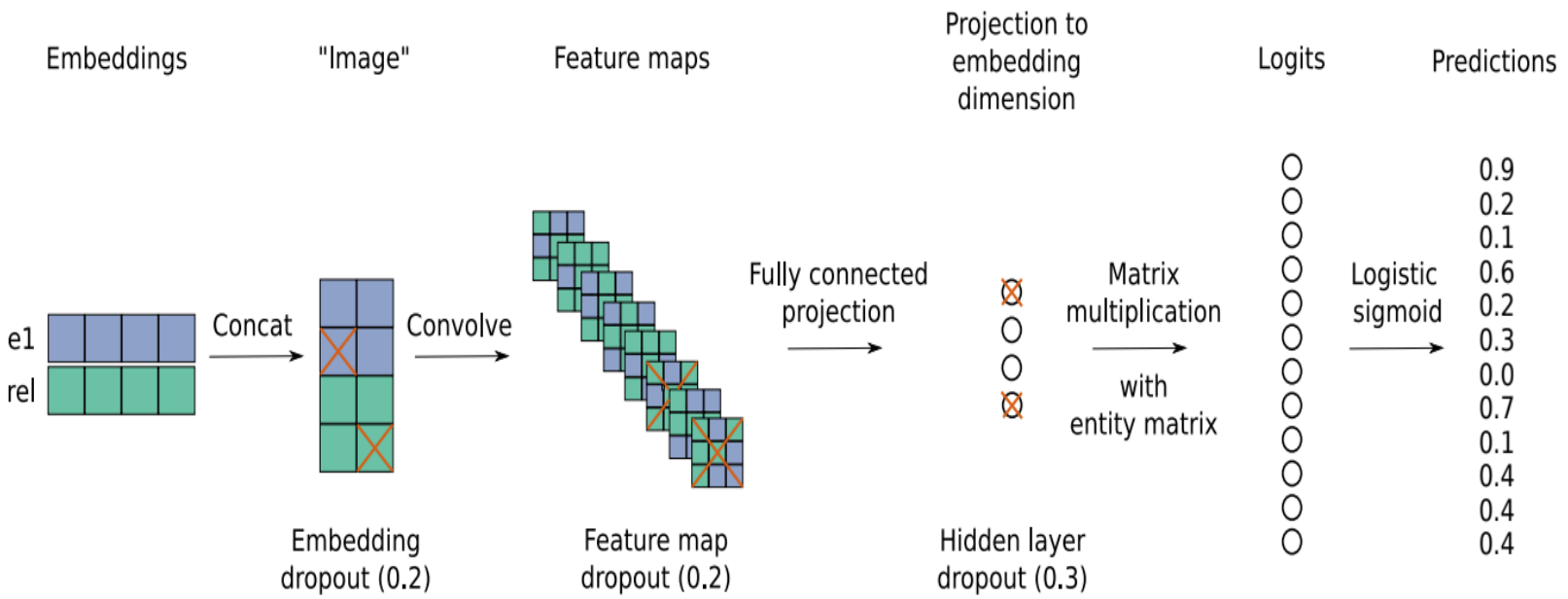

Figure: ConvE Approach

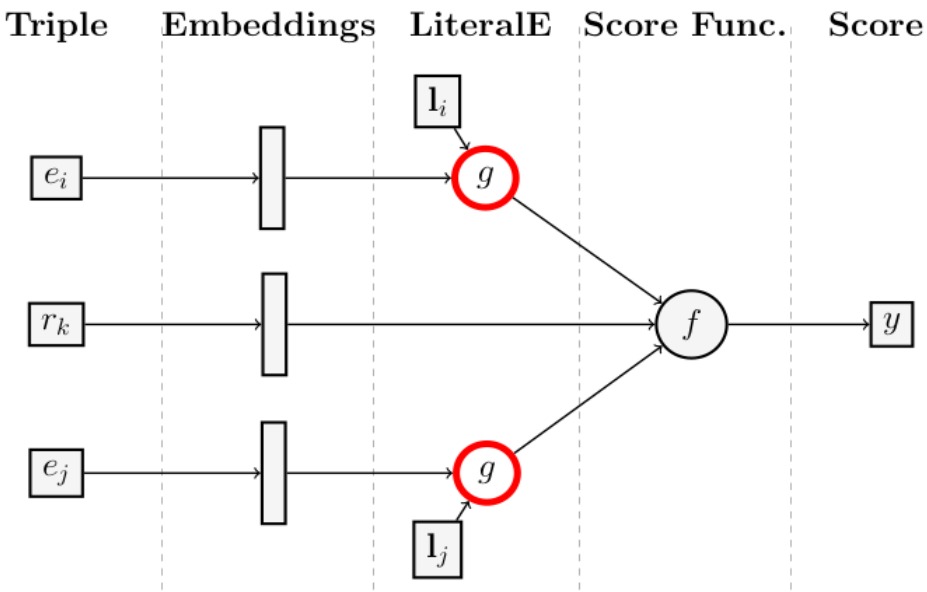

Figure: LiteralE Approach

Model	Scoring Function	# Parameters
DistMult	$\langle e_i, r_k, e_j \rangle$	$N_e H + N_r H$
ConvE	$f(vec(f([e_i; r_k] * \omega))\mathbf{W})e_j$	$N_e H + N_r H + C$
LiteralE	$f(vec(f([e_i^{lit}; r_k] * \omega))\mathbf{W})e_j^{lit}$	$\Lambda + (H + N_l)H$

Table: Model complexity in terms of number of parameters.

Dataset

Dataset:

Data set	#Relation	#Entity	#Train	#Valid	#Test
FB15k	1,345	14,951	483,142	50,000	59,071
FB15k-237	237	14,541	272,115	17,535	20,466
DB100k	470	**99,604**	597,572	50,000	50,000

Table: Statistics of Datasets used, where the columns respectively indicate the number of relations, entities, train / validation / test triplets.

Why FB15k-237 ?

- FB15k has a large number of test triples which can simply be obtained by inverting training triples. This results in a biased test set, for which a simple model which is symmetric with respect to object and subject entity is capable of achieving excellent results.
- But FB15k-237 does not suffer from invertible relation problem, for which KGC problem is more realistic.

Results for Serial Methods - ConvE, DistMult and LiteralE

Baseline Results:

Model (Dataset)	MR	MRR	Hits@10	Hits@3	Hits@1
ConvE (FB15k)	51	0.657	0.831	0.723	0.558
ConvE (FB15k-237)	244	0.325	0.501	0.356	0.237
ConvE (DB100k)	-	-	-	-	-
DistMult (FB15k)	97	0.654	0.824	0.733	0.546
DistMult (FB15k-237)	254	0.241	0.419	0.263	0.155
DistMult (DB100k)	-	0.233	0.448	0.301	0.115

Table: **Baseline Results:** On test set of FB15k, FB15k-237, DB100k. Results are taken from the original papers.

Our Results:

Model	Dataset	CPU					GPU				
		MR	MRR	Hits@10	Hits@3	Time	MR	MRR	Hits@10	Hits@3	Time
ConvE	FB15k-237	229	0.382	0.568	0.418	13.09	573	0.385	0.566	0.421	1.7
	FB15k	-	-	-	-	-	318	0.476	0.643	0.523	3.08
	DB100k	2219	0.48	0.655	0.523	176.23	2198	0.477	0.653	0.521	37.7
DistMult	FB15k-237	289	0.27	0.42	0.29	8.35	285	0.275	0.42	0.3	2
	DB100k	-	-	-	-	-	1752	0.291	0.447	0.334	29.2
LiteralE	FB15k-237	258	0.38	0.558	0.413	32.04	249	0.383	0.562	0.42	2.26
ConvE(2 layers)	FB15k-237	-	-	-	-	-	1091	0.375	0.562	0.411	3.12
ConvE(3 layers)	FB15k-237	-	-	-	-	-	819	0.381	0.562	0.42	2.16

Table: **Serial approach on CPU & GPU:** Each of the model were run for 100 epochs and for batch-size of 128, and running time was noted in hours.

Data Parallel Distributed Training

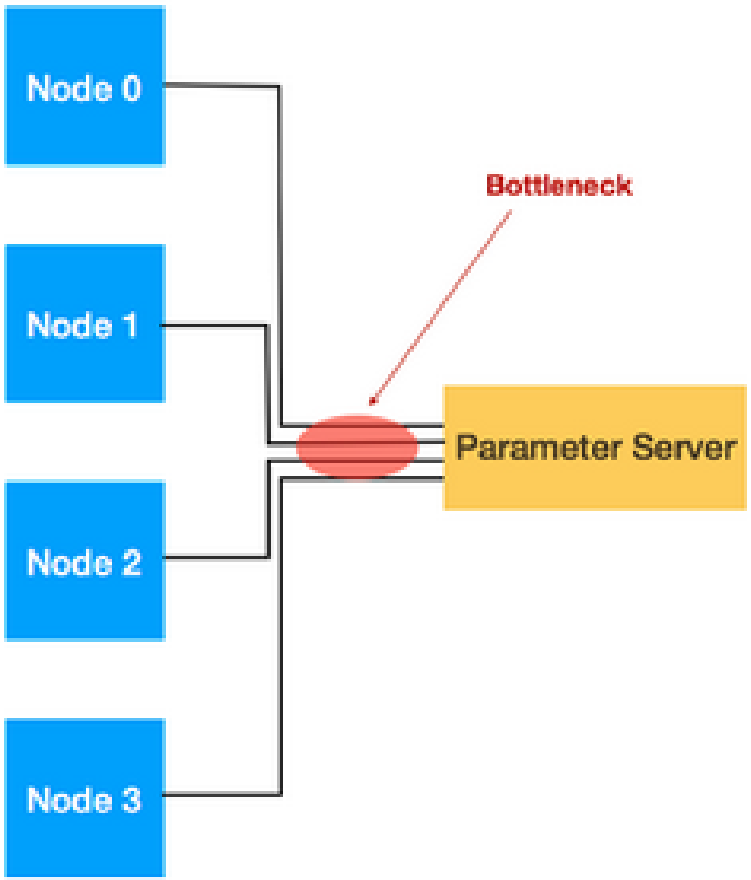

Figure: Parameter Server Strategy

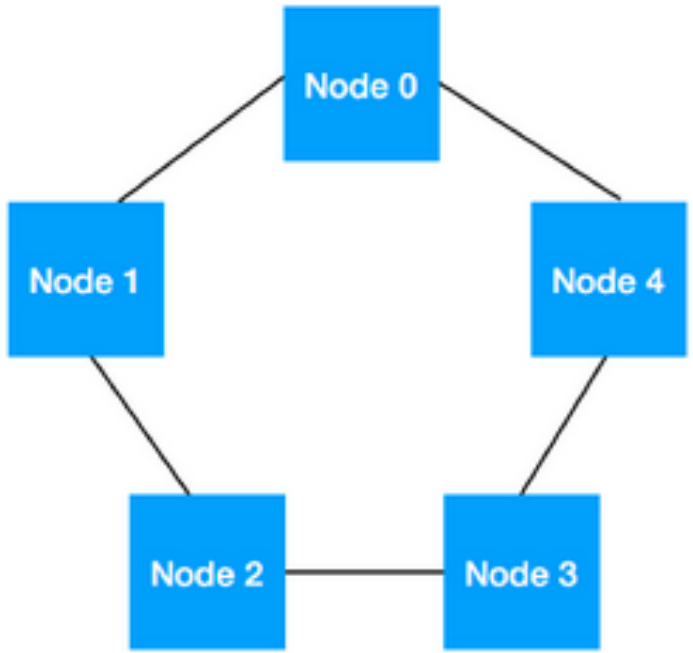

Figure: Ring All-Reduce Strategy

Data Parallel Distributed Training - Key Parameters

We'll discuss key parameters which we found that affects data parallel training and the efficient data pipeline strategy we have used.

- We have used the batch-size as large as possible, such that it can efficiently utilize cpu/gpu resources. With N workers, the effective batch-size becomes $N \times$ *batch-size*

- We have used scaled learning-rate w.r.t. number of workers making effective learning rate as $N \times$ *learning-rate*

- We have constructed a data pipeline, such that the data is sharded across the workers and use prefetching/ buffering to feed next batch to model, so CPU/GPU is never underutilized.

Data Parallel Distributed Training - Data Pipeline

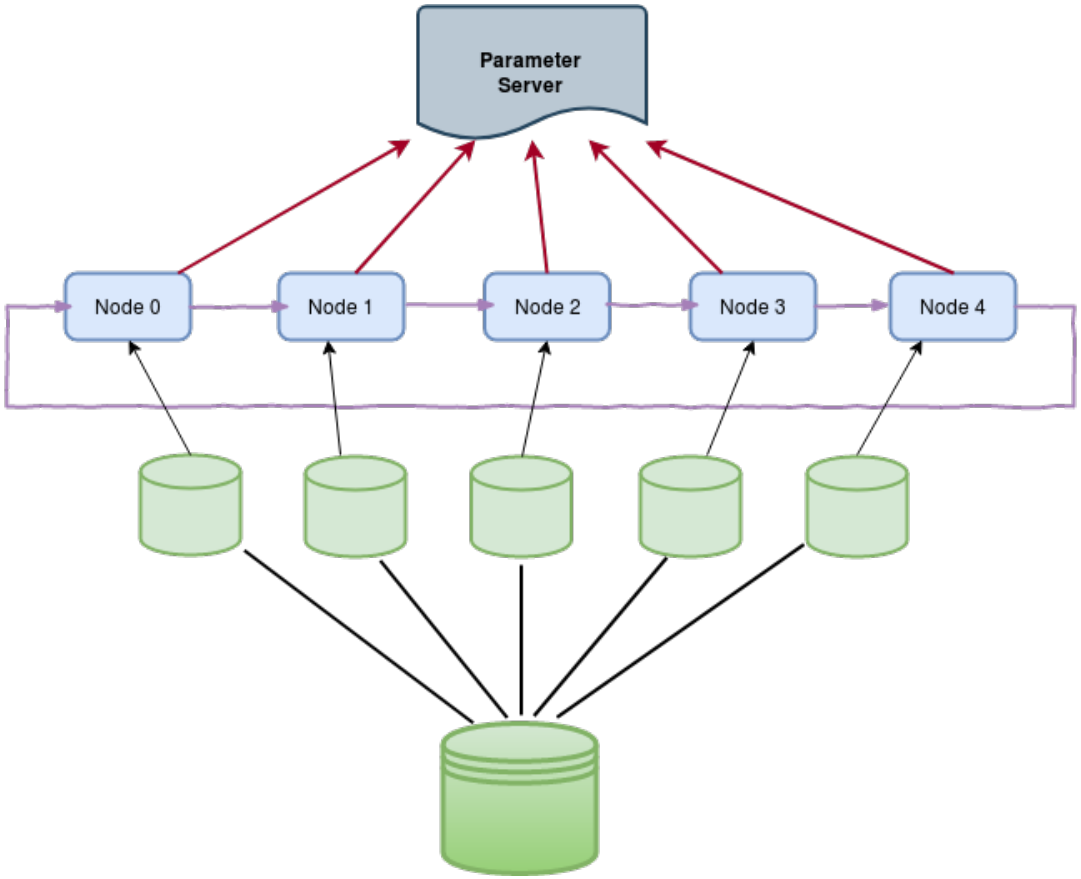

Figure: **Data Pipeline:** Sharded data pipeline across worker nodes for both Ring All-Reduce and Parameter-Server strategy.

Results : Parameter Server Strategy on CPU

Model (Dataset)	# PS / # Worker	Synchronous				Asynchronous				SSP			
		MRR	Hits @10	Time (Hrs)	Speed Up	MRR	Hits @10	Time (Hrs)	Speed Up	MRR	Hits @10	Time (Hrs)	Speed Up
ParConvE (FB15k-237)	1-PS & 4-W	0.382	0.579	12.2	1.07	0.377	0.56	3.16	4.14	0.382	0.564	4.5	2.90
	2-PS & 4-W	0.384	0.582	8.49	1.54	0.362	0.562	3.85	3.4	0.378	0.559	3.96	3.30
	1-PS & 6-W	-	-	-	-	0.375	0.558	1.94	6.74	0.376	0.554	1.79	7.13
	2-PS & 6-W	-	-	-	-	0.377	0.558	2.47	5.29	0.375	0.558	1.92	6.81
	3-PS & 6-W	-	-	-	-	0.373	0.552	1.45	9.02	0.372	0.551	2.26	5.79
ParConvE (DB100k)	1-PS & 4-W	0.534	0.685	168.55	1.04	0.309	0.485	38.1	4.62	0.31	0.497	36.95	4.76
ParDistMult (FB15k-237)	1-PS & 4-W	0.437	0.519	6.89	1.21	0.335	0.496	2.32	3.59	0.317	0.45	2.76	3.02
ParLiteralE (FB15k-237)	1-PS & 4-W	0.385	0.635	27.18	1.17	0.381	0.56	4.36	7.34	0.35	0.554	5.45	5.87

Table: **Parameter Server Strategy on CPU:** ParConvE, ParDistMult and ParLiteralE model were run for different number of combination of parameter-server and workers pair to find out if the parameter server becomes the bottleneck for different PS-Strategies. Dataset was sharded across the workers and model was run for 100 epochs, i.e. if there are 4 workers node then each worker runs for 100 epochs parallel on the per worker dataset with per worker batch-size of 128 and running time was noted in hours. Staleness parameter was set to 30 steps. We can see that async and ssp approaches scales linearly with no. of workers while synchronous approach fails badly for fixed number of epochs.

Results : Parameter Server Strategy on GPU

Model (Dataset)	# PS / # Worker	Synchronous				Asynchronous				SSP			
		MRR	Hits @10	Time (Hrs)	Speed Up	MRR	Hits @10	Time (Hrs)	Speed Up	MRR	Hits @10	Time (Hrs)	Speed Up
ParConvE (FB15k-237)	1-PS & 4-W	0.389	0.577	1.49	1.14	0.376	0.558	1.17	1.45	0.376	0.555	1.14	1.49
ParConvE (DB100k)	1-PS & 4-W	0.476	0.68	34.9	1.08	0.364	0.536	16.9	2.23	0.412	0.528	16.7	2.25
ParDistMult (FB15k-237)	1-PS & 4-W	0.35	0.494	2.1	0.95	0.332	0.487	0.64	3.15	0.334	0.487	0.66	3.03
ParLiteralE (FB15k-237)	1-PS & 4-W	0.381	0.574	2.17	1.04	0.376	0.550	1.62	1.39	0.376	0.552	1.59	1.42

Table: **Parameter Server Strategy on GPU - (with Single GPU per worker node):** Dataset was sharded across the 4 workers and model was run for 100 epochs, with per worker batch-size of 128 and running time was noted in hours. Staleness parameter was set to 30 steps.

Model (Dataset)	# GPU	Asynchronous				SpeedUp *
		MRR	Hits@10	Time	SpeedUp	
ParConvE (FB15k-237)	2	0.379	0.562	2.4	0.71	-
	4	0.382	0.562	1.07	1.58	1.06
ParConvE (DB100k)	2	0.424	0.602	21.2	1.77	-
	4	0.427	0.627	8.8	4.28	1.89
ParDistMult (FB15k-237)	4	0.335	0.486	0.45	4.44	1.46
ParLiteralE (FB15k-237)	4	0.379	0.558	0.58	3.89	2.74

Table: **Parameter Server Strategy on Single worker with multiple GPU:** Performance of GPU co-ordinated training for multiple GPU on a single worker node. This approach takes less running time than multi-worker single-gpu approach as in table 6. Last column compares the speedup of multiple-gpu/worker w.r.t. single-gpu/worker.

Results : Ring All-Reduce Strategy on CPU/GPU

Model (Dataset)	# Worker	Parameter Server *				All Reduce				SpeedUp w.r.t. *
		MRR	Hits@10	Time(hrs)	SpeedUp	MRR	Hits@10	Time(hrs)	SpeedUp	
ParConvE (FB15k-237)	4	0.377	0.56	3.16	4.14	0.366	0.542	0.94	14.38	3.47
ParConvE (DB100k)	4	0.31	0.497	36.95	4.76	0.246	0.41	10.65	16.54	3.47
ParConvE (FB15k-237)	6	0.376	0.554	1.79	7.13	0.329	0.472	0.56	23.37	3.28
ParDistMult (FB15k-237)	4	0.335	0.496	2.32	3.59	0.181	0.275	0.57	14.64	4.08
ParLiteralE (FB15k-237)	4	0.381	0.56	4.36	7.34	0.312	0.449	1.2	26.7	3.64

Table: **Ring All-Reduce Strategy on CPU:** Dataset was sharded across the workers and model was run for 100 epochs, with per worker batch-size of 128 and running time was noted in hours. We have compared the ring all-reduce against best PS strategy for same no. of workers and found out that **ring all-reduce achieves almost speedup of 4 compared to best PS strategy**, while its speedup w.r.t. to serial model is tremendous.

Model (Dataset)	# Worker	Parameter Server *				All Reduce				SpeedUp w.r.t. *
		MRR	Hits@10	Time(hrs)	SpeedUp	MRR	Hits@10	Time(hrs)	SpeedUp	
ParConvE (FB15k-237)	4	0.376	0.555	1.14	1.49	0.353	0.516	0.42	4.04	2.71
ParConvE (DB100k)	4	0.412	0.528	16.7	2.25	0.227	0.387	9.1	4.14	1.84
ParDistMult (FB15k-237)	4	0.332	0.487	0.64	3.15	0.179	0.271	0.2	10	3.17
ParLiteralE (FB15k-237)	4	0.376	0.552	1.59	1.42	0.349	0.507	0.42	5.38	3.79

Table: **Ring All-Reduce Strategy on GPU:** Dataset was sharded across the workers and model was run for 100 epochs, with per worker batch-size of 128 and running time was noted in hours. We have compared the ring all-reduce against best PS strategy for same no. of workers and found out that **ring all-reduce achieves almost speedup of 4 compared to best PS strategy**.

Speed-Up Analysis : ParConvE, ParDistMult, ParLiteralE

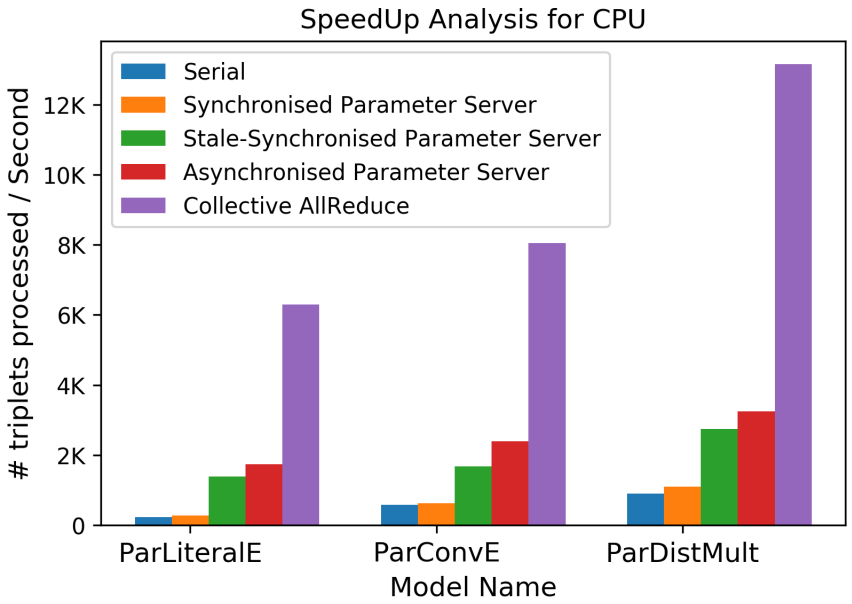

Figure: Speed-up graph for CPU for different approaches and models

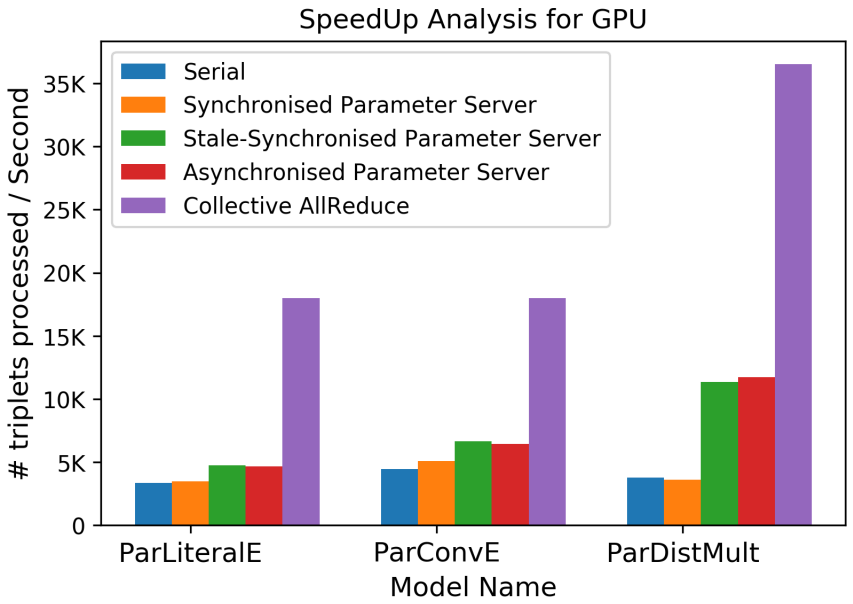

Figure: Speed-up graph for GPU for different approaches and models

Evaluation of Performance Metrics : ParConvE

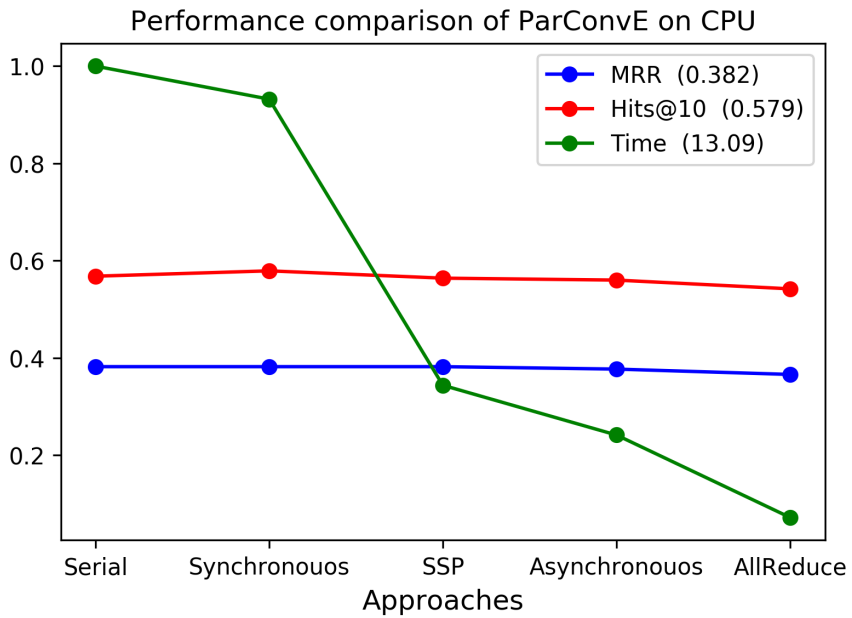

Figure: Performance comparison of ParConvE on CPU for FB15K-237. Time is scaled w.r.t. maximum time for any approach, while MRR and Hits@10 have their original values

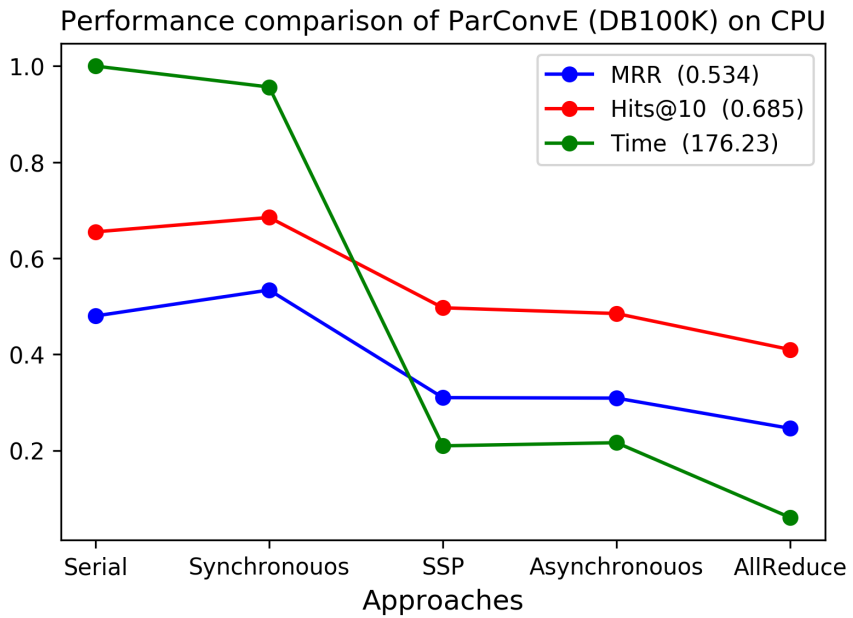

Figure: Performance comparison of ParConvE on CPU for DB100K. Time is scaled w.r.t. maximum time for any approach, while MRR and Hits@10 have their original values

Evaluation of Performance Metrics : ParDistMult & ParLiteralE

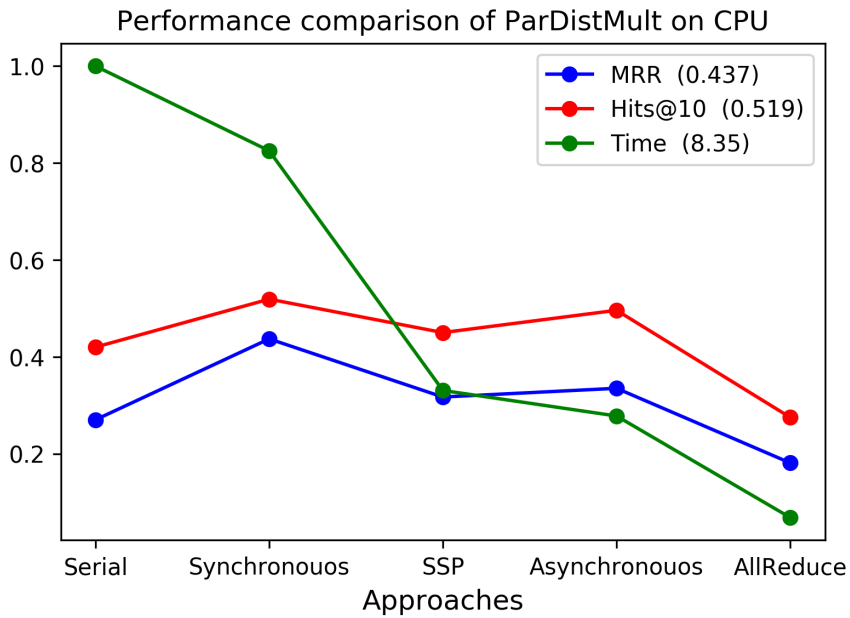

Figure: Performance comparison of ParDistMult on CPU for FB15K-237. Time is scaled w.r.t. maximum time for any approach, while MRR and Hits@10 have their original values

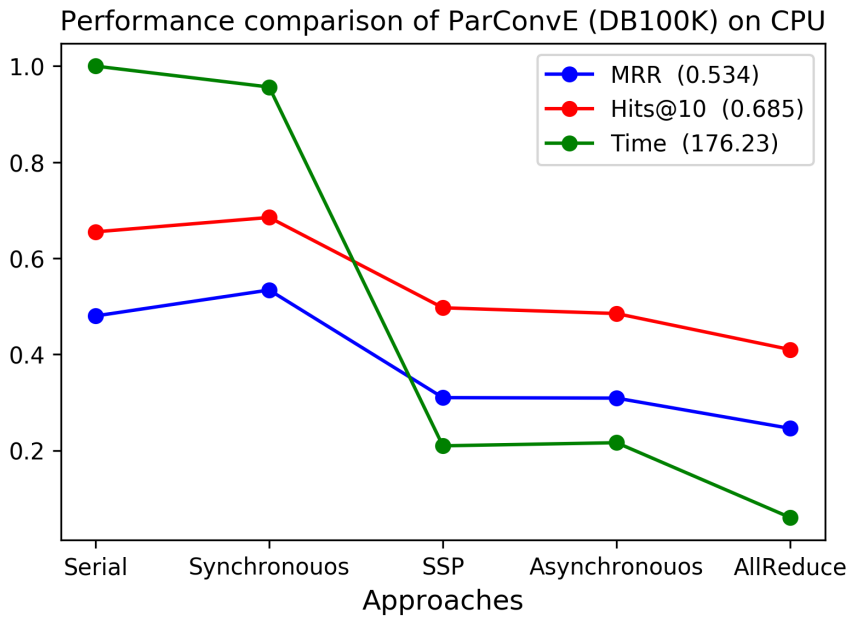

Figure: Performance comparison of ParConvE on CPU for DB100K. Time is scaled w.r.t. maximum time for any approach, while MRR and Hits@10 have their original values

Conclusion

- Using scalable batch-size, learning-rate w.r.t. number of workers and efficient data pipeline with next batch prefetching for efficient cpu/gpu utilization gives good speedup w.r.t. time.
- Synchronous PS Strategy fails badly w.r.t. time, as single parameter server becomes bottleneck. This can be solved by using multiple parameter servers or **Model Averaging Optimizer**, explained in future work.
- Asynchronous or SSP with little staleness parameter also gives speedup factor with negligible decrease in performance of evaluation metrics.
- Multiple GPU's per worker gives more speedup as compared to Single GPU per worker for parameter server strategies.
- Ring All-Reduce is the winner when it comes to speedup for distributed training, but for evaluation of performance metrics there is a little decrease in performance, which we think can be reduced if All-Reduce is run for more epochs.

Future Work

- **Horovod:** We would like to compare Uber's Horovod with Tensorflow's all-reduce, but as per literature tensorflow's ring all-reduce framework beats Horovod.
- **Model Averaging:** This is a synchronous strategy that synchronizes after every n steps instead of every step.
- **Elastic Averaging:** This is an asynchronous strategy, where local weights of worker and global weights of parameter server are updated after every n steps, by using elastic difference between current global weights and local weights.
- **Moving Averaging:** It uses decay value to maintain moving averages of weights, can be deployed to both sync and async PS approaches.
- **Asynchronous Ring All-Reduce:** Ring All-reduce send and receives their weight to and from neighbour workers after every steps, in asynchronous they will send and receive updates after fixed steps.

THANK YOU