Matrix Approxiations for Recommender Systems on TPUs

RASHMI VINAYAK, JASON YANG, and JUSTIN ZHANG

Recommender systems are prevalent across the internet. Many services rely on accurate recommendation systems, with deep learning recommenders becoming very popular due to their generalizable accuracy. However, as with many deep learning models, these recommenders have large magnitudes of parameters that have high computation cost. We seek to alleviate these issues by exploring the use of common matrix approximations, such as low rank, random Fourier features, and PCA, to compare their efficiency speedups on Google's TPU architecture versus traditional GPU and CPU setups.

We focus on a case study of training a DCN model over the movielens dataset, where we apply the matrix approximations for cross layer interactions. While low rank approximation is often the best generalizable approach for GPUs and CPUs in terms of high complexity reduction and preserved accuracy, our results suggest that random Fourier Features may scale better for large batch training.

Additional Key Words and Phrases: Recommender Systems, Deep Learning, Tensor Processing Units (TPUs), Cloud Training

ACM's Super Convoluted Template I don't Know How to Change:
Rashmi Vinayak, Jason Yang, and Justin Zhang. 2022. Matrix Approxiations for Recommender Systems on TPUs. 1, 1 (May 2022), 6 pages. https://doi.org/10.1145/hi-07400

1 INTRODUCTION

Recommendation systems are pivotal to a user's experience on many web applications such as social media and online marketplaces. Recent work has shown improved recommendations through a shift in architecture towards neural networks. A deep learning, neural network approach recommendation systems by further learning from previous user behaviors and modeling categorical data using embedding techniques, which are much more rich representations than traditional hot vectors. The current state of the art in neural network architecture based recommendation systems is the DCN model.

The DCN model features an embedding layer for all features, which then gets fed into multiple cross layers for feature interaction[4]. After the cross layer, the features are learned by the deep layers, with dimension size set by the user. For the DCN model, much of the computation complexity is accredited to the cross layer interactions.

There has been much recent work in reducing parameter size through matrix approximations in other architectures such that the reduction in accuracy is not too much. While these matrix approximation techniques are studied extensively on CPUs and GPUs, we aim to optimize approximation on Cloud Tensor Processing Units (TPUs). Our objective is to start with adapting current state of the art approximation techniques for TPUs to provide insight on future work and optimizations.

1.1 Related Work

1.1.1 Tensor Processing Units (TPUs).

Authors' address: Rashmi Vinayak, rvinayak@andrew.cmu.edu; Jason Yang, juncheny@andrew.cmu.edu; Justin Zhang, justinz@andrew.cmu.edu.

07400/2022/5-ART $15.00
https://doi.org/10.1145/hi-07400

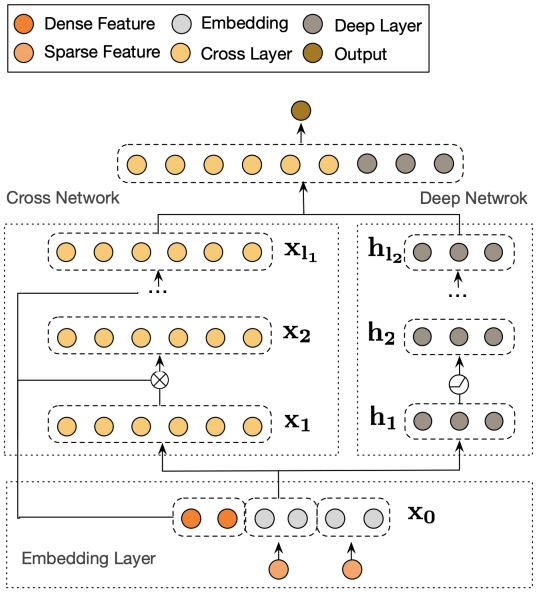

Fig. 1. The DCN model

The Tensor Processing Unit is hardware developed by Google for machine learning applications [3]. TPUs match the 99th percentile of CPU/GPU optimizations on neural net applications. The TPU is designed to be a coprocessor on the PCIe I/o bus, just like a GPU is. The host server sends instruction directly to the TPU. It is closer "in spirit" to a Floating Point Unit than a GPU. A TPU has quite low memory bandwidth compared to traditional GPU; however, Google has optimized TPU hardware configurations such that the bandwidth to system RAM memory is negligible.

1.1.2 *Case Study: DLRM model.*

In analyzing the DCN model over TPUs, we look to related work where Acun et al. analyze recommender systems over traditional GPU and CPU setups[2]. While their work dealt with the DLRM recommender model, their results and motivations are still especially useful since they deal with benchmarking large scale training for recommender models. Moreover, their observations that the main bottleneck for DLRM performance is due to their multilayer perceptron stack can be simplified to the simpler DCN model's main bottleneck, which is cross layer interaction.

Acun et al. find that as the number of dense and sparse features increase, training throughput reduces because of the increasing memory overhead from embedding operations. Thus, it is important to configure batch size accordingly; smaller batch sizes work better on GPUs due to their limited RAM while CPUs can rely on larger system RAM with low bandwidth connection. Since TPUs can rely on system memory, we suppose that TPUs are more efficient for large batch operations and scale better for larger datasets.

1.1.3 *Matrix Approximations.*

The crux of our work will be applying matrix approximations to improve the performance of cross layer interactions for TPUs. A major restriction for TPUs is their restriction on operations supported for guaranteed TPU performance operations. Specifically, TPUs require that no modifications are made to existing Tensorflow or Pytorch library functions. Thus, we focused our experiments on the following three approximations:

(1) **Low Rank**
We employ the following low rank approximation,

$$\mathbf{W} = \mathbf{UV}$$

where W is a dense matrix of size nm and U, V are of size ni, im for some lower projection dimension. Wang et al have found that a projection size reduction up to a quarter of the original size generally preserves accuracy[4]. As this is a native Tensorflow operation, it is readily supported by our TPU's requirements.

(2) **Principal Component Analysis (PCA)**
PCA calculates principal components, which are the vectors often calculated such that their average squared distance to the given points is minimized. Often done with eigen vector decomposition, the points are then projected to the vector which maximizes the variance within the points (the most "expressive" vector). While PCA is a relatively low cost computation, a major disadvantage of PCA is its reliance on linearity and correlation assumptions of its features, which is often not apparent within the data.

(3) **Random Fourier Features**

$$k(x, y) = \langle \phi(x), \phi(y) \rangle$$

Random Fourier Features is a technique based on the well studied kernel trick. The kernel trick is an efficient method of projecting data into a higher dimension, where, within that higher dimension, the data may be linearly separable. However, it is costly to calculate the kernel itself, so randomness is often employed. In this case, Tensorflow uses random fourier features, which creates low dimensional approximations of kernels functions at a low computational cost.

*1.1.4 **Parameter Configuration**.*

We follow the suggestions of Wu et al. for the parameter configuraton of our DCN model[5]. Based on their findings of desirable charateristics for ideal industry recommendation benchmark models, we preset our model to the following:

(1) Embedding dimesions: 32
(2) Cross and Deep Network Depth
(3) Layer Width: Dataset dependent

We follow these guidelines to allow our experiment to closely mirror industry practice.

1.2 Our Approach

Our objectives are as follows:

(1) Create foundational benchmarking for TPU efficiency benchmarking such that fine grained modifications of low level parameters (location of caching, embedding table placement, gradient synchronization, etc.) is possible
(2) Compare CPU/GPU/TPU benchmark times for different configurations and matrix transforms
(3) Obtain data to make a hypothesis on TPU specific efficiency improvements with not too much increase in error (with respect to low rank)

We approach initial benchmarking through Google Colab, which will allow ease of access between all hardware types along with easy parameter tuning. As there has been little to no work in creating benchmarks for recommender systems trained on TPUs, our work contributes a first attempt at doing so.

1.2.1 ***Design Choice and Changes.*** A major hurdle in working with TPUs is that they are proprietary to Google. The only way to utilize TPU hardware is either through Google Cloud or Google Collab. From our experiences, it was rather difficult to benchmark on Google Cloud due to the struggle of modification; it was not readily apparent to us how we could upload a modified version of the DCN model which would take advantage of different approximation techniques. Thus, we used Google Collab, which provided an easy to use interface for tweaking our model as well as interpreting our results.

Notably, our project was originally going to analyze the DLRM model. However, after meeting with project stakeholders at Google, they believed that the DCN model would be simpler to configure and benchmark for our purposes. Specifically, the model architecture includes a single feed-forward cross stack, while the DLRM model contains complex feature interaction, where categorical and numerical features are refed through an MLP stack for interaction crossing. Thus, the DCN model's simple cross layer stack was sufficient for our projects purpose, which is to find efficiencies for recommender systems on TPU hardware, where the main bottleneck is layer interaction.

Lastly, our experiments are ran on:

(1) CPU: Intel(R) Xeon(R) CPU @ 2.30GHz 4 Core
(2) GPU: NVIDIA Tesla T4
(3) TPU: TPU v2
(4) RAM: 13 GB

1.2.2 Dataset. For our experiments we use the widely conventional movielens dataset[1]. Specifically, we use the movielens1m dataset since it offers the most features (user ratings) for its size (1 million entries). Additionally, since this is a native Tensorflow_Datasets dataset, it is ensured to be fully efficient and preprocessed for TPUs.

2 EXPERIMENT SETUP

As stated, our experiments were done in Google Colab, where we implemented the DCN model in tensorflow. The main modifications we made was the ability to apply different approximation techniques via a flag parameter and projection dimension. This was done with Tensorflow classes and operations.

Next, we configure a Google Cloud bucket to host our dataset. This was done by first connecting our colab instance to the bucket and downloading the dataset to it via the Tensorflow_datasets library. Further preprocessing is done via mapping features to a vocabulary dictionary and configuring a 80/20 training/testing split with caching enabled.

Finally, we compile our benchmarked times with the time_it library, which provides duration of function calls in seconds. We take the average of a given experiment configuration over 10 repeated runs of training and testing our model to take account of cold starts and variability.

3 EXPERIMENT RESULTS

Our experimentation on CPU and GPU were as expected; low rank kept the RSME metric roughly the same as the results without approximation while decreasing computation time from around 2 hours to 30 minutes. As projection dimension scaled, the RSME increased greatly as expected. Thus, we concluded that PCA was not a good general approximation. Random fourier features did a good job of decreasing computation time while maintaining accuracy; it had a small increase in RSME while having performance gains similar to low rank (2 hours to 42 minutes on GPU).

For performances on TPUs, random fourier features have on average lower computing times than low rank. Especially as the dimensions are scaled lower, random fourier features has a higher rate

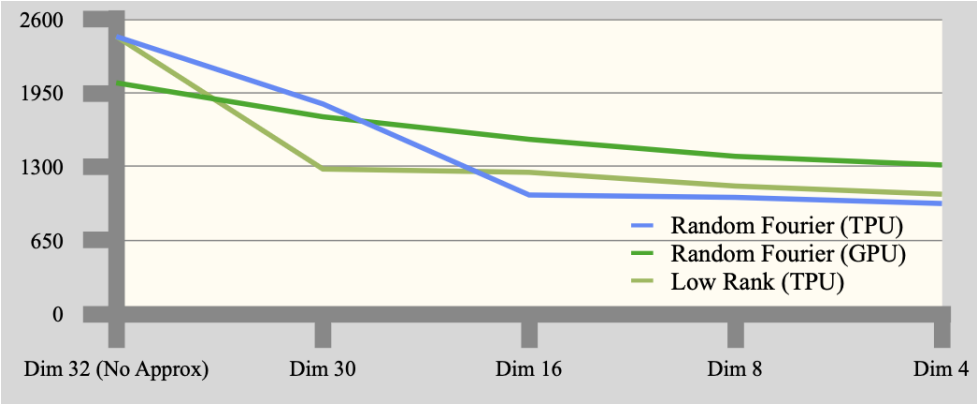

Fig. 2. Average Runtime (seconds) for Random Fourier and Low Rank

	Low Rank	Random Fourier	PCA
Dim 30	0.8219	0.8394	0.8542
Dim 16	0.8213	0.8433	0.9231
Dim 8	0.8353	0.8691	1.3121
Dim 4	0.8599	0.8848	1.9231

Fig. 3. RSME of the Different Approximations by their Dimension Reduction

of computing performance gain than low rank. These results imply that random fourier features could scale even better as the number of features and dimensions improve.

4 CONCLUSION

As expected, low rank approximation has the lowest RSME for all projection dimensions and the best computation time reduction for GPUs. PCA runs with very high error due to its reliance on linear data. However, on TPUs, random Fourier features has a lower average runtime along with a better rate of improvement as dimension is reduced. This implies that random Fourier features may scale better computation-wise as the number of features increase. Our results suggest Random Fourier features scale better for larger dimensions on TPUs with a larger margin of error than general low dimension.

4.1 Surprises

Mainly, I was surprised with the difficulty of learning how to benchmark the recommender model. While I knew I had to use Colab or Cloud, I had to piece together documentation on how to load, train, and run models with a benchmarking timer. I spent a lot of time trying to make Cloud work and ran into many issues about the flexibility of the experiments we were trying to run. In the end, Colab allowed us to do what we needed.

As for our results, the efficiency of random fourier features on TPUs were surprising. Intuitively, the projection to a higher dimension would seem to be more costly.

4.2 Future Work

The most direct future work is to run experiments with larger datasets. Given our constraints with Tensorflow and inherent computing constraints, the largest and most feature-full dataset for recommenders we could use was the movielens dataset. However, as our work with Google continues, we will gain access to larger computing resources which will allow us to train on larger datasets, like the Criteo dataset.

Moreover, due to our use of Colab, it was difficult to configure low level details such as the placement of embedding tables. Especially for TPUs, we could not reach this level of configuration without further permissions from Google. We suppose future work with table placement may provide further significant improvement.

Another area for improvement is the use of parallelization; we focused mainly on the application of a single GPU/TPU for the DCN model. However, the DCN model is inherently parallelizable since the cross stack and the deep stack and calculated independently. Thus, it is worthwhile to examine how the use of multiple GPU/TPU clusters can increase our performance.

REFERENCES

[1] 2021. Movielens 1M Dataset. https://grouplens.org/datasets/movielens/1m/

[2] Bilge Acun, Matthew Murphy, Xiaodong Wang, Jade Nie, Carole-Jean Wu, and Kim Hazelwood. 2020. Understanding Training Efficiency of Deep Learning Recommendation Models at Scale. arXiv:2011.05497 [cs.AR]

[3] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. arXiv:1704.04760 [cs.AR]

[4] Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed Chi. 2021. DCN V2: Improved Deep &amp Cross Network and Practical Lessons for Web-scale Learning to Rank Systems. In *Proceedings of the Web Conference 2021*. ACM. https://doi.org/10.1145/3442381.3450078

[5] Carole-Jean Wu, Robin Burke, Ed H. Chi, Joseph Konstan, Julian McAuley, Yves Raimond, and Hao Zhang. 2020. Developing a Recommendation Benchmark for MLPerf Training and Inference. arXiv:2003.07336 [cs.LG]