TT-REC: TENSOR TRAIN COMPRESSION FOR DEEP LEARNING RECOMMENDATION MODEL EMBEDDINGS

Chunxing Yin[1], Bilge Acun[2], Xing Liu[2], Carole-Jean Wu[2]

[1]Georgia Institute of Technology, [2]Facebook AI

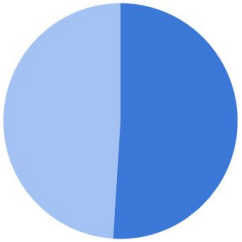
FACEBOOK AI

Outline

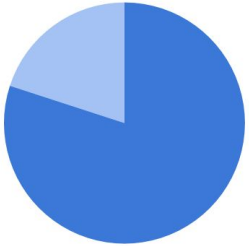
1. Recommendation Models
2. Tensor Train Decomposition
3. TT-Rec
    a. Memory Reduction
    b. Training Time
    c. Accuracy
4. Conclusion

Recommendation Models at Facebook

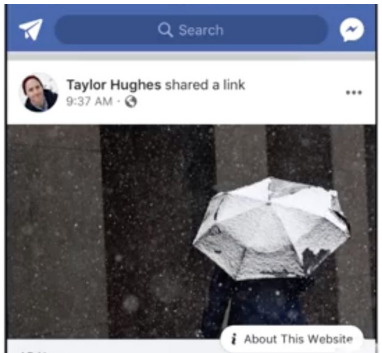
~50% of training



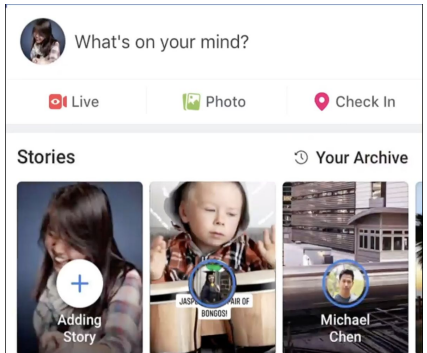
~80% of inference



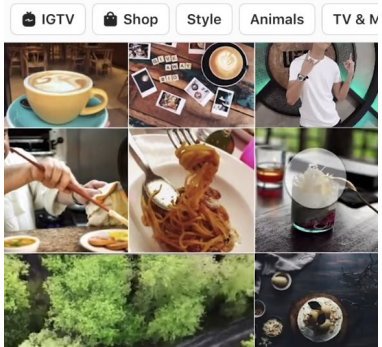
News Feed Ranking



Stories Ranking

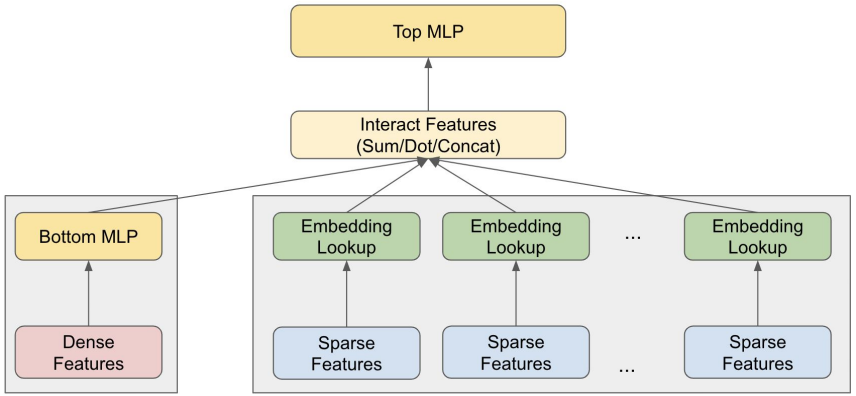


Instagram Explore



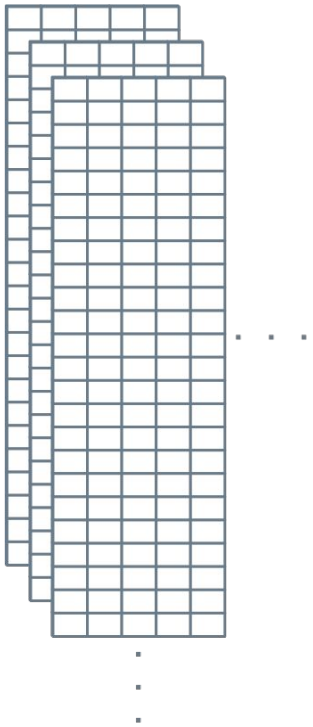
Deep Learning Recommendation Model - DLRM

- **Input**
  - Continuous features, e.g. age, time
  - Categorical features: e.g. videos watched
  - Label: positive/negative click
- **Embedding Table**
  - Each row maps an item to a floating point vector
  - Lookup the rows corresponding to items presented to the user, predict the user's reaction
- **Embedding Lookup**
  - Computes the sum or mean of 'bags' of embeddings

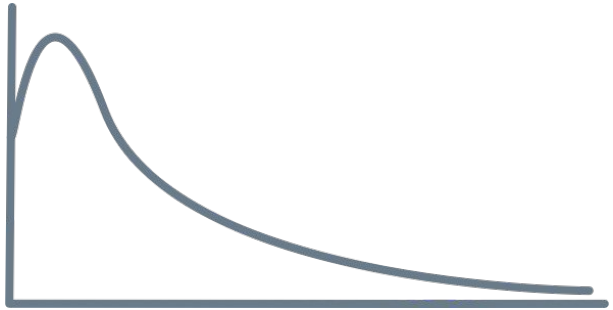


Challenges in Embedding Learning

Huge Vocabulary Size



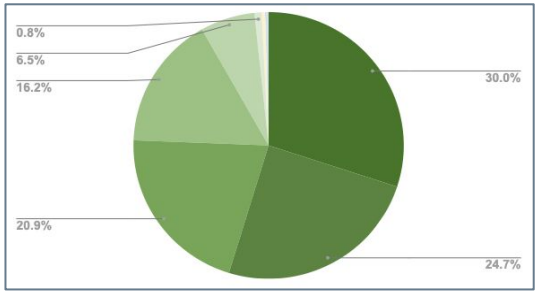
Skewed Data Distribution

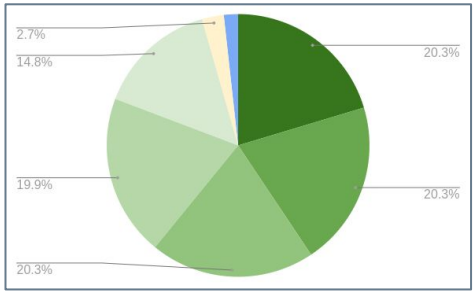


Recommendation Model Setup

- DLRM Model in MLPerf
  - Github: @facebookresearch/dlrm
- Datasets
  - Criteo Kaggle*
    - 7 days of ads click data
    - 13 numerical, 26 categorical features
  - Criteo Terabyte**
    - 24 days of click data, 4.3 billion records
    - 13 numerical, 26 categorical features
- Platform:
  - NVIDIA V100 GPU

*http://labs.criteo.com/2014/02/kaggle-display-advertising-challenge-dataset/,
**https://labs.criteo.com/2013/12/download-terabyte-click-logs/.





Kaggle Emb. Table Dimensions		Size (single precision)
10131227	16	0.65 GB
8351593	16	0.53 GB
7046547	16	0.45 GB
5461306	16	0.35 GB
2202608	16	0.14 GB
286181	16	0.018 GB
Others		0.018 GB
Total		**2.16 GB**

TeraByte Emb. Table Dimensions		Size (single precision)
9994222	64	2.56 GB
9980333	64	2.55 GB
9946608	64	2.55 GB
9758201	64	2.50 GB
7267859	64	1.86 GB
1333352	64	0.34 GB
Others		0.22 GB
Total		**12.58 GB**

Tensor Train Compression

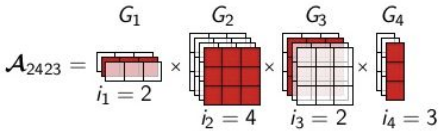
Tensor Train (TT) decomposition factorize a tensor as a product of small tensors

- For d-way tensor

$$\mathcal{A}(i_1, i_2, \ldots, i_d) = \mathcal{G}_1(:, i_1, :)\mathcal{G}_2(:, i_2, :) \ldots \mathcal{G}_d(:, i_d, :)$$

where G_k is a 3-way tensor of size $R_{k-1} \times N_k \times R_k$, and $R_0 = R_d = 1$. The sequence R_i is referred to as **TT-ranks**, and each tensor G_i is called a **TT-core**

- Small example*

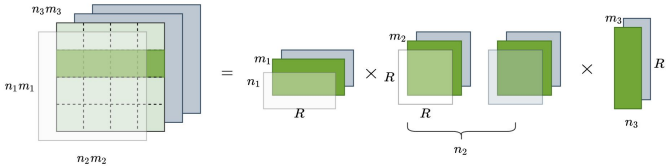


- **For matrix**

$$\mathcal{W}((i_1, j_1), (i_2, j_2), \ldots, (i_d, j_d)) = \mathcal{G}_1(:, i_1, j_1, :)\mathcal{G}_2(:, i_2, j_2, :) \ldots \mathcal{G}_d(:, i_d, j_d, :)$$

TT-matrix example

- Matrix W of size 5,000,000 x 24
- Factorize dimensions
  5,000,000 = 100 x 200 x 250,
  24 = 4 x 2 x 3
- Reshape W as a 6-way tensor ((100, 4), (200, 2), (250, 3))
- Decompose W using 3 TT-cores
  TT-core Shape:(1, 100, 4, R), (R, 200, 2, R),(R, 250, 3, 1)



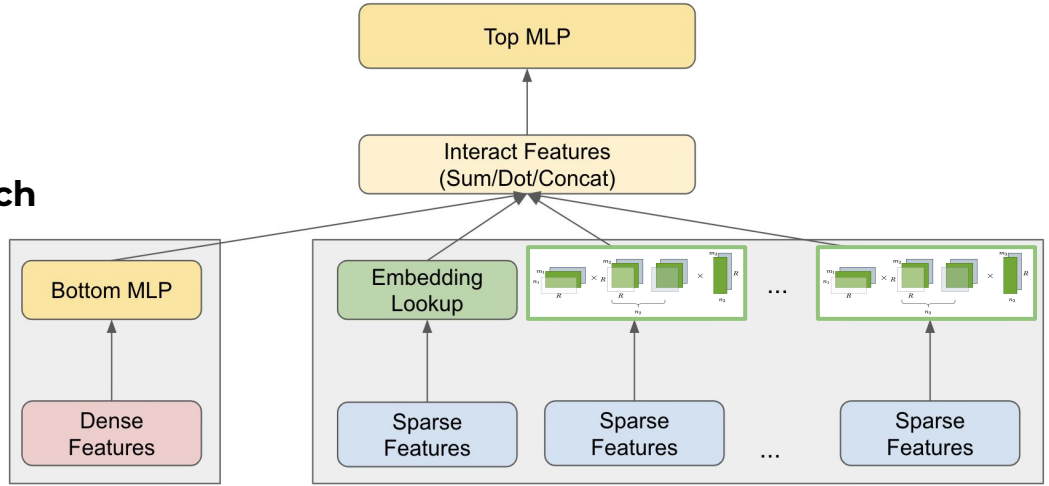
* Novikov, Alexander. Tensor Train decomposition in machine learning. Powerpoint presentation.

Tensor Train Compression for Embedding Tables in DLRM

- Network configuration
  - Replace emb. with TT format
  - Choose TT-rank
  - Randomly initialize TT-cores
- **Train TT-core weights from scratch**
- Compute for emb. vectors for embedding lookup

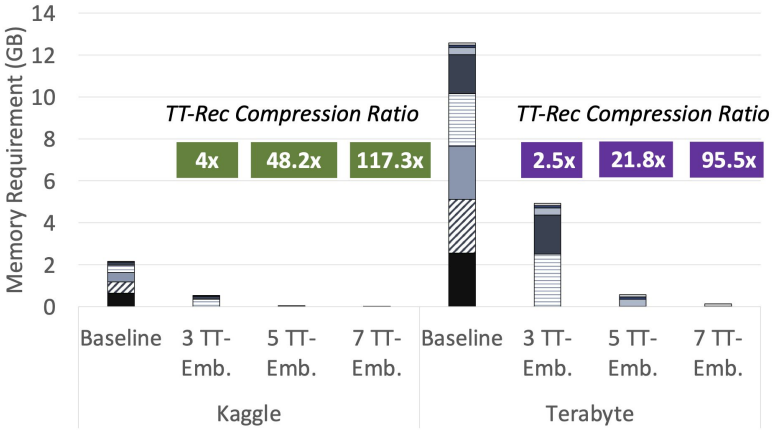
Challenges of Tensor Train

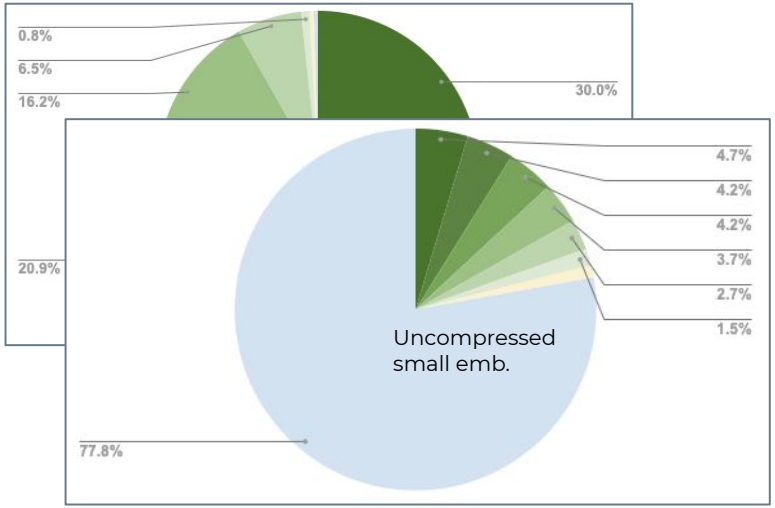
- TT-Embedding format:
  - Model accuracy degradation
  - Hyperparameter tuning
- TT-Embedding lookup:
  - Computation overhead
  - Efficient implementation, hybrid model



Memory Reduction with TT

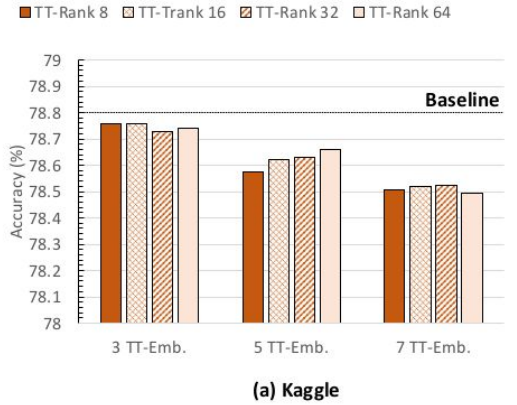
- Compress the largest 3 to 7 embeddings
- Single embedding table reduction up to 1200x
  - Store 10M x 16 emb. by 3 TT-cores:
    (1, 200, 2, R), (R, 200, 2, R), (R, 250, 4, 1)
- Overall model reduction ranges from 4x to 120x



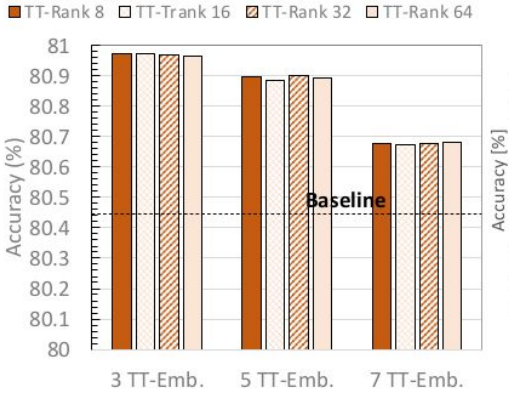


TT Model Quality

- With more emb. in TT format
  - Higher model reduction
  - Lower accuracy
  - For Kaggle, val. accuracy loss ranges from 0.03% to 0.3%
  - For Terabyte, TT-Rec outperforms baseline from 0.23% to 0.4%



(a) Kaggle

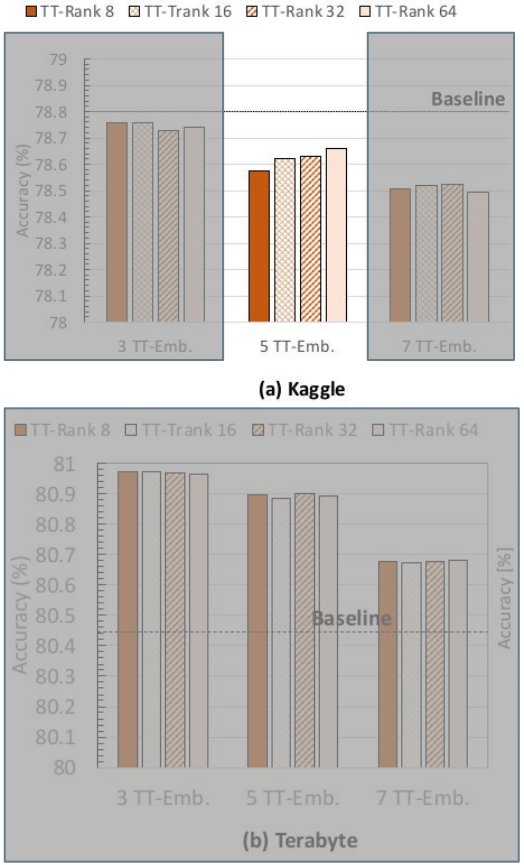


(b) Terabyte

* Note: Terabyte baseline have improved since the making of this plot.

TT Model Quality

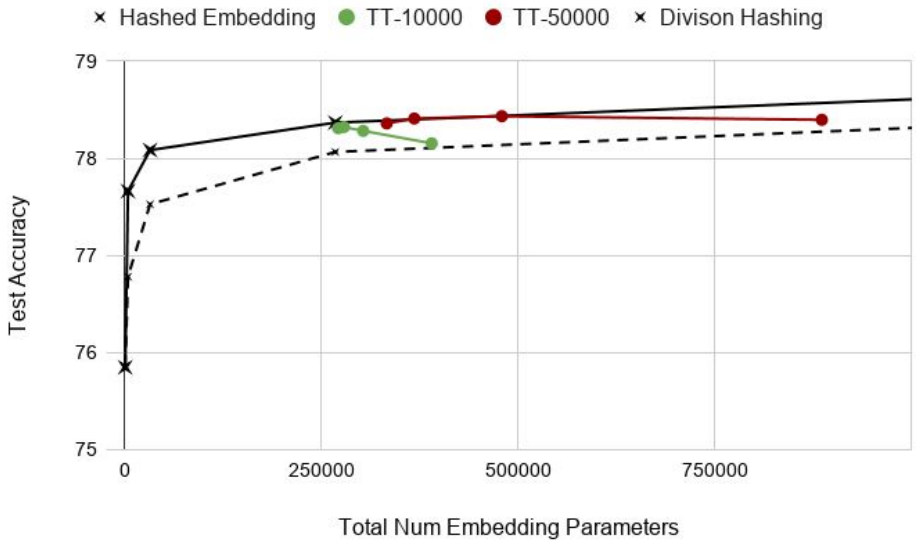
- With more emb. in TT format
  - Higher model reduction
  - Lower accuracy
  - For Kaggle, val. accuracy loss ranges from 0.03% to 0.3%
  - For Terabyte, TT-Rec outperforms baseline from 0.23% to 0.4%
- Using larger TT-ranks produces more accurate model at the expense of lower compression ratio



* Note: Terabyte baseline have improved since the making of this plot.

TT Comparison with Hashed Embeddings

- Apply hashing on row IDs
  - e.g. f(x) = x mod N
- Comparison with hashed embedding
  - Comparable accuracy
  - Avoid data dependence
  - Additional compression



Efficient Implementation of TT

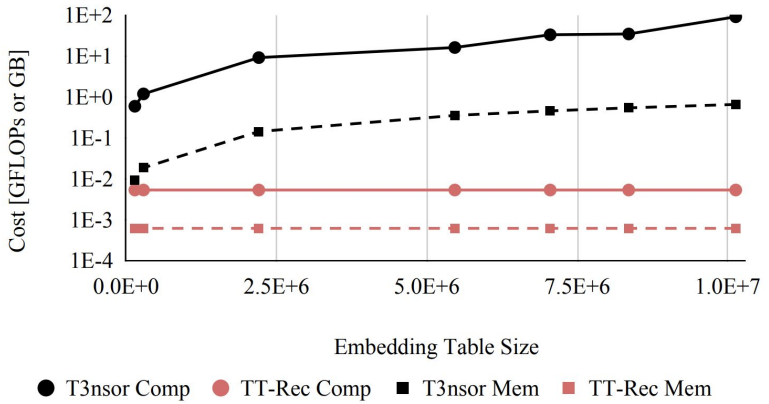
- Compute queried rows as a sequence of small matrix-matrix product

$$output_i = \sum_{k=offset[i]}^{offset[i+1]} \alpha_{indices[k]} w_{indices[k]}$$
$$= \sum_{k=offset[i]}^{offset[i+1]} \alpha_{indices[k]} \mathcal{G}_1(i_1^k, :, :) \dots \mathcal{G}_d(:, i_d^k, :),$$

- Iteratively compute B vectors via a single batched GEMM $tr_i = tr_{i-1}\mathcal{G}_i$
- Learn the gradient of the loss L w.r.t. the TT-cores through backward propagation

$$\frac{\partial L}{\partial \mathcal{G}_k(:, i_k, :, :)} = (w_i^{(k)})^T \frac{\partial L}{\partial w_i^{(k+1)}} \quad , \quad \frac{\partial L}{\partial w_i^{(k)}} = \frac{\partial L}{\partial w_i^{(k+1)}} \mathcal{G}_k^T(:, i_k, :, :)$$

- Roughly 10,000x lower memory footprint compared to T3nsor*; 2.4x faster than T3nsor on average

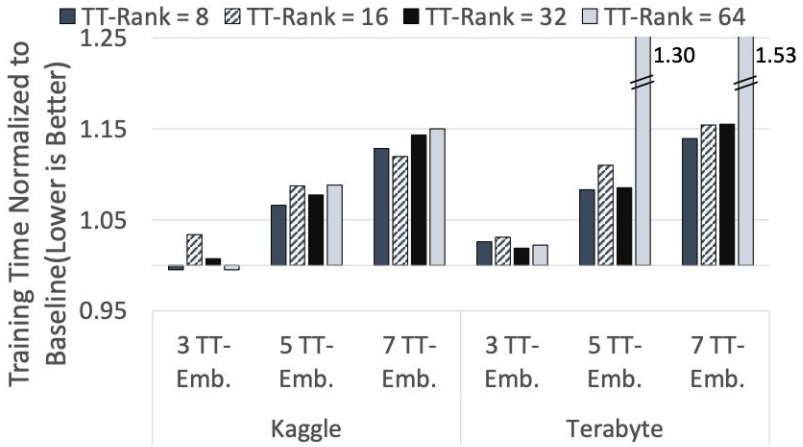


* Hrinchuk, O., Khrulkov, V., Mirvakhabova, L., Orlova, E. and Oseledets, I. Tensorized embedding layers for efficient model compression, 2020.

Training Time with TT

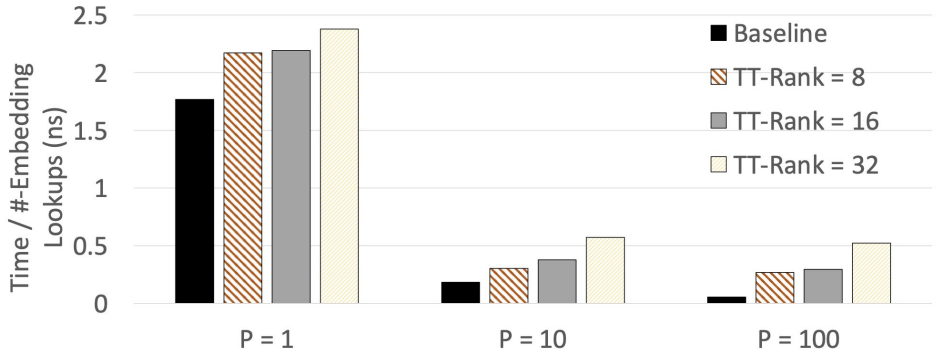
- Compare with Pytorch EmbeddingBag

- Increase emb. in TT format from 3 to 7
  - Reduces the model size by 46.5 and 37.4x for Kaggle and Terabyte respectively
  - Increase training time by 12.5% for Kaggle, and 11.8% for Terabyte with the optimal TT-rank
- Higher model size reduction come with higher training time overheads



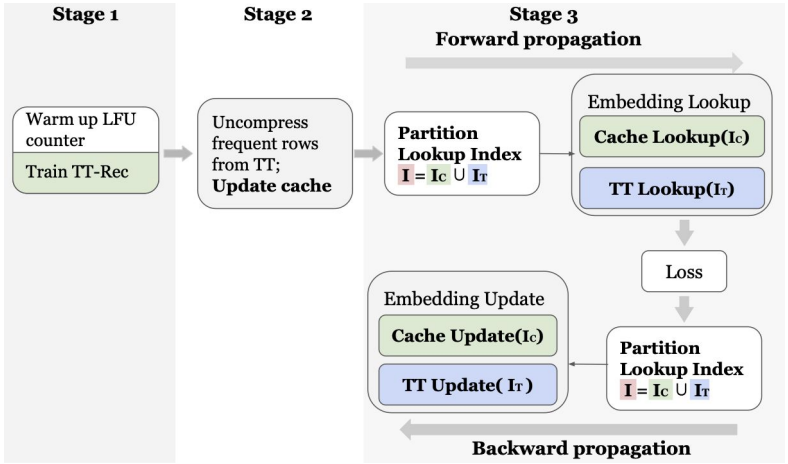
Benchmarking Performance of TT with Different Pooling Factors

- **Pooling factor (P):** the avg number of embedding lookups required per training sample
- Performance per training sample is better as P increases
- Performance gap between baseline and TT-lookup increases when P is larger
  - **Higher reuse of embedding vectors: potential caching opportunity**



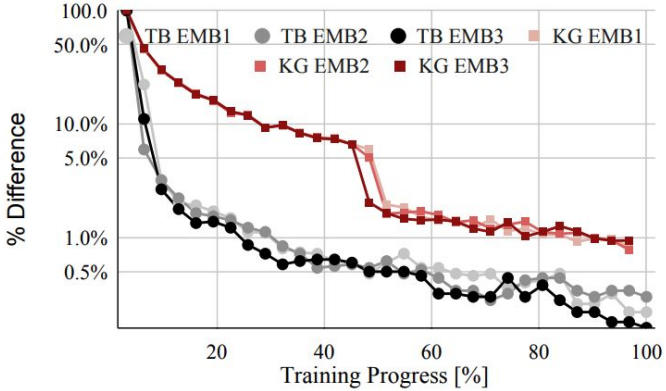
A Static Cache Design for TT

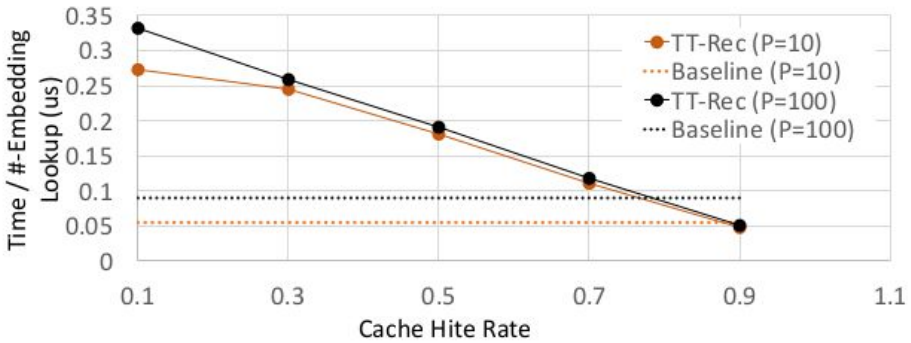
- Cache the uncompressed version of the most frequently accessed emb. vectors to reduce computation
- Partition emb. table into TT and cached subsets
  - Fetch *cached vectors* directly from cache
  - Learn the two subsets separately
- **Semi-dynamic cache**
  - Load the frequently accessed row every 1000s of iterations, initialize from TT cores
  - Discard learned weights in cache when cache eviction happens



Access Frequency of the Rows in Criteo Datasets

- Most-frequently-accessed 10k rows become stable after 50% and 10% of training data in Kaggle and Terabyte respectively
- As the cache hit rate increases, performance of TT-Rec improves and eventually outperforms the baseline when cache hit rate reaches 90%
- Devoting 0.01% worth of the embedding table memory requirement is sufficient





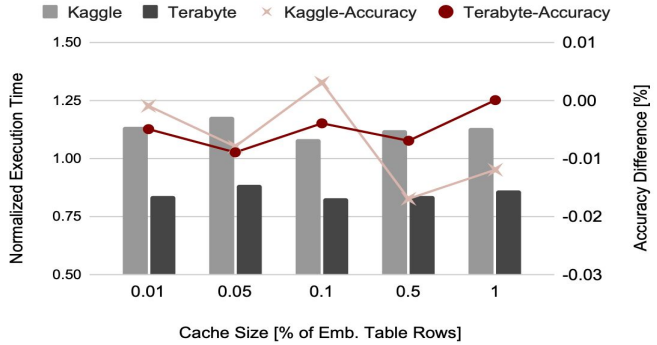
Execution Time and Accuracy using TT Cache

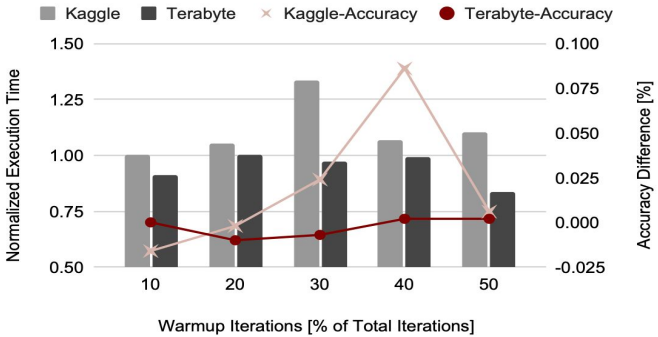
Cache Size

- Reduce training time for Terabyte, but not for Kaggle
  - Cache hit rate needs to be sufficiently high
- Devoting 0.01% worth of the emb. memory requirement is sufficient

Warm-up Iteration

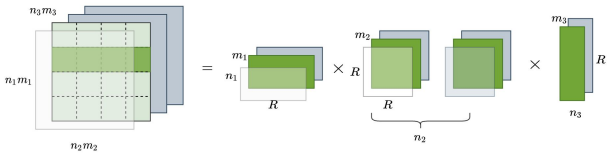
- LFU Cache is filled up during warmup, initialized from TT
- Longer warm-up: more overhead at the beginning, but higher cache hit rate afterwards





* Results are normalized to non-cached TT

Summary



Apply Tensor-Train compression for embedding tables

facebookresearch / FBTT-Embedding

Implement an efficient TT algorithm

Thank you

Github: @ facebookresearch/FBTT-Embedding

FACEBOOK AI