Memory-Efficient Tensorized Embedding Layers for Neural Networks

Chunxing Yin and Richard Vuduc,
Georgia Institute of Technology
Outline

1. Motivation
2. Tensor Train Decomposition
3. Model Accuracy
   a. Recommendation model
   b. Graph neural network
4. Training time and performance
Embedding Layer

- Deep Learning Recommendation System (DLRM)
  - ~50% of training
  - ~80% of inference

- Graph Neural Network (GNN)
  - Example graph
  - Visualization of node embedding
Challenges

- Tens of GB to TB size of embedding table
- Distribute on multiple CPUs
- **80% time** spent in host-device communication [1]
- GPUs in distributed GNN are underutilized
- Parameter update computes on CPU

General Embedding Compression Technique [2]

- Quantization
- Pruning
- Hashing

---


Tensorize Neural Network

- Replace full matrix/tensor parameters with a low-rank tensor decomposition
- A principled approach to compression

- In distributed GNN training
  - Enables data-parallelism
  - Reduce communication

<table>
<thead>
<tr>
<th>Network</th>
<th>Task</th>
<th>Compr. Ratio</th>
<th>Accuracy Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wide-ResNet [4]</td>
<td>Image</td>
<td>122x</td>
<td>2%</td>
</tr>
<tr>
<td>GRU [5]</td>
<td>Video</td>
<td>3000x</td>
<td>-120%</td>
</tr>
<tr>
<td>Transformer [6]</td>
<td>NLP</td>
<td>58.5x</td>
<td>4%</td>
</tr>
</tbody>
</table>

Background – Tensor Train Compression

Tensor Train (TT) decomposition factorize a tensor as a product of small tensors [7]

- For d-way tensor
  \[ A(i_1, i_2, \ldots, i_d) = G_1(\cdot, i_1, :) G_2(\cdot, i_2, :) \ldots G_d(\cdot, i_d, :) \]

  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

\[ W((i_1, j_1), (i_2, j_2), \ldots, (i_d, j_d)) = G_1(\cdot, i_1, :) G_2(\cdot, i_2, :) \ldots G_d(\cdot, i_d, :) \]

**TT-matrix example**

- Matrix \( W \) of size 5,000,000 x 24

- Factorize dimensions

\[
5,000,000 = 100 \times 200 \times 250,
24 = 4 \times 2 \times 3
\]

- Reshape \( W \) as a 6-way tensor

\((100, 4), (200, 2), (250, 3)\)

- Decompose \( W \) using 3 TT-cores

  TT-core Shape: \((1, 100, 4), (R, 200, 2, R), (R, 250, 3, 1)\)

Model Overview

Full Embedding

TT Embedding
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

<table>
<thead>
<tr>
<th>TeraByte Emb. Table Dimensions</th>
<th>Size (FP32)</th>
</tr>
</thead>
<tbody>
<tr>
<td>9994222</td>
<td>2.56 GB</td>
</tr>
<tr>
<td>9980333</td>
<td>2.55 GB</td>
</tr>
<tr>
<td>9946608</td>
<td>2.55 GB</td>
</tr>
<tr>
<td>9758201</td>
<td>2.50 GB</td>
</tr>
<tr>
<td>7267859</td>
<td>1.86 GB</td>
</tr>
<tr>
<td>1333352</td>
<td>0.34 GB</td>
</tr>
<tr>
<td>Others</td>
<td>0.22 GB</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>12.58 GB</strong></td>
</tr>
</tbody>
</table>
**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%

*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%

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

- Using larger TT-ranks produces more accurate model at the expense of lower compression ratio
Advantages of Tensorized Embedding

- Low rank representation
  - Compression
  - Preserve accuracy
  - Robust to overfitting and noise
- Generate a unique vector for each item
  - More stable than hashing
- Implicit item grouping and weight sharing

![Graph showing test accuracy vs. total num embedding parameters for different embedding methods.](image)
Connection to Graph

- TT-emb vector construction

  id: 2

  map to 3d coordinate
  \((n1 \times n2 \times n3\) grid\)

  \((0, 0, 2)\)

  \(G_1\)

  matmul

  \(G_2\)

  matmul

  \(G_3\)

  Embedding vector

- Connection to graph topology

  \((0, *, *)\)

  \((1, *, *)\)

  \((2, *, *)\)

  \(id \sim n_2 n_3\)

  \(id 0 \sim 2n_2 n_3\)

  \(id n_2 n_3 \sim 2n_2 n_3\)

  \(id 2n_2 n_3 \sim 3n_2 n_3\)
Connection to Graph

- TT-emb vector construction

- Connection to graph topology
Connection to Graph

• TT-emb vector construction

 id: 2
map to 3d index
\((n1 \times n2 \times n3)\) grid

\((0, 0, 2)\)

matmul

\(G_1\)

\(G_2\)

\(G_3\)

Embedding vector

• Connection to graph topology
Connection to Graph

- Parameter sharing through node reordering
- TT-cores correspond to recursive graph partitioning
- Align TT structure with graph topology to produce homophily representation
Accuracy with Node Reordering

- ognb-products graph (2.5M nodes) trained with Graph Attention Network (GAT)
- Outperform the full embedding baseline
- Fine-grained graph partitioning helps improving model accuracy
- Produce better node embedding than the original dataset

<table>
<thead>
<tr>
<th>Partition</th>
<th>0</th>
<th>4</th>
<th>16</th>
<th>256</th>
<th>800</th>
<th>1600</th>
<th>3200</th>
<th>#param Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full Emb</td>
<td>0.7485</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1x</td>
</tr>
<tr>
<td>Rank 8</td>
<td>0.7448</td>
<td>0.745</td>
<td>0.748</td>
<td>0.7578</td>
<td>0.7535</td>
<td>0.7615</td>
<td>0.7628</td>
<td>5762x</td>
</tr>
<tr>
<td>Rank 16</td>
<td>0.7544</td>
<td>0.7629</td>
<td>0.7564</td>
<td>0.7681</td>
<td>0.7756</td>
<td>0.7713</td>
<td>0.7626</td>
<td>1580x</td>
</tr>
<tr>
<td>Rank 32</td>
<td>0.7632</td>
<td>0.7719</td>
<td>0.7671</td>
<td>0.7678</td>
<td>0.79</td>
<td>0.7647</td>
<td>0.78</td>
<td>415x</td>
</tr>
</tbody>
</table>
Training Time of DLRM

- 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
Training Time of GNNs

- Full Emb: 90% time spent on update
- 30% time for emb lookup, 60% time for emb backprop
- Reduce training time of ogbn-papers100M by 4.6X
- Scales almost linearly with large TT-ranks
- Hardware
  - AWS EC2 g4dn-metal
  - 8 T4 GPUs, 2x24 cores, 384GB RAM
Summary

- Applied Tensor-train decomposition to compress embedding layers in recommendation system and GNNs
- Compress the embedding tables by 100x and 424x for the 2 models while preserving/improving the model accuracy
- Combine TT with hierarchical graph partitioning to generate homophily embedding
- Efficient implementation of TT-emb kernel