Lossless Compression Techniques for Embedding Tables in Substantial Deep Learning-Based Recommendation System

基於深度學習之推薦系統嵌入表無損壓縮

指導教授： 黃俊達、吳凱強　　組員：楊育陞、葉哲伍

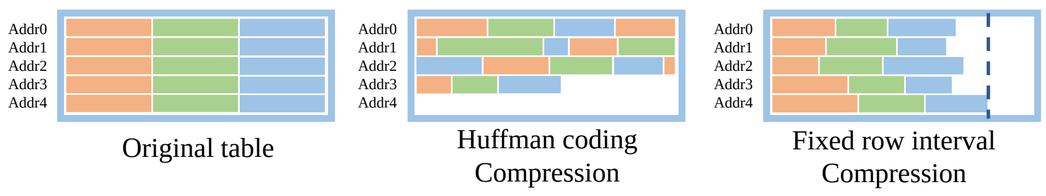
I. ABSTRACT

Deep learning-based recommendation systems are popular in a variety of fields. However, one of the most significant challenges is the high spatial cost of embedding tables. Our research focuses on lossless compression for embedding tables, which can effectively reduce spatial costs without sacrificing recommendation accuracy. We propose a new lossless compression method and compare the results of the experiments with the theoretical limit of lossless compression.

II. INTRODUCTION

Deep learning-based recommendation systems achieve high accuracy but come with a significant spatial and computational cost. One major contributor to spatial cost is embedding table, which contains the information of users and items. As a result, the compression for it becomes imperative.

Embedding table is a large matrix with typically large number of rows, each of uniform length. A well-known lossless compression method is Huffman coding. However, this approach introduces non-uniform row lengths, requiring extra handling. To ensure hardware efficiency, the compressed table must maintain the feasibility of random access, namely, to keep the row interval fixed. This limit the compression ratio to the longest row of table. Directly split the Huffman-coded table row by row may result in poor compression ratios, especially when longer codings share a row. To address these, we propose new coding methods.



III. Methodology

Instead of directly building the coding tree, we first decide the shape of the tree and then assign the characters to the tree based on their frequency. The latter is a fixed algorithm; therefore, our main goal is to find the optimal shape of the tree.

Construction of Coding Tree

In our project, the character length is 8-bit. As a result, the coding tree will have 256 leaves. It is impossible to find the optimal tree by enumeration of all structures. We propose two methods to address this problem. The first method is to split a character into two 4-bit characters. The second method is to use simulated annealing algorithm to find the optimal 8-bit coding tree.

Method 1: Split into two 4-bit

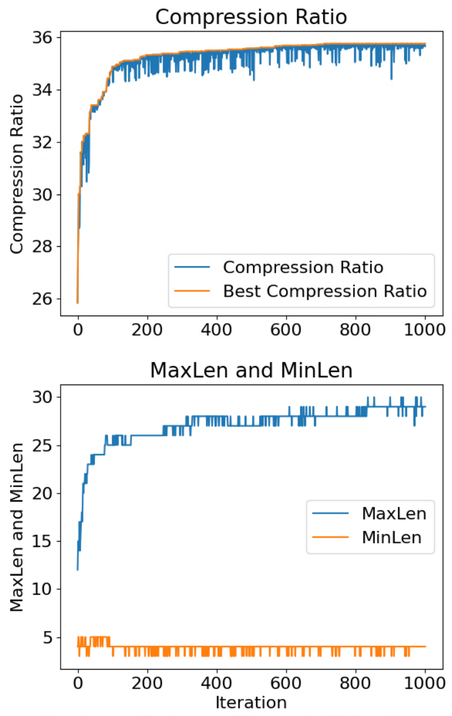
Converting 8-bit encoding to 4-bit encoding can enhance decoding speed and find the tree with the highest compression ratio by enumerating all structures. However, the compression ratio may be sacrificed in this process. To achieve even higher compression ratios, we employ two 4-bit trees, dividing the data into two sets and allocating them using a greedy algorithm.

The steps of the algorithm are roughly as follows. First, we identify the globally optimal tree by enumeration. Then, we exhaustively compress with the second tree starting from the row with the smallest compression ratio until there was no further improvement.

Method 2: Simulated Annealing for 8-Bit Coding Tree

In finding an optimal 8-bit coding tree, we approach it as an optimization problem and using simulated annealing algorithm to find the best coding tree.

The process begins with the setup of starting coding tree based on the data, followed by multiple iterations. Each iteration fine-tune the coding tree to explore neighboring structures. Then decide whether to accept the new coding tree based on the resultant compression ratio and annealing temperature, which is the parameter governing the willingness to accept new outcomes. By scheduling these parameters, the coding tree will converge toward optimal gradually.

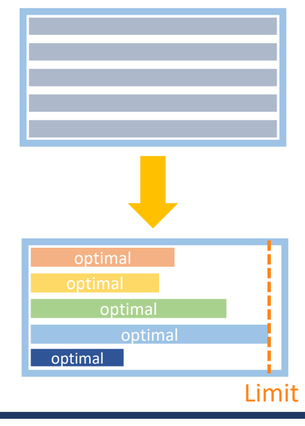


SA from Balance Tree - MT5_xl_emb

For the starting coding tree, we tried two types: the Huffman tree and the adjusted balanced tree. Given that not all characters necessarily appear in the data, we can prune unnecessary leaves to enhance the adjusted balanced tree.

Theoretical Limit

To find the theoretical limit of lossless compression, we perform 4-bit or 8-bit Huffman encoding individually for each row. At this stage, each row is the optimal compressed result. The row with the smallest compression ratio stands as the theoretical upper limit for these 4-bit or 8-bit data.



IV. Results

Compression Ratio / Table	# of MB	Single 4bit Tree	Two 4bit Tree	# of using Second Tree	Limit of 4bit Tree	Limit - Best	Balance Tree	SA from Balance Tree	Huffman	SA from Huffman	Limit of 8bit Tree	Limit-Best
DLRM_emb_19	5,117	3.12%	6.45%	3	6.93%	0.49%	10.74%	11.52%	-106.25%	-99.51%	23.24%	11.72%
DLRM_emb_00	5,105	4.88%	7.03%	2	7.52%	0.49%	12.11%	12.60%	-101.66%	-89.65%	24.80%	12.20%
DLRM_emb_21	5,077	2.54%	4.88%	1	5.57%	0.68%	9.86%	10.55%	-117.87%	-100.59%	20.99%	10.44%
DLRM_emb_09	4,932	5.57%	6.74%	10	7.62%	0.88%	11.43%	13.77%	-64.94%	-47.46%	24.90%	11.13%
DLRM_emb_20	3,282	6.05%	8.40%	5	8.98%	0.59%	12.11%	15.33%	-84.57%	-69.04%	25.98%	10.65%
MT5_xl_emb	512	18.76%	18.83%	1	18.84%	0.01%	23.01%	35.77%	33.29%	35.04%	36.03%	0.26%
DLRM_emb_10	378	6.84%	9.18%	2	10.35%	1.17%	17.29%	20.12%	-64.94%	-47.46%	26.86%	6.74%
BLOOM_word_emb	256	24.45%	24.95%	6	24.98%	0.02%	12.49%	43.32%	30.81%	38.48%	44.41%	1.09%
MT5_large_emb	256	12.72%	12.78%	10	12.79%	0.01%	12.46%	27.03%	24.38%	26.56%	27.70%	0.67%
DeBERTa_word_emb	196	7.28%	7.94%	10	8.17%	0.23%	9.03%	18.59%	-0.38%	15.62%	19.60%	1.01%
MT5_base_emb	192	7.60%	7.98%	2	8.40%	0.42%	11.00%	19.34%	15.25%	18.95%	20.81%	1.47%
LLaMA_tok_emb	131	15.14%	15.21%	1	15.22%	0.01%	12.38%	30.81%	30.39%	30.81%	30.94%	0.13%
MT5_small_emb	128	5.74%	5.81%	10	5.83%	0.02%	5.47%	14.55%	6.20%	9.74%	16.87%	2.32%
DLRM_emb_22	74	3.52%	5.96%	1	6.05%	0.10%	9.86%	10.06%	-107.32%	-90.43%	22.17%	12.11%
DLRM_emb_11	51	2.15%	4.00%	2	4.59%	0.59%	7.42%	8.89%	-89.65%	-68.75%	22.36%	13.47%
RoBERTa_word_emb	38	8.14%	8.63%	10	8.85%	0.23%	12.16%	19.99%	18.65%	19.87%	21.27%	1.28%
GPT2_word_emb	38	10.58%	10.73%	7	10.90%	0.18%	11.52%	24.02%	20.64%	23.60%	25.63%	1.61%
Average		8.53%	9.73%		10.09%	0.36%	11.78%	19.78%	-32.82%	-23.19%		5.78%

V. Conclusion

Our approaches achieve a compression ratio close to the theoretical limit. For Method 1, the average difference from the limit is 0.36%. For Method 2, the average difference from the limit is 5.78%. Excluding DLRM, the average difference from the limit would be 1.09%.

In conclusion, we have introduced a novel compression algorithm that exhibits commendable performance and is well-suited for hardware implementation.