Tensor-Train decomposition을 적용한 임베딩 테이블을 위한 연산 최적화 기법

유승민 O, 이하윤, 신동군
성균관대학교 전자전기컴퓨터공학과
sminy67@gmail.com, lhy920806@skku.edu, dongkun@skku.edu

Optimizing computation of embedding table with Tensor-Train decomposition

Seungmin Yu O, Hayun Lee, Dongkun Shin
Department of Electrical and Computer Engineering, Sungkyunkwan University

요 약

개인 맞춤형 추천 시스템은 전자상거래, 소셜 네트워크 서비스(SNS) 등 일상에 녹아 들어있다. 특히, 요즘 기업에서 딥러닝 기반 추천 시스템을 사용하는 사례가 늘어나고 있다. 하지만 딥 러닝 기반 추천 시스템 모델에서 사용하는 임베딩 테이블을 저장하는 데 필요한 엄청난 양의 메모리 증가로 인해 산업용 AI 데이터 센터의 리소스 대부분을 차지하고 있다. 이 문제를 극복하기 위한 해결책 중 하나는 Tensor-Train(TT) 분해 기법이다. TT 분해 기법은 임베딩 테이블을 여러 개의 TT-core로 분해하여 메모리양을 줄이는 대신 연산 수가 증가하게 된다. 본 논문에서는 TT 분해 기법이 적용된 임베딩 테이블에서 pooling factor 즉, 유저들이 하나의 임베딩 테이블에서 상호 작용하는 임베딩 벡터의 평균 개수와 불필요한 연산 수가 비례한다는 점을 분석하고 불필요한 연산을 줄이기 위한 Group Reduced TT-Gather and Reduce (GRT-GnR) 연산을 제안한다. 실험 결과, GRT-GnR 연산은 기존 TT-embedding 연산에 비해 같은 메모리 감소량 대비 latency를 29%까지 감소시켰다.

1. 서 론

개인 맞춤형 추천 시스템은 요즘 산업 AI 데이터 센터에서 어디서나 볼 수 있다. 전자상거래, 소셜 네트워킹, 콘텐츠 추천 등 다양한 애플리케이션으로 인해 개인 맞춤형 추천 시스템은 AI 데이터센터의 리소스 대부분을 소비한다. 또한, 유저와 수백만 개의 아이템 간의 상호 작용하는 입력을 활용하기 위해 딥 러닝 기반 추천 모델은 임베딩 테이블당 lookup (gather) 연산 및 pooling (reduce) 연산이 있는 임베딩 레이어를 사용한다. 한편, 유저와 아이템 간의 상호 작용하는 입력이 커짐에 따라 임베딩 레이어는 메모리 사용량이 크게 증가하여 단일 GPU나 Neural Processing Unit (NPU)에서 실행하기가 어려워진다.

이 문제를 해결하기 위해, [2], [3]에서는 Tensor-Train (TT) 분해 기법을 사용하였다. TT 분해 기법[1]은 크기가 큰 텐서를 여러 개의 크기가 작은 텐서인 TT-core들로 분해하는 기법이다. 이는 매우 높은 압축률로 인해 Fully-Connected layer나 Convolutional layer와 같은 심층 신경망에 사용된 유망한 압축 기술이다. 이와 같이 임베딩 레이어에서도 메모리 사용량을 줄이는 데는 성공하였으나 임베딩 레이어만의 고유한 연산으로 인해 TT 분해 기법을 적용하였을 때 연산 수가 증가하게 된다. 주로 행렬 곱 연산을 수행하는 FC layer나 Conv layer와는 달리 임베딩 레이어는 Gather and Reduce (GnR) 연산을 수행한다. Gather 연산은 유저가 과거 선호한 아이템을 임베딩 테이블에서 찾으며, gather 연산 후 찾아진 임베딩 벡터들은 reduce 연산에 의해 하나의 벡터로 더해진다. 결국, Gather 연산은 임베딩 테이블에서 임베딩 벡터를 접근하는 연산이기 때문에 GnR 연산을 메모리 집약적으로 만든다.

한 편, 임베딩 테이블을 TT 형식으로 분해한 TT-embedding에서는 gather 연산이 TT-core 사이의 행렬 곱 연산인 TT-gather 연산으로 변경되고 reduce 연산은 동일하게 진행된다. 기존 임베딩 테이블에서는 임베딩 벡터를 접근하는 연산이 연속적인 행렬 곱 연산으로 바뀜에 따라 TT-embedding의 전체 연산은 계산 집약적으로 변한다. 또한, TT-gather 연산은 각 유저가 상호 작용하는 임베딩 벡터들을 개별로 연산하기 때문에 중복 연산이 생긴다. 이는 유저당 gather 평균 연산 수인 pooling factor (P)가 증가할 수록 증가하여 결국, TT-embedding의 inference time을 많이 증가시킨다.

TT-gather 연산으로 인해 증가하는 연산 횟수를 줄이기 위해 [3]에서는 대다수의 유저가 선호하는 아이템은 한정되어 있다는 점에서 착안하여 소프트웨어 캐시를 제안한다. 소프트웨어 캐시는 대다수의 유저가 자주 접근하는 임베딩 벡터는 TT 형식으로 분해하지 않고 저장하여 이에 필요한

이 논문은 2022년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.IITP-2017-0-00914, 지능형 IoT 장치용 소프트웨어 프레임워크)

연산량을 줄인다. 하지만 이 기법은 단순히 많이 접근되는 임베딩 벡터의 연산량만을 줄여 주기 때문에 TT-gather 연산에서 생기는 중복 연산을 줄이는 데 근본적인 해결책이 되지 못한다.

본 논문에서는 pooling factor가 큰 환경에서 TT-gather 연산에 생기는 불필요한 연산을 분석한다. 그리고 불필요한 연산을 줄이기 위한 새로운 연산인 Group reduced TT-Gather and Reduce (GRT-GnR) 연산을 제안한다. GRT-GnR 연산은 임베딩 테이블을 TT-embedding으로 만드는 과정에서 생기는 연산을 줄이기 때문에 개인 맞춤형 추천시스템에만 국한되지 않고 일반적으로 사용 가능하다.

2. 배경 이론

Tensor-Train(TT) 분해 기법은 딥 러닝 모델의 가중치의 메모리 요구 사항을 줄이기 위한 효율적인 근사 방법이다. TT 분해의 중요한 부분은 메모리를 많이 차지하는 텐서를 TT-core라고 불리는 작은 크기의 3차원 텐서의 집합으로 분해하는 것이다. 만약 d 차원 텐서인 $\mathcal{A} \in \mathbb{R}^{n_1 \times n_2 \times \dots \times n_d}$에 TT 분해 기법을 적용하면 d 개의 3차원 텐서인 TT-core $\mathcal{G}_k \in \mathbb{R}^{r_k \times n_k \times r_{k+1}}$, $k = 1, 2, \dots, d$ 로 분해되고 텐서 $\mathcal{A}$는 아래와 같은 식으로 복원된다.

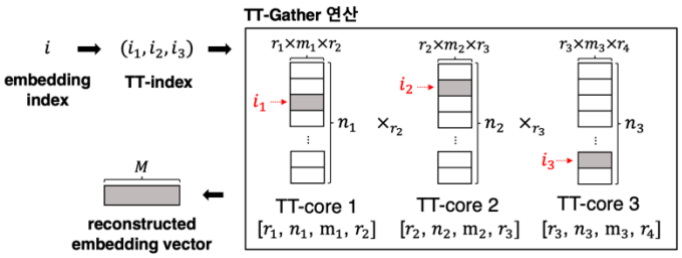
$$\mathcal{A}(i_1, i_2, \dots, i_d) = \mathcal{G}_1[i_1]\,\mathcal{G}_2[i_2] \dots \mathcal{G}_d[i_d] \quad (1)$$

이 때, $\mathcal{G}_k[i_k] \in \mathbb{R}^{r_k \times r_{k+1}}$ 는 k 번째 TT-core $\mathcal{G}_k$ 의 i 번째 슬라이스를 나타내고 r_k는 TT-core의 rank를 나타낸다.

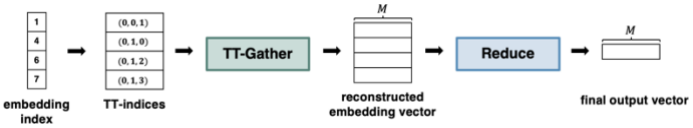
임베딩 테이블에 적용하기 위해 TT 분해 기법은 행렬 $W \in \mathbb{R}^{M \times N}$에도 적용될 수 있도록 일반화가 가능하다. 먼저 M 과 N을 $M = \prod_{k=1}^{d} m_k$와 $N = \prod_{k=1}^{d} n_k$로 분해하면 행렬 W는 $W \in \mathbb{R}^{(m_1 \times n_1) \times \dots \times (m_d \times n_d)}$로 나타낼 수 있다. 이 후, 행렬 W에 TT 분해 기법을 적용하면 d 개의 4차원 텐서인 TT-core $\mathcal{G}_k \in \mathbb{R}^{r_k \times m_k \times n_k \times r_{k+1}}$, $k = 1, 2, \dots, d$ 로 분해되고 행렬 W 는 아래와 같은 식으로 재구축된다.

$$\mathcal{W}((j_1, i_1), \dots, (j_d, i_d)) = \mathcal{G}_1[j_1, i_1]\,\mathcal{G}_2[j_2, i_2] \dots \mathcal{G}_d[j_d, i_d] \quad (2)$$

이 때, $\mathcal{G}_k[j_k, i_k] \in \mathbb{R}^{r_k \times r_{k+1}}$는 k번째 TT-core $\mathcal{G}_k$의 (j_k, i_k)번째 슬라이스를 나타내고 r_k는 TT-core의 rank를 나타낸다. 앞 선 수식들에 의해 임베딩 테이블의 메모리 요구량은 $N \times M$에서 $\sum_{i=0}^{d-1}(r_i \times r_{i+1} \times n_i \times m_i)$로 감소하여 TT 분해 기법을 유망한 압축 방법으로 만든다.



[그림 1] TT-gather 연산



[그림 2] TT-embedding에서의 연산 과정

3. TT-embedding

기존 임베딩 테이블와 TT-embedding의 차이점은 gather 연산이 TT-gather 연산으로 변경되는 점이다. 우리는 이 절에서 TT-gather 연산의 연산량 및 불필요한 연산이 생기는 과정을 설명한다.

3-1. TT-gather 연산

임베딩 테이블 $W \in \mathbb{R}^{M \times N}$가 TT 분해 기법에 의해 d개의 TT-core로 분해되면, 임베딩 테이블의 i번째 임베딩 벡터는 다음과 같은 식으로 복원된다.

$$\mathcal{W}(i) = \mathcal{G}_1[i_1]\,\mathcal{G}_2[i_2] \dots \mathcal{G}_d[i_d] \quad (3)$$

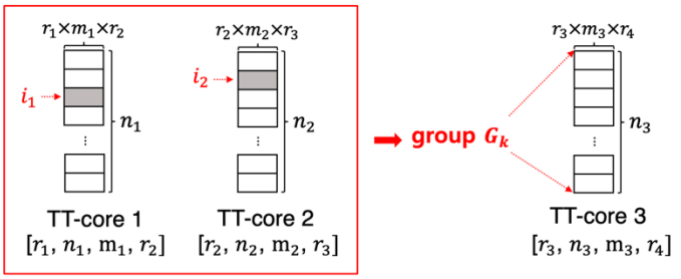
$\mathcal{G}_k[i_k] \in \mathbb{R}^{r_k \times m_k \times r_{k+1}}$은 k번째 TT-core의 i_k번째 슬라이스를 나타내며 $i = \sum_{k=1}^{d} i_k \prod_{c=i+1}^{d} n_c$이다. 우리는 $\{i_k\}_{k=1}^{d}$을 TT-index이라고 지칭한다. 또한, 수식 (3)에 의해 i번째 임베딩 벡터를 복원하기 위해 필요한 연산 수는 다음과 같다.

$$TT_G = 2\sum_{k=1}^{d}((r_k - 1) \times r_{k+1} \times \prod_{i=0}^{k} m_i) \quad (4)$$

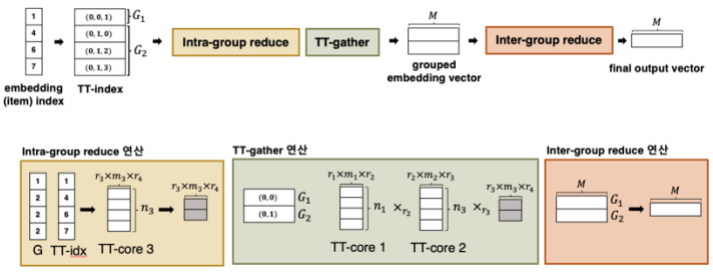
그림 1에서는 3개의 TT-core로 구성된 TT-embedding에서 i번째 임베딩 벡터의 TT-gather 연산 과정을 설명하고 있다. i는 각 TT-core의 TT-index로 변환된 후 TT-core 간 행렬 곱 연산이 진행하여 원래 임베딩 벡터를 복원한다.

3-2. TT-gather 연산에서의 중복 연산

개인 맞춤형 추천 시스템에서 각 유저는 여러 개의 아이템과 상호 작용하기 때문에 여러 개의 임베딩 벡터들을 접근한다. 이 과정에서 TT-gather 연산은 각 임베딩 벡터를 개별적으로 연산하기 때문에 같은 연산이 중복적으로 일어나게 된다. 그림 2에서 한 명의 유저가 4개의 임베딩 벡터를 접근할 때 3개의 TT-core를 가진 TT-embedding에서의 전체 연산 과정을 예시로 보여주고 있다. 먼저 1, 4, 6, 7번째 임베딩 벡터는 각각 (0, 0, 1), (0, 1, 0), (0, 1, 2), (0, 1, 3)인 TT-index로 변환된 후 임베딩 벡터 마다 개별적으로 TT-gather 연산이 진행된다. 하지만 4, 6, 7번째 임베딩 벡터를 복원하기 위한 TT-gather 연산이 진행될 때, 첫 번째 TT-core의 0번째 슬라이스 $\mathcal{G}_1[0]$와 두 번째 TT-core의 1번째 슬라이스 $\mathcal{G}_2[1]$ 사이의 행렬 곱 연산이 중복적으로 발생한다. 결국 pooling factor를 P라고 하면, 한 유저당 필요한 연산 수는 $P \times TT_G$ 만큼 발생한다. 따라서, pooling factor가 커질 수록 TT-gather 연산이 선형적으로 증가하게 된다.



[그림 3] TT-embedding에서의 group



[그림 4] GRT-GnR 연산

4. Group Reduced TT-Gather and Reduce (GRT-GnR) 연산

중복 계산 문제를 해결하기 위해, 우리는 Group Reduced TT-Gather and Reduce (GRT-GnR)이라는 TT-embedding의 대체 연산을 제안한다. GRT-GnR 연산은 행렬 연산은 결합 법칙이 성립한다는 점에서 비롯된다. 우리는 먼저 새로운 연산 단위인 group을 다음과 같이 정의한다. 이는 마지막 TT-core의 TT-index i_d, j_d가 주어졌을 때, 만약 나머지 TT-core의 TT-index인 $i_1, \dots, i_{d-1}$와 $j_1, \dots, j_{d-1}$가 서로 같으면 i_d와 j_d 는 서로 같은 group에 속한다고 정의한다. 즉, group은 마지막 TT-core의 TT-index를 제외한 나머지 TT-core들의 TT-index를 공유하는 임베딩 벡터들의 집합이다. 그림 3에서는 3개의 TT-core를 가진 TT-embedding에서 $\mathcal{G}_1[i_1]$와 $\mathcal{G}_2[i_2]$를 공유하는 group G_k를 나타내고 있다. 이처럼 만약 Group단위로 모든 임베딩 벡터들을 분리하면 다음과 같은 식으로 전체 임베딩 벡터들을 나타낼 수 있다.

$$G = \coprod_{k=1}^{N/n_d} G_k \ s.t. \ |G_k| = n_d$$

따라서 동일한 group에 포함된 임베딩 벡터의 경우, TT-gather가 수행되기 전에 마지막 TT-core의 TT-index 부분들을 먼저 누적하면, 결과의 변경 없이 중복 계산을 완전히 제거할 수 있다. 그리고 유저가 평균적으로 접근하는 group 수를 P_G라고 하면, 한 유저당 필요한 TT-gather 연산 수는 $P \times TT_G$에서 $P_G \times TT_G$로 줄어들게 된다.

그림 4에서는 GRT-GnR 연산의 전체 과정을 설명하고 있다. GRT-GnR 연산은 총 세 단계, Intra-group reduce 연산, TT-gather 연산, Inter-group reduce 연산으로 구성된다. Intra-group reduce 연산은 각 유저가 접근하는 임베딩 벡터들을 group단위로 묶는다. 그 후, 마지막 TT-core의 TT-index들을 group단위로 벡터 합을 진행하여 group마다 한 번씩 TT-gather 연산을 진행할 수 있도록 만든다. Intra-group reduce 연산을 통해 group단위로 나온 값을 각 group에 맞추어 TT-gather 연산을 진행한다. 마지막으로 group단위로 복원된 임베딩 벡터들을 group간 합 연산인 Inter-group reduce를 진행한다. 이 때, Intra-group reduce 연산 수와 Inter-group reduce 연산 수는 기존 reduce 연산 수보다 증가하지만 TT-gather 연산의 수가 확연히 줄어들기 때문에 큰 overhead가 되지 않는다.

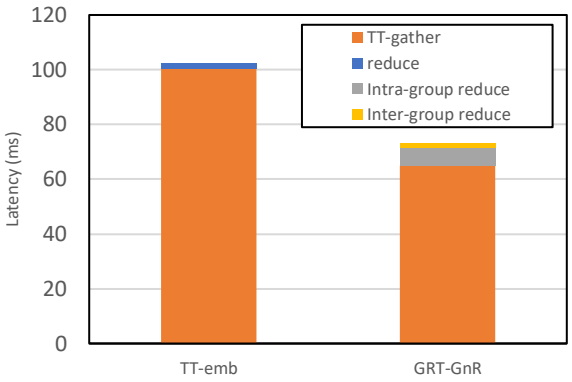
5. 실험 결과

5-1. 실험 환경

우리는 [3]에서 제안한 기존 TT-embedding 연산과 유사하게 GRT-GnR 연산을 구현하고 실험한다. 실험은 MovieLens 20m 데이터셋을 사용하여 기존 TT-embedding 연산과 GRT-GnR 연산의 latency를 비교한다. MovieLens 20m 데이터셋은 27,000개의 아이템(영화)을 138,000명의 유저가 평가한 데이터이다. 평가는 1~5로 평가되어 있지만 우리 실험에서는 유저가 아이템을 선호하는지가 중요하기 때문에 유저가 평점을 4이상 준 아이템들은 선호한다고 수정하였다. 그리고 pooling factor가 큰 환경에서 실험을 하기 위해 아이템을 4개 이하로 선호하는 유저들은 실험에서 제외하여 실제 실험에 사용된 유저 수와 아이템 수는 각각 65,000명과 20,500개이고 pooling factor는 45이다.

임베딩 테이블을 TT 분해할 때, 3개의 TT-core와 임베딩 테이블의 행과 열은 각각 [21, 22, 50], [4, 4, 4]로 분해한다. 또한, TT-rank로는 [1, 32, 32, 1]을 사용한다.

Latency를 분석하기 위해서 NVIDIA에서 제공하는 nsight system을 사용하여 profiling을 진행한다.



[그림 5] Latency 비교

5-2. 실험 결과

그림 5는 [3]에서 제안한 TT-embedding 연산과 GRT-GnR 연산을 비교하고 있다. 기존 TT-embedding 연산에서는 TT-gather 연산의 latency는 100.59ms 걸리는 데 반면 GRT-GnR 연산에서의 TT-gather 연산은 64.97ms로 latency가 36.5% 정도 개선된다. 또한, 전체 latency를 비교하였을 때, TT-embedding 연산은 102.38ms, GRT-GnR 연산은 73.52ms로 latency가 29% 정도 개선된다.

6. 결 론

본 논문에서는 TT-embedding 연산이 임베딩 벡터들을 개별적으로 연산할 때 불필요한 연산이 생긴다는 점을 찾아내고 이를 해결하기 위해 GRT-GnR 연산을 제안하였으며, latency를 29% 정도 개선한다.

참고문헌

[1] OSELEDETS, Ivan V. Tensor-train decomposition. SIAM Journal on Scientific Computing, 2011, 33.5: 2295-2317.

[2] HRINCHUK, Oleksii, et al. Tensorized embedding layers for efficient model compression. arXiv preprint arXiv:1901.10787, 2019.

[3] YIN, Chunxing, et al. Tt-rec: Tensor train compression for deep learning recommendation models. Proceedings of Machine Learning and Systems, 2021, 3: 448-462.