JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지 Vol. 26, No. 11: 1623~1629, Nov. 2022

확률분포 생성을 통한 근사 행렬 곱셈 간소화 방법

권오영[1] · 서경택[2*]

Probability distribution-based approximation matrix multiplication simplification algorithm

Oh-Young Kwon[1] · Kyoung-Taek Seo[2*]

[1]Professor, Department of Future Technology, Korea University of Technology & Education, Cheonan, 31253 Korea
[2*]Researcher, Office of Future Education Innovation (OFEI), Korea University of Technology & Education, Cheonan, 31253 Korea

요 약

행렬 곱셈은 과학 및 공학 분야에서 널리 사용되는 기본 연산이다. 딥러닝의 학습 알고리즘에도 행렬 곱셈이 많이 사용된다. 따라서 행렬 곱셈을 효과적으로 수행하기 위한 다양한 알고리즘들 개발하고 있다. 이중 행렬 곱셈의 연산량을 줄이는 방법으로 근사 행렬 곱셈 방법이 있다. 근사 행렬 곱셈은 행렬의 열과 행을 선택하기 위한 적절한 확률분포를 결정하고, 이 분포에 따라 행렬의 열과 행을 선택하여 근사 행렬 곱셈을 수행한다. 기존의 방법들을 행렬 곱셈에 참여하는 두 개의 행렬 A, B를 모두 고려하여 확률 분포를 생성한다. 본 논문은 행렬 A만을 대상으로 근사 행렬 곱셈에 사용될 행렬의 열과 행을 선택하는 확률 분포를 생성하는 방법을 제안하였다. 기존의 방법들과 제안된 방법들을 사용하여 1000x1000, 2000x2000, 3000x3000, 4000x4000, 5000x5000 행렬에 대하여 근사 행렬 곱셈을 수행하였다. 기존의 방법보다 제안한 방법을 적용한 근사 행렬 곱셈이 평균 0.02%에서 2.34%까지 원래 행렬 곱셈 결과에 더 근접하는 결과를 보였다.

ABSTRACT

Matrix multiplication is a fundamental operation widely used in science and engineering. There is an approximate matrix multiplication method as a way to reduce the amount of computation of matrix multiplication. Approximate matrix multiplication determines an appropriate probability distribution for selecting columns and rows of matrices, and performs approximate matrix multiplication by selecting columns and rows of matrices according to this distribution. Probability distributions are generated by considering both matrices A and B participating in matrix multiplication. In this paper, we propose a method to generate a probability distribution that selects columns and rows of matrices to be used for approximate matrix multiplication, targeting only matrix A. Approximate matrix multiplication was performed on 1000x1000 ~ 5000x5000 matrices using existing and proposed methods. The approximate matrix multiplication applying the proposed method compared to the conventional method has been shown to be closer to the original matrix multiplication result, averaging 0.02% to 2.34%.

키워드 : 행렬 곱셈, 유사도 샘플링, 연산 감소, 간소화
Keywords : Matrix multiplication, Similarity sampling, Computation reduction, Matrix Simplification

Received 27 January 2022, **Revised** 20 October 2022, **Accepted** 20 October 2022

*** Corresponding Author** Kyoung-Taek Seo (E-mail:kut1426@koreatech.ac.kr, Tel:+82-41-580-4743)
Researcher, Office of Future Education Innovation, Korea University of Technology & Education, Cheonan, 31253 Korea

Open Access http://doi.org/10.6109/jkiice.2022.26.11.1623 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

행렬 곱셈(Matrix Multiplication)은 공학 뿐 아니라 생물, 금융 등 다양한 분야에서 사용되는 알고리즘의 기본 연산으로 활용된다. 점점 더 복잡한 문제를 해결하는 방향으로 진행하고 있는 Deep Neural Networks는 문제들의 복잡성이 증가함에 따라 많은 행렬의 곱셈들이 사용되고 있다. 행렬 곱셈은 Convolution 또는 Fully-connected 레이어의 핵심 연산이고 대량의 연산시간을 소모하여 현재 **GotoBLAS** 알고리즘과 같은 다양한 알고리즘이 행렬 곱셈을 효과적으로 수행하기 위해 사용되고 있다[1]. 행렬 곱셈의 연산량을 줄이기 위하여 행렬의 열과 행을 선택하기 위한 적절한 확률분포를 결정하고, 이 분포에 따라 행렬의 열과 행을 선택하여 근사 행렬 곱셈을 수행한다. 기존 방법은 두 행렬 A, B를 이용하여 확률 분포를 계산한다. 본 논문에서 제안한 방법은 행렬 A만을 대상으로 확률 분포를 생성하여, 확률 분포를 계산하는 연산이 줄어드는 장점이 있다.

2장에서 행렬 곱셈의 연산을 감소하는 방법들과 근사 행렬 곱셈 방법에 대하여 설명하고, 3장에서 코사인 유사도와 유클리디안 거리에 기반한 확률 분포 생성 방법을 제안한다. 4장에서 제안한 방법들을 실험한 결과를 기술하고, 5장에 본 논문의 결론과 향후 연구 방향을 제시한다.

Ⅱ. 행렬 곱셈 연산 감소 방법

2.1. Fast Matrix Multiplication(FMM)

Strassen's algorithm은 divide-and-conquer 방식의 행렬 곱셈 알고리즘으로 곱셈을 덧셈으로 처리하여 복잡도를 낮출 수 있다[2]. $n \times n$ 크기의 두 행렬을 곱하면 $O(n^3)$의 시간이 소요되지만 이 알고리즘은 대략 $O(2^{2.807})$의 시간이 소요된다. ***Strassen's algorithm***을 변형한 ***Winograd algorithm***은 점근적으로 슈트라센 알고리즘과 동일한 복잡도를 지니지만 덧셈의 횟수를 줄인 방법이다[3]. 블록 행렬을 곱하는 기본 방법은 8개의 곱셈과 4번의 덧셈을 필요로 하지만 ***Strassen's algorithm***은 7번의 곱셈과 18번의 덧셈으로 수행하며, ***Winograd algorithm***의 경우 동일한 곱셈과 15번의 덧셈으로 수행할 수 있다.

2.2. Column-Row Sampling(CRS)

CRS(Column-Row Sampling)방법은 $m \times d$ 행렬 A와 $d \times n$ 행렬 B에서 k개의 열을 A에서 선택하고, 그에 해당하는 B의 k개의 행을 샘플링하여 행렬 C와 행렬 R을 형성한 뒤 AB의 근사치인 CR을 계산하는 방법이다[4].

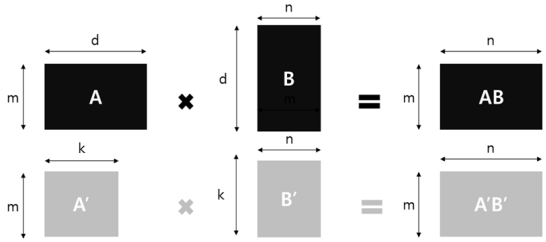

Fig. 1 Basic Idea for CRS

행렬 A와 행렬 B의 행렬곱셈을 다음과 같이 표현할 수 있다.

$$AB = \sum_{i=1}^{n} A^i B_i \tag{1}$$

여기서, A^i는 행렬 A의 i번째 열을 의미하고, B_i는 행렬 B의 i번째 행을 의미한다. 만일 확률분포 $\{p_i\}_{i=1}^{n}$이라면, 이 분포를 활용하여 행렬 A에서 k개의 열을 임의로 선택하여 행렬 C를 만들고, A의 열에 해당하는 B의 행을 k개 선택하여 행렬 R을 구성한다. [4]는 C와 R을 구성할 때 A에서 선택된 열 A^{i_t}와 B의 해당하는 행 B_{i_t}에 대하여 $1/\sqrt{kp_{i_t}}$ 값을 곱한다. 즉, C와 R의 열과 행은 각각 $C^t = A^{i_t}/\sqrt{kp_{i_t}}$와 $R_t = B_{i_t}/\sqrt{kp_{i_t}}$로 구성된다.

$$\begin{aligned} CR &= \sum_{t=1}^{k} C^t B_t \\ &= \sum_{t=1}^{k} \frac{A^{i_t}}{\sqrt{kp_{i_t}}} \frac{B_{i_t}}{\sqrt{kp_{i_t}}} \\ &= \sum_{t=1}^{k} \frac{1}{kp_{i_t}} A^{i_t} B_{i_t} \approx AB \end{aligned} \tag{2}$$

CR은 AB의 근사치로 $\| AB - CR \|_F^2$가 최소화 되는 p_i는 다음과 같다.

$$p_i = \frac{|A^i||B_i|}{\sum_{j=1}^{n}|A^j||B_j|} \quad (3)$$

여기서 $|A^i|$는 Euclidean Distance이고, $\| A \|_F$는 행렬의 Frobenius Norm이다.

CRS 방법은 확률분포 $\{p_i\}_{i=1}^{n}$를 계산하는 부담이 증가하지만, CR로 AB의 근사치를 구하는 시간이 $O(mdn)$에서 $O(mkn)$로 감소하게 된다. Approximating Matrix Multiplication [4]는 확률 p_i를 $O(1)$에 계산할 수 있는 Select 알고리즘을 제안하였다. [4]는 순차적으로 입력되고, 크기가 결정되지 않은 매우 큰 행렬을 다룰 때 적합한 알고리즘이다. [5]은 확률 p_i를 uniform weight를 갖는 $1/n$로 설정하고, nonuniform weight를 갖는 $\frac{w_i}{\sum_{j=1}^{n} w_j}$로 설정하여, 각각의 확률 분포를 이용한 근사 행렬곱셈의 오차를 조사하였다.

확률 분포 에 따라서 열과 행을 선택하는 것이 아니라 값이 큰 k개를 선택한 방법으로 결정적 top-k 샘플링 방법이 제안된다. 그리고, top-k 방법은 scaling factor를 포함하지 않아도 근사치 행렬곱셈의 오류가 적다고 주장하였다.[6] 또한 CRS 방법의 변형인 Bernoulli-CRS 방법을 제시되고 있으며 *Bernoulli* 분포 확률값(p_i)을 사용하여 행렬 A의 열에 부여하고$\sum_{i=1}^{n} p_i = k$ 가 되도록 행렬 A의 열을 선택하였다[7].

2.3. Blockwise CR Multiplication

CRS와 유사하나 하나의 컬럼을 선택하는 것이 아니라 몇 개의 컬럼을 블럭으로 선택한다[8]. 행렬 $A \in R^{m \times n}$와 행렬 $B \in R^{n \times p}$를 k개의 부분 행렬로 구성한다. 부분 행렬은 $\tau = n/k$개의 열이나 행으로 구성된다. 부분 행렬 $\widetilde{A^i} = R^{m \times \tau}$와 부분 행렬 $\widetilde{B_i} = R^{\tau \times p}$이라고 하면 $A = [\widetilde{A^1} \cdots \widetilde{A^k}]$와 $B = [\widetilde{B_1^T} \cdots \widetilde{B_k^T}]^T$가 된다. 블럭을 선택하기 위하여 각 블럭의 확률은 다음과 같이 정하였다. 벡터가 아니라 부분 행렬이 되므로 각 부분 행렬의 Frobenius norm을 사용하였다.

$$\prod_i = \frac{\| \widetilde{A^i} \|_F \| \widetilde{B_i} \|_F}{\sum_{l=1}^{k} \| \widetilde{A^l} \|_F \| \widetilde{B_l} \|_F} \quad (4)$$

*CRS*와 유사하게 위 확률을 이용하여 블럭을 선택하여 근사행렬값을 구할 수 있다. 이 방법은 행렬을 부분 행렬로 나누어서 처리하므로 분산처리등 다른 방법과 결합하여 대규모 행렬을 처리하는데 사용될 수 있다.

2.4. 근사행렬곱셈를 위한 확률분포생성

기존 제안된 대부분의 근사 행렬 곱셈을 구하는 방법은 다음과 같이 정리할 수 있다.

$$AB \approx CR = \frac{1}{k}\sum_{i=1}^{k}\frac{1}{p_i}A^i B_i \quad (5)$$

행렬의 열과 행을 선택하기 위한 적절한 확률분포를 결정하고, 이 분포에 따라 행렬의 열과 행을 선택하여 근사 행렬 곱셈을 수행한다. 행렬의 원소 값들이 정규분포를 따른다는 가정하고, 행렬을 구성하는 열 벡터들과 행 벡터들의 norm 크기로 확률값을 정한 것을 알 수 있다. 행렬이 구성되는 특징에 따라서 확률 분포를 결정하면 원래 행렬 곱셈과 유사한 근사 행렬 곱셈을 구할 수 있을 것이다.

Ⅲ. 유사도에 의한 확률 분포 결정

기존 방법은 행렬 *A*와 행렬 *B*를 모두 고려하여 행렬의 열과 행을 선택하기 위한 확률 분포를 결정하였다. 본 논문에서는 행렬 *A*의 연속된 두 열 벡터의 유사도를 기반으로 확률 분포를 설정하는 방법들을 제안하였다.

3.1. 코사인 유사도

두 벡터의 유사도를 구하는데 코사인 유사도를 사용할 수 있다. 행렬 A의 두 열 벡터 $|A^i|$와 $|A^j|$, $i \neq j$에 대하여 코사인 유사도는 $\frac{A^i \cdot A^j}{\| A^i \| \| A^j \|}$ 이다. 두 벡터가 방향이 같고 사이각이 작으면 유사도는 1에 가까운 값을 갖고, 서로 직교하면 0을 갖고, 반대 방향이면 -1에 가까운 값을 갖게 된다. -1에서 1사이의 값을 0에서 1사이의 값으로 치환하기 위하여 코사인 유사도를 식(6)과 같이 변경하였다.

$$\left(1 + \frac{A^i \cdot A^j}{\| A^i \| \| A^j \|}\right)/2 \quad (6)$$

인접한 두 열 벡터 $|A^{i-1}|$와 $|A^{i}|$에 대하여 유사도를 구하고, 이 값을 이용하여 $|A^{i}|$의 확률 값을 결정한다. 만일 유사도가 높으면 선택될 확률 값이 높게 부여되고, 서로 다르면 확률 값은 낮게 부여된다. 이렇게 확률 값을 부여하면 차이가 많이 발생하는 열 벡터를 제외할 가능성이 커진다. 반대로 유사도가 낮은 벡터에 높은 확률 값을 부여하면 차이가 많이 발생하는 열 벡터를 선택할 가능성이 높아진다. 본 논문에서 제안한 유사도를 기반으로 확률분포를 만드는 알고리즘은 표 1과 같다.

상대적 확률분포 생성 알고리즘을 변형하여 열 벡터를 2개씩 짝(pair)을 지어서 서로 유사도가 높은 열 벡터 짝의 확률 값을 크게 설정할 수 있다. 표 2의 짝 기반 확률분포 생성 알고리즘은 상대적 확률분포 생성 알고리즘에 비하여 연산 양을 절반으로 줄일 수 있다.

표 1과 표 2에 제시된 알고리즘은 인접한 열 벡터의 유사도가 비슷할 경우 높은 확률 값을 갖는다. 행렬 A의 열 벡터들이 유사도가 낮으면 서로 다른 특징을 갖고 있다고 생각할 수 있다. 유사도가 낮은 열 벡터에 높은 확률 값을 부여하려면 표에서 구한 s값을 $1-s$로 재 설정하면 된다. 코사인 유사도를 이용한 개별 값($s(1:n)$)은 [0~1]의 값이므로 *Bernoulli* 분포 확률값(p_i)으로도 사용할 수도 있다.

Table. 1 Relative Probability Distribution Generation

Algorithm 1) 상대적 확률분포 생성
1: $s_1 = 1$
2: **for** i **in** 2 **to** n **do**
3: $s_i = \left\{\left(1 + \frac{A^{i-1} \cdot A^{i}}{\| A^{i-1} \| \, \| A^{i} \|}\right)/2\right\}$
4: **end for**
5: *prob(1:n)* = $s(1:n)$/*sum*($s(1:n)$)

Table. 2 Pair-wise Probability Distribution Generation

Algorithm 2) 짝기반(pair-wise) 확률분포 생성
1: **for** i **in** 1 **to** *n/2* **do**
2: $s_{2i-1} = s_{2i} = \left\{\left(1 + \frac{A^{2i-1} \cdot A^{2i}}{\| A^{2i-1} \| \, \| A^{2i} \|}\right)/2\right\}$
3: **end for**
4: *prob(1:n)* = $s(1:n)$/*sum*($s(1:n)$)

3.2. 유클리디언 거리 유사도

행렬 A의 첫 번째 열은 norm값으로 거리 값을 설정하고, 인접한 두 벡터 사이의 유클리디안 거리를 구한다. 두 벡터 사이의 거리가 멀다면 벡터 간의 변화가 많이 발생한 것이라 생각할 수 있다. 표 1에 제시된 알고리즘에 첫 번째 줄의 1대신 행렬 A의 첫 번째 열을 놈을 구한다. 3번째 줄의 값은 인접한 두 열 벡터 $|A^{i-1}|$와 $|A^{i}|$의 유클리디언 거리로 변경하면 유클리디언 거리 기반의 유사도를 구할 수 있다. 표1의 5번째 줄 직전에 열 벡터의 거리를 [0~1] 사이로 정규화하는 코드를 추가하면 코사인 유사도와 동일하게 확률 값을 구할 수 있다. 그리고, 이 정규화 된 값을 *Bernoulli* 분포 확률값으로 사용할 수도 있다.

이전 논문들은 확률 값을 구하는데 행렬 A의 열 벡터 norm과 행렬 B의 행 벡터 norm을 모두 구하였다. 유사도는 행렬 A에 대해서만 연산을 수행하므로 확률 분포를 계산하는 연산량을 줄일 수 있다. 그리고, [4]의 Select 알고리즘에 대신 유사도 값을 사용할 수도 있다.

Ⅳ. 실험결과

본 논문에서 제안한 유사도 기반 확률 분포를 적용하여 근사 행렬 곱셈을 수행하였다. 행렬 곱셈 *AB*와 근사 행렬 곱셈 *CR*의 차이가 적을수록 식 (7)의 값은 1에 가까워진다. 이 값을 근접률이라고 하겠다.

$$1 - \frac{\| AB - CR \|_F^2}{\| AB \|_F^2} \tag{7}$$

실험은 Intel(R) Core(TM) i9-10900K CPU @ 3.70GHz 프로세서와 64GB 메모리를 갖춘 컴퓨터에서 수행되었다. [7]에서 제안한 방법인 *top-k*, *CRS*, *Bernoulli*과 본 논문에서 제안한 유사도를 사용한 방법을 비교하였다. [7]에서 제공한 코드에 제안한 방법을 추가하고, WSL[9]을 이용하여 Ubuntu를 설치하고, 실험을 수행하였다.

실험의 편의를 위해 $n \times n$ 정방행렬을 사용하였다. *k*는 행렬의 60%, 70%, 80%, 90%가 선택되는 값으로 정했다. 실험에 사용된 행렬의 크기 *n*에 대한 각 행렬에서 사용한 *k*값을 표 3에 표시하였다. 예로, 1000×1000 행렬의 경우 600개의 열 벡터를 선택하면 행렬 A의 60%에 해당하는 열 벡터들을 이용하여 근사 행렬 C를 구성한다.

Table. 3 k-value for each matrices.

n	60%	70%	80%	90%
1000	600	700	800	900
2000	1200	1400	1600	1800
3000	1800	2100	2400	2700
4000	2400	2800	3200	3600
5000	3000	3500	4000	4500

본 논문은 주어진 행렬의 열과 행을 선택하기 위해 행렬 A만을 대상으로 확률 값을 구하는 방법을 제안하였다. 따라서 수행시간보다는 식 (7)의 근접률을 측정하는 것이 제안한 방법의 효율성을 더 잘 보여줄 수 있다.

실험은 top-k(TK), CRS(CR), Bernoulli(BN), 상대코사인유사도(RC), 상대코사인유사도 Bernoulli(RB), 짝코사인유사도(PC), 거리유사도(DI) 방법을 사용하였다. 코사인 유사도가 높은 열 벡터, 거리가 가까운 열 벡터에 높은 확률 값이 부여되도록 하였다.

표 4는 1000×1000 행렬에 각 방법을 적용한 실험결과를 표시하였다. 그림 1은 표 4의 근접률 정보를 그래프로 표시한 것이다. 근사 행렬 곱셈에 대한 실험 결과는 그림 3에서 그림 6까지 각각 2000×2000, 3000×3000, 4000×4000, 5000×5000 행렬에 대한 근접률을 나타내고 있다.

[7]은 *top-k* 방법이 *scaling factor* $(1/kp_i)$를 포함하지 않아도 근사 행렬 곱셈의 오류가 적다고 주장하였다. 하지만 본 논문에서 실험한 결과 *top-k* 방법은 근사 행렬 곱셈에서 선택한 행렬의 양 만큼 근접률을 보였다.

Tale. 4 Approximation Rate of 1000×1000matrices

	60%	70%	80%	90%
TK	0.61292	0.71186	0.80997	0.90529
CR	0.93125	0.93328	0.94001	0.94204
BN	0.95423	0.96244	0.95918	0.98164
RC	0.92998	0.93516	0.93787	0.94123
RB	0.96669	0.96815	0.96811	0.96621
PC	0.92970	0.93451	0.93764	0.94211
DI	0.91773	0.92932	0.93338	0.93769

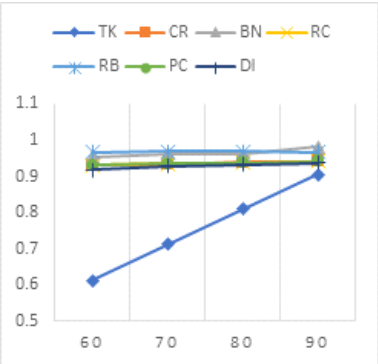

Fig. 2 1000×1000 matrix multiplication

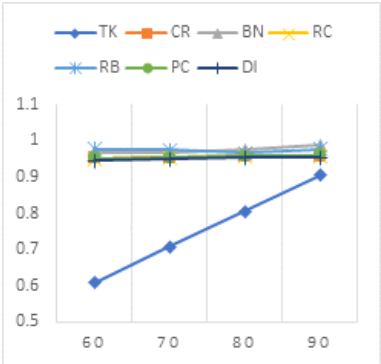

Fig. 3 2000×2000matrix multiplication

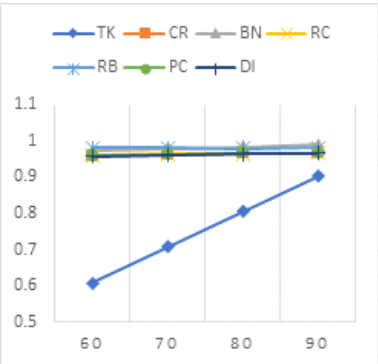

Fig. 4 3000×3000matrix multiplication

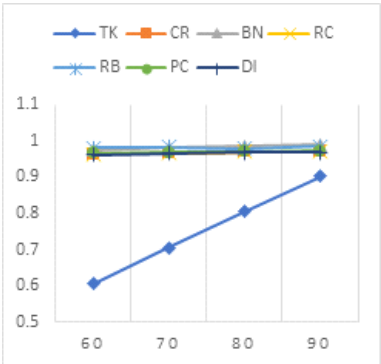

Fig. 5 4000×4000matrix multiplication

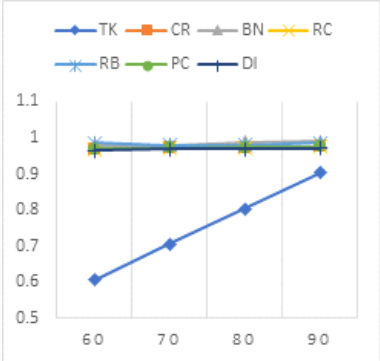

Fig. 6 5000×5000matrix multiplication

실험 결과 *scaling factor*를 곱하고, 비복원 추출을 사용한 방법이 *top-k* 방법보다 근사 행렬 곱셈의 근접률이 높았다. 그림 3에서 그림 6까지 실험 결과를 보면 CR방법과 본 논문에서 제안한 RC, PC, DI방법이 비슷한 성능을 보여주고 있다. 모든 행렬에 대하여 CR방법을 기준으로 RC, PC, DI의 근접률이 각각 평균 0.38%, 0.41%, 0.02% 개선되었다. k 값에 의하여 선택된 매트릭스의 크기에 따라 개선률이 다르지만 모든 결과를 평균하여 위와 같은 결과를 얻었다. CR과 유사한 근접률을 보이지만 본 논문에서 제안한 방법은 확률 분포를 생성하는 연산이 줄어드는 장점이 있다. CR방법 보다 BN방법이 근접률을 2.33%정도 더 개선할 수 있었다. 제안한 상대코사인유사도를 이용한 Bernoulli(RB)방법도 BN과 유사한 근접률 개선을 보였다. 실험 결과가 나타난 그림 3에서 그림 6까지 살펴보면 RB방법은 행렬의 크기가 작고, k 값을 60%, 70%가 되는 값으로 선택했을 때 BN보다 성능이 좋았다. BN방법은 k값이 커질수록 근접률이 많이 개선이 되었다. 즉 90%정도 행렬을 선택하면 BN방법의 근접률이 RB보다 높은 값을 갖게 된다. 반면 RB방법은 k값의 변화에도 근접률의 개선 정도가 일정하게 유지 되고 있었다.

V. 결 론

근사 행렬 곱셈을 수행하는 알고리즘은 행렬의 열과 행을 선택하기 위한 적절한 확률분포를 결정하고, 이 분포에 따라 행렬의 열과 행을 선택하여 근사 행렬 곱셈을 수행한다. 열과 행을 선택하기 위하여 기존에 제안한 대부분의 방법들을 행렬 곱셈에 참여하는 두 개의 행렬 A, B를 모두 고려하여 확률 분포를 생성한다. 본 논문에서는 하나의 행렬 A만을 사용하여 코사인 유사도와 유클리디언 거리에 의한 확률 분포를 결정하고, 이 확률 값들을 이용한 근사 행렬 곱셈을 수행하는 방안에 대한 연구와 실험을 진행하였다.

모든 크기의 행렬에 대하여 CR방법을 기준으로 RC, PC, DI의 근접률이 각각 평균 0.38%, 0.41%, 0.02% 개선되었고, BN과 RB방법은 근접률이 각각 평균 2.33% 2.34%로 개선되었다. 본 논문에서 제안한 방법은 행렬 A만을 대상으로 확률 분포를 생성하여, 확률 분포를 계산하는 연산이 줄어드는 장점이 있다.

최근 딥러닝에 사용되는 Convolution 또는 Fully-connected 레이어 연산에서 행렬 곱셈이 사용된다. 딥러닝을 수행하는 과정에 제안한 확률 분포 생성 방법을 적용한 근사 행렬 곱셈을 수행하면 전체적인 연산량을 감소 시켜서 학습을 빨리 진행할 수 있다. 본 논문에서 행렬의 원소들을 정규분포를 따르는 랜덤값으로 설정하였다. 향후 실제 딥러닝 학습 알고리즘에 제안한 방법을 적용하여 일정한 성능을 유지하면서 전체적인 연산량을 얼마나 감소할 수 있는지 추가적인 연구가 필요하다. 아울러 GPU를 활용하여 본 논문에서 제안한 방법을 포함한 식 (5)의 근사 행렬 곱셈을 수행하는 범용 라이브러리를 개발할 필요가 있다.

ACKNOWLEDGEMENT

This paper was studied by supporting the 2020 KOREATECH professor's educational research promotion.

References

[1] K. Goto and R. A. van de Geijn, “Anatomy of High Performance Matrix Multiplication,” *ACM Transactions on Mathematical Software*, vol. 34, no. 3, pp. 1-25, May 2008.

[2] J. -G. Dumas and V. Pan, “Fast Matrix Multiplication and Symbolic Computation,” *ArXiv, arXiv:1612.05766*, Dec. 2016.

[3] P. Lai, H. Arafat, V. Elango, and P. Sadayappan,

"Accelerating Strassen-Winograd's Matrix Multiplication Algorithm on GPUs," in *Proceedings of 20th Annual InternationalConference on High Performance Computing*, Bengaluru, India, pp. 139 – 148, 2013.

[4] P. Drineas, R. Kannan, and M. W. Mahoney, "Fast Monte Carlo Algorithms for Matrices I: Approximating Matrix Multiplication," *SIAM Journal on Computing*, vol. 36, no. 1, pp. 132 – 157, Jul. 2006.

[5] B. Plancher, C. D. Brumar, I. Brumar, L. Pentecost, S. Rama, and D. Brooks, "Application of Approximate Matrix Multiplication to Neural Networks and Distributed SLAM," in *Proceedings of 2019 IEEE High Performance Extreme Computing Conference (HPEC)*, Waltham: MA, USA, pp. 1-7, 2019.

[6] M. Adelman, K. Y. Levy, I. Hakimi, and M. Silberstein "Faster Neural Network Training with Approximate Tensor Operations," *arXiv, arXiv: 1805.08079*, Oct. 2021.

[7] Y. Xiong, Z. Zeng, R. Chakraborty, M. Tan, G. Fung, Y. Li, and V. Singh, "Nyströmformer: A Nyström-based Algorithm for Approximating Self-Attention," *ArXiv, arXiv:2102.03902*, Feb. 2021.

[8] N. Charalambides, M. Pilanci, and A. O. Hero, "Approximate Weighted Coded Matrix Multiplication," in *Proceedings of 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Toronto: ON, Canada, pp. 5095-5099, 2021.

[9] Microsoft, Install Linux on Windows with WSL [Internet]. https://learn.microsoft.com/ko-kr/windows/wsl/install.

권오영(Oh-Young Kwon)

연세대학교 (Ph.D-컴퓨터과학)
현 한국기술교육대학교 융합학과 교수
※관심분야 : 고성능컴퓨팅, 임베디드 시스템, 시스템 소프트웨어

서경택(Kyoung-Taek Seo)

한국기술교육대학교(석사-컴퓨터공학)
현 한국기술교육대학교 미래교육혁신처 기술연구원
※관심분야 : 임베디드 시스템, 시스템 소프트웨어