딥러닝 분산 학습 관련 연구

류성원
서울대학교 산업공학대학원
데이터 마이닝 연구실
lyusungwon@dm.snu.ac.kr

개요

본 논문에서는 딥러닝의 분산 학습에 관한 개념들과 그에 관련된 연구들을 간략하게 다루고자 한다. 딥러닝 분산은 크게 모델 분산과 데이터 분산으로 나누어 진다. 데이터 분산을 위해서는 모델의 복제가 이루어 지는데 이러한 복제된 모델들의 동기화 방법에 따라 동기적 동기화와 비동기적 동기화로 나누어진다. 또한 분산의 구조는 분산을 위한 노드들의 역할이나 커뮤니케이션 방식에 따라 파라미터 서버 구조와 집단 통신 구조로 나뉠 수 있다. 그리고 마지막으로 딥러닝의 분산 학습이 가장 적극적으로 활용되고 있는 분야 중의 하나인 강화학습에서의 분산 연구를 간략히 다룰 것이다.

키워드

딥러닝 분산 학습, 분산 동기화, 분산 구조, 강화학습 분산

1 서론

기존의 데이터 분산 처리 프레임워크들은 데이터 마이닝이나 기계 학습에 필요한 기능을 지원하는데 초점을 맞추어 설계되었다. 분산 처리는 많은 양의 데이터에 대한 연산을 수행할 때 군집의 장비를 활용하여 처리 속도를 가속화하는 방법이다. 분산 처리에서 필요한 장비간 통신, 연산 효율화를 위한 프레임워크들이 등장하여 보편적으로 사용되고 있다[10, 11].

하지만 딥러닝의 학습은 기존의 대규모 데이터 처리와는 다르다. 딥러닝의 학습은 단순히 데이터를 처리하는 것이 아니라 데이터의 처리를 통하여 모델의 파라미터를 업데이트 하는 것이 목적이다. 이를 위해서는 목적함수에 대한 파라미터들의 미분값을 구하는 역전파가 이루어져야 하는데 이러한 연산을 병렬로 가속화 하기 위하여 GPU 를 활용한다. 그렇기 때문에 딥러닝의 분산 처리는 CPU 보다는 GPU 자원의 분산 활용을 위하여 이루어진다. 그렇기 때문에 기존의 분산 처리 프레임워크들은 딥러닝 분산학습에 적합하지 않다.

본 논문에서는 딥러닝의 분산 학습에 관한 개념들과 그에 관련된 연구들을 다루고자 한다. 딥러닝 분산은 크게 모델 분산과 데이터 분산으로 나누어 진다. 모델 분산은 큰 사이즈의 모델을 여러 노드에 나누어서 보관하고 업데이트 하는 방법이고 데이터 분산은 데이터를 분할하여 모델의 학습을 병렬로 진행하는 방법이다. 기존의 분산 처리는 데이터 분산에 대응되기 때문에 이후에는 데이터 분산에 주로 초점을 맞출 것이다. 데이터 분산을 위해서는 모델의 복제가 이루어 지는데 이러한 복제된 모델들의 동기화 방법에 따라 동기적 동기화와 비동기적 동기화로 나누어진다. 또한 분산의 구조는 분산을 위한 노드들의 역할이나 커뮤니케이션 방식에 따라 파라미터 서버 구조와 집단 통신 구조로 나뉠 수 있다. 그리고 마지막으로 딥러닝의 분산 학습이 가장 적극적으로 활용되고 있는 분야 중의 하나인 강화학습에서의 분산 연구를 간략히 다룰 것이다.

2 분산의 종류

[1]은 여러 장비의 자원들을 활용하여 딥러닝을 학습하는 딥러닝 분산학습을 처음으로 정식화하였다. [1]은 딥러닝 분산을 목적에 따라 크게 두 가지로 분류하였다. 하나는 모델의 파라미터를 여러 장비에 나누어 보관하는 모델 분산이고 다른 하나는 모델의 학습을 가속화 하기 위하여 여러 장비에 데이터들의 학습을 병렬화 하는 데이터 분산이다.

2.1 모델 분산

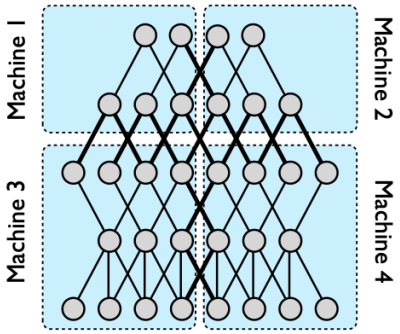

그림 1: 모델 분산 [1]

모델 분산이란 여러 장비에 모델을 나누어 담는 분산 방식이다. 이는 학습의 속도와는 관련이 없고 단순히 모델의 크기가 너무 커서 하나의 장비의 메모리에서 감당하기 힘들 경우 여러 머신을 이용하는 방식이다.

딥러닝이 많은 작업에서 우수한 성능을 보이고 있는데 특히 딥러닝의 크기에 따라서 성능이 향상되는 모습을 보이며 근래의 많은 딥러닝 모델들은 매우 큰 용량을 차지한다. 하지만 이를 학습하기 위해서는 메모리 또는 GPU 의 메모리에 올라가는 것이 중요한데 용량이 매우 큰 딥러닝 모델의 경우에는 하나의 장비의 GPU 에 올라가지 못한다. 그렇기 때문에 여러 장비의 메모리 또는 GPU 의 메모리를 활용하기 위하여 모델 분산을 제안한다.

모델 분산에서 중요한 부분은 각 장비간의 통신이다. 왜냐하면 딥러닝의 특성상 모델을 학습하기 위하여 모델의 파라미터들이 그래디언트를 연계적으로 역전파해야 하기 때문이다. 한번의 학습을 위하여 모든 파라미터들의 역전파를 구해야 하기 때문에 파라미터들 간에는 빈번한 통신이 발생할 수 밖에 없다. 그렇기 때문에 모델 분산을 할 때에는 파라미터들 간의 통신이 최소화가 되도록 모델을 분할하는 것이 좋다.

2.2 데이터 분산

데이터 분산은 딥러닝의 학습을 가속화 하기 위하여 사용하는 방법은 데이터 분산이다. 대규모 데이터 처리에서 일반적으로 처리속도를 가속화 하기 위하여 사용하는 분산은 딥러닝에서의 모델 분산보다는 데이터 분산에 가깝다. 하지만 기존의 데이터 분산처리 프레임워크들과 딥러닝 데이터 분산이 다른 점은 기존의 데이터 분산처리 프레임워크들의 목적은 데이터들을 처리하여 결과물을 내는 것이 목적이었으나 딥러닝 데이터 분산은 데이터를 학습하여 모델의 파라미터들을 업데이트 하는 것을 목적으로 한다는 점이다. 하지만 딥러닝 모델의 학습 방법은 최적화이기 때문에 데이터를 분산으로 학습하여 파라미터들을 업데이트 하는 것이 기존의 분산 처리만큼 쉽지 않고 이를 해결하기 위한 많은 연구들이 진행되고 있다. 앞으로 소개할 딥러닝 분산에 관한 연구들은 모두 데이터 분산에 속한다.

2.3 혼합

제안된 두 가지 분산 방법은 서로 분리된 것이 아니라 둘이 함께 사용될 수 있다.

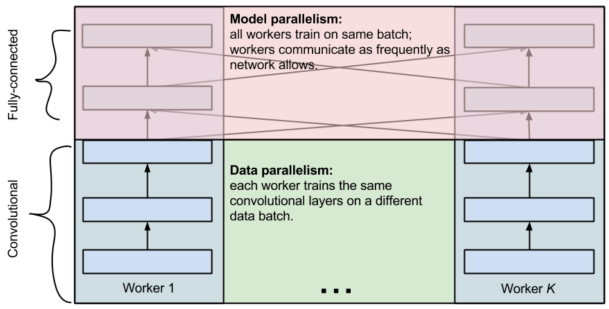

그림 2: 혼합 분산 활용 [2]

[2]에서는 컨볼루션 네트워크를 학습하기 위하여 모델 분산과 데이터 분산을 동시에 활용하는 방법을 제안한다. 이 논문은 일반적인 컨볼루션 네트워크가 컨볼루션 층과 풀리 커넥티트 층으로 나누어져 있고 컨볼루션 층이 웨이트 중에서 약 5%의 크기를 차지하지만 약 95%의 연산을 담당하고 풀리 커넥티드 층은 95%의 크기를 차지하지만 약 5%의 연산만들 담당하고 있다는 점을 지적하였다. 이를 통하여 풀리 커넥티드 층은 모델 분산을 통하여 웨이트들을 병렬화 하고 컨볼루션 층은 데이터 분산을 통하여 학습한 그래디언트들을 모델 분산에 전파하는 방법을 제안하였다. 이를 통하여 상당히 빠르면서 좋은 성능의 모델을 학습할 수 있었다.

3 동기화 방식

데이터 분산을 위해서 여러 장비에 한 모델을 복사해서 각각 학습하게 된다. 분할된 데이터들을 통하여 모델들의 파라미터들을 업데이트 하는데 이러한 파라미터들의 업데이트들은 분할된 데이터들로 인하여 제각각 이루어지게 된다. 이 복사본들의 동기화를 하는 방식이 딥러닝 분산 학습의 중요한 주제 중의 하나이다. 딥러닝 모델들의 파라미터 동기화 방식은 크게 동기적(Synchronous) 동기화 방식과 비동기적(Asynchornous) 동기화 방식으로 나뉘고 각각의 장점을 혼합하기 위한 방법들이 제안되고 있다.

3.1 비동기적 동기화

비동기적 동기화는 복제된 여러 모델들이 자신에게 할당된 데이터를 학습한 즉시 학습한 그래디언트를 각자 업데이트 하는 방식이다. 비동기적 방식의 장점은 시간 당 처리하는 데이터의 양이 많다는 점이다. 각 장비가 학습하는 시간의

편차가 있을 수 있는데 각 장비가 빨리 학습하는 대로 파라미터를 갱신하여 다음 학습을 할 수 있기 때문이다. 이러한 이유로 초기의 딥러닝 분산에는 비동기적 방식이 선호되었다.

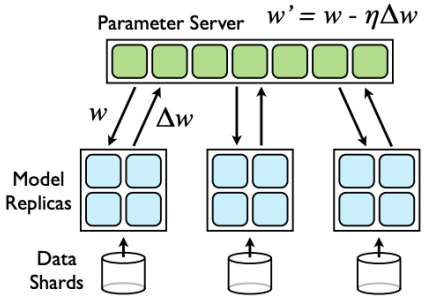

그림 3: Downpour SGD [1]

[1]에서는 비동기적 동기화 방법으로 Downpour SGD 를 제안하였다. Downpour 방식은 학습 노드들과 파라미터 서버를 담당하는 노드들을 분리한 파라미터 서버 분산 구조를 취하고 있는데 각 학습 노드들은 학습을 하는 즉시 각자 그래디언트를 파라미터 서버에 보내어 파라미터 서버를 비동기적으로 업데이트하는 방식이다. 이러한 구조에서는 파라미터 서버와 장비의 통신이 많아질 수 있다. 기존에는 이러한 통신을 할 때 마다 파라미터 서버 내용을 안전하게 보존하기 위하여 다른 통신들의 접근을 제한하는 잠금이 이루어 졌는데 [3]에서는 한 장비가 통신을 할때 잠금을 하지 않더라도 문제가 생길 확률이 낮고 오히려 갱신이 효율적으로 이루어 질 수 있다는 점을 보여주었다.

3.2 동기적 동기화

동기적 동기화는 각 스텝마다 각 모델들이 학습한 그래디언트를 합쳐서 모델을 업데이트하기 때문에 각 모델들의 파라미터들은 항상 같게 유지된다. 하지만 모든 장비의 그래디언트 계산이 끝나야만 파라미터를 업데이트 할 수 있어서 가장 느린 장비의 학습 속도에 맞춰진다는 단점이 있다.

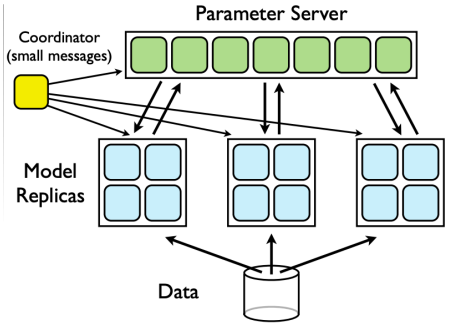

그림 3: Sandblaster L-BFGS [1]

[1]에서는 동기적 동기화 방식으로 Sandblaster L-BFGS 방식을 제안하였다. Sandblaster L-BFGS 방식은 파라미터 업데이트를 위하여 SGD 가 아닌 L-BFGS 방식을 활용하는데 Coordinator 가 메시지를 통하여 파라미터 서버와 모델 복사본들을 관리하여 동시에 학습하기 때문에 각 모델들의 파라미터들이 같게 유지된다.

3.3 혼합

동기적 동기화 방식에 비하여 비동기적 동기화 방식은 각자의 속도 대로 즉시 업데이트 하고 한 장비의 느린 학습 속도가 전체의 학습 속도를 늦추는 문제가 없기 때문에 동기화 방식에 비하여 빠르다.

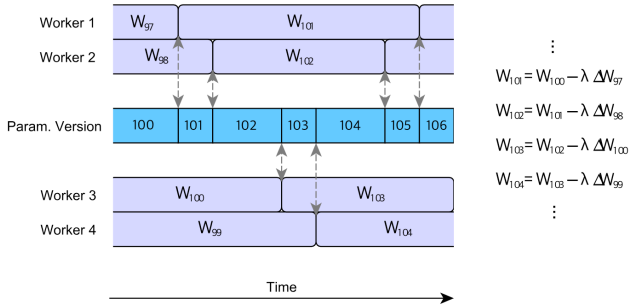

그림 4: 지연된 그래디언트 문제 [5]

하지만 비동기적 동기화 방식에는 동기적 동기화 방식이 가지지 않은 치명적인 문제점이 있는데 이는 지연된 그래디언트 문제(stale gradient problem)이다. 지연된 그래디언트란 비동기적 동기화 방식에서 각 모델이 그 모델의 파라미터의 그래디언트를 구하는 동안 다른 모델들이 그 모델의 파라미터를 업데이트 하기 때문에 파라미터를 업데이트를 할 시점이 오게 되면 처음과는 다른 파라미터를 업데이트하게 되는 문제점을 말한다. 그렇기 때문에 최적화가 효율적으로 일어나지 않고 그렇기 때문에 데이터를 활용하는 효율이 낮아지게 된다.

$$g_i = \frac{1}{\lambda}\sum_{l=1}^{\lambda}\Delta\theta_l$$
$$\theta_{i+1} = \theta_i - \alpha g_i.$$

SSGD [4]

$$c = \lfloor(\lambda/n)\rfloor$$
$$g_i = \frac{1}{c}\sum_{l=1}^{c}\alpha(\tau_{i,l})\Delta\theta_l,\ l \in \{1, 2, \ldots, \lambda\}$$
$$\theta_{i+1} = \theta_i - g_i,$$

ASGD [4]

이 문제를 해결하기 위하여 여러 방법이 제안되었다. [4]에서는 동기적 동기화 방식과 비동기적 동기화 방식을 절충하기 위하여 완화된 SGD 방법인 Asynchronous SGD 방법을 제안하였다. ASGD 란 비동기적 동기화 방식에서 각 모델이 학습하는 즉시 업데이트 하는 것이 아니라 k 개의 모델이 학습하기 까지를 기다려서 업데이트하는 것이다.

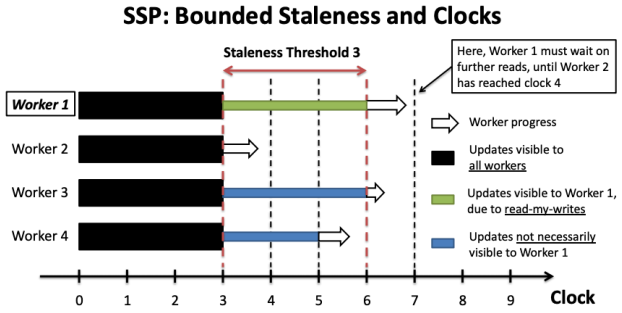

그림 5: SSP [5]

[5]는 모델간의 지연성(Staleness)를 제한할 파라미터 서버 구조(Stale Synchronous Parallel: SSP)를 제안하였다. SSP 에서는 특정 threshold 가 지나면 가장 느린 학습자를 기다리도록 강제되어 모델들 간의 지연성을 일정 수준 이하로 유지하도록 만든다.

4 분산학습 구조

동기화 방식과는 별개로 분산 학습 구조는 실행 측면에서 분산 학습에 중요한 영향을 미친다. 분산 학습 구조는 따라 노드들의 역할이나 노드간의 커뮤니케이션 방식이 정한다.

4.1 파라미터 서버 구조

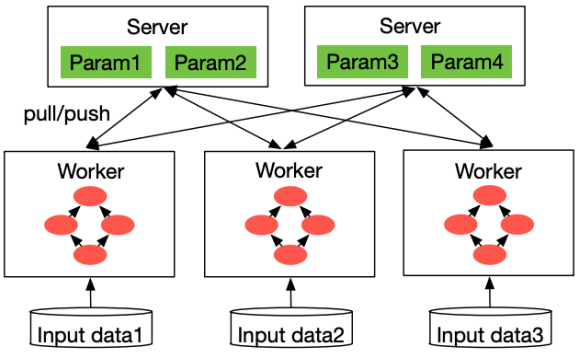

그림 6: 파라미터 서버 구조 [7]

파라미터 서버 방식은 노드들을 파라미터들을 보관만 하는 파라미터 서버와 분산 학습을 하는 노드로 나눈다. 서버 노드들은 모델의 파라미터들을 분할하여 각 분할만 보관한다. 그리고 분산 학습을 하는 노드는 파라미터들을 보관한 서버와만 통신을 한다. [1]에서의 분산 방법들은 모두 이 파라미터 서버 방식을 전제로 하고 있다.

4.2 집단 통신 구조

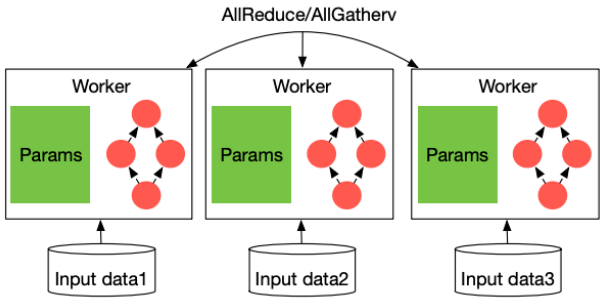

그림 7: 집단 통신 구조 [7]

집단 통신 구조(All-reduce/All-gather 방식)은 서버를 따로 두지 않고 각 워커들이 다른 데이터들에 대하여 각각 학습한 후 워커들 간의 집단 통신을 통하여 동기화 하는 방식이다. 주로 Message Passing Interface(MPI)에 기반하여 구현되는데 실제로 All-reduce/All-gather 와 같은 오퍼레이션이 사용되는 것이 아니라 Unidirectional Ring 과 같은 더 효율적인 방식을 통하여 동기화한다. 모든 노드가 다른 모든 노드들에게 그래디언트를 보내야 하기 때문에 커뮤니케이션이 매우 늘어날 수 도 있다.

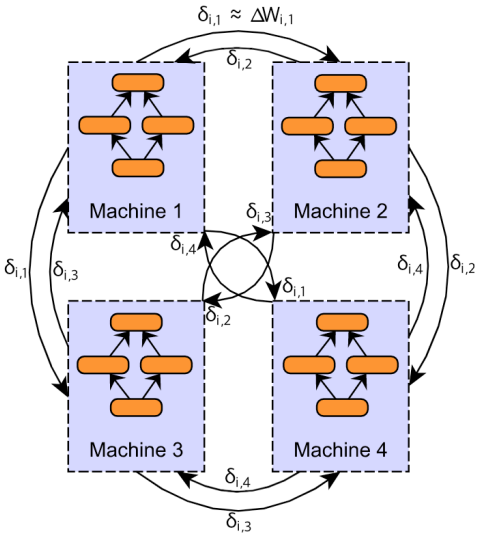

그림 8: [6]

[6]에서는 커뮤니케이션을 극도로 효율화 함으로써 이 집단 통신 구조를 약점을 보완하였다. 우선 분산 학습에서 각

모델의 그래디언트의 빈도가 드문점에서 착안하여 모든 그래디언트의 값을 통신하는 대신 0 이 아닌 그래디언트의 인덱스 값만을 통신하게 된다. 또한 그 그래디언트들의 값들을 단일 비트로 계층화(Quantize)하여 통신하는 각 그래디언트의 값들이 단일화 되도록 하였다. 마지막으로 엔트로피 코딩을 통하여 인덱스 값들 조차 압축함으로써 커뮤니케이션을 효율화하였다.

4.3 혼합

[7]에서는 이러한 파라미터 서버 구조의 통신 방식과 집단 통신 구조의 통신 방식의 성능을 측정해 보았는데 특이한 발견을 제보하였다. 파라미터 업데이트의 분포가 밀집한 모델의 경우에는 집단 통신 구조가 효과적이었고 분포가 희박한 경우에는 파라미터 서버 구조가 효과적이었다. 그리하여 [7]에서는 파라미터의 분포에 따라서 구조를 선택하는 혼합 구조를 제안하였다.

5 강화 학습 분산

기존 딥러닝 모델들은 학습 속도를 가속화 하기 위하여 의도적으로 데이터를 분산하여 저장한 뒤에 분산 학습을 한다. 반면 강화 학습 환경에서는 데이터를 시뮬레이터를 통하여 얻는다. 강화 학습은 환경에서 얻은 경험을 통하여 모델을 학습하고 학습한 모델이 환경에 영향을 주어 다른 경험을 얻기 때문에 순간 순간 새로운 데이터라고 할 수 있다. 그렇기 때문에 여러 시뮬레이터를 통하여 더 빠르게 많은 데이터를 모으는 것이 강화 학습 훈련의 관건이다. 그렇기 때문에 강화 학습 환경에서는 분산을 적극적으로 활용하려 하였다.

5.1 Gorila Framework

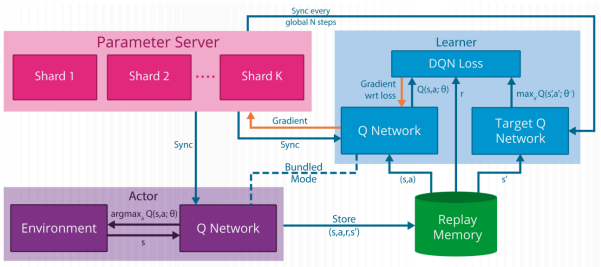

그림 8: Gorila Framework [8]

[8]은 기존의 딥러닝 분산 구조를 강화 학습에 적용하여 대규모 분산 구조를 제안하였다. 이 General Reinforcement Learning Architecture(Gorila)는 파라미터 서버와 행동자, 학습자를 나눈다. 행동자들은 각자 가지고 있는 모델로 시뮬레이터를 수행하며 데이터를 수집하여 재생 기억이라는 저장소에 저장한다. 학습자들은 이에 저장된 데이터를 분할하여 사용하며 파라미터 서버들과 통신하며 분산학습을 한다. 재생 기억 저장소의 데이터를 학습하기 때문에 동기적, 비동기적 동기화가 모두 가능하다. 이를 통하여 큰 강화 학습 모델을 빠르게 학습시키는데 성공하였다.

5.2 APE-X Framework

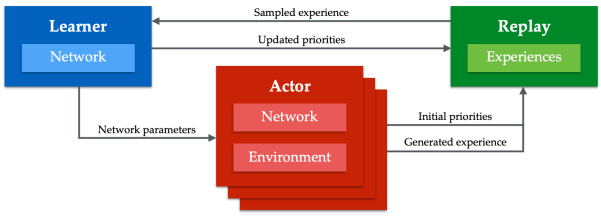

그림 9: APE-X Framework [9]

강화 학습 환경에서는 데이터의 경험을 여러 번 재사용하기 위하여 데이터들을 재생 기억이라는 저장소에 넣어두고 이를 무작위 추출하여 학습하는 방식을 취한다. 하지만 점차 효율적인 학습을 위하여 단순 표본 추출이 아닌 중요도 기반 표본 추출이 제안되었다. [9]에서는 분산 학습과 데이터의 중요도 기반 추출 방식을 모두 활용하여 시간 대비 최고의 성과를 보여주었다. 여러 행동자들은 지속적으로 활동하며 비동기적으로 데이터를 경험 재생에 넣는다. 반면 하나의 학습자는 경험 재생에 쌓인 데이터를 주기적으로 추출하여 학습한다. 이 때 단순 무작위 추출이 아닌 우선순위를 기반하여 추출하여 학습하기 때문에 효율적인 학습이 가능하다. 이러한 구조가 효율적인 이유는 학습자는 하나의 GPU 가 필요하지만 행동자들은 CPU 만 필요로 하기 때문에 여러 행동자가 활동하는 것이 가능하기 때문이다.

6 결론

본 논문에서는 기존의 분산 처리 방법과는 다른 딥러닝 분산 처리에서의 주요 개념과 그에 관련된 연구들을 다루었다. 딥러닝의 학습은 단순히 데이터를 처리하는 것이 아니라 데이터의 처리를 통하여 모델의 파라미터를 업데이트 하는 것이 목적이기 때문에 기존의 분산 처리에서 다루지 않던 개념들이 등장한다. 분산을 단순히 큰 모델의 파라미터들을 나누어 보관하기 위해 활용하기도 하고 데이터의 분산 처리를 위하여 활용하기도 한다. 데이터 분산의 효과는 모델의 동기화 방법과 분산 구조에 따라 크게 달라질 수 있다. 그렇기 때문에 상황에 맞는 동기화 방식과 분산 구조를 선택하는 것이 효과적인 분산 처리를 위해서 필수적이다. 그리고 현재 딥러닝 분산이 적극적으로 활용되고 있는 강화학습에서의 분산 구조를 다루었다. 딥러닝에서의 분산 처리는 기존의 대규모 데이터 분산 처리만큼 혁신적인 속도와 성능향상을 가져오고 있진 못하기 때문에 많은 연구의 기회가 남아있는 분야이다.

REFERENCES

[1] Dean, Jeffrey, et al. "Large scale distributed deep networks." Advances in neural information processing systems. 2012.

[2] Krizhevsky, Alex. "One weird trick for parallelizing convolutional neural networks." arXiv preprint arXiv:1404.5997 (2014).

[3] Recht, Benjamin, et al. "Hogwild: A lock-free approach to parallelizing stochastic gradient descent." Advances in neural information processing systems. 2011.

[4] Wei Zhang, Suyog Gupta, Xiangru Lian, and Ji Liu. Staleness-aware async-sgd for distributed deep learning. IJCAI, 2016.

[5] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B. Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. More effective distributed ml via a stale synchronous parallel parameter server. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 26, pages 1223–1231. Curran Associates, Inc., 2013.

[6] Nikko Strom. Scalable distributed dnn training using commodity gpu cloud computing. In Sixteenth Annual Conference of the International Speech Communication Association, 2015.

[7] Kim, Soojeong, et al. "Parallax: Automatic Data-Parallel Training of Deep Neural Networks." arXiv preprint arXiv:1808.02621 (2018).

[8] Nair, Arun, et al. "Massively parallel methods for deep reinforcement learning." arXiv preprint arXiv:1507.04296 (2015).

[9] Horgan, Dan, et al. "Distributed prioritized experience replay." arXiv preprint arXiv:1803.00933 (2018).

[10] Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." Communications of the ACM 51.1 (2008): 107-113.

[11] Zaharia, Matei, et al. "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing." Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012.