논문 21-46-10-08 The Journal of Korean Institute of Communications and Information Sciences '21-10 Vol.46 No.10
https://doi.org/10.7840/kics.2021.46.10.1604

병렬 딥러닝 기법에 관한 연구 동향 분석

윤 진 이•, 이 지 호*, 한 나 영**, 이 형 준°

A Survey on Parallel Deep Learning

JinYi Yoon•, JiHo Lee*, Nayoung Han**, HyungJune Lee°

요 약

딥러닝(Deep Learning)은 최근 다양한 분야에서 폭넓게 사용되고 있으며, 특히 자연어 처리, 음성 인식, 이미지 분류, 특징 추출 및 기계 번역과 같은 최신 기술에서의 급격한 발전을 가져왔다. 막대한 양의 데이터와 복잡한 태스크(Task)를 처리하기 위해 신경망(Neural Networks)이 커짐에 따라, 신경망의 계층(Layer)과 파라미터(Parameter)가 많아지면서 계산 집약적인 기술들이 가능해지도록 하였다. 이러한 큰 규모의 심층신경망(Deep Neural Networks, DNNs)이 한정적인 자원의 기기들에서 동작할 수 있을 뿐만 아니라 러닝(Learning)을 가속하기 위해 연합 학습(Federated Learning)이라는 병렬 처리(Parallelization) 기법이 등장하게 되었다. 본 논문에서는 데이터 병렬 처리(Data Parallelism), 모델 병렬 처리(Model Parallelism), 하이브리드 병렬 처리(Hybrid Parallelism), 그리고 파이프라인 병렬 처리(Pipeline Parallelism)의 네 가지 병렬 처리 방법을 소개하고자 한다.

Key Words : Deep Learning, Parallel Deep Learning, Federated Learning, Data Parallelism, Model Parallelism, Hybrid Parallelism, Pipeline Parallelism

ABSTRACT

Deep learning has been widely used in various fields, especially leading drastic development in the state-of-the-art technologies such as natural language processing, speech recognition, image classification, feature extraction, or machine translation. As the massive data and intricate tasks necessitate enlarged neural networks, the number of layers and parameters in neural networks become tremendous, resulting in great performance of compute-intensive technologies. To make large-scale deep neural networks(DNNs) scalable over resource-constrained devices and accelerate learning, some parallelization approaches have investigated under the name of federated learning. In this survey, we introduce four parallelism methods: data parallelism, model parallelism, hybrid parallelism, and pipeline parallelism.

Ⅰ. 서 론

최근 데이터양의 막대한 증가와 컴퓨터 처리 속도의 발전에 따라, 인공 지능(Artificial Intelligence, AI) 분야에서 딥러닝(Deep Learning)이 강력한 컴퓨팅 기술로 대두되었다. 사람의 사고방식을 사용해야지만 가

※ 이 성과는 2021년도 정부(과학기술정보통신부)의 재원으로 한국연구재단(No. NRF-2021R1A2B5B01002906)과 2019학년도 이화여자대학교 대학원 장학금 지원을 받아 수행된 연구임.

• First Author : Department of Computer Science and Engineering, Ewha Womans University, yjin3012@ewhain.net, 학생회원
° Corresponding Author: Department of Computer Science and Engineering, Ewha Womans University, hyungjune.lee@ewha.ac.kr, 정회원
* Department of Computer Science, University of Virginia, jiholee@virginia.edu
** Department of Mathematics, Ewha Womans University, hny0309@ewhain.net
논문번호 : 202107-167-B-RU, Received July 15, 2021; Revised August 10, 2021; Accepted August 10, 2021

능하다고 여겨졌던 자연어 처리(Natural Language Processing)[1], 음성 인식(Speech Recognition)[2], 이미지 분류(Image Classification)[3], 특징 추출(Feature Extraction)[4], 또는 기계 번역(Machine Translation)[5]와 같은 태스크(Task)를 컴퓨터를 통해 처리할 수 있게 되면서 다양한 분야에서 딥러닝의 급격한 발전이 이루어졌다. 기존의 기계 학습(Machine Learning)은 데이터의 부족과 한정적인 목적으로만 사용되었던 것에 비해, 더 깊고 복잡한 심층신경망(Deep Neural Networks, DNNs)을 통해 딥러닝은 방대한 양의 데이터로부터 스스로 의미 있는 정보를 추출할 수 있다.

딥러닝 구조는 뉴런(Neuron)으로 연결되어있는 신경망(Neural Networks)을 어떻게 형성하고 학습시키는지에 달려있다. 복잡한 데이터를 사용하거나 정확도를 향상시키기 위해 딥러닝 모델의 규모가 커지고 깊어지면서, 모델을 형성하는 신경망의 계층(Layer)과 파라미터(Parameter)의 수가 더 많아졌다. 이로 인해, 딥러닝을 하나의 디바이스(Device)에서 학습시키기에는 자원 부족, 복잡한 계산으로 인해 예기치 않게 긴 학습 시간, 심지어는 학습 실패 등의 문제를 겪게 되었다. 딥러닝에서의 이러한 문제를 해결하기 위해, 여러 컴퓨팅 자원들이 병렬로 처리할 수 있는 연합 학습(Federated Learning)에 관한 연구가 제안되고 있다[6]. 이처럼 학습 과정이나 추론 과정, 또는 모델을 분할하여 여러 대의 디바이스에서 학습시킴으로써, 확장성(Scalability)이나 학습 속도의 향상을 기대할 수 있다.

Ⅱ. 연합 학습의 개요

여러 컴퓨팅 자원 상에서 협력 학습을 진행하는 방법은 크게 학습 데이터를 나누는 데이터 병렬 처리 기법(Data Parallelism)과 신경망 모델을 분할하는 모델 병렬 처리 기법(Model Parallelism)으로 분류할 수 있다[7]. 데이터 병렬 처리 기법에서는 그림 1(a)에서와 같이 동일한 신경망을 여러 디바이스에 복제하고, 전체 학습 데이터를 나누어 각각의 디바이스는 분할된 데이터에 대한 학습을 진행한다[8]. 그림 1(b)에서의 모델 병렬 처리 기법은 학습 모델을 분할하여 디바이스가 모델의 일부분을 가지고 학습을 진행하는 방법이다[9]. 최근에는 그림 1(c)과 같이 데이터 병렬 처리 기법과 모델 병렬 처리 기법을 혼합한 하이브리드 병렬 처리 기법(Hybrid Parallelism)이 제안되었다.

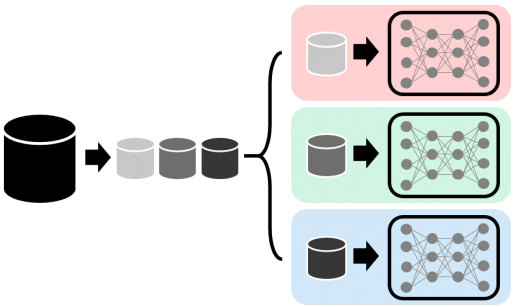
하지만 이러한 기존의 방식들은 여전히 지나치게 발생하는 통신 오버헤드(Overhead) 또는 자원 저활용(Under-utilization)의 문제를 겪고 있다. 이를 해결하기 위해 여러 디바이스가 동시에 일을 처리하는 파이프라인 병렬 처리 기법(Pipeline Parallelism)을 통해 빠른 학습이 가능하여지도록 하였다[10]. 기존에는 학습 데이터를 순차적으로 주입하여 이전 데이터의 역전파(Backpropagation)로 모델이 학습된 후, 다음 학습 데이터를 사용하여 전방 전달(Forward Pass)이 이

표 1. 연합 학습의 종류
Table 1. Types of federated learning

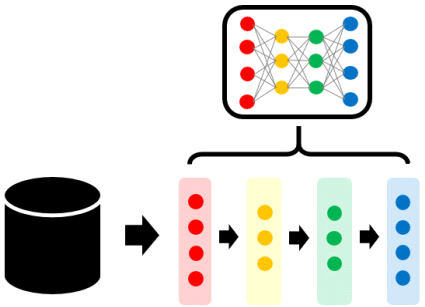
Divide Training Data	Divide Neural Network	Pipeline Scheduling
Data Parallelism	Model Parallelism	-
Hybrid Parallelism		-
-	Pipeline Parallelism	

표 2. 주요 번역된 용어
Table 2. Translated terminology

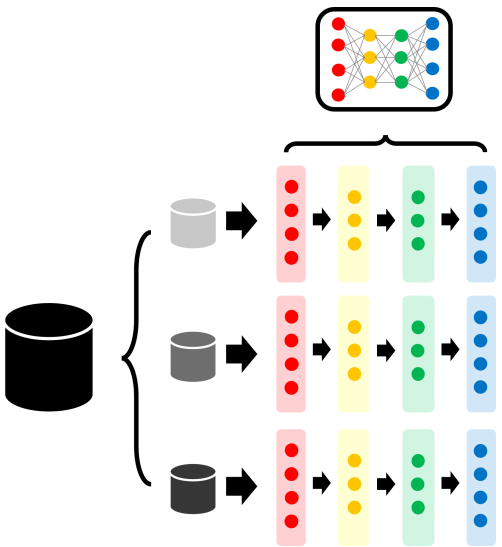
한글 표기	영어 표기
인공 지능	Artificial Intelligence, AI
기계 학습	Machine Learning
심층신경망	Deep Neural Networks, DNNs
신경망	Neural Networks
연합 학습	Federated Learning
데이터 병렬 처리 기법	Data Parallelism
모델 병렬 처리 기법	Model Parallelism
하이브리드 병렬 처리 기법	Hybrid Parallelism
파이프라인 병렬 처리 기법	Pipeline Parallelism
저활용	Under-utilization
역전파	Backpropagation
전방 전달	Forward Pass
후방 전달	Backward Pass
손실	Loss
경사도	Gradient
경사하강법	Gradient Descent
합성곱 신경망	Convolutional Neural Networks, CNNs
합성곱층	Convolutional Layer
완전 연결 계층	Fully Connected Layer, Dense Layer
정류 선형 유닛	Rectified Linear Unit, ReLU
과적합	Over-fitting
최적화기	Optimizer
가중치	Weight
스테일니스	Staleness



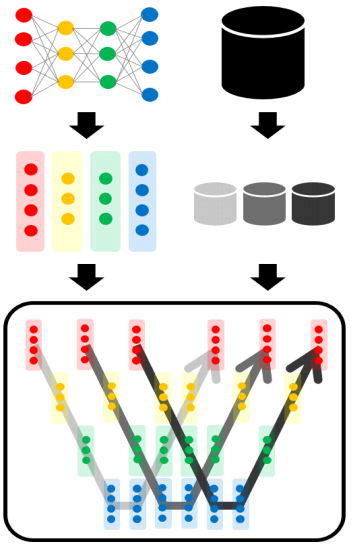
(a) 데이터 병렬 처리 기법에 대한 개요
(a) An overview of data parallelism



(b) 모델 병렬 처리 기법에 대한 개요
(b) An overview of model parallelism



(c) 하이브리드 병렬 처리 기법에 대한 개요
(c) An overview of hybrid parallelism



(d) 파이프라인 병렬 처리 기법에 대한 개요
(d) An overview of pipeline parallelism

그림 1. 각 병렬 처리 기법에 대한 개요
Fig. 1. An overview of parallelism

루어졌지만, 파이프라인 병렬 처리 기법에서는 그림 1(d)에서 보이는 것과 같이 여러 대의 디바이스가 신경망 일부를 맡고, 이전 데이터의 후방 전달(Backward Pass)이 끝나기 전에 다음 데이터의 학습을 시작하여, 여러 개의 배치(Batch) 데이터가 동시에 학습되어 디바이스의 활용을 높일 수 있도록 한다.

본 논문에서는 병렬 처리 관점에서 관련 연구들을 소개하고자 한다. 표 1에서는 연합 학습을 데이터를 분할하는 데이터 병렬 처리 기법, 모델을 분할하는 모델 병렬 처리 기법, 그 두 기법을 모두 포함하는 하이브리드 병렬 처리 기법, 그리고 여러 태스크를 동시에 처리하기 위해 모델 분할을 내재한 파이프라인 병렬 처리 기법으로 분류하였다. 일부에서는 파이프라인 병렬 처리를 위해 여러 배치 데이터가 동시에 학습이 진행되므로, 데이터 병렬 처리 기법을 포함한다고 정의하기도 한다.

관련 연구들을 소개하기에 앞서, 먼저 표 2에서 본 논문에서 번역되어 사용하는 용어들을 정리하고자 한다.

본 논문의 구성은 다음과 같다. III장에서는 데이터 병렬 처리 기법을 소개하고, IV장에서는 모델 병렬 처리 기법, V장에서는 하이브리드 병렬 처리 기법, VI장에서는 파이프라인 병렬 처리 기법에 대해 소개한다. 마지막으로 VII장에서 각 병렬 처리 기법 간의 주요 특성에 대한 비교를 제시하고 결론으로 본 논문을 마치고자 한다.

Ⅲ. 데이터 병렬 처리 기법

병렬식 데이터 처리에서는 그림 1(a)에서와 같이 학습시킬 데이터를 나누어, 각 디바이스가 다른 부분 데이터를 같은 학습 모델을 이용하여 학습하도록 한다. 위 방식은 심층신경망에 대한 부가적인 이해가 없이도 쉽게 적용할 수 있고, 신경망 전체를 복사해 사용하므로 어떠한 심층신경망에도 적용 가능한 방법이라는 장점이 있다. 또한, 다수의 데이터 학습을 여러 디바이스에서 동시에 진행할 수 있으므로 신경망 학습 속도를 높일 수 있다.

기계 학습에서는 손실 함수(Loss Function)의 값을 최소화하는 것은 중요한 문제이고, 이를 위해 데이터 병렬 처리에서는 데이터 일부를 이용하여 손실 함수의 경사도(Gradient)를 구하는 방법인 확률적 경사하강법(Stochastic Gradient Descent, SGD)[11]을 사용하였다. 하지만 기계 학습에 필요한 데이터의 양이 급격하게 증가하는 반면, 이를 처리하기 위한 저장 용량이나 네트워크 대역폭(Bandwidth)은 여전히 부족한 상황이다. 따라서 확률적 경사하강법보다 더 효율적인 분산형 데이터 분석 알고리즘이 필요하다고 여겨, *PSGD*[12]에서는 병렬식 확률적 경사하강법(Parallel Stochastic Gradient Descent)을 제안하였다. *PSGD*는 분산형 확률적 경사하강법 중 하나로, 각 프로세서(Processor)가 수행한 확률적 경사하강법의 결과를 마스터 루틴(Master Routine)에서 취합하여 평균을 구하는 방식이다. *PSGD*는 병렬 방식의 기계 학습에 유리하며, 입출력(Input/Output, I/O) 응답속도와는 독립적인 알고리즘이므로, 특히, 맵리듀스(MapReduce) 구현에 적합하다. 이로 인해 장애 허용(Fault Tolerance)을 보장하고 용이한 프로그래밍(Programming)이 가능해졌다. 그러나 *PSGD*는 여전히 장애 허용을 위한 오버헤드(Overhead)가 전체 스루풋(Throughput)을 저하시킬 수 있을 뿐만 아니라, 반복적인 연산에 적합하지 않아 숫자 집약적인 데이터 분석에는 적절하지 않다는 맵리듀스의 내재된 한계점들을 극복하지 못한 알고리즘이라고 할 수 있다. 또한, *PSGD*에서는 모든 프로세서가 작업을 끝내야만 다음 에포크(Epoch)를 수행할 수 있으므로 메모리 록킹(Memory Locking) 문제가 발생할 수 있다. 따라서 효율적인 메모리 사용이나 동기화 작업이 어렵다.

*PSGD*의 문제점을 해결하기 위하여 *Hogwild!*[13]라는 새로운 방식이 고안되었다. *Hogwild!*는 각 프로세서가 동시에 메모리에 접근하는 것을 허용하여 효율적인 메모리 사용 및 원활한 동기화가 가능하여지도록 하였다. 각 스레드(Thread)는 학습 데이터로부터 학습할 정보를 불러온 후, 경사도를 계산하여 파라미터를 업데이트한다. 이때, 최신 상태가 아직 반영되지 않은 솔루션을 이용하여 경사도를 계산하므로, 각 스레드가 동시에 업데이트를 진행할 수 있다. 즉, *Hogwild!*는 *PSGD*에서 발생하는 메모리 록킹 문제를 해결할 수 있는 방법이며, 데이터가 희소(Sparse)한 경우에는 위 방식을 통해 거의 최적에 가까운 솔루션을 얻을 수 있다는 것을 보였다. 반면, 데이터가 희소하지 않은 경우에는 적절하지 않은 알고리즘이다.

MALT[14]는 병렬식 데이터 처리의 효율성 개선을 위해 개발된 새로운 라이브러리이다. *MALT*는 그림 2와 같이 기존에 있는 여러 개의 기계 학습 소프트웨어(Software)들을 통합하여 병렬식 데이터 처리를 가능하게 하였다. 또한, 기계 학습에서의 효율적인 메모리 공유를 위해 개발자들이 데이터의 흐름을 관찰할 수 있는 기능을 지원하고, 간단하며 유연한 API(Application Programming Interface)를 제공한다. *MALT* 라이브러리 인터페이스는 단방향 RDMA(Remote Direct Memory Access)의 직접 메모리 액세스 특징을 이용하여 네트워크 프로세싱(Processing) 비용과 통신(Transmission) 비용을 감소시켰다. 그러나 *MALT*에서의 스칼라(Scalar)값 오류 및 비잔틴 실패(Byzantine Failures)가 발생했을 때 이를 해결하기가 어렵다는 한계점이 있다.

마지막으로, 데이터 병렬 처리의 활용에 대하여 살펴보고자 한다. *ACO-DP*[15]에서는 외판원 문제(Traveling Salesman Problem, TSP)에 데이터 병렬 처리 기법을 활용하였다. 기존의 개미 집단 최적화(Ant Colony Optimization, ACO)는 외판원 문제의 해결에 일반적으로 사용되었는데, 개체군을 기반으로 한 탐색 방법으로 경로 설정(Tour Construction)과 페로몬 업데이트의 두 단계로 나뉜다. 하지만 개미 집단 최적화에서의 병렬식 태스크 처리의 접근 방법은 메모리 접근 패턴의 불예측성이나 워프 다이버전스

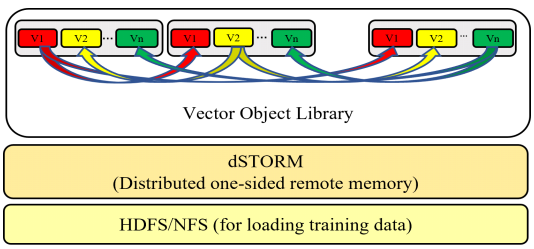


그림 2. *MALT* 구조[14]
Fig. 2. The architecture of *MALT*[14]

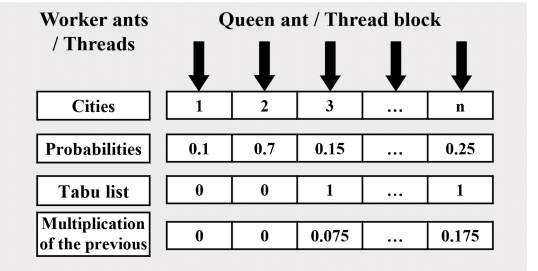


그림 3. 경로 설정 단계에서의 병렬식 데이터 처리[15]
Fig. 3. Data parallelism approach on the tour construction[15]

(Warp Divergence) 등의 문제가 있었다. 이를 보완하기 위해, *ACO-DP*에서는 최초로 이를 GPU에서의 병렬식 데이터 처리를 활용하여 해결하고자 하였으며, 특히 경로 설정 단계의 효율성을 높이고자 하였다. 그림 3에서와 같이 여왕개미(Queen Ant)와 일개미(Worker Ant) 두 종류의 개미가 있고, 각 스레드 블록은 여왕개미에 대응한다. 각 스레드 블록에서는 일개미들이 경로 설정을 위한 정보를 수집하고 최종 솔루션을 더 빨리 찾을 수 있도록 돕는다. 결과적으로, 실험을 통해 스레드 하나의 CPU를 사용하였을 경우보다 더 효율적임을 확인하였다.

그러나 대규모 학습 모델에서 파라미터 수가 많을 경우, 파라미터를 동기화하는 과정에서 큰 오버헤드가 발생할 수 있기 때문에 데이터 병렬 처리는 많은 파라미터를 다루기가 어렵다. 또한, 자원이 한정된 디바이스에서는 큰 규모의 모델을 학습할 수 없다는 한계점이 있다[16, 17]. 표 3에서 데이터 병렬 처리 기법에 관한 연구를 요약하여 정리하였다.

Ⅳ. 모델 병렬 처리 기법

모델 병렬 처리 기법은 그림 1(b)에서와 같이 주로 확장성(Scalability)을 위하여 신경망을 독립된 디바이스들에게 여러 개의 부분 모델로 분할하여 할당해, 동일한 학습 데이터 배치들을 사용해 순차적으로 학습을 진행한다. 모델 병렬 처리 기법의 분산 딥러닝은 메모리가 제한적인 엣지(Edge) 디바이스 환경에서도 확장이 가능하여 학습, 추론이 가능한 방향을 제시한다. 딥러닝 모델의 연산을 여러 디바이스들에게 나누어 진행함으로써 엣지 디바이스의 제한된 저장공간의 특성을 극복할 뿐만 아니라 연산 오버헤드(Overhead) 또한 감소시킬 수 있다.

모델 병렬 처리에서는 복잡하고 방대한 양의 데이터를 분류하기 위해 대규모의 모델을 가속화하여 학습하기 위해 *ImageNet*[18]과 같은 다수의 CPU, GPU에 모델을 분배해 병렬로 학습을 진행하는 방향으로 제시되었다. 이러한 학습 방식은 *Megatron-LM*[22], *GNMT*[23]과 같은 자연어 처리, 신경망 기계 번역 등 다양한 분야에서 적용될 수 있다. 그런데, 모델 병렬 처리에서는 모델의 파라미터들이 다른 기기들에게 분배되어 있고 주로 효율적인 병렬 처리를 위해 비동기적인 방식으로 파라미터를 업데이트되게 된다. 이때 여러 디바이스 상에서 병렬로 처리함에 따라, 최신 파라미터가 아닌 오래된 파라미터를 쓰게 되는 파라미터의 스테일니스(Staleness) 문제나 학습 속도를 결정하는 중요 파라미터들을 판별하여 업데이트하는 문제들을 해결하기 위한 방향들이 *STRADS*[24]에서 제시되었다.

사물 인식은 기계 학습의 주요 과제이며, 정확도를 높이기 위해 더 많은 양의 데이터를 학습시키는 것은 불가피한 문제이다. *ImageNet*[18]은 22,000개의 카테고리가 넘는 고해상도의 이미지로 구성된 데이터셋으로, 이를 학습하기 위해 GPU를 사용하여 더 큰 규모의 합성곱 신경망(Convolutional Neural Networks, CNNs)을 학습시키기 위한 새로운 합성곱 신경망 구조인 *AlexNet*[19]를 제안하였다. 이는 합성곱 신경망을 이용한 데이터 분류 기법의 가능성을 제시하였다는

표 3. 데이터 병렬 처리 기법에 대한 요약
Table 3. A summary of data parallelism

Paper	Problem	Methodology
PSGD[12]	To find distributive data analysis algorithms	Each processor runs SGD on local copy of parameters and then aggregate the solution by a master-routine
Hogwild![13]	To solve the performance destroying from memory locking and synchronization	Processors are can access to shared memory and update individual components of memory at the same time
MALT[14]	To enable long running machine learning tasks of existing frameworks	Provide data-parallel simple machine learning framework which has flexible API
ACO-DP[15]	To apply GPU implementation of ACO(Ant Colony Optimization) for TSP	Data parallelism approach on tour construction stage

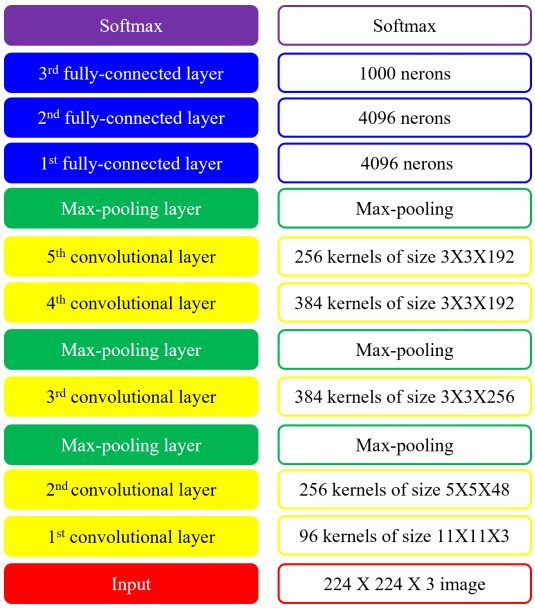


그림 4. *AlexNet*의 구조[19]
Fig. 4. The architecture of *AlexNet*[19]

점에서 큰 의의가 있다. *AlexNet*은 그림 4에서와 같이 총 8개의 계층으로 구성되어 있으며, 5개의 합성곱층(Convolutional Layer)과 3개의 완전 연결 계층(Fully Connected Layer, Dense Layer)으로 이루어져 있으며, 2개의 GPU에서 나뉘어 병렬로 처리된다. 모든 계층에서 정류 선형 유닛(Rectified Linear Unit, ReLU) 함수를 사용하여 학습을 가속하였고, 특히 첫 번째와 두 번째 계층에서는 LRN(Local Response Normalization)을 사용하여 에러율을 줄이는 구조이다. 또한, 과적합(Over-fitting)을 방지하기 위해 오버래핑 풀링(Overlapping Pooling), 데이터 증강(Augmentation), 드롭아웃(Dropout)을 이용하였다. 그러나, *AlexNet*은 *VGG*[20]나 *ResNet*[21]과 비교하였을 때, 시간이 더 오래 걸리는 모델이라는 한계가 있다.

Megatron-LM[22]는 자연어 처리 분야에서 초대형 모델을 학습시키기 위해 간단하면서도 효과적인 접근 방식을 제안했다. 병렬 방식의 효율성을 최대화하고 GPU 간의 통신을 줄이기 위해 트랜스포머(transformer) 네트워크를 사용한다. 트랜스포머 네트워크는 그림 5에서 보이는 것처럼 크게 다층 퍼셉트론(Multi-Layer Perceptron, MLP) 블록과 셀프 어텐션(Self-Attention) 블록으로 구성된다. 다층 퍼셉트론 블록은 결과 출력 간의 동기화 지점을 제거하여 통신을 요구하지 않고 직접 출력을 가져온다. 셀프 어텐션 블록은 입력의 행렬을 키(Key), 쿼리(Query), 값

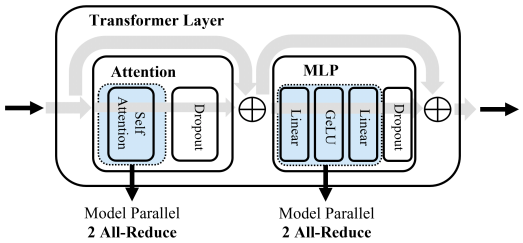


그림 5. 트랜스포머 계층의 도식적인 구조와 통신 오퍼레이션[22]
Fig. 5. The schematic architecture and communication operation of transformer layers[22]

(Value)으로 분할하여 각 GPU에서 지역적으로(Locally) 처리하는 구조로 하는 병렬 방식이다. 이 단순화된 두 개의 블록 접근 방식을 통해 모델 병렬식 처리를 하므로 그림 5에서 보이는 것과 같이 총 네 개의 통신 오퍼레이션(Operation)이 있는 전파 및 역전파 전달 경로에서 두 개의 축소(All-Reduce)된 오퍼레이션만을 사용하는 것을 가능하게 한다. 또한, *Megatron-LM*은 통신 오버헤드를 줄일 뿐만 아니라 GPU가 처리한 결과의 복사본을 사용해 파라미터를 정규화하므로, 추가 오류를 방지하며 대규모 모델을 학습할 수 있다. 이 논문의 가장 큰 의의는 자연어 처리 영역에서 수십억 개의 파라미터가 있는 모델을 간단한 방법으로 학습할 수 있고, 새로운 컴파일러나 라이브러리 없이도 쉽게 구현이 가능하다는 점이다. 그러나 아직은 최적화기(Optimizer)의 효율성과 메모리 풋 프린트(Footprint)를 개선할 필요가 있다는 한계가 있다.

신경망 기계 번역(Neural Machine Translation, NMT)은 학습 모델이 번역의 정확도를 높이는 것을 목적으로 한다. 전통적으로 인코더(Encoder)-디코더(Decoder) 방식이 사용되었으나, 인코더는 신경망 네트워크가 소스(Source) 문장으로부터 모든 정보를 고정된 길이의 벡터로 압축해 병목(Bottleneck) 현상을 야기할 수 있다. *GNMT*[23]은 이러한 문제를 해결하기 위해 확장된 버전의 인코드-디코드 접근 방식을 제안한다. 해당 프레임워크(Framework)에서는 순환 신경망(Recurrent Neural Networks, RNNs)이 인코더와 디코더에 사용되며, 소스 입력에서 관련된 특징(Feature)들을 추출하여 학습과 함께 정렬 및 번역을 진행하므로, 긴 문장의 성능이 향상된다. 그러나 모델의 파라미터들을 다수의 머신들에 나누는 관점보다는 순환 신경망을 통해 문제를 해결하는 점에 집중했다는 점에서 확장성 면에서의 한계가 있다.

한편 *STRADS*[24]는 모델의 파라미터를 스케줄링하

는 프로그래밍 가능한 모델 병렬 처리 시스템이다. 이는 기계 학습을 빠르게 수렴시키기 위해 어떤 파라미터들에 집중해 업데이트할지 우선순위를 정한다. 먼저 모델 병렬 처리를 위해 모델 파라미터들을 데이터의 분포를 분석해 종속성을 줄이기 위해 잠재 디리클레 할당(Latent Dirichlet Allocation, LDA)을 통해 워커 간 거의 독립적으로 할당한다. *STRADS*는 스케줄(Schedule)과 푸쉬 앤 풀(Push and Pull)이라는 두 단계로 이루어져 있다. 스케줄러(Scheduler)는 라쏘(Lasso) 방식을 통해 워커(Worker)에게 전달할 우선순위가 높은 파라미터를 결정한다. 그림 6과 같이 마스터(Master)들은 스케줄러로부터 받은 모델의 파라미터를 워커에게 푸쉬하여 부분 모델을 업데이트한다. 워커들은 자신들이 처리한 부분 결과를 스케줄러로 가져와 파라미터로서 커밋(Commit)한다. 이 과정을 통해 부분 모델을 담당하여 처리하고 있는 워커들 간에 모델 파라미터 종속성을 줄이면서 병렬적으로 학습을 진행할 수 있다. 이 작업은 효율적인 메모리 활용으로 학습 및 추론 단계에서 낮은 에러율(Error Rate)과 빠른 수렴을 입증했다. 하지만 *STRADS*는 단순한 다층 퍼셉트론 학습 모델에만 적용될 수 있다는 한계가 있고, 정확성과 속도 외에도 워커와 마스터 간의 통신 오버헤드 및 스테일니스(Staleness) 문제를 해결해야 한다는 문제가 남아있다.

그러나 위의 방식에서 소개된 모델 병렬 처리는 디바이스 간의 통신이 손실되는 링크(Link)가 없이 신뢰할 수 있는 네트워크라고 명시적 또는 암시적으로 가정하기 때문에, 불안정한 상황에서는 정상적으로 동작하지 않을 수 있다는 문제가 존재한다. 이 뿐만 아니라, 엣지 디바이스의 고전적인 문제인 다종(Heterogeneous)으로 이루어진 기기 환경에서 전체 디바이스들의 자원을 최적화해 활용하기 위해, 학습 모델은 디바이스 각기 다른 자원의 제약 및 커뮤니케이션 모델을 모두 고려하는 것이 필요하다. 또한, 학습 및 추론 과정에서 이전 데이터에 대한 학습이 마무리되기 전에 다음 데이터의 학습을 진행할 수 없고 하나의 데이터를 각 모델이 돌아가며 순차적으로 맡은 부분 신경망에 대한 학습을 진행하기 때문에, 모델 병렬 처리에서는 디바이스가 프로세싱을 완전하게 활용할 수 없는 자원 저활용 문제가 있다. 표 4에서 모델 병렬 처리 기법에 관한 논문들을 정리하여 소개한다.

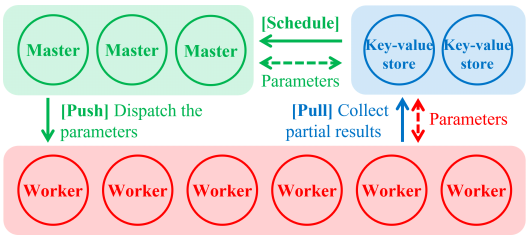


그림 6. *STRADS*에서 시스템 인터페이스의 전체적인 구조[24]
Fig. 6. Overall architecture of system interface in *STRADS*[24]

V. 하이브리드 병렬 처리 기법

데이터 병렬 처리나 모델 병렬 처리와 같은 기존의 병렬 처리 기법의 단점들을 극복하기 위해, 일부 연구에서는 그림 1(c)과 같이 여러 병렬 처리 기법을 병합한 하이브리드 병렬 처리 기법을 제안하였다. 모델 병렬 처리를 통해 큰 규모의 모델에 적용될 수 있어 데이터 병렬 처리만을 사용했을 때의 불가능했던 점을 극복했을 뿐만 아니라, 모델 병렬 처리의 저활용 문제 또한 여러 개의 데이터를 동시에 처리하여 해결하였다.

One Weird Trick[25]에서는 합성곱 신경망을 학습시키는 데에 병렬 처리 기법을 사용하였다. 해당 연구에서는 합성곱층이 전체 계산의 대부분을 지배하고 있는 반면, 완전 연결 계층은 파라미터 대부분을 차지

표 4. 모델 병렬 처리 기법에 대한 요약
Table 4. Summary of model parallelism

Paper	Problem	Methodology
AlexNet[19]	To classify highly challenging dataset	GPU based CNN model consists of both convolutional and fully-connected layers
Megatron-LM[22]	To train very large transformer models and implement a simple and efficient intra-layer model	MLP and self attention layer fuses groups of two GEMMs, eliminates a synchronization point
GNMT[23]	Fixed-length vector is a bottleneck in improving the performance of this basic encoder–decoder architecture	Basic encoder-decoder approach which learns to align and translate jointly
STRADS[24]	To design parameter scheduling, prioritization algorithm for accelerating ML training	Scheduling algorithm to decide which parameters to dispatch

하기 때문에, 합성곱층은 데이터 병렬 처리를 통해 계산 속도의 증가가 필요하며, 완전 연결 계층은 모델 병렬 처리를 통해 메모리 사용을 효율적으로 관리해야 한다는 것을 도출하였다. 즉, 하이브리드 병렬 처리 기법을 통해 각 계층의 분류에 따라 해당하는 병렬 처리 기법을 적용하였고, 이를 통해 6.16배의 속도가 증가함을 보였다. 하지만 일반적으로 합성곱층이 완전 연결 계층보다 앞에 위치하고, 그로 인해 전방 전달에서 전체 모델을 가지고 있는 데이터 병렬 처리에서 일부 모델을 가지고 있는 모델 병렬 처리로 전환되면서, 모든 디바이스가 합성곱층의 최종 아웃풋(Output)을 다른 모든 디바이스에 전달해야 한다. 후방 전달에서도 마찬가지로 완전 연결 계층에서 합성곱층으로 전달되는 경사도를 공유하기 위해 통신이 필요하고, 이러한 전달 방식으로 인해 지나친 통신 오버헤드가 발생한다는 한계가 있다.

여러 개의 병렬 처리 기법을 독립적으로 사용하는 연구들과는 다르게, 일부 연구에서는 동시에 사용할 수 있는 방법을 소개한다. *MP-DP*[26]은 데이터 병렬 처리의 문제점에 초점을 맞추었다. 데이터 병렬 처리에서는 워커(Worker)들이 모델의 전체를 복사해서 가지고 있어야 하고, 그로 인해 워커들끼리 주기적으로 파라미터를 공유해야 하므로, 워커의 수가 많을 경우 파라미터 동기화(Synchronization)를 위한 엄청난 오버헤드가 발생한다. 그래서 *MP-DP*에서는 디바이스의 수 관점에서 언제 데이터 병렬 처리만 사용하는 기법에서 데이터 병렬 처리와 모델 병렬 처리를 동시 적용하는 기법으로 전환하는 것이 가장 적절한 크로스오버(Cross-over) 시점인지 찾고자 하였다. 각 디바이스의 수에 따라서 데이터 병렬 처리와 하이브리드 병렬 처리의 실제 학습 시간을 측정하여, 최대의 학습 속도 향상을 달성할 수 있는 시점에서 병렬 처리 방식을 전환하였다. 하지만 미리 측정해둔 학습 시간은 실제로 학습이 진행될 때의 시간과 다르기 때문에, 최적 솔루션(Solution)과는 차이가 있다는 한계가 존재한다. 그뿐만 아니라, *MP-DP*는 전환 시점을 찾는 방법을 명확하게 제시한 것이 아닌, 실험에 기반하여 판단하기 때문에 반드시 전체 학습이 최소한 한 번은 이뤄져야지 알아낼 수 있다. 즉, 이 방법은 최초의 학습할 때는 쓸 수 없고 재학습 과정에서만 사용될 수 있다.

DDNN[27]은 클라우드(Cloud), 엣지(Edge), 그리고 엔드 디바이스(End Device)에 걸친 분산 학습 모델을 제안하였다. 추론 정확도를 더욱 높이는 동시에 네트워크 지연을 줄이기 위해, 그림 7와 같이 모델의 계층 중간에 지역적으로 결과를 처리하는 엑시트 포인트(Exit Point)를 다수 두었다. 이 지역 엑시트 포인트에서는 해당 지점에서 추론 결과를 종료할지를 결정할 뿐만 아니라 결과를 집계하여 엔드 디바이스, 엣지 디바이스 및 클라우드의 손실을 전역적으로도 최소화하는데 기여한다. 해당 논문에서는 지역적으로 분산된 엔드 디바이스가 데이터를 수집하고 훈련시킨다는 가정 하에, 학습 모델이 확장성이 큼을 보여준다. 학습 과정에서는 데이터 병렬 처리, 모델 병렬 처리를 사용하여 학습을 수행했으며, 다수의 디바이스들이 작동하지 않는 장애 상황에서도 정확성이 유지됨을 실험을 통해 입증했다. 이처럼 *DDNN*은 회복성이 있고 간단한 구현을 제공하지만, 학습이 불가능한 빈 이미지를 포함한 제한된 개수의 데이터셋을 사용하여 실험을 진행한 점에는 한계가 있다. 또한, 실제 실험 환경에서는 시스템(System) 아키텍처(Architecture)에 제안된 엣지 계층은 제외하고, 엔드 디바이스와 클라우드만을 사용하여 정밀히 검증되지 않았다고 볼 수 있다.

하이브리드 병렬 처리 기법은 데이터 병렬 처리 기법과 모델 병렬 처리 기법의 각 장점을 결합하였기 때문에 학습 모델의 확장성을 높일 수 있다. 이러한 특성을 살린 *DistBelief*[28]은 수많은 멀티코어(Multi-core) CPU를 사용하여 수십억 개의 파라미터를 학습할 수 있는 프레임워크(Framework)를 제안했다. 학습에 많은 기계를 사용할 때 생기는 파라미터 스테일니스(Staleness) 문제를 극복하기 위해, *DistBelief*는 모델 병렬 처리 기법과 분산 최적화 알고리즘을 결합하였다. 그림 8에서처럼 파라미터 서버가 비동기적으로 가중치(Weight) 파라미터를 업데이트하여, 부분 모델이 속도의 지연 없이 학습을 수행할 수 있다. 또한 코디네이터(Coordinator)를 두어, 데이터

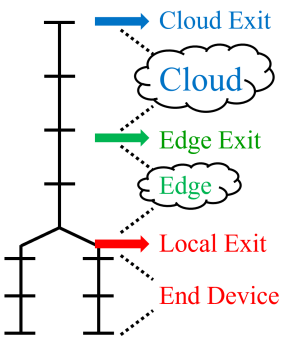


그림 7. 클라우드, 엣지, 엔드 디바이스로 구성된 *DDNN*의 구조[27]
Fig. 7. The architecture of *DDNN* consisting of cloud, edge, and end devices[27]

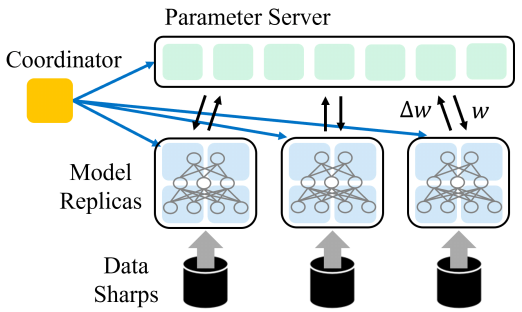


그림 8. *DistBelief*의 구조[28]
Fig. 8. The architecture of *DistBelief*[28]

배치를 분배하여 느려진 부분 모델을 관리한다. 이를 통해 *DistBelief*는 학습 속도를 올리면서도 수만 개의 CPU 코어를 사용하는 대규모의 학습 모델의 가능성을 보여주었다. 그러나 파라미터 서버에 대한 의존도가 높으며 CPU나 GPU와 같은 안정적인 통신 환경만을 고려하였다는 한계점이 있다.

표 5에서는 하이브리드 병렬 처리 기법에 관한 연구를 요약하였다.

Ⅵ. 파이프라인 병렬 처리 기법

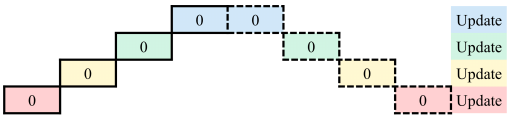
앞서 소개된 데이터 병렬 처리, 모델 병렬 처리, 그리고 하이브리드 병렬 처리는 본질적인 문제점을 가지고 있다. 예를 들어, 데이터 병렬 처리 기법은 모든 디바이스가 네트워크 모델의 완전한 복사본을 가지고 있어야하기 때문에, 큰 규모의 학습 모델에는 적합하지 않다. 큰 규모의 모델도 학습할 수 있도록 모델을 분할하는 모델 병렬 처리 기법 같은 경우에는 대부분의 디바이스가 가동되지 않는 상태(Idle State)이기 때문에 자원 저활용 문제가 심각하다. 이는 일반적인 러닝에서 각각의 데이터는 하나의 학습 데이터에 대해서도 전방 전달과 후방 전달이 순서대로 처리되어야 하므로, 한 번에 한 디바이스만 학습에 참여하고 다른 디바이스는 본인이 맡은 분할 모델의 차례가 올 때까지 기다려야하기 때문이다. 또한, 이전 데이터의 학습이 완료되기 전까지는 다른 데이터가 학습을 시작할 수 없으므로, 여러 대의 디바이스를 동시에 활용할 수 없다.

이러한 저활용 문제를 해결하기 위해, 최근 연구에서 파이프라인 스케줄링을 모델 병렬 처리 기법에 도입하였다. 대부분의 파이프라인 병렬 처리 기법에서는, 각 디바이스가 자신이 맡은 부분에 대한 계산을 완료하면, 바로 다음 분할 모델을 가지고 있는 디바이스에 계산 결과를 전달하고, 본인은 다음 데이터에 대해 학습을 진행한다. 이렇게 함으로써, 여러 대의 디바이스가 동시에 학습을 진행할 수 있고, 결과적으로 엄청난 속도 향상을 얻을 수 있다. 그뿐만 아니라, 모델을 분할하여 파이프라인 가속을 진행하기 때문에, 메모리를 효율적으로 관리할 수 있다. 이러한 파이프라인 병렬 처리 기법을 도입하기 위해서는 몇 가지 해결해야 하는 사항들이 있다. 먼저, 파이프라인 스케줄은 각 디바이스에 어떤 분할 모델을 할당해 있는지에 의해 결정된다. 두 번째로, 일부 경우에서는 같은 뉴런에 대한 파라미터가 여러 개의 버전(Version)이 생기므로 어떻게 관리할지 또한 중요한 요소이다. 마지막으로 파이프라인 러닝을 진행하기 위해 가장 최신의 최적화된 파라미터를 쓰지 못하고 업데이트 이전의 예전 파라미터를 쓰게 되는 스테일니스 문제도 발생한다.

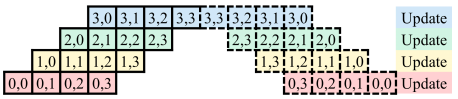
GPipe[29]는 여러 개의 배치를 동시에 처리하는 것이 아니라 하나의 배치를 분할하여 오버랩(Overlap)하는 방식이다. 그림 9(a)에서와 같이 단순하게 미니 배

표 5. 하이브리드 병렬 처리 기법에 대한 요약
Table 5. A summary of hybrid parallelism

Paper	Problem	Methodology	Hybrid Combination
One Weird Trick[25]	Train CNNs across multi-GPUs	Data parallelism in convolution layers and model parallelism in fully-connected layers	Data (in convolution layers) → Model (in fully connected layers)
MP-DP[26]	Find the optimal point to change from data-only to hybrid parallelism	An analytical framework based on experimental measurements	Data → Data + Model
DDNN[27]	Cloud, edge, end devices need to be learned jointly as a one learning model	distributed deep neural network framework with multiple local exits	Data → Data + Model
DistBelief[28]	Training distributed deep learning with large-scale clusters of machines	Two large-scale distributed optimization procedures: asynchronous parameter fetch and batch optimization	Data → Data + Model



(a) 기존의 미니 배치 기반 모델 병렬 처리 기법
(a) Basic model parallelism based on mini-batch

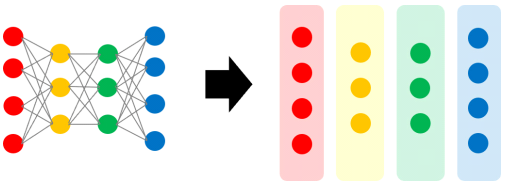


(b) *GPipe*의 마이크로 배치 기반 파이프라인
(b) Pipelined micro-batch in *GPipe*

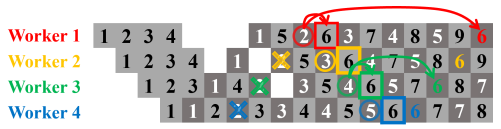
그림 9. 각 디바이스가 각 색의 블록을 맡는 경우의 파이프라인 스케줄 예시 (실선 박스는 전방 전달, 점선 박스는 후방 전달)[29]
Fig. 9. An example of work schedule where each device is in charge of the same colored blocks (boxes with full line are forward passes and boxes with dotted line are backward passes)[29]

치(Mini-batch)를 그림 9(b)처럼 마이크로 배치(Micro-batch)로 쪼개어 가동되지 않는 상태를 줄이는 방식이다. 마이크로 배치 분할을 통해 거의 선형 속도 증가를 얻을 수 있었으며, 세세하게 분할된 데이터를 사용함으로써 거대한 모델의 학습이 가능해졌다. 하지만 정확한 모델 분할과 태스크 사이의 균형 없이는, 속도 증진을 얻기 어렵다. 더욱이, 하나의 배치를 처리한 후에 각 디바이스가 파라미터를 갱신하고 다음 배치가 진행되기 때문에, 마이크로 배치를 사용함에도 불구하고 여전히 가동되지 않는 시간이 존재하기 때문에 다른 배치 기반 병렬 기법도 제안되고 있다.

가장 잘 알려진 파이프라인 병렬 처리 기법은 *PipeDream*[30]으로, 학습되는 데이터의 순서를 역전하지 못하는 계층 단위의 전방 전달과 후방 전달로 인해 자원 저활용 문제가 발생한다는 문제를 해결하고자 여러 개의 배치를 중첩하여 워커가 가동되지 않는 상태를 줄이는 방법을 제안하였다. 먼저, 그림 10(a)과 같이 계층 단위로 신경망을 분할한다. 그다음, 1-전방 전달-1-후방전달 (One-Forward-One-Backward, 1F1B) 기반 작업 스케줄을 세운다. 1F1B 스케줄은 각 계층을 맡은 디바이스가 그림 10(b)와 같이 한 번의 전방 전달과 한 번의 후방 전달을 반복한다는 것을 의미한다. 이를 통해 수렴 상태(Steady State) 이후에는 모든 디바이스가 최대로 활용될 수 있다. 예를 들어, 워커 2는 배치 2번의 후방 전달 후, 배치 5번의 전방 전달, 배치 3번의 후방 전달, 그리고 배치 6번의 전방 전달의 일련의 스케줄에 따라 일을 할 경우, 한 데이터의 학습 라운드트립(Round-trip) 과정에서 역전될



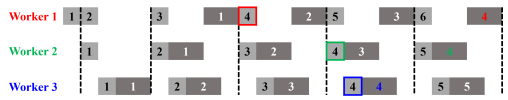
(a) 네 개의 계층으로 이루어진 신경망의 네 디바이스로의 분할
(a) Partitioned neural networks with four layers into four workers



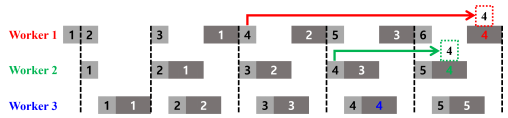
(b) 작업 스케줄 (연한 회색 박스는 전방 전달, 진한 회색 박스는 후방 전달이며 각 박스 안에는 배치 데이터의 고유 번호 표시)
(b) Work schedule (light-gray boxes are forward passes, whereas dark-gray boxes are backward passes, filled with batch ID)

그림 10. *PipeDream*에서의 모델 분할 방식과 작업 스케줄[30]
Fig. 10. Model partitioning and work schedule in *PipeDream*[30]

일이 없이 계속해서 작업할 수 있다. 하지만 데이터 병렬 처리와 비슷하게 다른 배치가 다른 디바이스에서 처리가 되고 있기 때문에, 파라미터를 세세하게 관리해야 한다. 일반적인 데이터 병렬 처리 과정에서는 이전 배치의 라운드트립이 끝난 후에 다음 데이터가 주입되기 때문에, 다음 데이터를 학습할 때는 가장 최신 업데이트된 파라미터를 사용할 수 있다. 즉, 배치 2번의 학습이 끝나고 배치 3번의 학습이 시작될 때는 배치 2번으로 업데이트된 파라미터를 쓸 수 있다는 것이다. 하지만 *PipeDream*에서는 그림 10(b)에서와 같이 2번으로 업데이트된 파라미터를 배치 6번부터 사용할 수 있게 된다. 이를 해결하기 위해, 가중치 스태싱(Weight Stashing)을 통해 전체 배치 관점에서 끝난 배치의 파라미터를 사용하는 것이 아니라, 각 워커의 관점에서 최신 파라미터를 사용할 수 있게 하였다. 예를 들어, 배치 6번을 학습하는 데에 모든 워커가 동일하게 배치 2번을 사용하는 것이 아니라, 워커 1은 배치 2번, 워커 2는 배치 3번, 워커 3은 배치 4번, 그리고 워커 4번은 배치 5번으로 업데이트된 파라미터를 사용하여 최대한 최신 파라미터를 사용할 수 있게 하였다. 하지만 단순하게 계층 단위로만 분할한다는 한계점이 존재한다. 뿐만 아니라, 가중치 스태싱을 사용함에도 불구하고 여전히 파라미터의 스테일니스 문제가 발생하는데, 이는 전방 전달을 할 때 가장 최신



(a) *PipeDream*의 실제 기반 파라미터
(a) Real weights of *PipeDream*



(b) *SpecTrain*의 예측 기반 파라미터
(b) Predicted weights of *SpecTrain*

그림 11. *PipeDream*과 *SpecTrain*에서의 파라미터 버전[31]
Fig. 11. Used weight versions in *PipeDream* and *SpecTrain*[31]

의 파라미터를 사용하지만, 후방 전달할 시점에서 판단하였을 때는 훨씬 더 이전 파라미터이기 때문이다.

이러한 가중치 스테일니스 문제를 해결하기 위해, *SpecTrain*[31]은 역전파를 할 때 예측한 파라미터 값을 사용한다. 그림 11(a)의 *PipeDream*은 배치 4번의 전방 전달된 값을 바탕으로 역전파에서 업데이트할 경사도(Gradient)가 계산된다. 이에 반해, *SpecTrain*은 그림 11(b)에서와 같이 현재 알고 있는 파라미터를 기반으로 후방 전달이 진행될 때의 파라미터를 예측한다. 정확한 예측을 통해 작업 대부분에서 정확도 감소가 없었다는 것을 보였다.

파이프라인 병렬 처리 기법에 대한 대표적인 연구들을 표 6에 정리하였다.

Ⅶ. 결 론

본 논문에서는 데이터, 모델, 하이브리드, 및 파이프라인 병렬 처리 기법에 대해 소개하였다. 데이터 병렬 처리 기법은 전체 데이터셋을 부분 데이터셋으로 분할하여, 학습 속도 향상을 얻고자 한다. 모델 분할 처리 기법은 큰 규모의 모델까지 포괄할 수 있도록 딥러닝 모델을 쪼개어 여러 워커에 할당한다. 일부 연구에서는 두 가지를 모두 고려한 하이브리드 병렬 처리 기법을 제안하였다. 이를 넘어서서, 최근에는 데이터와 모델 뿐만 아니라 다양한 특성을 분할하는 종합적인 병렬 처리 방식에 관한 연구를 진행하기도 하였다. 마지막으로, 최대한 자원을 효율적으로 활용할 수 있도록 파이프라인을 병렬 처리 기법에 도입하는 연구를 소개하였다.

데이터 병렬 처리 기법은 단순하게 학습한 데이터만 다르다는 간단한 방법으로, 모든 워커들이 네트워크 모델 전체를 복제해서 가지고 학습을 진행한다. 학습이나 추론 과정들이 여러 데이터에 대하여 동시다발적으로 여러 디바이스에서 발생하며, 다양한 파라미터 동기화 방법을 통해 속도 향상 및 정확성 향상을 목표로 한다. 모델 자체를 가지고 있기 때문에 어떤 신경망 모델에서나 사용될 수 있다는 장점이 있지만, 그 모델이 크면 메모리 자원이 한정적일 경우 적용될 수 없다는 한계가 있다. 모델을 병렬화해서 실행할 경우, 딥러닝 모델은 서로 다른 디바이스에 나눠서 할당된다. 모델 병렬 처리 기법의 가장 중요한 특성은 각 디바이스가 관리해야 하는 파라미터의 수를 줄임으로써 메모리의 한계가 있는 디바이스에서도 학습이 진행될 수 있도록 한다는 것이다. 하이브리드 병렬 처리 기법은 여러 개의 병렬 처리 기법의 결합으로, 일반적으로는 데이터 병렬 처리 기법과 모델 병렬 처리 기법을 모두 사용하는 방식이다. 마지막으로, 하나의 데이터에 대해 전방 전달부터 후방 전달까지의 차례가 주어졌을 때, 여러 데이터 혹은 데이터 일부를 병렬화하여 워커를 최대한 활용할 수 있다. 즉, 학습시키는 데이터를 더 잘게 쪼갠다거나, 서로 다른 부분을 맡은 디바이스가 동시에 학습에 참여할 수 있게 하는 등의 방법을 통해 워커들이 쉬는 시간을 최소화하는 것이다. 가장 어려운 문제 중 하나는 동시 학습으로 인해 여러 버전의 파라미터가 존재하고, 어느 파라미터를 선정하여 사용할지 판단하는 것이다.

정리하자면, 간단한 모델을 사용하여 학습할 경우에는 데이터 병렬 처리 기법이 가장 쉽고 빠르게 처리할 수 있는 방법이라고 할 수 있다. 반면, 크고 깊은

표 6. 파이프라인 병렬 처리 기법에 대한 요약
Table 6. A summary of pipelined parallel learning

Paper	Problem	Methodology
GPipe[29]	To solve the low utilization problem of model parallelism and enable to train giant networks	Reduce idle time and memory usage by splitting batches
PipeDream[30]	To minimize training time of DNNs over multiple GPUs	One-forward-one-backward pipeline scheduling over layer-wise partitioned DNNs
SpecTrain[31]	To solve the weight staleness issue	Predict weights of backpropagation time from forwarding

딥러닝 모델을 이용할 때는 모델 병렬 처리 기법이 적절하다고 할 수 있다. 여기에 자원 저활용 문제까지 고려한다면, 파이프라인 병렬 처리 방법을 도입한다면 큰 모델에서조차 빠르게 학습시킬 수 있는 방안이 될 수 있어, 주어진 컴퓨팅 자원의 제한 조건에 따라 적절한 처리 기법을 선택하는 것이 중요하다고 할 수 있다.

References

[1] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *IEEE Computational Intell. Mag.*, vol. 13, no. 3, pp. 55-75, 2018.

[2] L. Deng, G. Hinton, and B. Kingsbury, "New types of deep neural network learning for speech recognition and related applications: An overview," in *2013 IEEE Int. Conf. Acoustics, Speech and Signal Process.*, pp. 8599-8603, May 2013.

[3] T. H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma, "PCANet: A simple deep learning baseline for image classification?," *IEEE Trans. Image Process.*, vol. 24, no. 12, pp. 5017-5032, 2015.

[4] S. Dara and P. Tumma, "Feature extraction by using deep learning: A survey," in *2018 2nd Int. Conf. Electron., Commun. and Aerospace Technol. (ICECA)*, pp. 1795-1801, Mar. 2018.

[5] S. P. Singh, A. Kumar, H. Darbari, L. Singh, A. Rastogi, and S. Jain, "Machine translation using deep learning: An overview," in *2017 Int. Conf. Comput., Commun. and Electron. (Comptelix)*, pp. 162-167, Jul. 2017.

[6] Y. M. Park, S. Y. Ahn, E. J. Lim, Y. S. Choi, Y. C. Woo, and W. Choi, "Deep learning model parallelism," *Electron. and Telecommun. Trends*, vol. 33, no. 4, pp. 1-13, 2018.

[7] R. Mayer and H. A. Jacobsen, "Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1-37, 2020.

[8] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, et al., "Scaling distributed machine learning with the parameter server," in *11th USENIX Symp. OSDI 14*, pp. 583-598, 2014.

[9] R. J. Hewett and T. J. Grady II, "A linear algebraic approach to model parallelism in deep learning," arXiv preprint arXiv: 2006.03108, 2020.

[10] Y. Li, M. Yu, S. Li, S. Avestimehr, N. S. Kim, and A. Schwing, "Pipesgd: A decentralized pipelined sgd framework for distributed deep net training," in *Advances in NIPS*, pp. 8045-8056, 2018.

[11] L. Bottou and O. Bousquet, "13 the tradeoffs of large-scale learning," *Optimization for Mach. Learn.*, vol. 351, 2011.

[12] M. Zinkevich, M. Weimer, L. Li, and A. J. Smola, "Parallelized stochastic gradient descent," in *Advances in NIPS 2010*, pp. 2595-2603, Vancouver, Canada, Dec. 2010.

[13] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in NIPS 2011*, pp. 693-701, Granada, Spain, Dec. 2011.

[14] H. Li, A. Kadav, E. Kruus, and C. Ungureanu, "Malt: Distributed data-parallelism for existing ml applications," in *Proc. EuroSys 2015*, pp. 1-16, Bordeaux, France, Apr. 2015.

[15] J. M. Cecilia, J. M. Garcia, A. Nisbet, M. Amos, and M. Ujaldón, "Enhancing data parallelism for ant colony optimization on gpus," *J. Parall. and Distrib. Comput.*, vol. 73, no. 1, pp. 42-51, Jan. 2013.

[16] D. Rothchild, A. Panda, E. Ullah, N. Ivkin, I. Stoica, V. Braverman, J. Gonzalez, and R. Arora, "Fetchsgd: Communication-efficient federated learning with sketching," arXiv preprint arXiv:2007.07682, 2020.

[17] S. Wang, D. Li, and J. Geng, "Geryon: Accelerating distributed cnn training by network-level flow scheduling," in *INFOCOM 2020-IEEE Conf. Comput. Commun.*, pp. 1678-1687, 2020.

[18] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale

hierarchical image database," *IEEE Conf. CVPR*, pp. 248-255, Miami, United States, Jun. 2009.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in NIPS 2012*, vol. 25, pp. 1097-1105, Lake Tahoe, United States, Dec. 2012.

[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE CVPR*, pp. 770-778, Las Vegas, Unites States, Jun. 2016.

[22] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training multi-billion parameter language models using model parallelism," arXiv preprint arXiv:1909.08053, 2019.

[23] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, and M. Norouzi, "Google's neural machine translation system: Bridging the gap between human and machine translation," arXiv preprint arXiv:1609.08144, 2016.

[24] S. Lee, J. K. Kim, X. Zheng, Q. Ho, G. A. Gibson, and E. P. Xing, "On model parallelization and scheduling strategies for distributed machine learning," *J. Contribution*, 2014.

[25] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," arXiv preprint arXiv:1404.5997, 2014.

[26] S. Pal, E. Ebrahimi, A. Zulfiqar, W. Fu, V. Zhang, S. Migacz, and P. Gupta, "Optimizing multi-GPU parallelization strategies for deep learning training," *IEEE Micro*, vol. 39, no. 5, pp. 91-101, 2019.

[27] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," *IEEE 37th ICDCS*, pp. 328-339, Atlanta, United States, Jun. 2017.

[28] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, "Large Scale Distributed Deep Networks," *Advances in NIPS*, vol. 25, pp. 1223-1231, 2012.

[29] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, et al., "Gpipe: Efficient training of giant neural networks using pipeline parallelism," in *Advances in NIPS 2019*, pp. 103-112, Vancouver, Canada, Dec. 2019.

[30] A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, N. Devanur, G. Ganger, and P. Gibbons, "Pipedream: Fast and efficient pipeline parallel dnn training," arXiv preprint arXiv:1806.03377, 2018.

[31] C.-C. Chen, C.-L. Yang, and H.-Y. Cheng, "Efficient and robust parallel dnn training through model parallelism on multi-gpu platform," arXiv preprint arXiv:1809.02839, 2018.

윤 진 이 (JinYi Yoon)

2017년 2월 : 이화여자대학교 컴퓨터공학과 학사
2019년 2월 : 이화여자대학교 컴퓨터공학과 석사
2019년 3월~현재 : 이화여자대학교 컴퓨터공학과 박사과정

<관심분야> 엣지 컴퓨팅, 머신 러닝 기반 네트워크, 차세대 무선 네트워크, 이동 애드혹 네트워크, 로컬라이제이션

[ORCID:0000-0001-9457-4432]

한 나 영 (Nayoung Han)

2017년 2월 : 이화여자대학교 컴퓨터공학과 학사
2019년 2월 : 이화여자대학교 수학과 석사
2019년 3월~현재 : 이화여자대학교 수학과 박사과정

<관심분야> 대수적 부호이론 (Self-dual code, Self-orthogonal code, LCD code, Convolutional code, Quantum code), Cryptographic function, Design theory

[ORCID:0000-0002-1192-5434]

이 지 호 (JiHo Lee)

2019년 2월 : 단국대학교 소프트웨어학과 학사
2021년 2월 : 이화여자대학교 컴퓨터공학과 석사
2021년 8월~현재 : Univ. of Virginia 컴퓨터과학과 박사과정

<관심분야> 엣지 컴퓨팅, 머신 러닝 기반 네트워크, IoT 보안, 차세대 무선 네트워크, VANET

[ORCID:0000-0002-5099-4512]

이 형 준 (HyungJune Lee)

2001년 8월 : 서울대학교 전기공학부 학사
2006년 6월 : Stanford Univ. 전자공학과 석사
2010년 9월 : Stanford Univ. 전자공학과 박사
2012년 3월~현재 : 이화여자대학교 컴퓨터공학과 부교수

<관심분야> 사물인터넷, 분산 딥러닝, 엣지 컴퓨팅

[ORCID:0000-0003-4655-4298]