2018
Electronics and
Telecommunications
Trends

Deep Neural Network &
Object Recognition 특집

딥러닝 모델 병렬 처리

Deep Learning Model Parallelism

박유미 (Y.M. Park, parkym@etri.re.kr) 고성능컴퓨팅연구그룹 책임연구원
안신영 (S.Y. Ahn, syahn@etri.re.kr) 고성능컴퓨팅연구그룹 책임연구원
임은지 (E.J. Lim, ejlim@etri.re.kr) 고성능컴퓨팅연구그룹 책임연구원
최용석 (Y.S. Choi, shine24@etri.re.kr) 고성능컴퓨팅연구그룹 책임연구원
우영춘 (Y.C. Woo, ycwoo@etri.re.kr) IDX 원천기술연구실 책임연구원
최 완 (W. Choi, wchoi@etri.re.kr) IDX 원천기술연구실 책임연구원

Deep learning (DL) models have been widely applied to AI applications such image recognition and language translation with big data. Recently, DL models have becomes larger and more complicated, and have merged together. For the accelerated training of a large-scale deep learning model, model parallelism that partitions the model parameters for non-shared parallel access and updates across multiple machines was provided by a few distributed deep learning frameworks. Model parallelism as a training acceleration method, however, is not as commonly used as data parallelism owing to the difficulty of efficient model parallelism. This paper provides a comprehensive survey of the state of the art in model parallelism by comparing the implementation technologies in several deep learning frameworks that support model parallelism, and suggests a future research directions for improving model parallelism technology.

* DOI: 10.22648/ETRI.2018.J.330401

*이 논문은 2016년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임[No.2016-0-00087, 대규모 딥러닝 고속 처리를 위한 HPC 시스템 개발].

Ⅰ. 머리말

불과 십여 년 전 암흑 속의 AI 분야를 되살리고 꽃피우는 데 가장 큰 역할을 한 딥러닝 기술은 최근 4차 산업 혁명의 핵심 기술로 자리 잡으며 사람의 지능을 모사하는 다양한 응용에 적용되고 있다. 이미지 인식, 얼굴 인식, 음성 인식 등 인식 분야를 비롯하여 주식 시장에서는 주가를 예측하고, 의료계에서는 질병 진단 지원과 발병률 예측을 시도하며 심지어 음악, 미술 분야에서는 새로운 작품을 창조해 내는 단계에 이르고 있다.

이와 같이 응용의 범위가 넓고 다양해짐에 따라 응용들의 목표 성능을 높이기 위해 다양한 방식으로 딥러닝 모델이 진화하고 있다. 모델의 규모가 커지고 있고(Large Scale Model)[1], 모델들을 결합하여(Ensemble Model) 목표 성능을 높이고 있고[2], 이질적 형태의 특징을 함께 처리하기도 하며(Multimodal Model)[3], 기존의 모델을 다른 분야에 적용시키기도 한다(Transfer Learning)[4].

이러한 발전 추세 중 딥러닝 모델의 규모가 커진다는 것은 모델을 구성하는 뉴런이 많아지고 뉴런을 연결하는 은닉 계층이 깊어짐을 뜻한다. 이는 SW 엔지니어링 관점에서 볼 때, 딥러닝의 인퍼런스나 트레이닝 과정에 계산량이 많아지고 메모리 요구량이 커지면서 한 컴퓨터(또는 계산 디바이스)의 한계를 넘는 상황이 발생할 수 있기 때문에 다수의 컴퓨터가 이를 효율적으로 나누어 처리하기 위한 분산 처리 기술이 요구됨을 의미한다[5].

본고에서는 딥러닝 분산 처리 기술 중 하나인 모델 병렬 처리 기술을 고찰한다. Ⅱ장에서는 딥러닝 모델 병렬 처리의 필요성과 기술적 추세 및 이슈를 설명하고, Ⅲ장에서는 딥러닝 분산 프레임워크들의 모델 병렬 처리 기술을 소개하며, Ⅳ장에서 이를 비교 분석한다. 마지막으로 Ⅴ장에서 향후 연구가 필요한 방향을 점검하며 결론을 맺는다.

Ⅱ. 딥러닝 모델 병렬 처리 개요

1. 필요성

최근 딥러닝 모델들은 응용의 인식 성능을 높이기 위해 모델의 계층이 깊어지고, 고려해야 할 특징이 많아지는 대규모 네트워크로 구축되고 있다. 대규모 딥러닝 네트워크를 효율적이고 빠르게 트레이닝하기 위해 학계와 산업계에서는 다중 GPGPU(General-Purpose Computing on Graphics Processing Units)를 활용한 딥러닝 분산 처리 기술을 딥러닝 프레임워크 개발에 적용하고 있다[6]-[18].

현재까지 연구된 딥러닝 분산 처리 기술은 크게 트레이닝 데이터를 다수의 컴퓨터에서 분산 처리하는 데이터 병렬 처리(Data Parallelism)와 딥러닝 네트워크를 다수의 컴퓨터에서 분산 처리하는 모델 병렬 처리(Model Parallelism)로 나뉜다[1], [8].

이중 딥러닝 모델 병렬 처리는 딥러닝 네트워크를 하나의 컴퓨터(또는 GPGPU, FPGA 등 계산 디바이스)에서 처리하지 못할 경우, 네트워크를 분할하여 다수의 컴퓨터에서 처리하는 방법이다. 대표적인 예로 딥러닝 네트워크 트레이닝 과정에서 계산을 위해 메모리에 상주해야 하는 데이터인 파라미터(가중치, 그래디언트 등을 포함하여 트레이닝 과정에 학습되는 값들)의 개수가 늘어나 계산 가속용으로 이용하는 GPGPU 메모리에 한 번에 상주할 수 없는 경우에 모델 병렬 처리가 필요하다. [1]의 예시와 같이 학습 모델은 점점 대규모화되어 파라미터 개수가 전장 유전체 분석 모델의 경우 수억에서 수십억 개, 구글 브레인의 이미지 분석 딥러닝 모델에는 수십억 개에서 수백억 개, 뉴스 분석을 위한 토픽 모델에서는 1조 개에 이르고 있다.

〈표 1〉은 BVLC Caffe(v1.0.0)[19]에서 NVIDIA Titan X(12GB 메모리)를 이용하여 대규모 CNN 모델들의 ImageNet 트레이닝 중 실측한 데이터이다[20]. 총 6개의 딥러닝 모델에 다양한 해상도의 ImageNet 데이터에

〈표 1〉 대규모 CNN 모델의 파라미터 분석[20]

	Inception_v1 (googlenet)	Inception_v3	Resnet_50	Inception_resnet_v2	Resnet_152	VGG16
레이터 개수(컴포넌트수)	19(223)	13(417)	50(223)	467(1071)	152(674)	16(44)
Imagenet 데이터 크기	256×256	300×300	256×256	299×299	299×299	256×256
가중치 개수	13,393,411	24,721,100	31,754,269	56,119,101	66,488,387	138,357,544
가중치 저장 메모리 필요량(MB)	51	94	121	214	254	264
계산시간(ms)	207	289	205	267	176	220
미니배치 크기	64	22	18	6	7	80
총 메모리 사용량(MB)	11,695	11,643	11,651	11,785	11,999	11,591

대해 1회 반복(1 미니배치로 1회 반복) 시 메모리에 상주하는 가중치 개수와 규모, 계산 시간, 총 메모리 사용량을 측정하였다. 각 모델들에서의 총 메모리 사용량은 NVIDIA Titan X의 메모리 크기 12GB에 근접한다. 모델마다 미니배치의 크기를 달리하여 측정한 이유는 모델의 특성과 ImageNet 입력 데이터의 크기에 따라 필요한 메모리 요구량이 달라져 이를 GPGPU의 메모리를 최대한 이용할 수 있는 최대 미니배치를 적용한 결과이다. 만약 〈표 1〉의 미니배치보다 더 큰 미니배치를 적용할 경우, 예를들어 BVLC Caffe에서는 Titan X의 메모리 용량을 넘는 미니배치로 설정한 경우 트레이닝 실행이 불가능하다. SGD(Stochastic Gradient Descent) 기법을 이용하는 분산 딥러닝 모델의 경우, 미니배치 크기가 적을수록 정답에 늦게 수렴하거나 정확도가 저하되기 때문에 가능한 미니배치를 크게 설정하게 된다[8], [21], [22]. 위 실험과 동일한 실행 환경에서 딥러닝 모델 병렬 처리를 적용한다면 한 컴퓨터당 미니배치를 더 크게 설정할 수 있다.

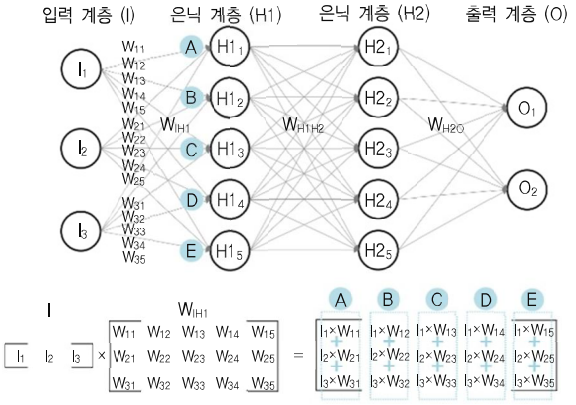
2. 기술적 이슈

본 절에서는 딥러닝 모델 병렬 처리의 기술 추세와 이슈를 정리한다. 첫째, 모델 분할을 설명하기에 앞서 현재 프레임워크들에서 제공하는 모델의 표현 방법을 고찰하고, 둘째, 모델 병렬 처리를 위한 모델 분할 방법과 이슈를 점검하며, 셋째, 스케줄링과 통신 방법 등 분할 모델의 분산 병렬 처리 기술에 대해 살펴본다.

가. 딥러닝 모델 표현(Representation)

사람의 신경망을 모사한 딥러닝 모델은 주로 방향성이 있는 비순환 그래프(DAG: Directed Acyclic Graph)로 표현된다. 노드로 표현되는 뉴런은 활성화 함수를 의미하며 입력이 뉴런을 통해 전달되어 가는 동안 동일한 단계에서 동일한 활성화 함수를 실행하는 뉴런들을 묶어 계층이라 부른다. 뉴런들을 연결하는 에지는 입력 계층으로부터 출력 계층으로 나아가는 포워드 방향과 출력 계층으로부터 입력 계층으로 오류가 역전파되는 백워드 방향을 내포하며 연결된 뉴런 사이의 가중치를 포함하고 있다.

딥러닝 모델 트레이닝과 인퍼런스 과정에서 에지는 뉴런 사이에 데이터 전달을 의미하는데, 이전 뉴런의 계



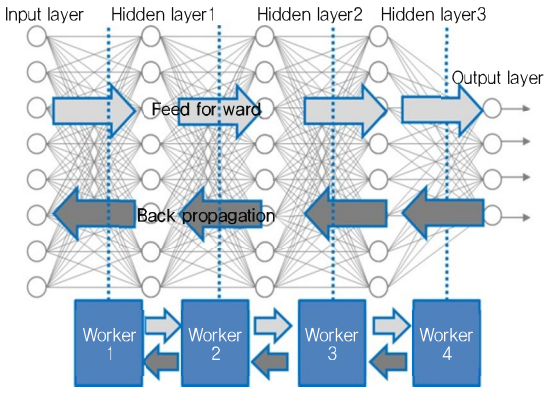
(그림 1) 딥러닝 모델 표현–그래프와 데이터구조

산 결과에 가중치를 곱하여 벡터, 매트릭스, 텐서 형태로 다음 뉴런으로 전달한다. (그림 1)의 예에서 입력 계층의 벡터값 I와 가중치(입력 계층 I와 은닉 계층 H1 사이) 매트릭스 $\mathbf{W}_{IH}$를 곱한 결과가 은닉 계층 H1에 입력으로 전달된다. (그림 1)의 $H1_1$ 뉴런은 $\mathbf{I}\times\mathbf{W}_{IH1}$ 중 첫 번째 열(A)의 합을 입력으로 받고, $H1_2$ 뉴런은 $\mathbf{I}\times\mathbf{W}_{IH1}$ 중 두 번째 열(B)의 합을 입력으로 받는다. 모델 병렬 처리 시 분할된 모델이 참조하는 가중치는 포워드 시와 백워드 시 직교하지 않을 수 있기[14], [16] 때문에 발생하는 이슈들을 본 장의 '다' 절에서 설명한다.

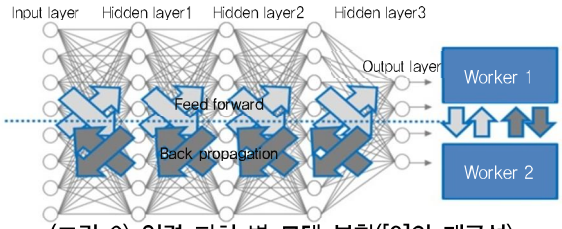
나. 딥러닝 모델 분할(Partitioning)

딥러닝 모델은 인공신경망(Artificial Neural Network)의 한 종류로 딥러닝 네트워크 모델이라고도 불린다. 네트워크는 구조적 연결성을 부각하는 용어이고 모델은 네트워크 구조를 포함한 동작까지 바라보는 포괄적인 용어로 쓰인다. 본 절에서는 딥러닝 네트워크의 연결성 측면에서 분할 방법을 기술하고자 하나 네트워크 분할 보다 모델 분할이라는 용어가 더 널리 쓰이고 있으므로 모델 분할이라 칭하겠다.

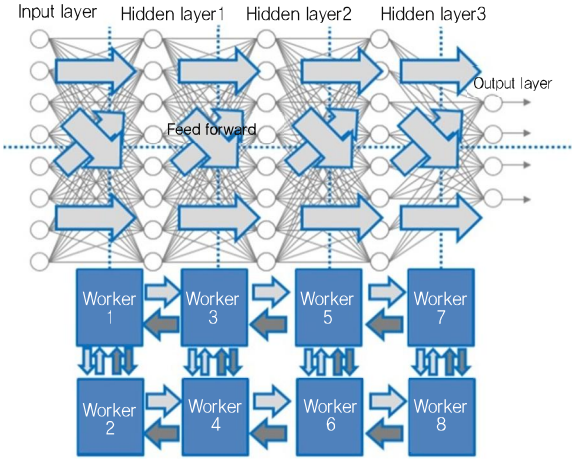
모델 분할 방법으로는 뉴럴 네트워크의 구조적 관점에서 계층별 분할[6], [9], [23]과 입력 피처 별 분할[23], [25], 하이브리드 방법들[8], [16] 그리고 뉴런 단위의 자유 분할 방법들이 연구되고 있다[16], [18].



(그림 2) 계층 별 모델 분할([9]의 재구성)



(그림 3) 입력 피처 별 모델 분할([9]의 재구성)



(그림 4) 혼합형(하이브리드) 모델 분할([9]의 재구성)

(그림 2)는 [9]의 그림을 재구성하여 DNN(Deep Neural Network)의 계층별 모델 분할의 사례를 보였다. 모델을 구성하는 다중 계층들을 한 계층씩 또는 여러 계층 묶음으로 나누어(세로 방향으로 분할) 각기 다른 컴퓨터 워커(Worker1, Worker2, Worker3, Worker 4)에 할당하여 처리한다. 딥러닝 트레이닝의 피드 포워드 시에는 i번째 계층에서 계산한 값들을 i+1번째 계층에 전달하고, 역전파 시에는 i번째 계층에서 오류로 편미분된 값들을 i−1번째 계층으로 전달한다. 따라서 계층 분할로 각 계층의 계산 작업을 처리하는 워커 들 간 데이터 전송이 필요하다.

두 번째 (그림 3)은 입력 피처를 그룹핑하는 분할 방법으로(가로 방향으로 분할) 초기 피처들을 두 그룹으로 나누어 서로 다른 컴퓨터 워커(Worker1과 Worker 2)에 할당하여 처리하는 사례이다. 입력 피처 별 분할은 i번째 계층에서 i+1번째 계층으로 넘어갈 때 Worker1이 로컬 가중치와 계산된 특징값을 Worker 2에 전달해야

하고, Worker 2도 마찬가지로 로컬 가중치와 계산된 특징값을 Worker 1에 전달해야 한다. 즉, 분산 워커들 사이에 로컬 파라미터를 매 계층, 매 피처 별 분할마다 전달해야 하는 의존성이 생기고 이로 인한 데이터 교환도 불가피하다.

세째로 (그림 4)와 같이 계층별 분할과 피처별 분할을 융합한 하이브리드 분할이 가능하다([9] 그림 재구성). 모델이 지극히 커서 이전 두 방법의 분할 만으로 처리가 어려운 경우 이용할 수 있으나 지나치게 많은 통신 오버헤드를 초래한다. (그림 4)에서는 피드 포워드(밝은 회색 화살표)시의 데이터 전달만 도시한 것으로 역전파(진한 회색 화살표) 시에는 이와 동일한 양과 횟수의 통신 비용이 발생한다.

마지막으로 계층별, 입력 피처별 구분 없이 뉴런 단위로 자유롭게 분할하는 방법이다. 즉, 딥러닝 네트워크를 부분 그래프로 자유롭게 분할하고 다른 워커에서 실행하는 방법이다. 모델 분할의 자유도를 높이는 대신 정교한 스케줄링이 필요하고 모델 병렬 처리의 복잡성이 증가할 수 있다.

상기 어떤 모델 분할 방법이든 계층 간에는 순차적으로 수행되어야 하는 '계층 간 의존성'이 내포되어 있다. 따라서 모델 병렬 처리 효과를 최대화하기 위해서는 계층 간 의존성을 효율적으로 다룰 수 있는 스케줄러가 필요하다.

다. 딥러닝 모델 병렬 처리

1) 파라미터 갱신 방법

모델 병렬 처리가 데이터 병렬 처리와의 차이점은 크게 두 가지로 요약된다. 첫째, 분할된 모델이 참조하는 파라미터가 서로 종속적일 수 있다는 점이다. '나' 절에서 보았듯이 분할된 모델은 실행의 선후 관계로 인해 의존성이 생긴다. 또한, 동일한 분할 모델이라 하더라도 피드 포워드 시와 역전파 시에 서로 다른 부분의 파라미터를 이용하고, 서로 다른 분할이라 해도 일부 파라미터를 공유할 수 있다. 이는 분할 모델 간 직교성이 없다는 의미이다. 일례로 '가'절에서 설명한 (그림 1)의 모델을 계층별로 분할했을 경우, 피드 포워드 시에는 파라미터 벡터를 열(column) 단위로 참조하지만, 역전파 시에는 행(row) 단위로 참조한다. 분할된 모델 간 종속적인 데다 직교하지 않는 성질로 인해 워커들이 아무 전략없이 전역 파라미터를 갱신한다면 트레이닝 수렴 속도가 느려지거나 알고리즘 오류가 발생할 수 있다[16].

둘째, 모델을 구조적으로 균등하게 분할했다 하더라도 분할된 모델의 파라미터가 각기 다른 속도로 수렴하므로 일부 분할 모델의 파라미터 처리에서 병목이 발생할 수 있다. 데이터 병렬 처리의 경우 모든 분산 워커가 로컬에서 전역 파라미터를 갱신하고 동기/비동기적으로 원격 파라미터 서버에 반영하는 방식만으로 처리가 가능하나[5] 모델 병렬 처리의 경우는 위에서 기술한 분할 모델의 특성을 고려한 병렬 처리 최적화 기법이 필요하다.

2) 통신 방법

모델 병렬 처리는 데이터 병렬 처리에 비해 트레이닝 과정에서 많은 통신 비용(통신 횟수×통신량)이 든다. 예를 들어 '나' 절에서 살펴본 DNN 모델 분할의 예(그림 2-4)에서 통신 횟수는 1 회 반복(하나의 미니배치로 1회 반복)시 다음과 같다. L은 계층 분할 개수, F는 피처 분할 개수이다.

- (그림 2) 계층 별 모델 분할: $(L-1)\times 2$
- (그림 3) 입력 피쳐별 분할: $(L-1)\times F\times(F-1)\times 2$
- (그림 4) 하이브리드 분할: $(L-1)\times F\times F\times 2$

위의 (그림 2)와 (그림 3)의 사례로 L과 F가 커질수록 통신 횟수가 L과 F^2의 곱에 비례하여 증가함을 알 수 있다. 통신 횟수는 L과 F가 결정되면 더 이상 줄일 수 없지만, 통신량을 줄여 전체적인 통신 비용을 감소시킬

수 있다. 예를 들어, 네트워크의 조밀한 연결 부위를 분할할수록 전달해야 할 데이터양이 많아지므로 연결이 성긴 네트워크를 분할하고 전달할 데이터를 압축[10]하거나 샘플링함[11]으로써 통신량을 줄일 수 있다.

딥러닝 모델 병렬 처리를 수행하는 다수의 컴퓨터 워커들 간의 통신 즉, 모델 파라미터를 전송하는 방법으로 널리 쓰이는 있는 방법으로는 Message Passing Interface(MPI), general-purpose RPC(gRPC), NVIDIA Collective Communications Library(NCCL)가 있다. 그리고 특정 분산 딥러닝 플랫폼을 위해 개발된 방식으로는 Gloo, MLSL, Baidu-allreduce 등이 있다.

MPI는 메시지 패싱 병렬 프로그래밍을 위해 표준화된 데이터 통신 라이브러리로서 프로세스 기준으로 작업을 할당하며 프로세스간 피어 투 피어 토폴로지 기반의 통신 방식이다. GPUDirect RDMA를 지원하기 위해 CUDA-Aware MPI를 지원한다. MPI는 Caffe2, Cognitive Toolkit(CNTK), Chainer MN 등에서 사용된다.

gRPC는 범용 목적으로 Google에 의해 개발된 오픈소스 프레임워크로서 클라이언트-서버 기반의 원격 프로시저 호출방식으로 분산 프로세스간 통신을 수행하며, Protobuf를 기반으로 메시지를 인코딩/ 디코딩하기 때문에 단순하며 XML 대비 매우 빠른 속도를 보인다.

NCCL은 NVIDIA에서 개발된 GPU 간 집합(Collective) 통신에 많이 사용되는 통신 방법이다. 노드 내부 GPU 간에는 NVLink, PCIe, GPU Direct P2P를 통해 통신을 하며(NCCL 1), 노드 간 GPU들 사이에서는 Socket(Ethernet) 또는 Infiniband(with GPU direct RDMA) 네트워크를 통해 통신한다(NCCL 2). 가장 빠른 통신 성능을 제공하는 하드웨어 구성은 NVLink를 지원하는 서버와 Tesla 계열 GPU 그리고 노드 간 Infiniband 네트워크로 연결하는 것이다. NCCL은 Ring 방식으로 CUDA kernel이 통신을 수행한다. Cognitive Toolkit(CNTK), Torch, NVCaffe, Caffe2, Tensorflow 등 대부분의 분산 딥러닝 플랫폼에서 사용되고 있다.

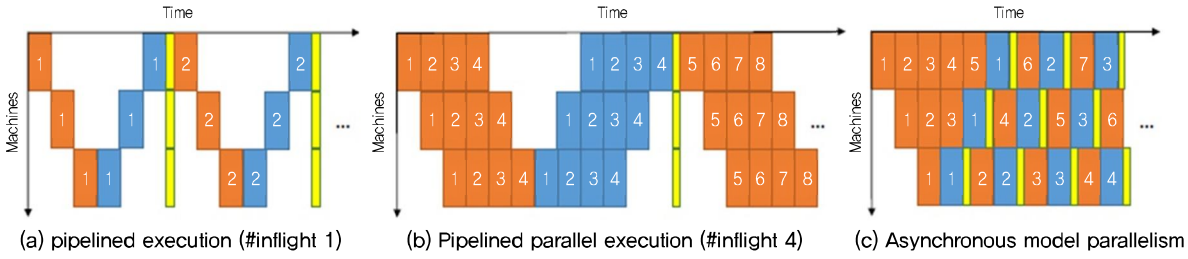
Gloo는 Facebook에서 개발된 기계학습에 유용한 집합 통신 라이브러리로 Caffe2에서 머신 간 통신을 위해 사용되고 있다[12]. Gloo는 barrier, broadcast, allreduce를 포함한다. 하부 네트워크로 IP 또는 Infiniband (or RoCE)를 지원하고, Infiniband가 있으면 GPU-Direct가 사용된다. 그러나 Redezvous 지원을 위해서는 Redis를 필요로 한다.

MLSL(Machine Learning Scaling Library)은 Intel에서 개발된 기계학습 확장을 위한 라이브러리로, MPI primitives를 기반으로 구현되었다. Intel Omni-Path, Infiniband, Ethernet을 지원한다.

Baidu-allreduce는 대규모 메시지 집합 통신을 지원한다. Point-to-Point MPI 프리미티브를 이용하여 Ring-Allreduce 알고리즘을 구현하였다[13].

3) 최적화 방법

분할된 모델은 서로 종속적이고 직교하지 않기 때문에 모델 분할 만으로는 병렬 처리의 효과를 보기 어렵다. 따라서 분할 실행의 성능을 높일 수 있는 최적화 방법이 연구되고 있다.



(a) pipelined execution (#inflight 1) (b) Pipelined parallel execution (#inflight 4) (c) Asynchronous model parallelism

(그림 5) 동기식/비동기식 파이프라이닝 비교[24]

가장 대표적인 최적화 방법은 파이프라이닝 방법이다[16], [24]. (그림 5)는 [24]에서 3대의 머신으로 동기식/비동기식 모델 인스턴스 파이프라이닝 할 때 걸리는 시간을 간트 차트로 표현한 그림이다.

박스는 분할된 모델 인스턴스에 대한 계산 시간을 표현하고, 붉은색 박스는 포워드 계산 시간이고, 파란색 박스는 백워드 계산 시간, 노란색 박스는 파라미터 갱신 시간이다. (a)는 분할된 모델 인스턴스 하나를 3대의 머신이 동기적으로 모델 병렬 처리를 하는 경우이고, (b)는 4개의 인스턴스를 3대의 컴퓨터가 동기적으로 모델 병렬 처리를 하는 경우이며, (c)는 4개의 인스턴스를 3대의 컴퓨터가 비동기적으로 모델 병렬 처리를 하는 경우이다. 비동기식으로 처리하는 경우, 스테일 그래디언트(로컬 가중치를 계산하는 중에 다른 워커가 글로벌 가중치를 수정함으로써 로컬 가중치가 최신상태가 아니게 됨)를 발생시킴에도 실험에서 빠른 수렴과 하드웨어 활용도를 높일 수 있다.

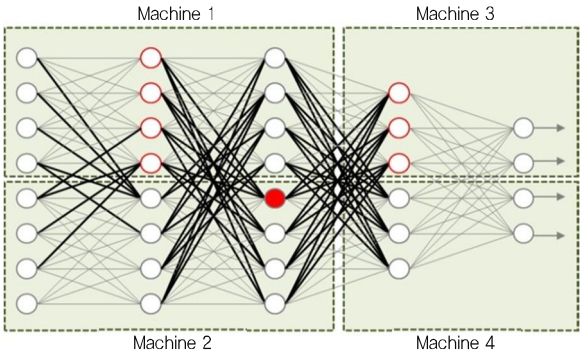
두번째 최적화 기법은 분할 모델 스케줄링이다[16]. [16]에서는 모델을 분석하여 상호 의존도와 우선순위를 파악하고 이를 기반으로 워크로드를 분배하는 방식으로 로드밸런싱하여 병렬 처리의 성능을 높이는 스케줄러를 개발하였다. 자세한 내용은 Ⅲ장 Petuum에서 설명한다.

Ⅲ. 딥러닝 프레임워크의 모델 병렬 처리 기술

본 장에서는 상기 세가지 기술적 추세와 이슈 관점에서 모델 병렬 처리를 지원하는 5개의 분산 딥러닝 프레임워크를 고찰한다.

1. DistBelief

DistBelief[23]는 2011년부터 시작된 구글 브레인 프로젝트에서 개발된 대규모 딥 뉴럴 네트워크의 트레이닝과 인퍼런스를 위한 구글의 1세대 분산 시스템이다. 수 십억 개의 파라미터를 가지는 대규모 딥 뉴럴 네트워크를 수천 대의 컴퓨터(수만 CPU 코어)에서 분산 트레이닝 할 목적으로 개발되었다.



(그림 6) 모델 병렬 처리 예

(그림 6)은 DistBelief에서 보이는 모델 병렬 처리의 한 예이다[23]. 5개 계층으로 구성된 딥 뉴럴 네트워크를 4개의 모델로(점선 직사각형) 분할하여 머신(Machine) 1, 2, 3, 4 에서 각기 실행하는 경우이다. 여기서 모델 분할은 모델을 구성하는 뉴런 연결의 분할을 의미한다. 뉴런은 그와 연결된 이전 뉴런으로부터 데이터와 가중치를 입력받아 계산한 결과를 연결된 다음 뉴런으로 전달해야 한다. 만약 모델이 분할되지 않았다면 컴퓨터의 로컬 메모리 상에서 계산의 반복만으로 처리될 수 있으나 모델이 분산되면 머신 간 통신(굵은 선)이 필요하다.

(그림 6)의 네트워크가 그림 왼쪽에서 오른쪽으로 피드 포워드 되는 경우라면, Machine 2의 빨간색 뉴런은 로컬 뉴런과 Machine 1의 뉴런으로부터 데이터를 전달받아 계산하고, 그 결과를 Machine 3과 Machine 4내의 연결 뉴런으로 전달해야 한다.

DistBelief 사용자(딥러닝 모델 개발자)는 python 기반의 스크립트 인터페이스를 이용하여 C++클래스로 구현된 레이어들을 조합함으로써 딥러닝 네트워크를 정의한다. 정의된 네트워크는 실행 패턴이 고정되어 단순한 피드 포워드 신경망 외 RNN과 같은 모델은 지원하지 못한다. DistBelief 프레임워크는 전역적으로 파라미터를 관리하는 서버와 로컬 가중치 계산용 워커 프로세스들로 구성되고, 트레이닝이나 인퍼런스 중에 정적 모

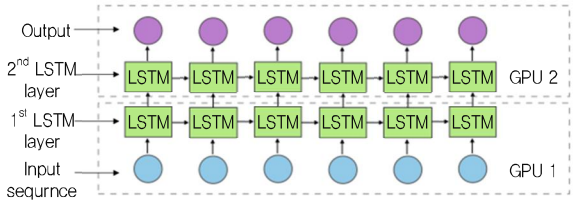
델 분할에 따른 분산 컴퓨터상의 계산 병렬화, 메시지 전달, 컴퓨터 간 동기화 작업을 실행함으로써 모델 병렬 처리를 지원한다.

모델 병렬 처리를 목적으로 개발된 Downpour SGD와 Sandblaster L-BFGS 알고리즘은 모델 분할의 병렬 처리 기술(1단계)에 더해 모델을 다중 컴퓨터에 복사하여 데이터를 분할하여 병렬 처리하기 위한 2단계 병렬화 기법이다. Downpour SGD는 워커들과 파라미터 서버 간 비동기식 파라미터 최적화 기술이고, Sanblaster L-BFGS는 하나의 중재자(coordinator)가 워커들과 파라미터 서버에 메시지를 보내며 배치 최적화를 관장하는 기술이다. 워커과 파라미터 서버 간 통신 방법으로 머신 간에는 메시지 패싱 방법을 머신 내에서는 멀티쓰레딩 방법을 사용한다.

2. MXNet

MXNet[7]은 CXXNet, Minerva, Purine2 개발자들이 연합하여, 과거의 경험을 반영하여 효율성과 유연성을 목표로 개발한 딥러닝 프레임워크로 2015년 github에 공개되었다.

모델 병렬 처리에 있어 MXNet은 특히 LSTM의 집중하고 있다[6]. (그림 7)은 2계층 LSTM 모델을 계층별로 분할하여 다른 GPU에서 실행하는 사례이다. GPU 1에서 첫 번째 입력 시퀀스의 계산을 계산을 마치면 결과를 GPU2로 전달한다. GPU2가 첫 번째 LSTM 계층의 결과를 받아 두 번째 LSTM 계층을 계산함과 동시에 GPU 1은 다음 입력 시퀀스의 트레이닝을 시작한다. 이 경우 분할 모델 간 공유부분이 없어 워커들 간의 파라미터 갱신 경쟁은 없지만, GPU 간 중간 결과를 전송할 때 통신이 빈번하게 발생한다. 따라서 통신 시간을 최소화하기 위해 이웃 계층은 같은 GPU에 배치시키고, 다중 GPU 간 워크로드가 균등하게 배분되도록 분할하고 다른 종류의 계층은 다른 계산을 한다는 것을 염두에 두며 모델 병렬 처리를 구현해야 한다.



(그림 7) MXNet에서 모델 병렬 처리 기법[6]

MXNet은 딥러닝 모델을 방향성 비순환 그래프(Directed Acyclic Graph) 형태의 심볼릭 그래프로 표현하도록 심볼릭(Symbolic) (또는 선언전(Declarative)) 프로그래밍과 지시적 (Imperative) 프로그래밍 인터페이스를 Python, C++ 등 프론트엔드 개발언어용 API로 제공한다. 이러한 이유는 Symbol API로 제공되는 심볼릭 프로그래밍은 뉴럴 네트워크의 계산 구조를 명시하고 최적화하기 쉬우나 네트워크 모델이 한번 고정되면 실행 중 변경이 어렵기 때문에 이를 보완하고자 파라미터 업데이트와 인터렉티브 디버깅에 유용한 지시적 프로그래밍도 함께 제공한다. NDArray라는 수학 계산이 가능한 고정 크기의 다차원 배열과 연산 함수들을 제공함으로써 지시적 프로그래밍이 가능하며, MXNet은 NDArray 연산 함수를 분산 시스템에 맞게 확장되거나 코드 실행을 지연시킴으로써 유연함을 제공한다. 계산양이 많아 작업 부하가 큰 NDArray 연산 함수(예, NDArray.dot())는 호출 시점에 디펜던시 엔진(Dependency engine)이라는 백엔드 엔진에 보내져 의존성 분석 후 실행되도록 코드 실행 지연시킨다. 디펜던시 엔진은 의존성 분석 스케줄링 알고리즘을 이용해 전달된 연산 함수들과 데이터 의존성을 분석한 후 이들이 독립적 작업들이라면 사용 가능한 하드웨어로 자동 병렬 실행시킴으로써 리소스의 효율과 병렬성을 높인다.

정리하면, 모델 개발자가 MXNet에서 제공하는 심볼릭 프로그래밍과 지시적 프로그래밍 방법을 이용하여 프로그래밍하면 MXNet에서는 모델 병렬 처리를 위해 모델의 계층별 분할과 NDArray 연산 함수 단위의 계산 병렬 처리를 자동으로 실행하는 구조이다.

3. Petuum

Petuum[14], [15]은 CMU에서 개발한 오픈 소스 분산 기계학습 프레임워크로서 구조적으로 파라미터 서버(PS server), 스케줄러(Scheduler), 워커(Worker)들로 구성된다. 데이터 병렬처리와 모델 병렬 처리를 지원하기 위해 기능적으로는 Bösen이라 불리는 비동기 분산 키-밸류 저장소와 STRADS(structure-Aware Dynamic Scheduler)라 명명된 모델 병렬화 스케줄러[16], 그리고 다양한 기계학습 라이브러리를 제공한다.

특히, 모델 병렬 처리를 위해 트레이닝 1회 반복을 기준으로 '반복 당 진행률(정답에 수렴하는 정도)'과 '반복 처리량' 향상이라는 보다 구체적인 목표로 분할 방법과 스케줄링 기법을 제공한다. 모델 분할 방법으로는 정적 분할과 동적 분할을 제공하는데 정적 분할은 도메인 지식에 기반하여 실행 전에 분할을 고정함으로써 '반복 처리량'을 향상시키는 데 중점을 두고, 동적 분할은 실행 중 모델 분할 간 상호 의존성 테스트를 수행하여 가장 관련이 적은 파라미터를 병렬 처리하도록 함으로써 '반복 당 진행률' 향상에 중점을 두었다.

모델 병렬 처리를 지원하기 위한 Petuum만의 가장 주요 특징 중 하나는 네트워크 구조를 인지하여 계산 작업의 우선순위를 매기고 세부 단위로 파라미터 업데이트 동작을 스케줄링하는 STRADS 스케줄러이다. 개발자가 SchMP 명령어를 모델의 목적에 맞게 구현하면 STRADS 서비스를 이용해 SchMP 명령어를 분산 클러스터에서 실행한다. 모델 개발자가 구현해야 하는 SchMP 명령어는 1) schedule(), 2) update() (push()[15]), 3) aggregate(pull()[15])이다. schedule()로 변경할 파라미터를 선택하고, 워커들은 schedule()에 의해 변경된 파라미터를 수신하여 update()로 자신의 로컬 파라미터를 변경하고, 스케줄러는 다시 aggregate()를 이용하여 워커들로부터 수정된 파라미터 값을 모아 전역 파라미터를 변경한다. 즉, 모델 개발자는 SchMP 명령어들에 응용에 특화된 스케줄러를 자유롭게 구현하여 STRADS 서비스에서 실행할 수 있다. [16]은 기계 학습 응용들에서 순위 기반 스케줄링, 라운드 로빈 스케줄링, 블록 기반 스케줄링 등 다양한 스케줄링 사례를 보인다.

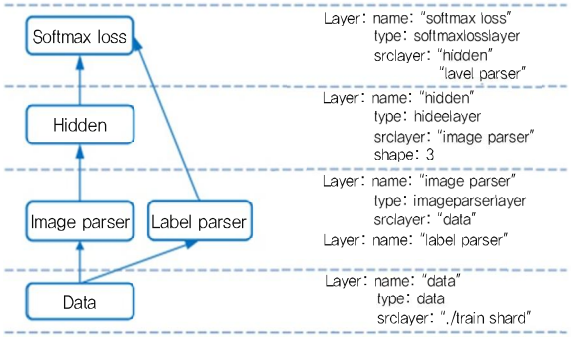
4. SINGA

싱가포르 대학이 개발하여 2015년 Apache 오픈소스로 공개한 SINGA는 분산 딥러닝 플랫폼의 이용성(Usability)과 확장성(Scalability) 향상을 목표로 개발된 분산 딥러닝 플랫폼[17]이다.

SINGA는 서버 그룹과 워커 그룹으로 구성되는데, 서버 그룹은 모델 파라미터의 완전한 복사본을 가지고 워커 그룹으로부터의 읽기/변경하기 요청을 처리하며 그들끼리 주기적으로 파라미터를 동기화한다. 워커 그룹은 하나의 서버 그룹에 속하여 그 서버 그룹이 관리하는 모델에 대해 파라미터를 트레이닝한다.

SINGA에서는 딥러닝 모델을 NeuralNet이라는 단방향으로 연결된 계층의 집합으로 정의하며 (그림 8)과 같이 기술한다. 계층은 데이터 계층, 파서 계층, 뉴런 계층, 로스 계층, 기타 계층(모델 분할 시 계층들을 연결하는 계층)으로 구분되고 각 계층의 입력 계층을 기술함으로써 단방향 연결이 표현된다.

딥러닝 모델(NeuralNet)을 분할하는 방법으로는 계층별 분할, 단일 계층 내 분할(Batch Dimension, Feature Dimension), 하이브리드 분할 방법을 지원한다. 단일



(그림 8) NeuralNet 형상[15]

계층 내 분할 방법으로는 피처 행렬을 열로 분할하는 피처 디멘젼 분할과 피처 행렬을 행으로 분할하는 배치 분할이 지원되고, 따라서 단일 계층 내 분할은 다수의 서브 레이어를 만든다. 실제 학습 모델의 분할은 NeuralNet 인스턴스를 생성할 때 일어나며, 클러스터 설정 정보에 따라 워커 그룹과 서버 그룹에 배치되어 실행된다. 각 하위 계층에는 위치 ID가 지정되며, 이 ID를 기준으로 한 워커에게 전달된다. 일부 연결 계층은 하위 계층을 연결하기 위해 자동으로 삽입되는데 예를 들어 두 개의 연결된 하위 계층이 두 개의 서로 다른 워커에 있는 경우 한 쌍의 브릿지 계층이 삽입되어 그 사이에 피쳐(및 그래디언트)를 전송한다. 두 개의 계층이 다른 차원으로 분할될 때 피쳐 행 (또는 열)을 연결하는 연결 계층과 피쳐 행 (또는 열)을 나누는 슬라이스 계층이 삽입된다.

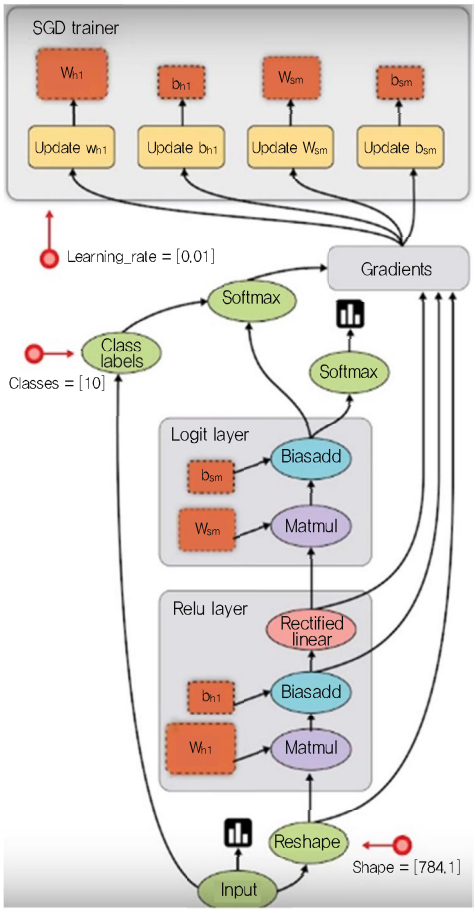
SINGA는 프로세스 내에서 쓰레드 간 또는 프로세스 간 통신을 위해 메시징 API를 제공하는데 사용자가 하부 구현기술(MPI 또는 ZeroMQ)을 선택할 수 있다.

5. TensorFlow

구글의 TensorFlow[18]는 DistBelief를 잇는 구글의 2세대 대규모 분산 머신러닝/딥러닝 프레임워크이다.

TensorFlow는 학습 알고리즘을 표현하기 위해 상태를 가지는 데이터플로우 그래프를 정의하여 실행시킨다. (그림 9)와 같이 TensorFlow가 지원하는 데이터플로우 그래프는 계산 단위를 나타내는 노드와 노드 간 전달되는 데이터의 입출력을 표현하는 기본 에지, 노드 간 실행 순서를 명시하는 특수 에지로 구성되는데, 에지는 데이터 또는 제어의 흐름을 나타내도록 방향성을 띤다. 이러한 에지들로 노드 간 의존성을 명시적으로 표현할 수 있기 때문에 시스템은 병렬 처리 가능한 오퍼레이션을 찾아내기가 쉽다.

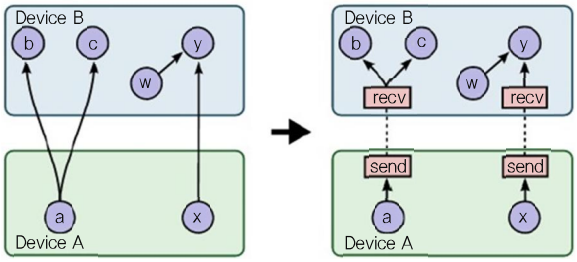
노드 간 의존성과 데이터 흐름을 제어하기 위한 다양한 연산자들을 제공하는데, 서브 그래프의 실행을 건너



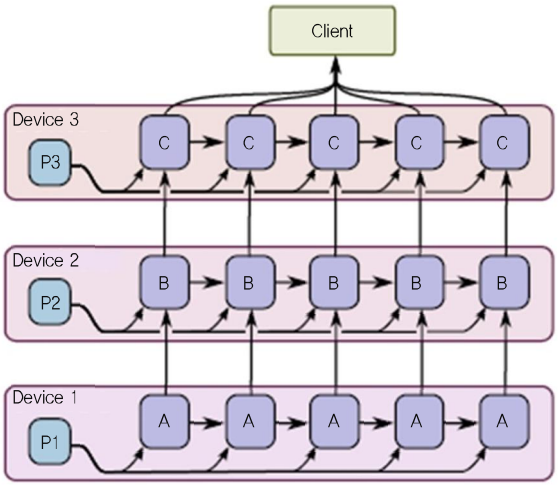
(그림 9) TensorFlow 데이터플로우 그래프[18]

뛸 수 있는 Switch와 Merge 연산자, 반복을 표현할 수 있는 Enter, Leave, NextIteration 연산자, 조건에 따라 데이터 흐름을 변경할 수 있는 if-conditional 및 while-loopf로 등이 지원된다. 딥러닝 모델을 학습시키기 위한 클라이언트 프로그램에서는 이와 같은 데이터플로우 그래프를 미리 선언적으로 구성한 후 실행시킨다. TensorFlow에서는 데이터플로우 그래프의 노드 별로 실행 디바이스를 지정할 수 있기 때문에 그에 따라 자유롭게 모델 분할이 가능하다. 모델을 분할하여 디바이스(CPU 또는 GPGPU, 기타 정의한 디바이스)에 배치한 후 (그림 10)과 같이 데이터 송수신이 필요한 부분에 자동으로 send/recv 노드가 추가되어 데이터의 송수신을 처리한다. 이때 디바이스가 단일 머신에 위치한 경우

에는 다중 디바이스 간(CPU와 CPU, CPU와 GPGPU, GPGPU와 GPGPU) 데이터 복사가 발생하고, 디바이스가 서로 다른 머신에 위치한 경우에는 TCP 또는 RDMA를 이용하여 원격 통신한다. (그림 11)은 언어 번역 등에 이용되는 시퀀스-투-시퀀스 학습을 위한 다계층 LSTM 모델을 3개의 디바이스에서 병렬 처리하는 사례이다. 모델 개발자는 계층별 디바이스 배치만으로 다계층 LSTM 모델의 병렬 처리를 수행할 수 있다.



(그림 10) TensorFlow 데이터플로우 그래프[18]



(그림 11) 다중 계층 LSTM 모델 병렬처리[18]

Ⅳ. 딥러닝 프레임워크의 모델 병렬 처리 기술 비교 분석

〈표 2〉는 상기 설명한 분산 딥러닝 프레임워크들의 모델 병렬 처리 기술을 Ⅱ장에서 설명한 기술적 이슈별로 비교 분석한다.

일반적으로 딥러닝 모델은 계산 노드 간의 단방향성 그래프로 표현하고 프레임워크에서 제공하는 연산 함수나 레이어 함수, 기계학습 라이브러리를 이용하여 모델을 개발한다. 모델 분할 방법은 대부분의 프레임워크에서 계층별 분할이 가능하고 DistBelief와 TensorFlow, Petuum에서는 분할 기준을 제한하지 않는다. 모델 분할은 대부분 개발자가 선언적 프로그래밍 방법으로 구현한 대로 정적으로 분할되고 실행되나 MXNet에서는 지시적 프로그램 방법을 제공하여 실행 중에 자동 계산 병렬이 가능하고, 또한 Petuum에서는 STRADS 스케줄러를 이용하여 실행 중 동적 분할 실행을 지원한다.

분산 딥러닝 트레이닝에서 파라미터 전달을 위해 주로

〈표 2〉 딥러닝 프레임워크들의 모델 병렬 처리 비교 분석

분산 딥러닝 프레임워크(개발자)	모델 표현 및 프로그래밍 방법	모델 분할 방법	통신 방법 또는 통신 미디어	특이 사항
DistBelief (Google)	DAG	DAG 기반 자유 분할	머신 내 멀티쓰레드 머신 간 메시지 패싱	-
MXNet (DMCL team)	계산 그래프 선언적 프로그래밍 지시적 프로그래밍	계층별 분할	ZeroMQ, MPI	파이프라이닝
PETUUM (CMU)	절차적 프로그래밍	스케줄러 STRADS에 의한 자유 분할 정적 분할/동적분할	파라미터 교환 채널 : 분산 공유 메모리 스케줄링 제어 채널	스케줄러 STRADS 동적 스케줄링, 파이프라이닝
SINGA (Singapore Univ.)	NeuralNet 지시적 프로그래밍	계층별 분할 + 단일 계층내 분할 (배치/피처 디멘전)	MPI, ZeroMQ	-
TensorFlow (Google)	207	노드별 자유 분할	gRPC over TCP, ROCE	-

이용되는 통신 기술은 MPI, gRPC, NCCL들이고, 이외 특정 분산 딥러닝 플랫폼을 위해 개발된 방식으로는 Gloo, MLSL, Baidu-allreduce 등이 있다.

앞서 살펴본 바와 같이, 각 프레임워크들은 효율적이고 확장성이 높은 모델 병렬 처리를 위하여 최적의 통신 비용을 찾기 위한 모델 분할 기술과 분할된 모델 간의 효율적인 스케줄링 기술을 지원한다.

V. 맺음말

본고에서는 딥러닝 모델 병렬 처리의 필요성과 기술적 이슈를 분석하고, 현재 모델 병렬 처리를 지원하는 딥러닝 분산 프레임워크들인 DistBelief, MXNet, Petuum, SINGA, TensorFlow에서의 모델 병렬 처리 기술을 이슈별로 비교 분석하였다.

모델 병렬 처리의 기술적 이슈는 SW 엔지니어링 관점에서 볼 때, 프로그램 실행 과정에서 계산(모델)을 나누어 처리하는 기술로서 분할된 계산(모델)은 작업 순서를 지켜야 하고 선 계산(모델) 작업의 결과를 후 계산 작업에 전달해야 하므로 빈번한 통신 때문에 효율성과 전반적인 성능을 높이기 어려운 기술이다. 이에 모델 병렬 처리는 소규모의 딥러닝 모델에는 적절치 않고, 트레이닝 중 파라미터가 한 컴퓨터 디바이스에 효율적으로 모두 상주할 수 없을 정도의 대규모 모델에 적용함을 전제로 해야 한다.

결론적으로 효과적인 대규모 딥러닝 모델 병렬 처리를 위해서는 모델 표현과 분할 과정에서 전체 모델을 분석하여 통신을 최소화하는 분할 전략과 분할된 모델을 성능이 각기 다른 디바이스에 효율적으로 배치하는 전략이 필요하고, 모델 분할 실행 시에는 전체 모델을 파악하며 계층 간의 의존성에 기반하여 흐름을 제어할 수 있는 코디네이터 또는 스케줄러가 필요하다. 또한, 모델 병렬 처리의 확장성을 고려한 분산 토폴로지 기술에 대한 종합적 연구가 필요하다.

약어 정리

CNN	Convolutional Neural Network
DAG	Directed Acyclic Graph
DNN	Deep Neural Network
GPGPU	General-Purpose computing on Graphics Processing Units
gRPC	General-purpose Remote Procedure Call
IR	Intermediate Representation
L-BFGS	Limited-memory Broyden Fletcher Goldfarb Shanno
LSTM	Long Short Term Memory
MPI	Message Passing Interface
NCCL	NVIDIA Collective Communications Library
PCIe	Peripheral Component Interconnect Express
RNN	Recurrent Neural Networks
ROCE	RDMA over Converged Ethernet
SGD	Stochastic Gradient Descent
STRADS	STRucture-Aware Dynamic Scheduler

참고문헌

[1] E.P. Xing and Q. Ho, "A New Look at the System, Algorithm and Theory Foundations of Large-Scale Distributed Machine Learning," KDD 2015 Tutorial.

[2] L. Rokach, "Ensemble-Based Classifiers," *Artif. Intell. Rev.*, vol. 33, no. 1-2, Feb. 2010, pp. 1-39.

[3] J. Ngiam et al., "Multimodal Deep Learning," *Proc. Int. Conf. Mach.* Learning, Bellevue, USA, 2011, pp. 1-9.

[4] S.J. Pan and Q. Yang, "A Survey on Transfer Learning," *IEEE Trans Knolw.* Data Eng., vol. 22, no. 10, 2010, pp. 1345-1359.

[5] 안신영 외, "딥러닝 분산 처리 기술 동향," 전자통신동향분석, 제31권 제3호, 2016, pp. 131-141.

[6] Training with Multiple GPUs Using Model Parallelism. https://mxnet.incubator.apache.org/faq/model_parallel_lstm.html

[7] T. Chen et al., "MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems," In *Proc. LearningSys*, Montreal, Canada, Oct. 10, 2015.

[8] A. Krizhevsky, "One Weird Trick for Parallelizing Convolutional Neural Networks," 2014, arXiv preprint arXiv: abs/1404.5997.

[9] K. Zhang, "Data Parallel and Model Parallel Distributed Training with Tensorflow," http://kuozhangub.blogspot.kr/2017/08/data-parallel-and-model-parallel.html

[10] A. Øland and B. Raj, "Reducing Communication Overhead in Distributed Learning by an Order of Magnitude (Almost)," In *IEEE Int. Conf. Acoustics, Speech Signal Process.*, Brisbane, Australia, 2015, pp. 2219–2223.

[11] T. Xiao et al., "Fast Parallel Training of Neural Language Models," *Int. Joint Conf. Artif. Intell.*, Melbourne, Australia, Aug. 2017. pp. 4193–4199.

[12] P. Goyal et al., "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour," June 2017, arXiv: 1706.02677.

[13] D. Amodei et al., "Deep Speech 2: End-to-End Speech Recognition in English and Mandarin." *ICML*, NY, USA, June 2016, pp. 173–182.

[14] E.P. Xing et al., "Petuum: A New Platform for Distributed Machine Learning on Big Data Eric," *IEEE Trans. Big Data*, vol. 1, no. 2, 2015, pp. 49–67.

[15] S. Lee et al., "On Model Parallelization and Scheduling Strategies for Distributed Machine Learning," *Int. Conf. Neural Inform. Process. Syst.*, vol. 2, 2014, pp. 2834–2842.

[16] J.K. Kim et al., "STRADS: a Distributed Framework for Scheduled Model Parallel Machine Learning," *Proc. Eur. Conf. Comput. Syst.*, London, UK, Apr. 2016, pp. 1–16.

[17] W. Wang et al., "SINGA: Putting Deep Learning in the Hands of Multimedia Users," In *ACM Multimedia*, Brisbane, Australia, Oct. 2015, pp. 25–34.

[18] M. Abadi et al., "TensorFlow: A System for Large-Scale Machine Learning," Proc. USENIX *Symp. Oper. Syst. Des. Implement.*, Savannah, GA, USA, 2016, pp. 265–283.

[19] J. Yangqing et al., "Caffe: Convolutional Architecture for Fast Feature Embedding," In *Proc. Int. Conf. Multimedia*, Orlando, FL, USA, Nov. 2014, pp. 675–678.

[20] S.Y. Ahn et al., "A Novel Shared Memory Framework for Distributed Deep Learning in High-Performance Computing Architecture," accepted in ICSE 2018.

[21] T.M. Breuel, "The Effects of Hyperparameters on SGD Training of Neural Networks," 2015, arXiv preprint arXiv: 1508.02788.

[22] P. Goyal et al., "Accurate, Large Minibatch SGD: Training Imagenet in 1 Hour," 2017, arXiv preprint arXiv: 1706.02677.

[23] J. Dean et al., "Large Scale Distributed Deep Networks," *NIPS'12*, vol. 1, Dec. 2012, pp. 1223–1231.

[24] A. Gaunt et al., "AMPNet: Asynchronous Model-Parallel Training for Dynamic Neural Networks," 2018, arXiv preprint arXiv: 1705.09786.

[25] D. Shrivastava et al., "A Data and Model-Parallel, Distributed and Scalable Framework for Training of Deep Networks in Apache Spark," 2017, arXiv preprint arXiv: 1708.05840v.