훈련 및 검증 성능 개선을 위한 텐서플로우 병렬 처리 기법*
(A Parallel Processing Scheme on TensorFlow for Improving Training and Validation Performance)

최 진 서† 강 동 현‡
(Jinseo Choi) (Donghyun Kang)

요 약 대부분의 딥 러닝(Deep Learning) 시스템은 모델의 훈련 및 검증을 위해 많은 시간을 소모한다. 그러나, 단일 쓰레드(Single Thread) 기반의 데이터 전처리 및 배치 과정으로 인해 대기 시간(Wait Time)이 발생하고 그 결과 GPU 및 CPU 의 사용률을 낭비하는 경향이 있다. 본 논문에서는 멀티 쓰레드(Multi Thread) 기반으로 모델의 훈련 및 검증 과정을 효율적으로 수행하기 위한 새로운 기법을 제안한다. 제안 기법은 모델 복사 과정을 사용함으로써 훈련과 검증 과정을 최대한 중첩(Overlapping)시키며, 그 결과 전반적인 CPU 와 GPU 의 사용률을 향상시킨다. 제안 기법을 평가하기 위해 우리는 텐서플로우(TensorFlow)을 이용하여 합성곱 신경망(CNN)을 구현하였다. 실험 결과, 제안 기법이 기존 기법 대비 전체 훈련 및 검증 시간을 22.4% 단축시키는 것을 확인할 수 있었다.

키워드 : 멀티 쓰레드, 딥 러닝, 텐서플로우, GPU 및 CPU 사용률

Abstract Most deep learning systems spend a lot of time on model training and validation. However, they sometimes tend to waste GPU and CPU resources because the pre-processing and batch processes based on a single thread result in a wait time. In this paper, we propose a new scheme that efficiently handles training and validation processes based on multi-threads. The proposed scheme can overlap the training and validation processes as much as possible by using a model copy operation that extends the processes with multi-threads. As a result, it improves the overall utilization of CPU and GPU. For evaluation, we implemented a convolutional neural network (CNN) using the TensorFlow framework. As a result, we clearly confirm that the proposed scheme saves the total training and validation time by up to 22.4% compared with the traditional schemes.

Key words : Multi-threads, deep learning, TensorFlow, GPU and CPU utilization

1. 서 론

딥 러닝(Deep learning)은 기계 학습(Machine learning)의 한 분야로 관련 기술이 다양한 분야에서 주목을 받고 있다[1,2,3]. 또한, 최근 컴퓨팅 하드웨어 기술이 발전함에 따라서, 고사양 컴퓨팅 환경 기반의 대규모 스케일 (large-scale) 모델에 대한 훈련(Training)이 새롭게 연구되고 있다[1,4,5]. 예를 들어, HyperCLOVA는 딥 러닝 기반의 자연어 처리를 위해 820 억개의 파라미터 정보를 학습(Learning) 및 추론(Inference)에 사용한다[1]. 일반적으로, 이러한 대규모 스케일 기반의 모델 훈련 및 검증(Validation)에서는 범용 CPU 와 함께 병렬 처리(Parallel Processing) 연산에 특화된 GPU 을 사용한다[2].

대부분의 딥 러닝 기반 소프트웨어 플랫폼(예: 텐서플로우, 파이토치)들 역시 모델의 훈련 및 검증을 위해 CPU 와 GPU 연산을 모두 사용한다[6,7]. CPU 는 연산에 필요한 기본 데이터(Raw Data)를 텐서(Tensor)로 변환하는 과정을 수행함으로써, GPU 의 연산이 효율적으로 처리될 수 있도록 돕는다. 그러나, 기본적인 모델 훈련 및 검증 과정에서의 동작은 CPU 레벨에서의 단일 쓰레드(Single-thread)로 수행되기 때문에 고성능의 GPU 을

* 이 논문은 2021 년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. NRF-2021R1I1A3047006).
† 학생회원 : 창원대학교 컴퓨터공학과 석사과정
jinseo@gs.cwnu.ac.kr
‡ 종신회원 : 창원대학교 컴퓨터공학과 조교수
donghyun@gs.cwnu.ac.kr

포함하고 있는 컴퓨팅 시스템에서 병목 현상(Bottleneck)을 야기한다[8,9]. 이러한 병목 현상의 문제점을 해결하기 위해 다양한 연구가 수행되었으며, tf.data 는 가장 대표적인 해결방안이다[10]. tf.data 는 Google 에서 제공하는 딥 러닝 라이브러리인 텐서플로우(TensorFlow)에 포함된 데이터 파이프라인을 멀티 쓰레드(Multi-thread) 기반으로 수행하기 위한 다양한 API 을 제공한다. 그러나, 실험을 통해 tf.data 에 대한 성능을 분석한 결과, 여전히 CPU 및 GPU 사용률(Utilization)과 전반적인 훈련 성능(Performance) 측면에서 개선이 필요하다는 사실을 확인할 수 있었다.

또한, 학습 모델의 훈련 성능을 개선하기 위해 병렬 처리(Parallel Processing)와 관련된 연구들이 많이 수행되었다. Downpour SGD 는 GPU 클러스터 환경에서 훈련 데이터와 학습 모델의 복사본을 각 GPU 에 분배하고 개별적으로 파이프라인을 구성하여 수행하는 기법이다. 갱신된 모든 모델의 가중치는 공유가중치 서버에 전송하여 가중치 평균값을 각 GPU 의 모델에 재분배하여 가중치를 반영한다. 이를 통해 모델 복사를 사용하지 않는 기존 방식보다 60% 빠른 수행 시간을 보였다[11]. Nimble 은 단일 GPU 환경에서 모델 학습에 필요한 연산을 여러 개의 GPU 스트림(Stream)을 생성하여 분산 처리하는 스케줄링 기법으로, 파이토치에서 기존방식 대비 처리속도가 향상되는 결과를 보여 주었다[12,13]. 그러나, 기존의 연구들은 모델 훈련 및 검증 과정에서 중첩 가능한 과정에 대해서는 제대로 연구하지 못하였다. 특히, 이 과정에서 낭비되는 CPU 와 GPU 의 사용률을 시스템 관점에서 분석하지 못하고 있다.

이에, 본 논문에서 우리는 기존 기법과는 다르게 고사양의 CPU 와 GPU 의 사용률을 향상시킴으로써, 전반적인 훈련 및 검증 성능을 향상시키는 새로운 기법을 제안한다. 제안하는 기법은 훈련 모델 복사(Training Model Copy) 과정을 통해서 GPU 에서의 훈련과 검증을 위한 단계를 중첩(Overlapping)시키고 훈련 및 검증 단계를 멀티 쓰레드 기반으로 수행함으로써, 효율적으로 성능을 개선한다. 실험을 통해 확인한 결과, 제안 기법은 CPU 및 GPU 사용률을 기존 기법 대비 상당히 향상시키며, 전체 훈련 및 검증 시간을 최대 22.4%만큼 단축시키는 것을 확인할 수 있었다.

마지막으로, 본 논문의 구성은 다음과 같다. 2 장에서는 딥러닝 모델의 학습과정 및 파이프라인의 동작 과정을 소개하며, 3 장에서는 제안 기법을 자세하게 설명한다. 4 장에서는 실험을 통해, 기존 방법과 제안 기법을 비교 및 평가한다. 마지막으로, 5 장에서는 본 논문의 결론을 맺는다.

2. 연구배경

2.1 훈련 및 검증 과정

딥 러닝은 입력 값과 가중치 값을 계산한 결과를 출력하는 기능을 가진 인공 뉴런(Neuron)들이 사람의 뇌 신경망 구조와 유사한 형태의 복잡한 네트워크를 형성한 심층 신경망(Deep Neural Network)을 사용하는 기계학습(Machine Learning)의 방법 중 하나이다[14]. 일반적으로 딥 러닝 모델은 훈련(Training) 과정과 검증(Validation) 과정으로 구분할 수 있으며, 빠른 병렬 처리(Parallel Processing)을 위해 GPU 에서 수행된다. 훈련 및 추론 과정에서 CPU 는 기본 데이터(Raw Data)를 일정 크기의 배치(Batch) 데이터로 가공하여 GPU 로 전달한다[15]. 일정 크기로 분할된 배치 데이터는 심층 신경망을 통해 추론 과정을 수행하며 전체 데이터가 모두 훈련 및 검증 과정을 완료할 때 딥 러닝에서는 1 에폭(Epoch)이 수행되었다고 말한다. 일반적으로 훈련 및 검증에서의 정확도(Accuracy)를 향상시키기 위해 에폭 과정은 여러 번 반복된다[16].

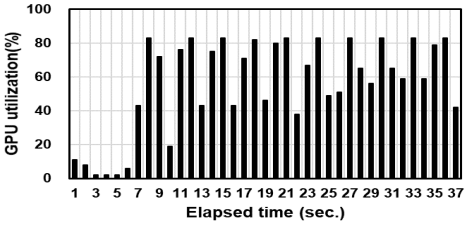


그림 1 tf.data 기반 모델의 GPU 사용률
Fig. 1 GPU utilization of the tf.data-based model

딥 러닝의 훈련 과정에서 데이터는 심층 신경망에 특성(Feature)으로 입력되고 이를 계산한 결과(즉, 추론 결과)와 입력된 정답(Label) 결과를 비교함으로써, 정확도(Accuracy)와 오차(Loss)를 계산한다. 또한, 오차 값을 줄이기 위해 오차 역 전파(Error backpropagation)를 수행함으로써, 심층 신경망 내부의 뉴런(Neuron)들의 가중치 값을 조정한다[14,15]. 반면, 검증 과정에서는 훈련 과정과 동일하게 입력 데이터를 처리하고 추론 결과를 계산하지만 오차 역 전파를 수행하지 않으며, 일반적으로 학습 데이터의 10% 정도의 양을 가진 별도의 데이터를 사용한다. 즉, 딥 러닝 모델의 전반적인 성능은 검증 과정 보다 9 배 많은 양의 데이터를 처리하고 오차 역 전파를 통해 더 많은 연산을 수행하는 훈련 과정의 수행 시간이 절대적인 영향을 미친다.

2.2 데이터 파이프라인

심층 신경망에서 뉴런들의 기본 입력은 텐서(Tensor) 이기 때문에 추론을 위해서는 기본 데이터를 실수(Float) 형태로 구성된 고차원 배열인 텐서(Tensor) 형태로 변경해야만 한다. 이러한 변환 과정을 우리는 데이터 전처리(Preprocessing) 과정이라고 부르며, 일반적으로 CPU 을 통해 데이터 전처리 과정이 진행된다. 데이터 전처리 과정은 다음과 같다[6].

(1) 원격 또는 로컬 저장소의 기본 데이터(Raw Data)을 메인 메모리로 로드(Load)한다.
(2) 메인 메모리로 로드 된 데이터를 텐서 형태로 변환한다.

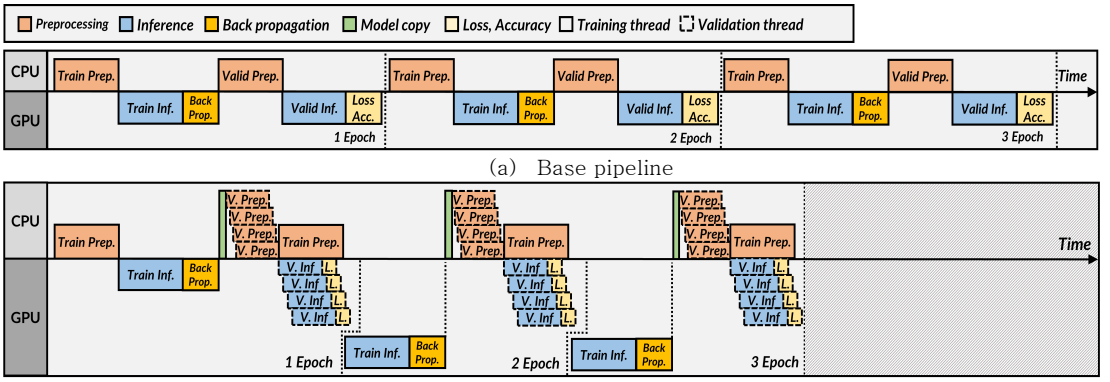


그림 2 기존 파이프라인과 제안기법의 동작과정 비교

Fig. 2 Comparison of processes between the base pipeline and proposed pipeline

(3) 학습 모델의 편향(Bias)을 감소시키기 위해 텐서 형태의 데이터를 무작위로 셔플(Shuffle)한다.

(4) 셔플된 텐서를 GPU 연산을 위한 배치(Batch) 크기로 분할한다.

전처리 과정 후, 셔플된 텐서는 배치 단위로 GPU 에 복사되어 훈련 및 검증이 진행되고 분할된 모든 배치에 대한 훈련 및 검증이 완료되면 에폭을 증가시킴으로써 다음 훈련 및 검증을 진행한다. 우리는 이러한 일련의 과정을 데이터 파이프라인(Data Pipeline)이라고 부른다[15]. 일반적으로, 이러한 데이터 파이프라인은 Numpy[17] 오픈소스 라이브러리를 활용하여 쉽게 구현이 가능하다. 그러나, Numpy 는 단일 쓰레드 기반의 데이터 파이프라인만을 지원하기 때문에 멀티 쓰레드 형태로 데이터 파이프라인을 구성하기 위해서는 케라스(Keras)와 tf.data 를 사용해야 한다[10,18]. 불행하게도, tf.data 역시 전처리 과정만을 멀티 쓰레드 형태로 수행하기 때문에 전반적인 모델의 훈련 및 검증 과정의 성능을 향상시키기 위해서는 한계를 가진다. 그림 1 은 tf.data 기반 딥 러닝 모델의 GPU 사용률을 보여주며, 본 실험에는 기본적인 MNIST 데이터가 사용되었다. 그림 1 에서 보여주듯이, 전체 훈련 및 검증을 수행하는 동안 GPU 의 사용률이 부분적으로 감소하며, 동일한 패턴이 반복되는 것을 확인할 수 있다. 이는, 고사양의 하드웨어(예: CPU, GPU, 등) 자원(Resource)을 낭비하는 심각한 결과를 초래하기 때문에 개선이 요구된다. 이에, 본 논문에서는 하드웨어 자원을 효율적으로 사용하기 위한 멀티 쓰레드 기반의 새로운 병렬 처리 기법을 제안한다.

3. 제안 기법

본 장에서는 본 논문이 제안하는 기법을 자세하게 설명하며, 제안 기법은 훈련 과정과 검증 과정을 중첩(Overlapping)시키기 위한 훈련 모델 복사(Model Copy) 과정과 CPU/GPU 사용률 개선을 위한 멀티 쓰레드(Multi-thread) 기반 검증 과정을 소개한다.

3.1 훈련 모델 복사 과정

앞에서 설명하였듯이, 기본적인 훈련 및 검증 과정은 중첩 없이 동작한다(즉, 훈련 완료 후 검증 과정 실행). 이는, 훈련 과정 중 오차 역 전파를 통해 추론을 위한 가중치 갱신(Update)이 빈번하게 발생하며, 검증 과정에서 서로 다른 가중치 값을 사용하여 추론을 진행하는 경우, Stale Gradient 문제가 발생하기 때문이다[19]. 그 결과, 일반적으로 최종 가중치 값을 계산하고 이를 기반으로 검증 과정을 진행한다.

우리는 이러한 단점을 개선하기 위해 훈련 쓰레드(Training-thread)와 검증 쓰레드(Validation-thread)로 훈련과 검증을 위한 쓰레드를 구분하였다. 그리고 훈련이 완료된 학습모델의 최종 가중치 값들을 검증 쓰레드로 복사하는 모델 복사(Model Copy) 과정을 설계하였다. 이는 CPU 에서 생성되어 실행되는 쓰레드로 훈련과 검증 과정을 간섭 없이 병렬적으로 수행한다. 훈련 쓰레드는 단일 쓰레드로 동작하고 검증 쓰레드는 쓰레드 풀에 의해 멀티 쓰레드로 동작한다. 그림 2 는 Numpy 기반 기존 데이터 파이프라인의 동작과 제안 기법에 대한 차이점을 보여준다. 그림 2 의 (a)에서 기존 Numpy 기반 훈련 및 검증 과정은 모든 과정이 직렬화(Serialization)되어 수행된다 (즉, 훈련 완료 후 검증 과정 실행). 반면, 그림 2 의 (b)에서 제안 기법은 훈련과정과 검증과정을 각 쓰레드로 구분하여 동작한다. 첫 번째 에폭에 대한 훈련 쓰레드가 완료되는 시점에서 쓰레드 풀은 모델 복사본을 통해 훈련 쓰레드와 검증 쓰레드를 중첩시킨다. 각 검증 쓰레드는 멀티 쓰레드 기반으로 데이터에 대한 전처리 과정을 수행한 후, 개별적으로 검증 과정을 수행한다. 이와 동시에 훈련 쓰레드는 유휴 CPU 리소스를 사용하여 다음 에폭 훈련에 대한 데이터 전처리를 수행한다. 훈련 쓰레드는 검증 쓰레드의 종료 유무와 관계없이 전처리 과정이 끝나면 다음 에폭 훈련을 수행한다. 검증 쓰레드는 자식 쓰레드의 검증 과정이 종료되면 다음 에폭에 대한 훈련 쓰레드가 종료될 때까지 대기한다. 즉, 훈련 쓰레드와 검증 쓰레드가 동시에 중첩(Overlapping)

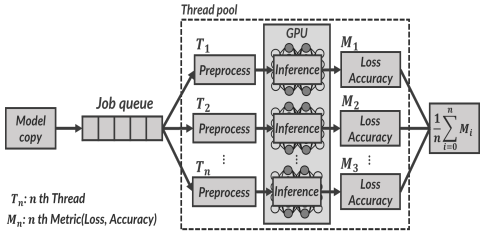


그림 3 제안 기법의 검증 과정
Fig. 3 Validation process of the proposed scheme

되어 수행된다. 모든 에폭에서의 훈련 과정은 *"전처리 (데이터 로드 → 텐서 변환 → 셔플 → 배치), 추론을 위한 연산, 그리고 오차 역 전파 과정"*을 반복적으로 수행해야하기 때문에 훈련 과정이 빠르게 재 시작되는 것은 CPU 와 GPU 의 유휴 시간(Idle Time)을 모두 감소시키고 사용률을 향상시킬 수 있다.

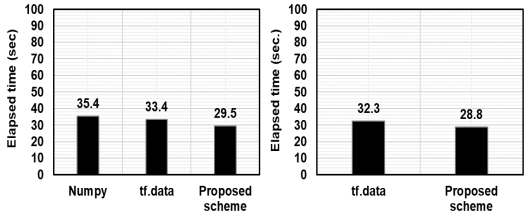
3.2 멀티 쓰레드 기반 검증 과정

검증 쓰레드는 모델 복사 과정을 통해 훈련 쓰레드와 독립적으로 동작할 수 있다. 그러나, 검증 쓰레드 역시 단일 쓰레드 기반으로 수행하는 경우, 전처리와 추론을 위한 연산 대한 일련의 과정을 직렬화하여 수행하게 된다. 우리는 이러한 직렬화 과정을 병렬적으로 처리하기 위해 멀티 쓰레드 기반 검증 과정을 추가적으로 설계하였다. 제안 기법에서는 훈련 쓰레드가 완료된 후 모델 복사가 수행될 때, 쓰레드 풀에서 사용자가 사전에 설정한 개수만큼 검증을 위한 쓰레드를 생성한다. 또한, 검증 과정에 필요한 데이터의 개수를 미리 확인하여 생성된 쓰레드의 개수에 맞게 동일한 크기로 검증을 위한 데이터를 분할한다. 이는 각각의 검증 쓰레드가 서로 다른 영역의 데이터에 접근하기 위함이다.

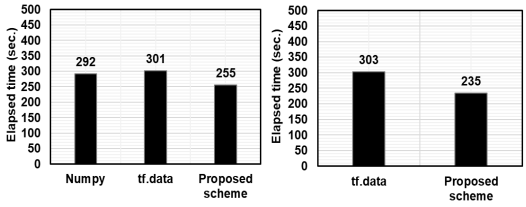
그림 3 은 제안 기법의 검증 과정을 보여준다. 그림에서 보여주듯이, 제안 기법에서는 추론 데이터에 대한 전처리 과정(텐서 변환 → 배치)과 추론을 위한 연산 과정이 병렬적으로 수행되는 것을 확인할 수 있다. 또한, 제안 기법에서는 모델 복사를 통해 모든 검증 쓰레드가 동일한 가중치 값으로 추론 연산을 수행하기 때문에 Stale Gradient 문제도 발생하지 않는다. 마지막으로, 검증 쓰레드에서 추론 연산 후 계산된 오차(Loss) 및 정확도(Accuracy)는 평균 값으로 계산하여 해당 에폭에 대한 정확도와 오차 값으로 반영한다. 본 과정을 정리하면, 제안 기법은 멀티 쓰레드 기반으로 검증 과정을 수행함으로써, 추론을 위해 필요한 일련의 데이터 전처리 과정을 CPU 가 병렬적으로 처리하도록 한다. 그리고 짧은 대기 시간과 함께 각 검증 쓰레드의 추론 연산을 수행할 수 있도록 GPU 로 전달한다. 그 결과 CPU 의 사용률을 향상시킬 수 있으며, GPU 가 전처리 과정으로 대기하는 시간을 효율적으로 감소시킬 수 있다.

4. 구현 및 실험

본 장에서는 제안한 기법의 구현 방법과 실험 및 평가 방법에 대해서 자세하게 설명한다.



(a) 1 Thread (b) 6 Thread
그림 4 MNIST 파이프라인 훈련성능 비교
Fig. 4 Comparison of training performance using the MNIST dataset



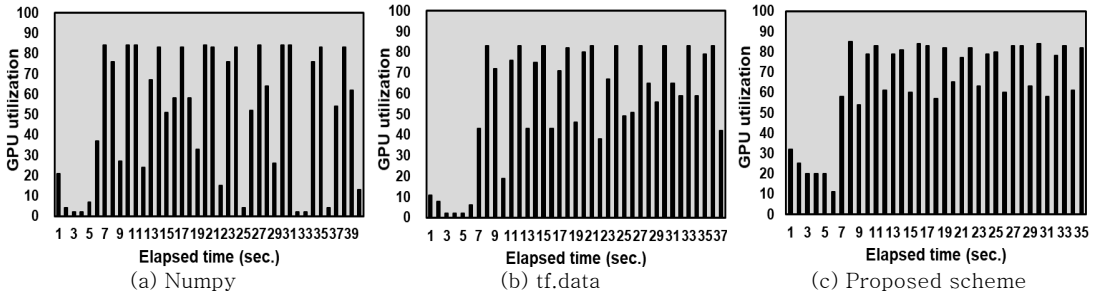
(a) 1 Thread (b) 6 Thread
그림 5 1GB MNIST 파이프라인 훈련성능 비교
Fig. 5 Comparison of training performance using the 1GB MNIST dataset

4.1 구현

본 제안 기법은 텐서플로우 커널 또는 케라스 라이브러리의 기존 코드를 수정하지 않고 사용자 수준에서 Python3.8 버전으로 구현하였다. 훈련 쓰레드와 검증 쓰레드는 Python 에서 제공하는 Threading 라이브러리를 사용하였으며 훈련 쓰레드는 CPU 에서 훈련데이터 전처리를 수행하는 동시에 검증 쓰레드는 멀티 쓰레딩을 통해 개별 쓰레드의 작업을 텐서플로우 커널으로 전달한다. 텐서플로우 커널은 요청받은 작업을 여러 개의 GPU 스트림에 할당하여 검증과정을 수행한다[12]. CPU 에서 동작중인 각 쓰레드가 동시에 작업을 요청함에 따라, 이 시점에서 CPU 활용률이 증가하고 검증 과정에서 여러 개의 모델이 병렬적으로 GPU 리소스를 활용함으로써 GPU 활용률 또한 증가하여 전체 수행시간이 단축되는 효과를 보인다.

4.2 실험 환경

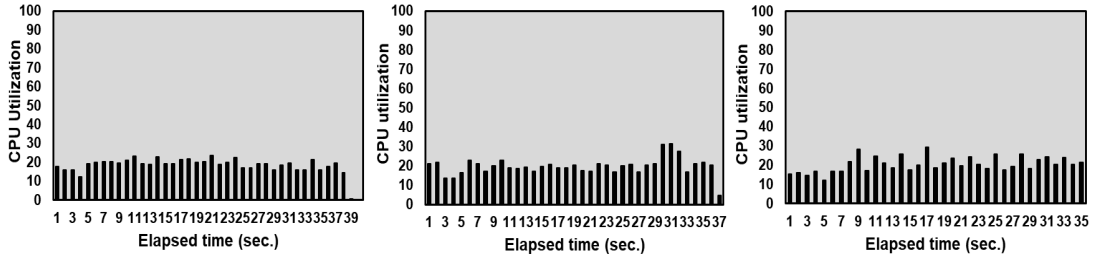
실험을 위해 우리는 Intel Core i5-9500 (3.00GHz, 6 Core) CPU, 32GB DRAM, Nvidia Geforce GTX 1650 GPU (4GB)가 포함된 컴퓨터에서 실험을 진행하였으며, 운영체제는 리눅스 커널 4.15 버전의 Ubuntu 16.04 LTS 를 사용하였다. 또한, 우리는 제안 기법의 평가를 위해 텐서플로우(버전 2.5.0)와 케라스(버전 2.5.0)을 사용하여 이미지 분류 모델에서 가장 많이 사용되는 합성곱 신경망(CNN: Convolutional Neural Network) 모델을 구현하였다. 훈련 및 검증 과정에 필요한 데이터는 가장 잘 알려진 MNIST[20] 데이터 셋(Data Set)을 사용하였다. 또한, 기본 데이터 셋의 데이터를 복사함으로써, 데



(a) Numpy (b) tf.data (c) Proposed scheme

그림 6 MNIST 단일 쓰레드 파이프라인 GPU 사용률 비교

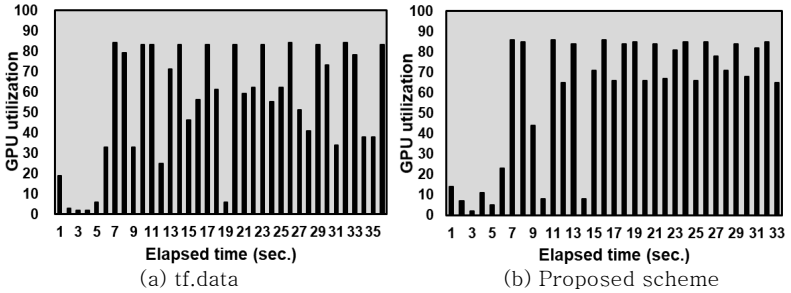
Fig. 6 Comparison of MNIST GPU utilization using the single thread pipeline



(a) Numpy (b) tf.data (c) Proposed scheme

그림 7 1 MNIST 단일 쓰레드 파이프라인 CPU 사용률 비교

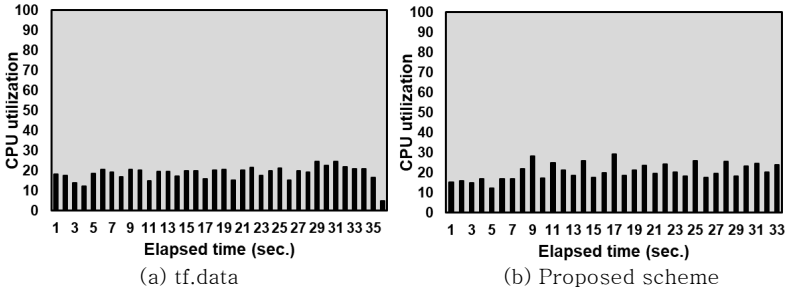
Fig. 7 Comparison of MNIST CPU utilization using a single thread



(a) tf.data (b) Proposed scheme

그림 8 MNIST 멀티 쓰레드 파이프라인 GPU 사용률 비교

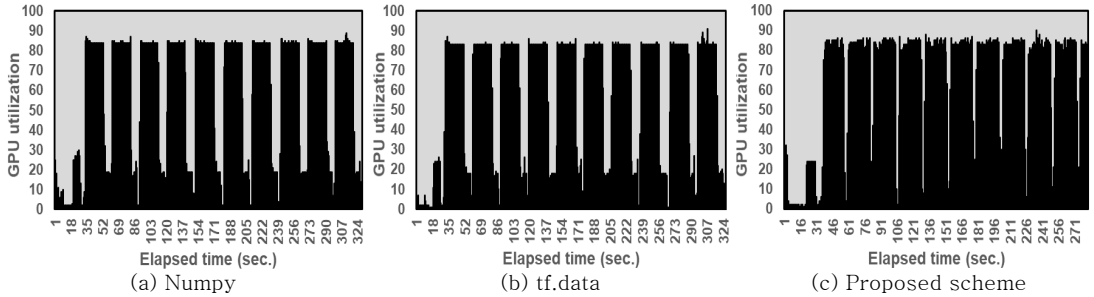
Fig. 8 Comparison of MNIST GPU utilization using the multi-thread pipeline



(a) tf.data (b) Proposed scheme

그림 9 MNIST 멀티 쓰레드 파이프라인 CPU 사용률 비교

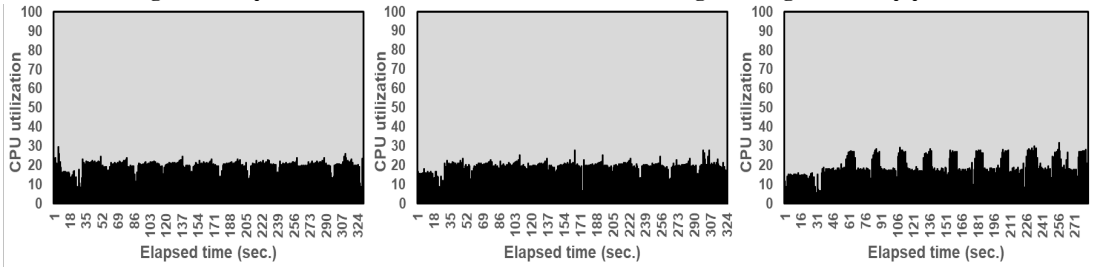
Fig. 9 MNIST Comparison of MNIST CPU utilization using the multi-thread pipeline



(a) Numpy (b) tf.data (c) Proposed scheme

그림 10 1GB MNIST 단일 쓰레드 파이프라인 GPU 사용률 비교

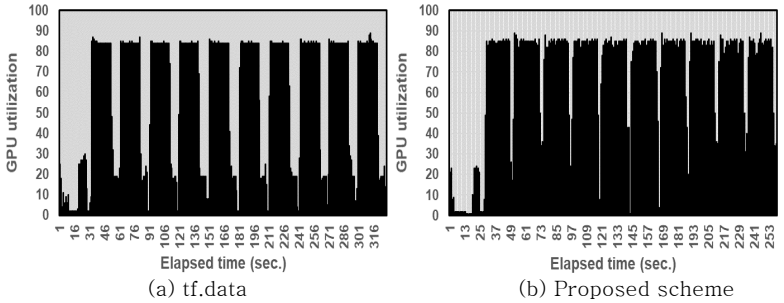
Fig. 10 Comparison of 1GB MNIST GPU utilization using the single thread pipeline



(a) Numpy (b) tf.data (c) Proposed scheme

그림 11 1GB MNIST 단일 쓰레드 파이프라인 CPU 사용률 비교

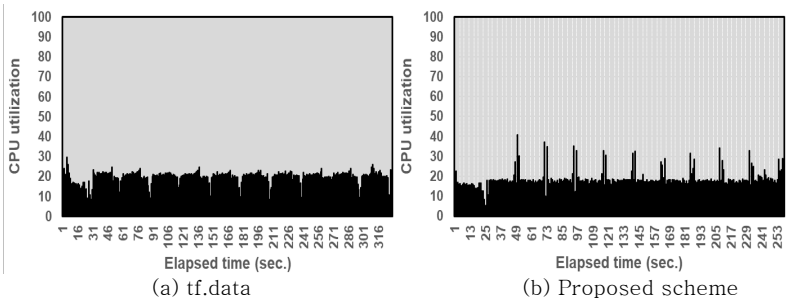
Fig. 11 1GB Comparison of 1GB MNIST CPU utilization using the single thread pipeline



(a) tf.data (b) Proposed scheme

그림 12 1GB MNIST 멀티 쓰레드 파이프라인 GPU 사용률 비교

Fig. 12 Comparison of 1GB MNIST GPU utilization using the multi-thread pipeline



(a) tf.data (b) Proposed scheme

그림 13 1GB MNIST 멀티 쓰레드 파이프라인 CPU 사용률 비교

Fig. 13 Comparison of 1GB MNIST CPU utilization using the multi-thread pipeline

이터 셋의 크기를 1GB 확장하고 이에 대한 평가도 함께 진행하였다. 마지막으로, 전체 에폭은 10 으로 설정하였으며, 모든 실험은 3 번씩 동일한 방식으로 수행하고 평균 값을 최종적으로 표시하였다. 또한, 평가가 진행되는 중 CPU 와 GPU 사용률을 측정하기 위해 리눅스에서 제공하는 sysstat[21], gpustat[22]을 사용하였다.

4.3 전체 훈련 및 검증 시간 평가

우리는 제안 기법의 성능 특성을 확인하기 위해, 전체 훈련 및 검증 시간에 대한 시간을 측정하였다. 또한, 제안 기법과 tf.data 의 경우, 훈련 및 검증 단계에서 사용할 쓰레드의 개수를 지정할 수 있기 때문에 쓰레드의 개수를 1(즉, 단일 쓰레드)과 6(즉, 멀티 쓰레드)으로 변경하며 실험을 진행하였다. 그림 4 와 그림 5 는 기본 MNIST 데이터 셋과 1GB 로 확장한 MNIST 데이터 셋의 평가 결과를 보여준다.

그림 4 의 (a)는 본 논문에서 tf.data 와 제안 기법의 쓰레드의 개수가 1 개일 때, 훈련 및 검증에 대한 전체 시간을 비교한 결과로, Numpy 와 tf.data 파이프라인의 훈련 시간보다 각각 16.6%, 11.6% 감소하는 결과를 보였다. 이는, 훈련 모델 복사를 통해 훈련 쓰레드와 검증 쓰레드가 중첩됨으로써 얻을 수 있는 향상으로 보인다. 한편, 그림 4 의 (b)는 쓰레드의 개수를 6 개로 설정했을 때 훈련 및 검증 시간에 대한 비교를 보여준다. 쓰레드 개수가 6 으로 설정되는 경우, tf.data 는 데이터를 배치 단위로 분할하는 과정에서 제안 기법은 검증 쓰레드 과정에서 병렬적으로 쓰레드가 동작한다. 그림에서 확인할 수 있듯이, 제안 기법이 배치 단위의 병렬 처리를 지원하는 tf.data 에 비해 10.8% 전체 훈련 및 검증 시간을 감소시키는 것을 확인할 수 있다. 이는 그림 4 의 (a)의 결과에서 보여준 훈련 및 검증 쓰레드의 중첩과 검증 과정에서의 멀티 쓰레드 동작이 전반적인 훈련 및 검증 시간을 단축시킬 수 있다는 사실을 명확하게 보여준다.

그림 5 는 1GB 크기로 MNIST 데이터 셋을 확장한 데이터에 대한 실험 결과를 보여준다. 그림 5 에서 보여주듯이, 전반적인 훈련 및 검증 시간에 대한 성능 패턴은 그림 4 와 동일하다는 것을 확인할 수 있다. 데이터 셋을 확장하는 경우, 기본 데이터 셋보다 더 많은 데이터에 대한 전처리 과정이 필요하다. 그 결과, 그림 5 의 (b)에서 보여주듯이(즉, 6 개의 쓰레드), tf.data 와 제안 기법의 성능 차이가 최대 22.4% 확대되는 것을 확인할 수 있다. 이러한 결과는 제안 기법이 멀티 쓰레드 기반의 검증 쓰레드가 더 많은 I/O 처리를 진행할 때 효과적이라는 사실을 분명하게 보여준다.

4.4 GPU 및 CPU 사용률 평가

제안 기법의 전반적인 훈련 및 검증 시간 감소에 대한 성능 이점을 더 명확하게 확인하기 위해, 우리는 제안 기법의 GPU 와 CPU 사용률에 대한 평가를 진행하였다. 그림 6, 그림 7 은 기본 MINST 데이터 셋을 입력으로 단일 쓰레드(즉, 쓰레드 개수 1) 환경에서의 측정 결과를 보여주며, 그림 8 과 그림 9 는 쓰레드의 개수가 6 인 경우 측정 결과를 각각 보여준다. 평가를 위한 사용률 측정은

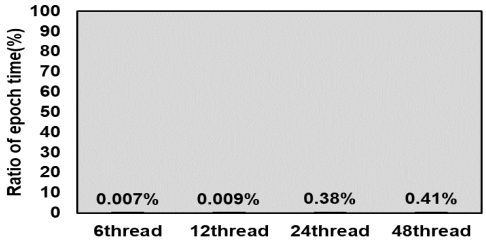


그림 14 에폭 훈련시간에 대한 모델 복사 오버헤드 비율

Fig. 14 The ratio of model copy overhead to epoch training time.

mpstat, gpustat 명령어를 사용하였으며, 1 초 간격으로 진행하였다. 또한, 그림 10, 그림 11, 그림 12, 그림 13 은 동일한 실험을 1GB MNIST 데이터 셋에 대한 입력으로 수행한 결과를 보여준다.

위의 실험 결과에서 보여주듯이, GPU 의 경우, 사용률이 높은 구간과 낮은 구간이 존재하며 동일한 패턴이 반복되는 것을 확인할 수 있다. 이는 훈련 과정 중 오차 역전파를 통해 가중치 갱신이 일어나는 구간, 검증 구간, 그리고 CPU 로 부터 다음 학습을 기다리는 구간으로 설명할 수 있다. 또한, 실험 결과에서 확인할 수 있듯이, 제안 기법의 GPU 사용률이 기존 Numpy 와 tf.data 에 비해 더 많은 양의 GPU 을 사용하고 있다는 것을 확인할 수 있다. 즉, 반복되는 패턴에서 GPU 사용률이 감소하는 구간이 기존 기법에 비해 현저히 낮다. 이는, 단일 쓰레드 실험에서는 모델 복사를 통한 훈련 및 검증 쓰레드의 중첩의 효과이며, 검증 단계에서 멀티 쓰레드를 사용함으로써 얻을 수 있는 성능 이득이라고 해석된다. 반면, tf.data 의 경우, 쓰레드 개수 가 증가함에도 불구하고 GPU 의 사용률 변화가 거의 나타나지 않는다. 이는 tf.data 가 배치 과정에서 수행하는 병렬 처리가 내부적인 병목 현상으로 인해 제한된다는 사실을 보여준다. 한편, 제안 기법의 경우, CPU 사용률 역시 다른 기법에 비해 개선되는 것을 확인할 수 있다. 이는, 앞에서 설명하였듯이, 제안 기법의 모델 복사를 통해 CPU 의 데이터 전처리 과정이 다른 기법 대비 빠르게 동작 및 반복되기 때문이다.

4.5 모델복사 Overhead 평가

마지막으로, 우리는 제안 기법에서 검증 쓰레드 실행 전 모델 복사가 훈련시간에 미치는 영향을 확인하기 위해 전체 훈련시간 중 1 에폭 수행시간에서 모델복사에 대한 비율을 측정하였다. 그림 14 는 합성곱 신경망(CNN) 모델과 1GB MNIST 데이터셋을 사용하여 검증 쓰레드 개수는 6 개에서 48 개로 설정했을 때 측정한 결과를 보여준다. 즉, 검증 쓰레드가 48 개이면 모델 객체복사를 48 번 수행한다. 그림 14 에서 보여주듯이, 검증 쓰레드가 6 개일 때, 1 에폭 훈련시간에 대한 모델복사 수행시간에 대한 비율이 0.01%만큼 차지하고 검증 쓰레드의 개수를 48 개로 확장했을 때, 0.41%만큼만 차지하는 것을 확인할 수 있다.

위의 실험결과에서 보여주듯이 검증 쓰레드 개수를 시스템 CPU 코어 개수보다 8 배 많은 48 개까지 확장하여도 1 에폭 훈련시간에서 차지하는 비율이 0.5% 이하로, 전체 훈련시간에 대해 미치는 영향이 작다고 할 수 있다. 향후, VGG16, ResNet50 과 같은 대규모 학습모델과 대용량 데이터셋을 사용하여 제안기법의 확장성에 대한 연구를 진행하고자 한다.

5. 결론

딥 러닝에서의 학습 및 추론은 다양한 분야에서 연구 및 사용되고 있으나 시스템 관점에서 GPU 및 GPU 사용률에 대한 분석은 아직 연구가 부족하다. 이에, 본 논문에서는 하드웨어 리소스를 최대한 효율적으로 사용하기 위한 기법을 제안하였다. 제안 기법은 멀티 쓰레드 기반으로 학습 및 검증 과정을 분리시켜 중첩하였다. 또한, 검증 과정에서의 사용률 최대화를 위해 멀티 쓰레드를 사용하였다. 우리는 제안 기법의 특성을 이해하기 위해, 기존 기법과 제안 기법을 실험 및 평가하였으며, 실험 결과 제안 기법이 최대 22.4% 전체 학습 및 검증 시간을 단축시킨다는 사실을 확인할 수 있었다. 이는 어플리케이션 단계에서의 수정으로 하드웨어 자원 사용률을 향상시킬 수 있다는 점에서 상당한 의미가 있다고 생각한다. 이에, 향후 연구로는 더 많은 실험 데이터 셋을 기반으로 제안 기법에 대한 실험 및 평가를 진행하고자 한다.

Reference

[1] B. Kim, H. Kim, S. W. Lee, G. Lee, D. Kwak, D. H. Jeon, S. Park, S. Kim, D. Seo, H. Lee, M. Jeong, S. Lee, M. Kim, S. H. Ko, S. Kim, T. Park, J. Kim, S. Kang, N. H. Ryu, K. M. Yoo, M. Chang, S. Seo, S. In, J. Park, K. Kim, H. Kim, J. Jeong, Y. G. Yeo, D. H, D. Park, M. Y. Lee, J. Kang, I. Kang, J. W. Ha, W. Park, N. Sung, “What changes can large-scale language models bring? intensive study on hyperclova: Billions-scale korean generative pretrained transformers,” arXiv preprint arXiv:2109.04650, 2021.

[2] Y. H. Yoo, H. J. Jeong, M. S. Mook, “Analyzing the Execution Time of Different DNN Hidden Layer Types During Inference Under GPU Contention,” *Journal of KIISE*, Vol. 26, No. 2, pp. 463-468, 2020. (in Korean)

[3] M. Cho, D. Kang, “L2LRU: Learning-based Page Movement Policy for LRU Page Replacement Policy,” *Journal of KIISE*, Vol. 48, No. 9, pp. 981-987, 2021. (in Korean)

[4] M. A. Zinkevich, M. Weimer, A. Smola, L. Li, “Parallelized stochastic gradient descent,” *Proc. of the International Conference on Neural Information Processing Systems*, Vol. 4, No. 1, pp. 2595-2603, 2010.

[5] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, Q. Le, A. Ng, “Large scale distributed deep networks,” *Proc. of the Advances in neural information processing systems*, Vol. 25, pp. 1223-1231, 2012.

[6] M. Abadi, P Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat. G. Iriving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, “TensorFlow: A System for Large-Scale Machine Learning,” *Proc. of the USENIX symposium on operating systems design and implementation*, pp. 265-283, 2016.

[7] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Demaison, A. Kopf, E. Yangm Z. Devito, M. Rasion, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala, "Pytorch: An imperative style, high-performance d eep learning library," *Proc. of the Advances in neur al information processing systems*32, Vol. 32, pp. 8 026-8037, 2019.

[8] M. Kuchink, A. Klimovic, J. Simsa, G. Amvrosiadis, V. Simith, “Plumber: Diagnosing and Removing Performance Bottlenecks in Machine Learning Data Pipelines,” arXiv preprint arXiv:2111.04131, 2021.

[9] J. Mohan, A. Phanishayee, A. Raniwala, V. Chidambaram, “Analyzing and Mitigating Data Stalls in DNN Training,” arXiv preprint arXiv:2007.06775, 2020.

[10] D. G. Murray, J. Simsa, A. Klimovic, and I. Indyk, “tf.data: A Machine Learning Data Processing Framework,” arXiv preprint arXiv:2101.12127, 2021.

[11] 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, A. T. Ng, “Large Scale Distributed Deep Networks,” *Proc. of the International Conference on Neural Information Processing Systems,* Vol. 1, pp. 1223–1231, 2012

[12] CUDA C++ programming guide, [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/ (accessed 2021, November 19)

[13] W. Kwon, G. I. Yu, E. Jeong, B. G. Chun, "Nimble: Lightweight and parallel gpu task scheduling for deep learning." arXiv preprint arXiv:2012.02732 2020.

[14] Y. LeCun, Y. Bengio, G. Hinton, “Deep learning,” Nature, Vol. 521, pp. 436-444, 2015.

[15] S. Serebryakov, D. Milojicic, N. Vassilieva, S. Fleischman, R. D. Clark, "Deep Learning Cookbook: Recipes for your AI Infrastructure and Applications," *Proc. of the International Conference on Rebooting Computing*, pp. 1-9, 2019.

[16] Y. Xu, Q, Qian, H. Li, R. Jin, “Why Does Multi-Epoch Training Help?,” arXiv preprint arXiv:2105.06015 2021.
[17] C. R. Harris, et al. “Array programming with NumPy,”*Nature,* Vol. 585, pp. 357–362, 2020.
[18] Keras, [Online]. Available: https://keras.io/api/, (accessed 2021, November 19)
[19] S. Dutta, G. Joshi, S. Ghosh, P. Dube, P, Nagpurka, “Slow and Stale Gradients Can Win the Race: Error-Runtime Trade-offs in Distributed SGD,” *Proc. of the International Conference on Artificial Intelligence and Statistics*, Vol. 84, pp. 803-812, 2018.
[20] L. Deng, “The mnist database of handwritten digit images for machine learning research,” *Proc. of the IEEE*, Vol. 29, No 6, pp. 141–142, 2012.
[21] sysstat, [Online]. Available: https://github.com/sysstat/sysstat, (downloaded 2021, November 19)
[22] gpustat, [Online]. Available: https://github.com/wookayin/gpustat, (downloaded 2021, November 19)

최 진 서
2022년 창원대학교 컴퓨터공학과 학사. 2022년~현재 창원대하교 컴퓨터공학과 석사과정. 관심분야는 운영체제, 스토리지 시스템, 기계학습

강 동 현
2000 년 한국산업기술대학교 컴퓨터공학과 학사. 2008 년 성균관대학교 전자전기컴퓨터공학과 석사. 2018 년 성균관대학교 전자전기컴퓨터공학과 박사. 2018년~2019 년 삼성전자 책임연구원. 2019년~2020 년 동국대학교(경주) 조교수. 2020 년~현재 창원대학교 조교수. 관심 분야는 스토리지 시스템, 시스템 소프트웨어, 운영체제