

BigDL 2.0: 노트북에서 분산 클러스터로 AI 파이프라인의 원활한 확장

Jason (Jinquan) Dai, Ding Ding, Dongjie Shi, Shengsheng Huang, Jiao Wang,
Xin Qiu, Kai Huang, Guoqiong Song, Yang Wang, Qiyuan Gong, Jiaming Song,
Shan Yu, Le Zheng, 첸이나, 덩 준웨이, 송 거

인텔

추상적인

대부분의 AI 프로젝트는 단일 노트북에서 실행되는 Python 노트북으로 시작합니다. 그러나 더 큰 데이터 세트를 처리하기 위해 확장하려면 일반적으로 산더미 같은 고통을 겪어야 합니다(실험 및 프로덕션 배포 모두). 여기에는 일반적으로 데이터 과학자가 사용 가능한 하드웨어 리소스(예: SIMD 명령, 다중 처리, 양자화, 메모리 할당 최적화, 데이터 분할, 분산 컴퓨팅 등)를 완전히 활용하기 위해 많은 수동 및 오류가 발생하기 쉬운 단계가 수반됩니다.

이 문제를 해결하기 위해 Apache 2.0 라이선스에 따라 <https://github.com/intel-analytics/BigDL/>에서 BigDL 2.0을 오픈 소스로 제공했습니다(원래 BigDL [19] 및 Analytics Zoo [18] 프로젝트 결합). BigDL 2.0을 사용하여 사용자는 노트북에서 기존 Python 노트북을 간단히 구축할 수 있으며(AutoML 지원 가능) 단일 노드에서 투명하게 가속할 수 있습니다(실험에서 최대 9.6배 속도 향상). 대규모 클러스터(실제 사용 사례에서 수백 대의 서버에 걸쳐). BigDL 2.0은 이미 많은 실제 사용자(Master card, Burger King, Inspur 등)가 생산에 채택했습니다.

1. 소개

엔드 투 엔드 데이터 분석 파이프라인에 AI 모델을 적용하는 것은 오늘날의 대규모 지능형 애플리케이션에서 중요한 역할을 합니다. 반면에 AI 프로젝트는 일반적으로 단일 노트북이나 워크스테이션에서 실행되는 Python 노트북으로 시작하며 고성능으로 더 큰 데이터 세트를 처리하기 위해 확장하려면 산더미 같은 고통을 겪어야 합니다(대규모 실험 및 생산 배포). 이를 위해 종종 데이터 과학자는 오류가 발생하기 쉬운 여러 단계를 수동으로 수행하고 코드 변경을 통해 가용 하드웨어 리소스(예: SIMD 명령 [26], 다중 처리 [31])를 완전히 활용해야 합니다. [27], 양자화 [25], 메모리 할 위치 최적화 [9], 데이터 분할, 분산

컴퓨팅 [39] [34] [20] [28] 등).

이러한 문제를 해결하기 위해 <https://github.com/intel-analytics/BigDL/>에서 BigDL 2.0 소스를 오픈했습니다. [com/intel - Apache 2.0 라이선스에 따른 analytics/BigDL/](https://github.com/intel-analytics/BigDL/)(원래 BigDL [19] 및 Analytics Zoo [18] 프로젝트 결합)로 사용자가 단일 노드에서 투명하게 가속화되는 중단 간 AI 파이프라인을 구축할 수 있습니다. (실험에서 최대 9.6배의 속도 향상) 큰 클러스터로 원활하게 확장됩니다(실제 사용 사례에서 수백 개의 노드에 걸쳐). 데이터 과학자가 대규모 분산 AI 애플리케이션을 쉽게 구축할 수 있도록 다음과 같은 설계 원칙을 채택했습니다.

- 표준 API. BigDL 2.0을 사용하면 사용자는 표준 API(예: Tensorflow [10] 또는 PyTorch [32])를 사용하여 랩톱에서 기존 Python 노트북을 간단하게 구축할 수 있습니다. 모든 조정, 가속 및 확장은 기본 툴킷에 의해 자동으로 처리됩니다.
- 중단 간 파이프라인. 툴킷은 전체 AI 파이프라인(데이터 전처리, 기능 변환, 초매개변수 튜닝 [37], 모델 훈련 및 추론, 모델 최적화 [17] [25] 및 배포 등)을 최적화하는 전체적인 접근 방식을 취해야 합니다.
- 투명한 가속. 이 툴킷은 최적화된 라이브러리, 가장 잘 알려진 구성 및 소프트웨어 최적화를 자동으로 통합하여 사용자가 교육 또는 추론을 위한 AI 파이프라인을 투명하게 가속화하도록 도와야 합니다.
- 원활한 확장. 툴킷은 데이터 과학자를 위한 간단하고 친숙한 API를 사용하여 중단 간 AI 파이프라인(분산 데이터 병렬 처리, 모델 교육, 조정 및 추론 포함)을 원활하게 확장해야 합니다.

이 논문의 나머지 부분은 다음과 같이 구성되어 있다. 비서. 2는 BigDL 2.0의 전체 설계를 나타냅니다. 비서. 3 과 초. 4 는 아키텍처, 기능 및

각각 투명한 가속 및 원활한 확장을 위한 API. 마지막으로 초. 5는 BigDL 2.0 및 Sec의 일부 실제 사용 사례를 공유합니다. 6 논문을 마칩니다.

2. 빅DL 2.0

초에 설명된 대로. 1, BigDL 2.0은 원래 BigDL [19] 과 Analytics Zoo [18] 프로젝트를 결합하고 엔드 투 엔드 AI 파이프라인을 투명하게 가속화하고 원활하게 확장합니다. 이러한 목표는 BigDL 2.0의 두 라이브러리, 즉 BigDL-Nano 및 BigDL-Orca를 통해 달성됩니다.

- BigDL-Nano. SIMD 명령 [26], 다중 처리 [27] [31], 양자화 [6], 메모리 할당 최적화 [9] [21], 모델 최적화 [5] 등과 같은 여러 최적화 기술을 활용하여 AI 파이프라인을 개발할 때 솔루션에 걸리는 시간을 크게 줄이는 최대 10배의 속도 향상을 관찰했습니다. 그러나 이러한 기술을 적용하려면 다양한 도구를 사용하고, 복잡한 단계를 따르고, 방해가 되는 코드를 변경하고, 데이터 과학자에게 복잡하고 오류가 발생하기 쉽고 유지 관리하기 어려운 많은 구성을 조정해야 합니다. 이 문제를 해결하기 위해 이러한 최적화를 BigDL Nano에 통합하여 사용자가 로컬 랩톱 또는 단일 서버에서 딥 러닝 파이프라인(가능한 AutoML [11] [29] 사용)을 투명하게 가속화할 수 있습니다.

- BigDL-오르카. 로컬 랩톱에서 분산 클러스터로 AI 애플리케이션을 확장할 때 실제로 핵심 과제는 분산 데이터 처리 및 AI 프로그램을 단일 통합 파이프라인으로 원활하게 통합하는 방법입니다. BigDL-Orca는 분산 실행을 위해 Big Data 및 AI 시스템(예: Apache Spark [38] [2] 및 Ray [31] [8])을 자동으로 프로비저닝합니다. 기본 시스템 위에 분산된 메모리 내 데이터 파이프라인(Spark Dataframes [12], Ray Datasets [8], TensorFlow Dataset [10], PyTorch DataLoader [32] 및 임의의 python 라이브러리), 딥 러닝(예: TensorFlow 및 PyTorch) 교육 및 분산 데이터 세트에 대한 추론을 투명하게 확장합니다(scikit-learn 스타일 API[15]를 통해).

3. BigDL-Nano를 통한 투명한 가속

그림 1은 BigDL-Nano의 아키텍처를 보여줍니다. 수십 가지 가속 기술 및 도구(하드웨어 특정 구성, SIMD 명령 [26], 다중 처리 [31] [27], 메모리 할당 최적화 [21] [9], 그래프 최적화 및 양자화 [5] 등)를 활용합니다. 4) 백엔드에서 모델 교육 및 추론 파이프라인을 모두 투명하게 가속화합니다. 각 가속 기술 및 라이브러리에 대해 BigDL-Nano가 적응적으로 적용됩니다.

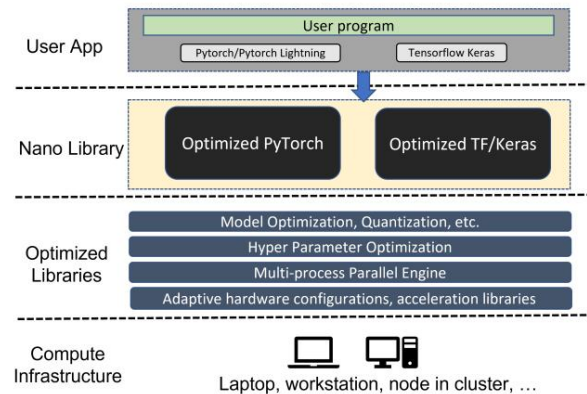


그림 1. BigDL-Nano 아키텍처

사용자의 실행 환경, 데이터 세트 및 모델을 기반으로 적절한 구성. BigDL-Nano는 이러한 모든 가속화를 사용자에게 투명하게 제공하여 데이터 과학자가 수동으로 다양한 구성을 조정하거나 다른 도구를 적용하거나 방해가 되는 코드 변경을 수행하지 않아도 되도록 합니다.

3.1. 교육 파이프라인 가속화

종단 간 교육 파이프라인을 가속화하기 위해 BigDL Nano는 사용자의 원래 Tensorflow 또는 PyTorch 프로그램에서 최소한의 변경이 필요한 투명한 API를 제공합니다. 예를 들어, PyTorch Lightning [22] 사용자의 경우 일반적으로 그림 2와 같이 BigDL-Nano를 사용하기 위해 라이브러리 가져오기를 변경하기만 하면 됩니다. 내부적으로는 일련의 교육 최적화(예: ISA 벡터화 [26], 개선된 메모리 할당 [21] [9], 다중 처리 [27] [31], PyTorch용 인텔 확장의 최적화 [5] 등)가 자동으로 활성화되어 그림과 같이 최대 5.8배의 속도 향상을 가져옵니다. 초. 3.4.

3.2. 추론 파이프라인 가속화

BigDL-Nano는 추론 파이프라인(예: 모델 최적화 및 양자화)을 가속화하기 위한 일련의 경량 API도 제공합니다. 그림 3은 BigDL-Nano를 사용하여 추론 단계에서 양자화 및 ONNX 런타임 [7]을 활성화하는 방법의 예를 보여줍니다. 다양한 최적화 도구(ONNX Runtime, INC [6], OpenVINO [4] 등 포함)를 자동으로 통합하여 Sec. 3.4.

3.3. AutoML

모델 개발 생산성을 최적화하기 위해 BigDL-Nano는 하이퍼파라미터 검색을 통해 내장된 AutoML [37] 자원도 제공합니다. 그림 4에서 볼 수 있듯이 사용자 프로그램에서 가져오기만 변경하면 BigDL-Nano는 검색 공간을 수집하여 기본 HPO 엔진 [13] [11] [29]에 전달하고 인스턴스화를 지연시킵니다.

```
bigdl.nano.pytorch.vision.datasets 가져오기 에서
→ 이미지폴더
bigdl.nano.pytorch.vision 가져오기 변환 에서
bigdl.nano.pytorch 가져오기 트레이너에서
torch.utils.data 에서 가져오기 DataLoader

data_transform = 변환.작성([
    transforms.Resize(256),
    transforms.ColorJitter(),
    transforms.RandomCrop(224),
    transforms.RandomHorizontalFlip(), transforms.Resize(128),
    transforms.ToTensor()

])

데이터 세트 = ImageFolder(args.data_path,
→ 변환=data_transform)
train_loader = DataLoader(데이터세트,
→ batch_size=batch_size, shuffle=True)

net = create_model(args.model, args.quantize) trainer = Trainer(max_epochs=1)
trainer.fit(net, train_loader)
```

그림 2. BigDL Nano를 사용한 PyTorch 훈련의 예. 일반적으로 pytorch 번개 가져오기 트레이너에서 bigdl.nano.pytorch 가져오기 트레이너로 가져오기를 변경하기만 하면 됩니다. 그리고 토치비전을 사용한 다면 비전 관련 수입도 바뀌어야 합니다.

다른 코드에서는 변경할 필요가 없습니다.

```
#... 모델 Net 정의
모델 = Net().to(장치) model.train()

#... 여기서 기차 루프 생략 # 트레이너 인스턴스화

트레이너 = bigdl.nano.pytorch.Trainer()
# 트레이너를 사용하여 모델을 양자화하고 onnx를 활성화합니다.
model_onnx_int8 = trainer.quantize(모델,
정밀도="int8", 가속기="onnxruntime", ...)
출력 = model_onnx_int8(데이터)
```

그림 3. BigDL-Nano를 사용하여 양자화 및 onnx를 사용한 PyTorch 추론의 예. 양자화 [6] 와 ONNX Runtime [7] 을 동시에 활성화하는 데 2줄의 추가 코드만 필요합니다 .

실제 파이프라인이 구성되고 각 시도에서 실행될 때까지 해당 객체.

3.4. 성능 결과

이 섹션에서는 BigDL-Nano의 일부 성능 결과를 제시합니다. 테스트에서 우리는 특정 하드웨어 플랫폼에서 ResNet50 [3]을 사용하여 고양이와 개를 분류하는 것과 같은 고전적인 이미지 분류 파이프라인을 실행하고 BigDL-Nano를 사용하거나 사용하지 않고 훈련 및 추론의 종단 간 성능을 측정했습니다.

우리는 동일한 파이프라인을 사용하는 두 가지 시나리오, 즉 "처음부터 훈련" 과 "전이 학습"을 테스트했습니다. ~ 안에

```
bigdl.nano.tf.keras 에서 가져오기 순차
bigdl.nano.tf.keras.layers import Dense 에서 ,
→ 병합, Conv2D
bigdl.nano.autoML.hpo.space를 공간으로 가져오기

모델 = 순차() model.add(Conv2D(
필터= space.Categorical(32, 64) ,
kernel_size= space.Categorical(3, 5) ,
보폭=2,
활성화="렐루",
입력_모양=입력_모양))
model.add(플래튼())
model.add(Dense(CLASSES, 활성화="소프트맥스"))
...
모델.컴파일(...) 모델.검색(...)
모델.핏(...)
```

그림 4. BigDL-Nano의 AutoML 예시. 사용자는 계층 인수 등에서 검색 공간을 지정한 다음 검색을 사용하여 최상의 하이퍼파라미터를 검색할 수 있습니다.

"train from scratch" 시나리오에서는 훈련 단계에서 모든 레이어의 가중치를 업데이트할 수 있는 반면, "transfer learning" 시나리오에서는 일부 레이어만 업데이트할 수 있고 나머지는 동결됩니다. "전이 학습" 시나리오에서 BigDL-Nano는 고정 레이어로 인해 훈련 단계에서 더 큰 속도 향상(최대 5.8배)을 제공합니다. 최적화(예: ONNX Runtime [7] 및 양자화 [6])가 추론을 위해 모든 계층에 적용되기 때문에 가속 속도 향상은 두 시나리오의 추론 단계에서 동일하게(최대 9.6x) 유지됩니다.

테스트 플랫폼의 경우 서버에서 예약한 랩톱 과 컨테이너를 선택했습니다 . 많은 데이터 사이언티스트가 로컬 실험을 위해 랩톱을 사용하지만 조직의 데이터 사이언티스트가 원격 서버(예: 클라우드)에서 할당된 컨테이너화된 환경을 사용하는 것도 일반적입니다. 노트북과 컨테이너의 세부 구성은 다음과 같습니다.

- 노트북 - 단일 8코어 Intel(R) Core (TM) i7-11800H CPU @ 2.30GHz, 12G 메모리, OS는 Ubuntu 20.04인 노트북
- 컨테이너 - 단일 소켓 Intel(R) Xeon(R) Platinum 8380H CPU @ 2.90GHz, 192G 메모리에 28개의 코어가 있는 docker [30] 컨테이너이며 OS는 Ubuntu 16.04입니다.

그림 5 와 그림 6 은 각각 노트북과 컨테이너에서 두 가지 테스트 시나리오에 대해 BigDL-Nano를 사용하여 훈련 및 추론의 속도 향상을 요약했습니다. 결과에서 알 수 있듯이 BigDL-Nano는 정확도 손실 없이 두 플랫폼 모두에서 학습 속도를 최대 5.8배, 추론 속도를 최대 9.6배 높일 수 있습니다. 코어가 더 많은 강력한 하드웨어(클라우드의 컨테이너와 같은)는 더 높은 가속률을 얻을 수 있습니다.

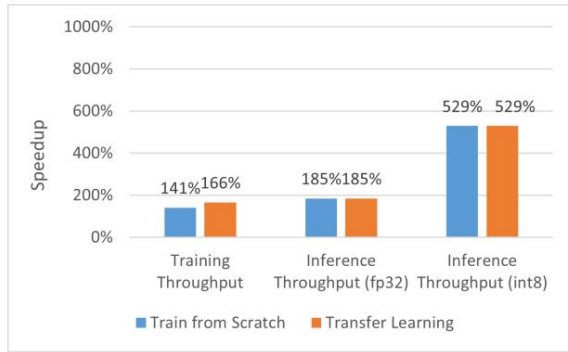


그림 5. 노트북에서 교육 및 추론을 위한 BigDL-Nano의 속도 향상

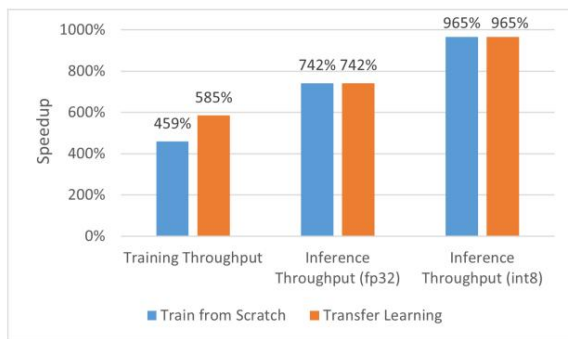


그림 6. BigDL-Nano의 속도 향상과 컨테이너에 대한 교육 및 추론

훈련과 추론 모두에서 노트북보다; int8(양자화 사용)은 일반적으로 fp32(ONNX Runtime 사용)보다 추론 속도가 빠릅니다.

4. BigDL-Orca를 통한 원활한 확장

그림 7은 BigDL-Orca의 전체 아키텍처를 보여줍니다. 엔드투엔드 AI 파이프라인을 랩탑에서 분산 클러스터로 원활하게 확장하기 위해 BigDL-Orca는 자동으로 Apache Spark [38] [2] 및/또는 Ray [31] [8]를 분산 클러스터를 위한 기본 실행 엔진으로 프로비저닝합니다. 데이터 처리 및 모델 교육/추론. 분산 엔진 위에서 사용자는 데이터 병렬 방식 (TensorFlow Dataset [10], PyTorch DataLoader [32], Spark Dataframes [12], Ray Datasets [8] 등)으로 데이터 파이프라인을 간단하게 구축할 수 있습니다. OpenCV [14], Pandas [36], SciPy [35], spaCy [24] 등과 같은 임의의 Python 라이브러리; 그런 다음 동일한 프로그램 내에서 사용자는 BigDL-Orca의 sklearn 스타일 Estimator API [15]를 사용하여 AI 모델(예: Tensorflow [10], PyTorch [32], MXNet [16] 등)을 분산 교육 및 추론을 위해 데이터를 처리했습니다.

4.1. 분산 데이터 처리 파이프라인

BigDL-Orca는 네 가지 유형의 분산 데이터 처리, 즉 TensorFlow Dataset 또는 PyTorch DataLoader, Spark Dataframe, Ray Dataset 및 XShards(임의의 Python 라이브러리)를 지원합니다.

Tensorflow Dataset 또는 PyTorch DataLoader. 사용자는 단일 노드 Tensorflow 또는 PyTorch 프로그램에서와 마찬가지로 표준 TensorFlow Dataset 또는 PyTorch DataLoader를 직접 사용하여 데이터 처리 파이프라인을 구축할 수 있습니다. 8. 내부적으로 BigDL-Orca는 클러스터의 각 노드에서 TensorFlow Dataset 또는 PyTorch DataLoader 파이프라인을 자동으로 복제하고, 입력 데이터를 사당하고, Apache Spark 및/또는 Ray를 사용하여 데이터 병렬 방식으로 데이터 파이프라인을 실행합니다.

Spark DataFrame 또는 Ray Dataset. Spark DataFrame은 사용자가 대규모 분산 데이터에 다양한 변환을 적용할 수 있는 공통 분산 데이터 구조입니다. Ray Dataset은 Ray 라이브러리 및 애플리케이션에서 데이터를 로드하고 교환하는 표준 방법인 Ray의 분산 데이터 구조입니다. Spark DataFrame과 Ray Dataset은 그림 9와 같이 BigDL-Orca를 사용할 때 데이터 변환 없이 TensorFlow/PyTorch 교육 또는 추론에 직접 사용할 수 있습니다.

XShards(임의의 Python 라이브러리). BigDL-Orca의 XShards API를 사용하면 분산 및 데이터 병렬 방식으로 기존 Python 코드를 사용하여 대규모 데이터 세트를 처리할 수 있습니다. 로컬 AI 파이프라인을 분산 클러스터로 확장할 때 사용자의 주요 과제는 분산 데이터 스토리지 또는 구조를 지원하기 위해 데이터 수집 또는 처리 코드를 다시 작성하는 것입니다(예: 새로운 분산 데이터 처리 라이브러리 사용). 이러한 코드 수정은 사용자가 새로운 API를 배워야 하며 사용자 코드와 새 라이브러리 간에 불일치가 있을 때 오류가 발생하기 쉽습니다.

XShards를 사용하여 사용자는 그림 10과 같이 원래 Python 코드를 재작성하는 대신 재구성하여 분산 데이터 로드 및 변환을 활성화할 수 있습니다. 본질적으로 XShards에는 자동으로 분할된(또는 분할된) Python 개체(예: Pandas [36]

Dataframe, Numpy [23] NDArray, Python 사전 또는 목록 등). XShards의 각 파티션은 Python 개체의 하위 집합을 저장하고 클러스터의 여러 노드에 분산됩니다. 사용자는 XShards.transform 샤드를 사용하여 데이터 병렬 방식으로 각 파티션에서 임의의 Python 코드를 실행할 수 있습니다.

4.2. 분산 교육 및 추론 파이프라인

BigDL-Orca는 투명하게 분산된 모델 교육 및 추론을 위해 sklearn 스타일 API(즉, Estimator)를 제공합니다. 분산 교육 및 추론을 수행하기 위해 사용자는 먼저 BigDL-Orca Estimator를 생성할 수 있습니다.

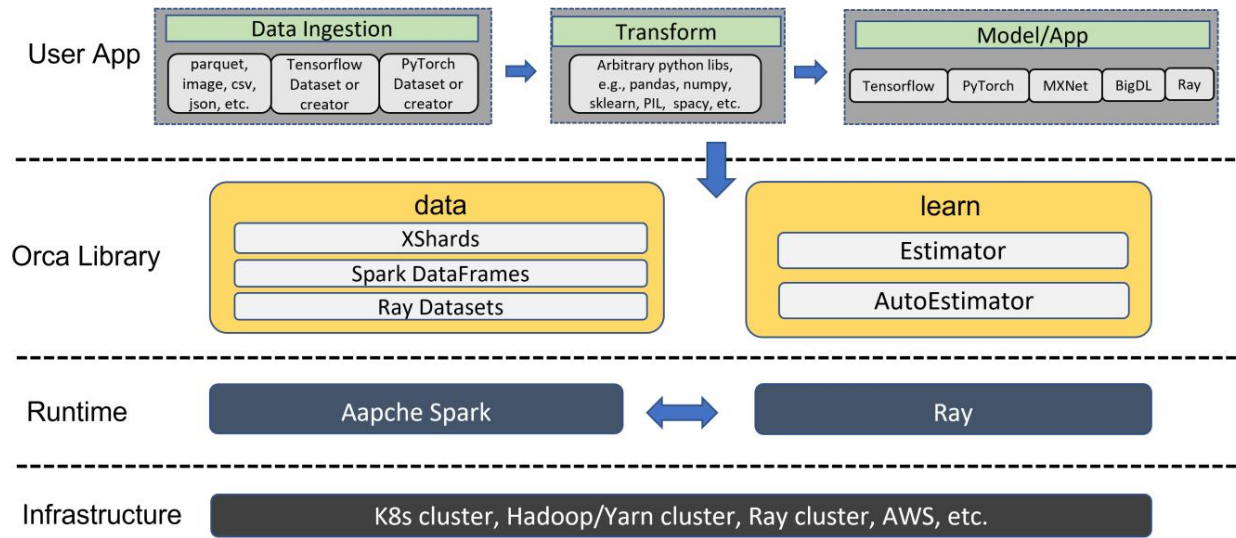


그림 7. BigDL-Orca 아키텍처

```

torchvision 에서 토치를
가져오기 데이터 세트 가져오기, bigdl.orca.learn.pytorch 에서 변환 가져오기 추정기

train_loader = torch.utils.data.DataLoader(
    데이터셋.MNIST(...),
    배치_크기=배치_크기,
    셔플=참)

est = Estimator.from_torch(모델=torch_model,
    → 옵티마이저=torch_optim, 손실=torch_criterion)
est.fit(데이터=train_loader)

```

그림 8. PyTorch 로더를 Estimator의 입력으로 사용하는 예

```

df = spark.read.parquet("data.parquet") # df는 Spark입니다.
→ 데이터프레임

est = Estimator.from_keras(keras_model=모델) est.fit(데이터=df,

    feature_cols=['사용자', '항목'], # 지정
    → 입력으로 사용할 열
    label_cols=['label']) # 다음을 지정합니다.
    → 레이블로 사용할 열

```

그림 9. Estimator는 Spark DataFrame을 직접 입력으로 사용합니다.

모든 표준(단일 노드) TensorFlow, Keras 또는 PyTorch 모델에서 데이터 병렬 처리 파이프라인을 입력으로 사용하여 Estimator.fit 또는 Estimator.predict 메서드를 호출합니다(그림 11 참조).

내부적으로 BigDL-Orca Estimator는 클러스터의 각 노드에서 모델을 복제하고 각 노드의 데이터 파티션(데이터 병렬 처리 파이프라인에서 생성됨)을 로컬 모델 복제본에 공급합니다.

```

def 부정(df, column_name):
    df[column_name] = df[column_name] * (-1) # 랜더
    → 코드
    변환 df

train_shards = shards.transform_shard(음수, → '가치')

```

그림 10. XShard 사용 예. 원래 변환 df[열 이름] = df[열 이름] * (-1)은 음의 함수로 재구성되고 변환 샤드를 사용하여 샤드 파티션에 적용됩니다.

```

#PySpark DataFrame train_df =
sqlcontext.read.parquet(...).withColumn(...) test_df = #TensorFlow 모델
...

tensorflow import keras 에서
...

model = keras.Model(inputs=[사용자, 아이템],
    → 출력 = 출력)
model.compile(옵티마이저= "아담", 손실=
    "sparse_categorical_crossentropy", 메트릭스=['정확도'])

#BigDL-Orca를 사용한 분산 학습 및 추론 est = Estimator.from_keras(keras_model=model)
est.fit(train_df, feature_cols=['user', 'item'],

    → label_cols=['라벨']) est.predict(test_df,
    feature_cols=['사용자', '항목'])

```

그림 11. Estimator를 사용하여 PySpark Dataframe에서 Tensorflow keras 모델을 훈련하고 추론을 수행하는 예.

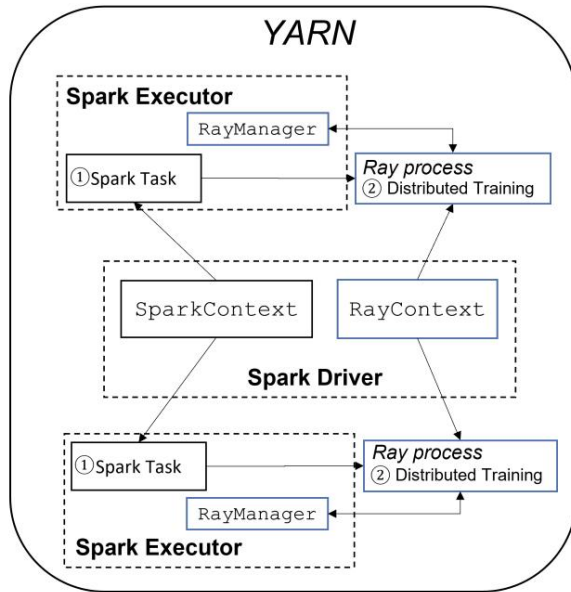


그림 12. RayOnSpark 아키텍처

다양한 백엔드 기술(예: torch.distributed [27], tf.distribute.MirroredStrategy, Horovod [33] 또는 BigDL[19]의 매개변수 동기화 계층)을 사용하여 모델 매개변수를 동기화합니다.

또한 BigDL-Orca는 AutoEstimator API를 제공하여 사용자가 자동 하이퍼파라미터 튜닝 [29] [11] 을 기본적으로 지원하여 모델을 교육하는 데 도움을 줄 수 있습니다. AutoEstimator에는 Estimator와 유사한 인터페이스가 있으며 검색 공간 및 검색 알고리즘 등과 같은 추가 조정 관련 구성이 있습니다.

4.3. 스파크 파이프라인의 레이

BigDL-Orca는 분산형 딥 러닝 교육 및 추론 외에도 RayOnSpark 지원을 통해 Ray [8]를 빅 데이터 플랫폼에 원활하게 통합합니다.

그림 12 는 RayOnSpark의 아키텍처를 보여줍니다. SparkContext는 여러 Spark 실행기 실행을 담당하는 드라이버에서 먼저 생성됩니다. RayOnSpark에서 Spark [38] 드라이버 프로그램은 RayContext를 생성하여 각 Spark 실행기와 함께 Ray 프로세스를 자동으로 시작합니다. RayOnSpark는 또한 Ray 프로세스를 관리하기 위해 각 Spark 실행자 측에 RayManager를 생성합니다(예: 훈련이 완료되면 자동으로 종료).

결과적으로 사용자는 Spark 프로그램 내에서 Ray 코드를 직접 작성할 수 있으므로 Ray 애플리케이션을 빅 데이터 처리 파이프라인에 원활하게 통합하고 메모리 내 Spark RDD 또는 DataFrame에서 직접 실행할 수 있습니다.

5. 실제 사용 사례

이 섹션에서는 Mastercard 및 Inspur에서 BigDL 2.0의 실제 사용 사례를 공유합니다.

Mastercard의 "AI at Scale". BigDL 2.0 위에 구축된 Mastercard는 전체 기계 학습 수명 주기(데이터 분석, 실험, 모델 교육, 배포, 리소스 최적화, 모니터링 등 포함)를 가속화하기 위해 "규모의 AI" 접근 방식을 채택했습니다[1]. 이는 하이브리드 데이터/ML 인프라(AI/ML 파이프라인 및 모델 수명 주기 관리를 자동화함)에서 BigDL 2.0으로 통합 빅 데이터 AI 아키텍처를 구축함으로써 달성됩니다. 결과적으로 Mastercard 엔지니어는 빅 데이터 분석(Spark 에코시스템) 및 딥 러닝(TensorFlow 및 Keras 사용)을 엔드 투 엔드 AI 애플리케이션에 원활하게 통합할 수 있습니다. 결과적으로 Mastercard는 특수 목적 프로세서의 추가 비용과 복잡성을 피할 수 있으며 AI 교육 작업은 평균 5시간 이내에 완료할 수 있습니다(최대 22억 명의 사용자와 수백 명의 사용자를 지원하기 위해 수백 대의 Intel Xeon 서버에서 실행). 수십억 개의 레코드).

Inspur의 스마트 교통 솔루션. Inspur는 BigDL 2.0을 사용하여 중단 간 CV(컴퓨터 비전) 기반 스마트 교통 솔루션을 구축했습니다. 이 솔루션은 빅데이터 전처리, 모델 교육, 추론 및 기존 빅데이터 처리 워크플로우와 통합된 통합 빅데이터 및 AI 분석 플랫폼을 제공합니다. 특히 분산 비디오 스트림 데이터 처리에서 분산 AI 모델 교육/추론(다중 개체 추적 및 OCR 포함), 차량 궤적 바인딩에 이르는 엔드 투 엔드 파이프라인을 구축하여 성능이 30% 향상되고 성능이 40% 향상됩니다. 낮은 비용. 자세한 내용은 이전 CVPR 2021 자습서 세션 [18] 을 참조하십시오.

6. 결론

본 논문에서는 오픈소스인 BigDL 2.0을 제시하였다. 빅데이터 AI 툴킷 (<https://github.com/intel-analytics/BigDL/>). BigDL 2.0을 사용하여 사용자는 랩톱에서 기존 Python 노트북(AutoML 지원 가능)을 간단하게 구축할 수 있으며 단일 노드에서 투명하게 가속할 수 있고(실험에서 표시된 대로 최대 9.6배 속도 향상) 원활하게 확장할 수 있습니다. 대규모 클러스터(실제 사용 사례에 표시된 것처럼 수백 대의 서버에 걸쳐)로 확장합니다. BigDL 2.0은 이미 많은 실제 사용자(예: Mastercard, Burger King, Inspur 등)가 프로덕션에서 채택했습니다.

참조

[1] BigDL을 사용하는 Mastercard의 "AI at Scale". <https://www.intel.com/content/www/us/en/develop/bigdl08/technical/ai-at-scale.html>

[2] 아파치 스파크 프로젝트. spark.apache.org. 2, 4 [3] 이미지 분류 및 미세 조정 예. <https://github.com/intel-analytics/BigDL/tree/master/examples>

branch-2.0/python/nano/example/pytorch/pl-finetune. 삼

[4] OpenVINO 툴킷의 Intel 배포. <https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html>. 2 [5] PyTorch용 Intel 확장. <https://github.com/intel/intel-extension-for-pytorch>. 2 [6] Intel 신경 압축기. <https://github.com/>

인텔/신경 압축기. 2, 3 [7] ONNX 런타임. [onnxruntime.ai](https://github.com/onnxruntime). 2, 3 [8] 레이 프로젝트. [ray.io](https://github.com/ray-project). 2, 4, 6 [9] tcmlloc. [tcmalloc](https://github.com/tcmalloc). 1, 2 <https://github.com/google/tcmalloc>

[10] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghe mawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Mur Ray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete War den, Martin Wicke, Yuan Yu 및 Xiaoqiang Zheng. Ten sorflow: 대규모 기계 학습을 위한 시스템입니다. 운영 체제 설계 및 구현에 관한 12회 USENIX 심포지엄(OSDI 16), 페이지 265–283, 2016. 1, 2, 4 [11] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta 및 Masanori Koyama. Optuna: 차세대 하이퍼파라미터 최적화 프레임워크. 지식 애지 발견 및 데이터 마이닝에 관한 제25회 ACM SIGKDD 국제 회의 진행, 페이지 2623–2631, 2019. 2, 6

[12] Michael Armbrust, Reynold S Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K Bradley, Xiangrui Meng, Tomer Kaf tan, Michael J Franklin, Ali Ghodsi 등. Spark SQL: Spark에서 관계형 데이터 처리. 데이터 관리에 관한 2015년 ACM SIGMOD 국제 회의 진행, 1383–1394페이지, 2015. 2, 4 [13] James Bergstra, Dan Yamins, David D Cox, et al. Hyperopt: 기계 학습 알고리즘의 하이퍼파라미터를 최적화하기 위한 Python 라이브러리입니다. 12th Python in science conference 회보, 13권, 20페이지. Citeseer, 2013. 2 [14] G. Bradski. OpenCV 라이브러리. Dobb 박사의 소프트웨어 도구 저널, 2000. 4 [15] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pe dregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas , Arnaud Joly, Brian Holt 및 Gael Varoquaux. 기계 학습 소프트웨어용 API 설계: scikit-learn 프로젝트의 경험. In ECML PKDD 워크숍: 데이터 마이닝 및 기계 학습용 언어, 108–122페이지, 2013. 2, 4 [16] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyan Zhang, Zheng Zhang. Mxnet: 이 기종 분산 시스템을 위한 유연하고 효율적인 기계 학습 라이브러리입니다. arXiv preprint arXiv:1512.01274, 2015. 4 [17] Jian Cheng, Pei-song Wang, Gang Li, Qinghao Hu, Han-qing Lu. 심층 컨벌루션 신경망의 효율적인 계산에 대한 최근의 발전. Frontiers of Information Technology & Electronic Engineering, 19(1):64–77, 2018. 1

[18] 제이슨 다이, 제이슨 리우, 양 왕. Analytics Zoo를 사용하여 Ray 및 Apache Spark의 엔드 투 엔드 빅 데이터 AI 파이프라인. IEEE 회의 컴퓨팅 비스. 패턴 인식 Tutorial., 2021. 1, 2, 6 [19] Jason Jinquan Dai, Yiheng Wang, Xin Qiu, Ding Ding, Yao Zhang, Yanzhang Wang, Xianyan Jia, Cherry Li Zhang, Yan Wan, Zhichao Li, et al. Bigdl: 빅 데이터용 분산 딥 러닝 프레임워크입니다. 클라우드 컴퓨팅에 관한 ACM 심포지엄 진행, 50–60페이지, 2019. 1, 2, 6 [20] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'auelio Ranzato, Andrew S'ni'ar, Paul Tucker, Ke Yang 등 대규모 분산 심층 네트워크. 신경 정보 처리 시스템의 발전, 25, 2012. 1 [21] Jason Evans. freebsd를 위한 확장 가능한 동시 malloc(3) 구현입니다. 프로세스에서 bsdcn 회의, 캐나다 오타와, 2006. 1, 2

[22] William Falcon과 The PyTorch Lightning 팀. 파이토치 라이트닝. <https://github.com/PyTorchLightning/pytorch-lightning>, 2019. 2 [23] Charles R Harris, K Jarrod Millman, Stefan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith , 등. numpy를 이용한 배열 프로그래밍 자연, 585(7825):357–362, 2020. 4

[24] 매튜 호니발과 이네스 몬타니. spaCy 2: Bloom 임베딩, 컨벌루션 신경망 및 증분 구문 분석을 통한 자연 언어 이해. 출현 예정, 2017. 4 [25] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, Dmitry Kalenichenko. 효율적인 정수 산술 전용 추론을 위한 신경망의 양자화 및 훈련. 컴퓨팅 버전 및 패턴 인식에 관한 IEEE 회의 진행, 페이지 2704–2713, 2018. 1 [26] Gregory Lento. 인텔 고급 벡터 확장으로 성능 최적화. 인텔 백서(2014년 9월), 2014. 1, 2 [27] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, et al. Pytorch 분산: 데이터 병렬 교육 가속화에 대한 경험. arXiv 프리프린트 arXiv:2006.15704, 2020. 1, 2, 6

[28] X. Li, G. Zhang, K. Li 및 W. Zheng. 딥 러닝 및 병렬화, 95–118페이지. 12 2016. 1 [29] Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, Ion Stoica. Tune: 분산 모델 선택 및 교육을 위한 연구 플랫폼입니다. arXiv preprint arXiv:1807.05118, 2018. 2, 6 [30] 더크 메르켈. Docker: 일관된 개발 및 배포를 위한 경량 Linux 컨테이너입니다. Linux 저널, 2014(239):2, 2014. 3 [31] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan 등. Ray: 새로운 {AI} 애플리케이션을 위한 분산 프레임워크입니다. 운영 체제 설계 및 구현에 관한 13회 USENIX 심포지엄(OSDI 18), 페이지 561–577, 2018. 1, 2, 4 [32] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Kline, 제밍

Lin, Natalia Gimelshein, Luca Antiga 등. Pytorch: 필수 스타일의 고성능 딥 러닝 라이브러리입니다. 신경 정보 처리 시스템의 발전, 32, 2019. 1, 2, 4

[33] 알렉산더 세르게예프와 마이크 델 발소. Horovod: tensorflow의 빠르고 쉬운 분산 딥 러닝, 2018. 6 [34] Karen Simonyan 및 Andrew Zisserman. 대규모 이미지 인식을 위한 매우 깊은 컨볼루션 네트워크. arXiv 프리프린트 arXiv:1409.1556, 2014. 1

[35] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stefan J. van der Walt, Matthew Brett, Joshua Wilson, K. 자로드 밀먼, 니콜라이 마요로프, 앤드류 R.J. Nelson, Eric Jones, Robert Kern, Eric Larson, C.J. Carey, · Ilhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, EA Quintero, Charles R. Harris, Anne M. Archibald, Antonio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt 및 SciPy 1.0 기여자. SciPy 1.0: Python의 과학적 컴퓨팅을 위한 기본 알고리즘. Nature Methods, 17:261–272, 2020. 4 [36] Wes McKinney. Python의 통계 컴퓨팅을 위한 데이터 구조. Stefan van der Walt 및 Jarrod Millman, 편집자, 제9회 Python in Science Conference 회보, 페이지 56 – 61, 2010. 4

[37] Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Yu-Feng Li, Wei-Wei Tu, Qiang Yang 및 Yang Yu. 테이킹 휴먼 아웃 오브 러닝 애플리케이션: A Survey on automated machine learning, 2019. 1, 2 [38] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, 마이클 J 프랭클린 외. Apache Spark: 빅 데이터 처리를 위한 통합 엔진입니다. ACM 통신, 59(11):56–65, 2016. 2, 4, 6 [39] Martin Zinkevich, Markus Weimer, Lihong Li, Alex Smola. 병렬화된 확률적 경사하강법. 신경 정보 처리 시스템의 발전, 23, 2010. 1