심화 교육과정

Data Loader, Better , Faster , Stronger

2022. 05. 23.

II 데이터 로더 관련 이야기

Large parquet dataset을 위한 Pytorch dataset, dataloader 튜닝

딥러닝을 하면서 처음 마주하는 고민은 " 데이터 로더를 어떻게 만들까?" 인 것 같습니다.

1. 방법은 가지고 있는 데이터의 형식에 따라, 양이 많고 적음에 따라, 사용할 프레임 워크에 따라 항상 다르게 만드는 방법 입니다.

2. 데이터 크기가 아주 작은 경우에는 데이터 전체를 메모리에 업로드해서 접근 하는 방법

데이터 로더의 조건

좋은 데이터 로더는 어떤 특징이 있을까?

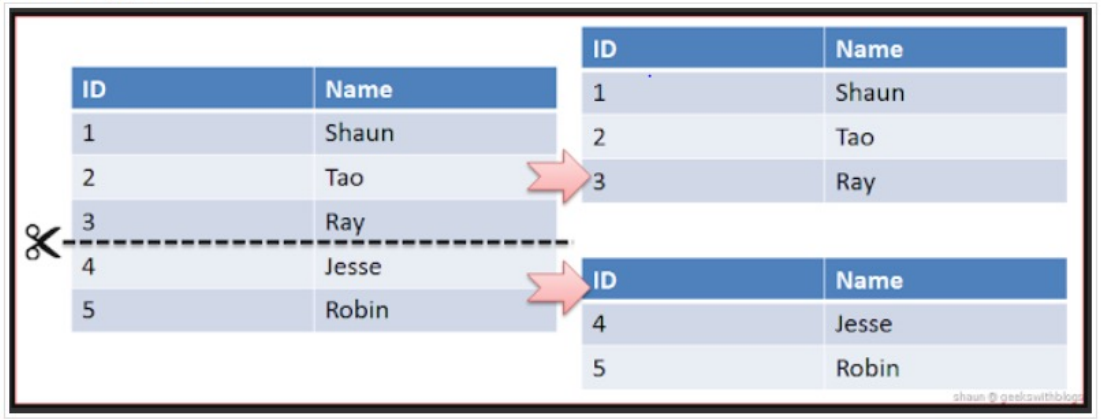
- 여러 파일로 쪼개져서 저장/읽기 기능
- 간단한 Key Value 구조
- Lazy data loading(메모리 보호)
- IO 병목을 막기 위한 데이터 캐시
- 데이터 custom transform 을 유연하게 적용하는 기능

데이터 로더의 조건

1. 여러 파일을 쪼개져서 저장 / 읽기 가능

- 분산 처리 데이터베이스에서는 데이터를 partition(분할) 또는 shard(조각)로 나누어 저장한다. 이는 병렬 처리에 유리한 구조이기 때문에 가급적 여러 파일인 형태를 그대로 유지하는 것이 좋다 (repartition에도 시간 비용이 들어가기 때문) 또한 data shuffle이 제공되어야 한다.

샤딩(Sharding)이란?



• 같은 테이블 스키마를 가진 데이터를 다수의 데이터베이스에 분산하여 저장하는 방법을 의미합니다.

데이터 로더의 조건

2. 간단한 Key, value 구조

- pandas나 Spark, h5의 경우 column과 데이터 타입 조회가 쉽다. tfrecord는 직렬화되어 있기 때문에 feature key 값과 데이터 타입 조회가 어렵다.

Key value 구조의 장점은 명확하다. 개발적으로는 데이터 집합 내의 요소들을 key 로 구분하여 알아보기 쉽게 해주며, 자료구조적으로는 O(1)에 가까운 접근 시간을 가진다.
즉 알아보기도 쉽고 사용하기도 쉬우면서 심지어 매우 빠르다.

데이터 로더의 조건

3. lazy data loading(메모리 보호)

- 메모리가 TB급이 아니라면 학습 데이터 전체를 메모리에 업로드하는 것이 불가능하다.
  따라서 학습에 필요한 부분만 로드하는 기능이 필요하다.

4. IO 병목을 막기 위한 데이터 캐시

- 모든 데이터를 메모리에 로드하고 사용할 수 없기 때문에 반드시 학습 중간에 IO 로딩을 거쳐야 한다.
  이 과정에서 IO 로딩 속도 문제로 병목이 발생하므로 IO 로딩 병렬화와 캐시 기능이 제공되어야 한다.

5. 데이터에 custom transform을 유연하게 적용 가능

- 데이터 변형에는 문자열 형식의 label을 정수형인 index로 바꾸는 정적인 변환 뿐만 아니라
  image augmentation, NLP token masking 같은 동적인 변환도 존재한다.
- 데이터 로더를 만드는 과정에서 동적 변환이 쉽게 지원되도록 구현해야 한다.

다른 곳은 어떻게 처리하고 있을까 ?

우리가 원하는 기능을 누군가 이미 만들어두지 않았을까? 데이터셋과 데이터 로더 관련 기술을 찾아보면 다음과 같은 도구를 찾을 수 있다.

- Google: TensorFlow - tfrecord, tf.data API
- Facebook: Pytorch - torch.utils.data API
- Uber: Petastorm https://github.com/uber/petastorm#pytorch-api
- Nvidia: Dali

다른 곳은 어떻게 처리하고 있을까 ?

Uber: Petastorm
- https://github.com/uber/petastorm#pytorch-api

Petastorm
•공식 API: https://petastorm.readthedocs.io/en/latest
•우버 블로그: https://eng.uber.com/petastorm/

우버에서도 큰 데이터에 대한 학습 방법을 고민한 것 같다. 그 결과로 TB급의 parquet 데이터를 직접 다룰 수 있는 Petastorm이라는 오픈소스를 발표했다.
Petastorm은 Python 기반의 ML 프레임워크(TensorFlow, PyTorch, PySpark)를 모두 지원한다는 큰 장점이 있고 pure Python 코드로 작성되었다는 것이 특징이다.

실질적으로 원격 저장소에서 직접 데이터를 읽을 수 있도록 PySpark를 지원한다는 점이 큰 장점으로 느껴졌다.

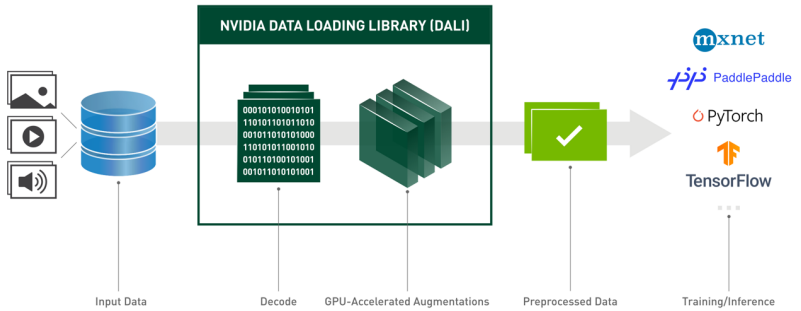
* PySpark : 대규모의 데이터를 처리할 수 있는 기능을 제공하는 빅데이터 처리용 플랫폼이다.
SQL, 기계학습, 딥러닝 및 그래프 처리를 위한 모델이 내장된 빅데이터 처리용 데이터 분석 엔진을 말한다.
- 관련 자료 : https://sexydatadesigner.tistory.com/144

다른 곳은 어떻게 처리하고 있을까 ?

Nvidia: Dali

GPU에는 jpeg을 비롯한 여러 data decoding 기능을 갖고 있는데요, 지금까지 DL Framework들은 이들을 활용하지 않고 CPU를 이용해 왔었습니다.
보통 Data Argumentation을 Data Layer에서 돌리시거나 큰 이미지를 사용하셨다면 어느 시점에서 GPU가 많더라도 CPU가 점유율이 100%가 되면서 GPU가 노는 안습인 상황을 맞으셨을 거에요.
이제 Framework(TF, PyTorch, MxNet)에 DALI 라이브러리를 추가하시면 GPU가 jpeg, raw image, LMDB, TFRecord, RecordIO의 Decoding을 시작합니다.

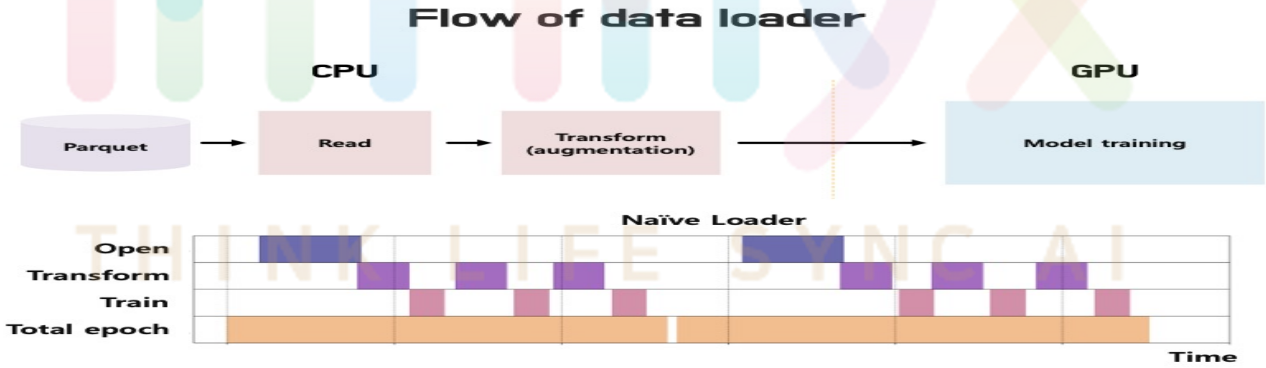
개발 문서 https://docs.nvidia.com/deeplearning/dali/user-guide/docs/index.html



PyTorch 데이터 로더 하는 방법 설명

PyTorch 데이터 로더

일반적인 데이터 로더의 흐름은 다음과 같습니다.



순차적으로 이루어지는 경우 실행 시간을 그래프를 그려보면 학습 시간 전체는 각 과정이 소요하는 시간의 합이라고 할 수 있다.

•y축(위쪽부터): open, read+transform, train, epoch total

•x축: 시간

학습 시간을 단축시키기 위해서 파일 open 과정과 transform 과정에 소모되는 시간을 단축시키는 것이 중요하다. GPU에서 모델 학습이 이루어지기 때문에 CPU는 학습을 위한 데이터 준비를 병렬적으로 준비해야 한다.

PyTorch 데이터 로더 하는 방법 설명

용어 정리

- Data prefetch 란 앞으로 연산에 필요한 data들을 미리 가져오는 것을 의미

PyTorch에서 prefetch 기능을 잘 지원하고 있는지 확인할 필요가 있다. 그 전에 PyTorch의 데이터 로더 방식을 간단하게 리뷰해보겠다.

https://pytorch.org/tutorials/beginner/dataloadingtutorial.html

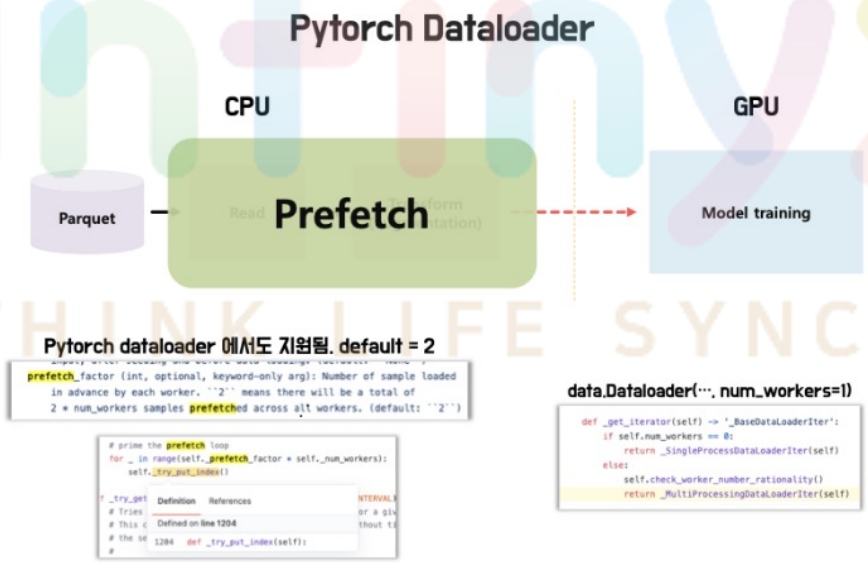
```
class PytorchDataset(torch.utils.data.Dataset):
    def __init__(self, data_files):
        ''' initialize '''
        self.data = <data_files>

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        sample = self.transform(self.data[idx])
        return sample
```

- PyTorch 에서는 custom data를 사용할 때 1개의 sample 접근할수 있는 Dataset 클래스를 정의한다. 이 클래스는 객체를 torch.utils.data.DataLoader() 에 인자로 넘기는 것만으로 손쉽게 배치 데이터를 얻을 수 있다.

예에서 볼 수 있듯이 직관적인 데이터 접근 인터페이스를 제공하는 것과 데이터셋 구성의 자유도가 높다는 것이 장점이지만 자유도가 높은 만큼 복잡한 구성에서 코드 가독성이 떨어지는 단점이 발생할 수 있다.

PyTorch 데이터 로더 하는 방법 설명

우리가 원하는 prefetch 기능이 제공되는지 확인해 보겠다.



torch.utils.data.DataLoader에서 num_workers와 prefetch_factor를 사용해 병렬화와 메모리 캐시에 대한 기능을 제공한다. num_workers > 0 일 때 두 기능 모두 활성화된다.

Transform – augmentation 속도 비교

albumentations는 numpy, opencv 기반으로 구현된 image transform 라이브러리이다. 사용법은 torchvision.transforms과 유사하며 torchvision보다 약 2~15배 빠른 속도를 나타낸다.

다음은 하나의 image에 대해 transform을 적용하는 작업의 1000회 실행 시간을 측정하여 평균을 낸 값이다.

표 3 라이브러리별 image transform 1000회 실행 시간 비교

라이브러리	CPU
torchvision	30.54260밀리초
albumentations	2.55252밀리초
korina	79.2268347밀리초

Transform – augmentation compose 의미

torchvision.transforms과 albumentations은 여러 변환 기법을 compose 함수로 묶어 처리할 수 있기 때문에 하나의 객체로 관리할 수 있다.

```
album_transforms = A.Compose([
        A.Resize(250, 250),
        A.ShiftScaleRotate(
                        shift_limit=0.05,
                        scale_limit=0.05,
                        rotate_limit=15,
                        p=0.5),
        A.RandomCrop(250, 250),
        A.HorizontalFlip(p=0.5),
        #A.RandomBrightnessContrast(p=0.2),
        #A.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)),
        ToTensorV2(),
    ])
```

CPU vs GPU

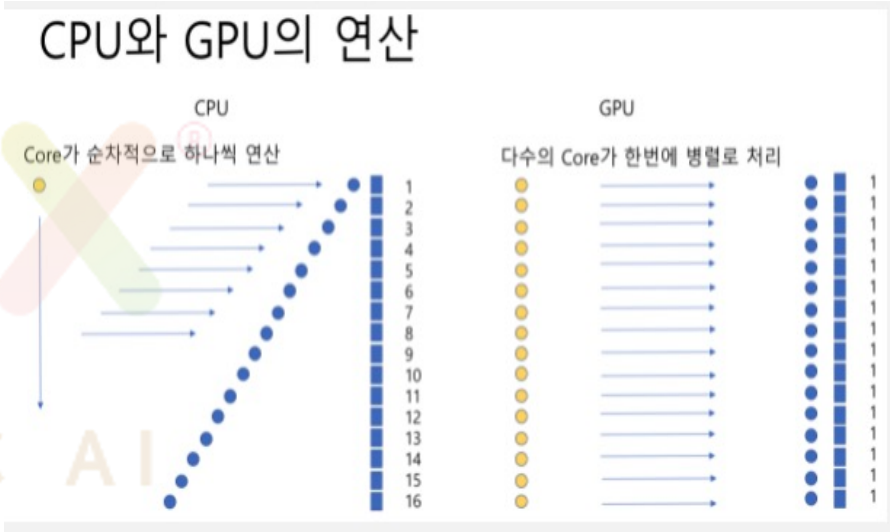
간단하게 보는 CPU GPU 연산 차이

기본적인 차이
1 . CPU와 달리 GPU 코어가 아주 많다.
* 코어 : CPU의 Core 즉 CPU의 핵심적인 역할을 수행해내는 중심부 역할을 말하며 이 코어에서 시스템의 모든 연산을 처리한다고 보면 되겠다, 즉 CPU에서의 코어가 많은 경우 컴퓨터의 성능을 가장 좌우한다고 볼 수 있다.

2. CPU는 복잡한 계산을 빠르게 할 수 있지만 모두 직렬 처리

3. GPU는 간단한 계산을 빠르게 할 수 있고, 많은 연산을 병렬 처리 가능

간단하게 보는 CPU GPU 연산 과정



장단점
CPU
1. 복잡한 계산을 코어 갯수 만큼씩 처리한다.
2. 예로 복잡한 팩토리얼 계산식을 2개 계산해야한다고 가정하면 CPU로 계산을 해주면 빨리할 수 있다.
3. 단점 간단하고 많은 계산식은 오래 걸림

GPU
1. 간단한 아주 많은 계산식을 동시에 빠르게 처리 가능
2. 예로 1000개의 덧셈식을 한번에 병렬로 처리가 가능
3. 단점 초기에 알고리즘을 하드웨어 병렬로 부여해 주어야하고 복잡한 식을 입력하면 도리어 CPU 연산속도 보다 느려질 수 있다.


감사합니다.
THINK LIFE SYNC AI
Infinyx