Sử dụng với PyTorch

Tài liệu này là phần giới thiệu nhanh về cách sử dụng bộ dữ liệu với PyTorch, với trọng tâm cụ thể là về cách để có được
các đối tượng torch.Tensor nằm ngoài bộ dữ liệu của chúng tôi cũng như cách sử dụng PyTorch DataLoader Tập dữ liệu ôm mặt
với hiệu suất tốt nhất.

Định dạng tập dữ liệu

Theo mặc định, bộ dữ liệu trả về các đối tượng python thông thường: số nguyên, số float, chuỗi, danh sách, v

Thay vào đó, để có được các tensor PyTorch, bạn có thể đặt định dạng của tập dữ liệu thành pytorch bằng cá Dataset.with_format():

```
>>> from datasets import Dataset
>>> data = [[1, 2],[3, 4]]
>>> ds = Dataset.from_dict({"data": data})
>>> ds = ds.with_format("torch")
>>> ds[0]
{'data': tensor([1, 2])}
>>> ds[:2]
{'data': tensor([[1, 2],
                 [3, 4]])}
```

[!TIP]
Đối tượng Dataset là một trình bao bọc của bảng Mũi tên, cho phép đọc nhanh không sao chép từ các mảng trong tập dữ liệu sang các tensor PyTorch.

Để tải dữ liệu dưới dạng tensor trên GPU, hãy chỉ định đối số thiết bị:

```
>>> import torch
>>> device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
>>> ds = ds.with_format("torch", device=device)
>>> ds[0]
{'data': tensor([1, 2], device='cuda:0')}
```

Mảng N chiều

Nếu tập dữ liệu của bạn bao gồm các mảng N chiều, bạn sẽ thấy theo mặc định chúng là được coi là cùng một tenxơ nếu hình dạng cố định:

```
>>> from datasets import Dataset
>>> data = [[[1, 2],[3, 4]],[[5, 6],[7, 8]]]# fixed shape
>>> ds = Dataset.from_dict({"data": data})
>>> ds = ds.with_format("torch")
>>> ds[0]
{'data': tensor([[1, 2],
                 [3, 4]])}
```

```
>>> from datasets import Dataset
>>> data = [[[1, 2],[3]],[[4, 5, 6],[7, 8]]]# varying shape
>>> ds = Dataset.from_dict({"data": data})
>>> ds = ds.with_format("torch")
>>> ds[0]
{'data': [tensor([1, 2]), tensor([3])]}
```

Tuy nhiên logic này thường yêu cầu so sánh hình dạng và sao chép dữ liệu chậm.
Để tránh điều này, bạn phải sử dụng rõ ràng loại tính năng Mảng và chỉ định hình dạng của tensor:

```
>>> from datasets import Dataset, Features, Array2D
>>> data = [[[1, 2],[3, 4]],[[5, 6],[7, 8]]]
>>> features = Features({"data": Array2D(shape=(2, 2), dtype='int32')})
>>> ds = Dataset.from_dict({"data": data}, features=features)
>>> ds = ds.with_format("torch")
>>> ds[0]
{'data': tensor([[1, 2],
                 [3, 4]])}
>>> ds[:2]
{'data': tensor([[[1, 2],
                  [3, 4]],

                 [[5, 6],
                  [7, 8]]])}
```

Các loại tính năng khác

Dữ liệu ClassLabel được chuyển đổi chính xác thành tensor:

```
>>> from datasets import Dataset, Features, ClassLabel
>>> labels = [0, 0, 1]
>>> features = Features({"label": ClassLabel(names=["negative", "positive"])})
>>> ds = Dataset.from_dict({"label": labels}, features=features)
>>> ds = ds.with_format("torch")
>>> ds[:3]
{'label': tensor([0, 0, 1])}
```

Các đối tượng chuỗi và nhị phân không thay đổi vì PyTorch chỉ hỗ trợ số.

Các loại tính năng Hình ảnh và Âm thanh cũng được hỗ trợ.

> [!TIP]
> Để sử dụng loại tính năng Hình ảnh, bạn sẽ cần cài đặt thêm tầm nhìn như
> pip install datasets[vision] .

```
>>> from datasets import Dataset, Features, Audio, Image
>>> images = ["path/to/image.png"] * 10
>>> features = Features({"image": Image()})
>>> ds = Dataset.from_dict({"image": images}, features=features)
>>> ds = ds.with_format("torch")
>>> ds[0]["image"].shape
torch.Size([512, 512, 4])
>>> ds[0]
{'image': tensor([[[255, 215, 106, 255],
                   [255, 215, 106, 255],
                   None
                   [255, 255, 255, 255],
                   [255, 255, 255, 255]]], dtype=torch.uint8)}
>>> ds[:2]["image"].shape
torch.Size([2, 512, 512, 4])
>>> ds[:2]
{'image': tensor([[[[255, 215, 106, 255],
                    [255, 215, 106, 255],
                    None
                    [255, 255, 255, 255],
                    [255, 255, 255, 255]]]], dtype=torch.uint8)}
```

[!TIP]
Để sử dụng loại tính năng Âm thanh, bạn sẽ cần cài đặt thêm âm thanh như
`pip install datasets[audio]` .

```
>>> from datasets import Dataset, Features, Audio, Image
>>> audio = ["path/to/audio.wav"] * 10
>>> features = Features({"audio": Audio()})
>>> ds = Dataset.from_dict({"audio": audio}, features=features)
>>> ds = ds.with_format("torch")
>>> ds[0]["audio"]["array"]
tensor([ 6.1035e-05,1.5259e-05,1.6785e-04,..., -1.5259e-05,
            -1.5259e-05,1.5259e-05])
>>> ds[0]["audio"]["sampling_rate"]
tensor(44100)
```

Đang tải dữ liệu

Giống như các đối tượng torch.utils.data.Dataset, Bộ dữ liệu có thể được truyền trực tiếp đến PyTorch Trình tải dữ liệu:

```
>>> import numpy as np
>>> from datasets import Dataset
>>> from torch.utils.data import DataLoader
>>> data = np.random.rand(16)
>>> label = np.random.randint(0, 2, size=16)
>>> ds = Dataset.from_dict({"data": data, "label": label}).with_format("torch")
>>> dataloader = DataLoader(ds, batch_size=4)
>>> for batch in dataloader:
...print(batch)
{'data': tensor([0.0047, 0.4979, 0.6726, 0.8105]), 'label': tensor([0, 1, 0, 1])}
{'data': tensor([0.4832, 0.2723, 0.4259, 0.2224]), 'label': tensor([0, 0, 0, 0])}
{'data': tensor([0.5837, 0.3444, 0.4658, 0.6417]), 'label': tensor([0, 1, 0, 0])}
{'data': tensor([0.7022, 0.1225, 0.7228, 0.8259]), 'label': tensor([1, 1, 1, 1])}
```

Tối ưu hóa tải dữ liệu

Có một số cách bạn có thể tăng tốc độ tải dữ liệu, điều này có thể giúp bạn tiết kiệm thời gian, đặc biệt nếu bạn đang làm việc với các tập dữ liệu lớn.
PyTorch cung cấp khả năng tải dữ liệu song song, truy xuất hàng loạt chỉ mục thay vì riêng lẻ, và phát trực tuyến để lặp lại tập dữ liệu mà không cần tải nó xuống đĩa.

Sử dụng nhiều Công nhân

Bạn có thể song song việc tải dữ liệu với đối số num_workers của PyTorch DataLoader và nhận được thông lượng cao hơn.

Dưới mui xe, DataLoader bắt đầu các quy trình num_workers.
Mỗi quy trình sẽ tải lại tập dữ liệu được chuyển đến DataLoader và được sử dụng để truy vấn các ví dụ.
Tải lại tập dữ liệu bên trong một công nhân không làm đầy RAM của bạn vì nó chỉ đơn giản là bản đồ bộ nhớ lại tập dữ liệu từ đĩa của bạn.

```
>>> import numpy as np
>>> from datasets import Dataset, load_from_disk
>>> from torch.utils.data import DataLoader
>>> data = np.random.rand(10_000)
>>> Dataset.from_dict({"data": data}).save_to_disk("my_dataset")
>>> ds = load_from_disk("my_dataset").with_format("torch")
>>> dataloader = DataLoader(ds, batch_size=32, num_workers=4)
```

Truyền dữ liệu

Truyền phát tập dữ liệu bằng cách tải nó dưới dạng IterableDataset. Điều này cho phép bạn lặp lại dần dần qua tập dữ liệu từ xa mà không cần tải nó xuống đĩa và hoặc qua các tệp dữ liệu cục bộ.
Tìm hiểu thêm về loại tập dữ liệu nào phù hợp nhất với trường hợp sử dụng của bạn trong việc lựa chọn giữa tập dữ liệu thông thường hoặc hướng dẫn về tập dữ liệu có thể lặp lại.

Một tập dữ liệu có thể lặp lại từ các tập dữ liệu kế thừa từ torch.utils.data.IterableDataset để bạn có thể chuyển nó tới torch.utils.data.DataLoader:

```
>>> import numpy as np
>>> from datasets import Dataset, load_dataset
>>> from torch.utils.data import DataLoader
>>> data = np.random.rand(10_000)
>>> Dataset.from_dict({"data": data}).push_to_hub("<username>/my_dataset")# Upload to the Huggin
>>> my_iterable_dataset = load_dataset("<username>/my_dataset", streaming=True, split="train")
>>> dataloader = DataLoader(my_iterable_dataset, batch_size=32)
```

If the dataset is split in several shards (i.e. if the dataset consists of multiple data files), then
bạn có thể phát song song bằng cách sử dụng num_workers :

```
>>> my_iterable_dataset = load_dataset("deepmind/code_contests", streaming=True, split="train")
>>> my_iterable_dataset.num_shards
39
>>> dataloader = DataLoader(my_iterable_dataset, batch_size=32, num_workers=4)
```

Trong trường hợp này, mỗi nhân viên được cấp một tập hợp con của danh sách các phân đoạn để truyền phá

Điểm kiểm tra và sơ yếu lý lịch

Nếu bạn cần một DataLoader mà bạn có thể kiểm tra và tiếp tục trong quá trình đào tạo, bạn có thể sử dụng StatefulDataLoader từ torchdata:

```
>>> from torchdata.stateful_dataloader import StatefulDataLoader
>>> my_iterable_dataset = load_dataset("deepmind/code_contests", streaming=True, split="train")
>>> dataloader = StatefulDataLoader(my_iterable_dataset, batch_size=32, num_workers=4)
>>> # save in the middle of training
>>> state_dict = dataloader.state_dict()
>>> # and resume later
>>> dataloader.load_state_dict(state_dict)
```

This is possible thanks to IterableDataset.state_dict() and IterableDataset.load_state_dict().

phân phối

Để phân chia tập dữ liệu của bạn cho các nút đào tạo, bạn có thể sử dụng datasets.distributed.split_dataset_by_node():

```
hệ điều hành nhập khẩu
từ tập dữ liệu. nhập phân phối chia_dataset_by_node

ds = split_dataset_by_node(ds, rank=int(os.environ["RANK"]), world_size=int(os.environ["WORLD_SIZE
```

Điều này hoạt động cho cả tập dữ liệu kiểu bản đồ và tập dữ liệu có thể lặp lại.
Tập dữ liệu được phân chia cho nút ở thứ hạng xếp hạng trong nhóm các nút có kích thước world_size .

Đối với bộ dữ liệu kiểu bản đồ:

Mỗi nút được gán một đoạn dữ liệu, ví dụ: hạng 0 được cấp đoạn đầu tiên của tập dữ liệu.

Đối với bộ dữ liệu có thể lặp lại:

If the dataset has a number of shards that is a factor of world_size (i.e. if dataset.num_shards % world_size == 0),
sau đó các phân đoạn được phân bổ đều trên các nút, được tối ưu hóa nhất.
Mặt khác, mỗi nút giữ 1 ví dụ ngoài world_size, bỏ qua các ví dụ khác.

Điều này cũng có thể được kết hợp với torch.utils.data.DataLoader nếu bạn muốn mỗi nút sử dụng nhiều công nhân để tải dữ liệu.