Sử dụng bộ dữ liệu với TensorFlow

Tài liệu này là phần giới thiệu nhanh về cách sử dụng bộ dữ liệu với TensorFlow, với một mục đích cụ thể tập trung vào cách để có được
các đối tượng tf.Tensor ra khỏi tập dữ liệu của chúng tôi và cách truyền dữ liệu từ Tập dữ liệu ôm mặt đối tượng với các phương thức Keras
like model.fit() .

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 tenxơ TensorFlow, bạn có thể đặt định dạng của tập dữ liệu thành tf :

```
>>> from datasets import Dataset
>>> data = [[1, 2],[3, 4]]
>>> ds = Dataset.from_dict({"data": data})
>>> ds = ds.with_format("tf")
>>> ds[0]
{'data': <tf.Tensor: shape=(2,), dtype=int64, numpy=array([1, 2])>}
>>> ds[:2]
{'data': <tf.Tensor: shape=(2, 2), dtype=int64, numpy=
array([[1, 2],
       [3, 4]])>}
```

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

Điều này có thể hữu ích khi chuyển đổi tập dữ liệu của bạn thành một đối tượng Tensor hoặc để viết một máy phát điện để tải TF
mẫu từ nó. Nếu bạn muốn chuyển đổi toàn bộ tập dữ liệu sang Tensor, chỉ cần truy vấn toàn bộ tập dữ liệu:

```
>>> ds[:]
{'data': <tf.Tensor: shape=(2, 2), dtype=int64, numpy=
array([[1, 2],
       [3, 4]])>}
```

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("tf")
>>> ds[0]
{'data': <tf.Tensor: shape=(2, 2), dtype=int64, numpy=
 array([[1, 2],
        [3, 4]])>}
```

Mặt khác, tập dữ liệu được định dạng TensorFlow sẽ tạo ra RaggedTensor thay vì một tenxơ đơn:

```
>>> 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': <tf.RaggedTensor [[1, 2], [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("tf")
>>> ds[0]
{'data': <tf.Tensor: shape=(2, 2), dtype=int64, numpy=
  array([[1, 2],
         [3, 4]])>}
>>> ds[:2]
{'data': <tf.Tensor: shape=(2, 2, 2), dtype=int64, numpy=
  array([[[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("tf")
>>> ds[:3]
{'label': <tf.Tensor: shape=(3,), dtype=int64, numpy=array([0, 0, 1])>}
```

Chuỗi và đối tượng nhị phân cũng được hỗ trợ:

```
>>> from datasets import Dataset, Features
>>> text = ["foo", "bar"]
>>> data = [0, 1]
>>> ds = Dataset.from_dict({"text": text, "data": data})
>>> ds = ds.with_format("tf")
>>> ds[:2]
{'text': <tf.Tensor: shape=(2,), dtype=string, numpy=array([b'foo', b'bar'], dtype=object)>,
 'data': <tf.Tensor: shape=(2,), dtype=int64, numpy=array([0, 1])>}
```

Bạn cũng có thể định dạng rõ ràng các cột nhất định và giữ nguyên các cột khác:

```
>>> ds = ds.with_format("tf", columns=["data"], output_all_columns=True)
>>> ds[:2]
{'data': <tf.Tensor: shape=(2,), dtype=int64, numpy=array([0, 1])>,
 'text': ['foo', 'bar']}
```

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("tf")
>>> ds[0]
{'image': <tf.Tensor: shape=(512, 512, 4), dtype=uint8, numpy=
  array([[[255, 215, 106, 255],
          [255, 215, 106, 255],
          None
          [255, 255, 255, 255],
          [255, 255, 255, 255]]], dtype=uint8)>}
>>> ds[:2]
{'image': <tf.Tensor: shape=(2, 512, 512, 4), dtype=uint8, numpy=
  array([[[[255, 215, 106, 255],
           [255, 215, 106, 255],
           None
           [255, 255, 255, 255],
           [255, 255, 255, 255]]]], dtype=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("tf")
>>> ds[0]["audio"]["array"]
<tf.Tensor: shape=(202311,), dtype=float32, numpy=
array([ 6.1035156e-05,1.5258789e-05,1.6784668e-04, ...,
          -1.5258789e-05, -1.5258789e-05,1.5258789e-05], dtype=float32)>
>>> ds[0]["audio"]["sampling_rate"]
<tf.Tensor: shape=(), dtype=int32, numpy=44100>
```

Đang tải dữ liệu

Mặc dù bạn có thể tải các mẫu và lô riêng lẻ chỉ bằng cách lập chỉ mục vào tập dữ liệu của mình, nhưng điều
sẽ không hoạt động nếu bạn muốn
to use Keras methods like fit() and predict() . You could write a generator function that
xáo trộn và tải các đợt
from your dataset and fit() on that, but that sounds like a lot of unnecessary work. Instead, if
bạn muốn phát trực tuyến
dữ liệu từ tập dữ liệu của bạn một cách nhanh chóng, chúng tôi khuyên bạn nên chuyển đổi tập dữ liệu của
tf.data.Dataset bằng cách sử dụng
to_tf_dataset() method.

Lớp tf.data.Dataset bao gồm nhiều trường hợp sử dụng - nó thường được tạo từ Tensors
trong bộ nhớ hoặc sử dụng chức năng tải để đọc tệp trên đĩa
or external storage. The dataset can be transformed arbitrarily with the map() method, or
methods like batch()
and shuffle() can be used to create a dataset that's ready for training. These methods do not
sửa đổi dữ liệu được lưu trữ
theo bất kỳ cách nào - thay vào đó, các phương thức xây dựng một biểu đồ đường dẫn dữ liệu sẽ được thực
tập dữ liệu được lặp đi lặp lại,
usually during model training or inference. This is different from the map() method of Hugging
Các đối tượng Bộ dữ liệu khuôn mặt,
chạy chức năng bản đồ ngay lập tức và lưu các cột mới hoặc đã thay đổi.

Vì toàn bộ quy trình tiền xử lý dữ liệu có thể được biên dịch trong tf.data.Dataset , nên điều này
Cách tiếp cận cho phép ồ ạt
tải và huấn luyện dữ liệu song song, không đồng bộ. Tuy nhiên, yêu cầu về đồ thị
việc biên soạn có thể là một hạn chế,
particularly for Hugging Face tokenizers, which are usually not (yet!) compilable as part of a TF
đồ thị. Kết quả là,
chúng tôi thường khuyên bạn nên xử lý trước tập dữ liệu dưới dạng tập dữ liệu Ôm khuôn mặt, nếu tùy ý
Các hàm Python có thể
used, and then converting to tf.data.Dataset afterwards using to_tf_dataset() to get a
tập dữ liệu theo đợt đã sẵn sàng cho
đào tạo. Để xem các ví dụ về phương pháp này, vui lòng xem các ví dụ hoặc sổ ghi chép về

máy biến áp .

Using to_tf_dataset()

Using to_tf_dataset() is straightforward. Once your dataset is preprocessed and ready,
chỉ cần gọi nó như vậy:

```
>>> from datasets import Dataset
>>> data = {"inputs": [[1, 2],[3, 4]], "labels": [0, 1]}
>>> ds = Dataset.from_dict(data)
>>> tf_ds = ds.to_tf_dataset(
                columns=["inputs"],
                label_cols=["labels"],
                batch_size=2,
                shuffle=True
                )
```

Đối tượng tf_ds được trả về ở đây hiện đã hoàn toàn sẵn sàng để huấn luyện và có thể được chuyển trực tiếp
model.fit() . Note
rằng bạn đặt kích thước lô khi tạo tập dữ liệu và do đó bạn không cần chỉ định nó khi
calling fit() :

```
>>> model.fit(tf_ds, epochs=2)
```

For a full description of the arguments, please see the to_tf_dataset() documentation. In many
trường hợp,
bạn cũng sẽ cần thêm collate_fn vào cuộc gọi của mình. Đây là một chức năng có nhiều
các phần tử của tập dữ liệu
và kết hợp chúng thành một đợt duy nhất. Khi tất cả các phần tử có cùng độ dài, phần tử tích hợp
trình đối chiếu mặc định sẽ
đủ, nhưng đối với các tác vụ phức tạp hơn, có thể cần một bộ đối chiếu tùy chỉnh. Đặc biệt, nhiều
nhiệm vụ có mẫu
với độ dài chuỗi khác nhau sẽ yêu cầu một bộ đối chiếu dữ liệu có thể đệm các lô một cách chính xác.
Bạn có thể xem các ví dụ
về điều này trong các ví dụ NLP của máy biến áp và
sổ ghi chép, trong đó độ dài chuỗi thay đổi là rất phổ biến.

Nếu bạn thấy việc tải bằng to_tf_dataset chậm, bạn cũng có thể sử dụng num_workers
lý lẽ. Vòng quay này
lên nhiều quy trình con để tải dữ liệu song song. Tính năng này mới xuất hiện gần đây và vẫn còn phần nào
thử nghiệm - vui lòng gửi
một vấn đề nếu bạn gặp bất kỳ lỗi nào trong khi sử dụng nó!

Khi nào nên sử dụng to_tf_dataset

Người đọc tinh tế có thể nhận thấy ở điểm này rằng chúng tôi đã đưa ra hai cách tiếp cận để
đạt được mục tiêu tương tự - nếu bạn
muốn chuyển tập dữ liệu của mình sang mô hình TensorFlow, bạn có thể chuyển đổi tập dữ liệu thành mô hì
Tensor hoặc dict của Tensors
using .with_format('tf') , or you can convert the dataset to a tf.data.Dataset with
to_tf_dataset() . Either of these
can be passed to model.fit() , so which should you choose?

Điều quan trọng cần nhận ra là khi bạn chuyển đổi toàn bộ tập dữ liệu sang Tensor s, nó ở dạng tĩnh
và nạp đầy đủ vào
ĐẬP. Điều này đơn giản và thuận tiện, nhưng nếu áp dụng bất kỳ điều nào sau đây, có lẽ bạn nên sử dụng
to_tf_dataset()
thay vì:

- Your dataset is too large to fit in RAM. to_tf_dataset() streams only one batch at a time,
  vì vậy thậm chí rất lớn
  bộ dữ liệu có thể được xử lý bằng phương pháp này.
- You want to apply random transformations using dataset.with_transform() or the
  đối chiếu_fn . Đây là
  phổ biến ở một số phương thức, chẳng hạn như tăng cường hình ảnh khi đào tạo các mô hình thị giác,
  hoặc che giấu ngẫu nhiên khi tập luyện
  masked language models. Using to_tf_dataset() will apply those transformations
  tại thời điểm một lô được tải, có nghĩa là các mẫu giống nhau sẽ có kết quả khác nhau
  tăng cường mỗi lần
  chúng đã được tải. Đây thường là những gì bạn muốn.
- Dữ liệu của bạn có thứ nguyên thay đổi, chẳng hạn như văn bản đầu vào trong NLP bao gồm nhiều thứ r
  số lượng token. Khi bạn tạo một lô gồm các mẫu có kích thước thay đổi,
  giải pháp tiêu chuẩn là

đệm các mẫu ngắn hơn theo chiều dài của mẫu dài nhất. Khi bạn truyền mẫu từ
một tập dữ liệu với to_tf_dataset ,
bạn có thể áp dụng phần đệm này cho từng lô thông qua collate_fn của mình. Tuy nhiên, nếu bạn muốn
chuyển thành
một tập dữ liệu như vậy thành Tensor dày đặc, thì bạn sẽ phải đệm các mẫu theo chiều dài của
mẫu dài nhất trong
toàn bộ tập dữ liệu! Điều này có thể tạo ra một lượng lớn phần đệm, gây lãng phí bộ nhớ và
làm giảm tốc độ mô hình của bạn.

Hãy cẩn thận và hạn chế

Right now, to_tf_dataset() always returns a batched dataset - we will add support for
bộ dữ liệu chưa được sắp xếp sớm!