Pytorch TensorFlow

Ask a question Open in Colab

Собираем все воедино

В последних нескольких разделах мы старались делать большую часть работы вручную. Мы изучили, как работают токенизаторы, рассмотрели токенизацию, преобразование во входные идентификаторы, дополнении, усечении и маски внимания.

Однако, как мы видели в разделе 2, 🤗 Transformers API может обработать все это для нас с помощью высокоуровневой функции, которую мы рассмотрим здесь. Когда вы вызываете свой `tokenizer` непосредственно на предложении, вы получаете обратно входы, готовые к передаче в вашу модель:

```
from transformers import AutoTokenizer

checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)

sequence = "I've been waiting for a HuggingFace course my whole life."

model_inputs = tokenizer(sequence)
```

Здесь переменная `model_inputs` содержит все, что необходимо для нормальной работы модели. Для DistilBERT это идентификаторы входов, а также маска внимания. В других моделях, принимающих дополнительные входы, они также будут формироваться выходами объекта `tokenizer`.

Как мы увидим на нескольких примерах ниже, это очень мощный метод. Во-первых, он может токенизировать одну последовательность:

```
sequence = "I've been waiting for a HuggingFace course my whole life."

model_inputs = tokenizer(sequence)
```

Он также обрабатывает несколько последовательностей одновременно, без каких-либо изменений в API:

```
sequences = ["I've been waiting for a HuggingFace course my whole life.", "So have I!"]

model_inputs = tokenizer(sequences)
```

Он может быть дополнен исходя из нескольких целей:

```
# Дополнение последовательностей до максимальной длины последовательности
model_inputs = tokenizer(sequences, padding="longest")

# Дополнение последовательностей до максимальной длины модели
# (512 для BERT или DistilBERT)
model_inputs = tokenizer(sequences, padding="max_length")

# Дополнение последовательностей до заданной максимальной длины
model_inputs = tokenizer(sequences, padding="max_length", max_length=8)
```

Он также может выполнять усечение последовательностей:

```
sequences = ["I've been waiting for a HuggingFace course my whole life.", "So have I!"]

# Усечение последовательностей, длина которых превышает максимальную длину модели
# (512 для BERT или DistilBERT)
model_inputs = tokenizer(sequences, truncation=True)

# Усечение последовательностей, длина которых превышает заданную максимальную длину
model_inputs = tokenizer(sequences, max_length=8, truncation=True)
```

Объект `tokenizer` может выполнять преобразование в тензоры конкретных фреймворков, которые затем могут быть напрямую переданы в модель. Например, в следующем примере кода мы задаем токенизатору возвращать тензоры для различных фреймворков - `"pt"` возвращает тензоры PyTorch, `"tf"` возвращает тензоры TensorFlow, а `"np"` возвращает массивы NumPy:

```
sequences = ["I've been waiting for a HuggingFace course my whole life.", "So have I!"]

# Вернуть тензоры PyTorch
model_inputs = tokenizer(sequences, padding=True, return_tensors="pt")

# Вернуть тензоры TensorFlow
model_inputs = tokenizer(sequences, padding=True, return_tensors="tf")

# Вернуть массивы NumPy
model_inputs = tokenizer(sequences, padding=True, return_tensors="np")
```

Специальные токены

Если мы посмотрим на идентификаторы входа, возвращаемые токенизатором, то увидим, что они немного отличаются от тех, что мы получали ранее:

```
sequence = "I've been waiting for a HuggingFace course my whole life."

model_inputs = tokenizer(sequence)
print(model_inputs["input_ids"])

tokens = tokenizer.tokenize(sequence)
ids = tokenizer.convert_tokens_to_ids(tokens)
print(ids)
```

```
[101, 1045, 1005, 2310, 2042, 3403, 2005, 1037, 17662, 12172, 2607, 2026, 2878, 2166, 1012, 102]
[1045, 1005, 2310, 2042, 3403, 2005, 1037, 17662, 12172, 2607, 2026, 2878, 2166, 1012]
```

Один идентификатор токена был добавлен в начале, а другой - в конце. Давайте декодируем две последовательности идентификаторов, приведенные выше, чтобы понять, в чем дело:

```
print(tokenizer.decode(model_inputs["input_ids"]))
print(tokenizer.decode(ids))
```

```
"[CLS] i've been waiting for a huggingface course my whole life. [SEP]"
"i've been waiting for a huggingface course my whole life."
```

Токенизатор добавил специальное слово `[CLS]` в начале и специальное слово `[SEP]` в конце. Это связано с тем, что модель была предварительно обучена с ними, поэтому для получения тех же результатов при инференсе нам нужно добавить и их. Обратите внимание, что некоторые модели не добавляют специальные слова или добавляют другие слова; модели также могут добавлять эти специальные слова только в начале или только в конце. В любом случае, токенизатор знает, какие из них ожидаются, и справится с этим сам.

Подведение итогов: От токенизатора к модели

Теперь, когда мы рассмотрели все отдельные шаги, которые использует объект `tokenizer` при работе с текстами, давайте в последний раз посмотрим, как он может работать с множественными последовательностями (дополнение!), очень длинными последовательностями (усечение!) и несколькими типами тензоров с помощью своего основного API:

```
import torch
from transformers import AutoTokenizer, AutoModelForSequenceClassification

checkpoint = "distilbert-base-uncased-finetuned-sst-2-english"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
model = AutoModelForSequenceClassification.from_pretrained(checkpoint)
sequences = ["I've been waiting for a HuggingFace course my whole life.", "So have I!"]

tokens = tokenizer(sequences, padding=True, truncation=True, return_tensors="pt")
output = model(**tokens)
```