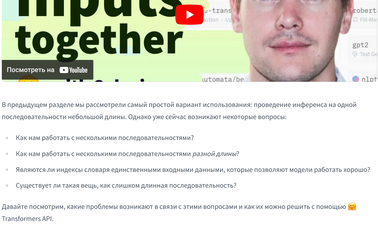
Pytorch TensorFlow

Обработка нескольких последовательностей

Open in Colab



В предыдущем разделе мы рассмотрели самый простой вариант использования: проведение инференса на одной последовательности небольшой длины. Однако уже сейчас возникают некоторые вопросы:

- Как нам работать с несколькими последовательностями?
- Как нам работать с несколькими последовательностями *разной длины*?
- Являются ли индексы словаря единственными входными данными, которые позволяют модели работать хорошо?
- Существует ли такая вещь, как слишком длинная последовательность?

Давайте посмотрим, какие проблемы возникают в связи с этими вопросами и как их можно решить с помощью 🤗 Transformers API.

Модели ожидают батч входов

В предыдущем упражнении вы видели, как последовательности преобразуются в списки чисел. Давайте преобразуем этот список чисел в тензор и передадим его в модель:

```
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)

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

tokens = tokenizer.tokenize(sequence)
ids = tokenizer.convert_tokens_to_ids(tokens)
input_ids = torch.tensor(ids)
# Эта строка вызовет ошибку.
model(input_ids)
```

```
IndexError: Dimension out of range (expected to be in range of [-1, 0], but got 1)
```

О нет! Почему это не удалось? Мы следовали шагам из конвейера в разделе 2.

Проблема в том, что мы отправили в модель одну последовательность, в то время как 🤗 модели Transformers по умолчанию ожидают несколько предложений. Здесь мы попытались сделать все то, что токенизатор делал за кулисами, когда мы применяли его к `последовательности`. Но если вы *приглядитесь*, то увидите, что токенизатор не просто преобразовал список входных идентификаторов в тензор, а добавил к нему еще одно измерение:

```
tokenized_inputs = tokenizer(sequence, return_tensors="pt")
print(tokenized_inputs["input_ids"])
```

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

Давайте попробуем еще раз и добавим новое измерение:

```
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)

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

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

input_ids = torch.tensor([ids])
print("Input IDs:", input_ids)

output = model(input_ids)
print("Logits:", output.logits)
```

Мы выводим входные идентификаторы, а также результирующие логиты - вот результат:

```
Input IDs: [[ 1045,  1005,  2310,  2042,  3403,  2005,  1037, 17662, 12172,  2607, 2026,  2878,  2166,  1
Logits: [[-2.7276,  2.8789]]
```

Батчинг - это отправка нескольких предложений через модель одновременно. Если у вас есть только одно предложение, вы можете просто создать батч с одной последовательностью:

```
batched_ids = [ids, ids]
```

Это батч из двух одинаковых последовательностей!

> ✏️ **Попробуйте!** Преобразуйте этот список `batched_ids` в тензор и пропустите его через вашу модель. Проверьте, что вы получаете те же логиты, что и раньше (но дважды)!

Батчинг позволяет модели работать, когда вы подаете ей несколько последовательностей. Использование нескольких последовательностей так же просто, как и создание батча с одной последовательностью. Однако есть и вторая проблема. Когда вы пытаетесь собрать в батч два (или более) предложения, они могут быть разной длины. Если вы когда-нибудь работали с тензорами, то знаете, что они должны иметь прямоугольную форму, поэтому вы не сможете напрямую преобразовать список входных идентификаторов в тензор. Чтобы обойти эту проблему, мы обычно прибегаем к *дополнению (pad)* входных данных.

Дополнение входов

Следующий список списков не может быть преобразован в тензор:

```
batched_ids = [
    [200, 200, 200],
    [200, 200]
]
```

Чтобы обойти эту проблему, мы будем использовать *дополнение (padding)*, чтобы придать тензорам прямоугольную форму. Дополнение обеспечивает одинаковую длину всех предложений, добавляя специальное слово *токен дополнения* к предложениям с меньшим количеством значений. Например, если у вас есть 10 предложений с 10 словами и 1 предложение с 20 словами, то для дополнения все предложения будут состоять из 20 слов. В нашем примере результирующий тензор выглядит следующим образом:

```
padding_id = 100

batched_ids = [
    [200, 200, 200],
    [200, 200, padding_id],
]
```

Идентификатор токена дополнения можно найти в `tokenizer.pad_token_id`. Давайте используем его и отправим наши два предложения в модель по отдельности и батчем:

```
model = AutoModelForSequenceClassification.from_pretrained(checkpoint)

sequence1_ids = [[200, 200, 200]]
sequence2_ids = [[200, 200]]
batched_ids = [
    [200, 200, 200],
    [200, 200, tokenizer.pad_token_id],
]

print(model(torch.tensor(sequence1_ids)).logits)
print(model(torch.tensor(sequence2_ids)).logits)
print(model(torch.tensor(batched_ids)).logits)
```

```
tensor([[ 1.5694, -1.3895]], grad_fn=<AddmmBackward>)
tensor([[ 0.5803, -0.4125]], grad_fn=<AddmmBackward>)
tensor([[ 1.5694, -1.3895],
        [ 1.3373, -1.2163]], grad_fn=<AddmmBackward>)
```

Что-то не так с логитами в наших батчах: во втором ряду должны быть те же логиты, что и для второго предложения, но мы получили совершенно другие значения!

Это связано с тем, что ключевой особенностью моделей Transformer являются слои внимания (attention layers), которые *контекстуализируют* каждый токен. Они учитывают токены дополнений, так как рассматривают все токены последовательности. Чтобы получить одинаковый результат при прохождении через модель отдельных предложений разной длины или при прохождении батча с одинаковыми предложениями и дополнениями, нам нужно указать слоям внимания игнорировать дополняющие токены. Для этого используется маска внимания (attention mask).

Маски внимания

Маски внимания (Attention masks) - это тензоры той же формы, что и тензор входных идентификаторов, заполненные 0 и 1: 1 означает, что соответствующие токены должны "привлекать внимание", а 0 означает, что соответствующие токены не должны "привлекать внимание" (т.е. должны игнорироваться слоями внимания модели).

Дополним предыдущий пример маской внимания:

```
batched_ids = [
    [200, 200, 200],
    [200, 200, tokenizer.pad_token_id],
]

attention_mask = [
    [1, 1, 1],
    [1, 1, 0],
]

outputs = model(torch.tensor(batched_ids), attention_mask=torch.tensor(attention_mask))
print(outputs.logits)
```

```
tensor([[ 1.5694, -1.3895],
        [ 0.5803, -0.4125]], grad_fn=<AddmmBackward>)
```

Теперь мы получим такие же логиты для второго предложения в батче.

Обратите внимание, что последнее значение второй последовательности - это идентификатор дополнения (padding ID), который в маске внимания имеет значение 0.

> ✏️ **Попробуйте!** Примените токенизацию вручную к двум предложениям, использованным в разделе 2 ("I've been waiting for a HuggingFace course my whole life." и "I hate this so much!"). Пропустите их через модель и проверьте, что вы получите те же логиты, что и в разделе 2. Теперь объедините их в батч с использованием токена дополнения, а затем создайте соответствующую маску внимания. Проверьте, что при прохождении через модель вы получаете те же результаты!

Более длинные последовательности

В моделях Transformer существует ограничение на длину последовательностей, которые мы можем передавать моделям. Большинство моделей работают с последовательностями длиной до 512 или 1024 токенов и терпят крах при необходимости обработки более длинных последовательностей. Есть два решения этой проблемы:

- Использовать модель с большей поддерживаемой длиной последовательности.
- Усечение (truncate) последовательностей.

Модели имеют разную поддерживаемую длину последовательности, а некоторые специализируются на работе с очень длинными последовательностями. Одним из примеров является Longformer, а другим - LED. Если вы работаете над задачей, требующей очень длинных последовательностей, мы рекомендуем вам обратить внимание на эти модели.

В противном случае мы рекомендуем использовать усечение последовательностей, указав параметр `max_sequence_length`:

```
sequence = sequence[:max_sequence_length]
```