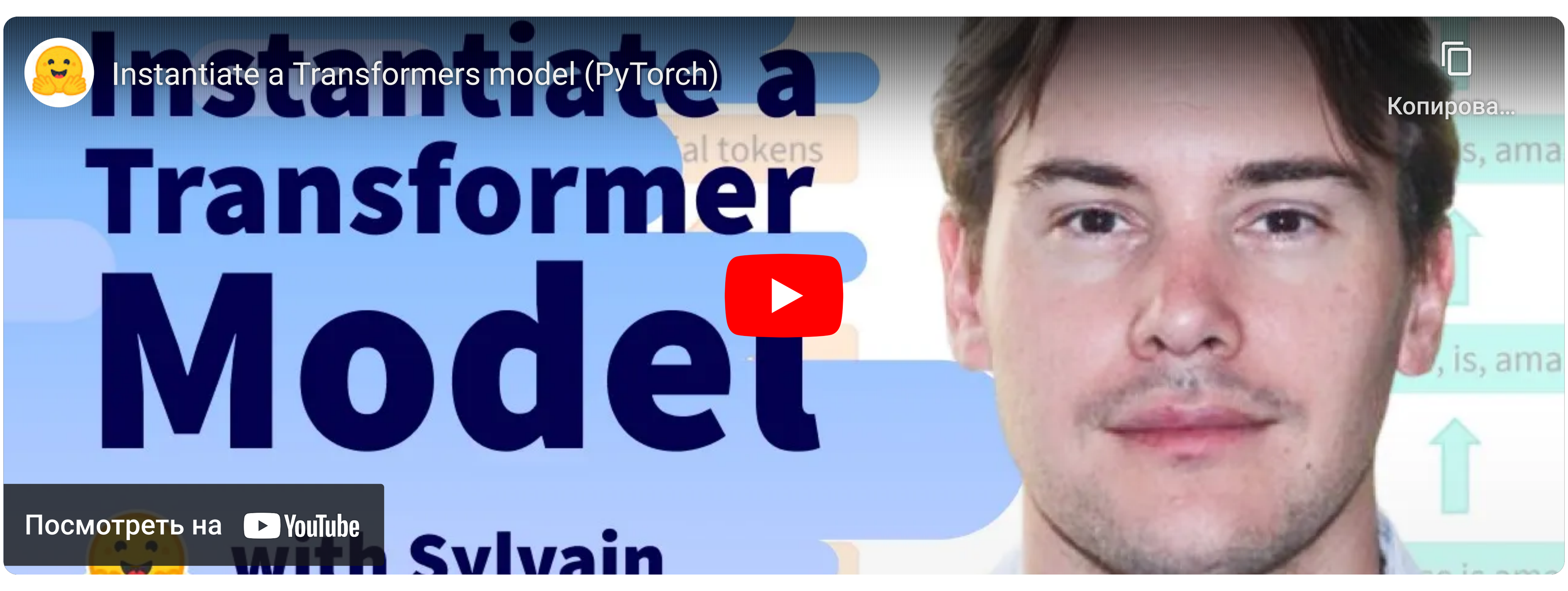
Pytorch TensorFlow Ask a question Open in Colab

Модели



В этом разделе мы подробно рассмотрим создание и использование модели. Мы будем использовать класс `AutoModel`, который удобен, когда вы хотите инстанцировать любую модель из контрольной точки.

Класс `AutoModel` и все его представители на самом деле являются простыми обертками для широкого спектра моделей, доступных в библиотеке. Это умная обертка, поскольку она может автоматически определить архитектуру модели, подходящую для вашей контрольной точки, а затем инстанцировать модель с этой архитектурой.

Однако если вы знаете тип модели, которую хотите использовать, вы можете напрямую использовать класс, определяющий ее архитектуру. Давайте рассмотрим, как это работает на примере модели BERT.

Создание Transformer

Первое, что нам нужно сделать для инициализации модели BERT, - загрузить объект конфигурации:

```
from transformers import BertConfig, BertModel

# Создание конфигурации
config = BertConfig()

# Создание модели на основе конфигурации
model = BertModel(config)
```

Конфигурация содержит множество атрибутов, которые используются для создания модели:

```
print(config)
```

```
BertConfig {
  [...]
  "hidden_size": 768,
  "intermediate_size": 3072,
  "max_position_embeddings": 512,
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  [...]
}
```

Хотя вы еще не видели, что делают все эти атрибуты, вы должны узнать некоторые из них: атрибут `hidden_size` определяет размер вектора `hidden_states`, а `num_hidden_layers` определяет количество слоев в модели Transformer.

Различные методы загрузки

При создании модели из конфигурации по умолчанию она инициализируется случайными значениями:

```
from transformers import BertConfig, BertModel

config = BertConfig()
model = BertModel(config)

# Модель инициализируется случайным образом!
```

Модель можно использовать и в таком состоянии, но она будет выдавать тарабарщину; сначала ее нужно обучить. Мы могли бы обучить модель с нуля для конкретной задачи, но, как вы видели в Главе 1, это потребовало бы много времени и большого количества данных, а также оказало бы немалое влияние на окружающую среду. Чтобы избежать ненужных и дублирующих усилий, крайне важно иметь возможность обмениваться уже обученными моделями и повторно их использовать.

Загрузить уже обученную модель Transformer очень просто - мы можем сделать это с помощью метода `from_pretrained()`:

```
from transformers import BertModel

model = BertModel.from_pretrained("bert-base-cased")
```

Как вы видели ранее, мы можем заменить `BertModel` на эквивалентный класс `AutoModel`. В дальнейшем мы будем поступать именно так, поскольку таким образом мы получаем код, не зависящий от контрольных точек; если ваш код работает на одной контрольной точке, он должен без проблем работать и на другой. Это касается даже разных архитектур, если контрольная точка была обучена для схожей задачи (например, задачи анализа настроений).

В приведенном выше примере кода мы не использовали `BertConfig`, а вместо этого загрузили предварительно обученную модель через идентификатор `bert-base-cased`. Это контрольная точка модели, которая была обучена самими авторами BERT; более подробную информацию о ней можно найти в ее карточке модели.

Теперь эта модель инициализирована всеми весами контрольной точки. Ее можно использовать непосредственно для инференса на задачах, для которых она была обучена, а также для дообучения на новой задаче. Обучаясь с предварительно подготовленными весами, а не с нуля, мы можем быстро добиться хороших результатов.

Веса были загружены и кэшированы (чтобы последующие вызовы метода `from_pretrained()` не загружали их заново) в папке кэша, которая по умолчанию находится в *~/.cache/huggingface/transformers*. Вы можете настроить папку кэша, установив переменную окружения `HF_HOME`.

Идентификатор, используемый для загрузки модели, может быть идентификатором любой модели на Model Hub, если она совместима с архитектурой BERT. Полный список доступных контрольных точек BERT можно найти здесь.

Методы сохранения

Сохранить модель так же просто, как и загрузить ее - мы используем метод `save_pretrained()`, который аналогичен методу `from_pretrained()`:

```
model.save_pretrained("directory_on_my_computer")
```

При этом на диск сохраняются два файла:

```
ls directory_on_my_computer

config.json pytorch_model.bin
```

Если вы посмотрите на файл *config.json*, то узнаете атрибуты, необходимые для построения архитектуры модели. Этот файл также содержит некоторые метаданные, такие как место создания контрольной точки и версию 🤗 Transformers, которую вы использовали при последнем сохранении контрольной точки.

Файл *pytorch_model.bin* известен как *словарь состояний (state dictionary)*; он содержит все веса вашей модели. Эти два файла неразрывно связаны друг с другом; конфигурация необходима для того, чтобы знать архитектуру модели, а веса модели - это ее параметры.

Использование модели Transformer для инференса

Теперь, когда вы знаете, как загружать и сохранять модель, давайте попробуем использовать ее для прогнозирования. Модели Transformer могут обрабатывать только числа - числа, которые генерирует токенизатор. Но прежде чем мы обсудим токенизаторы, давайте узнаем, какие входные данные (входы) принимает модель.

Токенизаторы могут позаботиться о приведении входных данных к тензорам соответствующего фреймворка, но чтобы помочь вам понять, что происходит, мы кратко рассмотрим, что нужно сделать перед передачей входных данных в модель.

Допустим, у нас есть несколько последовательностей:

```
sequences = ["Hello!", "Cool.", "Nice!"]
```

Токенизатор преобразует их в индексы словаря, которые обычно называются *идентификаторами входов (input IDs)*. Теперь каждая последовательность представляет собой список чисел! В результате на выходе получаем:

```
encoded_sequences = [
    [101, 7592, 999, 102],
    [101, 4658, 1012, 102],
    [101, 3835, 999, 102],
]
```

Это список закодированных последовательностей: список списков. Тензоры принимают только прямоугольную форму (подумайте о матрицах). Этот “массив” уже имеет прямоугольную форму, поэтому преобразовать его в тензор очень просто:

```
import torch

model_inputs = torch.tensor(encoded_sequences)
```

Использование тензоров в качестве входов в модель

Использовать тензоры с моделью очень просто - мы просто вызываем модель с входами:

```
output = model(model_inputs)
```

Хотя модель принимает множество различных аргументов, только идентификаторы входов являются необходимыми. О том, что делают остальные аргументы и когда они нужны, мы расскажем позже, но сначала нам нужно подробнее рассмотреть токенизаторы, которые формируют входные данные (входы), которые может понять модель Transformer.