```
1 !pip install -q transformers datasets tokenizers
```

```
.1 MB 5.4 MB/s
90 kB 49.4 MB/s
.3 MB 32.8 MB/s
96 kB 44.8 MB/s
95 kB 46.8 MB/s
9 kB 6.3 MB/s
32 kB 45.2 MB/s
.1 MB 35.7 MB/s
43 kB 49.5 MB/s
92 kB 48.1 MB/s
71 kB 43.3 MB/s
60 kB 44.5 MB/s
```


До mers.

Прив теперь ты познакомишься с моделя
транс . В этом задании будет предложено решить
с пом лиз сентимента) и сделать небольшое
иссле

```
1 im
2 im
3 fr
4
5 import torch
6 import torch.nn as nn
7
8 from transformers import GPT2ForSequenceClassification, GPT2TokenizerFast, GPT2Config
9 from datasets import load_dataset
```

```
1 device = "cuda" if torch.cuda.is_available else "cpu"
```

```
1 !nvidia-smi
```

```
Thu Nov 18 19:09:38 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 495.44       Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K80           Off  | 00000000:00:04.0 Off |                    0 |
| N/A   73C    P0    73W / 149W |   7879MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
```

```
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Датасет, который мы будем использовать сегодня – тексты из английского твиттера. Они уже почищенны от никнеймов, знаков препинания и прочего.

```
1 emotion_dataset = load_dataset("emotion")
```

```
Downloading:                                    3.62k/? [00:00<00:00,
                                                56.7kB/s]
Downloading:                                    3.28k/? [00:00<00:00,
                                                89.4kB/s]
Using custom data configuration default
Downloading and preparing dataset emotion/default (download: 
Downloading:                                    1.66M/1.66M [00:00<00:00,
100%                                            6.72MB/s]
Downloading:                                    204k/204k [00:00<00:00,
100%                                            2.63MB/s]
```

Посмотри, из чего состоит `emotion_dataset`:

```
1 # emotion_dataset
```

```
1 # emotion_dataset["train"]
```

```
1 # emotion_dataset["train"]["text"][0]
```

```
1 # emotion_dataset["train"]["label"][0]
```

```
1 # len(emotion_dataset["train"])
```

Для перевода текста в токены мы будем использовать предобученный BPE-токенайзер.

```
1 tokenizer = GPT2TokenizerFast.from_pretrained("distilgpt2")
2 tokenizer.pad_token = tokenizer.eos_token # У gpt2 нет pad токенов. Вместо них воспольз
```

```
Downloading: 100%    0.99M/0.99M [00:00<00:00, 887kB/s]
Downloading: 100%    446k/446k [00:00<00:00,
```

Подготовь класс, который принимает датасет, токенайзер и имя используемой части (`train`, `validation`, `test`). Используй его для получения данных для обучения.

P.S. Посмотри, как работает токенайзер (docs) и подумай, как его надо добавить в датасет.

Немного примеров, как работает токенайзер. Это поможет с написанием датасета.

```
# tokenizer.tokenize(emotion_dataset["train"]["text"][0])
```

```
# tokenizer.encode(emotion_dataset["train"]["text"][0])
```

```
# tokenizer.encode_plus(emotion_dataset["train"]["text"][0])
```

```
# tokenizer.encode_plus(emotion_dataset["train"]["text"][0], return_tensors="pt")
```

```
tokenizer.encode_plus(
    emotion_dataset["train"]["text"][0],
    max_length=128, # максимальная длина текста
    padding="max_length", # надо ли добавлять паддинг в конце?
    return_tensors="pt", # возвращает pytorch тензоры
)
```

```
{'input_ids': tensor([[   72, 42547,  1254, 42659, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256,
         50256, 50256, 50256, 50256, 50256, 50256, 50256, 50256]]), 'attention_mask'
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0]])}
```

```
# Если надо, попрактикуйся работать с токенайзером здесь
```

```
class TweetDataset(torch.utils.data.Dataset):
    def __init__(self, part, dataset=emotion_dataset, tokenizer=tokenizer, max_length=1
        self.part = part
        self.dataset = dataset
        self.tokenizer = tokenizer
        self.max_length = max_length

        self.labels = np.unique(dataset[part]["label"])
        self.label2num = {l: num for num, l in enumerate(self.labels)}

    def __getitem__(self, idx):
        """
        Return dict with tokens, attention_mask and label
        """
        text = self.dataset[self.part]["text"][idx]
        label = self.dataset[self.part]["label"][idx]

        tokenizer_output = tokenizer.encode_plus(
            text,
            max_length=self.max_length,
            padding="max_length",
            return_tensors="pt")

        target = self.label2num[label]
        return {
            "input_ids": tokenizer_output['input_ids'].squeeze(0),
            "mask": tokenizer_output['attention_mask'].squeeze(0),
            "target": target
        }

    def __len__(self):
        """
        Return length of dataset
        """
        return len(self.dataset[self.part])
```

Создай `train`, `validation` и `test` части датасета. Загрузи их в `DataLoaders`.

```
train_dataset = TweetDataset(part='train')
valid_dataset = TweetDataset(part='validation')
test_dataset = TweetDataset(part='test')
```

```
batch_size = 64 # Задай batch_size

train_loader = torch.utils.data.DataLoader(
    dataset=train_dataset, batch_size=batch_size, shuffle=True
)
valid_loader = torch.utils.data.DataLoader(
    dataset=valid_dataset, batch_size=batch_size, shuffle=False
)
```

```
test_loader = torch.utils.data.DataLoader(
    dataset=test_dataset, batch_size=batch_size, shuffle=False
)
```

```
# batch = next(iter(train_loader))
```

Начнем с нуля.

Попробуем обучить модель трансформер с нуля решать данную задачу.

[ ] ↳ *15 cells hidden*

Fine-tuning

Теперь другой подход: загрузим модель, которая обучалась решать задачу Language Modeling. Посмотрим, получим ли мы прирост в качестве.

[ ] ↳ *7 cells hidden*

Отчет

Покажи здесь, что ты выполнил по этой работе. Ответь на несколько вопросов:

- Какой подход оказался лучше?
  Предобученная модель показала качество лучше (0.92 vs 0.89) и получалось это качество быстрее (6 vs 10 эпох)
- На какие слова модель большего всего обращала внимание?
  Обе модели смотрели примерно на одинаковые слова, но предобученная модель лучше замечала смысловую связь между словами. Так же у второй модели есть головы, которые смотрят просто на предыдущее слово.
- На каких слоях/головах модель обращала внимание?
  На всех слоях видно влияние других слов на слово(тавтология какая-то)

[ ] ↳ *1 cell hidden*