Text Summarization. Homework

Всем привет! Это домашка по суммаризации текста.

На семинаре мы рассмотрели базовые модели для суммаризации текста. Попробуйте теперь улучшить два метода: TextRank и Extractive RNN. Задание достаточно большое и требует хорошую фантазию, тут можно эксперементировать во всю.

Для сдачи заданий надо получить определенное качество по test-у:

- 1 задание: 0.35 BLEU
- 2 задание: 0.35 BLEU

Если ваш подход пробивает это качество – задание считается пройденным. Плюсом будет описание того, почему вы решили использовать то или иное решение.

Датасет: gazeta.ru

P.S. Возможно, в датасете находятся пустые данные. Проверьте эту гипотезу, и если надо, сделайте предобратоку датасета.

`Ноутбук создан на основе семинара Гусева Ильи на кафедре компьютерной лингвистики МФТИ.`

Загрузим датасет и необходимые библиотеки

In [ ]:
```
!wget -q https://www.dropbox.com/s/43l702z5a5i2w8j/gazeta_train.txt
!wget -q https://www.dropbox.com/s/k2egt3sug0hb185/gazeta_val.txt
!wget -q https://www.dropbox.com/s/3gki5n5djs9w0v6/gazeta_test.txt
```

In [ ]:
```
!pip install -Uq razdel allennlp torch fasttext OpenNMT-py networkx pymorphy2 nltk rouge==0.3.1 summa
!pip install -Uq transformers youtokentome
!pip install -U sentence-transformers
```

```
     |████████████████████████████████| 512kB 11.1MB/s
     |████████████████████████████████| 71kB 8.4MB/s
     |████████████████████████████████| 204kB 14.1MB/s
     |████████████████████████████████| 61kB 8.8MB/s
     |████████████████████████████████| 1.4MB 26.8MB/s
     |████████████████████████████████| 61kB 9.1MB/s
     |████████████████████████████████| 317kB 44.8MB/s
     |████████████████████████████████| 133kB 50.1MB/s
     |████████████████████████████████| 266kB 45.8MB/s
     |████████████████████████████████| 1.3MB 52.0MB/s
     |████████████████████████████████| 51kB 7.7MB/s
     |████████████████████████████████| 61kB 9.4MB/s
     |████████████████████████████████| 2.5MB 50.1MB/s
     |████████████████████████████████| 61kB 9.0MB/s
     |████████████████████████████████| 8.2MB 57.9MB/s
     |████████████████████████████████| 71kB 10.5MB/s
     |████████████████████████████████| 7.0MB 53.6MB/s
     |████████████████████████████████| 890kB 51.2MB/s
     |████████████████████████████████| 1.1MB 43.9MB/s
     |████████████████████████████████| 2.9MB 52.6MB/s
  Building wheel for fasttext (setup.py) ... done
  Building wheel for nltk (setup.py) ... done
  Building wheel for summa (setup.py) ... done
  Building wheel for overrides (setup.py) ... done
  Building wheel for jsonnet (setup.py) ... done
  Building wheel for configargparse (setup.py) ... done
  Building wheel for sacremoses (setup.py) ... done
ERROR: botocore 1.19.30 has requirement urllib3<1.27,>=1.25.4; python_version != "3.4", but you'll have urllib3 1.24.3 which is incompatible.
     |████████████████████████████████| 1.4MB 9.1MB/s
     |████████████████████████████████| 1.7MB 29.8MB/s
     |████████████████████████████████| 2.9MB 52.2MB/s
ERROR: allennlp 1.2.2 has requirement transformers<3.6,>=3.4, but you'll have transformers 4.0.0 which is incompatible.
Collecting sentence-transformers
  Downloading https://files.pythonhosted.org/packages/f5/5a/6e41e8383913dd2ba923cdcd02be2e03911595f4d2f9de559ecbed80d2d3/sentence-transformers-0.3.9.tar.gz (64kB)
     |████████████████████████████████| 71kB 5.2MB/s
Collecting transformers<3.6.0,>=3.1.0
  Using cached https://files.pythonhosted.org/packages/3a/83/e74092e7f24a08d751aa59b37a9fc572b2e4af3918cb66f7766c3affb1b4/transformers-3.5.1-py3-none-any.whl
Requirement already satisfied, skipping upgrade: tqdm in /usr/local/lib/python3.6/dist-packages (from sentence-transformers) (4.41.1)
Requirement already satisfied, skipping upgrade: torch>=1.6.0 in /usr/local/lib/python3.6/dist-packages (from sentence-transformers) (1.7.0+cu101)
Requirement already satisfied, skipping upgrade: numpy in /usr/local/lib/python3.6/dist-packages (from sentence-transformers) (1.18.5)
Requirement already satisfied, skipping upgrade: scikit-learn in /usr/local/lib/python3.6/dist-packages (from sentence-transformers) (0.22.2.post1)
Requirement already satisfied, skipping upgrade: scipy in /usr/local/lib/python3.6/dist-packages (from sentence-transformers) (1.4.1)
Requirement already satisfied, skipping upgrade: nltk in /usr/local/lib/python3.6/dist-packages (from sentence-transformers) (3.5)
Collecting tokenizers==0.9.3
  Using cached https://files.pythonhosted.org/packages/4c/34/b39eb9994bc3c999270b69c9eea40ecc6f0e97991dba28282b9fd32d44ee/tokenizers-0.9.3-cp36-cp36m-manylinux1_x86_64.whl
Requirement already satisfied, skipping upgrade: sacremoses in /usr/local/lib/python3.6/dist-packages (from transformers<3.6.0,>=3.1.0->sentence-transformers) (0.0.43)
Requirement already satisfied, skipping upgrade: dataclasses; python_version < "3.7" in /usr/local/lib/python3.6/dist-packages (from transformers<3.6.0,>=3.1.0->sentence-transformers) (0.8)
Requirement already satisfied, skipping upgrade: protobuf in /usr/local/lib/python3.6/dist-packages (from transformers<3.6.0,>=3.1.0->sentence-transformers) (3.12.4)
Requirement already satisfied, skipping upgrade: requests in /usr/local/lib/python3.6/dist-packages (from transformers<3.6.0,>=3.1.0->sentence-transformers) (2.23.0)
Requirement already satisfied, skipping upgrade: regex!=2019.12.17 in /usr/local/lib/python3.6/dist-packages (from transformers<3.6.0,>=3.1.0->sentence-transformers) (2019.12.20)
Requirement already satisfied, skipping upgrade: packaging in /usr/local/lib/python3.6/dist-packages (from transformers<3.6.0,>=3.1.0->sentence-transformers) (20.4)
Requirement already satisfied, skipping upgrade: sentencepiece==0.1.91 in /usr/local/lib/python3.6/dist-packages (from transformers<3.6.0,>=3.1.0->sentence-transformers) (0.1.91)
Requirement already satisfied, skipping upgrade: filelock in /usr/local/lib/python3.6/dist-packages (from transformers<3.6.0,>=3.1.0->sentence-transformers) (3.0.12)
Requirement already satisfied, skipping upgrade: typing-extensions in /usr/local/lib/python3.6/dist-packages (from torch>=1.6.0->sentence-transformers) (3.7.4.3)
Requirement already satisfied, skipping upgrade: future in /usr/local/lib/python3.6/dist-packages (from torch>=1.6.0->sentence-transformers) (0.16.0)
Requirement already satisfied, skipping upgrade: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages (from scikit-learn->sentence-transformers) (0.17.0)
Requirement already satisfied, skipping upgrade: click in /usr/local/lib/python3.6/dist-packages (from nltk->sentence-transformers) (7.1.2)
Requirement already satisfied, skipping upgrade: six in /usr/local/lib/python3.6/dist-packages (from sacremoses->transformers<3.6.0,>=3.1.0->sentence-transformers) (1.15.0)
Requirement already satisfied, skipping upgrade: setuptools in /usr/local/lib/python3.6/dist-packages (from protobuf->transformers<3.6.0,>=3.1.0->sentence-transformers) (50.3.2)
Requirement already satisfied, skipping upgrade: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from requests->transformers<3.6.0,>=3.1.0->sentence-transformers) (2020.11.8)
Requirement already satisfied, skipping upgrade: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (from requests->transformers<3.6.0,>=3.1.0->sentence-transformers) (1.24.3)
Requirement already satisfied, skipping upgrade: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests->transformers<3.6.0,>=3.1.0->sentence-transformers) (2.10)
Requirement already satisfied, skipping upgrade: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from requests->transformers<3.6.0,>=3.1.0->sentence-transformers) (3.0.4)
Requirement already satisfied, skipping upgrade: pyparsing>=2.0.2 in /usr/local/lib/python3.6/dist-packages (from packaging->transformers<3.6.0,>=3.1.0->sentence-transformers) (2.4.7)
Building wheels for collected packages: sentence-transformers
  Building wheel for sentence-transformers (setup.py) ... done
  Created wheel for sentence-transformers: filename=sentence_transformers-0.3.9-cp36-none-any.whl size=101036 sha256=a800f233a142d12338361d21f01b0865a2ea346d91668b524dab156d676d0e23
  Stored in directory: /root/.cache/pip/wheels/fc/89/43/f2f5bc00b03ef9724b0f6254a97eaf159a4c4ddc024b33e07a
Successfully built sentence-transformers
Installing collected packages: tokenizers, transformers, sentence-transformers
  Found existing installation: tokenizers 0.9.4
    Uninstalling tokenizers-0.9.4:
      Successfully uninstalled tokenizers-0.9.4
  Found existing installation: transformers 4.0.0
    Uninstalling transformers-4.0.0:
      Successfully uninstalled transformers-4.0.0
Successfully installed sentence-transformers-0.3.9 tokenizers-0.9.3 transformers-3.5.1
```

In [ ]:
```python
import random
import pandas as pd

def read_gazeta_records(file_name, shuffle=True, sort_by_date=False):
    assert shuffle != sort_by_date
    records = []
    with open(file_name, "r") as r:
        for line in r:
            records.append(eval(line)) # Simple hack
    records = pd.DataFrame(records)
    if sort_by_date:
        records = records.sort("date")
    if shuffle:
        records = records.sample(frac=1)
    return records
```

In [ ]:
```python
train_records = read_gazeta_records("gazeta_train.txt")
val_records = read_gazeta_records("gazeta_val.txt")
test_records = read_gazeta_records("gazeta_test.txt")

device = 'cuda'
```

1 задание: TextRank (порог: 0.35 BLEU)

TextRank - unsupervised метод для составления кратких выжимок из текста. Описание метода:

1. Сплитим текст по предложениям
2. Считаем "похожесть" предложений между собой
3. Строим граф предложений с взвешенными ребрами
4. С помощью алгоритм PageRank получаем наиболее важные предложения, на основе которых делаем summary.

Функция похожести можно сделать и из нейросетевых(или около) моделек: FastText, ELMO и BERT. Выберете один метод, загрузите предобученную модель и с ее помощью для каждого предложениия сделайте sentence embedding. С помощью косинусной меры определяйте похожесть предложений.

Предобученные модели можно взять по ссылке.

In [ ]:
```python
from nltk.translate.bleu_score import corpus_bleu
from rouge import Rouge

def calc_scores(references, predictions, metric="all"):
    print("Count:", len(predictions))
    print("Ref:", references[-1])
    print("Hyp:", predictions[-1])

    if metric in ("bleu", "all"):
        print("BLEU: ", corpus_bleu([[r] for r in references], predictions))
    if metric in ("rouge", "all"):
        rouge = Rouge()
        scores = rouge.get_scores(predictions, references, avg=True)
        print("ROUGE: ", scores)
```

In [ ]:
```python
from transformers import AutoTokenizer, AutoModel
import torch

tokenizer = AutoTokenizer.from_pretrained('sentence-transformers/distilbert-multilingual-nli-stsb-quora-ranking')
encoder = AutoModel.from_pretrained('sentence-transformers/distilbert-multilingual-nli-stsb-quora-ranking')
encoder = encoder.to('cuda')
```

In [ ]:
```python
from itertools import combinations
from sentence_transformers import util
from tqdm import tqdm_notebook as tqdm
import networkx as nx
import numpy as np
import pymorphy2
import razdel


def your_super_words_similarity(sentence1, sentence2):
    similarity = util.pytorch_cos_sim(sentence1, sentence2)
    return similarity

def mean_pooling(model_output, attention_mask):
    token_embeddings = model_output[0] #First element of model_output contains all token embeddings
    input_mask_expanded = attention_mask.unsqueeze(-1).expand(token_embeddings.size()).float()
    sum_embeddings = torch.sum(token_embeddings * input_mask_expanded, 1)
    sum_mask = torch.clamp(input_mask_expanded.sum(1), min=1e-9)
    return sum_embeddings / sum_mask


def gen_text_rank_summary(text, model, tokenizer, calc_similarity=your_super_words_similarity, summary_part=0.1, lower=True, morph=None):
    '''
    Составление summary с помощью TextRank
    '''
    # Разбиваем текст на предложения
    sentences = [sentence.text.lower() for sentence in razdel.sentenize(text)]
    n_sentences = len(sentences)


    # Токенизируем предложения
    max_length = max([len(sentences[i]) for i in range(n_sentences)])
    encoded_sentences = tokenizer(sentences, padding=True, truncation=True, max_length=128, return_tensors='pt').to('cuda')


    with torch.no_grad():
        model_output = model(**encoded_sentences)

    #Perform pooling. In this case, mean pooling
    sentence_embeddings = mean_pooling(model_output, encoded_sentences['attention_mask'])


    # При необходимости лемматизируем слова
    if morph is not None:
        sentences_words = [[morph.parse(word)[0].normal_form for word in words] for words in sentences_words]


    # Для каждой пары предложений считаем близость
    pairs = combinations(range(n_sentences), 2)
    scores = [(i, j, calc_similarity(sentence_embeddings[i], sentence_embeddings[j])) for i, j in pairs]

    # Строим граф с рёбрами, равными близости между предложениями
    g = nx.Graph()
    g.add_weighted_edges_from(scores)

    # Считаем PageRank
    pr = nx.pagerank(g)
    result = [(i, pr[i], s) for i, s in enumerate(sentences) if i in pr]
    result.sort(key=lambda x: x[1], reverse=True)

    # Выбираем топ предложений
    n_summary_sentences = max(int(n_sentences * summary_part), 1)
    result = result[:n_summary_sentences]

    # Восстанавливаем оригинальный их порядок
    result.sort(key=lambda x: x[0])

    # Восстанавливаем текст выжимки
    predicted_summary = " ".join([sentence for i, proba, sentence in result])
    predicted_summary = predicted_summary.lower() if lower else predicted_summary
    return predicted_summary

def calc_text_rank_score(records, model=encoder, tokenizer=tokenizer, calc_similarity=your_super_words_similarity, summary_part=0.07, lower=True, nrows=1000, morph=None):
    references = []
    predictions = []


    for text, summary in tqdm(records[['text', 'summary']].values[:nrows]):
        summary = summary if not lower else summary.lower()
        # text = text if not lower else text.lower()
        references.append(summary)

        predicted_summary = gen_text_rank_summary(text, model, tokenizer, calc_similarity, summary_part, lower, morph=morph)

        predictions.append(predicted_summary)

    calc_scores(references, predictions)
```

In [ ]:
```python
calc_text_rank_score(test_records, calc_similarity=your_super_words_similarity)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:78: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
```

```
Count: 1000
Ref: эдвард сноуден вновь выступил с критикой мировых it-гигантов, сравнив их бизнес-модель с узаконенным посягательством на частную жизнь. кроме того, он раскритиковал общий регламент по защите данных (gdpr), принятый в ес, заявив о том, что это была «хорошая попытка», которая не смогла решить существующие проблемы.
Hyp: проблема не в защите данных, а в сборе данных», — считает эдвард сноуден. если бы я хотел быть в безопасности, я бы все еще сидел на гавайях, зарабатывал огромную кучу денег и шпионил за всеми вами…
BLEU:  0.35680524184287987
ROUGE:  {'rouge-1': {'f': 0.14838756839543601, 'p': 0.15708511708534825, 'r': 0.15305973405267267}, 'rouge-2': {'f': 0.03391559864727089, 'p': 0.03592655148074149, 'r': 0.035106416182252916}, 'rouge-l': {'f': 0.12358441521780654, 'p': 0.1393488184655866, 'r': 0.13548438116016723}}
```

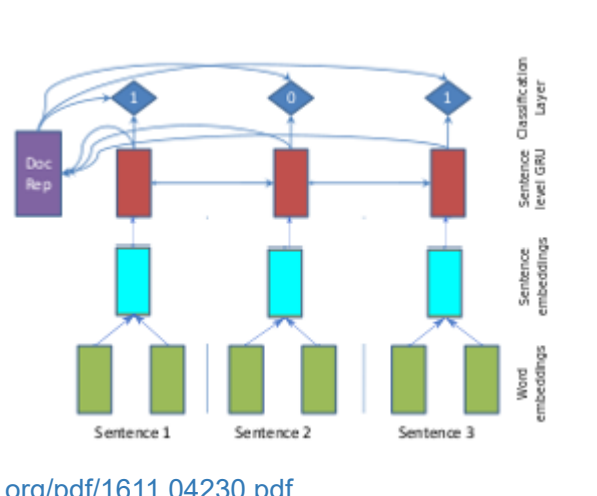
2 Задание: Extractive RNN (порог: 0.35 BLEU)

Второй метод, который вам предлагается улучшить – поиск предложений для summary с помощью RNN. В рассмотренной методе мы использовали LSTM для генерации sentence embedding. Попробуйте использовать другие архитектуры: CNN, Transformer; или добавьте предобученные модели, как и в первом задании.

P.S. Тут предполагается, что придется изменять много кода в ячееках (например, поменять токенизацию).

Модель

Картинка для привлечения внимания:



Статья с оригинальным методом: https://arxiv.org/pdf/1611.04230.pdf

Список вдохновения:

- https://towardsdatascience.com/understanding-how-convolutional-neural-network-cnn-perform-text-classification-with-word-d2ee64b9dd0b Пример того, как можно применять CNN в текстовых задачах
- https://arxiv.org/pdf/1808.08745.pdf Очень крутой метод генерации summary без Transformers
- https://towardsdatascience.com/super-easy-way-to-get-sentence-embedding-using-fasttext-in-python-a70f34ac5b7c – простой метод генерации sentence embedding
- https://towardsdatascience.com/fse-2b1ffa791cf9 – Необычный метод генерации sentence embedding
- https://github.com/UKPLab/sentence-transformers – BERT предобученный для sentence embedding

P.S. Выше написанные ссылки нужны только для разогрева вашей фантазии, можно воспользоваться ими, а можно придумать свой.

Комментарий к заданию: Если посмотреть на архитектуру ~~почти~~ SummaRuNNer, то в ней есть два главных элемента: первая часть, которая читает предложения и возвращает векторы на каждое предложение, и вторая, которая выбирает предложения для суммаризации. Вторую часть мы не трогаем, а первую меняем. На что меняем – как вы решите. Главное: она должна иметь хорошее качество и встроиться в текущую модель.

In [ ]:
```python
import copy
import random

def build_oracle_summary_greedy(text, gold_summary, calc_score, lower=True, max_sentences=30):
    '''
    Жадное построение oracle summary
    '''
    gold_summary = gold_summary.lower() if lower else gold_summary
    # Делим текст на предложения
    sentences = [sentence.text.lower() if lower else sentence.text for sentence in razdel.sentenize(text)][:max_sentences]
    n_sentences = len(sentences)
    oracle_summary_sentences = set()
    score = -1.0
    summaries = []
    for _ in range(min(n_sentences, 2)):
        for i in range(n_sentences):
            if i in oracle_summary_sentences:
                continue
            current_summary_sentences = copy.copy(oracle_summary_sentences)
            # Добавляем какое-то предложения к уже существующему summary
            current_summary_sentences.add(i)
            current_summary = " ".join([sentences[index] for index in sorted(list(current_summary_sentences))])
            # Считаем метрики
            current_score = calc_score(current_summary, gold_summary)
            summaries.append((current_score, current_summary_sentences))
        # Если получилось улучшить метрики с добавлением какого-либо предложения, то пробуем добавить ещё
        # Иначе на этом заканчиваем
        best_summary_score, best_summary_sentences = max(summaries)
        if best_summary_score <= score:
            break
        oracle_summary_sentences = best_summary_sentences
        score = best_summary_score
    oracle_summary = " ".join([sentences[index] for index in sorted(list(oracle_summary_sentences))])
    return oracle_summary, oracle_summary_sentences

def calc_single_score(pred_summary, gold_summary, rouge):
    return rouge.get_scores([pred_summary], [gold_summary], avg=True)['rouge-2']['f']
```

In [ ]:
```python
from tqdm import tqdm_notebook as tqdm

def calc_oracle_score(records, nrows=None, lower=True):
    references = []
    predictions = []
    rouge = Rouge()

    for text, summary in tqdm(records[['text', 'summary']].values[:nrows]):
        summary = summary if not lower else summary.lower()
        references.append(summary)
        predicted_summary, _ = build_oracle_summary_greedy(text, summary, calc_score=lambda x, y: calc_single_score(x, y, rouge))
        predictions.append(predicted_summary)

    calc_scores(references, predictions)

calc_oracle_score(test_records)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:8: TqdmDeprecationWarning: This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
```

```
Count: 5770
Ref: центробанк россии одобрил проект «норильского никеля» по созданию в россии уникальной экосистемы, которая позволит «оцифровать» (токенизировать) разного рода товары и услуги, в том числе металлы. электронная площадка будет выступать в качестве оператора, обеспечивающего процесс перевода прав на владение активом в цифровой вид (токен) при помощи блокчейна. эксперты называют этот проект выдающимся достижением для рынка.
Hyp: эта система имеет массу вариантов применения, как в интересах бизнеса, так и на государственном уровне – согласно «дорожной карте», подготовленной «ростехом», до 2024 года на ее развитие будет потрачено 28,4 млрд рублей, в том числе 9,5 млрд бюджетных средств. уникальная экосистема, которую создаст блокчейн-платформа «норникеля», позволит «оцифровать» (токенизировать) любые активы и обеспечит последующее обращение размещенных токенов asset-backed coin ( abc -токенов).
BLEU:  0.4330710925589385
ROUGE:  {'rouge-1': {'f': 0.3544658747486028, 'p': 0.4316554001222412, 'r': 0.3186857136306109}, 'rouge-2': {'f': 0.20304390374943623, 'p': 0.2534298580037621, 'r': 0.18137320016490807}, 'rouge-l': {'f': 0.3037603325246813, 'p': 0.40026927941223245, 'r': 0.29455278805443086}}
```

In [ ]:
```python
from rouge import Rouge
import razdel

def add_oracle_summary_to_records(records, max_sentences=30, lower=True, nrows=1000):
    rouge = Rouge()
    sentences_ = []
    oracle_sentences_ = []
    oracle_summary_ = []
    if nrows is not None:
        records = records.iloc[:nrows].copy()
    else:
        records = records.copy()

    for text, summary in tqdm(records[['text', 'summary']].values):
        summary = summary.lower() if lower else summary
        sentences = [sentence.text.lower() if lower else sentence.text for sentence in razdel.sentenize(text)][:max_sentences]
        oracle_summary, sentences_indicies = build_oracle_summary_greedy(text, summary, calc_score=lambda x, y: calc_single_score(x, y, rouge),
                                                                         lower=lower, max_sentences=max_sentences)
        sentences_ += [sentences]
        oracle_sentences_ += [list(sentences_indicies)]
        oracle_summary_ += [oracle_summary]
    records['sentences'] = sentences_
    records['oracle_sentences'] = oracle_sentences_
    records['oracle_summary'] = oracle_summary_
    return records

ext_train_records = add_oracle_summary_to_records(train_records, nrows=30000)
ext_val_records = add_oracle_summary_to_records(val_records, nrows=None)
ext_test_records = add_oracle_summary_to_records(test_records, nrows=None)
```

Используй `pickle` для сохранения записей, чтобы потом не пересоздавать их потом. Если решаешь задание в колабе, можешь подключить свой гугл диск и сохранить данные в нём.

In [ ]:
```python
#SAVE
import pickle
from google.colab import drive
drive.mount('/content/drive')

with open("/content/drive/MyDrive/test_records.bin", 'wb') as file:
    pickle.dump(ext_test_records, file)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

In [ ]:
```python
#LOAD
import pickle
from google.colab import drive
drive.mount('/content/drive')

with open('/content/drive/MyDrive/train_records.bin', 'rb') as f:
    ext_train_records = pickle.load(f)

with open('/content/drive/MyDrive/test_records.bin', 'rb') as f:
    ext_test_records = pickle.load(f)

with open('/content/drive/MyDrive/val_records.bin', 'rb') as f:
    ext_val_records = pickle.load(f)
```

```
Mounted at /content/drive
```

(!)

Если надо, поменяйте код генератора датасета и батчевалки

In [ ]:
```python
import random
import math
import razdel
import torch
import numpy as np
from rouge import Rouge


from torch.utils import data


class ExtDataset(data.Dataset):
    def __init__(self, records, tokenizer, lower=True, max_sentence_length=50, device=torch.device('cpu')):
        self.records = records
        self.num_samples = records.shape[0]
        self.tokenizer = tokenizer
        self.lower = lower
        self.rouge = Rouge()
        self.max_sentence_length = max_sentence_length
        self.device = device

    def __len__(self):
        return self.records.shape[0]

    def __getitem__(self, idx):
        cur_record = self.records.iloc[idx]

        input_ids, attention_mask = self.tokenizer(cur_record['sentences'], padding=True, truncation=True, max_length=self.max_sentence_length, return_tensors='pt').values()
        outputs = [int(i in cur_record['oracle_sentences']) for i in range(len(cur_record['sentences']))]
        return {'input_ids': input_ids, 'attention_mask': attention_mask, 'outputs': outputs}
```

In [ ]:
```python
# Это батчевалка
def collate_fn(records):
    max_length = max(len(sentence) for record in records for sentence in record['input_ids'])
    max_sentences = max(len(record['outputs']) for record in records)
    new_inputs = torch.zeros((len(records), max_sentences, max_length))
    new_attention_mask = torch.zeros((len(records), max_sentences, max_length))
    new_outputs = torch.zeros((len(records), max_sentences))

    for i, record in enumerate(records):
        for j, (sentence, mask) in enumerate(zip(record['input_ids'], record['attention_mask'])):
            new_inputs[i, j, :len(sentence)] += np.array(sentence)
            new_attention_mask[i, j, :len(sentence)] += np.array(mask)
        if j < max_sentences:
            for k in range(j+1, max_sentences):
                new_inputs[i, k, 0], new_inputs[i, k, 1]  = 101, 102
                new_attention_mask[i, k, 0:2] = 1
        new_outputs[i, :len(record['outputs'])] += np.array(record['outputs'])
    return {'features': new_inputs.type(torch.LongTensor), 'attention_mask': new_attention_mask.type(torch.LongTensor), 'targets': new_outputs}
```

In [ ]:
```python
import numpy as np

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

from torch.nn.utils.rnn import pack_padded_sequence as pack
from torch.nn.utils.rnn import pad_packed_sequence as unpack


class SentenceTaggerRNN(nn.Module):
    def __init__(self,
                 vocabulary_size,
                 token_embedding_dim=256,
                 sentence_encoder_hidden_size=768,
                 hidden_size=256,
                 bidirectional=True,
                 sentence_encoder_n_layers=2,
                 sentence_encoder_dropout=0.3,
                 sentence_encoder_bidirectional=True,
                 n_layers=1,
                 dropout=0.3):
        super(SentenceTaggerRNN, self).__init__()

        num_directions = 2 if bidirectional else 1
        assert hidden_size % num_directions == 0
        hidden_size = hidden_size // num_directions

        self.hidden_size = hidden_size
        self.n_layers = n_layers
        self.dropout = dropout
        self.bidirectional = bidirectional

        # Your sentence encoder model
        self.sentence_encoder = encoder # pretrained distilbert-multilingual-nli-stsb-quora-ranking

        self.rnn_layer = nn.LSTM(
            sentence_encoder_hidden_size,
            hidden_size,
            n_layers,
            dropout=dropout,
            bidirectional=bidirectional,
            batch_first=True)
        self.dropout_layer = nn.Dropout(dropout)
        self.content_linear_layer = nn.Linear(hidden_size * 2, 1)
        self.document_linear_layer = nn.Linear(hidden_size * 2, hidden_size * 2)
        self.salience_linear_layer = nn.Linear(hidden_size * 2, hidden_size * 2)
        self.tanh_layer = nn.Tanh()


    def forward(self, inputs, attention_mask, hidden=None):
        batch_size = inputs.size(0)
        sentences_count = inputs.size(1)
        tokens_count = inputs.size(2)

        inputs = inputs.reshape(-1, tokens_count)
        attention_mask = attention_mask.reshape(-1, tokens_count)

        with torch.no_grad():
            embedded_sentences = self.sentence_encoder(inputs, attention_mask=attention_mask)[0]

        embedded_sentences = torch.mean(embedded_sentences, 1)

        embedded_sentences = embedded_sentences.reshape(batch_size, sentences_count, -1)
        outputs, _ = self.rnn_layer(embedded_sentences, hidden)
        outputs = self.dropout_layer(outputs)
        document_embedding = self.tanh_layer(self.document_linear_layer(torch.mean(outputs, 1)))
        content = self.content_linear_layer(outputs).squeeze(2)
        salience = torch.bmm(outputs, self.salience_linear_layer(document_embedding).unsqueeze(2)).squeeze(2)
        return content + salience

model = SentenceTaggerRNN(tokenizer.vocab_size)
```

```
/usr/local/lib/python3.6/dist-packages/torch/nn/modules/rnn.py:61: UserWarning: dropout option adds dropout after all but last recurrent layer, so non-zero dropout expects num_layers greater than 1, but got dropout=0.3 and num_layers=1
  "num_layers={}".format(dropout, num_layers))
```

Обучение

In [ ]:
```python
device = torch.device('cuda')

loaders = {
    'train': data.DataLoader(
        ExtDataset(
            ext_train_records,
            tokenizer=tokenizer
        ),
        batch_size=32,
        collate_fn=collate_fn
    ),
    'valid': data.DataLoader(
        ExtDataset(
            ext_val_records,
            tokenizer=tokenizer
        ),
        batch_size=32,
        collate_fn=collate_fn
    ),
    'test': data.DataLoader(
        ExtDataset(
            ext_test_records,
            tokenizer=tokenizer
        ),
        batch_size=1,
        collate_fn=collate_fn
    ),
}

lr = 1e-4
num_epochs = 1

optimizer  = torch.optim.Adam(model.parameters(), lr=lr)
criterion = nn.BCEWithLogitsLoss()
```

In [ ]:
```python
from tqdm.notebook import trange, tqdm


def train():
    model.to(device)
    pbar_loader = trange(len(loaders["train"]) + len(loaders["valid"]), desc=f"Train Loss: {0}, Valid Loss: {0}")
    for e in trange(num_epochs, desc="Epoch"):
        train_loss = 0
        valid_loss = 0
        train_it = 0
        valid_it = 0

        model.train()
        for batch in loaders["train"]:
            features = batch["features"].to(device)
            mask = batch['attention_mask'].to(device)
            targets = batch["targets"].to(device)

            logits = model(features, attention_mask=mask)


            loss = criterion(logits, targets)
            train_loss += loss.item()
            train_it += 1

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            pbar_loader.update()
            pbar_loader.set_description(
                f"Train Loss: {train_loss / train_it:.3}, Valid Loss: {0}"
            )

        model.eval()
        with torch.no_grad():
            for batch in loaders["valid"]:
                features = batch["features"].to(device)
                mask = batch['attention_mask'].to(device)
                targets = batch["targets"].to(device)

                logits = model(features, attention_mask=mask)

                loss = criterion(logits, targets)
                valid_loss += loss.item()
                valid_it += 1

                pbar_loader.update()
                pbar_loader.set_description(
                    f"Train Loss: {train_loss / train_it:.3},"
                    f" Valid Loss: {valid_loss / valid_it:.3}"
                )
        print(
            f"Epoch {e}; Train Loss: {train_loss / train_it:.3},"
            f" Valid Loss: {valid_loss / valid_it:.3}"
        )
        pbar_loader.reset()
```

In [ ]:
```python
train()
```

```
Epoch 0; Train Loss: 0.187, Valid Loss: 0.176
```

In [ ]:
```python
device = torch.device("cuda")

references = []
predictions = []
model.eval()
for num, item in tqdm(enumerate(loaders["test"]), total=len(loaders["test"])):

    with torch.no_grad():

        logits = model(item["features"].to(device), item['attention_mask'].to(device))[0]

    record = ext_test_records.iloc[num]
    predicted_summary = []

    idxs = torch.argsort(logits, dim=0, descending=True)[0:3]
    for idx in idxs:
        predicted_summary.append(record['sentences'][idx])

    predicted_summary = " ".join(predicted_summary)
    references.append(record['summary'].lower())
    predictions.append(predicted_summary)

calc_scores(references, predictions)
```

```
Count: 5770
Ref: несмотря на пандемию covid-19, которая парализовала всю мировую спортивную жизнь, в английской премьер-лиге назвали лучшего футболиста по итогам февраля. им стал новичок «манчестер юнайтед» бруну фернандеш, который переехал на «олд траффорд» в январе.
Hyp: 16 марта хавбек «манчестер юнайтед» бруну фернандеш был признан лучшим игроком апл в феврале. португалец играет на туманном альбионе всего два месяца, но уже стал открытием и одним из ключевых игроков «красных дьяволов». всего за «манчестер юнайтед» 25-летний футболист провел девять матчей, забив три мяча и отдав четыре результативных передачи.
BLEU:  0.4318954336585758
ROUGE:  {'rouge-1': {'f': 0.26434887443125404, 'p': 0.2503187859166187, 'r': 0.2975100765247292}, 'rouge-2': {'f': 0.11285862072924831, 'p': 0.10567725846968763, 'r': 0.13006355860634336}, 'rouge-l': {'f': 0.22780874788640282, 'p': 0.22793865178773226, 'r': 0.27052602791414326}}
```

Вообще получилось так, что при обучении качество BLEU только уменьшалось. Даже при использовании взвешенного лосса, лучший скор который получился - выше