Implementation of a framework to fine-tune GPT/GPT2 based models on abstractive summarization.

David Lorenzo Alfaro

Jul 05, 2022

CONTENTS:

Abstractive summarization has experienced a surge of interest thanks to recent advancements on Transformer-based encoder-decoder models, with standout proposals like PEGASUS, that incorporates explicit pre-training objectives tailored for such a task, exhibiting unprecedented level of performance on natural language generation tasks. However, the humongous amount of data required and the massive computational cost attached to the pre-training of these architectures imposes a substantial burden on their availability in languages other than English, with the very exception of their multi-lingual homologous.

The recent large Spanish language models from the MarIA project, based on the RoBERTa and GPT-2 architectures, have shown promising results, pushing the state-of-the-art on multiple natural language understanding tasks. However, encoder- and decoder-only systems pose as an architecturally suboptimal approach to resolve sequence-to-sequence tasks. In this work, we explore the applicability of these language models for abstractive summarization.

In this page, we present the documentation of a fully-documented API-like tool to fine-tune GPT-like architectures on abstractive summarization.

CHAPTER

ONE

EXPLORATORY DATA ANALYSIS OF THE *XL-SUM* DATASET.

As part of the MSc. dissertation: Applying large Spanish language models to Natural Language Processing tasks.

David Lorenzo Alfaro

July, 2022

[28]:
```
import pandas as pd
import matplotlib.pyplot as plt
from transformers import AutoTokenizer, GPT2LMHeadModel, RobertaForMaskedLM
from utils import get_tokenized_text
from typing import Iterable, Union
from pathlib import Path
```

1.1 Preliminary data analysis

Let us first load the data from disk. It is a prerrequisite to either have it downloaded locally and located in a subdirectory called `summaries/`, or make the appropriate modifications to the code to load it (e.g., via the HuggingFace's datasets library, `wget` request/s). In particular, we gathered the data from the *XL-Sum* official GitHub repository

[10]:
```
DATA_DIR = Path('summaries/all')
TRAIN_PATH = Path(DATA_DIR, 'train.jsonl')
VAL_PATH = Path(DATA_DIR, 'val.jsonl')
TEST_PATH = Path(DATA_DIR, 'test.jsonl')
```

The necessary columns in the dataset are as listed: * `text`: content of an article. * `summary`: summary of an article. * `id`: unique identifier of the article.

Indeed, other than article-summary pairs, all remaining information may be disregarded for the task that we are aimed at conducting.

Let us first load the training dataset to inspect some of its properties

[11]:
```
df_train = pd.read_json(TRAIN_PATH, lines=True)[["id", "summary", "text"]]
```

Now, let us print the first k elements of the training dataset

[12]:
```
top_k=3
for i, (idx, s,t) in enumerate(zip(df_train.id, df_train.summary, df_train.text)):
    if i >= top_k:
```

(continues on next page)

(continued from previous page)

```
        break
    print(f'id:{idx}\nText: {t} \nSummary: {s}', end="\n\n")
```

```
id:140930_ultnot_siria_onu_comida_ch
Text: De no recibir más dinero, las raciones de la ONU en Siria se terminarán en dos␣
↪meses. En un informe al Consejo de Seguridad de la ONU, Amos dijo que las raciones del␣
↪PMA destinadas a los 4.000.000 de sirios ya han sido recortadas para poder llegar a la␣
↪mayor cantidad de personas posible. Amos también hizo un llamado a juntar suministros␣
↪para proteger a los sirios del frío, en miras al próximo invierno.
Summary: La directora de ayuda humanitaria de Naciones Unidas, Valerie Amos, advirtió␣
↪que de no invertir más dinero, el Programa Mundial de Alimentos tendrá que detener sus␣
↪operaciones en Siria en dos meses.

id:130809_ultnot_protesta_cachemira_protesa_aa
Text: Manifestantes hindúes gritan consignas contra el gobierno en la región de␣
↪Cachemira administrada por India. Las manifestaciones tuvieron lugar después de los␣
↪rezos especiales Eid, en la ciudad de Srinagar y en diversas ciudades de la región. La␣
↪policía dice que la respuesta se produjo cuando la muchedumbre se volvió violenta y␣
↪empezó a lanzarles palos y piedras. Varios policías y manifestantes resultaron heridos.
↪ El jueves a la noche diversos líderes separatistas fueron arrestados para evitar que␣
↪lideraran las protestas.
Summary: La policía en la región de Cachemira administrada por India, lanzó gas␣
↪lacrimógeno y disparó balas de goma para dispersar una protesa contra supuestas␣
↪violaciones de los derechos humanos llevadas a cabo por las fuerzas del gobierno.

id:media-37220890
Text: Así lo vemos en esta animación que muestra cómo el río busca nuevos caminos. Si␣
↪bien el mundo ha perdido millones de litros de agua, también se han ganado 115.000␣
↪kilómetros cuadrados. La zona donde ha habido un mayor aumento de agua en el mundo es␣
↪en la meseta tibetana (donde el derretimiento de glaciares está creando lagos). Pero␣
↪es en el Amazonas donde esta lucha es más evidente.
Summary: La naturaleza es un hueso duro de roer.
```

Alternatively, one can resort to the built-in function head available in `pd.DataFrame` objects.

[13]:
```
df_train.head(top_k)
```

[13]:
```
                                            id  \
0             140930_ultnot_siria_onu_comida_ch
1  130809_ultnot_protesta_cachemira_protesa_aa
2                                media-37220890

                                            summary  \
0  La directora de ayuda humanitaria de Naciones ...
1  La policía en la región de Cachemira administr...
2             La naturaleza es un hueso duro de roer.

                                               text
0  De no recibir más dinero, las raciones de la O...
1  Manifestantes hindúes gritan consignas contra ...
2  Así lo vemos en esta animación que muestra cóm...
```

1.1.1 Impact of the limited dimensionality of the Language Models for Abstractive Summarization

Both GPT-2 and RoBERTa work on the subword level. All spanish language models from the MarIA project have a limited dimensionality of 512 tokens (or subwords). Furthermore, in order to generate summaries on a decoder-only system, both the text and the summary must fit into the model, which imposes further restrictions on the amount of instances in the dataset that can be processed in as as-is fashion (i.e., without needing to do any further modification in the text/summary of those instances before feeding them into the model). In summarization tasks, where the prominent practice is to strive for models with larger dimensionalities, this shortcoming is of vital relevance because it necessarily constraints the capabilities of the models to work with large texts.

1.1.2 Training set

Let us inspect the number of *valid* instances (those that fit into the model) in the training set of data, using the GPT-2 model, and a RoBERTa2RoBERTa (a.k.a. RoBERTaShared) model. To that end, we will compute the number of article and summary tokens using the GPT-2 tokenizer for the spanish LM, and the number of article tokens using the RoBERTa tokenizer. Note that there is no need to compute the per-summary no. of tokens because this model will be subsequently used as the *warmed-up* models of an encoder-decoder system. However, we will also yield this statistic in order to further study some properties about the length of the summaries, which can be of vital relevance in the generation of the summaries.

Beforehand, let us define some useful functions that will enable us to comptue the number of tokens of a collection of tokens using a tokenizer (`calc_document_tokens`), and to plot histograms for article and summary length (also in terms of subwords) and a scatterplot relating these two random variables.

[57]:
```
def calc_document_tokens(documents:Iterable, tokenizer:Union[str, Path]):
    """ Compute the number of tokens of a collection of documents using a pretrained␣
↪tokenizer

    Parameters
    ----------
    documents : Iterable
        Collection of documents

    tokenizer : Union[str, Path]
        Model identifier of a predefined tokenizer hosted inside a model repo
         on huggingface.co
    """
    # Load tokenizer
    tokenizer = AutoTokenizer.from_pretrained(tokenizer)

    return [len(get_tokenized_text(doc, tokenizer)) for doc in documents]

def plot_stats(article_tokens:Iterable, summary_tokens:Iterable, hist_bins=20, scatter_
↪marker_size=3):
    """ Plot some statistics about the article and summary tokens.

    Parameters
    ----------
    article_tokens : Iterable
        Collection of per-sample article tokens

    summary_tokens : Iterable
```

(continues on next page)

(continued from previous page)

```
        Collection of per-sample summary tokens

    hist_bins : int
        Number of bins used to discretize data for histograms, defaults to 20

    scatter_marker_size : int
        Marker size of the scatter-plot, defaults to 3
    """
    fig, (ax1, ax2, ax3) = plt.subplots(ncols=3)
    ax1.hist(article_tokens, bins=hist_bins)
    ax2.hist(summary_tokens, bins=hist_bins)
    ax3.scatter(article_tokens, summary_tokens, s=scatter_marker_size)
    fig.set_size_inches(12,4)
    fig.tight_layout()
```

Now, let us compute the of tokens for the articles and summaries using the tokenizers of the GPT-2 and RoBERTa checkpoints.

[17]:
```
checkpoint_gpt2 = "PlanTL-GOB-ES/gpt2-base-bne"
gpt2_article_tokens_train = calc_document_tokens(df_train.text, checkpoint_gpt2)
gpt2_summary_tokens_train = calc_document_tokens(df_train.summary, checkpoint_gpt2)
```

```
Special tokens have been added in the vocabulary, make sure the associated word␣
↪embeddings are fine-tuned or trained.
Special tokens have been added in the vocabulary, make sure the associated word␣
↪embeddings are fine-tuned or trained.
```

[54]:
```
checkpoint_roberta = "PlanTL-GOB-ES/roberta-base-bne"
roberta_article_tokens_train = calc_document_tokens(df_train.text, checkpoint_roberta)
roberta_summary_tokens_train = calc_document_tokens(df_train.summary, checkpoint_roberta)
```

```
Token indices sequence length is longer than the specified maximum sequence length for␣
↪this model (2183 > 512). Running this sequence through the model will result in␣
↪indexing errors
```

To perform fine-tuning on a GPT-2 based model, at least 1 token should be reserved for a special token, which will serve as a separator between the article and the summary of each input sample. As you may see from the training sources, our particular take is to use the `<|sep|>` gram as the separator token, albeit a different sequence may be used, so long it does not conflict with an existing entry in the vocabulary of the tokenizer.

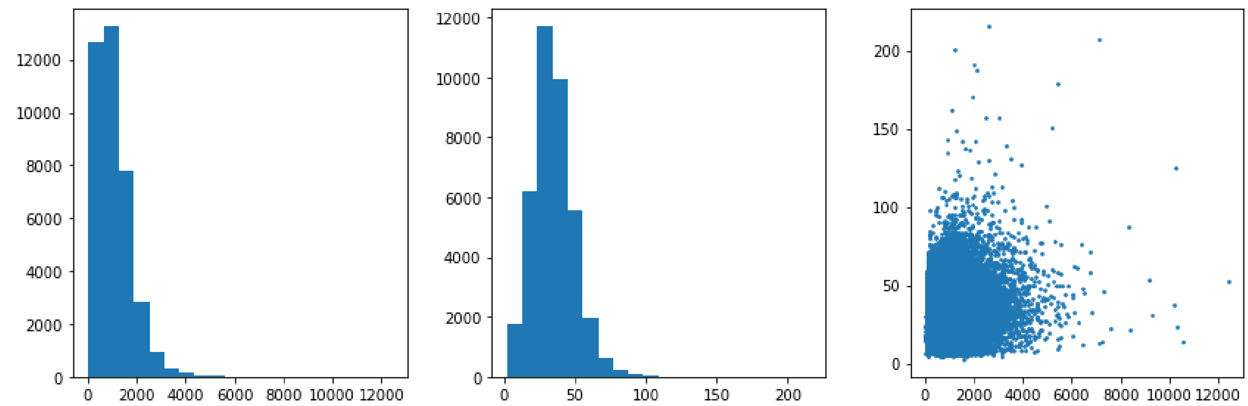
On the other hand, although we that the input maximum length is constraint to 512 tokens, a programmatic way to check such limitation is by accessing to the `config.n_positions` property of a GPT-2 HuggingFace checkpoint.

[40]:
```
gpt2_max_n_tokens = GPT2LMHeadModel.from_pretrained(checkpoint_gpt2).config.n_positions
gpt2_valid_train = len(list(filter(lambda x: sum(x)<=gpt2_max_n_tokens-1, zip(gpt2_
↪article_tokens_train,gpt2_summary_tokens_train))))
gpt2_invalid_train = len(gpt2_article_tokens_train) - gpt2_valid_train
# out of the box... how many articles can be processed by the gpt-2 model?
print(f'Using the "{checkpoint_gpt2}" model, {gpt2_valid_train} out of {len(gpt2_article_
↪tokens_train)} training samples have <= {gpt2_max_n_tokens-1} tokens and thus are␣
↪subject to application out of the box, whereas {gpt2_invalid_train} will cause an␣
↪error unless further measures are taken (e.g., via truncation of info. in a meaningful␣
↪fashion).')
```

```
Using the "PlanTL-GOB-ES/gpt2-base-bne" model, 9692 out of 38110 training samples have
↪<= 511 tokens and thus are subject to application out of the box, whereas 28418 will␣
↪cause an error unless further measures are taken (e.g., via truncation of info. in a␣
↪meaningful fashion).
```

Let us further inspect some properties about the samples in the training dataset: * How are the length of the articles and summaries distributed? * Are the length of the article and its summary correlated?

[58]:
```
plot_stats(gpt2_article_tokens_train, gpt2_summary_tokens_train)
```



From the graphs we can tell: * vast majority of the articles are around 500 to 1500 tokens long, whereas regular summary length revolves around 20 to 50 tokens. * summary length seems to be independent from the length of the article.

We can also calculate some descriptive statistics summarizing the central tendency, dispersion and shape of the distribution of the article and summary tokens.

[68]:
```
pd.DataFrame(list(zip(gpt2_article_tokens_train,gpt2_summary_tokens_train)), columns=[
↪"gpt2_article_tokens_train", "gpt2_summary_tokens_train"]).describe()
```

[68]:

	gpt2_article_tokens_train	gpt2_summary_tokens_train
count	38110.000000	38110.000000
mean	1034.331068	34.986119
std	769.745530	14.608020
min	31.000000	2.000000
25%	466.000000	25.000000
50%	956.000000	34.000000
75%	1415.000000	43.000000
max	12421.000000	216.000000

Conclusions: * Mean number of article tokens in the training set: 1034 ± 770. 770 is a large standard deviation, hence the mean is not a very informative statistic (e.g., because of the influence of extreme values). The median may be more useful: 956 tokens.

- Mean number of summary tokens in the training set: 35 ± 15. The value of the median is very similar: 34.

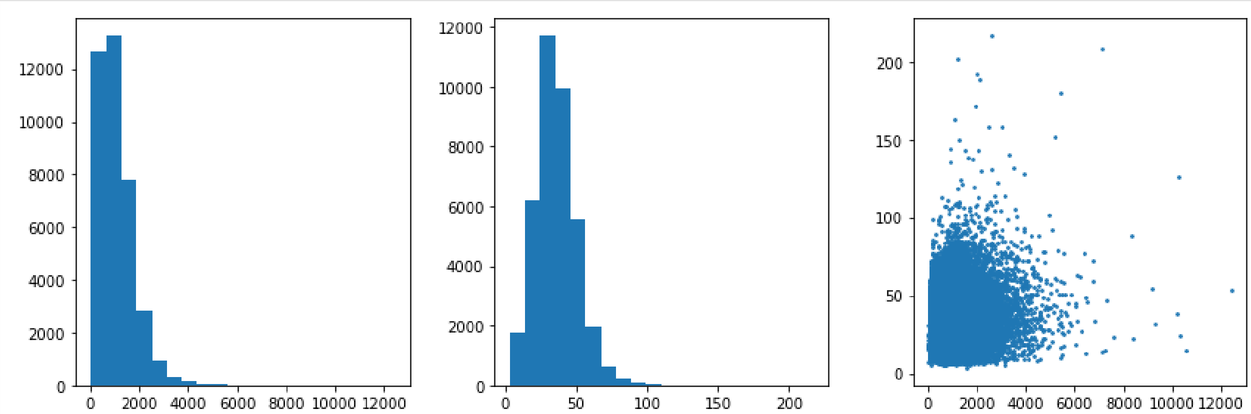
Let us do the same for the RoBERTa LM, considering that the maximum number of tokens that the model can handle is two less than that reported of the model, being as we need to reserve two extra positions (or segments) for two special tokens. To double-check the maximum input of the length we can access to the `config.max_position_embeddings` property of the RoBERTa HuggingFace checkpoint.

[30]:
```
roberta = RobertaForMaskedLM.from_pretrained(checkpoint_roberta)
```

[39]:
```
roberta_max_n_tokens = RobertaForMaskedLM.from_pretrained(checkpoint_roberta).config.max_position_embeddings - 2 # reserve tokens for [CLS] and [EOS]
roberta_valid_train = len(list(filter(lambda x: x<=roberta_max_n_tokens, roberta_article_tokens_train)))
roberta_invalid_train = len(roberta_article_tokens_train) - roberta_valid_train
# out of the box... how many articles can be processed by the roberta model?
print(f'Using the "{checkpoint_roberta}" model, {roberta_valid_train} out of {len(roberta_article_tokens_train)} training samples have <= {roberta_max_n_tokens} tokens and thus are subject to application out of the box, whereas {roberta_invalid_train} will cause an error unless further measures are taken (e.g., via truncation of info. in a meaningful fashion).')
```

```
Using the "PlanTL-GOB-ES/roberta-base-bne" model, 10235 out of 38110 training samples have <= 512 tokens and thus are subject to application out of the box, whereas 27875 will cause an error unless further measures are taken (e.g., via truncation of info. in a meaningful fashion).
```

[61]:
```
plot_stats(roberta_article_tokens_train, roberta_summary_tokens_train)
```



[69]:
```
pd.DataFrame(list(zip(roberta_article_tokens_train,roberta_summary_tokens_train)), columns=["roberta_article_tokens_train", "roberta_summary_tokens_train"]).describe()
```

[69]:

	roberta_article_tokens_train	roberta_summary_tokens_train
count	38110.000000	38110.000000
mean	1035.331068	35.986119
std	769.745530	14.608020
min	32.000000	3.000000
25%	467.000000	26.000000
50%	957.000000	35.000000
75%	1416.000000	44.000000
max	12422.000000	217.000000

Analogous conclusions can be inferred when using the RoBERTa tokenizer, with nearly negligible differences.

1.1.3 Validation set

Now, let us conduct this experiment on the validation and test datasets:

[42]:
```
df_val = pd.read_json(VAL_PATH, lines=True)[["id", "summary", "text"]]
```

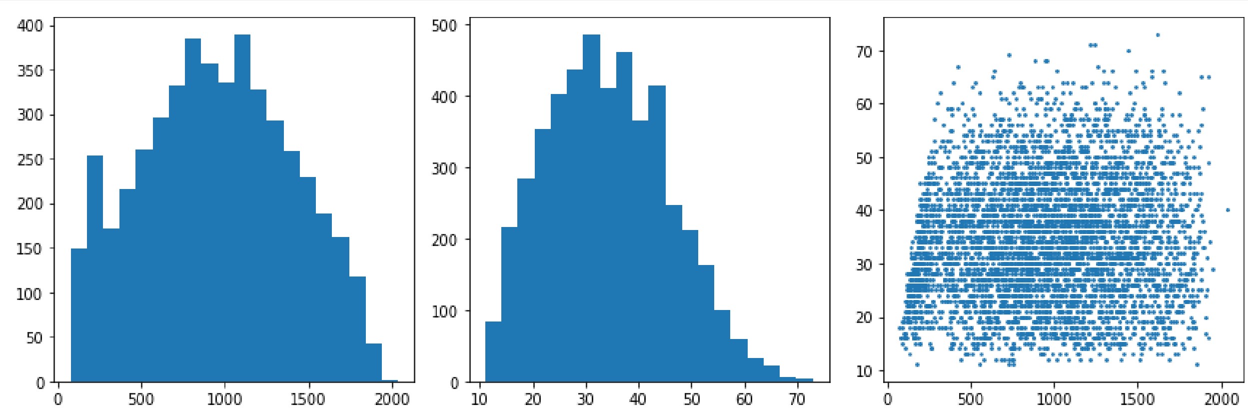
[41]:
```
gpt2_article_tokens_val = calc_document_tokens(df_val.text, checkpoint_gpt2)
gpt2_summary_tokens_val = calc_document_tokens(df_val.summary, checkpoint_gpt2)
```

```
Special tokens have been added in the vocabulary, make sure the associated word␣
↪embeddings are fine-tuned or trained.
Special tokens have been added in the vocabulary, make sure the associated word␣
↪embeddings are fine-tuned or trained.
```

[44]:
```
gpt2_valid_val = len(list(filter(lambda x: sum(x)<=gpt2_max_n_tokens-1, zip(gpt2_article_
↪tokens_val,gpt2_summary_tokens_val))))
gpt2_invalid_val = len(gpt2_article_tokens_val) - gpt2_valid_val
print(f'Using the "{checkpoint_gpt2}" model, {gpt2_valid_val} out of {len(gpt2_article_
↪tokens_val)} validation samples have <= {gpt2_max_n_tokens-1} tokens and thus are␣
↪subject to application out of the box, whereas {gpt2_invalid_val} will cause an error␣
↪unless further measures are taken.')
```

```
Using the "PlanTL-GOB-ES/gpt2-base-bne" model, 814 out of 4763 validation samples have
↪<= 511 tokens and thus are subject to application out of the box, whereas 3949 will␣
↪cause an error unless further measures are taken.
```

[70]:
```
plot_stats(gpt2_article_tokens_val, gpt2_summary_tokens_val)
```



[71]:
```
pd.DataFrame(list(zip(gpt2_article_tokens_val,gpt2_summary_tokens_val)), columns=["gpt2_
↪article_tokens_val", "gpt2_summary_tokens_val"]).describe()
```

[71]:

	gpt2_article_tokens_val	gpt2_summary_tokens_val
count	4763.000000	4763.000000
mean	950.719295	34.226748
std	446.983980	11.507983
min	75.000000	11.000000
25%	615.000000	25.000000
50%	951.000000	33.000000
75%	1286.000000	42.000000
max	2038.000000	73.000000

We can draw similar conclusions to those obtained for the training set: * Vast majority of the articles are around 600 to 1200 tokens long, whereas regular summary length revolves around 20 to 45 tokens. * Summary length seems to be independent from the length of the article. * Mean number of article tokens in the validation set: 950 ± 447. Fairly close to the median: 951 tokens. * Mean number of summary tokens in the validation set: 34 ± 12. The value of the median is very similar: 33.

In order to avoid verbosity, we eschew reproducing these experiments using the RoBERTa tokenizer, since very similar results are to expect.

[55]:
```
roberta_article_tokens_val = calc_document_tokens(df_val.text, checkpoint_roberta)
roberta_summary_tokens_val = calc_document_tokens(df_val.summary, checkpoint_roberta)
```

```
Token indices sequence length is longer than the specified maximum sequence length for
↪this model (780 > 512). Running this sequence through the model will result in
↪indexing errors
```

[46]:
```
roberta_valid_val = len(list(filter(lambda x: x<=roberta_max_n_tokens, roberta_article_
↪tokens_val)))
roberta_invalid_val = len(roberta_article_tokens_val) - roberta_valid_val
print(f'Using the "{checkpoint_roberta}" model, {roberta_valid_val} out of {len(roberta_
↪article_tokens_val)} training samples have <= {roberta_max_n_tokens} tokens and thus
↪are subject to application out of the box, whereas {roberta_invalid_val} will cause an
↪error unless further measures are taken.')
```

```
Using the "PlanTL-GOB-ES/roberta-base-bne" model, 903 out of 4763 training samples have
↪<= 512 tokens and thus are subject to application out of the box, whereas 3860 will
↪cause an error unless further measures are taken.
```

1.1.4 Test set

[48]:
```
df_test = pd.read_json(TEST_PATH, lines=True)[["id", "summary", "text"]]
```

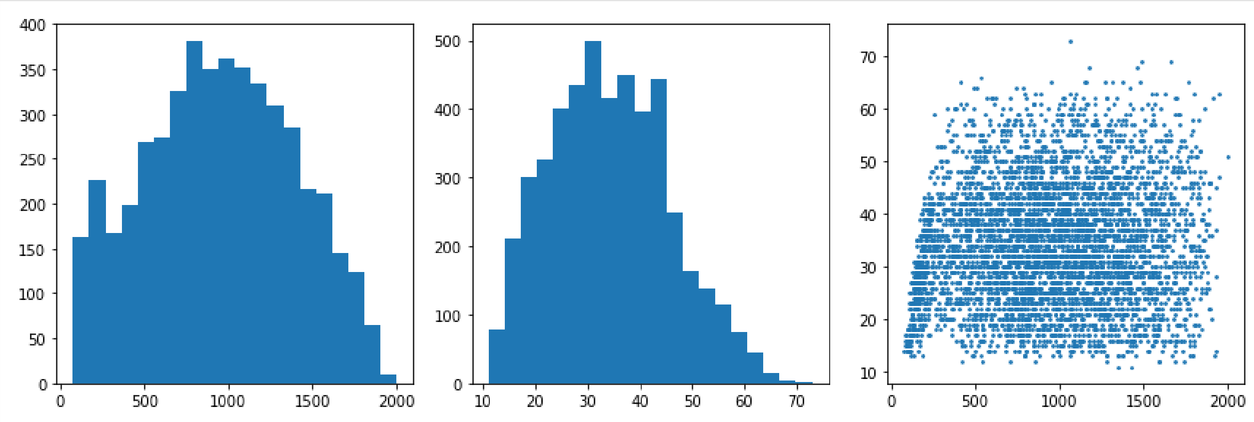
[49]:
```
gpt2_article_tokens_test = calc_document_tokens(df_test.text, checkpoint_gpt2)
gpt2_summary_tokens_test = calc_document_tokens(df_test.summary, checkpoint_gpt2)
```

```
Special tokens have been added in the vocabulary, make sure the associated word
↪embeddings are fine-tuned or trained.
Special tokens have been added in the vocabulary, make sure the associated word
↪embeddings are fine-tuned or trained.
```

[50]:
```
gpt2_valid_test = len(list(filter(lambda x: sum(x)<=gpt2_max_n_tokens-1, zip(gpt2_
↪article_tokens_test,gpt2_summary_tokens_test))))
gpt2_invalid_test = len(gpt2_article_tokens_test) - gpt2_valid_val
print(f'Using the "{checkpoint_gpt2}" model, {gpt2_valid_test} out of {len(gpt2_article_
↪tokens_test)} test samples have <= {gpt2_max_n_tokens-1} tokens and thus are subject
↪to application out of the box, whereas {gpt2_invalid_test} will cause an error unless
↪further measures are taken.')
```

```
Using the "PlanTL-GOB-ES/gpt2-base-bne" model, 804 out of 4763 test samples have <= 511
↪tokens and thus are subject to application out of the box, whereas 3949 will cause an
↪error unless further measures are taken.
```

[72]:
```
plot_stats(gpt2_article_tokens_test, gpt2_summary_tokens_test)
```



[73]:
```
pd.DataFrame(list(zip(gpt2_article_tokens_test,gpt2_summary_tokens_test)), columns=[
↪"gpt2_article_tokens_test", "gpt2_summary_tokens_test"]).describe()
```

[73]:

	gpt2_article_tokens_test	gpt2_summary_tokens_test
count	4763.000000	4763.000000
mean	948.605921	34.231787
std	444.319426	11.396173
min	75.000000	11.000000
25%	616.000000	26.000000
50%	951.000000	33.000000
75%	1283.000000	42.000000
max	2002.000000	73.000000

We can draw similar conclusions to those obtained for the training and validation set, which enable us to assert that, overall, **the number of article and summary tokens across the different datasets are independent and identically distributed**: * Vast majority of the articles are around 600 to 1200 tokens long, whereas regular summary length revolves around 20 to 45 tokens. * Summary length seems to be independent from the length of the article. * Mean number of article tokens in the test set: 959 ± 444. Fairly close to the median: 951 tokens. * Mean number of summary tokens in the test set: 34 ± 11. The value of the median is very similar: 33.

In order to avoid verbosity, we eschew reproducing these experiments using the RoBERTa tokenizer, since very similar results are to expect.

[56]:
```
roberta_article_tokens_test = calc_document_tokens(df_test.text, checkpoint_roberta)
roberta_summary_tokens_test = calc_document_tokens(df_test.summary, checkpoint_roberta)
```

```
Token indices sequence length is longer than the specified maximum sequence length for␣
↪this model (1248 > 512). Running this sequence through the model will result in␣
↪indexing errors
```

[52]:
```
roberta_valid_test = len(list(filter(lambda x: x<=roberta_max_n_tokens, roberta_article_
↪tokens_test)))
roberta_invalid_test = len(roberta_article_tokens_test) - roberta_valid_test
print(f'Using the "{checkpoint_roberta}" model, {roberta_valid_test} out of {len(roberta_
↪article_tokens_test)} training samples have <= {roberta_max_n_tokens} tokens and thus␣
↪are subject to application out of the box, whereas {roberta_invalid_test} will cause␣
↪an error unless further measures are taken.')
```

```
Using the "PlanTL-GOB-ES/roberta-base-bne" model, 897 out of 4763 training samples have
↪<= 512 tokens and thus are subject to application out of the box, whereas 3866 will␣
↪cause an error unless further measures are taken.
```

(continues on next page)

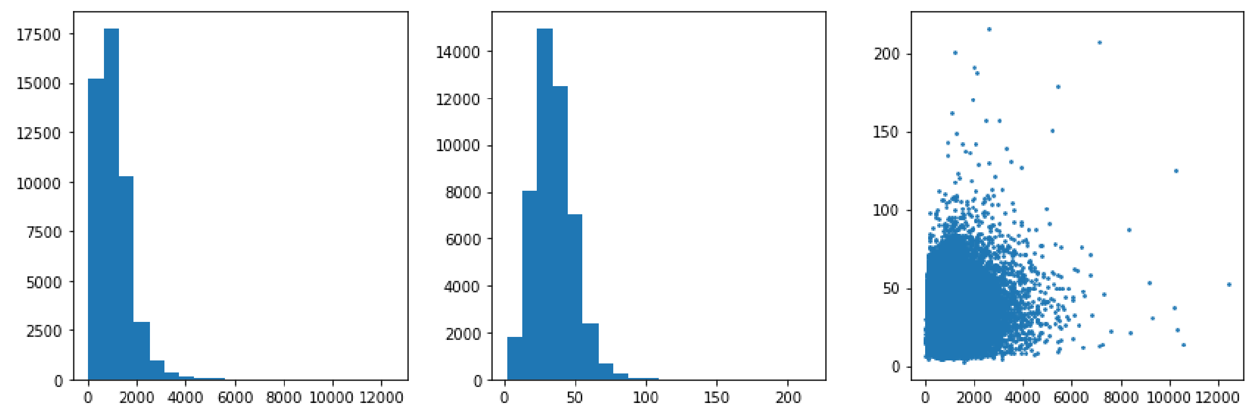
(continued from previous page)

1.2 Recap

Let us bring all data together to draw the final conclusions. To that end, and considering differences between the number of subwords using the different tokenizers, we will repeat the previous process using the training, validation and test sets altogether.

[78]:
```
gpt2_article_tokens = gpt2_article_tokens_train + gpt2_article_tokens_val + gpt2_article_
↪tokens_test
gpt2_summary_tokens = gpt2_summary_tokens_train + gpt2_summary_tokens_val + gpt2_summary_
↪tokens_test
```

[79]:
```
plot_stats(gpt2_article_tokens, gpt2_summary_tokens)
```



[87]:
```
df_gpt2_tokens = pd.DataFrame(list(zip(gpt2_article_tokens,gpt2_summary_tokens)),␣
↪columns=["gpt2_article_tokens", "gpt2_summary_tokens"])
df_gpt2_tokens.describe()
```

[87]:

	gpt2_article_tokens	gpt2_summary_tokens
count	47636.000000	47636.000000
mean	1017.399509	34.834768
std	717.547947	14.036877
min	31.000000	2.000000
25%	506.000000	25.000000
50%	955.000000	34.000000
75%	1379.000000	43.000000
max	12421.000000	216.000000

- Vast majority of the articles are around 500 to 1300 tokens long, whereas regular summary length revolves around 25 to 45 tokens.
- Summary length seems to be independent from the length of the article.
- Mean number of article tokens in the training set: 1017 ± 718. Fairly close to the median: 955 tokens.
- Mean number of summary tokens in the training set: 35 ± 14. The value of the median is very similar: 34.

Some implications of these statistics is that when fine-tuning our models, no matter the data that is used to (i) train the models, (ii) validate or supervise the training process, and to (iii) assess goodness of the resulting models, if we aim at maximising performance metrics like ROUGE, one thing to bear in mind is that the summaries should have a length of around 35 ± 14 tokens. This is essentially due to the fact that, even if we solely use samples that can be entirely fitted into the model (i.e., without the need to trim/remove information from the original article and/or summary), the extent of the summary in terms of no. of subwords is independent of the size of the article. We can, indeed, double check it ourselves:

```
[90]: df_gpt2_tokens[df_gpt2_tokens.gpt2_article_tokens <= gpt2_max_n_tokens].describe()
```

```
[90]:        gpt2_article_tokens  gpt2_summary_tokens
count         12063.000000         12063.000000
mean            231.739120            34.641383
std             131.832378            11.038794
min              31.000000             4.000000
25%             126.000000            27.000000
50%             180.000000            34.000000
75%             337.000000            41.000000
max             512.000000            98.000000
```

CHAPTER

TWO

GPT2SUMMARIZER (GPT2_SUMMARIZER.PY)

class gpt2_summarizer.**GPT2Summarizer**(*checkpoint_name: str*, *batch_size: int*, *num_train_epochs: int*, *gradient_accumulation_steps: int*, *num_workers: int*, *device: torch.device*, *output_dir: Union[str, pathlib.Path]*)

This class serves as a common interface for GPT-2 based fine-tuned architectures for abstractive summarization, both at training and inference time.

Notes

This class is not meant to be instantiated - use instead the specialized subclasses defined for training and inference accordingly.

__init__(*checkpoint_name: str*, *batch_size: int*, *num_train_epochs: int*, *gradient_accumulation_steps: int*, *num_workers: int*, *device: torch.device*, *output_dir: Union[str, pathlib.Path]*) → None

Constructor of a *GPT2Summarizer* instance. This method is solely meant to be invoked by the subclasses implementing this interface.

Parameters

- **checkpoint_name** (*str*) – Model id of a pretrained HuggingFace Transformer hosted inside a model repo on huggingface.co
- **batch_size** (*int*) – Training batch size
- **num_train_epochs** (*int*) – Number of training epochs
- **gradient_accumulation_steps** (*int*) – Number of gradient accumulation steps
- **num_workers** (*int*) – Number of workers available
- **device** (*torch.device*) – torch.device object representing the device on which a torch.Tensor is or will be allocated.
- **output_dir** (*Union[str, Path]*) – Output directory whereby outputs generated at training are stored.

property _config_file

Filepath whereby the configuration file of a fine-tuned model is to be loaded/stored from.

Raises NotImplementedError – To be implemented by all subclasses.

property _model_file

Filepath whereby a model is to be loaded/store from.

Raises NotImplementedError – To be implemented by all subclasses.

`attrs_to_str()`
Meaningful string-like representation of the training attributes to fine-tune the model, serving as a straightforward and effective fashion to uniquely identify the model.

Raises `NotImplementedError` – To be implemented by all subclasses.

`property batch_size: int`
Training batch size, i.e., number of samples processed in parallel by the model.

Returns Training batch size

Return type int

`beam_search`(*context*, *max_length=60*, *beam_size=4*, *temperature=1*)
Performs beam search from *context*, generating up to *max_length* tokens and keeping *beam_size* hypotheses at each generation step.

Parameters

- **context** (*array-like*) – Context tokenized text
- **max_length** (*int, optional*) – Maximum length, in terms of tokens, of the generated summary, by default 60
- **beam_size** (*int, optional*) – Keep the most likely *beam_size* of hypotheses at each generation step to eventually choose the hypothesis that has the overall highest probability, by default 4
- **temperature** (*int, optional*) – Introduce randomness of the predictions by scaling the model logits before applying softmax, by default 1. Values for temperature range in (0,1], where values closer to 1 indicate less randomness

Returns

Return type *beam_size* generated sequences along with their respective scores

Notes

Original code by Rohit Kumar Singh:

(1) https://github.com/SKRohit/Generating_Text_Summary_With_GPT2/blob/master/utils.py

`generate_beam_sample`(*context*, *max_length=60*, *beam_size=4*, *temperature=1*)
Generate summary from *context* with a maximum length of *max_length* tokens using beam search.

Parameters

- **context** (*array-like*) – Context tokenized text
- **max_length** (*int, optional*) – Maximum length, in terms of tokens, of the generated summary, by default 60
- **beam_size** (*int, optional*) – Keep the most likely *beam_size* of hypotheses at each time step to eventually choose the hypothesis that has the overall highest probability, by default 4
- **temperature** (*int, optional*) – Introduce randomness of the predictions by scaling the model logits before applying softmax, by default 1. Values for temperature range in (0,1], where values closer to 1 indicate less randomness

Returns *beam_size* generated sequences sorted in decreasing order of score (larger scores signify better hipothetical quality of summary)

Return type _type_

`generate_sample`(*context*, *max_length=60*, *temperature=1*, *top_k=10*, *top_p=0.5*) → str

Generate summary from *context* with a maximum length of *max_length* tokens

Parameters

- **`context`** (*`array-like`*) – Context tokenized text
- **`max_length`** (*`int, optional`*) – Maximum length, in terms of tokens, of the generated summary, by default 60
- **`temperature`** (*`int, optional`*) – Introduce randomness of the predictions by scaling the model logits before applying softmax, by default 1. Values for temperature range in (0,1], where values closer to 1 indicate less randomness
- **`top_k`** (*`int, optional`*) – Perform top-k filtering (only if *top_k* > 0), by default 10
- **`top_p`** (*`float, optional`*) – Perform nucleus filtering (only if *top_p* > 0), by default 0.5

Returns Generated summary in plain text

Return type str

`generate_sample_huggingface`(*context*, *max_length=60*, ***gen_kwargs*)

Generate summary from *context* with a maximum length of *max_length* tokens using HuggingFace's functions for text generation

Parameters

- **`context`** (*`array-like`*) – Context tokenized text
- **`max_length`** (*`int, optional`*) – Maximum length, in terms of tokens, of the generated summary, by default 60

Returns Generated summary in plain text

Return type str

`generate_summaries_from_dataset`(*test_data_dir: dataset.GPT2SumDataset*, *max_length=100*, *temperature=1*, *top_k=10*, *top_p=0.5*, *out_path: Optional[Union[str, pathlib.Path]] = None*) → None

Generate summaries from test data and store them in disk.

Parameters

- **`test_data_dir`** (*`array-like`*) – Test data from which samples are withdrawn
- **`max_length`** (*`int, optional`*) – Maximum length, in terms of tokens, of the generated summary, by default 100
- **`temperature`** (*`int, optional`*) – Introduce randomness of the predictions by scaling the model logits before applying softmax, by default 1. Values for temperature range in (0,1], where values closer to 1 indicate less randomness
- **`top_k`** (*`int, optional`*) – Perform top-k filtering (only if *top_k* > 0), by default 0
- **`top_p`** (*`float, optional`*) – Perform nucleus filtering (only if *top_p* > 0), by default 0.0
- **`out_path`** (*`Union[str, Path], optional`*) – Custom filepath to store the generated summaries, by default None

`property gradient_accumulation_steps: int`

Number of K mini-batches of size *batch_size* to run before performing a backward pass.

Returns Number of gradient accumulation steps

Return type int

`log_generated_summaries`(*data*, *num=1*, *eval_step=False*, *max_length=60*, *temperature=1*, *top_k=10*, *top_p=0.5*) → None

Log *num* generated summaries from *data*.

Parameters

- **data** (*array-like*) – Dataset from which samples are withdrawn
- **num** (*int, optional*) – number of summaries to generate (and log), by default 1
- **eval_step** (*bool, optional*) – Whether to log the article and actual summary, by default False
- **max_length** (*int, optional*) – Maximum length, in terms of tokens, of the generated summary, by default 100
- **temperature** (*int, optional*) – Introduce randomness of the predictions by scaling the model logits before applying softmax, by default 1. Values for temperature range in (0,1], where values closer to 1 indicate less randomness
- **top_k** (*int, optional*) – Perform top-k filtering (only if *top_k* > 0), by default 10
- **top_p** (*float, optional*) – Perform nucleus filtering (only if *top_p* > 0), by default 0.5

Notes

Code adaptation by Rohit Kumar's work:

(1) https://github.com/SKRohit/Generating_Text_Summary_With_GPT2/blob/master/utils.py

`property num_train_epochs: int`

Number of training epochs, i.e., number of complete passes through the entire training dataset.

Returns Number of training epochs

Return type int

`property num_workers: int`

Number of processing elements (typically in terms of number of CPU cores available) at your disposal to process data loading in parallel. In practice, *num_workers* equals the number of samples that can be loaded in parallel.

Returns Number of workers

Return type int

Notes

Multi-process data loading (pytorch): https://pytorch.org/docs/stable/data.html#multi-process-data-loading

`property output_dir: pathlib.Path`

Path of the directory of the generated model and training statistics.

Returns Directory whereby the trained models, along with their configuration and other statistics are allocated

Return type Path

`sample_sequence`(*context*, *length: int*, *temperature=1*, *top_k=0*, *top_p=0.0*) → torch.Tensor

Generate *length* new tokens based on a context (*context*).

Parameters

- **`context`** (*`array-like`*) – Context tokenized text
- **`length`** (*`int`*) – Number of tokens to generate
- **`temperature`** (*`int, optional`*) – Introduce randomness of the predictions by scaling the model logits before applying softmax, by default 1. Values for temperature range in (0,1], where values closer to 1 indicate less randomness
- **`top_k`** (*`int, optional`*) – Perform top-k filtering (only if *top_k* > 0), by default 0
- **`top_p`** (*`float, optional`*) – Perform nucleus filtering (only if *top_p* > 0), by default 0.0

Returns Tensor containing the tokenized text of the context and the generated tokens

Return type torch.Tensor

Notes

Original code by Thomas Wolf:

(1) https://github.com/huggingface/transformers/blob/5c3b32d44d0164aaa9b91405f48e53cf53a82b35/examples/run_generation.py

`tokenize_input``()` → Callable

Ensure text is tokenized before feeding it into the model for summary generation.

Parameters **`func`** (*`Callable`*) – Callable function

Returns Callable object with the appropiate parametrization

Return type Callable

`property tokenizer`

HuggingFace Tokenizer object, which is targeted at preparing the inputs for a model. By default, the tokenizer to use is that of the checkpoint (pre-trained model)

Returns Tokenizer for the model

Return type Any

Notes

Documentation of HuggingFace Tokenizer class: https://huggingface.co/docs/transformers/main_classes/tokenizer

`top_k_top_p_filtering``(`*logits: torch.Tensor*, *top_k=0*, *top_p=0.0*, *filter_value=- inf*`)` → torch.Tensor

Filter a distribution of logits using top-k and/or nucleus (top-p) filtering

Parameters

- **`logits`** (*`torch.Tensor`*) – Logits distribution shape (vocabulary size)
- **`top_k`** (*`int, optional`*) – Keep only top k tokens with highest probability (top-k filtering), by default 0. Top-k filtering is performed for *top_k* > 0
- **`top_p`** (*`float, optional`*) – Keep the top tokens with cumulative probability >= top_p (nucleus filtering), by default 0.0. Nucleus filtering is performed for *top_p* > 0
- **`filter_value`** (*`float, optional`*) – Logits filter value, by default -float('Inf')

Returns Filtered distribution of logits

Return type torch.Tensor

Notes

Original code by Thomas Wolf:

(1) https://gist.github.com/thomwolf/1a5a29f6962089e871b94cbd09daf317

(2) https://github.com/huggingface/transformers/blob/5c3b32d44d0164aaa9b91405f48e53cf53a82b35/examples/run_generation.py

CHAPTER

THREE

TRAINGPT2SUMMARIZER (GPT2_SUMMARIZER_TRAIN.PY)

class gpt2_summarizer_train.**TrainGPT2Summarizer**(*checkpoint_name: str*, *data_dir: Union[str, pathlib.Path]*, *batch_size: int*, *num_train_epochs: int*, *gradient_accumulation_steps: int*, *max_grad_norm: float*, *lr: float*, *n_gpu: int*, *num_workers: int*, *device: torch.device*, *output_dir: Union[str, pathlib.Path]*, *seed: int*)

This class provides a basic interface to train a GPT-2 based pre-trained model for abstractive summarization, whereby fine-tuning can be simply achieved by instantiating a class object and subsequently invoking the *train* function.

__init__(*checkpoint_name: str*, *data_dir: Union[str, pathlib.Path]*, *batch_size: int*, *num_train_epochs: int*, *gradient_accumulation_steps: int*, *max_grad_norm: float*, *lr: float*, *n_gpu: int*, *num_workers: int*, *device: torch.device*, *output_dir: Union[str, pathlib.Path]*, *seed: int*) → None

Constructor of a *TrainGPT2Summarizer* instance, which provides all necessary functionality to allow for the fine-grained tuning of a GPT2-like architecture for abstractive summarization. A prior step to train any model is to have data well formatted. To that end, please refer to the *prepare_data.py* module and its documentation, should you require it.

Parameters

- **checkpoint_name** (*str*) – Model id of a pretrained HuggingFace Transformer hosted inside a model repo on huggingface.co
- **data_dir** (*Union[str, Path]*) – Parent directory containing at least the training and validation datasets to fine tune the model. The data should be formatted in such way that it can be processed by a *GPT2SumDataset* object. Refer to the *prepare_data.py* script for further information.
- **batch_size** (*int*) – Training batch size
- **num_train_epochs** (*int*) – Number of training epochs
- **gradient_accumulation_steps** (*int*) – Number of gradient accumulation steps
- **max_grad_norm** (*float*) – Max norm of the gradients. This helps leveraging the exploding gradients problem, whereby large gradient vectors are rescaled so that their norm is at most *max_grad_norm*
- **lr** (*float*) – Initial learning rate
- **n_gpu** (*int*) – Number of GPUs available
- **num_workers** (*int*) – Number of workers available
- **device** (*torch.device*) – torch.device object representing the device on which a torch.Tensor is or will be allocated.

- **output_dir** (*Union[str, Path]*) – Output directory whereby outputs generated at training are stored.
- **seed** (*int*) – Initialization state of a pseudo-random number generator to grant reproducibility of the experiments

property _config_file: pathlib.Path
Output filepath of the configuration file (in *json* format) of the trained model. This file is dumped to the output directory (refer to *self.output_dir* prop.), and named after the "named" parameters utilized at training (refer to *self.attrs_to_str()* module).

Returns Output filepath for the configuration file of the fine-tuned model

Return type Path

property _model_file: pathlib.Path
Output filepath for the trained model binary (in *bin* format). This file is dumped to the output directory (refer to *self.output_dir* prop.), and named after the "named" parameters utilized at training (refer to *self.attrs_to_str()* module).

Returns Output filepath for the trained model binary

Return type Path

_save_train_stats(*training_stats: list*, *precision=4*) → pandas.core.frame.DataFrame
Save the statistics generated throughout the training process.

Parameters

- **training_stats** (*list*) – Collection of per-epoch statistics
- **precision** (*int, optional*) – Number of decimal places to use for floating-point valued fields, by default 4

Returns Statistics generated throughout the training process arranged in a DataFrame

Return type pd.DataFrame

attrs_to_str(*add: Optional[str] = None*, *test_data_name=''*) → str
Yield a string-like representation of the training attributes to fine-tune the model, serving as a straightforward and effective fashion to uniquely identify the model.

Parameters

- **add** (*str, optional*) – Substring to append at the end of the string model descriptor, by default None
- **test_data_name** (*str*) – Identifier of the test dataset to generate the necessary filepaths. By default ""

Returns Textual representation of the training parameters utilized for fine-tuning

Return type str

compute_loss(*logits: torch.Tensor*, *labels: torch.Tensor*, *sum_idx: torch.Tensor*) → float
Compute loss over the logits w.r.t. truth labels, considering to that end the subset of scores yielded for reference summaries.

Parameters

- **logits** (*torch.Tensor*) – Raw, unnormalized scores outputted by the last layer of the model.
- **labels** (*torch.Tensor*) – Collection of labels (tokenized actual summaries)
- **sum_idx** (*torch.Tensor*) – Per sample article/summary separator index

Returns Computed loss

Return type float

`eval()` → Tuple[torch.Tensor, float, float]

Evaluate performance of the model on the validation set

Returns Perplexity of the model, average loss and elapsed time

Return type Tuple[torch.Tensor, float, float]

`property loss_func: Callable`

Function that computes the cross entropy loss between input and target, ignoring the padding token.

Returns Cross entropy loss function

Return type Callable

`property max_grad_norm: float`

Max norm of the gradients. This helps leveraging the exploding gradients problem, whereby large gradient vectors are rescaled so that their norm is at most *max_grad_norm*.

Returns Max norm of the gradients

Return type float

`save_trained_model`(*model_file: Optional[Union[str, pathlib.Path]] = None*, *config_file: Optional[Union[str, pathlib.Path]] = None*) → None

Dump trained model (in *bin* format) and the configuration parameters (in *json* format) of the GPT2 model.

Parameters

- **model_file** (*Union[str,Path], optional*) – Custom ouput model filepath, if not specified, it is dumped to *self._model_file*, by default None
- **config_file** (*Union[str,Path], optional*) – Custom ouput configuration filepath, if not specified, it is dumped to *self._config_file*, by default None

`train`(*num_warmup_steps=200*, *num_training_steps=80000*) → pandas.core.frame.DataFrame

Train a GPT-like architecture to generate abstractive summaries utilizing the AdamW (Decoupled Weight Decay Regularization) optimizer, gradient clipping, cross-entropy loss and a linear scheduler, with a learning rate that decreases linearly from the initial lr set in the optimizer to 0, after a warmup period (*num_warmup_steps*) and during *num_training_steps* training steps, where the learning rate increases from 0 to the initial lr set in the optimizer.

Parameters

- **num_warmup_steps** (*int, optional*) – Number of steps for the warmup phase of the linear scheduler, by default 100
- **num_training_steps** (*int, optional*) – Total number of training steps for the linear scheduler, by default 80000

Returns Statistics generated throughout the training process arranged in a DataFrame

Return type pd.DataFrame

Notes

The code for this method is largely based on the following work: (1) https://mccormickml.com/2019/07/22/BERT-fine-tuning/#43-training-loop (2) https://colab.research.google.com/github/kozodoi/website/blob/master/_notebooks/2021-02-19-gradient-accumulation.ipynb#scrollTo=ISFvH2p8dqYQ (3) https://github.com/SKRohit/Generating_Text_Summary_With_GPT2/blob/master/train_gpt2_summarizer.py

CHAPTER

FOUR

INFERENCEGPT2SUMMARIZER (GPT2_SUMMARIZER_INFERENCE.PY)

class gpt2_summarizer_inference.**InferenceGPT2Summarizer**(*checkpoint_name: str*, *train_data_name: str*, *batch_size: int*, *num_train_epochs: int*, *gradient_accumulation_steps: int*, *num_workers: int*, *device: torch.device*, *output_dir: Union[str, pathlib.Path]*)

__init__(*checkpoint_name: str*, *train_data_name: str*, *batch_size: int*, *num_train_epochs: int*, *gradient_accumulation_steps: int*, *num_workers: int*, *device: torch.device*, *output_dir: Union[str, pathlib.Path]*) → None

Constructor of a *InferenceGPT2Summarizer* instance, which provides all necessary functionality to allow the use of a fine-tuned GPT2-like architecture for abstractive summarization at inference i.e., to generate summaries on unseen data, both in an as-is basis (see *self.generate_sample*) or provided a *GPT2SumDataset* (see *self.generate_summaries_from_dataset*).

Parameters

- **checkpoint_name** (*str*) – Model id of a pretrained HuggingFace Transformer hosted inside a model repo on huggingface.co
- **train_data_name** (*str*) – Identifier of the training dataset on which the model has been trained.
- **batch_size** (*int*) – Training batch size
- **num_train_epochs** (*int*) – Number of training epochs
- **gradient_accumulation_steps** (*int*) – Number of gradient accumulation steps
- **num_workers** (*int*) – Number of workers available
- **device** (*torch.device*) – torch.device object representing the device on which a torch.Tensor is or will be allocated.
- **output_dir** (*Union[str, Path]*) – Output directory whereby outputs generated at training are stored.

property _config_file: pathlib.Path

Filepath of the configuration file (in *json* format) of the trained model. This file is loaded from the output directory (refer to *self.output_dir* prop.), and named after the "named" parameters utilized at training (refer to *self.attrs_to_str()* module).

Returns Input filepath for the configuration file of the fine-tuned model

Return type Path

property _model_file: pathlib.Path

Filepath for the trained model binary (in *bin* format). This file is loaded from the output directory (refer to *self.output_dir* prop.), and named after the "named" parameters utilized at training (refer to *self.attrs_to_str()* module).

Returns Input filepath for the trained model binary

Return type Path

`attrs_to_str`(*add: Optional[str] = None*, *test_data_name=''*) → str

Yield a string-like representation of the training attributes to fine-tune the model, serving as a straightforward and effective fashion to uniquely identify the model.

Parameters

- `add` (*str, optional*) – Substring to append at the end of the string model descriptor, by default None
- `test_data_name` (*str*) – Identifier of the test dataset to generate the necessary filepaths. By default, ""

Returns Textual representation of the training parameters utilized for fine-tuning

Return type str

`property config: transformers.models.gpt2.configuration_gpt2.GPT2Config`

GPT2 configuration object. It is used to instantiate a GPT-2 model according to the specified arguments, defining the model architecture.

Returns GPT2 configuration object

Return type GPT2Config

`property model: transformers.models.gpt2.modeling_gpt2.GPT2LMHeadModel`

GPT2 model fine tuned for abstractive summarization

Returns GPT2 model fine tuned for abstractive summarization

Return type GPT2LMHeadModel

`setup_model`(*config: Optional[transformers.models.gpt2.configuration_gpt2.GPT2Config] = None*, *state_dict: Optional[dict] = None*) → None

Setup fine-tuned model from the configuration object and the learned parameters.

Parameters

- `config` (*GPT2Config, optional*) – Custom model architecture configuration object. If not specified, the configuration used is that of *self.config*, which instantiates a *GPT2Config* object from *self._config_file* json configuration file
- `state_dict` (*dict, optional*) – Custom *state_dict* object. If not specified, the learned parameters used are those of *self.state_dict*, which are deserialized from *self._model_file*

`property state_dict: dict`

Python dictionary object that maps each layer with learnable parameters (i.e., weights and biases) to its parameter tensor.

Returns Model learnable parameters

Return type dict

CHAPTER FIVE

UTILS (UTILS.PY)

`utils.add_special_tokens`(*tokenizer='PlanTL-GOB-ES/gpt2-base-bne'*)

Returns GPT2 tokenizer after adding separator and padding tokens

Parameters **tokenizer** (`str, optional`) – Model id of a pretrained HuggingFace tokenizer hosted inside a model repo on huggingface.co , by default "PlanTL-GOB-ES/gpt2-base-bne"

Returns

Return type GPT2 tokenizer after adding separator and padding tokens

`utils.compute_rouge_score`(*test_summaries: Iterable*, *generated: Iterable*, *results_path: Union[str, pathlib.Path]*, *score_format='csv'*)

Compute average ROUGE scores of *generated* summaries, using *test_summaries* as reference.

Parameters

- **test_summaries** (`Iterable`) – Reference summaries
- **generated** (`Iterable`) – Generated summaries
- **results_path** (`Union[str, Path]`) – Filepath to store ROUGE metrics
- **score_format** (`str, optional`) – Format to store assessment results, by default "csv"

`utils.detokenize_input`(*func: Callable*) → Callable

Detokenize text input, when required.

Parameters **func** (`Callable`) – Callable function

Returns Callable object with the appropiate parametrization

Return type Callable

`utils.format_time`(*elapsed: float*) → str

Format time in seconds to hh:mm:ss time format.

Parameters **elapsed** (`float`) – Time in seconds

Returns Time formatted in hh:mm:ss

Return type str

`utils.get_tokenized_text`(*text: str*, *tokenizer*, *convert_ids_to_tokens=False*) → list

Returns tokenized text using the tokenizer *tokenizer*

Parameters

- **text** (`str`) – Text to tokenize
- **tokenizer** (*PreTrainedTokenizer* or *PreTrainedTokenizerFast*) – Tokenizer

- **convert_ids_to_tokens** (*bool, optional*) – Whether to convert ids to tokens, by default False

Returns Tokenized text

Return type list

utils.**load_serialized_data**(*filename: str*, *return_dict_values: bool = False*) → Union[dict, Any]

Utility to load serialized data (and other optional stored values) from disk using *pickle*.

Parameters

- **filename** (*str*) – Filename of the file to be loaded.
- **return_dict_values** (*bool, optional*) – If set to True, returns the values just the values of the dictionary containing all stored data, defaults to False.

Returns Loaded data

Return type Union[dict, Any]

utils.**makedir**(*path: Union[str, pathlib.Path]*, *remove_filename: bool = False*, *recursive: bool = True*, *exist_ok: bool = True*) → None

Creates directory from path if not exists.

Parameters

- **path** (*Union[str,Path]*) – Path of the directory to be created.
- **remove_filename** (*bool, optional*) – If set to True, it attempts to remove the filename from the path, defaults to False
- **recursive** (*bool, optional*) – Creates directories recursively (i.e., create necessary subdirectories if necessary), by default True
- **exist_ok** (*bool, optional*) – If set to False, it arises an error if *path* directory exists, by default True

utils.**prepare_input_summarizer**(*text: str*, *tokenizer*, *max_input=512*, *gpt2_summary_length: int = None*, *as_tokens=False*, ***spacy_kwargs*)

Prepare input to be handled by the Transformer model

Parameters

- **text** (*str*) – Raw, unprocessed text
- **tokenizer** (*PreTrainedTokenizer* or *PreTrainedTokenizerFast*) – Tokenizer object used to control for the number of tokens in *text*
- **max_input** (*int, optional*) – Maximum length, in terms of tokens, that the input can handle, by default 512
- **gpt2_summary_length** (*int, optional*) – Length, in terms of tokens, reserved to generate a summary when using decoder-only based summarizers (e.g., GPT-2), by default None. If you wish to use a different architecture (e.g., encoder-decoder), do NOT specify this argument.
- **as_tokens** (*bool, optional*) – Return the input as tokens (hence not as plain text).

Returns Prepared input to be handled by a Transformer and a flag indicating whether the text has been trimmed.

Return type (Union[str,torch.Tensor], bool)

utils.**set_seed**(*seed: int*, *gpu_mode: bool*)

Set initialization state of a pseudo-random number generator to grant reproducibility of the experiments

Parameters

- **seed** (*int*) – Seed
- **gpu_mode** (*bool*) – Whether there are GPU's available

utils.**split_text_into_sentences_spacy**(*text*, *spacy_model='es_core_news_sm'*)
Splits text into sentences using the Spacy library.

Parameters

- **text** (*str*) – Text to be splitted into sentences.
- **spacy_model** (*str or SpaCy pretrained model object, optional*) – SpaCy pretrained model used to split text into sentences or pretrained model identifier, defaults to 'es_core_news_sm'.

Returns List of sentences in *text*.

Return type list

Notes

SpaCy builds a syntactic tree for each sentence, a robust method that yields more statistical information about the text than NLTK. It performs substancially better than NLTK when using not polished text.

utils.**store_serialized_data**(*data*, *out_filename*, *protocol: int = 5*) → None
Utility to dump precomputed data to disk using *pickle*.

Parameters

- **data** (*_type_*) – Data to serialize
- **out_filename** (*str, optional*) – Path for the output file
- **protocol** (*int, optional*) – Protocol used for *pickle*, by default pickle.HIGHEST_PROTOCOL

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

u