自然語言處理與應用
Natural Language Processing and Applications

Hugging Face Tutorial (II)

Instructor: 林英嘉 (Ying-Jia Lin)
2025/04/28

Course GitHub

Slido # NLP_0428

What you will learn in this tutorial

- Training and evaluating GPT-2 / T5 on abstractive summarization
  - Dataset: Chinese abstractive summarization
  - Objective: Cross-entropy
  - Main packages: PyTorch, Hugging Face, ROUGE

Why do we need to learn GPT-2?

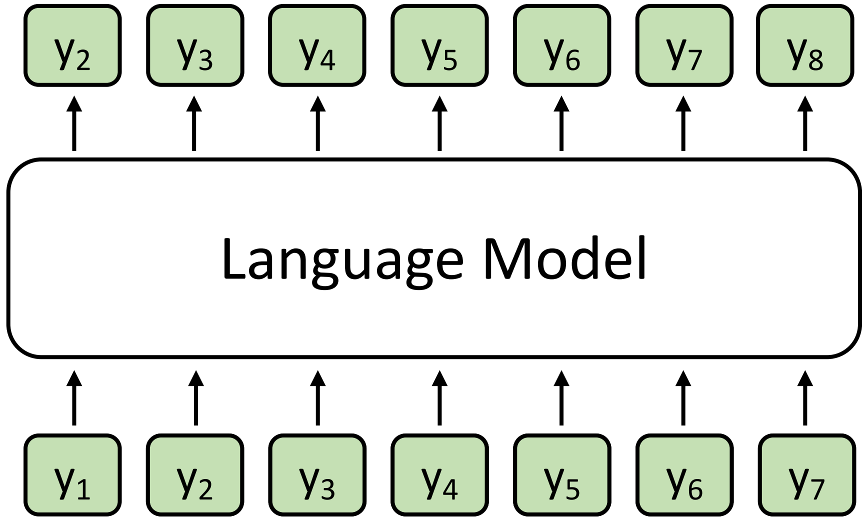
- GPT-2 is the **last** open-source model in the GPT series from OpenAI.
- Language generation is hard, and GPT-2 is a good start point.
- GPT-2 is not big. It is feasible for low-budget machines.
  - GPT-2 (12-layer; 124M), GPT-2-medium (24-layer; 345M), GPT-2-large (36-layer; 762M), GPT-2-xl (48-layer; 1.5B)

[Recap] 訓練模型最大化每個時間點的機率

模型如何進行預先訓練 (pre-training)？

$$P(y_t|y_1, y_2, \dots, y_{t-1})$$ ← Next-token prediction

$P(y_2|\, y_1)$ $P(y_3|y_1, y_2)$ $\cdots$ $P(y_8|y_1, y_2, y_3, y_4, y_5, y_6, y_7)$



Self-attention 不可及的範圍

目標函數：

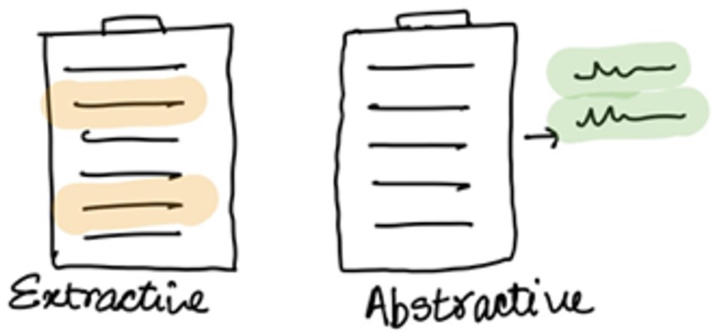
$$\prod_{t=1}^{n} P(y_t|y_1, y_2, \dots, y_{t-1})$$ ← Language Modeling

= Generative Pre-training (GPT)

NLP

Introduction to Text Summarization

- Extractive summarization
  - Generate a short text summary for a document by selecting salient sentences in the documents.
- Abstractive summarization
  - Generate novel words and phrases not featured in the source text.



Example:
The Queen's Birthday holiday road toll stands at zero for the first time since records began.

Ext summary: The Queen's holiday road toll stands at zero.
Abs summary: The Queen's holiday slashes road toll.

Task: Chinese Abstractive Summarization

- Dataset: LCSTS (A Large-Scale Chinese Short Text Summarization Dataset)[1]

Short Text: 水利部水资源司司长陈明忠今日在新闻发布会上透露，根据刚刚完成的水资源管理制度的考核，有部分省接近了红线的指标，有部分省超过红线的指标。在一些超过红线的地方，将对一些取用水项目进行区域的限批，严格地进行水资源论证和取水许可的批准。

Summarization: 部分省超过年度用水红线指标 取水项目将被限批

[1] Hu, Baotian, Qingcai Chen, and Fangze Zhu. "LCSTS: A Large Scale Chinese Short Text Summarization Dataset." Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. 2015.

Hugging Face | Search models, datasets, users... | Models | Datasets | Spaces | Posts | Docs | Pricing | Log In | Sign Up

Documentations

https://huggingface.co/docs

Search across all docs

- **Transformers**
  State-of-the-art ML for Pytorch, TensorFlow, and JAX.
- **Diffusers**
  State-of-the-art diffusion models for image and audio generation in PyTorch.
- **Hub**
  Host Git-based models, datasets and Spaces on the Hugging Face Hub.
- **Datasets**
  Access and share datasets for computer vision, audio, and NLP tasks.
- **Gradio**
  Build machine learning demos and other web apps, in just a few lines of Python.
- **Hub Python Library**
  Client library for the HF Hub: manage repositories from your Python runtime.
- **Huggingface.js**
  A collection of JS libraries to interact with Hugging Face, with TS types included.
- **Transformers.js**
  Community library to run pretrained models from Transformers in your browser.
- **Inference API (serverless)**
  Experiment with over 200k models easily using the serverless tier of Inference Endpoints.
- **Inference Endpoints (dedicated)**
  Easily deploy models to production on dedicated, fully managed infrastructure.
- **PEFT**
  Parameter efficient finetuning methods for large models.

Installation

Basically, you need to install PyTorch first before you install transformers.

(Recommended)

```
pip install transformers
```

(If you want to try new things)

```
git clone
https://github.com/huggingface/transformers.git
cd transformers
pip install -e .
```

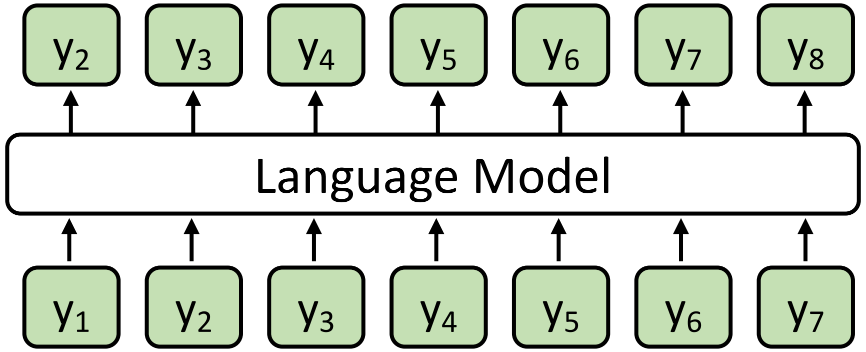
Packages required in this tutorial

We need the following packages for this tutorial. On Colab, you may not need to re-install these packages for they may have already been installed.

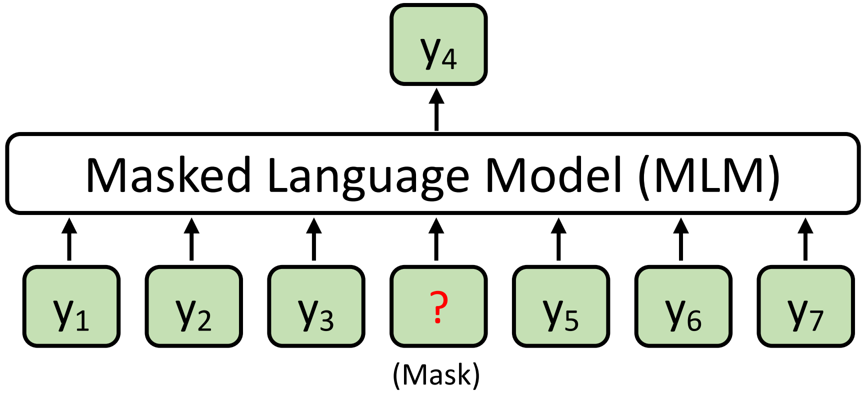
```
!pip install torch==2.3.1 --index-url
https://download.pytorch.org/whl/cu121
!pip install transformers==4.37.0
!pip install datasets==2.21.0
!pip install accelerate==0.21.0
!pip install rouge==1.0.1
!pip install tqdm==4.66.5
!pip install jieba==0.42.1
```

Language Model vs. Masked Language Model

Casual Language Model
(E.g., GPT)

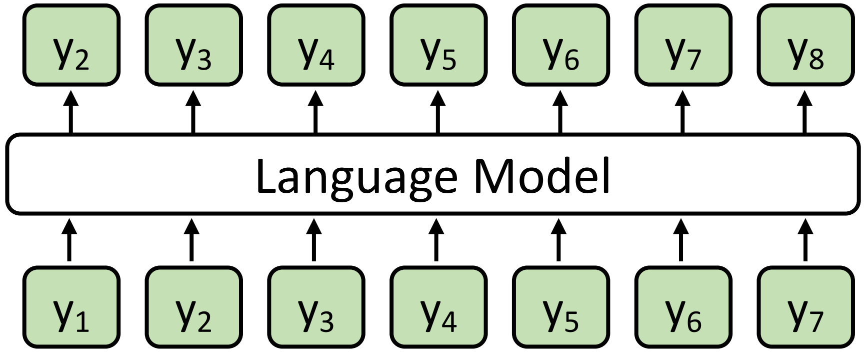


Masked Language Model
(E.g., BERT)

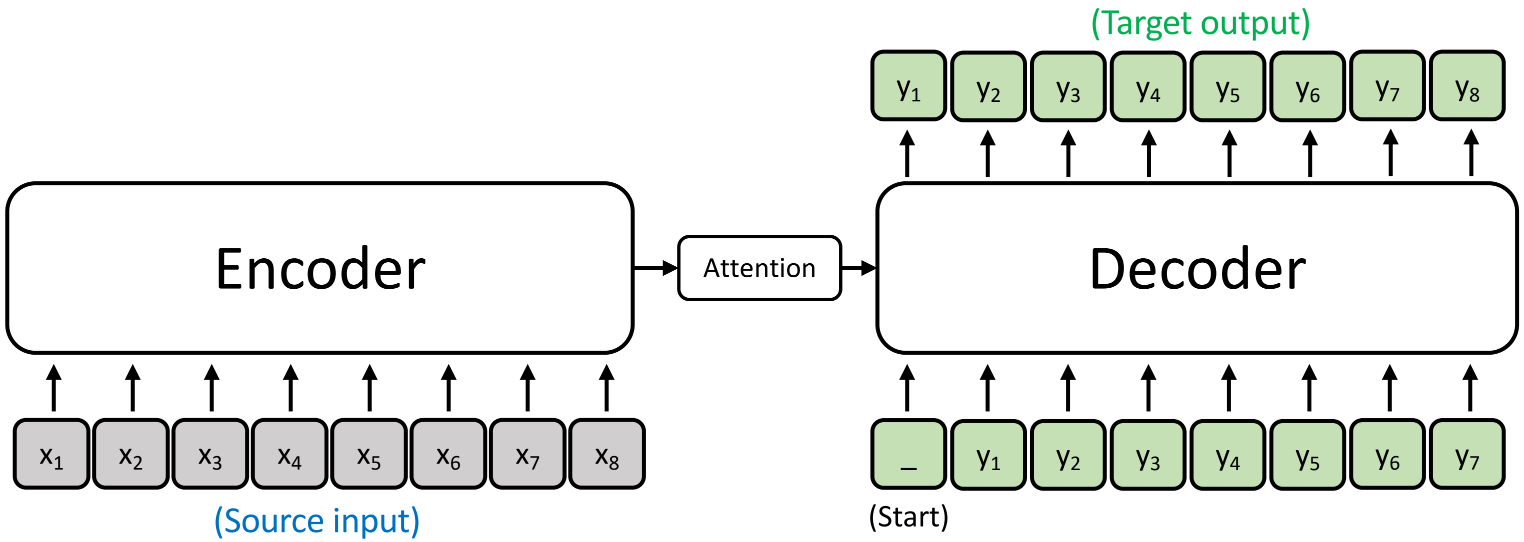


Language Model vs. Seq2seq Model

Casual Language Model
(E.g., GPT)



Sequence to sequence Model
(E.g., T5; Vanilla Transformers)



Load LCSTS via Hugging Face Dataset

- [Hugging Face dataset link]

```
raw_train = load_dataset(
    "hugcyp/LCSTS", split="train", cache_dir="./cache/"
).to_list()
raw_val = load_dataset(
    "hugcyp/LCSTS", split="validation", cache_dir="./cache/"
).to_list()
```

Sometimes checking the Hugging Face dataset is slow, it will be faster if we transform the dataset object into a list using `.to_list()`.

Let’s try it

1. 用 load_dataset() 把 train 和 validation 分開讀進來
2. 把 dataset 轉成 pandas dataframe，觀察 1~2 筆 samples

Auto Classes (models, tokenizers)

- In many cases, the architecture you want to use **can be guessed** from the name or the path of the pretrained model you are supplying to the from_pretrained method.
- AutoClasses are here to do this job for you so that you automatically retrieve the relevant model given the name/path to the pretrained weights/config/vocabulary:
  - Ex: model = AutoModel.from_pretrained('bert-base-cased') will create an instance of the BERT base model (cased).

https://huggingface.co/transformers/v3.0.2/model_doc/auto.html

Generative Models in Auto Classes

Casual Language Model (E.g., GPT, Llama)	AutoModelForCausalLM (link) GPT2LMHeadModel (link) LlamaForCausalLM (link)
Masked Language Model (E.g., BERT)	AutoModelForMaskedLM (link) BertForMaskedLM (link)
Sequence to sequence Model (E.g., T5; Vanilla Transformers)	AutoModelForSeq2SeqLM (link) T5ForConditionalGeneration (link)

Outline

- GPT-2 (native PyTorch)
- T5 (Hugging Face Dataset and Trainer)

Import packages

```python
from transformers import AutoTokenizer
from transformers import GPT2LMHeadModel
from datasets import load_dataset
from tqdm import tqdm
import torch
from torch.utils.tensorboard import SummaryWriter
from rouge import Rouge
import jieba
```

Load the Tokenizer and the Model

- Hugging Face model link

```
model_name = "uer/gpt2-chinese-cluecorpussmall"
tokenizer = AutoTokenizer.from_pretrained(
    model_name,
    padding_side="left”,
)
tokenizer.add_special_tokens({"eos_token": "<|endoftext|>"})
model.resize_token_embeddings(len(tokenizer))
```

Optional

Let’s try it

1. 載入模型
2. print 出 model config

Left padding

In text generation, sometimes we set padding_side='left', when loading tokenizer.

If padding is at the end of the prompt, new tokens may be generated after the padding, which is illogical.

<s>	Recite	the	first	law	<s>	<pad>	<pad>	<pad>	
<s>	How	are	you	<s>	<pad>	<pad>	<pad>	<pad>	
<s>	Who	is	the	first	president	of	U.S.	<s>	

Left-padding

Aligning the new tokens.

<pad>	<pad>	<pad>	<s>	Recite	the	first	law	<s>	
<pad>	<pad>	<pad>	<pad>	<s>	How	are	you	<s>	
<s>	Who	is	the	first	president	of	U.S.	<s>	

About Padding

	Fine-tuning	Test time (inference)
策略一	Left padding + set [PAD] as -100 in labels (this tutorial)	Left padding
策略二	Right padding + set [PAD] as -100 in labels	Use test_batch_size as 1 (this tutorial)

- Fine-tuning 的策略一可搭配 Test time 的任一策略
- Fine-tuning 的策略二只建議搭配 Test time 的策略二

Set [PAD] as -100 in labels

https://github.com/huggingface/transformers/blob/main/src/transformers/models/gpt2/modeling_gpt2.py#L1264-L1267

```
r"""
labels (`torch.LongTensor` of shape `(batch_size, sequence_length)`, *optional*):
    Labels for language modeling. Note that the labels **are shifted** inside the model, i.e. you can set
    `labels = input_ids` Indices are selected in `[-100, 0, ..., config.vocab_size]` All labels set to `-100`
    are ignored (masked), the loss is only computed for labels in `[0, ..., config.vocab_size]`
"""
```

Load the Tokenizer and the Model

- [Hugging Face model link]()

```
model_name = "uer/gpt2-chinese-cluecorpussmall"
tokenizer = AutoTokenizer.from_pretrained(
    model_name,
    padding_side="left",
)
tokenizer.add_special_tokens({"eos_token": "<|endoftext|>"})
model.resize_token_embeddings(len(tokenizer))
```

Increasing the size will add newly initialized vectors at the end.
Reducing the size will remove vectors from the end. (link)

resize_token_embeddings

- 新增 embedding 的情況：
  - 新增的 embedding 還沒有被訓練過
  - 透過 torch 的 nn.Embedding 來新增 token 的向量
  - https://github.com/huggingface/transformers/blob/v4.51.3/src/transformers/modeling_utils.py#L2802-L2807
  - torch 的 nn.Embedding 的初始化範圍是 normal distribution 0到1之間

Let’s try it

1. 載入 tokenizer
2. 測試句子：「今天天氣真好」，print 出 tokenized tokens

Create the PyTorch Dataset

- You can use the Hugging Face dataset without building a PyTorch dataset class.
- However, language generation is really complicated. We suggest building your first code with native PyTorch.

```python
class LCSTSDataset(torch.utils.data.Dataset):
    def __init__(self, raw_data) -> None:
        super().__init__()
        self.data = raw_data
        self.token_replacement = [
            ["：", ":"],
            ["，", ","],
            ["“", '"'],
            ["”", '"'],
            ["？", "?"],
            ["……", "..."],
            ["！", "!"],
        ]

    def __getitem__(self, index):
        d = self.data[index]
        # Substitute some full-width punctuations with half-width ones
        for k in d:
            for tok in self.token_replacement:
                d[k] = d[k].replace(tok[0], tok[1])
        return d

    def __len__(self):
        return len(self.data)
```

To prevent out-of-vocabulary tokens from being transformed into [UNK]

Run:

```python
train_set = LCSTSDataset(raw_train)
val_set = LCSTSDataset(raw_val)
```

Create the PyTorch DataLoader for Data Batching

1. Input data for training (source + summary)
2. Labels (auto-regressive; thus, basically input_ids themselves)
3. Input data for predictions (source only)

Overview

In [ ]:

```
def collate_fn(batch):
    complete_text = [
        f"[CLS]{example['text']}[SEP]{example['summary']}<|endoftext|>"
        for example in batch
    ]
    complete_text = tokenizer.batch_encode_plus(
        complete_text,
        padding=True,
        truncation=True,
        return_tensors="pt",
        add_special_tokens=False,
    )
    # Set label padding tokens to -100 for loss masking
    labels = torch.where(
        condition=complete_text.input_ids != tokenizer.pad_token_id,
        input=complete_text.input_ids,
        other=-100,
    )
    complete_text["labels"] = labels
    complete_text = {k: complete_text[k].to(device) for k in complete_text}

    infer_text = [example["text"] for example in batch]
    infer_text = tokenizer.batch_encode_plus(
        infer_text,
        padding=True,
        truncation=True,
        return_tensors="pt",
    )
    infer_text = {k: infer_text[k].to(device) for k in infer_text}
    return complete_text, infer_text
```

Create the PyTorch DataLoader for Data Batching: 1. Input Data for Training

```python
def collate_fn(batch):
    complete_text = [
        f"[CLS]{example['text']}[SEP]{example['summary']}<|endoftext|>"
        for example in batch
    ]
    complete_text = tokenizer.batch_encode_plus(
        complete_text,
        padding=True,
        truncation=True,
        return_tensors="pt",
        add_special_tokens=False,
    )
```

You can omit the [CLS] token before fine-tuning!

[CLS]	Source article	[SEP]	Summary	<|endoftext|>

uer/gpt2-chinese-cluecorpussmall uses BertTokenizer

https://huggingface.co/uer/gpt2-chinese-cluecorpussmall/blob/main/config.json#L26

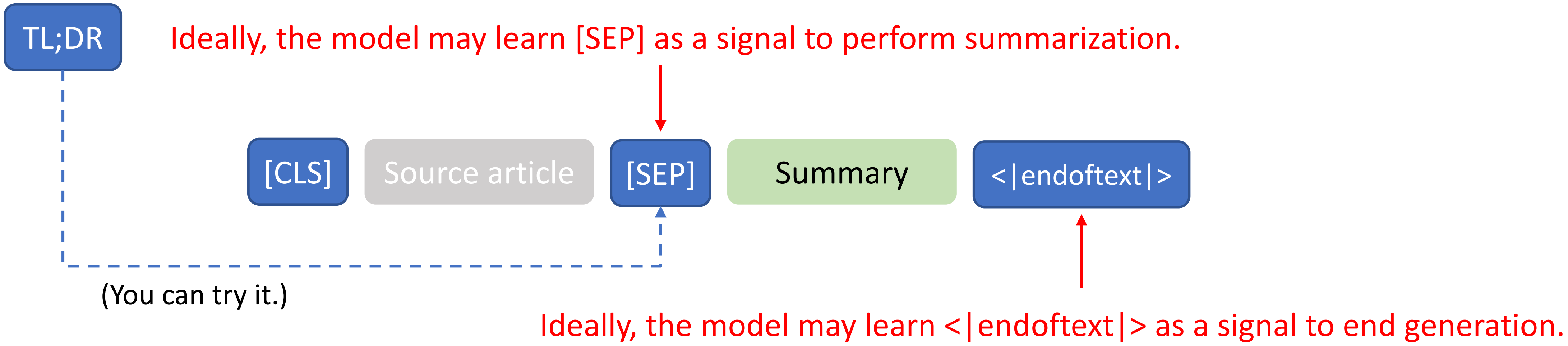
```
"task_specific_params": {
  "text-generation": {
    "do_sample": true,
    "max_length": 320
  }
},
"tokenizer_class": "BertTokenizer",
"vocab_size": 21128
```

https://huggingface.co/ckiplab/gpt2-base-chinese/blob/main/config.json#L32

```
"task_specific_params": {
  "text-generation": {
    "do_sample": true,
    "max_length": 50
  }
},
"tokenizer_class": "BertTokenizerFast",
"vocab_size": 21128
```

English GPT-2 for Text Summarization[2]

> To induce summarization behavior, we can add the text TL;DR: after the article [2]



[2] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. OpenAI blog, 1(8), 9.

Create the PyTorch DataLoader for Data Batching: 2. Labels (auto-regressive)

(We are still at the `collate_fn` function.)

```python
# Set label padding tokens to -100 for loss masking
labels = torch.where(
    condition=complete_text.input_ids != tokenizer.pad_token_id,
    input=complete_text.input_ids,
    other=-100,
)
complete_text["labels"] = labels
complete_text = {k: complete_text[k].to(device) for k in complete_text}
```

this is sentence a		[PAD]	[PAD]	this	is	sentence	a
	labels	-100	-100	1212	318	6827	317
this is is is sentence b		this	is	is	is	sentence	b
	labels	1212	318	318	318	6827	347

GPT2LMHeadModel shifts logits and labels

https://github.com/huggingface/transformers/blob/main/src/transformers/models/gpt2/modeling_gpt2.py#L1300-L1301

Shape of lm_logits: [batcg_size, seq_length, hidden_size]

```python
1296            if labels is not None:
1297                # move labels to correct device to enable model parallelism
1298                labels = labels.to(lm_logits.device)
1299                # Shift so that tokens < n predict n
1300                shift_logits = lm_logits[..., :-1, :].contiguous()
1301                shift_labels = labels[..., 1:].contiguous()
1302                # Flatten the tokens
1303                loss_fct = CrossEntropyLoss()
1304                loss = loss_fct(shift_logits.view(-1, shift_logits.size(-1)), shift_labels.view(-1))
```

:-1 代表取到倒數第二個

1: 代表答案後移一格

.contiguous():
https://youtu.be/FGJSGAt_9Ks?si=GMxxS2Oi742e2BhI&t=1261

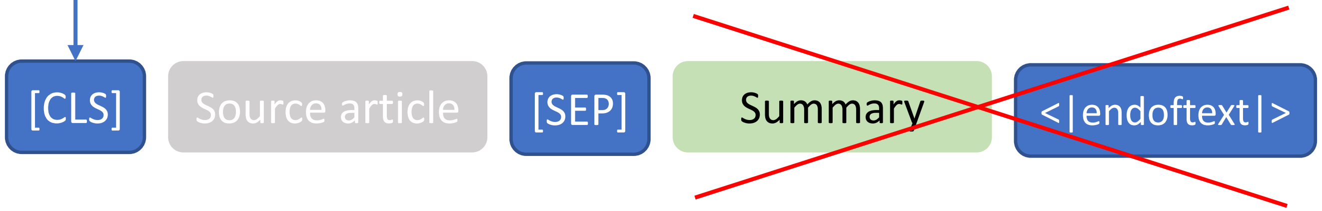
Create the PyTorch DataLoader for Data Batching: 3. Input Data for Predictions

(We are still at the `collate_fn` function.)

We will use `infer_text` during evaluations.

```
    infer_text = [example["text"] for example in batch]
    infer_text = tokenizer.batch_encode_plus(
        infer_text,
        padding=True,
        truncation=True,
        return_tensors="pt",
    )
    infer_text = {k: infer_text[k].to(device) for k in infer_text}
    return complete_text, infer_text
```

If you omit the [CLS] token during fine-tuning, then you don't need to add [CLS] before evaluations!



Create the PyTorch DataLoader for Data Batching: 1. Input Data for Training [Recap]

```python
def collate_fn(batch):
    complete_text = [
        f"[CLS]{example['text']}[SEP]{example['summary']}<|endoftext|>"
        for example in batch
    ]
    complete_text = tokenizer.batch_encode_plus(
        complete_text,
        padding=True,
        truncation=True,
        return_tensors="pt",
        add_special_tokens=False,
    )
```

You can omit the [CLS] token before fine-tuning!

[CLS] | Source article | [SEP] | Summary | <|endoftext|>

Create the PyTorch DataLoader for Data Batching

```
train_loader = torch.utils.data.DataLoader(
    train_set,
    batch_size=TRAIN_BATCH_SIZE,
    shuffle=True,
    collate_fn=collate_fn,
)
val_loader = torch.utils.data.DataLoader(
    val_set,
    batch_size=VAL_BATCH_SIZE,
    shuffle=False,
    collate_fn=collate_fn,
)
```

Prevent a model from overfitting on data order

We don’t need to use the test set because there is no public test labels (summaries).

See https://huggingface.co/datasets/hugcyp/LCSTS/viewer/default/test

Set up Optimizer

```
optimizer = torch.optim.AdamW(model.parameters(), lr=LR)
```

- **optimizers** (`Tuple[torch.optim.Optimizer, torch.optim.lr_scheduler.LambdaLR]`, *optional*, defaults to `(None, None)`) — A tuple containing the optimizer and the scheduler to use. Will default to an instance of AdamW on your model and a scheduler given by get_linear_schedule_with_warmup() controlled by `args`.

Trainer

Trainer

Seq2SeqTrainer

TrainingArguments

Seq2SeqTrainingArguments

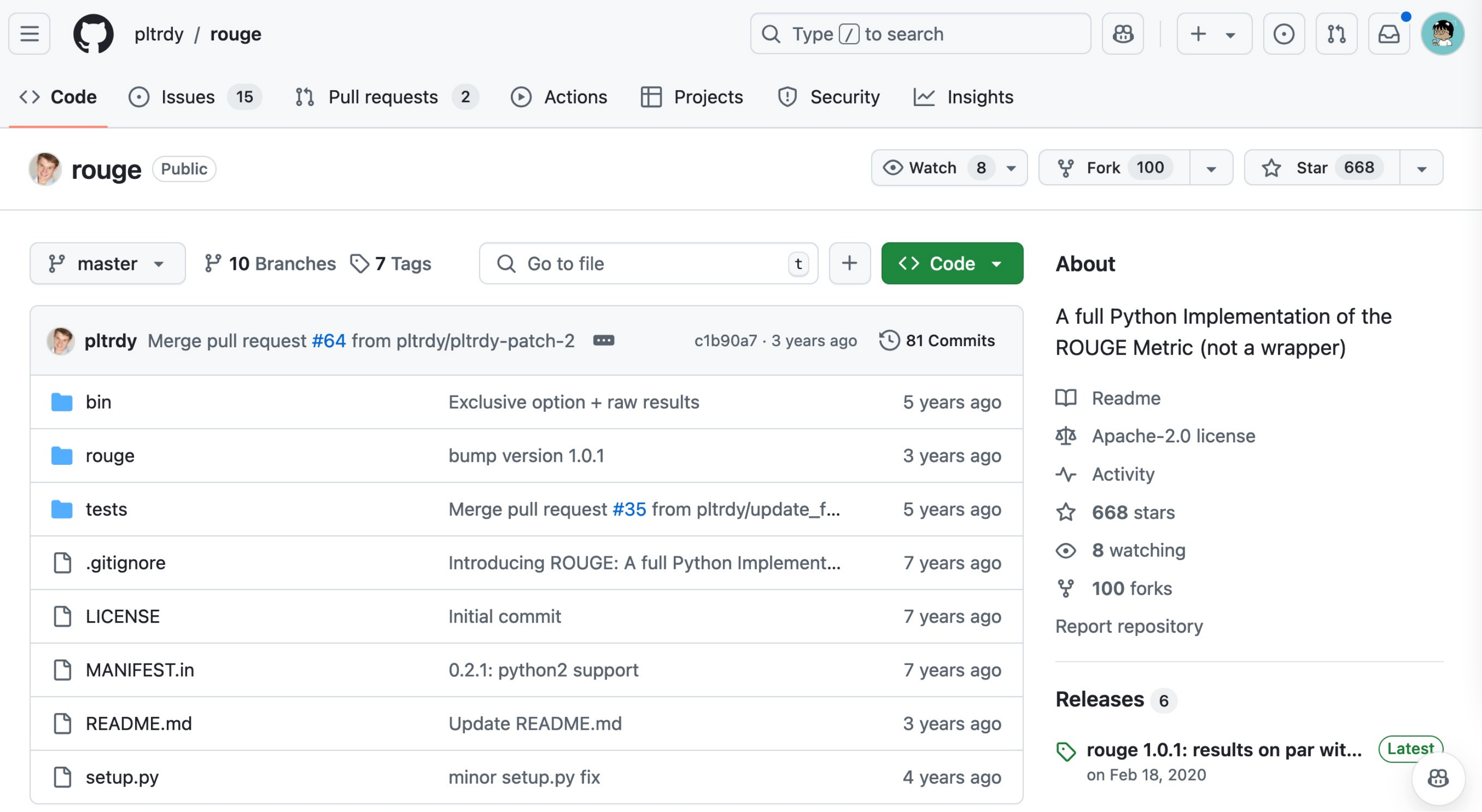
https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html
https://huggingface.co/docs/transformers/main_classes/trainer

Let’s try it

1. 開 train 一下模型

Set up Evaluation Metric

```
rouge_metric = Rouge()
```



https://github.com/pltrdy/rouge

ROUGE Score[3]

- ROUGE (Recall-Oriented Understudy for Gisting Evaluation)
- Mainly for text summarization
- Metric Input: Summary (prediction), Reference (gold summary)
- Common metrics: ROUGE-1, ROUGE-2, ROUGE-L
  - L: Longest common subsequence
- Please note that current papers calculate ROUGE-F as default!!!
  - In other words, ROUGE-1F, ROUGE-2F, ROUGE-LF

Lin, Chin-Yew. "Rouge: A package for automatic evaluation of summaries." Text summarization branches out. 2004.

ROUGE-1 Example

predictions = ["The", "cat", "sat", "on", "the", "mat"]

references = ["A", "cat", "was", "sitting", "on", "the", "mat"]

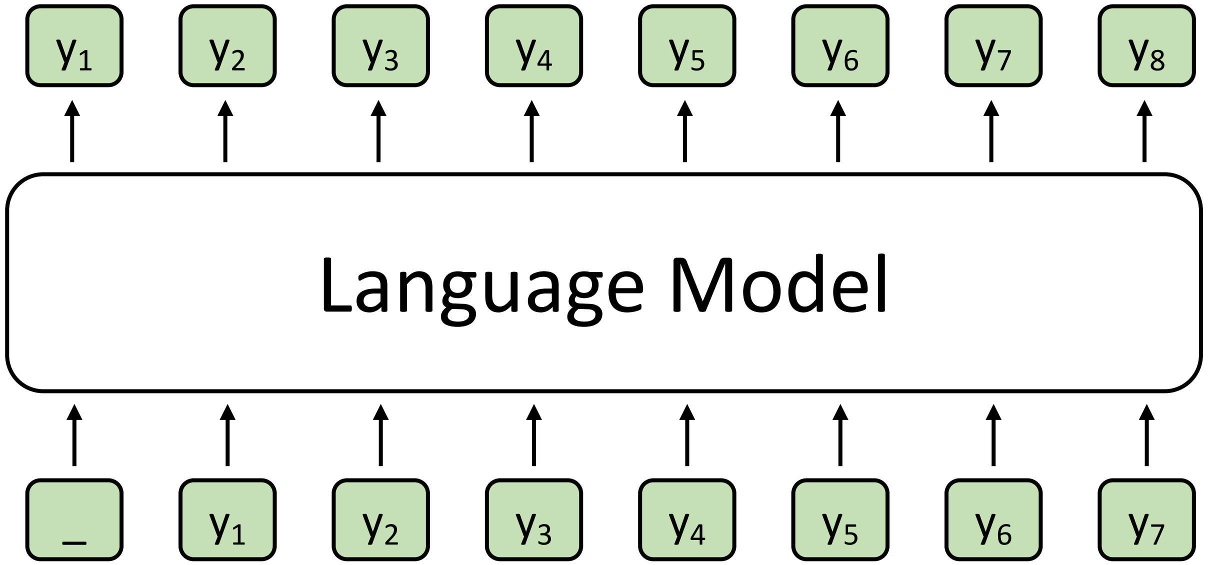
ROUGE-1 recall = Number of matching unigrams / Number of unigrams in the reference = 4/7

ROUGE-1 precision = Number of matching unigrams / Number of unigrams in the machine-generated summary = 4/6

ROUGE-1 F1-score = Harmonic mean of the precision and the recall = 2 * 4/7 * 4/6 / (4/7 + 4/6)

Generating step-by-step

Casual Language Model
(E.g., GPT)



- Training time: update the model through hidden states
- Test time: perform decoding for each step

Decoding strategies

- Popular decoding strategies:
  - Greedy decoding
  - Beam search
  - Top-k sampling
  - Top-p sampling

https://huggingface.co/blog/how-to-generate

How to let a model generate step-by-step?

- We can use [model.generate()]
- There are two main advantages of model.generate():
  1. We don't need to write a decoding loop on our own.
  2. We don't need to implement decoding strategies by ourselves.

[Useful!!] https://huggingface.co/blog/how-to-generate

Pseudo code of a decoding loop

```
for step in range(max_length):
    # Forward pass through the model
    outputs = model.decoder(
        input_ids=decoder_input_ids,
        encoder_outputs=encoder_outputs
    )

    # Take the last hidden state's logits
    logits = outputs.logits[-1]  # shape: (vocab_size,)

    # Apply softmax to get probabilities (optional, since argmax is invariant)
    probs = softmax(logits)

    # Select the token with the highest probability (greedy decoding)
    next_token_id = argmax(probs)

    # Append the predicted token to the decoder inputs
    decoder_input_ids.append(next_token_id)

    # If EOS token is generated, stop decoding
    if next_token_id == EOS_TOKEN_ID:
        break
```

Perform Evaluations

Input: [CLS] Source article [SEP]

Expected output of `model.generate()`:

[CLS] Source article [SEP] Summary <|endoftext|>

Gold: Summary

Perform Evaluations

Output Part

```python
def do_evaluate(
    tokenizer,
    model,
    validation_loader,
    rouge_metric,
    inner_check=False,
):
    pbar = tqdm(validation_loader)
    pbar.set_description(f"Evaluating")

    predictions = []
    references = []
    count = 0
    for ground_truth, inputs in pbar:
        output = [
            s.split("[SEP]")[1].replace(" ", "").split("<|endoftext|>")[0]
            for s in tokenizer.batch_decode(
                model.generate(
                    **inputs,
                    max_new_tokens=200,
                    pad_token_id=tokenizer.eos_token_id,
                )
            )
        ]
```

Expected output of `model.generate()`:

[CLS] | Source article | [SEP] | Summary | <|endoftext|>

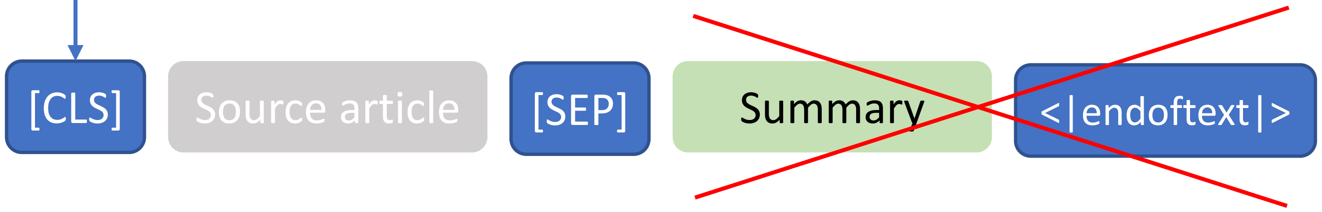
Create the PyTorch DataLoader for Data Batching: 3. Input Data for Predictions

(We are still at the `collate_fn` function.)

We will use `infer_text` during evaluations.

```python
    infer_text = [example["text"] for example in batch]
    infer_text = tokenizer.batch_encode_plus(
        infer_text,
        padding=True,
        truncation=True,
        return_tensors="pt",
    )
    infer_text = {k: infer_text[k].to(device) for k in infer_text}
    return complete_text, infer_text
```

If you omit the [CLS] token during fine-tuning, then you don't need to add [CLS] before evaluations!



Perform Evaluations

Output Part

```
def do_evaluate(
    tokenizer,
    model,
    validation_loader,
    rouge_metric,
    inner_check=False,
):
    pbar = tqdm(validation_loader)
    pbar.set_description(f"Evaluating")

    predictions = []
    references = []
    count = 0
    for ground_truth, inputs in pbar:
        output = [
            s.split("[SEP]")[1].replace(" ", "").split("<|endoftext|>")[0]
            for s in tokenizer.batch_decode(
                model.generate(
                    **inputs,
                    max_new_tokens=200,
                    pad_token_id=tokenizer.eos_token_id,
                )
            )
        ]
```

If you don’t set max_new_tokens, Hugging Face will also count the input tokens!

Perform Evaluations

Output Part

```
def do_evaluate(
    tokenizer,
    model,
    validation_loader,
    rouge_metric,
    inner_check=False,
):
    pbar = tqdm(validation_loader)
    pbar.set_description(f"Evaluating")

    predictions = []
    references = []
    count = 0
    for ground_truth, inputs in pbar:
        output = [
            s.split("[SEP]")[1].replace(" ", "").split("<|endoftext|>")[0]
            for s in tokenizer.batch_decode(
                model.generate(
                    **inputs,
                    max_new_tokens=200,
                    pad_token_id=tokenizer.eos_token_id,
                )
            )
        ]
```

For each batch, first finished sentences should have <|endoftext|> rather than [PAD] at the end.

More details: https://github.com/huggingface/transformers/blob/b880508440f43f80e35a78ccd2a32f3bde91cb23/src/transformers/generation_utils.py#L1248-L1251

Perform Evaluations

Target and Scoring Parts

```python
        targets = [
            s.split("[SEP]")[1].replace(" ", "").replace("<|endoftext|>", "")
            for s in tokenizer.batch_decode(ground_truth["input_ids"])
        ]
        assert len(output) == len(targets)
        output = [" "] if output == [""] else output
        predictions.extend([" ".join(jieba.lcut(o)) for o in output])
        references.extend([" ".join(jieba.lcut(t)) for t in targets])
        count += 1
        if count > 100 and inner_check:
            break

    score = rouge_metric.get_scores(predictions, references, avg=True)
    if inner_check:
        print("Validation using 100 examples: ", score)
    else:
        print(score)

    return score, predictions, references
```

ground_truth:

[CLS] | Source article | [SEP] | Summary | <|endoftext|>

Perform Evaluations

Target and Scoring Parts

```
        targets = [
            s.split("[SEP]")[1].replace(" ", "").replace("<|endoftext|>", "")
            for s in tokenizer.batch_decode(ground_truth["input_ids"])
        ]
        assert len(output) == len(targets)
        output = [" "] if output == [""] else output
        predictions.extend([" ".join(jieba.lcut(o)) for o in output])
        references.extend([" ".join(jieba.lcut(t)) for t in targets])
        count += 1
        if count > 100 and inner_check:
            break

    score = rouge_metric.get_scores(predictions, references, avg=True)
    if inner_check:
        print("Validation using 100 examples: ", score)
    else:
        print(score)

    return score, predictions, references
```

Character-level bi-gram: (「看」、「電」)(「電」、「視」)

Word-level bi-gram: (「看」、「電視」)

Perform Evaluations

Target and Scoring Parts

```python
        targets = [
            s.split("[SEP]")[1].replace(" ", "").replace("<|endoftext|>", "")
            for s in tokenizer.batch_decode(ground_truth["input_ids"])
        ]
        assert len(output) == len(targets)
        output = [" "] if output == [""] else output
        predictions.extend([" ".join(jieba.lcut(o)) for o in output])
        references.extend([" ".join(jieba.lcut(t)) for t in targets])
        count += 1
        if count > 100 and inner_check:
            break

    score = rouge_metric.get_scores(predictions, references, avg=True)
    if inner_check:
        print("Validation using 100 examples: ", score)
    else:
        print(score)

    return score, predictions, references
```

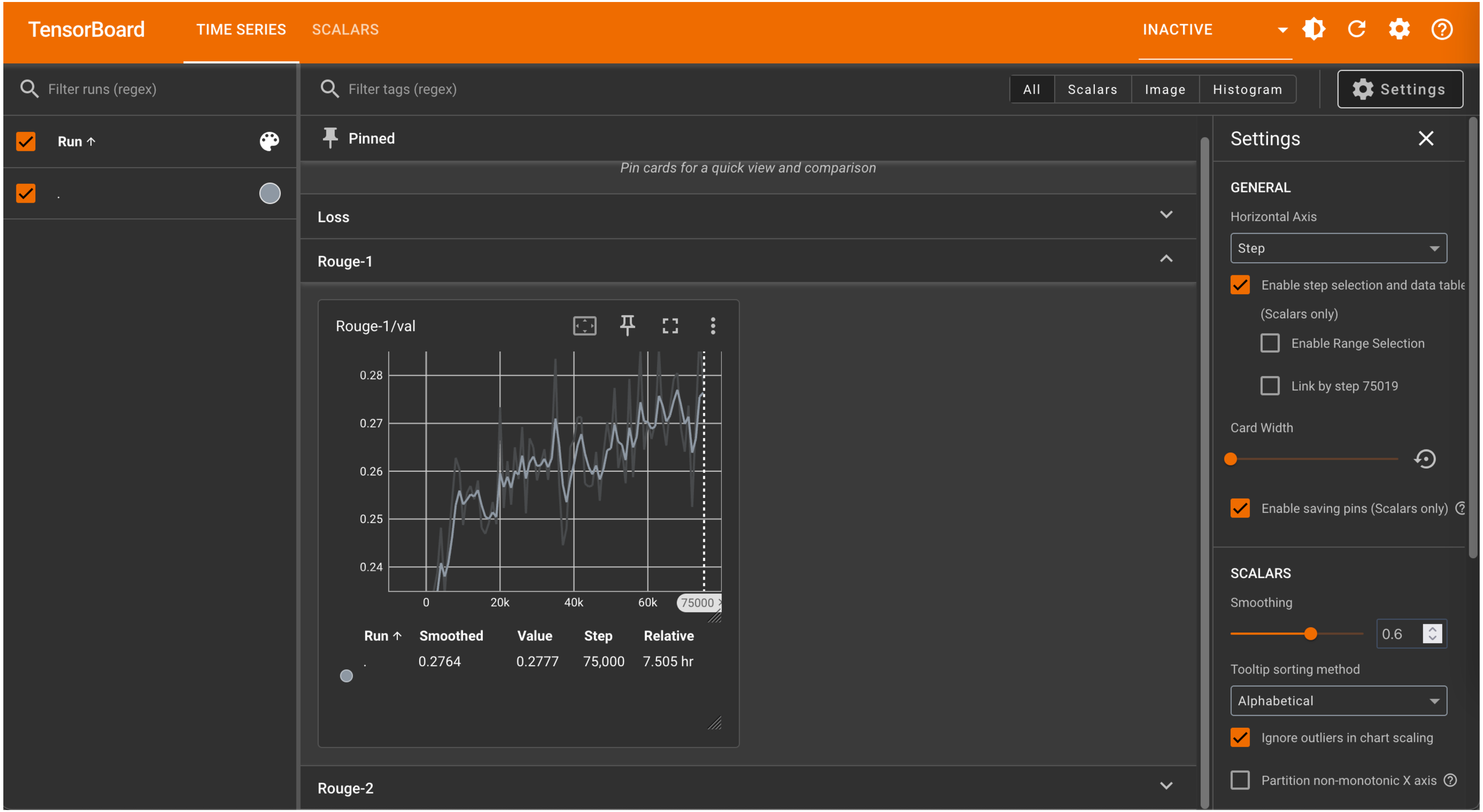
avg=True: perform averaging for all the tested instances

Training Loop

Evaluating the subset of the validation data at each 1,000 steps

```python
step_i = 0
for epoch in range(NUM_EPOCHS):
    pbar = tqdm(train_loader)
    pbar.set_description(f"Training epoch [{epoch+1}/{NUM_EPOCHS}]")
    for inputs, _ in pbar:
        optimizer.zero_grad()
        loss = model(**inputs).loss
        loss.backward()
        optimizer.step()
        pbar.set_postfix(loss=loss.item())
        writer.add_scalar("Loss/train", loss.item(), step_i)

        if step_i % 1000 == 0 and step_i != 0:
            score, pres, refs = do_evaluate(
                tokenizer=tokenizer,
                model=model,
                validation_loader=val_loader,
                rouge_metric=rouge_metric,
                inner_check=True,
            )
            print(f"Rouge scores on step{step_i} of epoch {epoch}:", score)
            print("Predictions:", pres[:5])
            writer.add_scalar("Rouge-1/val", score["rouge-1"]["f"], step_i)
            writer.add_scalar("Rouge-2/val", score["rouge-2"]["f"], step_i)
        step_i += 1
```


TensorBoard
TIME SERIES
SCALARS
INACTIVE
Filter runs (regex)
Filter tags (regex)
All
Scalars
Image
Histogram
Settings
Run ↑
Pinned
Pin cards for a quick view and comparison
Loss
Rouge-1
Rouge-1/val
0.28
0.27
0.26
0.25
0.24
0
20k
40k
60k
75000
Run ↑ Smoothed Value Step Relative
0.2764 0.2777 75,000 7.505 hr
Rouge-2
Settings
GENERAL
Horizontal Axis
Step
Enable step selection and data table
(Scalars only)
Enable Range Selection
Link by step 75019
Card Width
Enable saving pins (Scalars only)
SCALARS
Smoothing
0.6
Tooltip sorting method
Alphabetical
Ignore outliers in chart scaling
Partition non-monotonic X axis

Let’s try it

1. 看一下你自己的 tensorboard

Training Loop

Evaluating the full validation set at each end of epoch

```python
    score, pres, refs = do_evaluate(
        tokenizer=tokenizer,
        model=model,
        validation_loader=val_loader,
        rouge_metric=rouge_metric,
    )
    torch.save(model, f"{SAVED_DIR}/ep{epoch}.ckpt")
```

Outline

- GPT-2 (native PyTorch)
- T5 (Hugging Face Dataset and Trainer)

T5 [4]

[4] Raffel, Colin, et al. "Exploring the limits of transfer learning with a unified text-to-text transformer." *Journal of machine learning research* 21.140 (2020): 1-67.

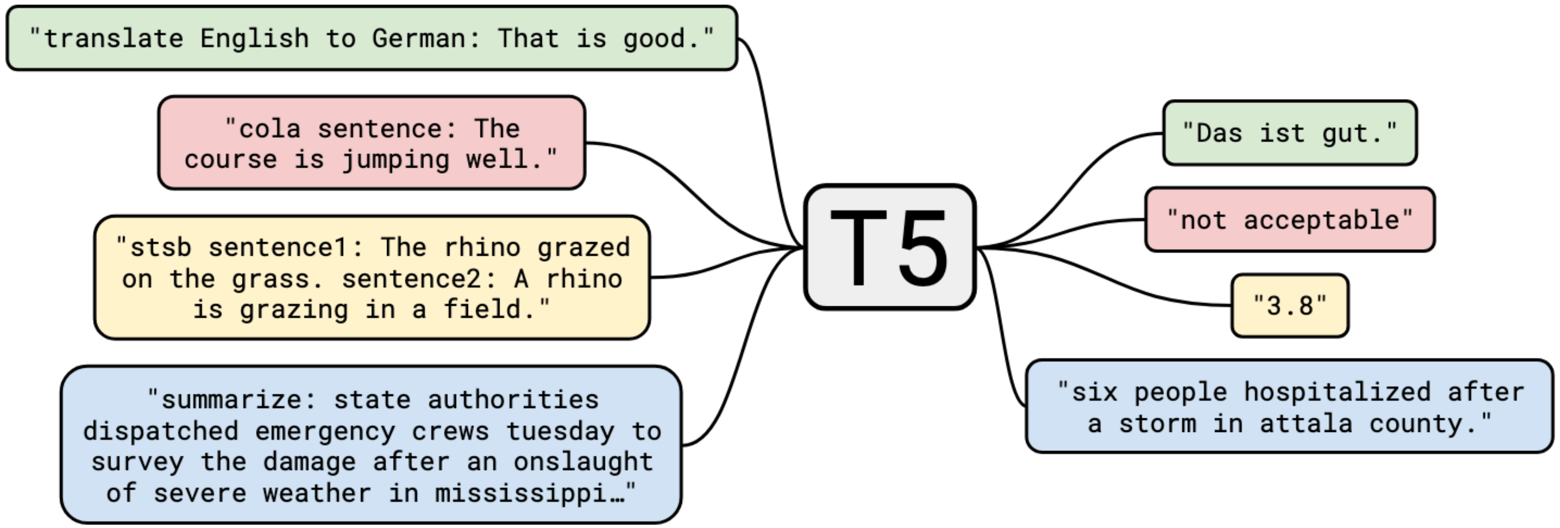
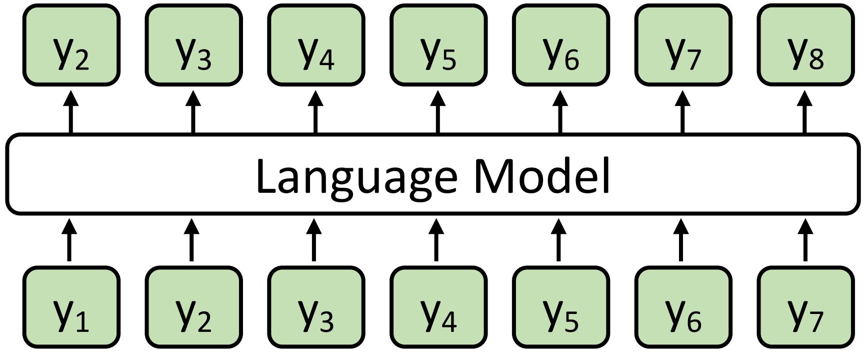


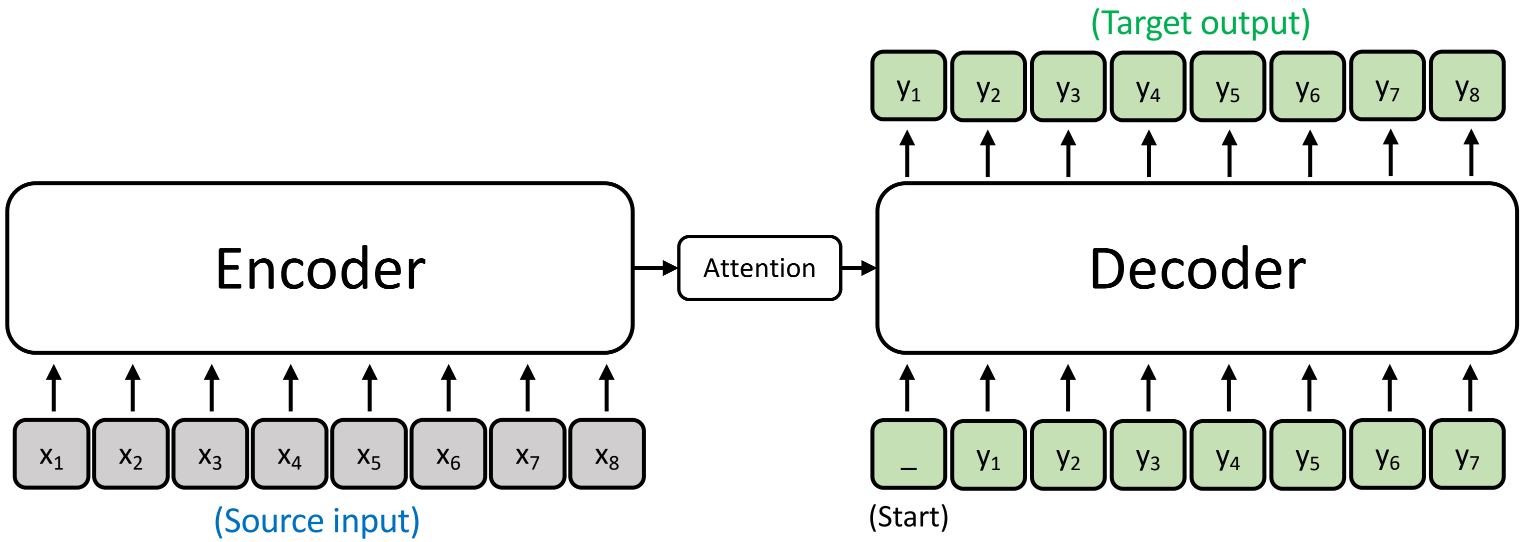
Figure 1: A diagram of our text-to-text framework. Every task we consider—including translation, question answering, and classification—is cast as feeding our model text as input and training it to generate some target text. This allows us to use the same model, loss function, hyperparameters, etc. across our diverse set of tasks. It also provides a standard testbed for the methods included in our empirical survey. "T5" refers to our model, which we dub the "**T**ext-**t**o-**T**ext **T**ransfer **T**ransformer".

Language Model vs. Seq2seq Model

Casual Language Model
(E.g., GPT)



Sequence to sequence Model
(E.g., T5; Vanilla Transformers)



Import packages

```python
from transformers import AutoTokenizer
from transformers import AutoModelForSeq2SeqLM
from transformers import DataCollatorForSeq2Seq
from transformers import Seq2SeqTrainingArguments
from transformers import Seq2SeqTrainer
from datasets import load_dataset
from rouge import Rouge
import numpy as np
import pickle
import os
import jieba
```

Load the LCSTS dataset

- [Hugging Face dataset link]

```
data_name = "hugcyp/LCSTS"
raw_train = load_dataset(data_name, split="train")
raw_valid = load_dataset(data_name, split="validation")
raw_small_valid = raw_valid.select(range(100))
```

- raw_small_valid will be used for inner_check (validation).
- We will later process the dataset with the built-in map function, so we don't use `.to_list()` here.

Load the Tokenizer and the Model

- [Hugging Face model link]()

```
model_name = "google/mt5-small"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
```

Q1: Do we need left-padding?
A1: No, because we are now using a **sequence-to-sequence model**, which will first compress input sequences though the encoder.

Q2: Do we need to add [EOS] by ourselves?
A2: No, **mT5** has </s> as the [EOS] token.

Data pre-processing (inputs; source articles)

```python
token_replacement = [
    ["：", ":"],
    ["，", ","],
    ["“", '"'],
    ["”", '"'],
    ["？", "?"],
    ["…", "..."],
    ["！", "!"],
]
```

```python
def replace_tokens(examples):
    for k in ["text", "summary"]:
        for i, _ in enumerate(examples[k]):
            for tok in token_replacement:
                examples[k][i] = examples[k][i].replace(tok[0], tok[1])
    return examples


def preprocess_function(examples):
    examples = replace_tokens(examples)
    model_inputs = tokenizer(examples["text"], padding=True, truncation=True)
    labels = tokenizer(
        text_target=examples["summary"],
        max_length=200,
        truncation=True,
    )
    model_inputs["labels"] = labels["input_ids"]

    return model_inputs
```

Data pre-processing (labels; summaries)

```python
token_replacement = [
    ["：", ":"],
    ["，", ","],
    ["“", '"'],
    ["”", '"'],
    ["？", "?"],
    ["……", "..."],
    ["！", "!"],
]
```

```python
def replace_tokens(examples):
    for k in ["text", "summary"]:
        for i, _ in enumerate(examples[k]):
            for tok in token_replacement:
                examples[k][i] = examples[k][i].replace(tok[0], tok[1])
    return examples


def preprocess_function(examples):
    examples = replace_tokens(examples)
    model_inputs = tokenizer(examples["text"], padding=True, truncation=True)
    labels = tokenizer(
        text_target=examples["summary"],
        max_length=200,
        truncation=True,
    )
    model_inputs["labels"] = labels["input_ids"]

    return model_inputs
```

Data pre-processing (executions)

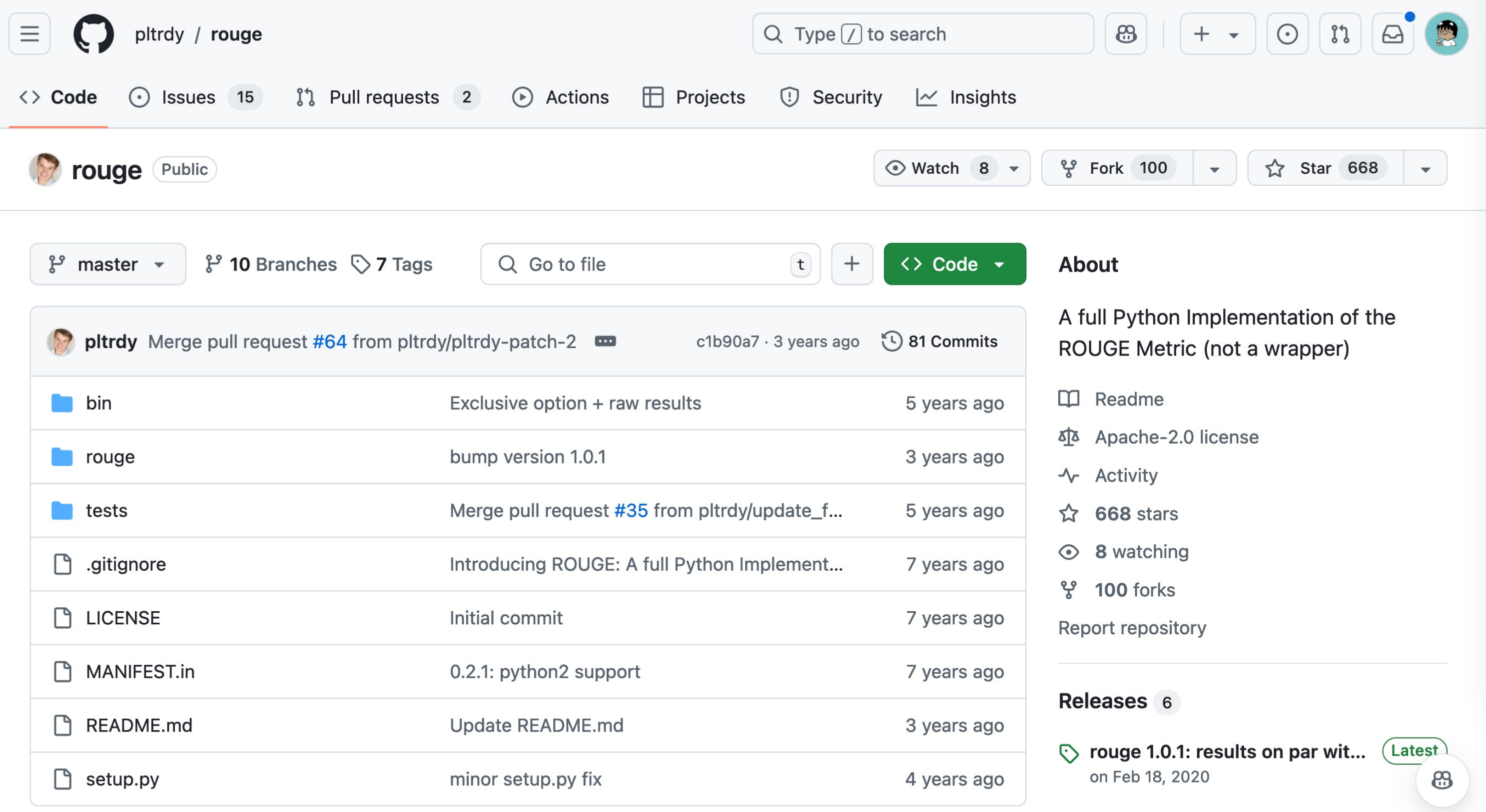
```
train = raw_train.map(preprocess_function, batched=True)
valid = raw_valid.map(preprocess_function, batched=True)
small_valid = raw_small_valid.map(preprocess_function, batched=True)
```

```
data_collator = DataCollatorForSeq2Seq(
    tokenizer=tokenizer,
    model=model_name,
)
```

DataCollatorForSeq2Seq dynamically pads batched data and transforms padded labels into -100.
The operation provided by this object does a similar job like collate_fn.

Set up Evaluation Metric

```
rouge_metric = Rouge()
```



https://github.com/pltrdy/rouge

Build `compute_metrics` for evaluations

```python
def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    decoded_preds = tokenizer.batch_decode(predictions, skip_special_tokens=True)
    labels = np.where(labels != -100, labels, tokenizer.pad_token_id)
    decoded_labels = tokenizer.batch_decode(labels, skip_special_tokens=True)

    predictions = [" ".join(jieba.lcut(o)) for o in decoded_preds]
    references = [" ".join(jieba.lcut(t)) for t in decoded_labels]
    result = rouge_metric.get_scores(predictions, references, avg=True)
    score = {f"{rouge_i}_f": v["f"] for rouge_i, v in result.items()}
    prediction_lens = [
        np.count_nonzero(pred != tokenizer.pad_token_id) for pred in predictions
    ]
    score["gen_len"] = np.mean(prediction_lens)
    return {k: v for k, v in score.items()}
```

-100 is transformed by DataCollatorForSeq2Seq.
Convert labels with -100 to pad_token_id for decoding.

Setting up TrainingArguments (link)

```
training_args = Seq2SeqTrainingArguments(
    output_dir="./results/mt5",
    evaluation_strategy="steps",
    save_strategy="steps",
    eval_steps=1000,
    save_steps=10000,
    learning_rate=2e-5,
    per_device_train_batch_size=32,
    per_device_eval_batch_size=32,
    weight_decay=0.01,
    save_total_limit=3,
    num_train_epochs=3,
    predict_with_generate=True,
    logging_dir=f"./logs/{model_prefix}",
    logging_steps=1,
    push_to_hub=False,
)
```

Default using Greedy search. Seq2seqTrainer Supports beam search only.

Setting up Trainer (training part)

```
trainer = Seq2SeqTrainer(
    model=model,
    args=training_args,
    train_dataset=train,
    eval_dataset=small_valid,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)
trainer.args._n_gpu = 1
trainer.train()
results = trainer.predict(valid)
for metric in ["1", "2", "l"]:
    rouge_item = f"test_rouge-{metric}"
    print(f"{rouge_item}: ", results.metrics[rouge_item])
```

Setting up Trainer (evaluation part)

```python
trainer = Seq2SeqTrainer(
    model=model,
    args=training_args,
    train_dataset=train,
    eval_dataset=small_valid,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)
trainer.args._n_gpu = 1
trainer.train()
results = trainer.predict(valid)
for metric in ["1", "2", "l"]:
    rouge_item = f"test_rouge-{metric}"
    print(f"{rouge_item}: ", results.metrics[rouge_item])
```

Thank you!

Instructor: 林英嘉

✉️ yjlin@cgu.edu.tw

TA: 吳宣毅

✉️ m1161007@cgu.edu.tw