



# Natural Language Processing

GPT-2 and T5 Tutorial 2024/11/05



# What you will learn in this tutorial

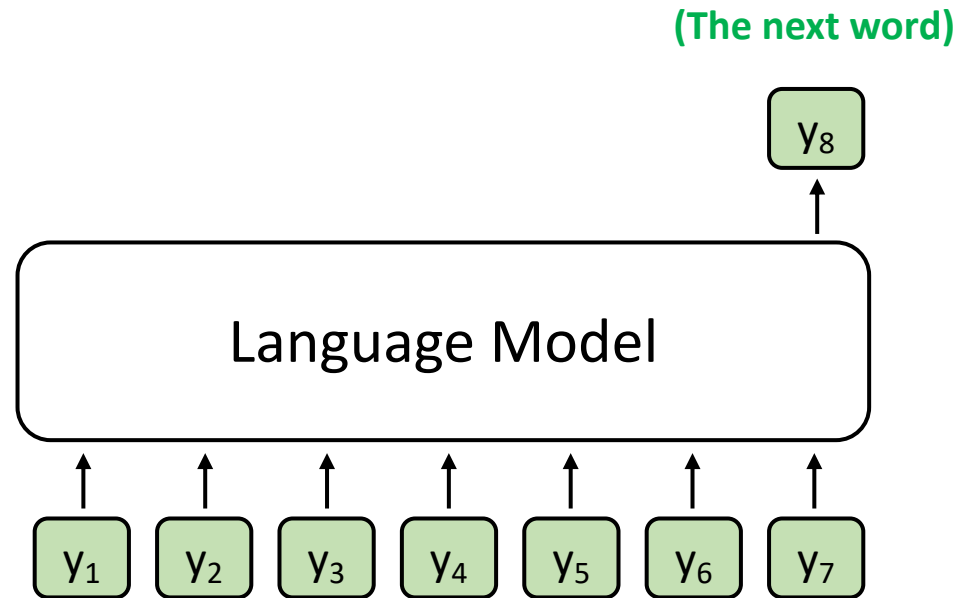
---

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



# Language Model

---



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

- A model that assigns probabilities to upcoming words is called a **language model**.
- The task involving predictions of upcoming words is **language modeling**.

# 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)



# 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)<sup>[1]</sup>

**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

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.



## Documentations

<https://huggingface.co/docs>

### • Hub

Host Git-based models, datasets and Spaces on the Hugging Face Hub.

### • Hub Python Library

Client library for the HF Hub: manage repositories from your Python runtime.

### • Inference API (serverless)

Experiment with over 200k models easily using the serverless tier of Inference Endpoints.

### • Transformers

State-of-the-art ML for Pytorch, TensorFlow, and JAX.

### • Datasets

Access and share datasets for computer vision, audio, and NLP tasks.

### • Huggingface.js

A collection of JS libraries to interact with Hugging Face, with TS types included.

### • Inference Endpoints (dedicated)

Easily deploy models to production on dedicated, fully managed infrastructure.

### • Diffusers

State-of-the-art diffusion models for image and audio generation in PyTorch.

### • Gradio

Build machine learning demos and other web apps, in just a few lines of Python.

### • Transformers.js

Community library to run pretrained models from Transformers in your browser.

### • 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

---

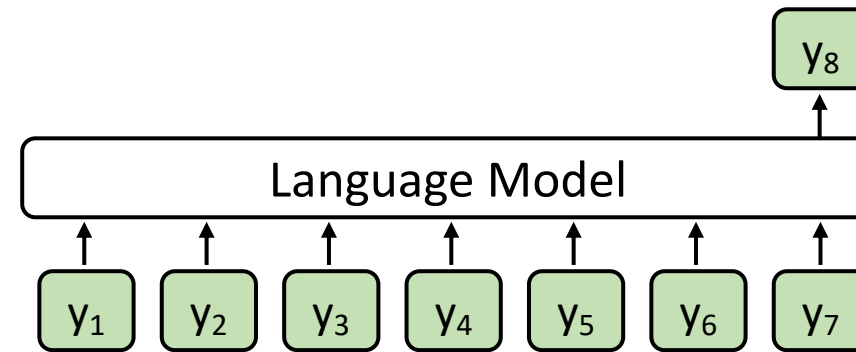
We need the following packages for this tutorial. On Colab, you may not need to re-install these packages for they may have already been installed.

```
! 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
```

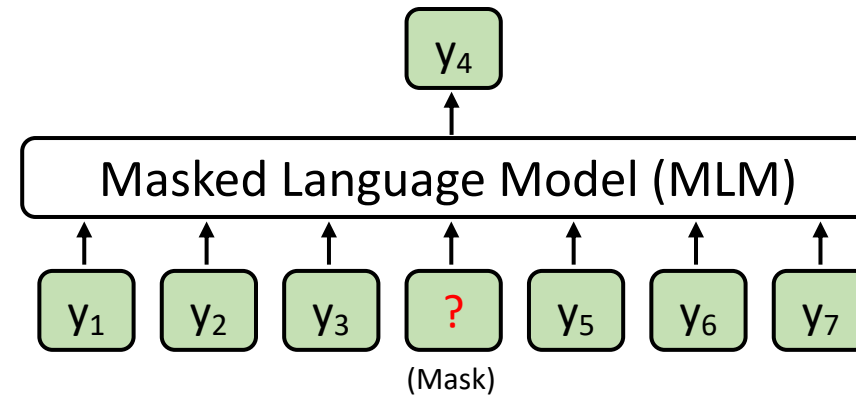


# Language Model vs. Masked Language Model

Casual Language Model  
(E.g., GPT)

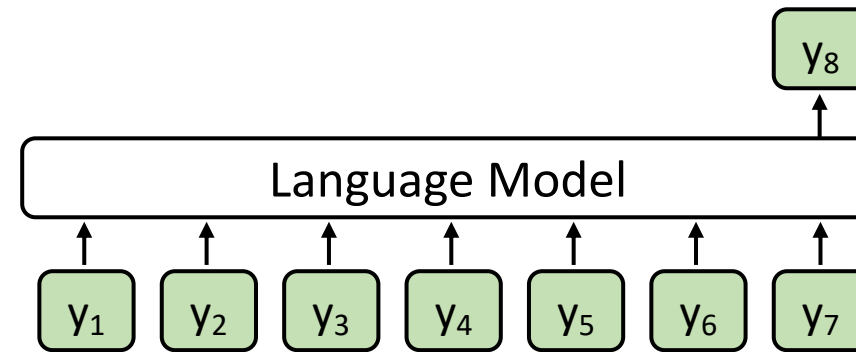


Masked Language Model  
(E.g., BERT)

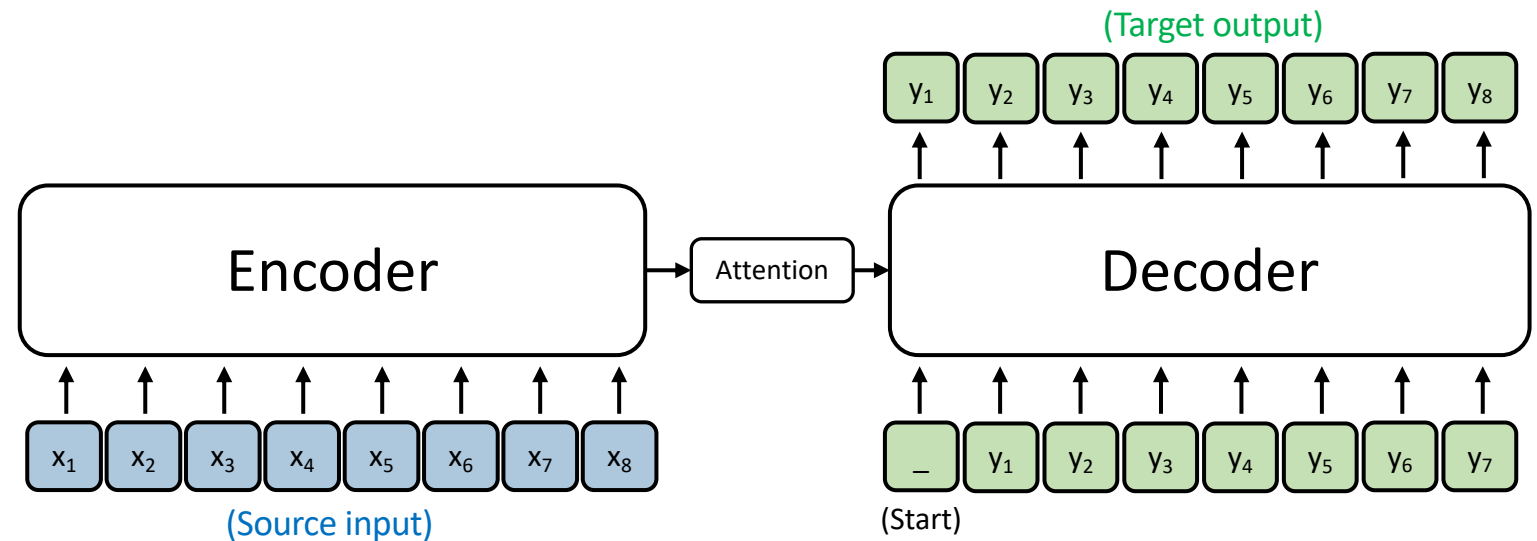


# Language Model vs. Seq2seq Model

Casual Language Model  
(E.g., GPT)



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



# Load the LCSTS 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()`.



# 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 `BertModel`).

[https://huggingface.co/transformers/v3.0.2/model\\_doc/auto.html](https://huggingface.co/transformers/v3.0.2/model_doc/auto.html)



# Generative Models in Auto Classes

---

Casual Language Model  
(E.g., GPT)

AutoModelForCausalLM ([link](#))  
GPT2LMHeadModel ([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

---

```
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

# 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

---

- When fine-tuning GPT-2
  - Left padding + set [PAD] as -100 in labels (this tutorial)
  - Right padding + set [PAD] as -100 in labels
- When using GPT-2 for test (inference time)
  - Use test\_batch\_size as 1 (this tutorial)
  - Left padding



# Set [PAD] as -100 in labels

---

[https://github.com/huggingface/transformers/blob/main/src/transformers/models/gpt2/modeling\\_gpt2.py#L1264-L1267](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](#))



# 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.

```
1 class LCSTSDataset(torch.utils.data.Dataset):
2     def __init__(self, raw_data) -> None:
3         super().__init__()
4         self.data = raw_data
5         self.token_replacement = [
6             [":", ":"],
7             [",", ","],
8             ["'", "'"],
9             ["'", "'"],
10            ["?", "?"],
11            ["...", "..."],
12            ["!", "!"],
13        ]
14
15    def __getitem__(self, index):
16        d = self.data[index]
17        # Substitute some full-width punctuations with half-width ones
18        for k in d:
19            for tok in self.token_replacement:
20                d[k] = d[k].replace(tok[0], tok[1])
21        return d
22
23    def __len__(self):
24        return len(self.data)
```

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

Run:

```
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)

# Create the PyTorch DataLoader for Data Batching:

## 1. Input Data for Training

---

```
1 def collate_fn(batch):
2     complete_text = [
3         f"[CLS]{example['text']}[SEP]{example['summary']}<|endoftext|>"
4         for example in batch
5     ]
6     complete_text = tokenizer.batch_encode_plus(
7         complete_text,
8         padding=True,
9         truncation=True,
10        return_tensors="pt",
11        add_special_tokens=False,
12    )
```

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



# uer/gpt2-chinese-cluecorpussmall uses BertTokenizer

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

```
20     "task_specific_params": {  
21         "text-generation": {  
22             "do_sample": true,  
23             "max_length": 320  
24         }  
25     },  
26     "tokenizer_class": "BertTokenizer",  
27     "vocab_size": 21128
```

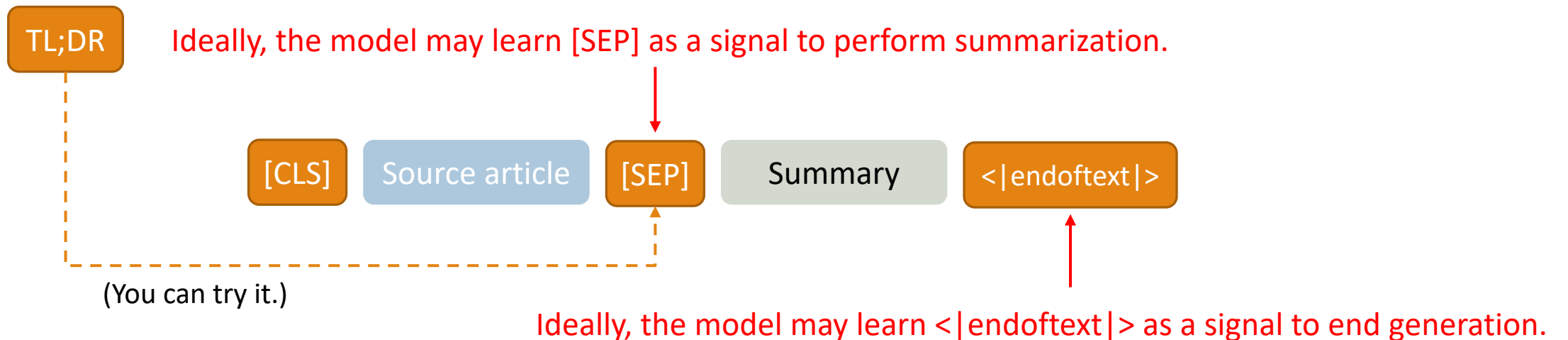
<https://huggingface.co/ckiplab/gpt2-base-chinese/blob/main/config.json#L32>

```
26     "task_specific_params": {  
27         "text-generation": {  
28             "do_sample": true,  
29             "max_length": 50  
30         }  
31     },  
32     "tokenizer_class": "BertTokenizerFast",  
33     "vocab_size": 21128
```



# English GPT-2 for Text Summarization

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



[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.)

```
13 # Set label padding tokens to -100 for loss masking
14 labels = torch.where(
15     condition=complete_text.input_ids != tokenizer.pad_token_id,
16     input=complete_text.input_ids,
17     other=-100,
18 )
19 complete_text["labels"] = labels
20 complete_text = {k: complete_text[k].to(device) for k in complete_text}
```

This is sentence A.

This is is is sentence B.

	[PAD]	[PAD]	this	is	sentence	a
labels	-100	-100	1212	318	6827	317
	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](https://github.com/huggingface/transformers/blob/main/src/transformers/models/gpt2/modeling_gpt2.py#L1300-L1301)

```
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))
```



# 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**.

```
22 infer_text = [example["text"] for example in batch]
23 infer_text = tokenizer.batch_encode_plus(
24     infer_text,
25     padding=True,
26     truncation=True,
27     return_tensors="pt",
28 )
29 infer_text = {k: infer_text[k].to(device) for k in infer_text}
30 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  train_loader = torch.utils.data.DataLoader(  
2      train_set,  
3      batch_size=TRAIN_BATCH_SIZE,  
4      shuffle=True, ← Prevent a model from overfitting on data order  
5      collate_fn=collate_fn,  
6  )  
7  val_loader = torch.utils.data.DataLoader(  
8      val_set,  
9      batch_size=VAL_BATCH_SIZE,  
10     shuffle=False,  
11     collate_fn=collate_fn,  
12 )
```

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](https://huggingface.co/docs/transformers/main_classes/trainer)



# Set up Evaluation Metric

```
rouge_metric = Rouge()
```

The screenshot shows the GitHub repository page for `pltrdy / rouge`. The repository is public and has 8 watches, 100 forks, and 668 stars. The main content area displays a list of files and their commit history:

File	Commit Message	Time Ago
bin	Exclusive option + raw results	5 years ago
rouge	bump version 1.0.1	3 years ago
tests	Merge pull request #35 from pltrdy/update_f...	5 years ago
.gitignore	Introducing ROUGE: A full Python Implement...	7 years ago
LICENSE	Initial commit	7 years ago
MANIFEST.in	0.2.1: python2 support	7 years ago
README.md	Update README.md	3 years ago
setup.py	minor setup.py fix	4 years ago

On the right side, the 'About' section describes the repository as 'A full Python Implementation of the ROUGE Metric (not a wrapper)'. It also lists the README, Apache-2.0 license, and activity. The 'Releases' section shows the latest release, `rouge 1.0.1: results on par wit...`, dated Feb 18, 2020, marked as 'Latest'.

<https://github.com/pltrdy/rouge>



# ROUGE Score

---

- 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-1**F**, ROUGE-2**F**, ROUGE-L**F**

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"]

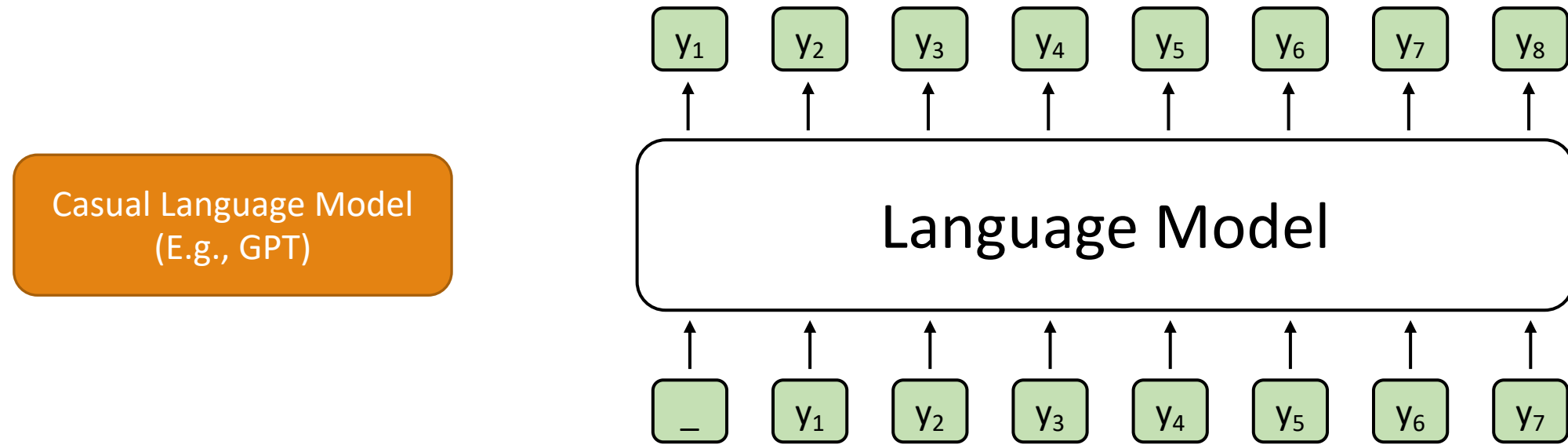
ROUGE-1 **recall** = Number of matching unigrams / Number of unigrams in the reference =  $4/7$

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



- 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.

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



# Perform Evaluations

---

Input:

[CLS]

Source article

[SEP]

Expected output of `model.generate()`:

[CLS]

Source article

[SEP]

Summary

<|endoftext|>

Gold:

Summary

# Perform Evaluations Output Part

```
1  def do_evaluate(  
2      tokenizer,  
3      model,  
4      validation_loader,  
5      rouge_metric,  
6      inner_check=False,  
7  ):  
8      pbar = tqdm(validation_loader)  
9      pbar.set_description(f"Evaluating")  
10  
11      predictions = []  
12      references = []  
13      count = 0  
14      for ground_truth, inputs in pbar:  
15          output = [  
16              s.split("[SEP]")[1].replace(" ", "").split("<|endoftext|>")[0]  
17              for s in tokenizer.batch_decode(  
18                  model.generate(  
19                      **inputs,  
20                      max_new_tokens=200,  
21                      pad_token_id=tokenizer.eos_token_id,  
22                  )  
23              )  
24          ]
```

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**.

```
22 infer_text = [example["text"] for example in batch]
23 infer_text = tokenizer.batch_encode_plus(
24     infer_text,
25     padding=True,
26     truncation=True,
27     return_tensors="pt",
28 )
29 infer_text = {k: infer_text[k].to(device) for k in infer_text}
30 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

```
1  def do_evaluate(  
2      tokenizer,  
3      model,  
4      validation_loader,  
5      rouge_metric,  
6      inner_check=False,  
7  ):  
8      pbar = tqdm(validation_loader)  
9      pbar.set_description(f"Evaluating")  
10  
11      predictions = []  
12      references = []  
13      count = 0  
14      for ground_truth, inputs in pbar:  
15          output = [  
16              s.split("[SEP]")[1].replace(" ", "").split("<|endoftext|>")[0]  
17              for s in tokenizer.batch_decode(  
18                  model.generate(  
19                      **inputs,  
20                      max_new_tokens=200,  
21                      pad_token_id=tokenizer.eos_token_id,  
22                  )  
23              )  
24          ]
```

If you don't set max\_new\_tokens, Hugging Face will also count the input tokens!



# Perform Evaluations Output Part

```
1  def do_evaluate(  
2      tokenizer,  
3      model,  
4      validation_loader,  
5      rouge_metric,  
6      inner_check=False,  
7  ):  
8      pbar = tqdm(validation_loader)  
9      pbar.set_description(f"Evaluating")  
10  
11     predictions = []  
12     references = []  
13     count = 0  
14     for ground_truth, inputs in pbar:  
15         output = [  
16             s.split("[SEP]")[1].replace(" ", "").split("<|endoftext|>")[0]  
17             for s in tokenizer.batch_decode(  
18                 model.generate(  
19                     *inputs,  
20                     max_new_tokens=200,  
21                     pad_token_id=tokenizer.eos_token_id,  
22                 )  
23             )  
24         ]
```

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](https://github.com/huggingface/transformers/blob/b880508440f43f80e35a78ccd2a32f3bde91cb23/src/transformers/generation_utils.py#L1248-L1251)

# Perform Evaluations Target and Scoring Parts

```
25     targets = [  
26         s.split("[SEP]")[1].replace(" ", "").replace("<|endoftext|>", "")  
27         for s in tokenizer.batch_decode(ground_truth["input_ids"])  
28     ]  
29     assert len(output) == len(targets)  
30     output = [" "] if output == [""] else output  
31     predictions.extend([" ".join(jieba.lcut(o)) for o in output])  
32     references.extend([" ".join(jieba.lcut(t)) for t in targets])  
33     count += 1  
34     if count > 100 and inner_check:  
35         break  
36  
37     score = rouge_metric.get_scores(predictions, references, avg=True)  
38     if inner_check:  
39         print("Validation using 100 examples: ", score)  
40     else:  
41         print(score)  
42  
43     return score, predictions, references
```

ground\_truth:

[CLS]

Source article

[SEP]

Summary

<|endoftext|>

# Perform Evaluations Target and Scoring Parts

```
25     targets = [  
26         s.split("[SEP]")[1].replace(" ", "").replace("<|endoftext|>", "")  
27         for s in tokenizer.batch_decode(ground_truth["input_ids"])  
28     ]  
29     assert len(output) == len(targets)  
30     output = [" "] if output == [""] else output  
31     predictions.extend([" ".join(jieba.lcut(o)) for o in output])  
32     references.extend([" ".join(jieba.lcut(t)) for t in targets])  
33     count += 1  
34     if count > 100 and inner_check:  
35         break  
36  
37     score = rouge_metric.get_scores(predictions, references, avg=True)  
38     if inner_check:  
39         print("Validation using 100 examples: ", score)  
40     else:  
41         print(score)  
42  
43     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

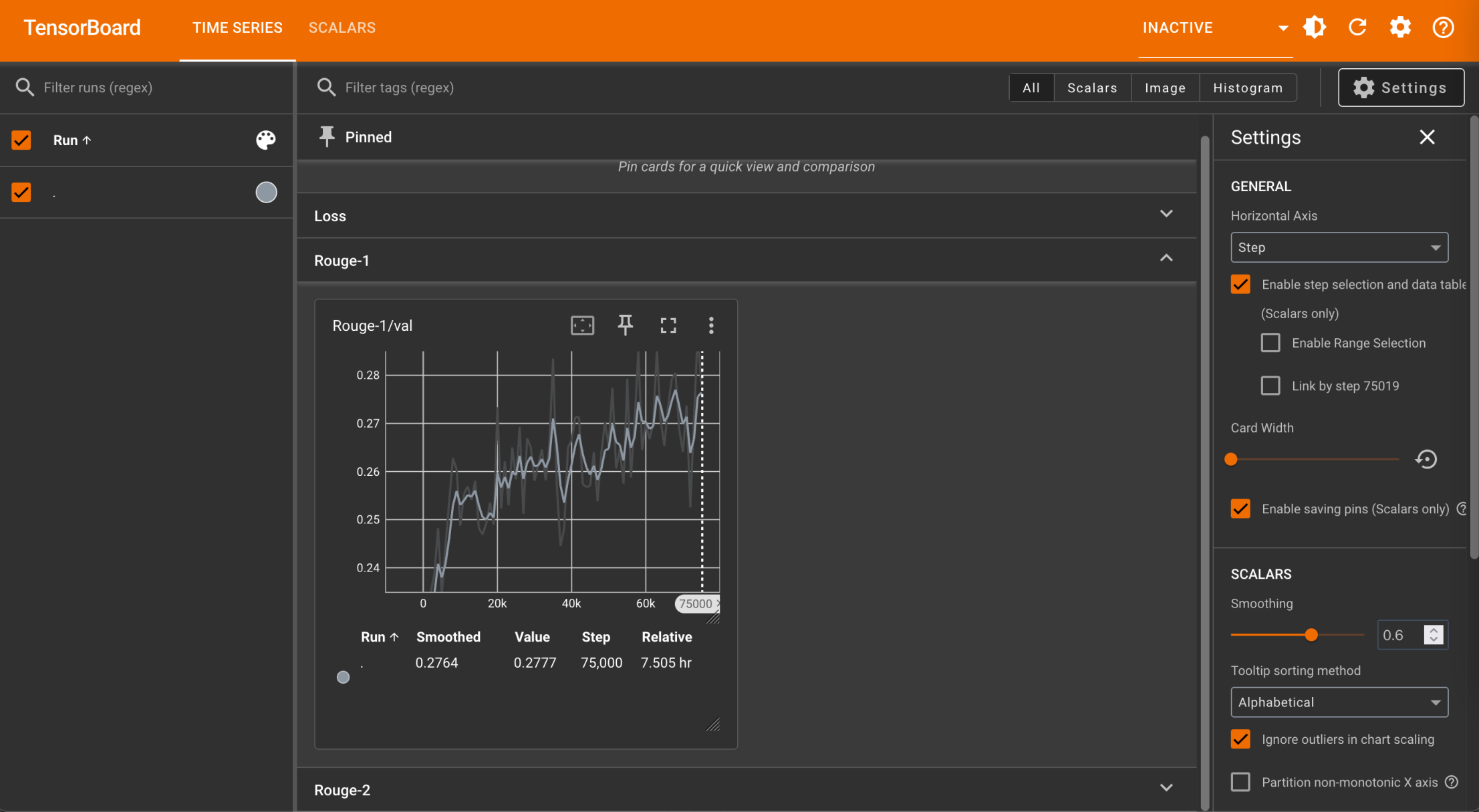
```
25     targets = [  
26         s.split("[SEP]")[1].replace(" ", "").replace("<|endoftext|>", "")  
27         for s in tokenizer.batch_decode(ground_truth["input_ids"])  
28     ]  
29     assert len(output) == len(targets)  
30     output = [" "] if output == [""] else output  
31     predictions.extend([" ".join(jieba.lcut(o)) for o in output])  
32     references.extend([" ".join(jieba.lcut(t)) for t in targets])  
33     count += 1  
34     if count > 100 and inner_check:  
35         break  
36  
37     score = rouge_metric.get_scores(predictions, references, avg=True)  
38     if inner_check:  
39         print("Validation using 100 examples: ", score)  
40     else:  
41         print(score)  
42  
43     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

```
1  step_i = 0
2  for epoch in range(NUM_EPOCHS):
3      pbar = tqdm(train_loader)
4      pbar.set_description(f"Training epoch [{epoch+1}/{NUM_EPOCHS}]")
5      for inputs, _ in pbar:
6          optimizer.zero_grad()
7          loss = model(**inputs).loss
8          loss.backward()
9          optimizer.step()
10         pbar.set_postfix(loss=loss.item())
11         writer.add_scalar("Loss/train", loss.item(), step_i)
12
13         if step_i % 1000 == 0 and step_i != 0:
14             score, pres, refs = do_evaluate(
15                 tokenizer=tokenizer,
16                 model=model,
17                 validation_loader=val_loader,
18                 rouge_metric=rouge_metric,
19                 inner_check=True,
20             )
21             print(f"Rouge scores on step{step_i} of epoch {epoch}:", score)
22             print("Predictions:", pres[:5])
23             writer.add_scalar("Rouge-1/val", score["rouge-1"]["f"], step_i)
24             writer.add_scalar("Rouge-2/val", score["rouge-2"]["f"], step_i)
25         step_i += 1
```



# Training Loop

Evaluating the  
full validation  
set at each end  
of epoch

```
26  ✓ score, pres, refs = do_evaluate(  
27      tokenizer=tokenizer,  
28      model=model,  
29      validation_loader=val_loader,  
30      rouge_metric=rouge_metric,  
31  )  
32  torch.save(model, f"{SAVED_DIR}/ep{epoch}.ckpt")
```

# Outline

---

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

# T5 [3]

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.

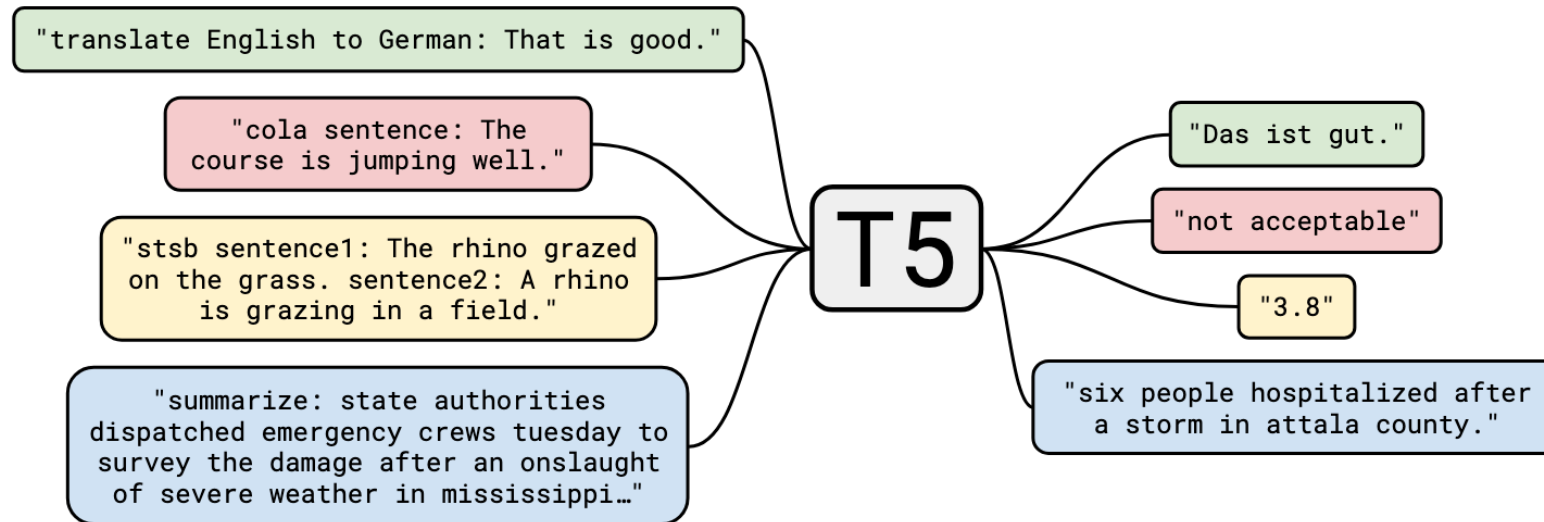
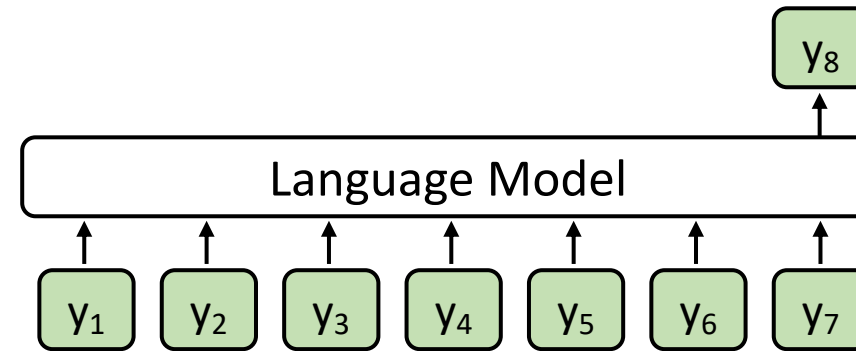


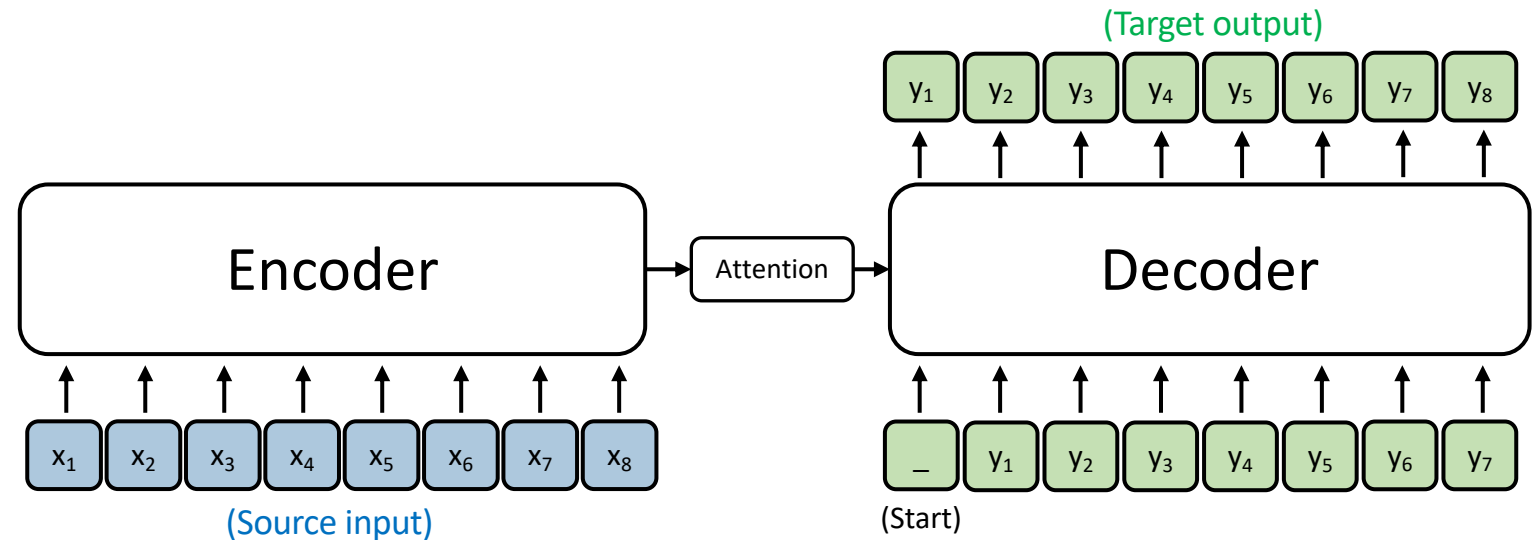
Figure 1: A diagram of our text-to-text framework. Every task we consider—including translation, question answering, and classification—is cast as feeding our model text as input and training it to generate some target text. This allows us to use the same model, loss function, hyperparameters, etc. across 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

---

```
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)
```

Q: Do we need left-padding?

A: No, because we are now using a **sequence-to-sequence model**, which will first compress input sequences through the encoder.

Q: Do we need to add [EOS] by ourselves?

A: No, **mT5** has </s> as the [EOS] token.



# Data pre-processing (inputs; source articles)

```
1 token_replacement = [  
2     [":", ":"],  
3     [",", ","],  
4     ["'", "'"],  
5     ["'", "'"],  
6     ["?", "?"],  
7     [".....", "..."],  
8     ["!", "!"],  
9 ]  
  
12 def replace_tokens(examples):  
13     for k in ["text", "summary"]:  
14         for i, _ in enumerate(examples[k]):  
15             for tok in token_replacement:  
16                 examples[k][i] = examples[k][i].replace(tok[0], tok[1])  
17     return examples  
18  
19  
20 def preprocess_function(examples):  
21     examples = replace_tokens(examples)  
22     model_inputs = tokenizer(examples["text"], padding=True, truncation=True)  
23     labels = tokenizer(  
24         text_target=examples["summary"],  
25         max_length=200,  
26         truncation=True,  
27     )  
28     model_inputs["labels"] = labels["input_ids"]  
29  
30     return model_inputs
```



# Data pre-processing (labels; summaries)

```
1 token_replacement = [  
2     [":", ":"],  
3     [",", ","],  
4     ["'", "'"],  
5     ["'", "'"],  
6     ["?", "?"],  
7     ["...", "..."],  
8     ["!", "!"],  
9 ]  
  
12 def replace_tokens(examples):  
13     for k in ["text", "summary"]:  
14         for i, _ in enumerate(examples[k]):  
15             for tok in token_replacement:  
16                 examples[k][i] = examples[k][i].replace(tok[0], tok[1])  
17     return examples  
  
18  
19  
20 def preprocess_function(examples):  
21     examples = replace_tokens(examples)  
22     model_inputs = tokenizer(examples["text"], padding=True, truncation=True)  
23     labels = tokenizer(  
24         text_target=examples["summary"],  
25         max_length=200,  
26         truncation=True,  
27     )  
28     model_inputs["labels"] = labels["input_ids"]  
29  
30     return model_inputs
```

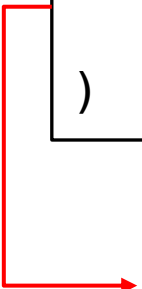


# Data pre-processing (executions)

---

```
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()
```

The screenshot shows the GitHub repository page for `pltrdy / rouge`. The repository is public and has 8 watchers, 100 forks, and 668 stars. The main content area displays a list of files and their commit history:

File	Commit Message	Time Ago
bin	Exclusive option + raw results	5 years ago
rouge	bump version 1.0.1	3 years ago
tests	Merge pull request #35 from pltrdy/update_f...	5 years ago
.gitignore	Introducing ROUGE: A full Python Implement...	7 years ago
LICENSE	Initial commit	7 years ago
MANIFEST.in	0.2.1: python2 support	7 years ago
README.md	Update README.md	3 years ago
setup.py	minor setup.py fix	4 years ago

On the right side, the 'About' section describes the repository as 'A full Python Implementation of the ROUGE Metric (not a wrapper)'. It also lists the license as Apache-2.0 and provides links to the README, activity, and stars. The 'Releases' section shows the latest release, `rouge 1.0.1: results on par wit...`, dated Feb 18, 2020, with a 'Latest' badge.

<https://github.com/pltrdy/rouge>



# Build compute\_metrics for evaluations

```
1 def compute_metrics(eval_pred):
2     predictions, labels = eval_pred
3     decoded_preds = tokenizer.batch_decode(predictions, skip_special_tokens=True)
4     labels = np.where(labels != -100, labels, tokenizer.pad_token_id)
5     decoded_labels = tokenizer.batch_decode(labels, skip_special_tokens=True)
6
7     predictions = [" ".join(jieba.lcut(o)) for o in decoded_preds]
8     references = [" ".join(jieba.lcut(t)) for t in decoded_labels]
9     result = rouge_metric.get_scores(predictions, references, avg=True)
10    score = {f"{rouge_i}_f": v["f"] for rouge_i, v in result.items()}
11    prediction_lens = [
12        np.count_nonzero(pred != tokenizer.pad_token_id) for pred in predictions
13    ]
14    score["gen_len"] = np.mean(prediction_lens)
15    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](#))

```
1  training_args = Seq2SeqTrainingArguments(  
2      output_dir="./results/mt5",  
3      evaluation_strategy="steps",  
4      save_strategy="steps",  
5      eval_steps=1000,  
6      save_steps=10000,  
7      learning_rate=2e-5,  
8      per_device_train_batch_size=32,  
9      per_device_eval_batch_size=32,  
10     weight_decay=0.01,  
11     save_total_limit=3,  
12     num_train_epochs=3,  
13     predict_with_generate=True,  
14     logging_dir=f"./logs/{model_prefix}",  
15     logging_steps=1,  
16     push_to_hub=False,  
17 )
```

Default using Greedy search. Seq2seqTrainer Supports beam search only.



## Setting up Trainer (training part)

```
1  trainer = Seq2SeqTrainer(  
2      model=model,  
3      args=training_args,  
4      train_dataset=train,  
5      eval_dataset=small_valid,  
6      data_collator=data_collator,  
7      compute_metrics=compute_metrics,  
8  )  
9  trainer.args._n_gpu = 1  
10 trainer.train()  
11 results = trainer.predict(valid)  
12 for metric in ["1", "2", "l"]:  
13     rouge_item = f"test_rouge-{metric}"  
14     print(f"{rouge_item}: ", results.metrics[rouge_item])
```

## Setting up Trainer (evaluation part)

```
1  trainer = Seq2SeqTrainer(  
2      model=model,  
3      args=training_args,  
4      train_dataset=train,  
5      eval_dataset=small_valid,  
6      data_collator=data_collator,  
7      compute_metrics=compute_metrics,  
8  )  
9  trainer.args._n_gpu = 1  
10 trainer.train()  
11 results = trainer.predict(valid)  
12 for metric in ["1", "2", "l"]:  
13     rouge_item = f"test_rouge-{metric}"  
14     print(f"{rouge_item}: ", results.metrics[rouge_item])
```

Thank you!

