GAI Project 2.b Text summarization

F74101254 資訊系 張暐俊

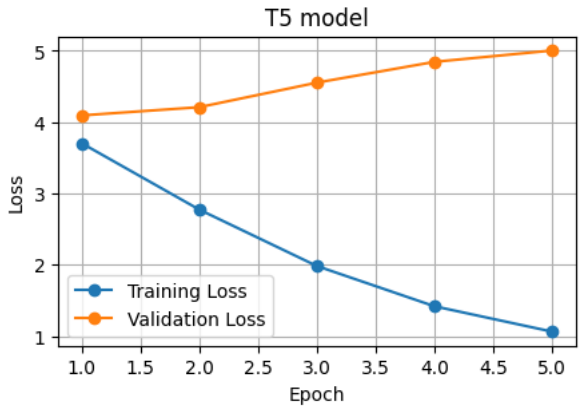
- **Model**

  分別使用了兩種模型：T5、GPT2，前者是助教在範例 code 中使用，後者為建議我們在訓練比較時使用。為了比較兩者的測驗結果，因此兩者都有使用。

  在選擇模型上，皆是選擇已經 pretrained 的 model 去做 finetune。

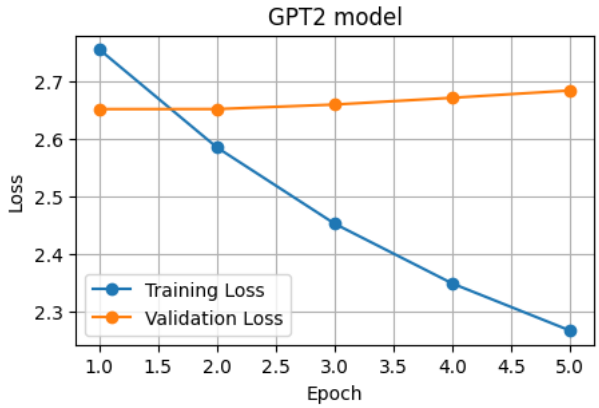
  1. **T5**

```
# https://huggingface.co/Langboat/mengzi-t5-base
t5_model_checkpoint="Langboat/mengzi-t5-base"

t5_tokenizer = T5Tokenizer.from_pretrained(t5_model_checkpoint, cache_dir="./cache/")
t5_model = T5ForConditionalGeneration.from_pretrained(t5_model_checkpoint, cache_dir="./cache/").to(device)
```

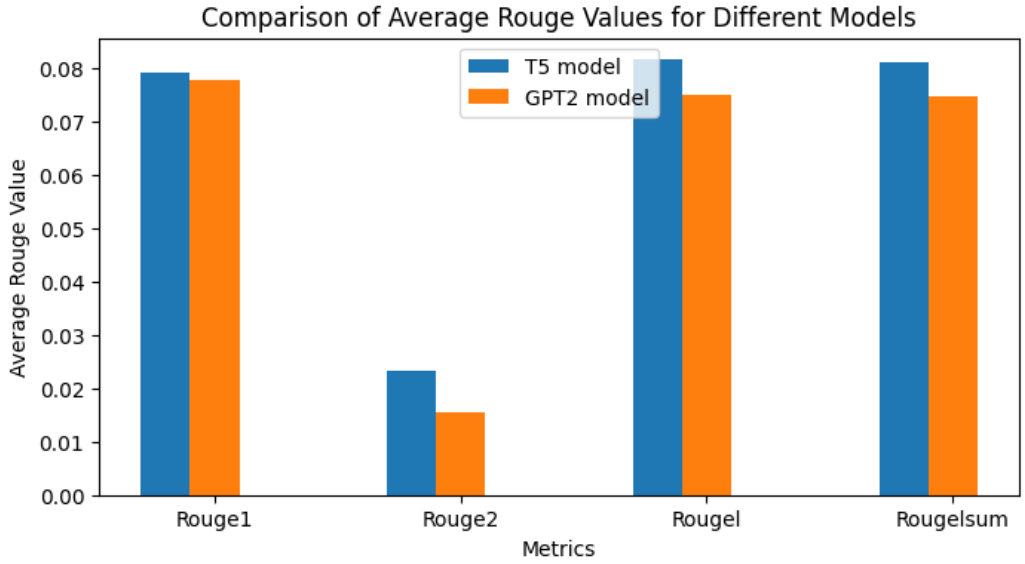


  2. **GPT2**

```
# https://huggingface.co/uer/gpt2-chinese-cluecorpussmall
GPT2_model_checkpoint = "uer/gpt2-distil-chinese-cluecorpussmall"

gpt2_model = GPT2LMHeadModel.from_pretrained(GPT2_model_checkpoint, cache_dir="./cache/").to(device)
gpt2_tokenizer = BertTokenizer.from_pretrained(GPT2_model_checkpoint, cache_dir="./cache/")
```



首先，以 loss rate 來看的話，GPT2 的表現相對較優。另外，在訓練時長上，T5 模型的訓練時長相較於 GPT2 來說得較長，但這可能與選擇的 pretrain model 有關係。



上圖為兩者模型的 rouge 分數比較圖。

而根據 T5 和 GPT2 兩個模型，前者是 Seq2Seq 模型，後者更擅長處理 text generation 的作業。而在我查到的資料中，也說明以 text summaration 的部分來說的話，T5 模型似乎更拿手，從上圖的評測結果或許可見一斑。

- **Dataset**

```
# 載入資料集
data = load_dataset("hugcyp/LCSTS", cache_dir="./cache/")
dataset_ratio = 0.01

# 切割資料
train_size = int(len(data["train"])*dataset_ratio)
validation_size = int(len(data["validation"])*dataset_ratio)
test_size = int(len(data["test"])*dataset_ratio)
```

使用 load_dataset 將資料集從 hugging face 抓下來之後再對各種類別的資料集做切割。要是不做切割的話，光是 train 的 dataset 就有 2,400,000 筆左右的資料，訓練時長上至少要 10 小時以上起跳。

1. T5

```
comment = "总结："

def t5_tokenize(batch):
    texts = [comment + doc for doc in batch["text"]]

    # 做token的動作
    tokenized = t5_tokenizer(texts, max_length=128, truncation=True)
    tokenized_outputs = t5_tokenizer(text_target=batch["summary"], max_length=32, truncation=True)
    tokenized["labels"] = tokenized_outputs["input_ids"]

    return tokenized
```

將「总结：」後方加上我們的 text 欄位後，標記為 input，並將我們的 summary 欄位標記為 labels，讓模型去做比較與計算。

以 train dataset 的第一筆資料為例：

```
Sample 0

text:  新华社受权于18日全文播发修改后的《中华人民共和国立法法》，修改后的立法法分为“总则”“法律”“行政法规”“地方性法规、自治条例和单行条例、规章”“适用与备案审查”“附则”等6章，共计105条。

summary:  修改后的立法法全文公布

decode_text:  总结:新华社受权于18日全文播发修改后的《中华人民共和国立法法》,修改后的立法法分为“总则”“法律”“行政法规”“地方性法规、自治[illegible]“适用与备案审查”“附则”等6章,共计105条。
</s>

decode_summary:  修改后的立法法全文公布</s>
```

Notebook editor cells

2. GPT2

```
comment = "总结："

def gpt2_tokenize(batch):
    texts = []

    # 將text跟summary串接在一起
    for text in batch["text"]:
        concatenated_text = text + comment
        texts.append(concatenated_text)

    # 做token的動作
    tokenized = gpt2_tokenizer(texts, batch["summary"], padding='max_length', max_length=160, truncation=True)
    tokenized_outputs = gpt2_tokenizer(batch["summary"], padding='max_length', max_length=32, truncation=True)
    tokenized["labels"] = tokenized_outputs["input_ids"]

    return tokenized
```

首先在 text 欄位的後方加上「总结：」，並將他跟 summary 欄位一併輸入進 tokenizer 裡做 padding。不先結合在一起再 tokenized 的原因是

希望兩者之間不是默認唯一整個句子，而是兩個句子。將此新生成之結果標記為 input。一樣，將 summary 欄位 tokenized 之後，設定為 label，讓模型去做比較。

```
Sample 0

text:  新华社受权于18日全文播发修改后的《中华人民共和国立法法》，修改后的立法法分为“总则”“法律”“行政法规”“地方性法规、自治条例和单行条例、规章”“适用与备案审查”“附则”等6章，共计105条。

summary:  修改后的立法法全文公布

decode_text:  [CLS] 新 华 社 受 权 于 18 日 全 文 播 发 修 改 后 的 《 中 华 人 民 共 和 国 立 法 法 》 ， 修 改 后 的 立 法 法 分 为 [UNK] 总 则 [UNK] [UNK] 法 律 [UNK] [UNK] 行 政
法 规 [UNK] [UNK] 地 方 性 法 规 、 自 治 条 例 和 单 行 条 例 、 规 章 [UNK] [UNK] 适 用 与 备 案 审 查 [UNK] [UNK] 附 则 [UNK] 等 6 章 ， 共 计 105 条 。 总 结 ： [SEP] 修 改 后
的 立 法 法 全 文 公 布 [SEP] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [
PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]

decode_summary:  [CLS] 修 改 后 的 立 法 法 全 文 公 布 [SEP] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD]
```

- **Train**

訓練方式是使用 trainer 以及 fine tune 的方式，而非從助教的範例 code 去做改動。

1. T5

下圖為 trainer 的各個參數設定

```
t5_training_args = Seq2SeqTrainingArguments(
    report_to="none",
    output_dir="./saved_models",
    evaluation_strategy="epoch",
    learning_rate=0.0001,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    weight_decay=0.01,
    save_total_limit=1,
    num_train_epochs=5,
    metric_for_best_model="rougeL",
    fp16=True,
    predict_with_generate=True,
)
```

```
t5_trainer = Seq2SeqTrainer(
    model=t5_model,
    args=t5_training_args,
    data_collator=t5_data_collator,
    train_dataset=t5_tokenized_dataset["train"],
    eval_dataset=t5_tokenized_dataset["validation"],
    tokenizer=t5_tokenizer,
    compute_metrics=t5_compute_metrics,
)
```

要提到的是，使用的是 Seq2Seq 的 training argument 跟 trainer，而非一般情況下使用的 trainer。

同樣的，data collector 也要配合 T5 模型的特性，去使用 Seq2Seq 的。

```
t5_data_collator = DataCollatorForSeq2Seq(tokenizer=t5_tokenizer, model=t5_model)
```

2. **GPT2**

下圖為 trainer 的各個參數設定

```
gpt2_training_args = TrainingArguments(
    report_to="none",
    output_dir="./saved_models",
    evaluation_strategy="epoch",
    learning_rate=0.0001,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    weight_decay=0.01,
    save_total_limit=1,
    num_train_epochs=5,
    metric_for_best_model="rougeL",
)
```

```
gpt2_trainer = Trainer(
    model=gpt2_model,
    args=gpt2_training_args,
    data_collator=gpt2_data_collator,
    train_dataset=gpt2_tokenized_dataset["train"],
    eval_dataset=gpt2_tokenized_dataset["validation"],
    tokenizer=gpt2_tokenizer,
    compute_metrics=gpt2_compute_metrics,
)
```

GPT2 則是使用一般的 training argument 跟 trainer 即可。

而在 data collector 的 function 上，我們選用的是生成語言使用的。

```
gpt2_data_collator = DataCollatorForLanguageModeling(tokenizer=gpt2_tokenizer, mlm=False)
```

最後，則是用 trainer 去呼叫 train() 這個 function，就可以開始訓練了。

```
[ ]  trainer.train()
     trainer.save_model("result")
```

- **Evaluation**

在計分方式上選用的是範例 code 中使用的「rouge」。

```
rouge_metric = evaluate.load("rouge")
```

並一樣使用 trainer 的 function，也就是呼叫 evaluate()。

```
[ ]  trainer.evaluate()
```

1. **T5**

下圖為 T5 使用的 compute metrics

```
def t5_compute_metrics(eval_pred):
    predictions, labels = eval_pred

    decoded_preds = t5_tokenizer.batch_decode(predictions, skip_special_tokens=True)
    labels = np.where(labels!=-100, labels, t5_tokenizer.pad_token_id)
    decoded_labels = t5_tokenizer.batch_decode(labels, skip_special_tokens=True)

    result = rouge_metric.compute(predictions=decoded_preds, references=decoded_labels, use_stemmer=True)

    gen_len = [np.count_nonzero(pred!=t5_tokenizer.pad_token_id) for pred in predictions]
    result["gen_len"] = np.mean(gen_len)

    return {k: round(v, 4) for k, v in result.items()}
```

evaluate function 評測出來的結果

```
{'eval_loss': 5.004746913909912,
 'eval_rouge1': 0.0942,
 'eval_rouge2': 0.031,
 'eval_rougeL': 0.0977,
 'eval_rougeLsum': 0.0977,
 'eval_gen_len': 10.7674,
 'eval_runtime': 3.7991,
 'eval_samples_per_second': 22.637,
 'eval_steps_per_second': 1.579,
 'epoch': 5.0}
```

2. GPT2

下圖為 GPT2 使用的 compute metrics

```
def gpt2_compute_metrics(eval_pred):
    predictions, labels = eval_pred

    predictions = np.argmax(predictions, axis=-1)
    decoded_preds = gpt2_tokenizer.batch_decode(predictions, skip_special_tokens=True)
    labels = np.where(labels!=-100, labels, gpt2_tokenizer.pad_token_id)
    decoded_labels = gpt2_tokenizer.batch_decode(labels, skip_special_tokens=True)

    result = rouge_metric.compute(predictions=decoded_preds, references=decoded_labels, use_stemmer=True)

    gen_len = [np.count_nonzero(pred!=gpt2_tokenizer.pad_token_id) for pred in predictions]
    result["gen_len"] = np.mean(gen_len)

    return {k: round(v, 4) for k, v in result.items()}
```

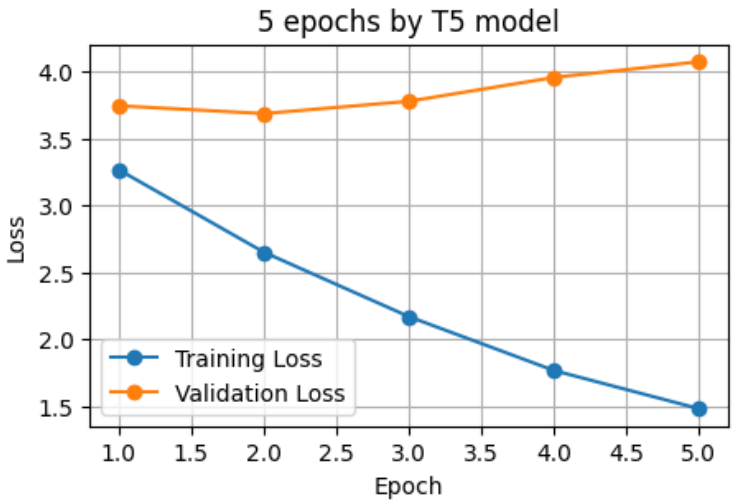
evaluate function 評測出來的結果

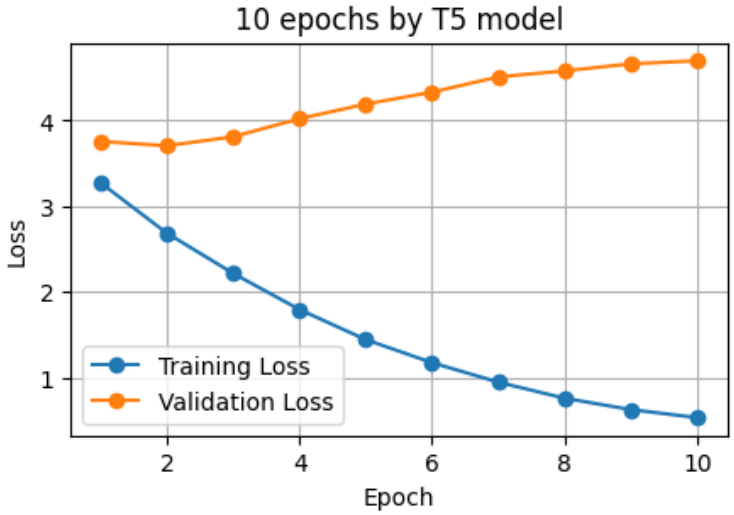
```
{'eval_loss': 2.6843278408050537,
 'eval_rouge1': 0.0751,
 'eval_rouge2': 0.0158,
 'eval_rougeL': 0.0723,
 'eval_rougeLsum': 0.0724,
 'eval_gen_len': 160.0,
 'eval_runtime': 1.9239,
 'eval_samples_per_second': 44.702,
 'eval_steps_per_second': 3.119,
 'epoch': 5.0}
```

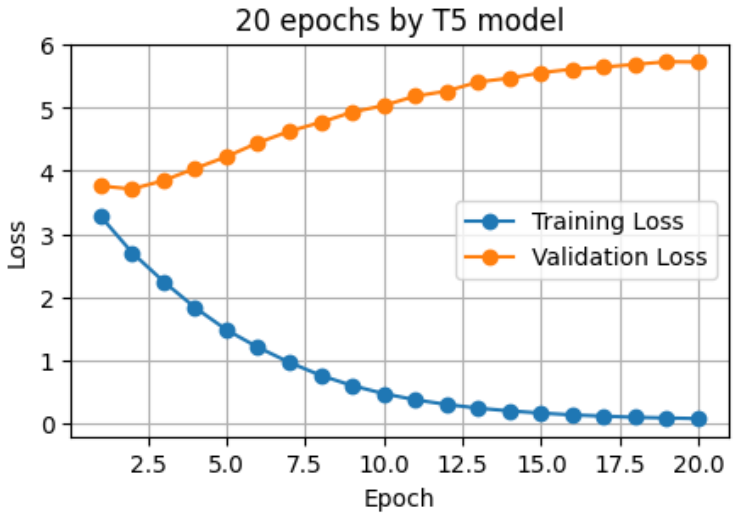
- **Bonus**

  1. **比較不同 epochs 數量下 loss rate 的不同**

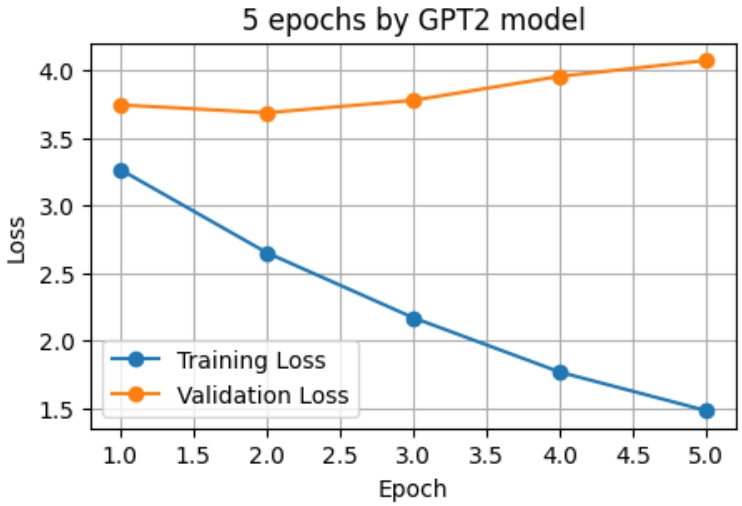
     A. **T5**

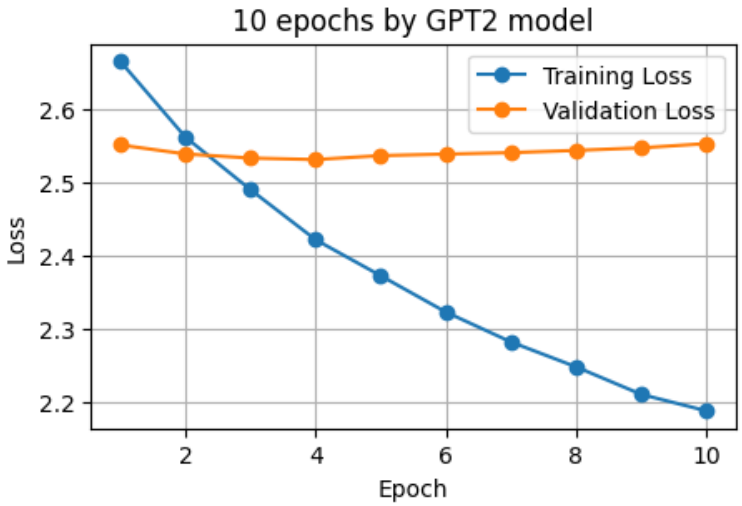


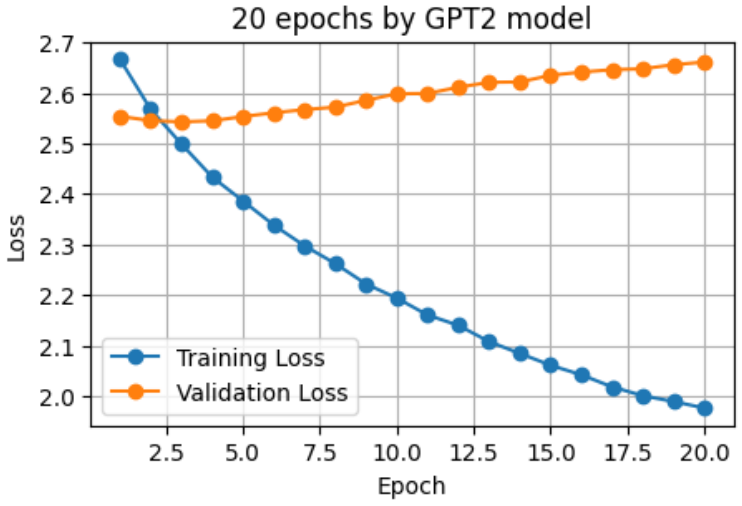




B. GPT2

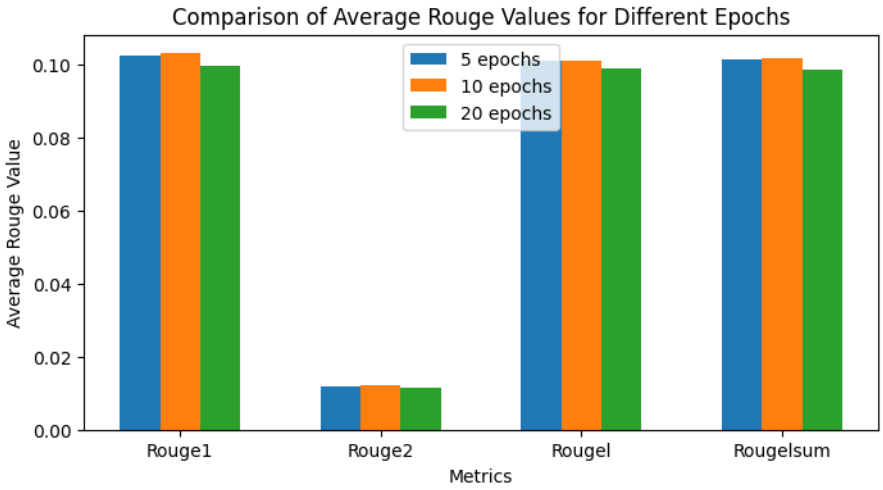

5 epochs by GPT2 model
Loss
Epoch
Training Loss
Validation Loss
4.0
3.5
3.0
2.5
2.0
1.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0


10 epochs by GPT2 model
Loss
Epoch
Training Loss
Validation Loss
2.6
2.5
2.4
2.3
2.2
2
4
6
8
10

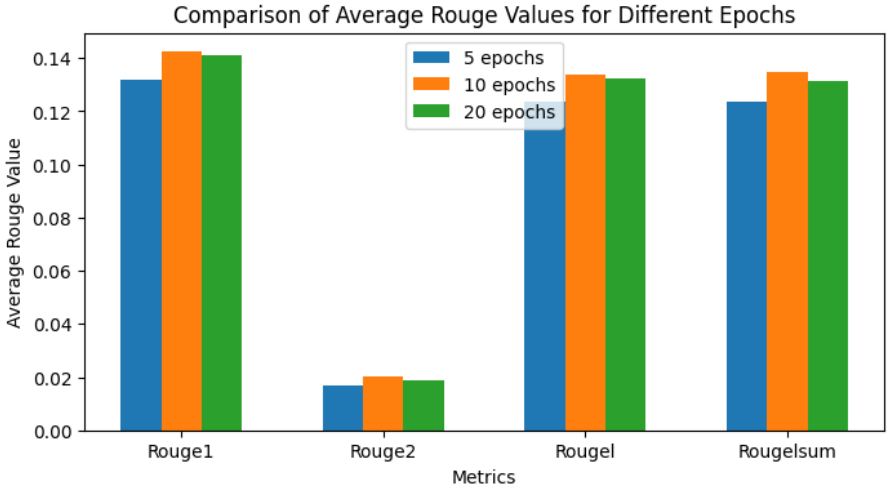

20 epochs by GPT2 model
Loss
Epoch
Training Loss
Validation Loss
2.7
2.6
2.5
2.4
2.3
2.2
2.1
2.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

2. 比較不同 epochs 數量下 rouge 的評分結果

A. T5



B. GPT2



總結來說，無論是 T5 還是 GPT2 哪一個模型，並不是 epochs 數量愈大，效果就越好。