**Group 1**  
D12922010 林鉦育  
D12922014 莊謹譽

# Introduction

This code mainly refers to the lecturer's sample1 and Hugging Face Tutorials, including the content of NLP Course Chapter 0~42, Transformers Tutorial3 and PEFT LoRA4, to fine-tune and experiment the optimization of a pre-training model.

# Method

The method can be mainly divided into the following steps:

1. Pre-processing

2. Model training

3. Post-processing

## **Pre-processing**

In order to facilitate the subsequent model training, all medical records and annotation data are packaged together and converted to a TSV file.

### Directory settings

To make the code able to run on Windows, Google Colab and MacBook Pro M2, we need to set the directories accordingly.

import sys

from pathlib import Path

if "google.colab" in sys.modules:

from google.colab import drive

drive.mount('/content/drive')

RES\_DIR\_PATH = Path("/content/drive/MyDrive/aicup/res")

OUT\_DIR\_PATH = Path("/content/drive/MyDrive/aicup/out")

else:

RES\_DIR\_PATH = Path(Path.cwd().parent, "res")

OUT\_DIR\_PATH = Path(Path.cwd().parent, "out")

if not RES\_DIR\_PATH.exists():

RES\_DIR\_PATH.mkdir(parents=True)

if not OUT\_DIR\_PATH.exists():

OUT\_DIR\_PATH.mkdir(parents=True)

### Pre-processing main flow

Declare DatasetPreprocessor for subsequent uses.

from typing import List

from itertools import chain

class DatasetPreprocessor:

def \_\_init\_\_(

self,

ans\_file\_path: Path,

med\_dir\_path: Path,

output\_tsv\_path: Path,

) -> None:

self.ans\_file\_path = ans\_file\_path

self.med\_dir\_path = med\_dir\_path

self.output\_tsv\_path = output\_tsv\_path

@staticmethod

def read\_file(path:Path, encoding:str="utf-8-sig") -> List[str]:

with open(path, encoding=encoding) as fr:

return fr.readlines()

@staticmethod

def write\_file(path:Path, data\_list: List, encoding:str="utf-8") -> None:

with open(path, "w", encoding=encoding) as fw:

for data\_line in data\_list:

fw.write(data\_line)

def create\_tsv(self):

ans\_file\_dict = self.get\_answer\_dict()

med\_ans\_pair\_list = chain.from\_iterable([

self.process\_medical\_report(med\_file\_name, ans\_file\_dict)

for med\_file\_name in ans\_file\_dict.keys()

])

self.write\_file(self.output\_tsv\_path, med\_ans\_pair\_list)

def get\_answer\_dict(self):

"""

處理 anwser.txt 標註檔案

output : annotation dicitonary

"""

from collections import defaultdict

ans\_dict = defaultdict(list)

lines = self.read\_file(self.ans\_file\_path)

for line in lines:

items = line.strip("\n").split("\t")

items\_file = items[0]

items\_data = {}

items\_data["phi"] = items[1]

items\_data["st\_idx"] = int(items[2])

items\_data["ed\_idx"] = int(items[3])

items\_data["entity"] = items[4]

if len(items) == 6:

items\_data["normalize\_time"] = items[5]

ans\_dict[items\_file].append(items\_data)

return ans\_dict

def process\_medical\_report(self, med\_file\_name, ans\_file\_dict,):

"""

處理單個病理報告

output : 處理完的 sequence pairs

"""

med\_file\_path = Path(self.med\_dir\_path, med\_file\_name).with\_suffix('.txt')

med\_report = "".join(self.read\_file(med\_file\_path))

bounary, item\_idx, phi\_info, phi\_pairs = 0, 0, "", []

new\_line\_idx = 0

for char\_idx, char in enumerate(med\_report):

if char == "\n":

new\_line\_idx = char\_idx + 1

med\_info\_seg = med\_report[bounary:new\_line\_idx]

if med\_info\_seg == "\n":

continue

phi\_info = phi\_info.strip("\\n") if phi\_info else "PHI:Null"

med\_info = med\_info\_seg.strip().replace("\t", " ")

phi\_pair = f"{med\_file\_name}\t {new\_line\_idx}\t {med\_info}\t {phi\_info}\n"

phi\_pairs.append(phi\_pair)

bounary = new\_line\_idx

phi\_info = ""

med\_item = ans\_file\_dict[med\_file\_name][item\_idx]

if char\_idx == med\_item["st\_idx"]:

phi = med\_item["phi"]

entity = med\_item["entity"]

normalize\_time = med\_item.get("normalize\_time", "")

if normalize\_time:

phi\_info += f"{phi}:{entity}=>{normalize\_time}\\n"

else:

phi\_info += f"{phi}:{entity}\\n"

if item\_idx == len(ans\_file\_dict[med\_file\_name]) - 1:

continue

item\_idx += 1

return phi\_pairs

### Export TSV files

Since the two tasks have the same format except of focusing on different data, we convert and merge the two datasets to avoid data imbalance during training and facilitate the following model training.

DatasetPreprocessor(

ans\_file\_path=Path(RES\_DIR\_PATH, "First\_Dataset/answer.txt"),

med\_dir\_path=Path(RES\_DIR\_PATH, "First\_Dataset/First\_Phase\_Text\_Dataset"),

output\_tsv\_path=Path(OUT\_DIR\_PATH, "first\_train\_single\_line.tsv"),

).create\_tsv()

DatasetPreprocessor(

ans\_file\_path=Path(RES\_DIR\_PATH, "Second\_Dataset/answer.txt"),

med\_dir\_path=Path(RES\_DIR\_PATH, "Second\_Dataset/Second\_Phase\_Text\_Dataset"),

output\_tsv\_path=Path(OUT\_DIR\_PATH, "second\_train\_single\_line.tsv"),

).create\_tsv()

DatasetPreprocessor.write\_file(

Path(OUT\_DIR\_PATH, "merged\_train\_single\_line.tsv"),

DatasetPreprocessor.read\_file(Path(OUT\_DIR\_PATH, "first\_train\_single\_line.tsv")) +

DatasetPreprocessor.read\_file(Path(OUT\_DIR\_PATH, "second\_train\_single\_line.tsv")),

)

## Model Training

The code bases on the lecturer's sample code and Hugging Face Tutorials. However, not all is used in the end. Those unused parts will be discussed in“Discuss in Experiment & Discussion”.

### Install packages & mount folders

To enable the code running on Windows, Google Colab and MacBook Pro M2, we need to install the package and mount folders in different ways.

import sys

from pathlib import Path

if "google.colab" in sys.modules:

from google.colab import drive

drive.mount('/content/drive')

!pip install torch --quiet

!pip install transformers datasets evaluate peft accelerate sentencepiece --quiet

!pip install numpy matplotlib tqdm --quiet

!pip install islab-opendeid --quiet

else:

%pip install torch --quiet

%pip install transformers datasets evaluate peft accelerate sentencepiece --quiet

%pip install numpy matplotlib tqdm --quiet

%pip install islab-opendeid --quiet

%pip install ipywidgets nbformat nbclient widgetsnbextension pandas-profiling --qui

### Define TsvDatasetHelper to read TSV

The target TSV are the output of pre-processing.

from pathlib import Path

from datasets import load\_dataset, Features, Value

class TsvDatasetHelper:

TsvFeatures = Features(

{

"fid": Value("string"),

"idx": Value("int64"),

"content": Value("string"),

"label": Value("string"),

}

)

@staticmethod

def get\_dataset(tsv\_path: Path):

data\_files = tsv\_path.as\_posix()

print(data\_files)

return load\_dataset(

path="csv",

delimiter="\t",

data\_files=data\_files,

features=TsvDatasetHelper.TsvFeatures,

column\_names=TsvDatasetHelper.TsvFeatures.keys(),

keep\_default\_na=False,

)

### Load TSV Dataset

tsv\_path = Path(RES\_DIR\_PATH, "Demo\_Dataset/opendid\_set1.tsv")

dataset = TsvDatasetHelper.get\_dataset(tsv\_path)

dataset

### Print TSV Dataset

from pprint import pprint

pprint(dataset["train"][0])

pprint(dataset["train"][1])

pprint(dataset["train"][7])

### Select samples for Exploratory Data Analysis

from torch.utils.data import random\_split

print(len(dataset["train"]))

sub\_datasets = random\_split(dataset["train"], [20000, 65736])

print(len(sub\_datasets[0]))

for i in range(4):

pprint(sub\_datasets[0][i])

### Define TokenHelper

class TokenHelper:

Bos = "<|endoftext|>"

Eos = "<|END|>"

Pad = "<|pad|>"

Sep = "\n\n####\n\n"

SpecialTokens = {

"bos\_token": Bos,

"eos\_token": Eos,

"pad\_token": Pad,

"sep\_token": Sep,

}

@staticmethod

def add\_token(med\_info, phi\_info):

return "{}{}{}{}{}".format(

TokenHelper.Bos,

med\_info,

TokenHelper.Sep,

phi\_info,

TokenHelper.Eos,

)

### Define the Hugging Face checkpoint & revision

checkpoint = "EleutherAI/pythia-70m" # "EleutherAI/pythia-70m-deduped"

revision = "step3000"

### Set tokenizer and show special tokens

from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from\_pretrained(checkpoint, revision=revision)

# 印出本來的 special token

pprint(tokenizer.special\_tokens\_map)

tokenizer.add\_special\_tokens(TokenHelper.SpecialTokens)

tokenizer.padding\_side = "left"

# 印出之後的 special token

pprint(tokenizer.special\_tokens\_map)

### Show input\_ids and attention\_mask before and after tokenizer processing

raw\_inputs = [

TokenHelper.add\_token(

" 9364819.RAN\\nMINTANIA, JEFFRY ",

" ID: 9364819.RAN\\nNAME: MINTANIA, JEFFRY ",

),

TokenHelper.add\_token(" This is a sentence ", " PHI: NULL "),

]

result = tokenizer(raw\_inputs, padding=True)

for attention\_mask in result["attention\_mask"]:

print(attention\_mask)

print("=" \* 120)

for input\_id in result["input\_ids"]:

print(input\_id, "\n")

print(tokenizer.decode(input\_id))

print("-" \* 120)

### Modify Collator

import torch

def collate\_batch\_with\_prompt\_template(

batch,

tokenizer,

template="<|endoftext|> \_\_CONTENT\_\_\n\n####\n\n\_\_LABEL\_\_ <|END|>",

IGNORED\_PAD\_IDX=-100,

):

"""template: \_\_CONTENT\_\_ and \_\_LABEL\_\_ will be replaced with the content and the corresponding labels."""

# default template: {bos} {data['content']} {sep}

texts = [

template.replace("\_\_LABEL\_\_", data["label"]).replace(

"\_\_CONTENT\_\_", data["content"]

)

for data in list(batch)

]

encoded\_seq = tokenizer(texts, padding=True)

indexed\_tks = torch.tensor(encoded\_seq["input\_ids"])

attention\_mask = torch.tensor(encoded\_seq["attention\_mask"])

encoded\_label = torch.tensor(encoded\_seq["input\_ids"])

encoded\_label[encoded\_label == tokenizer.pad\_token\_id] = IGNORED\_PAD\_IDX

return indexed\_tks, encoded\_label, attention\_mask

### Create PyTorch Dataloader

from torch.utils.data import DataLoader

train\_data = list(sub\_datasets[0])

train\_dataloader = DataLoader(

train\_data,

batch\_size=3,

shuffle=False,

collate\_fn=lambda batch: collate\_batch\_with\_prompt\_template(batch, tokenizer),

)

for batch in train\_dataloader:

for batch\_item in batch:

print(batch\_item.shape)

print(batch\_item)

print("=" \* 120)

break

### Create PyTorch Dataloader of The Lecturer

The lecturer ever used OpenDeidBatchSampler。

from islab.aicup import OpenDeidBatchSampler

BATCH\_SIZE = 8

bucket\_train\_dataloader = DataLoader(

train\_data,

batch\_sampler=OpenDeidBatchSampler(train\_data, BATCH\_SIZE),

collate\_fn=lambda batch: collate\_batch\_with\_prompt\_template(batch, tokenizer),

pin\_memory=True,

)

# for idx, batch in enumerate(bucket\_train\_dataloader):

# print(batch)

# print(batch[0].shape)

# print(batch[1].shape)

# break

### Set Model Config (PEFT & LoRA)

Theoretically, [following the instructions of Hugging Face](https://huggingface.co/docs/peft/task_guides/image_classification_lora#load-and-prepare-a-model), PEFT and LoRA should enhance the speed because the amount of training parameters is reduced to 0.77% ("trainable params: 667493 || all params: 86466149 || trainable%: 0.77"). But, for some unknown reasons, it took longer than the original version, so config=peft\_config was not actually set in the end.

from peft import LoraConfig, TaskType

from peft import AutoPeftModelForCausalLM

from peft import get\_peft\_model

from transformers import AutoConfig

checkpoint = "EleutherAI/pythia-70m" # "EleutherAI/pythia-70m-deduped"

peft\_config = LoraConfig(

task\_type=TaskType.CAUSAL\_LM,

inference\_mode=False,

r=8,

lora\_alpha=32,

lora\_dropout=0.1,

)

# the model config to which we add the special tokens

config = AutoConfig.from\_pretrained(

checkpoint,

bos\_token\_id=tokenizer.bos\_token\_id,

eos\_token\_id=tokenizer.eos\_token\_id,

pad\_token\_id=tokenizer.pad\_token\_id,

sep\_token\_id=tokenizer.sep\_token\_id,

output\_hidden\_states=False,

)

model = AutoModelForCausalLM.from\_pretrained(

checkpoint, revision="step3000", config=config

)

model = AutoPeftModelForCausalLM.from\_pretrained(

checkpoint, revision="step3000", config=config

)

model = get\_peft\_model(model, peft\_config)

model.print\_trainable\_parameters()

model

### Accelerate Model Training

Apply Adam and adjust PyTorch to make it run as soon as possible in Windows, Google Colab, and MacBook Pro M2.

from transformers import get\_linear\_schedule\_with\_warmup

from torch.optim import AdamW

EPOCHS = 3 # CHANGE TO THE NUMBER OF EPOCHS YOU WANT

optimizer = AdamW(model.parameters(), lr=3e-5) # YOU CAN ADJUST LEARNING RATE

device = torch.device(

"mps"

if torch.backends.mps.is\_available()

else "cuda"

if torch.cuda.is\_available()

else "cpu"

)

model.resize\_token\_embeddings(len(tokenizer))

model.to(device)

### Model Training

from tqdm.auto import tqdm

global\_step = EPOCHS \* len(bucket\_train\_dataloader)

progress\_bar = tqdm(range(global\_step), desc="Step")

total\_loss = 0

model.train()

for epoch in range(EPOCHS):

model.train()

total\_loss = 0

predictions, true\_labels = [], []

for seqs, labels, masks in bucket\_train\_dataloader:

seqs = seqs.to(device)

labels = labels.to(device)

masks = masks.to(device)

model.zero\_grad()

outputs = model(seqs, labels=labels, attention\_mask=masks)

logits = outputs.logits

loss = outputs.loss

loss = loss.mean()

total\_loss += loss.item()

loss.backward()

optimizer.step()

progress\_bar.update(1)

avg\_train\_loss = total\_loss / len(bucket\_train\_dataloader)

print(f"Epoch {epoch}:\nAverage train loss: {avg\_train\_loss}")

### Read validation set

valid\_data = DatasetHelper.get\_dataset(

Path(RES\_DIR\_PATH, "Demo\_Dataset/opendid\_valid.tsv")

)

valid\_list = list(valid\_data["train"])

valid\_list[0]

### Export the Outcome

from tqdm.notebook import tqdm

from islab.aicup import aicup\_predict

from datetime import datetime

BATCH\_SIZE = 32

# datetime object containing current date and time in format YY-mm-dd\_H-M-S

now\_dt\_string = datetime.now().strftime("%Y-%m-%d\_%H-%M-%S")

path = Path(OUT\_DIR\_PATH, f"{now\_dt\_string}\_answer.txt")

merged\_model = model.merge\_and\_unload()

with open(path, "w", encoding="utf8") as f:

for i in tqdm(range(0, len(valid\_list), BATCH\_SIZE)):

with torch.no\_grad():

seeds = valid\_list[i : i + BATCH\_SIZE]

outputs = aicup\_predict(merged\_model, tokenizer, input=seeds)

for o in outputs:

f.write(o)

f.write("\n")

## Post-processing

Use Regular Expression to do post-processing. Because the training data is handled in row by row, there will be multiple data fields in a row. That would make the model consider these data fields to be all the same, so we need to separate them one by one.

### Define function and constant

import re

def read\_file(path, encoding = 'utf-8-sig'):

with open(path , encoding = encoding) as fr:

return fr.readlines()

PHI\_KEYS = [

'PATIENT', 'DOCTOR', 'USERNAME',

'PROFESSION',

'ROOM', 'DEPARTMENT', 'HOSPITAL', 'ORGANIZATION', 'STREET', 'CITY', 'STATE', 'COUNTRY', 'ZIP', 'LOCATION-OTHER',

'AGE',

'DATE', 'TIME', 'DURATION', 'SET',

'PHONE', 'FAX', 'EMAIL', 'URL', 'IPADDR',

'SSN', 'MEDICALRECORD', 'HEALTHPLAN', 'ACCOUNT', 'LICENSE', 'VECHICLE', 'DEVICE', 'BIOID', 'IDNUM',

'OTHER'

]

NORMALIZED\_PHI\_KEYS = ['DATE', 'TIME', 'DURATION', 'SET']

### Post-processing main flow

filename = '2023-11-27\_08-47-30\_answer'

filename = 'esun\_answer'

valid\_data = dict()

for valid in read\_file(f'../out/first\_phase\_valid.txt'):

if valid == '\n':

continue

[fid, start, content] = valid.split('\t')

if fid in valid\_data:

valid\_data[fid].append((start, content))

else:

valid\_data[fid] = [(start, content)]

with open (f'../out/{filename}\_post.txt' , 'w' , encoding = 'utf-8') as fw:

for answer in read\_file(f'../out/{filename}.txt'):

### == Data sample ===========================================

### fid phi start end value normalized

### file8786 DOCTOR 2975 2984 V Strimel

### file8786 DATE 4462 4471 12/6/2067 2067-06-12

### ==========================================================

fid, phi, start, end, value, \*normalized = answer.rstrip().split('\t')

# if fid != 'file9392':

# continue

if len(normalized) > 0 and '\\n' in normalized[0]:

normalized = [normalized[0].split('\\n', 1)[0]]

if '\\n' not in value:

if phi in PHI\_KEYS:

if phi in NORMALIZED\_PHI\_KEYS and len(normalized) == 0:

# print(phi, value)

continue

fw.write('\t'.join([fid, phi, start, end, value, \*normalized]))

fw.write('\n')

# print(answer)

continue

phi\_info = phi + ': ' + value

# print(phi\_info, start, end, normalized)

phi\_maps = list(dict.fromkeys(phi\_info.split('\\n')))

for phi\_item in phi\_maps:

# print(phi\_item)

if ': ' not in phi\_item:

continue

[phi, value] = phi\_item.split(': ', 1)

if phi not in PHI\_KEYS:

continue

for sindex, content in valid\_data[fid]:

if int(sindex) == int(start) + 1:

for found in re.finditer(re.escape(value), content):

new\_start = int(sindex) + found.start()

new\_end = int(sindex) + found.end()

if phi in NORMALIZED\_PHI\_KEYS and len(normalized) == 0:

# print(phi, value)

continue

fw.write('\t'.join([fid, phi, str(new\_start), str(new\_end), value, \*(normalized if phi in NORMALIZED\_PHI\_KEYS else [])]))

fw.write('\n')

# print('\t'.join([fid, phi, str(new\_start), str(new\_end), value, \*(normalized if phi in NORMALIZED\_PHI\_KEYS else [])]))

break

# Experiment & Discussion

Here are four parts：

1. Single Line vs Multi Line
2. Special Token & Collator
3. PEFT & LoRA
4. Other CausalLM Checkpoint

## Single Line vs Multi Line

Because medical records often have multiple data fields in one row, the model can only recognize one label and consider all the data fields to be the same, causing errors in the judgment.

Therefore, to solve this problem, we tried to separate, making one data field corresponding to one line, and then proceed to train the model. However, the experimental results were not that good. Due to the context lost, make the model hard to identify data fields.In the end, the Single Line method was adopted, combined with post-processing.

The following code (the location of the bold underline) is the difference between Single Line and Multi Line.

def process\_medical\_report(self, med\_file\_name, ans\_file\_dict,):

"""

處理單個病理報告

output : 處理完的 sequence pairs

"""

med\_file\_path = Path(self.med\_dir\_path, med\_file\_name).with\_suffix('.txt')

med\_report = "".join(self.read\_file(med\_file\_path))

bounary, item\_idx, phi\_info, phi\_pairs = 0, 0, "", []

new\_line\_idx = 0

for char\_idx, char in enumerate(med\_report):

if char == "\n":

new\_line\_idx = char\_idx + 1

med\_info\_seg = med\_report[bounary:new\_line\_idx]

if med\_info\_seg == "\n":

continue

phi\_info = phi\_info.strip("\\n") if phi\_info else "PHI:Null"

med\_info = med\_info\_seg.strip().replace("\t", " ")

**phi\_pair = f"{med\_file\_name}\t {new\_line\_idx}\t {med\_info}\t {phi\_info}\n"**

**phi\_pairs.append(phi\_pair)**

bounary = new\_line\_idx

phi\_info = ""

def process\_medical\_report(self, med\_file\_name, ans\_file\_dict,):

"""

處理單個病理報告

output : 處理完的 sequence pairs

"""

med\_file\_path = Path(self.med\_dir\_path, med\_file\_name).with\_suffix('.txt')

med\_report = "".join(self.read\_file(med\_file\_path))

bounary, item\_idx, phi\_info, phi\_pairs = 0, 0, "", []

new\_line\_idx = 0

for char\_idx, char in enumerate(med\_report):

if char == "\n":

new\_line\_idx = char\_idx + 1

med\_info\_seg = med\_report[bounary:new\_line\_idx]

if med\_info\_seg == "\n":

continue

phi\_info = phi\_info.strip("\\n") if phi\_info else "PHI:Null"

med\_info = med\_info\_seg.strip().replace("\t", " ")

**for info in phi\_info.split("\\n"):**

**phi\_pair = f"{med\_file\_name}\t {new\_line\_idx}\t {med\_info}\t {info}\n"**

**phi\_pairs.append(phi\_pair)**

bounary = new\_line\_idx

phi\_info = ""

## Special Token & Collator

The lecturer's sample code adds additional special tokens and change the special token mapping in the original model. For example, the original model mapped *eos\_token* to *<|endoftext|>*, but after the process, it mapped to *<|END|>*. The Collator provided by aicup is also based on such a modified mapping.

Therefore, we tried to pull out this section and add special tokens according the annotations of the original model. Unfortunately, the result was not outstanding and made it hard to experiment a new model. So we still keep it as it is.

{

"bos\_token": "<|endoftext|>",

"eos\_token": "<|endoftext|>",

"unk\_token": "<|endoftext|>"

}

class TokenHelper:

Bos = "<|endoftext|>"

Eos = "<|END|>"

Pad = "<|pad|>"

Sep = "\n\n####\n\n"

SpecialTokens = {

"bos\_token": Bos,

"eos\_token": Eos,

"pad\_token": Pad,

"sep\_token": Sep,

}

@staticmethod

def add\_token(med\_info, phi\_info):

return "{}{}{}{}{}".format(

TokenHelper.Bos,

med\_info,

TokenHelper.Sep,

phi\_info,

TokenHelper.Eos,

)

import torch

def collate\_batch\_with\_prompt\_template(

batch,

tokenizer,

template="<|endoftext|> \_\_CONTENT\_\_\n\n####\n\n\_\_LABEL\_\_ <|END|>",

IGNORED\_PAD\_IDX=-100,

):

"""template: \_\_CONTENT\_\_ and \_\_LABEL\_\_ will be replaced with the content and the corresponding labels."""

# default template: {bos} {data['content']} {sep}

texts = [

template.replace("\_\_LABEL\_\_", data["label"]).replace(

"\_\_CONTENT\_\_", data["content"]

)

for data in list(batch)

]

encoded\_seq = tokenizer(texts, padding=True)

indexed\_tks = torch.tensor(encoded\_seq["input\_ids"])

attention\_mask = torch.tensor(encoded\_seq["attention\_mask"])

encoded\_label = torch.tensor(encoded\_seq["input\_ids"])

encoded\_label[encoded\_label == tokenizer.pad\_token\_id] = IGNORED\_PAD\_IDX

## PEFT & LoRA

Under the Pretrain-and-Fine-Tuning methodology, an LLM is used with a Downstream Task for training. This approach would have a lot of training parameters and take a long-time training. Make it hard to experiment on the local side.

Therefore, we tried PEFT and LoRA, as a plugin in the original LLM, to reduce the amount of training parameters and speed up the training process. However, the actual execution was not as good as expected. Sometimes it even took longer, so it was finally shelved and given up.

from peft import LoraConfig, TaskType

from peft import AutoPeftModelForCausalLM

from peft import get\_peft\_model

from transformers import AutoConfig

checkpoint = "EleutherAI/pythia-70m" # "EleutherAI/pythia-70m-deduped"

peft\_config = LoraConfig(

task\_type=TaskType.CAUSAL\_LM,

inference\_mode=False,

r=8,

lora\_alpha=32,

lora\_dropout=0.1,

)

# the model config to which we add the special tokens

config = AutoConfig.from\_pretrained(

checkpoint,

bos\_token\_id=tokenizer.bos\_token\_id,

eos\_token\_id=tokenizer.eos\_token\_id,

pad\_token\_id=tokenizer.pad\_token\_id,

sep\_token\_id=tokenizer.sep\_token\_id,

output\_hidden\_states=False,

)

model = AutoModelForCausalLM.from\_pretrained(

checkpoint, revision="step3000", config=config

)

model = AutoPeftModelForCausalLM.from\_pretrained(

checkpoint, revision="step3000", config=config

)

model = get\_peft\_model(model, peft\_config)

model.print\_trainable\_parameters()

model

## Other CausalLM Checkpoint

In addition to Pythia-70m, there are many CausalLM Checkpoints on Hugging Face, such as Pythia with more parameters. Theoretically, the more parameters, the stronger the language model capability, and the results should be better. However, in fact, after running a few times, no significant results were achieved. Instead, the TPU computing quota purchased on Google Colab was consumed rapidly. So, in the end, a lightweight model was adopted.

# Conclusion

In terms of the results of the competition, our performance is not that good because we only improved the existing methods and did not propose a better solution. In the contrast, in terms of learning effects, we did put many concepts and knowledge into practice. It helps laying a solid foundation for future study of NLP and other related fields.

# Contribution

* 林鉦育 D12922010
  + EDA exploratory data analysis
  + GPT LLM experiment
* 莊謹譽 D12922014
  + Preprocessing
  + Postprocessing
  + Other Experiment

# Reference

1. [AI CUP 2023 隱私保護與醫學數據化競賽 – 線上課程教材](https://drive.google.com/drive/folders/1NFSK-MfS64Bp8MsOyZJ8TJLzSTIKiGE8)
2. [Hugging Face NLP Course - 課程簡介](https://huggingface.co/learn/nlp-course/zh-TW/chapter0/1?fw=pt)
3. [Hugging Face Transformers 簡介](https://huggingface.co/docs/transformers/v4.36.1/zh/index)
4. [Hugging Face PEFT - LoRA](https://huggingface.co/docs/peft/conceptual_guides/lora)