**第一組**  
D12922010 林鉦育  
D12922014 莊謹譽

# Introduction

此次比賽主要參考講師所提供的範例程式碼1，並搭配Hugging Face上的教學來微調並實驗如何優化一個預訓練模型，其中包含NLP Course Chapter 0~4的內容**錯誤! 找不到參照來源。**、Transformers教程**錯誤! 找不到參照來源。**以及PEFT中的LoRa**錯誤! 找不到參照來源。**。

# Method

方法主要可分為以下步驟：

1. 後處理
2. 前處理
3. 模型訓練
4. 後處理

## **前處理**

為了方便後續模型訓練，先將所有病歷如：1.txt，及標註資料answer.txt打包轉成TSV檔。

### 設定資料夾

為讓程式碼能在Google Colab與MacBook Pro M2上執行，在路徑上進行調整。

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)

### 前處理主要流程

宣告DatasetPreprocessor，以便後續重複利用。

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

### 匯出TSV檔

由於「病患隱私資訊擷取」與「時間資訊正規化」兩項任務，其病歷與標注資料格式皆一致，只是側重內容不同，如前者缺乏時間相關標注資料，後者缺乏其他隱私資訊，故為避免訓練時資料不平衡，在此將兩份資料合併，以利後續模型訓練。

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")),

)

## 模型訓練

以講師範例程式碼為基礎1，並搭配Hugging Face NLP Course Chapter 0~4的內容**錯誤! 找不到參照來源。**、Transformers教程**錯誤! 找不到參照來源。**以及PEFT中的LoRa**錯誤! 找不到參照來源。**來做修改，但並沒有所有修改都用上，沒用上的部分會在Experiment & Discussion做討論。

### 依不同環境安裝套件並掛載資料夾

為讓程式碼能在Google Colab與MacBook Pro M2上執行，以不同方法安裝套件。

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

### 定義 TsvDatasetHelper 以便後續讀取TSV檔

處理對象為前處理之TSV檔

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,

)

### 載入 TSV Dataset

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

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

dataset

### 印出 TSV Dataset 並檢視

from pprint import pprint

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

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

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

### 挑選 20000 個 Sample 來做探索式資料分析(Exploratory Data Analysis, EDA)

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])

### 定義 TokenHelper 以便後續處理

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,

)

### 宣告接下來要在 Hugging Face 中使用的 checkpoint 與 revision

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

revision = "step3000"

### 設定 tokenizer 並展示其 special token

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)

### 展示 tokenizer 編碼前後的 input\_ids 與 attention\_mask

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)

### 修改Collator

有實驗不同Token，但最後還是採用講師版本。

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

### 建立個人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

### 建立講師PyTorch Dataloader

講師有使用他們的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

### 模型訓練設定 (PEFT與LoRA加速)

理論上，[照Hugging Face的教學操作](https://huggingface.co/docs/peft/task_guides/image_classification_lora#load-and-prepare-a-model)，速度的確有提升，因為訓練參數量下降至0.77%("trainable params: 667493 || all params: 86466149 || trainable%: 0.77")。但不知為何，實際套用後時間還變得更長，因此最後並沒有真的設定config=peft\_config

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

### 模型訓練加速

套用Adam，並調整PyTorch，好在Windows環境、Google Colab環境、乃至MacBook Pro M2晶片上，都能用最快速度運行。

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)

### 模型訓練

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}")

### 讀取驗證集

valid\_data = DatasetHelper.get\_dataset(

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

)

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

valid\_list[0]

### 匯出驗證結果

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")

## 後處理

透過Regular Expression處理資料欄位都在同一行的問題。因為訓練時是以一行為單位進行訓練，因此當一行有多個資料欄位，會被模型視做同一個，所以需再對其進行細部的切分。

### 宣告函式與PHI常數

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']

### 後處理主要流程

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

此次比賽主要實驗了四個部分：

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

## Single Line vs Multi Line

因為病歷中常常會出現一行有多個資料欄位，以至於一行有多個資料標註，導致模型只能辨識出一種標註，而且還把所有資料欄位黏在一起，造成判定上的錯誤。

因此，為解決這個問題，有嘗試將這種情況分成多行，確保一行只對應到一種資料，再下去訓練模型。然而實驗結果並不好，因為失去前後文，所以很多資料反而抓不出來。

故最終還是採用Single Line的方法，再搭配後處理來解決資料欄位黏在一起的問題。

以下程式碼(粗體底線所在位置)即是Single Line跟Multi Line差異所在。

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

講師範例程式碼有額外添加Special Token，但在添加的過程中，也改了原始Special Token的Mapping。例如，原本模型將eos\_token對應到<|endoftext|>，但修改後卻對應到<|END|>。而aicup所提供的Collator也是直接依據這樣的Mapping下去做處理。

因此曾經試著把這段拉出來，以原本模型的標註來添加Special Token，但嘗試後效果也沒有特別突出，而且每試一個模型都要去修改，所以最後決定維持原樣。

[special\_tokens\_map.json · EleutherAI/pythia-70m at main (huggingface.co)](https://huggingface.co/EleutherAI/pythia-70m/blob/main/special_tokens_map.json)

{

"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

在原本Pretrain-and-Fine-Tuning的方法下，都是拿一個LLM搭配一個Downstream Task下去訓練。但這會導致訓練的參數過多，訓練時間過長的問題(原本LLM的參數也沒有Freeze)。同時，這也不利在Local端Trial and Error。

故想嘗試PEFT跟LoRA，只在原本LLM中外掛一個Sub Task，同時Freeze原本LLM，以降低訓練參數量，並加速訓練過程。

然而這樣的效果只在Hugging Face教學範例上發生，實際運用卻不如預期，有時甚至要跑更久，因此最後擱置放棄。

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

除了Pythia-70m，Hugging Face上還有許多CausalLM的Checkpoint，例如參數量更多Pythia。理論上，參數量越多，語言模型能力越強，結果應該更好，但實際上跑了幾個，並沒有達到顯著的效果，反而已快耗盡在Google Colab上購買的TPU運算額度，所以最終還是採用輕量化的模型。

# Conclusion

以比賽結果來講，此次專題僅在既有方法上改良，並沒有提出更好的解法，因此結果頂多差強人意；但以學習效果來講，這次專題將許多觀念和知識實際操作一遍，也算為往後學習NLP與其他相關領域的專業，打下一個穩固的基礎。

# 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)