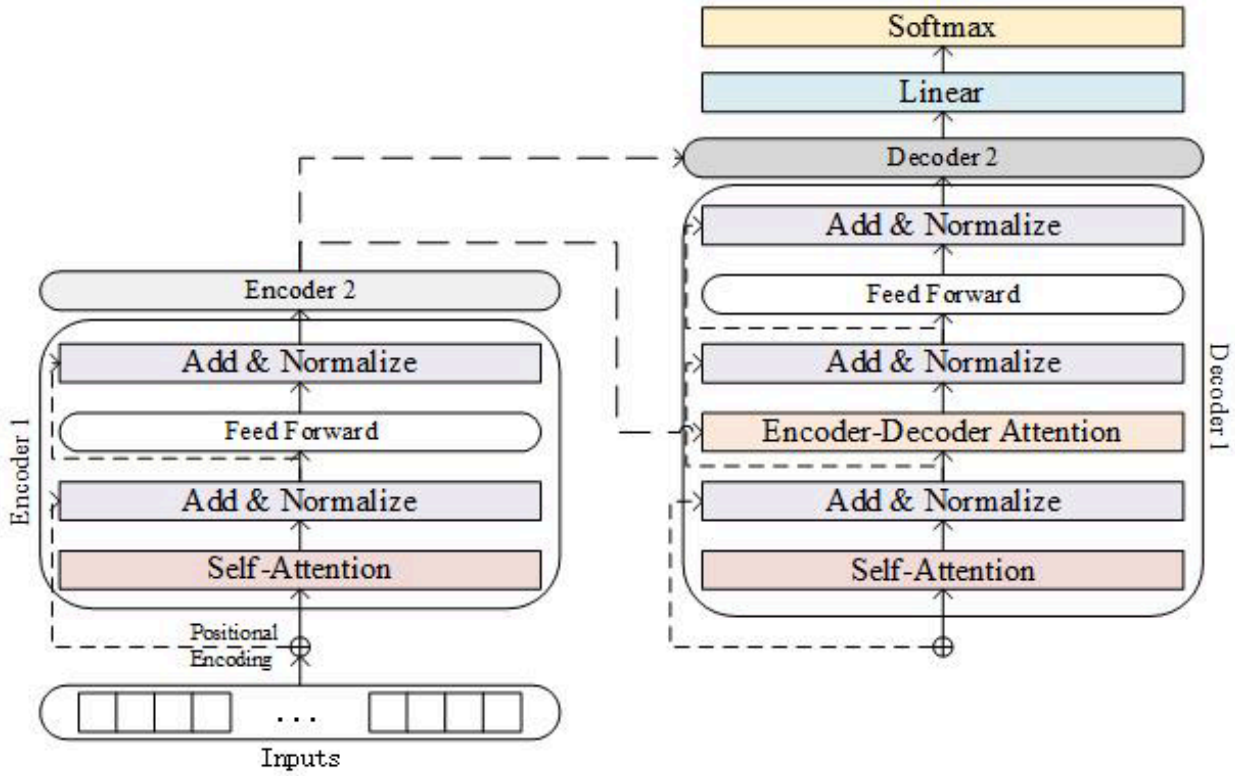
ADL HW2 report

B12902007 林映辰

Q1: Model

1. Model

本作業使用的 pre-trained model 為 google/mt5-small。mt5 是一種 Text-to-Text Transfer Transformer，包含了 encoder 和 decoder。



以上為 T5 架構，圖來自：https://www.researchgate.net/figure/Architecture-of-the-T5-model_fig2_371619795

先把輸入的文字轉換成 tokens，並進行 embedding 轉換成向量，並進行 positional encoding。

Encoder 部分：會經過 multi-layer encoder，每個 layer 包含 self-attention 和 Feed-Forward Network，self-attention 和 Feed-Forward Network 的後面皆會進行 Add & Normalize。

Decoder 部分：會經過 multi-layer decoder，每個 layer 會先用 self-attention 來處理上下文的資訊，再來有 encoder-decoder attention 讓輸出能有原文的資訊，最後有 Feed-Forward Network。self-attention、encoder-decoder attention、Feed-Forward Network 的後面皆會進行 Add & Normalize。

最後，先透過 linear layer 把 decoder 輸出轉換成大小為詞彙表大小的向量，再透過 softmax layer 將向量轉換成對應每個詞彙的機率分布。

在進行 text summarization 時，會先把文字轉換成 tokens 並進行 embedding 變成向量。在 training 時，會輸入 maintext (文章內容) 和 title (文章標題)，讓 model 透過 gradient descent 來最小化 cross-entropy loss，提升模型在 text summarization 的表現。

2. Preprocessing

tokenization: 訓練時會把文章內容和標題分別放在 inputs 和 targets 並進行 tokenization，會對 inputs 和 targets 分別設定 max_source_length (1024) 和 max_target_length (128)，如果長度超過會進行 truncation，長度不足會進行 padding，而且會將 padding token 的 pad_token_id 設為 -100 讓計算 loss 時忽略這些 token。

data split: 因為只有一個 dataset 可拿來訓練，我將 dataset 的 90% 做為 training set，10% 做為 validation set。

Q2: Training

1. Hyperparameter

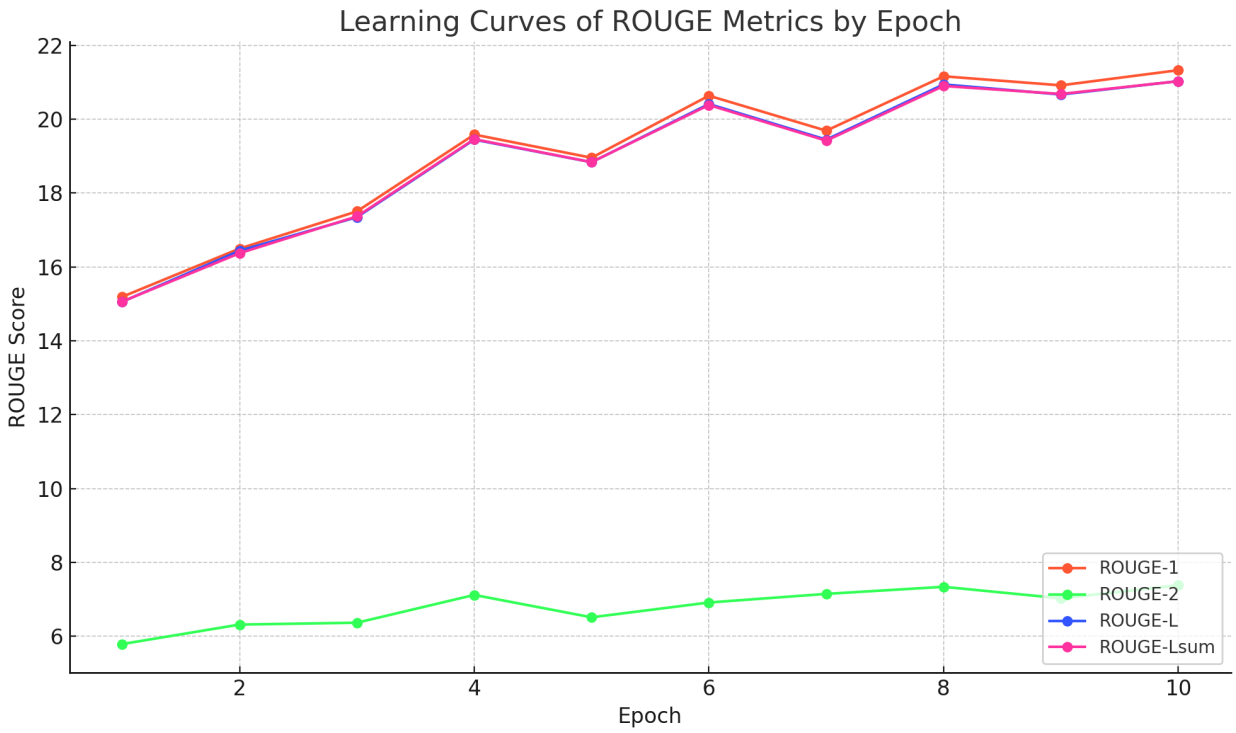
以下是我訓練時的 hyperparameter

```
CUDA_VISIBLE_DEVICES=0 python train.py \
  --model_name_or_path "google/mt5-small" \
  --tokenizer_name "google/mt5-small" \
  --train_file "data/train.jsonl" \
  --output_dir "./output" \
  --num_train_epochs 10 \
  --learning_rate 1e-3 \
  --per_device_train_batch_size 4 \
  --gradient_accumulation_steps 2 \
  --max_source_length 1024 \
  --max_target_length 128 \
  --num_beams 4 \
  --with_tracking \
  --seed 1 \
```

一開始我使用了較小的 num_train_epochs (3) 和較小的 learning rate (1e-5)，發現模型的表現沒有很好，表現提升的速度也較慢。所以後來我選擇了 num_train_epochs = 10, learning_rate = 1e-3 提升模型穩定性與表現，我也嘗試過更大的 num_train_epochs 像是 15 或 20，但發現還是在 num_train_epochs = 10 時表現較好。也設了較高的 per_device_train_batch_size (4), max_source_length (1024), max_target_length (128) 來降低 GPU 的使用。

2. Learning Curves

我使用 epoch 做為橫軸來做圖，每個 epoch 有 2443 步。我是直接讓模型訓練時輸出包含每個 epoch 的表現的 json 檔，再進行繪圖。



Q3: Generation Strategies

1. Strategies

輸出為一個機率分布，有以下幾種方法可選擇下一個字為何

Greedy: 每一步都選擇機率最高的字，但有可能會因為 local optimum 忽略 global optimum。

Beam Search: 每個步驟保留機率最大的 k 條 path，k 為 beam size，以增加找到 global optimum 的可能性，最後選擇保留的 path 中機率最高的。

Top-k Sampling: 只從機率分佈前 k 可能的字進行 sampling。但是若機率分佈過於分散或集中的話可能會造成問題。

Top-p Sampling: 從機率高到低，將字加入 subset，直到 subset 的機率加起來超過 p 則停止繼續加入。每次 sampling 只從這個 subset 中的字進行選擇。

Temperature: 設 temperature 為 T，將用於模型輸出機率分佈的 softmax 函數改成 $P(x_i) = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$，所以 T 越高機率分布會越 uniform，生成內容的多樣性會越高，T 越低機率分佈則會越集中。

2. Hyperparameters

Greedy

使用的 hyperparameters：max_length = 128, num_beams = 1, do_sample = False

表現：


```
{
  "rouge-1": {
    "r": 0.23762695498953199,
    "p": 0.2755915552845401,
    "f": 0.24875083992274233
  },
  "rouge-2": {
    "r": 0.08918196254894578,
    "p": 0.09917962923544067,
    "f": 0.09142800293260947
  },
  "rouge-l": {
    "r": 0.21310851024722877,
    "p": 0.24731431539066687,
    "f": 0.22303460465365516
  }
}
```


Beam Search

(1) 使用的 hyperparameters：max_length = 128, num_beams = 4, do_sample = False

表現：


```
{
  "rouge-1": {
    "r": 0.25774890715539994,
    "p": 0.2850609043750727,
    "f": 0.2640797087610001
  },
  "rouge-2": {
    "r": 0.10169665930549254,
    "p": 0.11066568966422302,
    "f": 0.1032603947019046
  },
  "rouge-l": {
    "r": 0.2306981709652734,
    "p": 0.25548750857641506,
    "f": 0.23641413212465523
  }
}
```


(2) 使用的 hyperparameters：max_length = 128, num_beams = 8, do_sample = False

表現：


```
{
  "rouge-1": {
    "r": 0.2594171374394943,
    "p": 0.2844871012215363,
    "f": 0.26470098668271946
  },
  "rouge-2": {
    "r": 0.10351701764517206,
    "p": 0.11188705854059694,
    "f": 0.10471860836204569
  },
  "rouge-l": {
    "r": 0.23209792378310473,
    "p": 0.25491636602308576,
    "f": 0.2368733858596052
  }
}
```


Top-k Sampling

(1) 使用的 hyperparameters：max_length = 128, top_k = 100 do_sample = True

表現：


```
{
  "rouge-1": {
    "r": 0.20430907549848018,
    "p": 0.21555784634465314,
    "f": 0.20467358069650066
  },
  "rouge-2": {
    "r": 0.06829959485128059,
    "p": 0.07064005024079789,
    "f": 0.06765499502606141
  },
  "rouge-l": {
    "r": 0.18056749465691097,
    "p": 0.19042630270979727,
    "f": 0.1807813649875297
  }
}
```


(2) 使用的 hyperparameters：max_length = 128, top_k = 50, do_sample = True

表現：


```
{
  "rouge-1": {
    "r": 0.21026062383995078,
    "p": 0.22228792156770072,
    "f": 0.2104757482374338
  },
  "rouge-2": {
    "r": 0.07161502003326231,
    "p": 0.07335613991032904,
    "f": 0.07037223077201576
  },
  "rouge-l": {
    "r": 0.18689861592811527,
    "p": 0.1978346378418078,
    "f": 0.1870687902387409
  }
}
```


Top-p Sampling

(1) 使用的 hyperparameters：max_length = 128, top_p = 0.9, do_sample = True

表現：


```
{
  "rouge-1": {
    "r": 0.22032315735221167,
    "p": 0.23833493871805475,
    "f": 0.2235205197816259
  },
  "rouge-2": {
    "r": 0.07765764864542384,
    "p": 0.08209306977738891,
    "f": 0.07773833456684562
  },
  "rouge-l": {
    "r": 0.19610155034700472,
    "p": 0.21210845476436901,
    "f": 0.19886207311132012
  }
}
```


(2) 使用的 hyperparameters：max_length = 128, top_p = 0.85, do_sample = True

表現：


```
{
  "rouge-1": {
    "r": 0.21931630592596396,
    "p": 0.23802396292759204,
    "f": 0.22262781832847853
  },
  "rouge-2": {
    "r": 0.07675608108252299,
    "p": 0.08122053661168323,
    "f": 0.07677130762724649
  },
  "rouge-l": {
    "r": 0.19439734088484545,
    "p": 0.21097226629612206,
    "f": 0.1972130742947298
  }
}
```


Temperature

(1) 使用的 hyperparameters：max_length = 128, temperature = 0.9, do_sample = True

表現：


```
{
  "rouge-1": {
    "r": 0.21810511507053212,
    "p": 0.2343354518774601,
    "f": 0.22025950830368707
  },
  "rouge-2": {
    "r": 0.07626956574418119,
    "p": 0.07960238534076713,
    "f": 0.07583035040577299
  },
  "rouge-l": {
    "r": 0.19419451388548664,
    "p": 0.20883895440541114,
    "f": 0.19610403296824355
  }
}
```


(2) 使用的 hyperparameters：max_length = 128, temperature = 0.8, do_sample = True

表現：

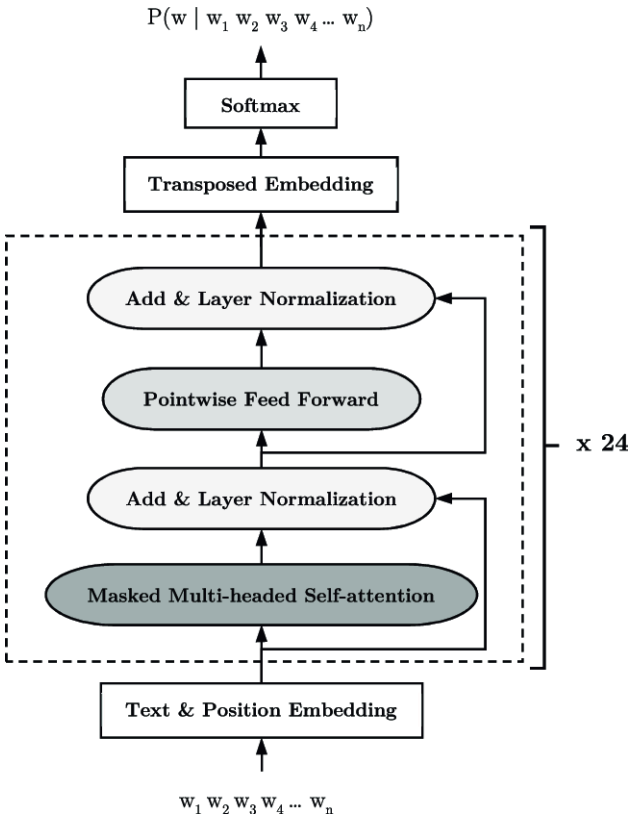

```
{
  "rouge-1": {
    "r": 0.22108682801508248,
    "p": 0.24297243624428652,
    "f": 0.22598201595394973
  },
  "rouge-2": {
    "r": 0.07782997059703141,
    "p": 0.0835226991935143,
    "f": 0.07852701186713908
  },
  "rouge-l": {
    "r": 0.19640441408513104,
    "p": 0.21598514446703643,
    "f": 0.20069236808680763
  }
}
```


評估了各個策略的表現，我最後選擇的策略為 Beam Search，使用的 hyperparameters 為：max_length = 128, num_beams = 8, do_sample = False，最後的表現為：


```
{
  "rouge-1": {
    "r": 0.2594171374394943,
    "p": 0.2844871012215363,
    "f": 0.26470098668271946
  },
  "rouge-2": {
    "r": 0.10351701764517206,
    "p": 0.11188705854059694,
    "f": 0.10471860836204569
  },
  "rouge-l": {
    "r": 0.23209792378310473,
    "p": 0.25491636602308576,
    "f": 0.2368733858596052
  }
}
```


Bonus: Applied GPT-2 on Summarization

1. Model



以上為 GPT-2 架構，圖來自：https://www.researchgate.net/figure/Structure-of-the-applied-GPT-2-medium-architecture_fig2_365625866

先把輸入的文字轉換成 tokens，並進行 embedding 轉換成向量，並進行 positional embeddings。

接著，會經過 multi-layer decoder，每一層中都包含 masked multi-head self-attention 和 Feed-Forward Network，masked multi-head self-attention 和 Feed-Forward Network 後面皆會進行 Add & Normalize。

最後，先透過 transposed embedding 把 decoder 輸出轉換成大小為詞彙表大小的向量，再透過 softmax layer 將向量轉換成對應每個詞彙的機率分布。

我預計使用與前面相似的 hyperparameters 來訓練：

```
CUDA_VISIBLE_DEVICES=0 python train.py \
  --model_name_or_path "openai-community/gpt2" \
  --tokenizer_name "openai-community/gpt2" \
  --train_file "data/train.jsonl" \
  --output_dir "./output" \
  --num_train_epochs 10 \
  --learning_rate 1e-3 \
  --per_device_train_batch_size 4 \
  --gradient_accumulation_steps 2 \
  --max_source_length 1024 \
  --max_target_length 128 \
  --num_beams 4 \
  --with_tracking \
```

```
    --seed 1
```

但最後沒有訓練出可有效用於 text summarization 的模型。