Applied Deep Learning Homework 3 Report

R10922123 周昱豪

Q1: Model

Model configuration


```
{
  "_name_or_path": "google/mt5-small",
  "architectures": [
    "MT5ForConditionalGeneration"
  ],
  "d_ff": 1024,
  "d_kv": 64,
  "d_model": 512,
  "decoder_start_token_id": 0,
  "dropout_rate": 0.1,
  "eos_token_id": 1,
  "feed_forward_proj": "gated-gelu",
  "initializer_factor": 1.0,
  "is_encoder_decoder": true,
  "layer_norm_epsilon": 1e-06,
  "model_type": "mt5",
  "num_decoder_layers": 8,
  "num_heads": 6,
  "num_layers": 8,
  "pad_token_id": 0,
  "relative_attention_max_distance": 128,
  "relative_attention_num_buckets": 32,
  "tie_word_embeddings": false,
  "tokenizer_class": "T5Tokenizer",
  "torch_dtype": "float32",
  "transformers_version": "4.19.0.dev0",
  "use_cache": true,
  "vocab_size": 250100
}
```


- Encoder-decoder architecture，把NLP problem轉為text-to-text format.
- Input sequence丟進去encoder後會output一個帶有語義的vector，之後再餵進去decoder，那會把label向右移一個單位一起餵進去decoder來算出loss。

- Decoder端的token要一個一個餵進去，然後第一個token是START當作是generate出來的開頭，之後會output一個機率分布，機率最高的token再餵進去decoder，產生下一個token，以此類推，最後generate END這個token就結束。

Preprocessing

- Tokenization: SentencePiece
  - Unsupervised text tokenizer，事先指定vocabulary的大小。
  - 可以直接在原本的句子上訓練，這樣對於不同tokenizer就不會有依賴性。
  - 會把sequence看作是unicode sequence，這樣whitespace就會視為基礎符號而不是特殊符號，這樣就可逆。
  - 有implement subword units，像是byte-pair-encoding (BPE) 或是 unigram language model。

Q2: Training

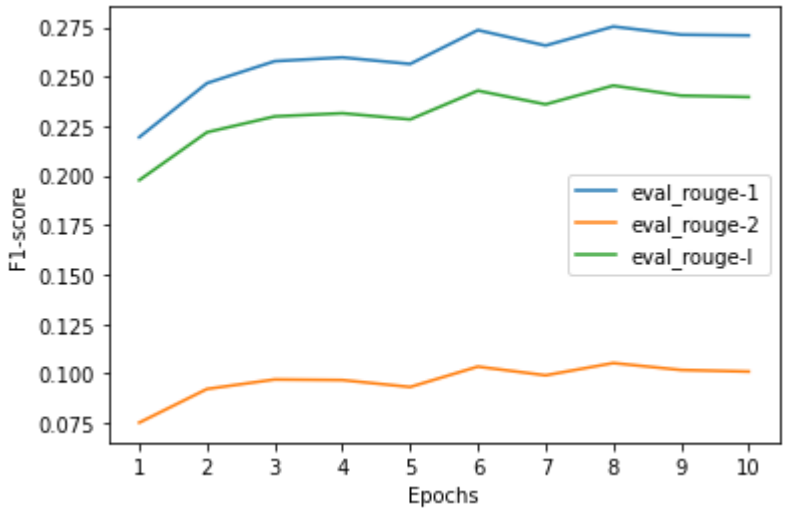
Hyperparameter

- Training batch size設為 `16` ，使用A5000訓練，有24GB的記憶體空間，因為batch size越大效果會有些許的提升，並且也有使用adafactor來降低GPU記憶體的使用量。
- Learning rate設為 `5e-4` ，原本是使用AdamW with learning rate=3e-5，之後調整為adafactor後發現效果沒有太好，所以有調過[1e-5, 1e-4, 5e-4]，最後發現5e-4效果較好。
- Training epoch設為20，原本為5，與同學討論過後發現訓練久一點效果會變好，所以就直接設20，結果確實好不少。

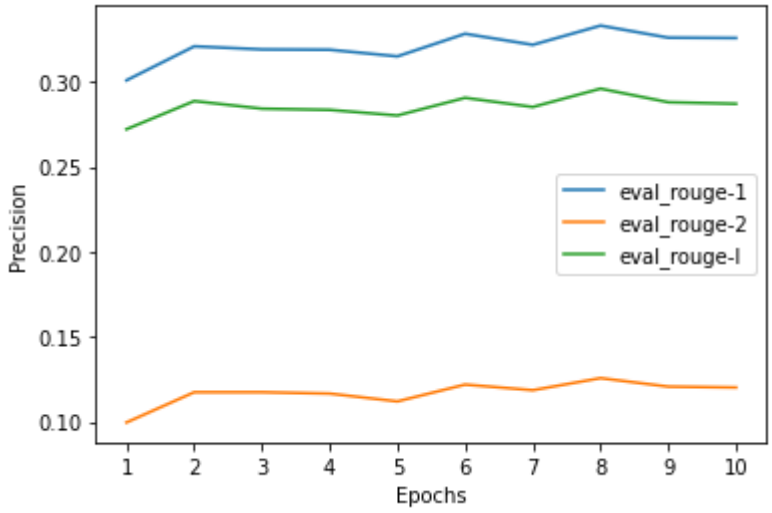
Learning Curves

- 這裡只畫出10個epoch即可表現出他的趨勢。

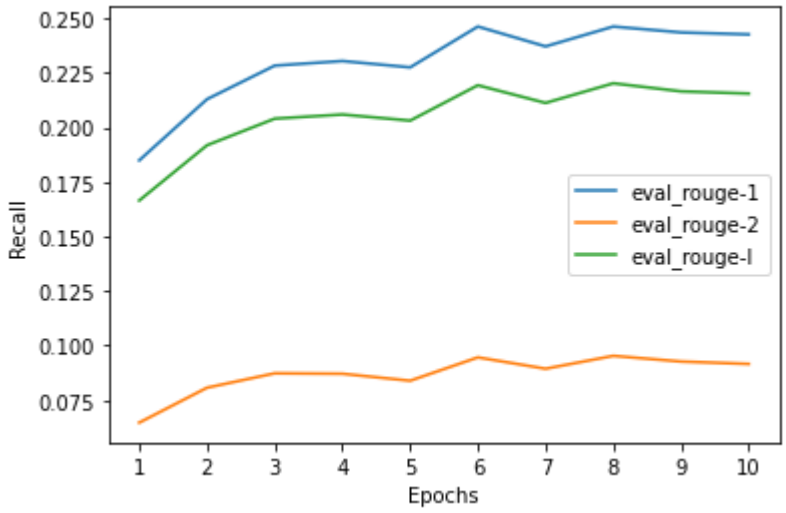
F1-score



Precision



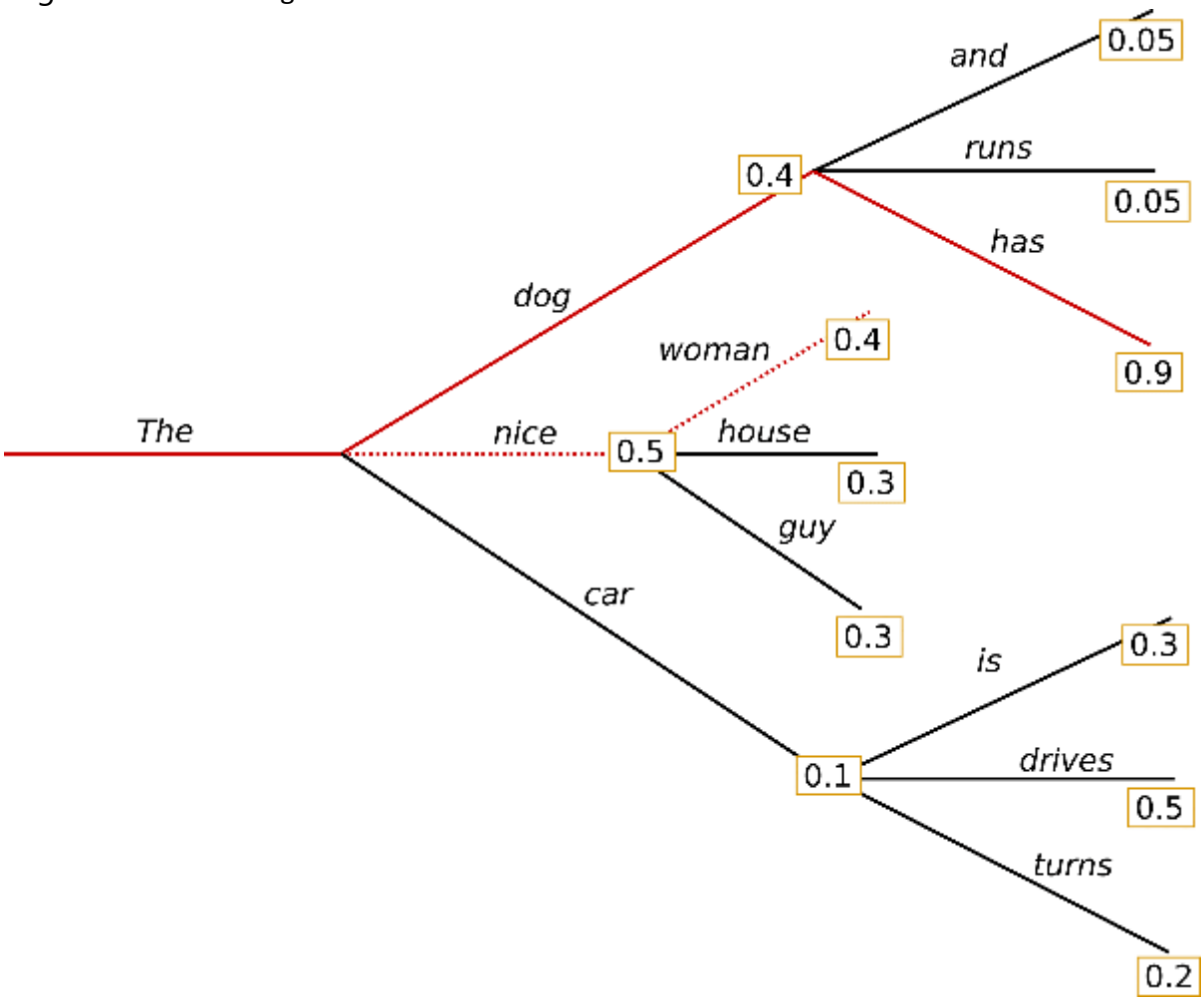
Recall



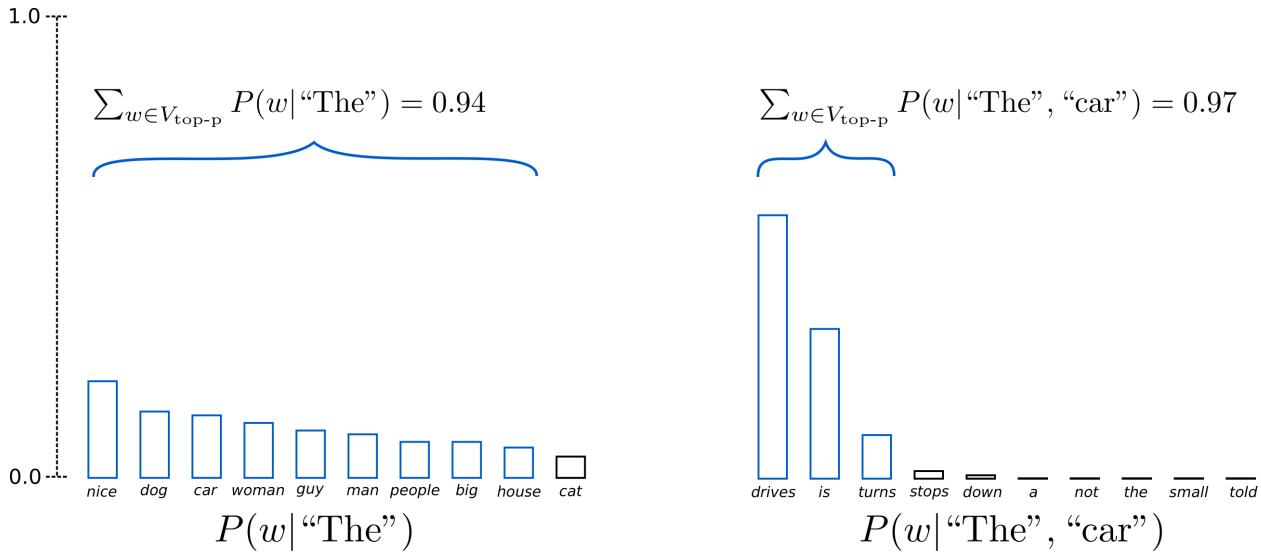
Q3: Generation Strategies

Stratgies

- Greedy
  - 很直觀的從當前word裡面挑選機率最高的當作next word。
  - 缺點就是會忽略後面可能會出現較高機率的word。
- Beam Search
  - Beam search就是來改善Greedy的作法，他會往後樹狀的去看更後面word的機率，並把整條path的機率乘起來，取機率最高的path，以下為截自huggingface官網num_beams=2的例子。
  - `The dog has` 這條path乘起來機率是0.36，比 `The nice woman` 的0.2還要高，所以會選擇generate `The dog has` 。



- Top-k Sampling
  - 簡單來說就是只取前K個最高機率的word，來決定誰要被generate。
  - 但這樣有個問題就是K不是動態調整的，假設機率分布較平坦，K會去限制模型的發展。
- Top-p Sampling
  - 他會取前n個機率總和大於p的word來sample，以下圖huggingface例子(p=0.92)說明。
  - 左圖為sample 9個word來達到累增機率>p的這個條件，而右圖只需要取3個word就可以讓累增機率>p了，就可以時間top-k做不到的動態調整sample word的數量。



- Temperature
  - 它會讓原本的機率分布更sharp一點，舉huggingface的例子，假設下一個word的機率分布原本是[0.5, 0.4, 0.1]，設定 `temperature=0.7` 後會變成[0.75, 0.23, 0.02]，它會讓原本機率高的更高，機率低的會變低，這樣我們就更不會去generate機率較低的word。

Hyperparameters

- Greedy v.s. Beam search (num_beams = 5)

metrics	Greedy	Beam search
predict_rouge-1_f	0.2524	0.2659
predict_rouge-1_p	0.3022	0.298
predict_rouge-1_r	0.2327	0.2576
predict_rouge-2_f	0.0934	0.1063
predict_rouge-2_p	0.1086	0.1172
predict_rouge-2_r	0.0881	0.1046
predict_rouge-l_f	0.2248	0.2361
predict_rouge-l_p	0.2698	0.265
predict_rouge-l_r	0.2073	0.2288

- Beam search (num_beams = 5 v.s. num_beams = 10)

metrics	num_beams=5	num_beams=10
predict_rouge-1_f	0.2659	0.2634
predict_rouge-1_p	0.298	0.2904
predict_rouge-1_r	0.2576	0.2585
predict_rouge-2_f	0.1063	0.1061
predict_rouge-2_p	0.1172	0.1154
predict_rouge-2_r	0.1046	0.1054
predict_rouge-l_f	0.2361	0.2342
predict_rouge-l_p	0.265	0.2589
predict_rouge-l_r	0.2288	0.2298

- Top-k sampling

metrics	k=50	k=100
predict_rouge-1_f	0.2002	0.1836
predict_rouge-1_p	0.2177	0.1986
predict_rouge-1_r	0.1981	0.182
predict_rouge-2_f	0.0603	0.0561
predict_rouge-2_p	0.0641	0.0596
predict_rouge-2_r	0.0617	0.057
predict_rouge-l_f	0.1749	0.1608
predict_rouge-l_p	0.1903	0.1742
predict_rouge-l_r	0.1733	0.1595

- Top-p sampling

metrics	p=0.85	p=0.92
predict_rouge-1_f	0.1728	0.165
predict_rouge-1_p	0.1864	0.1764
predict_rouge-1_r	0.1714	0.1656
predict_rouge-2_f	0.0525	0.0479
predict_rouge-2_p	0.0563	0.0506
predict_rouge-2_r	0.0532	0.0491
predict_rouge-l_f	0.1523	0.1453
predict_rouge-l_p	0.1645	0.1553
predict_rouge-l_r	0.1512	0.1461

- Temperature

metrics	Temperature=0.3	Temperature=0.7
predict_rouge-1_f	0.2496	0.2172
predict_rouge-1_p	0.296	0.244
predict_rouge-1_r	0.2316	0.2082
predict_rouge-2_f	0.0914	0.0721
predict_rouge-2_p	0.1052	0.0794
predict_rouge-2_r	0.0869	0.0709
predict_rouge-l_f	0.2215	0.1912
predict_rouge-l_p	0.2632	0.2148
predict_rouge-l_r	0.2055	0.1835

- My final generation strategy is Beam search with num_beams=5，看起來效果最好，且num_beams=10效果反而下降了一些，prediction時間也拉長了一點。