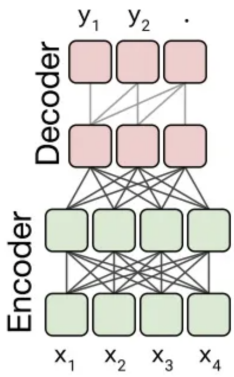
112-1 ADL HW2 Report

`b10902138 陳德維`

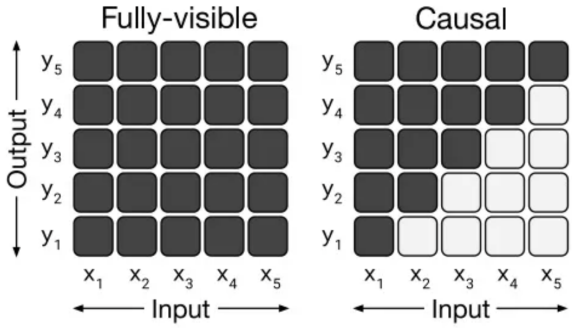
Q1: Model

- Model

- **Model Name:**
  - `mt5-small` - Multilingual Text-to-Text Transfer Transformer
- **Model Architecture:**
  - `Encode-Decoder style Transformer`



    - **Encoder layer** (6 layers): with *Fully-visible* masking (Dark grey line)
    - **Decoder layer** (6 layers): with Encoder-decoder attention and *casual* masking (Light grey line)



    - Encoder layer passes its output to the decoder layer
- **How it works on text summarization:**
  - For mt5-small, it is pretrained on a vast amount of multilingual data and perfromed various NLP tasks by using a text-to-text approach. Hence it has the ability to cast the task: `text summarization` (actually for many other NLP tasks) as feeding the model text as input and training it to generate the output text, which is the summary. It adds a prefix to the input to handle different NLP tasks better, adding `summarize:` for text summarization in this case. Then, we tokenize our input text and feed it to the model. Last, we generate the output with some gerneration algorithm and decode it to be our summary. During both pre-training and fine-tuning,

the model learns to generate summaries by understanding the textual content, maintaining coherence, and ensuring the important information is included in the generated summary.

- Preprocessing

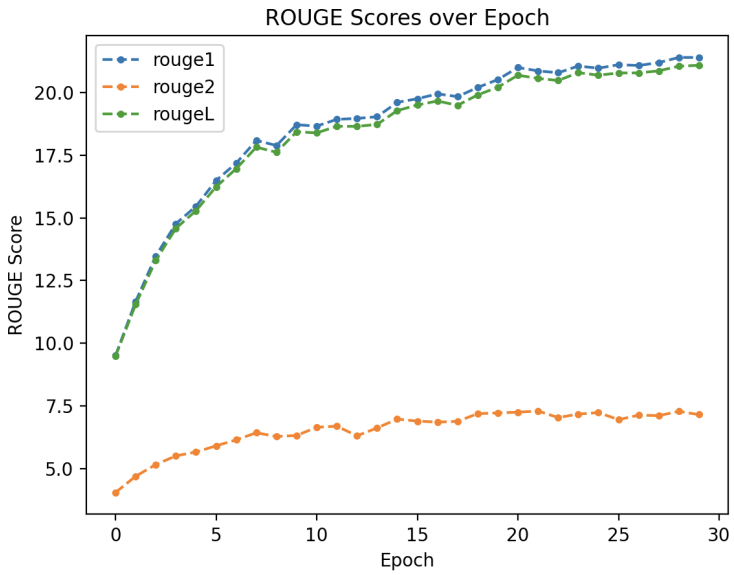
- **Prefix:**
  - We add a prefix `summarize:` to each input text to have better performance on the pre-trained mt5-small model.
- **Tokenization:**
  - I use the `T5Tokenizer` from HuggingFace.
    1. While it's hard to handle pure Chinese input, it maps each symbol to an unique token id.
       - Special tokens are added:
         - `</s>`: **End Token**, marks the end of a segment or document
         - `<pad>`: **Pad Token**, for padding sequences to a fixed length
         - `<unk>`: **Unknown Token**, used to represent unknown or out-of-vocabulary words
  - With this tokenizer, we convert our text input into tokens and prepare it as model inputs.
- **Data Cleaning:**
  - Padding/Truncation: I pad or truncate all input to `max_text_length`, which I use 256 in my fine-tuning.
  - Replace all `<pad>` - **Pad token** in the label to -100 to ignore padding in the loss.

Q2: Training

- **Hyperparameters:**
  - Pre-trained Model: `mt5-small`
    - `seed`: 1006 (A special date to me)
    - `model_name_or_path`: mt5-small (Given from spec)
    - `source_prefix`: "summarize: " (Required for t5-style model)
    - `max_source_length`: 256 (Given from spec)
    - `max_target_length`: 64 (Given from spec)
    - `pad_to_max_length`: True (Seems to be better with)
    - `num_train_epochs`: 30 (After tuning, this has the best performance)
    - `per_device_train_batch_size`: 4 (Limited by my GPU resource)
    - `gradient_accumulation_steps`: 2 (Limited by my GPU resource)
  - Optimizer: `adafactor` (from HuggingFace T5 Finetuning Tips)
    - `learning_rate`: $5e^{-4}$ (After tuning, this has the best performance)
    - `learning_rate_scheduler`: linear (After tuning, this has the best performance)
    - `warm_ratio`: 10% (After tuning, this has the best performance)
    - `clip_threshold`: 1.0
    - `eps`: ($1e^{-30}$, $1e^{-3}$)
    - `beta1`: None
    - `relative_step`: False
    - `scale_parameter`: False
    - `warmup_init`: False
- **Learning Curve:**

- ROGUE from:

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



- Raw Data:

```
Epoch 0: {'rouge1':  9.5045, 'rouge2': 4.0379, 'rougeL':  9.4823, 'rougeLsum':
9.4722}
Epoch 1: {'rouge1': 11.6643, 'rouge2': 4.6873, 'rougeL': 11.5422, 'rougeLsum':
11.5426}
Epoch 2: {'rouge1': 13.4786, 'rouge2': 5.1542, 'rougeL': 13.3016, 'rougeLsum':
13.3438}
Epoch 3: {'rouge1': 14.7673, 'rouge2': 5.5008, 'rougeL': 14.5832, 'rougeLsum':
14.5909}
Epoch 4: {'rouge1':  15.464, 'rouge2': 5.6556, 'rougeL': 15.2879, 'rougeLsum':
15.2821}
Epoch 5: {'rouge1': 16.4945, 'rouge2': 5.9022, 'rougeL': 16.2441, 'rougeLsum':
16.2563}
Epoch 6: {'rouge1': 17.1868, 'rouge2': 6.1409, 'rougeL': 16.9648, 'rougeLsum':
16.9488}
Epoch 7: {'rouge1':  18.094, 'rouge2': 6.4234, 'rougeL': 17.8238, 'rougeLsum':
17.7983}
Epoch 8: {'rouge1':  17.889, 'rouge2': 6.2746, 'rougeL': 17.6199, 'rougeLsum':
17.6108}
Epoch 9: {'rouge1': 18.7207, 'rouge2': 6.3153, 'rougeL': 18.4391, 'rougeLsum':
18.4321}
Epoch 10:{'rouge1': 18.6641, 'rouge2': 6.6442, 'rougeL': 18.3946, 'rougeLsum':
18.3551}
Epoch 11:{'rouge1': 18.9344, 'rouge2': 6.6854, 'rougeL': 18.6566, 'rougeLsum':
18.6237}
Epoch 12:{'rouge1': 18.9714, 'rouge2': 6.3029, 'rougeL': 18.6536, 'rougeLsum':
18.6278}
Epoch 13:{'rouge1': 19.0373, 'rouge2': 6.6148, 'rougeL': 18.7316, 'rougeLsum':
18.7046}
Epoch 14:{'rouge1': 19.6139, 'rouge2': 6.9704, 'rougeL': 19.2818, 'rougeLsum':
19.2517}
Epoch 15:{'rouge1': 19.7571, 'rouge2': 6.8883, 'rougeL': 19.5103, 'rougeLsum':
```

```
19.4882}
Epoch 16:{'rouge1': 19.9511, 'rouge2': 6.8495, 'rougeL': 19.6648, 'rougeLsum':
19.6121}
Epoch 17:{'rouge1':  19.842, 'rouge2': 6.8793, 'rougeL': 19.4958, 'rougeLsum':
19.5102}
Epoch 18:{'rouge1': 20.2066, 'rouge2': 7.1858, 'rougeL': 19.9084, 'rougeLsum':
19.8926}
Epoch 19:{'rouge1': 20.5221, 'rouge2': 7.2156, 'rougeL':  20.218, 'rougeLsum':
20.2073}
Epoch 20:{'rouge1': 21.0032, 'rouge2': 7.2447, 'rougeL': 20.6949, 'rougeLsum':
20.6928}
Epoch 21:{'rouge1': 20.8688, 'rouge2': 7.2921, 'rougeL': 20.5696, 'rougeLsum':
20.5351}
Epoch 22:{'rouge1': 20.7931, 'rouge2': 7.0298, 'rougeL': 20.4835, 'rougeLsum':
20.4419}
Epoch 23:{'rouge1': 21.0616, 'rouge2': 7.1692, 'rougeL': 20.7995, 'rougeLsum':
20.7727}
Epoch 24:{'rouge1': 20.9757, 'rouge2': 7.2296, 'rougeL': 20.6961, 'rougeLsum':
20.6677}
Epoch 25:{'rouge1': 21.1148, 'rouge2': 6.9473, 'rougeL': 20.7864, 'rougeLsum':
20.7693}
Epoch 26:{'rouge1': 21.0842, 'rouge2': 7.1251, 'rougeL': 20.7885, 'rougeLsum':
20.7564}
Epoch 27:{'rouge1': 21.1995, 'rouge2': 7.1088, 'rougeL': 20.8703, 'rougeLsum':
20.8554}
Epoch 28:{'rouge1': 21.4055, 'rouge2': 7.2766, 'rougeL': 21.0571, 'rougeLsum':
21.037}
Epoch 29:{'rouge1': 21.4052, 'rouge2': 7.1478, 'rougeL': 21.0904, 'rougeLsum':
21.0505}
```

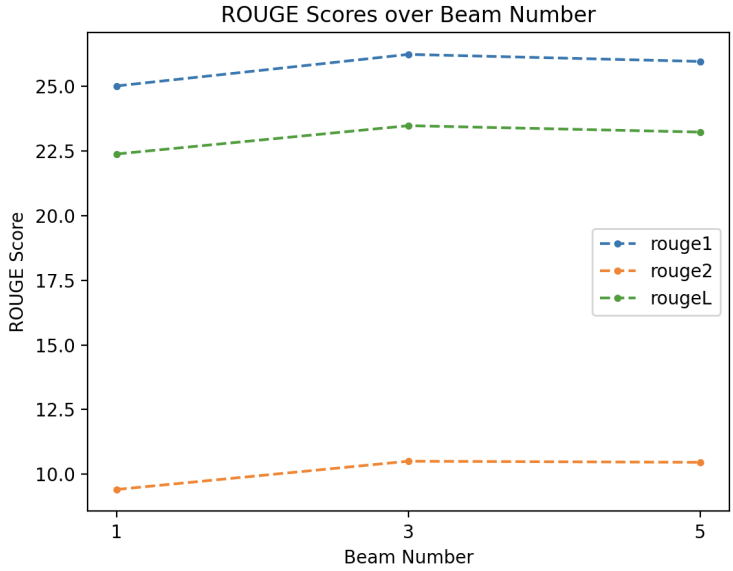
Q3: Generation Strategy

- **Strategy:**
  - Greedy
    - The model selects the token with highest score at each step, resulting in a local optimal choice.
  - Beam Search
    - A trimmed searching algorithm extended from greedy, instead of choosing one token at each step, specifying `beam number = n`, it tracks the top `n` highest score token for each step and proceed with them, ending with `n` predicted outputs and keeps the most promising one.
  - Top-k Sampling
    - Randomly samples from the top `k` highest score at each step. Randomness gives a more diversity predicted answer. Smaller `k` results in more constrained and conservative sampling (k=1 is greedy), while larger `k` gives more diverse and creative outputs.
  - Top-p Sampling
    - A.k.a. Nucleus sampling, samples from the smallest set of tokens whose cumulative probability exceeds a threshold `p`. Smaller `p` results in more constrained and conservative sampling, while larger `p` gives more diverse and creative outputs.
  - Temperature
    - Applying a temperature hyperparameter `τ` to the softmax for computing the probability distribution.

$$P(w) = \frac{exp(s_{w/\tau})}{\sum_{w' \in V} exp(s_{w'}/\tau)}$$

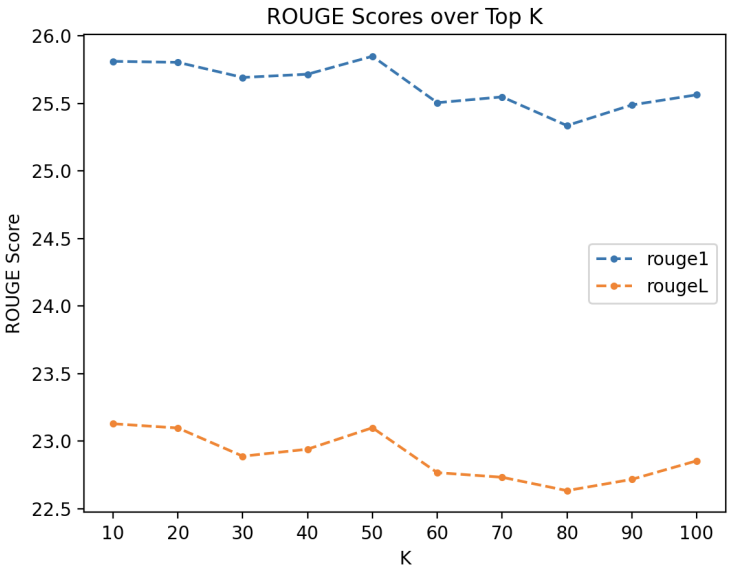
    - Smaller `τ` results in more constrained and conservative sampling, while larger `τ` gives more diverse and creative outputs.

- **Hyperparameters:**
    - **ROGUE** from `tw_rogue`'s `f1-score` * 100
    - **General setting**:
        - `pad_to_max_length`: True
        - `max_source_length`: 256
        - `max_target_length`: 64
    - **Strategy**:
        - *Greedy* & *Beam Search*:
            - Extra Config:
                - `num_beams`: 1~3
            - Result:



            - Raw Data:
                - n=1: `{"rouge-1": 25.023210145212016, "rouge-2": 9.409571998321532, "rouge-l": 22.3900892934053}`
                - n=3: `{"rouge-1": 26.243396505367633, "rouge-2": 10.502798507636231, "rouge-l": 23.485434720915918}`
                - n=5: `{"rouge-1": 25.97040680259727, "rouge-2": 10.459695369062331, "rouge-l": 23.237683138272694}`
        - *Top-k Sampling*:
            - Extra Config:
                - `top_k`: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100
                - `num_beams`: 3
                - `top_p`: 1
                - `temperature`: 1
                - `do_sample`: True

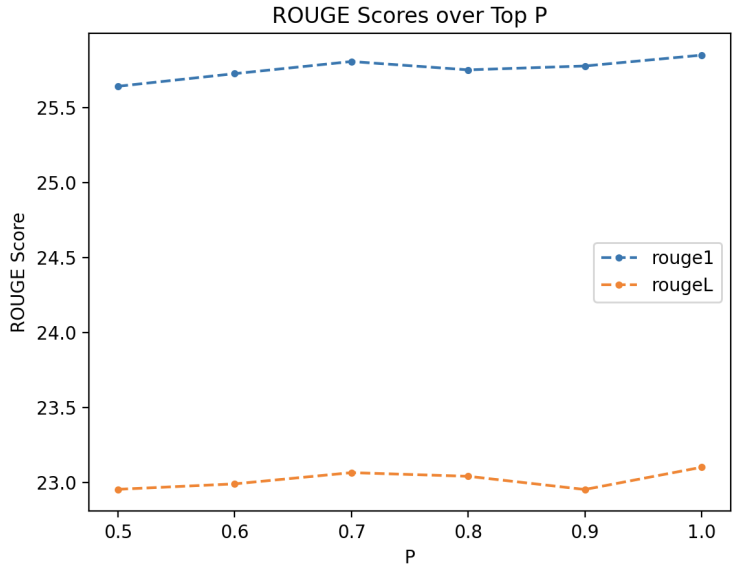
- Result: (Didn't show `rogue2` to emphasize the changes of rouge1, rougeL)



- Raw data:
  - n=10: `{"rouge-1": 25.810834735583672, "rouge-2": 10.131189974874824, "rouge-l": 23.129143201185553}`
  - n=20: `{"rouge-1": 25.803207749160688, "rouge-2": 10.145129606249661, "rouge-l": 23.097810687093833}`
  - n=30: `{"rouge-1": 25.691518968050183, "rouge-2": 10.000808087204126, "rouge-l": 22.88922565049978}`
  - n=40: `{"rouge-1": 25.71556389615391, "rouge-2": 10.164527205635933, "rouge-l": 22.94106571094801}`
  - n=50: `{"rouge-1": 25.848126335026034, "rouge-2": 10.182116944673767, "rouge-l": 23.099687450121884}`
  - n=60: `{"rouge-1": 25.50465577322006, "rouge-2": 10.003358580651666, "rouge-l": 22.7675642320746}`
  - n=70: `{"rouge-1": 25.547577317067756, "rouge-2": 9.976650208815688, "rouge-l": 22.73339165904095}`
  - n=80: `{"rouge-1": 25.335353740891055, "rouge-2": 9.820889181962116, "rouge-l": 22.634564222102927}`
  - n=90: `{"rouge-1": 25.488003813772536, "rouge-2": 10.033591409115246, "rouge-l": 22.717668256176644}`
  - n=100: `{"rouge-1": 25.56271294643664, "rouge-2": 9.9859919053637, "rouge-l": 22.854821679876707}`

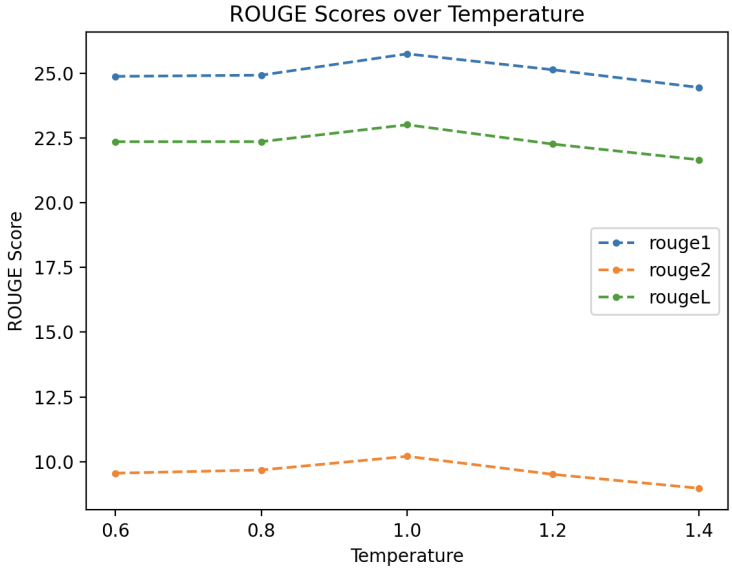
- *Top-p Sampling*:
  - Extra Config:
    - `top_p` : 0.5, 0.6, 0.7, 0.8, 0.9, 1.0
    - `top_k` : 50
    - `num_beams` : 3
    - `temperature` : 1
    - `do_sample` : True

- Result: (Didn't show `rogue2` to emphasize the changes of rouge1, rougeL)



  - Raw data:
    - p=0.5: `{"rouge-1": 25.64058386033518, "rouge-2": 9.951699946868612, "rouge-l": 22.952634759313103}`
    - p=0.6: `{"rouge-1": 25.724795312919564, "rouge-2": 10.037950969102132, "rouge-l": 22.988950157740966}`
    - p=0.7: `{"rouge-1": 25.805641846531863, "rouge-2": 10.09631675166291, "rouge-l": 23.063791149482785}`
    - p=0.8: `{"rouge-1": 25.750055219617657, "rouge-2": 10.116143826965558, "rouge-l": 23.039495082891822}`
    - p=0.9: `{"rouge-1": 25.775856165151882, "rouge-2": 10.118151704674164, "rouge-l": 22.951460128080743}`
    - p=1.0: `{"rouge-1": 25.848126335026034, "rouge-2": 10.182116944673767, "rouge-l": 23.099687450121884}`
- *Temperature*:
  - Extra Config:
    - `temperature` : 0.6, 0.8, 1.0, 1.2, 1.4
    - `top_k` : 50
    - `num_beams` : 3
    - `top_p` : 1
    - `do_sample` : True

- Result:



  - Raw data:
    - temp=0.6: `{"rouge-1": 24.88914162890027, "rouge-2": 9.550457424257882, "rouge-l": 22.363428156578816}`
    - temp=0.8: `{"rouge-1": 24.935094084727165, "rouge-2": 9.673700474310177, "rouge-l": 22.365453134088156}`
    - temp=1.0: `{"rouge-1": 25.759052530759654, "rouge-2": 10.201863280548263, "rouge-l": 23.01884107674031}`
    - temp=1.2: `{"rouge-1": 25.14390633663046, "rouge-2": 9.50437504336373400, "rouge-l": 22.269738920677327}`
    - temp=1.4: `{"rouge-1": 24.460186191243473, "rouge-2": 8.969869027698502, "rouge-l": 21.66578079860883}`

- **My final generation strategy**:
  - As we can see, using ***Beam Search*** with `beam number = 3` has the highest score `{"rouge-1": 26.243396505367633, "rouge-2": 10.502798507636231, "rouge-l": 23.485434720915918}`.