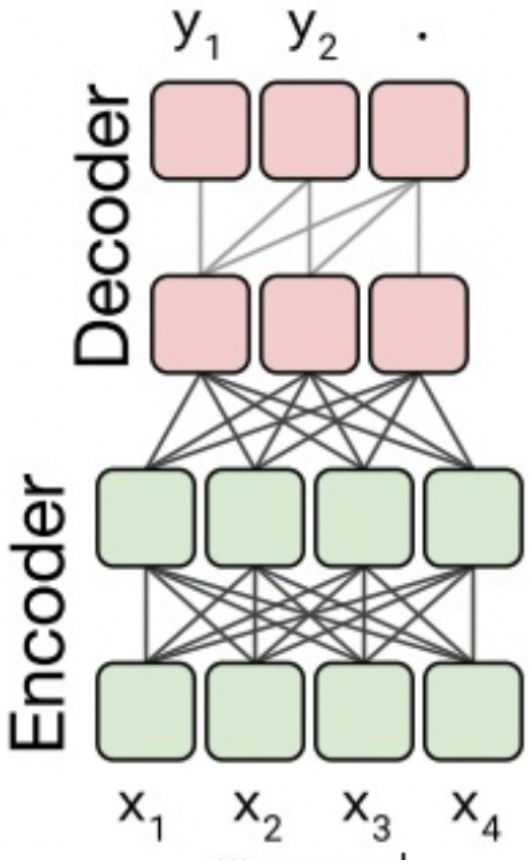
*tags: **ADL***

HW3

Q1

Model



The model is an encoder-decoder Transformer.
The decoder takes encoder's output and then generates sequence with different length of the inputs' length. Then we can use the probs from the decoder output to adopt some strategies to decode the title.

Preprocessing

First get rid of some special characters

```
article['maintext'].strip().replace('\n', '').\
replace('\r', '').replace(' ', '')\
.replace('`', '')
```

Then we tokenize maintext and title, with `max_length=1024, 128` , respectively.

```
def preprocess_function(examples):
        # print('43', type(examples))
    inputs = examples["maintext"]
    model_inputs = tokenizer(inputs, max_length=args.max_plen,
                             truncation=True, padding='max_length')

    with tokenizer.as_target_tokenizer():
        labels = tokenizer(examples["title"], max_length=args.max_qlen,
                           truncation=True, padding='max_length')

    model_inputs["labels"] = labels["input_ids"]
    model_inputs['aid'] = examples['id']
    # print('51', model_inputs)
    return model_inputs
```

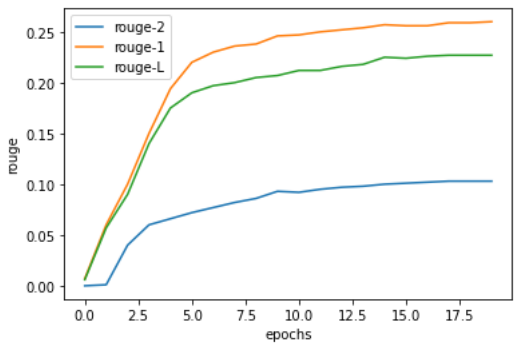
Q2

hyperparams

```
lr = 3e-5
batch_size = 2
gradient_accumulation_steps = 2
max_input_length = 1024
max_target_length = 64
20 epochs
```

Since we're training from a pretrained model, so smaller lr is better, and batch_size=2 is too small, so we use gradient accumulation to achieve effective batch size of 4.

Learning curves



Q3

Stratgies

GREEDY:

simply use the token with greatest probability

BEAM:

keep track of the `beam_size`– most probable sequence to decode the result

TOP_K:

sample the word from the distribution restricted to the top-k probable words

TOP_P:

sample a subset from the vocabulary with the most probability mass

$$w_i \sim V^{(p)}$$
$$\text{where}$$
$$V^{(p)} = \sup_{V' \subset V} \sum_{x \in V'} P(x|w_1 \cdots w_{i-1}) \geq p$$

TEMPERATURE:

Controls how diverse the distribution is, higher temperature leads to more uniform result, and thus more diverse

$$P(w_t) = \frac{e^{s_w/\tau}}{\sum_{w' \in V} e^{s_{w'}/\tau}}$$

Hyperparameters

method	rouge-1	rouge-2	rouge-L
greedy	0.2503	0.0902	0.02219
beam(size=5)	0.2627	0.1030	0.2302
beam(size=10)	0.2626	0.1049	0.2311
temperature=0.7	0.2136	0.0721	0.1890
temperature=0.3	0.2454	0.088	0.2184
top_k(k=25)	0.200	0.0619	0.1766
top_k(k=50)	0.1918	0.0597	0.1691
top_p(p=0.92)	0.1676	0.051	0.1490
top_p(p=0.98)	0.1568	0.0471	0.1400

Extremely high value of p hinders the performance, and `beam_size` of `10` is slightly better but takes a lot of time. With `temp=0.3`, which leads to sharper distribution, performes better than `temp=0.7`

I choose `beam_size=5` as my final decision.

RL

Algorithm

$$L_{rl} = (r(\hat{y}) - r(y^s)) \sum_{t=1}^{n} \log p(y_t^s | y_1^s, \ldots, y_{t-1}^s, x) \quad (15)$$

We can see that minimizing L_{rl} is equivalent to maximizing the conditional likelihood of the sampled sequence y^s if it obtains a higher reward than the baseline $\hat{y}$, thus increasing the reward expectation of our model.

Where $\hat{y}$ is the baseline output, which uses the word with max probability, and y^s is the output sampled from the probability distribution. So if some sampled word performes better than baseline, by minimizing the L_{rl}, the probability of the sampled sequence will increase

Reward function

$rouge1 + rouge2 + rougeL$

hyperparam

```
lr = 1e-5
batch_size = 2
gradient_accumulation_steps = 2
max_input_length = 1024
max_target_length = 64
15 epochs(fine tune on supervised learning checkpoint)
```

Compare to Supervised Learning

The loss from the original supervised learning increases, and L_{rl} also increases with the reward, this may mean that the baseline is becoming stronger.
And rouge score improves slightly.
RL: 0.2705, 0.1050, 0.2372

ref:

https://arxiv.org/pdf/1705.04304.pdf

(https://arxiv.org/pdf/1705.04304.pdf)

https://github.com/rohithreddy024/Text-Summarizer-Pytorch (https://github.com/rohithreddy024/Text-Summarizer-Pytorch)