Text Summarization part 3

NLG Intro

applications: Document Summarization, E-mail Summarization
Meeting Summarization
Write story, narratives, poetry

NLG: Any task involving text production for human consumption requires natural language generation

how to generate language?

at each time step t, our model computes a vector of scores for each token in our vocabulary, $S \in \mathbb{R}^{|V|}$

$S = f(\{y_{<t}\})$ $f(\ )$ is model

Then, compute a probability distribution P over these scores (usually with a softmax function):

$$P(y_t = w | \{y_{<t}\}) = \frac{\exp(S_w)}{\sum_{w' \in V} \exp(S_{w'})}$$

decoding algorithm defines a function to select a token from this distribution:

$\hat{y}_t = g(P(y_t | \{y_{<t}\}))$ $g(.)$ is decoding algorithm

exhaustive search Decoding

want to find a (length T) translation y that maximizes

$$P(y|x) = P(y_1|x)P(y_2|y_1,x)P(y_3|y_1,y_2,x)\cdots P(y_T|y_1,\cdots y_{T-1},x)$$

$$= \prod_{t=1}^{T} P(y_t|y_1,\cdots y_{t-1},x)$$

could try compute all possible sequences y

This means that on each step t of the decoder, we're tracking V possible partial translations, where V is vocab size

Complexity is far too expensive: $O(V^T)$

NLG Decoding Strategy

Greedy decoding

select the highest probability token in $P(y_t|y_{<t})$

$$\hat{y}_t = \underset{w \in V}{\operatorname{argmax}} P(y_t = w|y_{<t})$$

在每一步找出当前这一步的最优解，然后把这个解当成正确的解，代入下一步进行计算

Stop criterion:

到 <END> token 时停止 decode

实际code: np.argmax(conditional_probability)

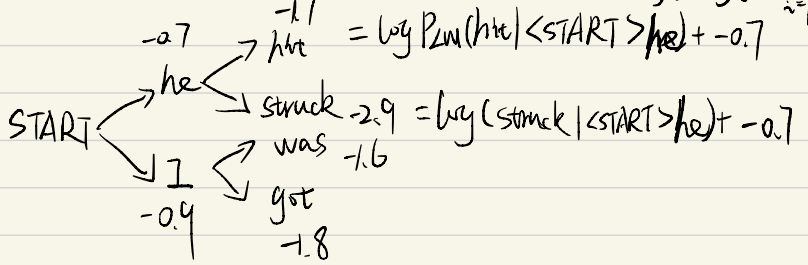
Greedy search的问题:
①局部最优解不是全局的最优解

②一错全错

Beam decoding:

从 top-1 到 top-N 个 (usually N: 5-10), 每步看top-N个最后score求和

ex: Beam size = k = 2 Blue numbers = score$(y_1,\dots,y_t) = \sum_{i=1}^{t} \log P_{LM}(y_i \mid y_1,\dots,y_{i-1},x)$



$= \log P_{LM}(\text{hit} \mid \langle START \rangle \text{ he}) + -0.7$

$= \log(\text{struck} \mid \langle START \rangle \text{ he}) + -0.7$

For each of the k hypothesis, find top k next words and calculate scores.

Stop Criterion
In beam search decoding, different hypothesis may produce <END> tokens on different timesteps
when a hypothesis produces, hypothesis is complete

beam search是在生成 softmax 后进行的, 也就是 inference 部分

What's the effect of changing beam size k?

Small k has similar problem to greedy decoding (k=1)
Ungrammatical, unnatural, nonsensical, incorrect

Larger k means consider more hypothesis

increasing k reduces some problems above
Larger k is more compute expensive

But increasing k can introduce other problems:
- for NMT, increasing k too much decreases BLEU score
  - large-k beam search produces too short translations (even with score normalization!)

In open-ended tasks like chit-chat dialogue, large k can make output more generic

Effect of beam size in chitchat dialogue

Low beam size: More on-topic but nonsensical;
High beam size: Converges to safe, 'correct' response, but it's generic and less relevant

Penalize longer sequences

$$s(Y) = \sum_{t=1}^{T} \log P(y_t | y_{<t}, x)$$

↓
shorter sequences will score better!

Solution: normalization $s(Y) = \frac{1}{|Y|} \sum_{t=1}^{|Y|} \log P(y_t | \{y\}_{<t})$

Solution: Normalize by token length relative to reference sequence

$$s(Y) = \frac{1}{lp(Y)} \sum_{t=1}^{|Y|} \log P(y_t | \{y\}_{<t}) \qquad lp(Y) = \frac{(5+|Y|)^{\alpha}}{(5+1)^{\alpha}}$$

α 在 paper 中推荐 0.5–0.7

Repetitive Problem 重复性问题

重复的单词会使 loss 一直下降

how to reduce repetition:

① Don't repeat n-grams (Hacky, but works!)

② Minimize additional loss term for minimizing hidden state similarity ($LSTM_s$)

$\hat{y}_t = g(\log P(y_t | \{y\}_{<t}) - s(h_t, h_{t-m}))$ 希望避免相近的 hidden state

③ 2020 年提出: Unlikelihood objective

Penalize generation of already-seen tokens

Sequence-level Unlikelihood Training

Given a set of undesired tokens C, lower their likelihood in context

$$L_{MLE}^t = -\log P(y_t^* | \{y^*\}_{<t}) \qquad L_{UL}^t = -\sum_{y_{neg} \in C} \log(1 - P(y_{neg} | \{y^*\}_{<t}))$$

$C = \{y^*\}_{<t}$ ← Unlikelihood objective lowers the probability of certain tokens

$$\mathcal{L}^t_{ULE} = \mathcal{L}^t_{MLE} + \alpha \mathcal{L}^t_{UL}$$

④

F² Softmax Avoid likelihood issues by factorizing the softmax

$$P(y_t | w_h | \{y\}_{<t}) = \frac{e^{U_{nh}}}{\sum_{m=1}^{M} e^{U_{mh}}}$$

↓

$$P(y_t = w_n | \{y\}_{<t}) = \left(\frac{e^{V_{ph}}}{\sum_{c=1}^{C} e^{V_{ch}}}\right)\left(\frac{e^{U_{nh}}}{\sum_{m=1}^{M} e^{U_{mh}}}\right)$$

实际上是应用了MF的思想

sampling strategy

random: Sampling

g=sample a token from the distribution of tokens

$$\hat{y}_t \sim P(y_t | w | \{y\}_t)$$

Too random: Temperature Scaling

Too much randomness: distribution has too much entropy

Solution: Make the distribution more "peaky" with temperature scaling

$$P(x_i | x_{1:i-1}) = \frac{\exp(u_i/t)}{\sum_j \exp(u_j/t)}$$

Raise the temperature $t>1$: P becomes more uniform

More diverse output (probability is spread around vocab)

Lower the temperature $t<1$: P becomes more spiky
Less diverse output (probability is concentrated on top words)

Top-k sampling:

Random sample token from top k highest probability tokens in $P(\cdot)$

only sample from the top k tokens in the distribution

$\hat{y}_t \sim P^*(y_t = w \mid y_{<t})$

Question of top-k Sampling:

概率很低的单词被 Top-k sampling 完全排除掉

Flat Distribution: 很容易舍弃与 top-k 概率相近的单词

Peaked distribution: 可能有的单词概率很低但是因为在 top-k 里被留下来了。

Top-p (Nucleus) Sampling

p: percent

$\sum_{x \in V(p)} p(x|x_{1:t-1}) \geq p$

本质上 Top-p Sampling & Top-k Sampling 都是从 truncated Vocabulary distribution 中 sample token，区别在于置信区间的选择

通常 top-k 和 top-p Sampling 我们都会一起用，然后做 trade-on

transformer:
先做 top-k 再做 top_p

Research: paperswithcode.com

ResearchGate.net
duebenchmarks.com 中文数据集

Evaluate Metrics:

① ROUGE-N

N: N-gram

measures the number of matching 'n-grams' between our model-generated text and a 'reference'

↓

参考答案

Precision = $\frac{\#\text{ of n-grams found in model and reference}}{\#\text{ of n-grams in model}}$

ex: Model: "the hello a cat dog fox jumps"

reference text: "the fox jumps" → ['the', 'fox', 'jumps']

$$\frac{Count_{match}(gram_n)}{Count(gram_n)} = \frac{3}{7} = 43\% \text{ precision}$$

ex: Rouge-1 Rouge-Vm Rouge-2 都是指的n-gram

F1-score: $2 * \frac{precision * recall}{precision + recall}$

② Rouge-L

measures the longest common subsequence (LCS) between our model output and reference

ex:

Model → "the hello a cat dog fox jumps"

Common subsequence length=2

Common subsequence length=1

reference text: "the fox jumps"

longest Common subsequence length=2

Precision:

$$\frac{LCS(gram_n)}{count(gram_n)} = \frac{2}{7} = 29\% \text{ precision}$$

Recall:

$$\frac{LCS(gram_n)}{Count(gram_n)} = \frac{2}{3} = 66\%$$

"reference text"

F1 score:

$$2 * \frac{Precision * recall}{precision + recall} = 0.6$$

ROUGE-S

S for skip-Gram

ex: model: "the brown fox jumps"

"the fox"

reference text

Precision

$$\frac{match(gram_2)}{Count(gram_2)} = \frac{1}{3} = 33\%$$

Recall: "the brown fox jumps"

"the fox"

reference text

$$\frac{match(gram_2)}{Count(gram_2)} = \frac{1}{1} = 100\%$$

ROUGE 的缺点(cons):

无法去覆盖语义的表达

Solution: 通过给多个表达方式(可能都是一个意思),在计算时分别对它们做计算,也就是生成好几个分值,然后再对这些Rouge的分值做average,这样会一定程度的缓解

```
!pip install rouge
```