DialoGPT: Large-Scale Generative Pre-training for Conversational Response Generation

120220225 조유빈

1. DialoGPT 논문 내용
   A. Neural response 생성은 text generation의 서브 카테고리이며 natural하게 텍스트를 생성하는 목적을 공유
      i. 텍스트 요약, paraphrasing과 같은 텍스트 생성보다 더욱 많은 one-to-many 문제를 야기함
      ii. 사람의 대화는 informal, noisy한 텍스트 chat 형식으로 informal abbreviations 또는 syntactic/lexical errors를 포함
   B. 최근 open-domain neural response 생성은 content 또는 style inconsistency와 lack of long-term contextual information로 인해 어려움
   C. DialoGPT
      i. Reddit 커멘트에서 추출된 147M의 대화로 학습
      ii. GPT-2와 같은 트랜스포머 기반 구조는 self-attention 메커니즘을 사용하여 long-term dependency information을 보존되도록 해서 content consistency를 향상시킴
      iii. 세밀한 대화 흐름에서 joint distribution of P(target, source)를 캡쳐
2. Method
   A. GPT-2를 기반으로 학습

```
model = load_model(GPT2LMHeadModel(config), args.init_checkpoint,
                   args, verbose=True)
```

      i. GPT-2는 일반적인 transformer language model로 masked multi-head self-attention layers의 stack을 활용하여 매우 큰 web-text 데이터로 학습

되었음

B. GPT-2로 긴 텍스트 multi-turn 대화 세션을 모델링하고 LM을 생성 태스크로 구성

```
class GPT2LMHeadModel(GPT2PreTrainedModel):
    def __init__(self, config):
        super(GPT2LMHeadModel, self).__init__(config)
        self.transformer = GPT2ModelFP16(config)
        self.lm_head = GPT2LMHead(self.transformer.wte.weight, config)
        self.apply(self.init_weights)
```

C. 첫번째로 대화세션 안에서 모든 대화 turns을 concatenate시켜 긴 text x_1, …, x_N (N은 시퀀스 길이)을 만들고 끝에는 end-of-text token을 넣음

```
def _get_response(output_token, past):
    out = torch.tensor([[]], dtype=torch.long, device=device_f)

    while True:
        output_token, past = model.forward(output_token, past=past)
        output_token = output_token[:, -1, :].float()
        indices_to_remove = output_token < torch.topk(output_token, top_k)[0][..., -1, None]
        output_token[indices_to_remove] = -float('Inf')
        output_token = torch.multinomial(F.softmax(output_token, dim=-1), num_samples=1)

        out = torch.cat((out, output_token), dim=1)

        if output_token.item() == end_token.item():
            break
```

D. 모델은 아래 식을 최적화하도록 함

$$p(T|S) = \prod_{n=m+1}^{N} p(x_n|x_1, \cdots, x_{n-1}) \quad (1)$$

i. Source sentence: $S = x_1, \cdots, x_m$, Target sentence: $T = x_{m+1}, \cdots, x_N$

ii. Multi-turn dialogue session $T_1, \cdots, T_K$는 $p(T_K, \cdots, T_2|T_1)$로 볼 수 있고 이는 $p(T_i|T_1, \cdots, T_{i-1})$ 조건부 확률을 product한 것과 같음

✓ $p(T_K, \cdots, T_2|T_1)$ 을 최적화하는 것은 모든 $p(T_i|T_1, \cdots, T_{i-1})$ source-target pair를 최적화하는 것

E. Mutual Information Maximization

i. Open-domain 텍스트 생성 모델의 bland하고 uninformative함을 해결하기 위해, objective function을 maximum mutual information(MMI) scoring function으로 대체

ii. MMI에서 파라미터는 source와 target 간의 상호정보(mutual information)을 최대화하도록 학습됨

$$\hat{T} = \arg\max_T \{(1-\lambda) logp(T|S) + \lambda logp(S|T)\}$$

iii. MMI는 pretrained backward model가 given response로부터 source sentence를 예측하도록 적용. 즉, $P(Source|Target)$

- ✓ 먼저 top-K sampling을 통해 hypothesis set 생성
- ✓ 모든 hypothesis를 re-rank하기 위해 $P(Source|Hypothesis)$ 조건부확률을 사용
- ✓ 직관적으로 backward model (pre-trained model) likelihood를 최대화하는 것은 bland hypothesis에 불이익을 줌
  - A. 빈번하고 반복적인 hypothesis는 많은 source query와 연관이 가능하기 때문에 모든 쿼리에 대해 확률 값이 낮아지기 때문
  - B. 따라서 특정 쿼리에 대해 높은 확률 값을 가지지 않게 됨

3. Implement result

```
User:hello
Bot: Hey, how are you?
User:I am good
Bot: That's good. Are you sure?
User:yes, i am pretty good
Bot: That's good.
User:quit
```