자연어처리 프로젝트

220220018 천호진

120220107 강희주

220200013 이해중

목차

개요

Language model은 단어 시퀀스에 확률을 할당(assign)하는 일을 하는 모델이다. 이를 조금 풀어서 쓰면, Language model은 가장 자연스러운 단어 시퀀스를 찾아내는 모델이다. 단어 시퀀스에 확률을 할당하게 하기 위해서 가장 보편적으로 사용되는 방법은 Language model이 이전 단어들이 주어졌을 때 다음 단어를 예측하도록 하는 것이다.

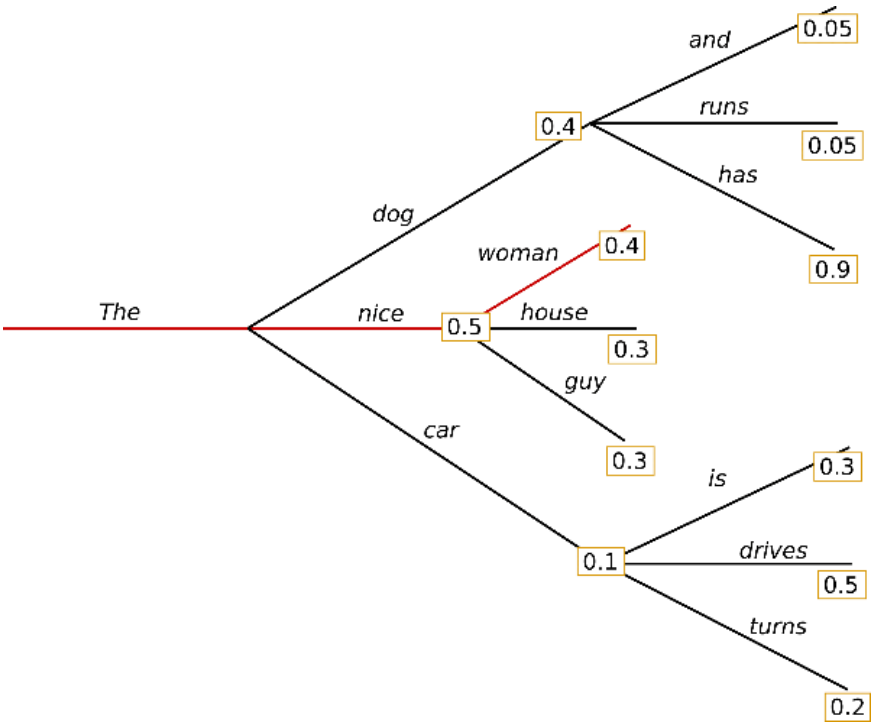
Language model로부터 응답을 생성하는 것을 모델 디코딩이라고 한다. 여러가지 모델 디코딩 방법이 존재하고 각자의 장단점이 있다.

문장을 구성하는 각 토큰을 w_i, 이 토큰의 확률을 $P(w_i)$라고 하자. 문장을 구성하는 토큰들이 서로 독립이라고 하면 문장 $(w_1, w_2, ..., w_n)$ 의 확률 $P(w_1, ..., w_n) = \prod_{i=1}^{n} P(w_i)$이다. 디코딩은 language model로부터 확률이 높은 문장을 출력하는 것이다.

Greedy Decoding

자연어 생성은 단어들의 시퀀스를 아웃풋으로 예측해내는 태스크이다. 일반적으로 생성 모델은 각각의 디코딩 타임 스텝에서 전체 단어 사전에 대한 확률 분포를 예측한다. 따라서 실제로 단어를 생성해내기 위해서는 모델의 예측 확률 분포를 이용해 각 타임 스탭의 단어로 변환하는 과정이 필요하다. 이렇게 모델이 예측한 확률 분포에 대해 디코딩하기 위해서는 예측된 확률 분포에 따라 가능한 모든 아웃풋 시퀀스의 조합을 탐색(search)해야 한다. 그런데 일반적으로 단어 사전은 수만 개의 토큰을 포함하고 있기 때문에 전체 공간을 탐색하는 것은 계산적으로 불가능하다. 따라서 실제로는 휴리스틱한 방법을 사용해 “충분히 좋은” 아웃풋 시퀀스를 생성해내도록 한다.

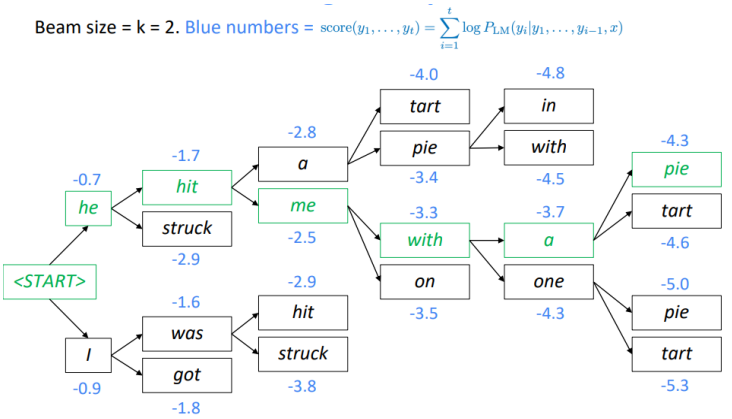
가장 직관적이며 단순한 디코딩 방법은 아웃풋 시퀀스에서 생성된 각각의 확률 분포에서 가장 값이 높은 토큰을 선택하는 greedy decoding이다. Greedy decoding은 첫 번째 토큰부터 마지막 토큰까지 각 토큰을 예측할 때 확률이 가장 높은 토큰을 선택하는 디코딩 방법이다. Greedy decoding은 단순하고 빠르다는 장점이 있으나, 각 시점에 확률이 가장 높은 토큰을 선택하는 것이 문장의 확률을 최대화하는 것은 아니다. 또한 디코딩은 이전 토큰으로부터 다음 토큰을 예측하는 작업이므로 이전 시점에 어떤 토큰이 선택되었는지가 중요하게 작용한다. 따라서 확률이 가장 높은 토큰 하나만 고려하는 greedy decoding보다 각 시점에서 토큰을 확률적으로 선택할 수 있는 방법이 좋을 것이다.



Beam search

Greedy decoding의 단점을 보완한 방법으로 beam search가 있다. 각각의 time-step에서 가능도가 가장 높은 하나의 토큰을 선택하는 Greedy decoding과 달리 beam search에서는 각 스텝에서 탐색의 영역을 K개의 가장 가능도가 높은 토큰들로 유지하며 다음 단계를 탐색한다. 다시 말해, beam search의 핵심 아이디어는 디코더의 각 time-step에서 가장 확률이 높은 k (beam size)개의 부분 번역을 추적하는 것이다. 이렇게 생성된 부분 번역을 hypothesis라고 하며, hypothesis $y_1, \dots, y_t$는 각각의 score를 가지게 된다. 각 단계에서 score가 높은 k개의 hypothesis를 추적하면서 score가 높은 hypothesis을 검색한다.

$$score(y_1, \dots, y_t) = log\, P_{LM}(y_1, \dots, y_k \mid x) = \sum_{i=1}^{t} log\, P_{LM}(y_i | y_1, \dots, y_{i-1}, x)$$



greedy decoding에서 <END> token이 나오면 text 생성을 종료하지만 beam search decoding에서 서로 다른 hypothesis은 다른 time-step에서 <END> token을 생성하여 종료하게 된다. 즉, 사전 정의된 T라는 최대 time-step에 도달할때까지 beam-search 과정을 중단하게 되며, <END> token으로 완료된 hypothesis가 사전에 정의된 n개가 된다면 beam-search 과정을 중단하게 된다. 그 결과 완료된 hypothesis의 list를 최종적으로 얻게되며 그 중에서 가장 높은 score를 가지는 hypothesis를 최종 결과물로 사용할 수 있다. 하지만 log probability 성질에 의해 hypothesis의 sequence의 길이가 상대적으로 짧은 hypothesis의 score가 높을 것이고, sequence의 길이가 긴 hypothesis는 score가 낮은 문제점이 존재한다. 이처럼 단어가 많이 생성될 수록 joint probability 값이 작아지는 문제점을 해결하기 위해 각 hypothesis별로 가지고 있는 sequence의 길이로 나눠줌으로써 위의 문제점을 해결할 수 있다.

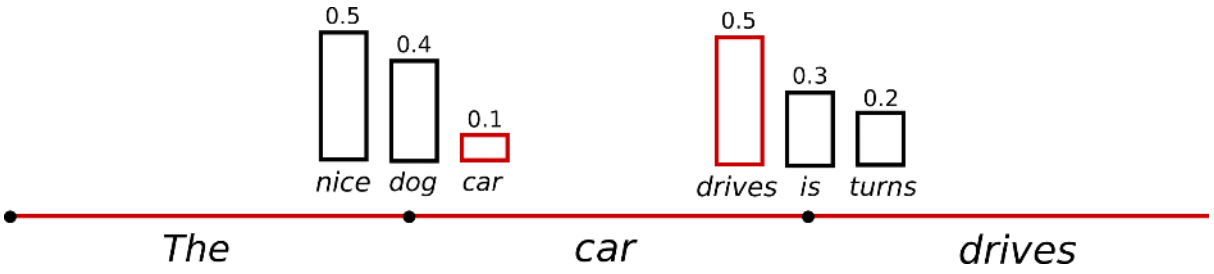
$$score(y_1, \dots, y_t) = \frac{1}{t}\sum_{i=1}^{t} log\, P_{LM}(y_i|y_1, \dots, y_{i-1}, x)$$

Beam Search는 global optimal solution을 보장하지는 않지만 Exhaustive Search보다 훨씬 효율적이다. 또한 k를 크게 가져가면 더 넓은 영역을 탐색할 수 있으므로 더 좋은 타겟 시퀀스를 생성할 수 있지만, 그만큼 속도가 느려지기 때문에 조절이 필요하다. 일반적으로 기계번역 태스크에서는 beam size k를 5 혹은 10으로 설정하며, 우리의 프로젝트 코드에서 k = 5로 설정하였다.

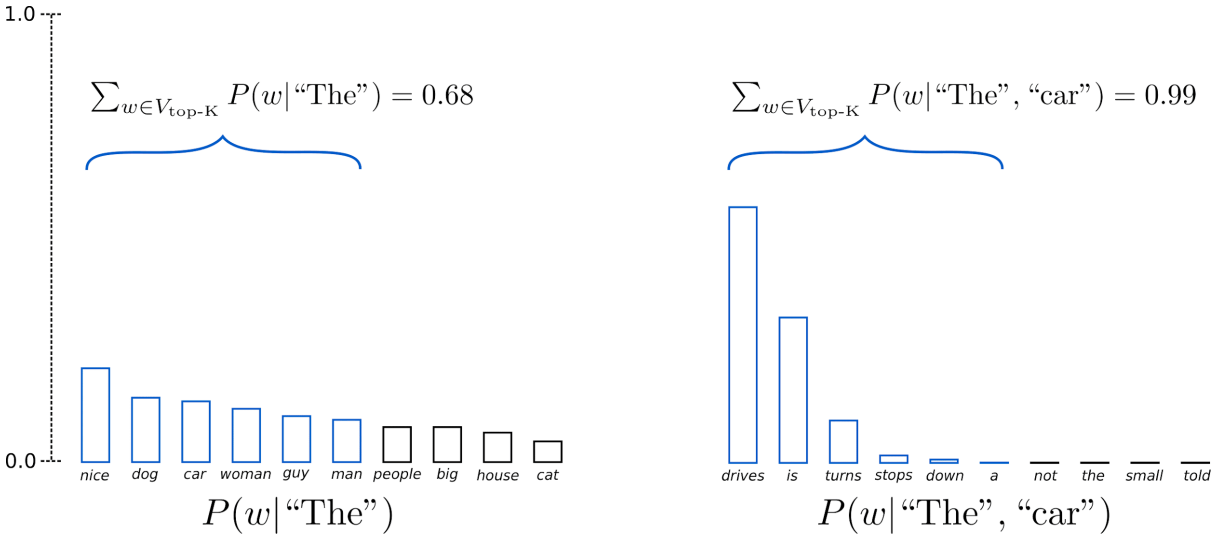
Sampling

디코딩의 다른 방법으로 Sampling, Top-K Sampling, Top-p Sampling이 있다.

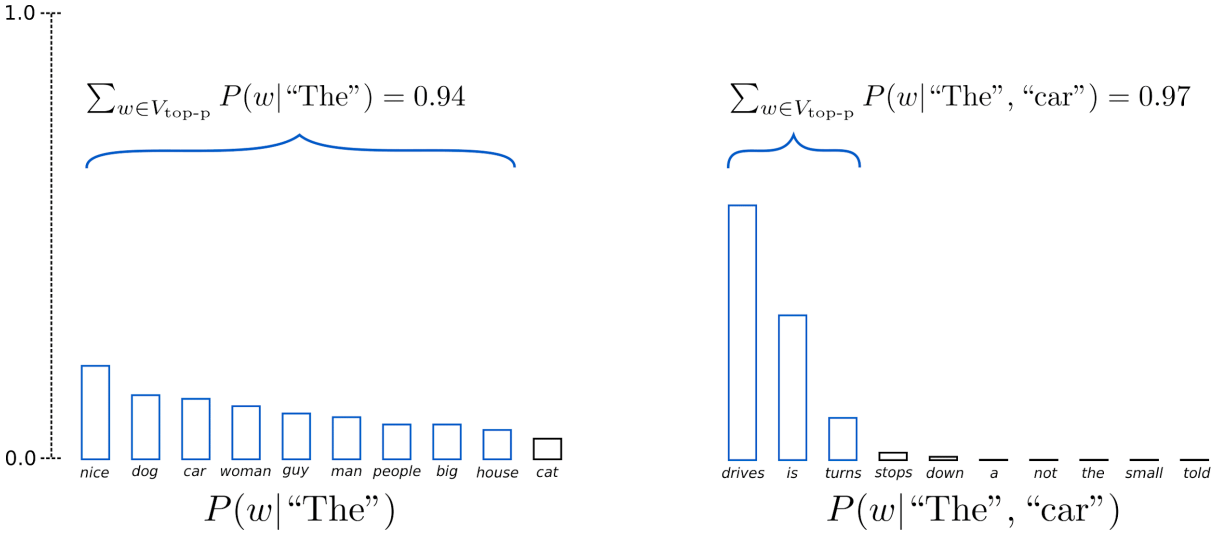
Sampling은 각 시점에서 각 토큰의 확률에 따라 선택하는 방법이다. 확률이 높은 토큰만이 선택되는 방법이 아니기 때문에 다양한 문장을 출력할 수 있다. 그러나 일반적인 sampling 방법은 확률이 아주 낮은 단어가 선택될 수도 있다는 문제가 있다.



Top-K 샘플링은 확률이 낮은 토큰이 선택되는 문제를 해결하기 위해 각 시점에서 확률이 가장 높은 K개의 단어들 중에서 확률에 따라 샘플링하는 방법이다. 그러나 Top-K 샘플링의 경우에도 확률이 높은 상위 K개의 토큰 중에 확률이 낮은 토큰이 존재한다면 확률이 낮은 토큰이 선택될 가능성이 있다.



Top-p 샘플링은 이를 해결하기 위한 샘플링 방법이다. 확률이 높은 순서대로 확률을 더해 합이 p를 넘을 때까지 토큰을 선택하고, 합이 p를 넘으면 토큰 선택을 중단한다. 선택된 토큰들 중에서 확률에 따라 샘플링하는 방법이 Top-p 샘플링이다.



이처럼 다양한 디코딩 방법이 존재하고, 어떤 방법을 사용하는지에 따라 얼마나 흥미로운 문장을 생성하는지, 얼마나 사람처럼 문장을 생성하는지 등 모델의 특성이 달라진다.

DialoGPT의 저자는 DialoGPT가 “I don’t know. I’m OK.”와 같이 문맥이 맞지 않고 어정쩡한 답을 출력하는 경우가 있다고 했고, 그 이유가 likelihood P(T|S)의 최대화는 Source로부터 Target을 추정하지만 Target으로부터 Source를 추정할 수 있는지는 고려하지 않기 때문이라고 주장했다. 저자는 어정쩡한 응답 생성 문제를 해결하기 위해 P(T|S)와 P(S|T)를 함께 사용하여 T와 S 사이의 mutual information을 최대화하는 Maximize Mutual Information 디코딩 방법을 제안했다.

DialoGPT의 저자는 MMI decoding이 top-k sampling보다 다양한 metric에서 더 좋은 성능을 보이고, 다양한 응답을 출력한다는 것을 밝혔다. 그러나 MMI decoding은 P(S|T)를 추정하기 위해 DialoGPT와 같은 구조의 reverse model을 필요로 하므로 학습과 추론에 2배의 리소스가 필요하다는 단점이 있다.

많은 리소스를 필요로 한다는 문제를 해결하기 위해서, 본 프로젝트는 작은 크기의 discriminator를 사용하여 문맥적으로 적절하고 ‘사람 같은’ 응답을 생성하는 것을 목표로 한다. 데이터셋의 label은 1로, DialoGPT가 생성한 응답은 0으로 분류하는 discriminator를 학습시키고, beam search를 사용하여 여러 개의 응답을 DialoGPT로부터 생성한다. 그 다음 여러 개의 응답 중 discriminator가 가장 높은 출력을 내는 응답 문장을 선택한다.

MMI decoding

DialoGPT는 source 문장 S로부터 target 문장 T를 추정하는 모델로, likelihood P(T|S)가 최대화되도록 학습된다. 즉, DialoGPT가 추정한 target 문장 $\hat{T}$는 $\hat{T} = \arg arg\, logp(T|S)$ 이다. 논문의 저자들은 이와 같이 일반적인 MLE 방식이 source로부터 target을 추정하지만 target으로부터 source를 추정할 수 있는지는 고려하지 않기 때문에 bland, uninformative한 응답이 생성된다고 추측했다.

이런 문제를 해결하기 위해 DialoGPT의 저자는 $\hat{T}$를 추정하는 objective function을 P(T|S)와 P(S|T)의 합을 최대화하여 T와 S 사이의 mutual information을 최대화하도록 아래와 같이 수정했다.

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

이 방법을 Maximization of Mutual Information, MMI라고 한다. MMI를 위해서는 P(S|T)를 추정해야 하는데 이 때 T로부터 S를 추정하는 pre-trained backward model을 사용한다. MMI는 먼저 top-k sampling을 사용하여 hypothesis set을 $P(Hypothesis|Source)$로 생성한다. 다음으로 mutual information을 기준으로 hypothesis를 re-rank한다. 이를 위해서 조건부 확률 $P(Source|Hypothesis)$를 사용한다. $P(Source|Hypothesis)$를 최대화하는 것은 bland hypothesis에 패널티를 준다.

Method	NIST N-2	NIST N-4	BLEU B-2	BLEU B-4	METEOR	Entropy E-4	Dist D-1	Dist D-2	Avg Len
PERSONALITYCHAT	0.78	0.79	11.22%	1.95%	6.93%	8.37	5.8%	18.8%	8.12
Training from scratch:									
DIALOGPT (117M)	1.23	1.37	9.74%	1.77%	6.17%	7.11	5.3%	15.9%	9.41
DIALOGPT (345M)	2.51	3.08	16.92%	4.59%	9.34%	9.03	6.7%	25.6%	11.16
DIALOGPT (762M)	2.52	3.10	17.87%	5.19%	9.53%	9.32	7.5%	29.3%	10.72
Training from OpenAI GPT-2:									
DIALOGPT (117M)	2.39	2.41	10.54%	1.55%	7.53%	10.77	8.6%	39.9%	12.82
DIALOGPT (345M)	3.00	3.06	16.96%	4.56%	9.81%	9.12	6.8%	26.3%	12.19
DIALOGPT (345M, Beam)	**3.4**	**3.5**	**21.76%**	**7.92%**	10.74%	10.48	**12.38%**	**48.74%**	11.34
DIALOGPT (762M)	2.84	2.90	18.66%	5.25%	9.66%	9.72	7.76%	29.93%	11.19
DIALOGPT (762M, Beam)	2.90	2.98	21.08%	7.57%	10.11%	10.06	11.62%	44.07%	10.68
DIALOGPT (345M, MMI)	3.28	3.33	15.68%	3.94%	**11.23%**	**11.25**	9.39%	45.55%	17.21
Human	3.41	4.25	17.90%	7.48%	10.64%	10.99	14.5%	63.0%	13.10

Table 3: 6K Reddit multi-reference evaluation. “Beam” denotes beam search. “Human” represents the held-out ground truth reference.

MMI가 greedy, beam search보다 METEOR, Entropy에서 높은 점수를 얻었고 BLEU에서는 조금 낮은 점수를 얻었다. 결과적으로 MMI-reranking 기법이 더 다양한 응답을 생성한다는 것을 알 수 있다.

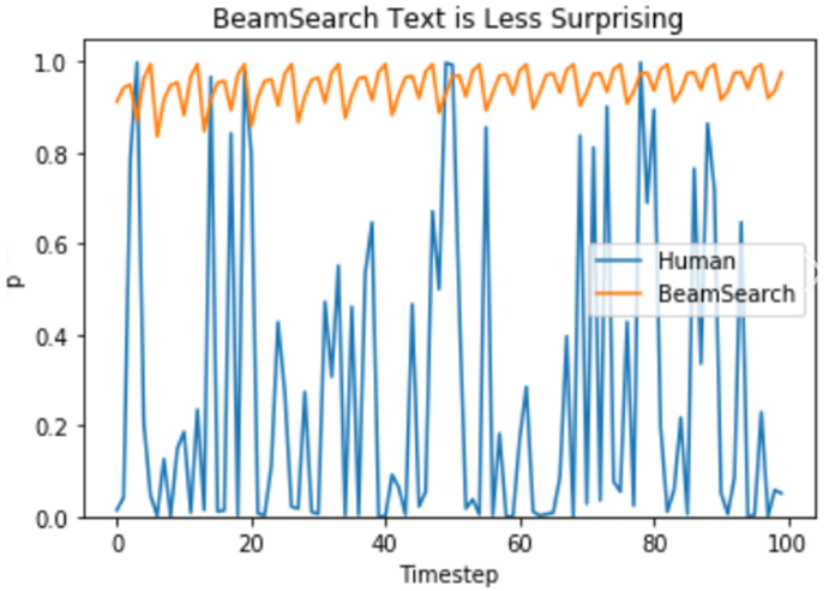
Beam Search + Discriminator

DialoGPT는 bland, uniformative한 응답을 생성하기 때문에 사람과 대화를 나눌 때 챗봇이 생성하는 응답이 어색하게 느껴질 수 있다는 단점이 있다. 이를 해결하기 위해 MMI 방식을 추가로 사용하지만, MMI 방식의 문제점은 objective function 에서 P(S|T)를 예측하기 위해서 Target으로부터 Source를 예측하는 reverse model이 필요하다는 점이다. 이 점 때문에 학습과 추론에 사용되는 비용이 기본적인 DialoGPT의 2배가 되며, fine tuning 할 때도 2배의 시간과 리소스가 요구된다.

따라서 MMI의 reverse model을 사용하지 않으면서 DialoGPT의 응답을 자연스럽게 만들어 주는 방법으로 hypothesis re-rank를 수행할 때 실제 데이터의 lable과 DialoGPT가 생성한 응답을 구분하는 discriminator를 사용했다. 이를 위해 Beam search를 이용해 모델을 통해 예측된 확률 분포에서 가장 확률이 높은 k개의 sequence를 선택한 후, discriminator를 적용하는 방식을 선택하였다. discriminator는 실제 응답에 대해서 1을 출력하기 때문에, beam search를 통해 선택된 k개의 sequence에 discriminator가 적용되었을 때 출력되는 값을 확인한 후 1에 가장 가까운 sequence를 최종적으로 출력함으로써 응답이 실제처럼 더 자연스럽게 만들어주었다.

작은 크기의 discriminator를 사용하면 reverse model을 사용할 때보다 resource를 많이 절약할 수

있다는 장점이 있다. 또한 discriminator의 사용이 beam search의 단점을 보완해준다.



위의 그래프를 보면 알 수 있듯이, 인간의 답변은 확률이 높은 문장도 있지만 낮은 문장도 존재하기 때문에 인간의 대답은 흥미롭게 느껴지는 특징이 있다. 하지만 beam search는 확률이 높은 문장을 생성하는데 집중하기 때문에 확률이 낮은문장은 아예 출력되지 않는다. 따라서 beam search를 사용한 챗봇과의 대화에서는 흥미를 느끼기 어렵다는 단점이 있다.

하지만 우리가 구현한 챗봇에서는 beam search로 여러 개의 문장을 생성한 다음에 discriminator를 사용하여 그 중 가장 인간의 대답에 가까운 시퀀스를 선택해 출력하기 때문에 확률이 높지만 인간처럼 흥미로운 대답을 하지 않는다는 beam search의 단점을 보완할 수 있다는 장점이 있다.

구현 방법, 코드

DialoGPT 모델은 Hugging face의 transformers 패키지에서 AutoModelForCausalLM과 AutoTokenizer를 사용하였다. 이것은 pre-trained DialoGPT-medium 모델이다.

아래는 DialoGPT 모델 생성 코드와 Beam Search를 사용해 DialoGPT로부터 하나의 source 문장에 대해 5개의 target 문장을 생성하여 출력한 결과이다.

DialoGPT 모델 코드

```
model_name = "microsoft/DialoGPT-medium"

enc = AutoTokenizer.from_pretrained(model_name)

model = AutoModelForCausalLM.from_pretrained(model_name)
```

DialoGPT 출력

Input : 45 up votes and 60 down, does this obamajoker guy really have 44 other accounts to up vote himself with or is reddit infested with chantards.

Output1 : It's a bot.

Output2 : It's a bot

Output3 : He's a bot.

Output4 : He's a troll.

Output5 : I think it's just a bot.

Label : If you view him as a troll he's actually quite amusing.

Input : Awesome. I wish rotate was tied to the up arrow though instead of space.

Output1 : Yeah, that would be a lot better.

Output2 : Yeah, that would be nice.

Output3 : Yeah, that would be a lot easier.

Output4 : That's a good idea. I'll add that.

Output5 : That's a good idea, I'll add that.

Label : Or both. I personally HATE up being rotate. Z and X are my preferred choices.

Input : is 65535 a zip code perhaps? edit s area zip

Output1 : It's a zip code.

Output2 : No, it's a city.

Output3 : No, it's a zip code.

Output4 : Nope, it's a city.

Output5 : Nope, it's a zip code.

Label : haven't looked it up yet, but I'm guessing somewhere around southern Illinois

Discriminator 모델은 간단하게 LSTM과 fully connected layer로 구성하였다. Transformer 기반의 모델을 사용할 수 있지만 개념 검증을 위해 단순한 모델을 사용하였다. LSTM은 hidden_size=512인 bidirectional LSTM을 10 layer 사용하였다. Fully connected layer를 2층 사용하여 2048차원의 hidden states로 1차원의 확률을 추정하였다. Hidden layers의 활성화함수는 Leaky ReLU를 사용하였고 출력층의 활성화함수는 Sigmoid를 사용하였다. 그리고 Binary Cross Entropy Loss를 lr=0.001의 Adam Optimizer를 사용해 최소화하도록 학습시켰다.

Discriminator 모델 구조, Discriminator 학습 코드

```
class Discriminator(nn.Module):

    def __init__(self):

        super(Discriminator, self).__init__()



        self.num_layers = 10

        self.lstm = nn.LSTM(input_size=1, hidden_size=512, num_layers=self.num_layers,
batch_first=True, bidirectional=True, dropout=0.3)

        self.fc1 = nn.Linear(512*2*self.num_layers, 256)

        self.fc2 = nn.Linear(256, 1)

        self.leakyrelu = nn.LeakyReLU()

        self.sigmoid = nn.Sigmoid()



    def forward(self, input):

        batch_size = input.size(0)

        h0 = torch.zeros((2*self.num_layers, batch_size, 512)).to(device)

        c0 = torch.zeros((2*self.num_layers, batch_size, 512)).to(device)

        input = input.reshape(batch_size, -1, 1).float()

        _, (h, c) = self.lstm(input, (h0, c0))

        h = h.reshape(batch_size, 1024*self.num_layers)

        output = self.leakyrelu(self.fc1(self.leakyrelu(h)))

        output = self.sigmoid(self.fc2(output))



        return output



out_real = []

out_fake = []

for epoch in range(10):

  i = 0
```

```
  for batch in train_dataset:

      #print(input_ids.shape, position_ids.shape, token_ids.shape, label_ids.shape)

      batch = tuple(t.to(device) for t in batch)

      input_ids, position_ids, token_ids, label_ids, *_ = batch

      input_ids = input_ids[0].reshape(1, -1)

      label_ids = label_ids[0].reshape(1, -1)



      #모델 출력 생성 (out)

      input_ids =
torch.cat((torch.tensor(cut_seq_to_eos(input_ids[0].tolist())).reshape(1, -1),
torch.tensor([[EOS_ID]])), dim=1).to(device).long()



      out = model.generate(input_ids, pad_token_id=-1, max_length=1000,
eos_token_id=EOS_ID, num_beams=5, num_return_sequences=1)

      out = out[0][input_ids.size(-1)+1:-1].reshape(1, -1)



      label_ids = label_ids[label_ids!=-1]

      label_ids = torch.tensor(cut_seq_to_eos(label_ids.tolist())).reshape(1,
-1).to(device)



      #discriminator 입력 생성 (질문과 응답을 이어붙임)

      real_input = torch.cat((input_ids, label_ids), dim=1).to(device)

      fake_input = torch.cat((input_ids, out), dim=1).to(device)

      #print(label_ids)



      #discriminator 타겟 생성

      tgt_real = torch.ones((1, 1)).to(device).float()

      tgt_fake = torch.zeros((1, 1)).to(device).float()
```

```
#real 역전파

output_real = D(real_input)

loss_real = criterion(output_real, tgt_real)

loss_real.backward()

optimizer.step()

optimizer.zero_grad()

#fake 역전파

output_fake = D(fake_input)

loss_fake = criterion(output_fake, tgt_fake)

loss_fake.backward()

optimizer.step()

optimizer.zero_grad()



out_real.append(output_real.cpu().item())

out_fake.append(output_fake.cpu().item())



if i % 100 == 0:

  print(epoch, i, output_real, output_fake)

  torch.save(D.state_dict(), 'D_lstm10_fc2.pkl')

i += 1
```

학습에 사용된 데이터는 Reddit dataset을 사용하였다. 컴퓨팅 자원의 한계로 인해 Reddit dataset에서 10000개의 문장을 무작위로 선택하여 학습 데이터로 사용하였고, 1000개의 문장을 무작위로 선택하여 테스트 데이터로 사용하였다. Discriminator 모델은 총 50000 iteration 학습되었다.

데이터셋 코드

```
num_train_samples = 10000

num_test_samples = 1000

train_dataset = []

test_dataset = []



for i, batch in enumerate(train_dataloader):

    if i < num_train_samples:

        train_dataset.append(batch)

    if i >= num_train_samples and i < num_train_samples+num_test_samples:

        test_dataset.append(batch)

    if i == num_train_samples+num_test_samples:

        break

    if i % 100 == 0:

        print(i)
```

50000 iteration 학습 후 1000개의 테스트 데이터에 대해 Discriminator 모델을 평가하였다. Discriminator 모델은 1000개의 쌍 중 820개를 성공적으로 분류하였다.

테스트 코드

```
model.eval()

out_real_test = []

out_fake_test = []



for batch in tqdm(test_dataset):

    batch = tuple(t.to(device) for t in batch)

    input_ids, position_ids, token_ids, label_ids, *_ = batch

    input_ids = input_ids[0].reshape(1, -1)

    label_ids = label_ids[0].reshape(1, -1)



    #모델 출력 생성 (out)

    input_ids =
torch.cat((torch.tensor(cut_seq_to_eos(input_ids[0].tolist())).reshape(1, -1),
torch.tensor([[EOS_ID]])), dim=1).to(device).long()



    out = model.generate(input_ids, pad_token_id=-1, max_length=1000,
eos_token_id=EOS_ID, num_beams=5, num_return_sequences=1)

    out = out[0][input_ids.size(-1)+1:-1].reshape(1, -1)



    label_ids = label_ids[label_ids!=-1]

    label_ids = torch.tensor(cut_seq_to_eos(label_ids.tolist())).reshape(1,
-1).to(device)



    #discriminator 입력 생성 (질문과 응답을 이어붙임)

    real_input = torch.cat((input_ids, label_ids), dim=1).to(device)

    fake_input = torch.cat((input_ids, out), dim=1).to(device)



    output_real = D(real_input)
```

```
    output_fake = D(fake_input)



    out_real_test.append(output_real.cpu().item())

    out_fake_test.append(output_fake.cpu().item())



np.count_nonzero(out_real_test>out_fake_test) / len(out_real_test)



>> 0.82
```

실험 결과 **(Benchmark)**

제안하고 모델이 생성하는 출력에 대해 2-gram과 4-gram의 수를 계산하여 다양성의 정도를 평가하기 위해 Dist-n ($n = 1, 2$)과 Entropy-n ($n = 4$)을 측정하였다.

	Dist-1	Dist-2	Entropy-4
DialoGPT (medium)	0.4006	0.7338	2.3761
Ours (discriminator)	0.3731	0.7282	2.5301

본 프로젝트에서 제안한 방법과 greedy search의 성능을 비교했을 때 Dist-1, Dist-2는 greedy search보다 약간 낮지만 비슷했고, Entropy-4는 더 높았다. Beam search와 discriminator를 사용한 디코딩 방법이 모델 구조 대비 다양한 응답을 생성한다는 것을 알 수 있다.

텍스트 생성 결과

```
>> User:Hi man
Beam+Discriminator Output 1 :  I'm not your friend, buddy! Score : 0.3236
Beam+Discriminator Output 2 :  I'm not your man, friend! Score : 0.3204
Beam+Discriminator Output 3 :  Hey man, how's it goin? Score : 0.3034
Beam+Discriminator Output 4 :  Hey man, how's it goin '? Score : 0.3007
Beam+Discriminator Output 5 :  Hey, how's it goin? Score : 0.2968
Greedy Output :  Hey man


>> User:What's your favorite movie?
Beam+Discriminator Output 1 :  The Big Lebowski. Score : 0.5231
Beam+Discriminator Output 2 :  The Big Lebowski Score : 0.4511
Beam+Discriminator Output 3 :  I don't really have a favorite. I like a lot of movies. Score : 0.373
Beam+Discriminator Output 4 :  I don't watch a lot of movies, so I don't really have one. Score : 0.3648
Beam+Discriminator Output 5 :  I don't really watch a lot of movies. Score : 0.3133
Greedy Output :  I don't really watch movies.


>> User:I'm tired. I want to sleep.
Beam+Discriminator Output 1 :  I feel ya Score : 0.2979
Beam+Discriminator Output 2 :  I'm sleepy. I want sleep. Score : 0.2863
Beam+Discriminator Output 3 :  Sleeping is for the weak. Score : 0.2776
Beam+Discriminator Output 4 :  I'm sleepy. I want a nap. Score : 0.2764
Beam+Discriminator Output 5 :  I'm awake. I want sleep. Score : 0.2738
Greedy Output :  I'm tired. I want to sleep.
```

참고문헌

- Yizhe Zhang, Michel Galley, Jianfeng Gao, Zhe Gan, Xiujun Li, Chris Brockett, and Bill Dolan. 2018. Generating informative and diverse conversational responses via adversarial information maximization. NeurIPS.
- Jiwei Li, Michel Galley, Chris Brockett, Jianfeng Gao, and Bill Dolan. 2016a. A diversity-promoting objective function for neural conversation models. NAACL.
- Zhang, Y., Sun, S., Galley, M., Chen, Y. C., Brockett, C., Gao, X., ... & Dolan, B. (2019). Dialogpt: Large-scale generative pre-training for conversational response generation. arXiv preprint arXiv:1911.00536.