NLP 3주차 스터디

작성일시	@2024년 2월 3일 오후 4:57
복습	☐
텍스트	5.1 ~ 5.7 장

텍스트 생성

GPT-2를 사용해 언어 모델의 텍스트 생성 원리를 이해하고,

다양한 디코딩 전략이 생성된 텍스트에 미치는 영향 살펴보기

5.1 일관성 있는 텍스트 생성의 어려움

시퀀스나 토큰 분류 같이 작업에 특화된 헤드에서 예측 생성은 매우 간단함

1. 모델이 일련의 로짓을 출력하고 최댓값을 선택해 예측 클래스를 얻기
2. 소프트 맥스 함수를 적용해 클래스별 예측 확률 얻기

하지만, 모델의 확률 출력을 **텍스트**로 변환하려면 **디코딩** 방법이 필요함

▫ 텍스트 생성만의 어려움

1. 디코딩은 **반복적**으로 수행되므로 모델의 정방향 패스를 한 번 통과할 때 보다 **많은 계산**이 필요
2. 생성된 텍스트의 품질과 다양성은 **디코딩 방법**과 이에 관련된 **하이퍼파라미터**에 따라 달라짐

▫ 디코딩 수행과정 이해하기

GPT-2의 사전 훈련 방법과 텍스트 생성에 적용하는 과정 알아보기

사전 훈련

- GPT-2는 시작 프롬프트 또는 **문맥 시퀀스** x = x1, … , xk가 주어질 때 텍스트에 등장하는 **토큰 시퀀스** y = y1, … , yt의 **확률 P(y|x)**를 추정하도록 사전 훈련된다.

- 직접 **P(y|x)**를 추정하기 위해 충분한 훈련 데이터를 획득하기란 불가능하므로, 일반적으로 확률의 연쇄 법칙을 사용해 **조건부 확률의 곱**으로 나타낸다.

$$P(y_1, \ldots, y_t \mid x) = \prod_{t=1}^{N} P(y_t \mid y_{<t}, x)$$

: y<t는 시퀀스 y1, … , yt-1을 간략화한 식

> 📢 이 조건부 확률로 **자기회귀 언어 모델링**은 문장의 **이전 단어**가 주어지면
>
> 다음 단어를 예측 가능. 위 식의 우항에 위치한 확률이 이를 설명. 이런 사전 훈련 목표는
>
> 과거와 미래의 문맥을 **모두 사용**해 마스킹된 토큰을 예측하는 BERT와 **다름.**

텍스트 생성

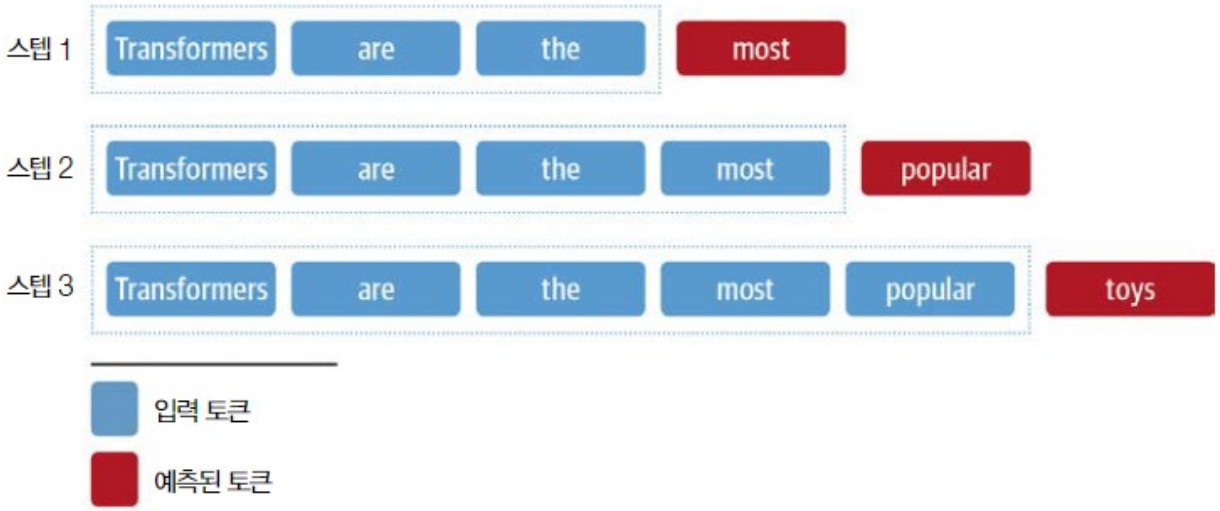


그림 5-3 스텝마다 새로운 단어를 입력 시퀀스에 추가해 텍스트 생성하기

- 'Transformers are the' 같은 프롬프트로 시작하면 모델은 다음 토큰을 예측.
- 다음 토큰이 결정되면 이를 프롬프트에 추가해 새로운 입력 시퀀스를 만들고 또 다른 토큰 생성.
- 이 과정을 특수한 시퀀스 종료 토큰이나 사전 정의된 최대 길이에 도달할 때까지 반복.

출력 시퀀스가 입력 프롬프트에 따라 결정되므로 이런 텍스트 생성을 **조건 텍스트 생성**이라 함

디코딩

이 과정의 핵심은 각 타임스텝에서 어떤 토큰을 선택할지 결정하는 **디코딩 방법**에 있음.

$$P(y_t = w_i \mid y_{<t},\ x) = \text{softmax}(z_{t,i})$$

언어 모델의 **헤드**는 각 스텝에서 어휘사전에 있는 토큰마다 **로짓** zt, i을 생성하므로

소프트맥스를 적용하면 가능한 다음 토큰 wi에 대한 확률 분포를 얻는다

대부분의 디코딩 방법은 다음과 같은 **y^**를 선택해 전체적으로 **확률이 가장 높은 시퀀스**를 찾는다

$$\hat{y} = \underset{y}{\operatorname{argmax}}\ P(y \mid x)$$

직접 **y^**를 찾으려면 언어 모델로 가능한 **모든** 시퀀스를 평가해야 함.

>> 너무 오래 걸림. 따라서, **근사적인 방법**을 사용.

5.2 그리디 서치 디코딩

위의 근사적인 방법을 알아보고 고품질 텍스트를 생성하는 더 복잡한 알고리즘 구축하기

▫그리디 서치 방법

연속적인 모델 출력에서 이산적인 토큰을 얻는 가장 간단한 디코딩 방법은

각 타임스텝에서 확률이 가장 높은 토큰은 **greedily**(탐욕적)으로 선택하는 것

$$\hat{y}_t = \underset{y_t}{\operatorname{argmax}}\ P(y_t \mid y_{<t}, x)$$

그리드 서치 구현하기

▼ 언어 모델링 헤더를 가진 15억 개 파라미터의 GPT-2 로드하기

```
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

# 코랩의 경우 gpt2-xl을 사용하면 메모리 부족 에러가 발생합니다.
# 대신 "gpt2" 또는 "gpt2-large"로 지정하거나 코랩 프로를 사용하세요.
device = "cuda" if torch.cuda.is_available() else "cpu"
model_name = "gpt2"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name).to(
```

📢 🤗 트랜스포머스는 GPT-2 같은 자기회귀 모델을 위해 `generate()` 함수를 제공하지만

작동 방식을 이해하기 위해 **직접** 디코딩 메소드 구현하기

▼ 그림 5-3의 텍스트 생성하기

```
import pandas as pd

# “Transformers are the”를 입력 프롬프트로 사용
input_txt = "Transformers are the"
# 토크나이저 객체를 사용해 input_txt 토큰화 중. 출력은 파이토치 텐서로
#input_ids는 토큰화된 입력 시퀀스인 것임.
input_ids = tokenizer(input_txt, return_tensors="pt")["input_
iterations = []

# 여덟 번의 타임스텝 동안 디코딩 수행
n_steps = 8
```

```
# 대안을 시각적으로 보여주기 위해 타임스텝마다 확률이 가장 높은 토큰 다섯
choices_per_step = 5

# 그래디언트 계산 수행하지마 > 메모리 아끼기
with torch.no_grad():
  for _ in range(n_steps):
        # 각 반복마다 정보를 저장할 딕셔너리 생성
    iteration = dict()
        # 현재 반복에서 모델에 입력으로 주어지는 텍스트를 디코딩하여 저장
    iteration["Input"] = tokenizer.decode(input_ids[0])
        # 모델에 입력 텐서를 주고 출력(모델의 예측)을 계산
    output = model(input_ids=input_ids)

    # 첫 번째 배치의 마지막 토큰의 로짓을 선택해 소프트맥스를 적용
        # [0, -1, :]는 출력 텐서의 마지막 시퀀스의 모든 토큰에 대한 로
    next_token_logits = output.logits[0, -1, :]
    next_token_probs = torch.softmax(next_token_logits, dim=
    sorted_ids = torch.argsort(next_token_probs, dim=-1, desc

    # 가장 높은 확률의 토큰을 저장
    for choice_idx in range(choices_per_step):
      token_id = sorted_ids[choice_idx]
      token_prob = next_token_probs[token_id].cpu().numpy()
      token_choice = (
          f"{tokenizer.decode(token_id)} ({100 * token_prob:
      )
      iteration[f"Choice {choice_idx+1}"] = token_choice

    # 예측한 다음 토큰을 입력해 추가
    input_ids = torch.cat([input_ids, sorted_ids[None, 0, Non
    iterations.append(iteration)

pd.DataFrame(iterations)
```

- 각 타입스텝에서 프롬프트의 마지막 토큰에 대한 로짓을 선택하고 소프트맥스를 적용해 확률 분포 얻기. 그다음 확률이 가장 높은 토큰을 다음 토큰으로 선택하고 입력 시퀀스에 추가한 후에 이 과정을 다시 반복.

	Input	Choice 1	Choice 2	Choice 3	Choice 4	Choice 5
0	Transformers are the	most (9.76)	same (2.94)	only (2.87)	best (2.38)	first (1.77)
1	Transformers are the most	common (22.90)	powerful (6.88)	important (6.32)	popular (3.95)	commonly (2.14)
2	Transformers are the most common	type (15.06)	types (3.31)	form (1.91)	way (1.89)	and (1.49)
3	Transformers are the most common type	of (83.13)	in (3.16)	. (1.92)	, (1.63)	for (0.88)
4	Transformers are the most common type of	particle (1.55)	object (1.02)	light (0.71)	energy (0.67)	objects (0.66)
5	Transformers are the most common type of particle	. (14.26)	in (11.57)	that (10.19)	, (9.57)	accelerator (5.81)
6	Transformers are the most common type of parti...	They (17.48)	₩n (15.19)	The (7.06)	These (3.09)	In (3.07)
7	Transformers are the most common type of parti...	are (38.78)	have (8.14)	can (7.98)	're (5.04)	consist (1.57)

: 각 스텝에서 가능한 다른 문장도 볼 수 있는데, 이는 텍스트 생성의 **반복적인 특징**을 보여준다

예측하는데 한 번의 정방향 패스로 충분한 시퀀스 분류 등의 작업과 달리,

텍스트 생성은 **한 번에 하나의 출력 토큰을 디코딩**함.

내장함수 generate() 사용하기

🤗 트랜스포머스에 내장된 함수를 사용해 더 복잡한 디코딩 방법 알아보기

▼ generate()

```
input_ids = tokenizer(input_txt, return_tensors="pt")["input_
output = model.generate(input_ids, max_new_tokens=n_steps, do
print(tokenizer.decode(output[0]))
```

- `do_sample=False` : 앞선 결과를 재현하기 위해 샘플링 끄기
- 생성 토큰의 개수를 `max_new_tokens` 매개변수로 지정

```
Transformers are the most common type of particle. They are
```

▼ OpenAI의 유니콘 기사 재현하기

```
# 긴 텍스트 시퀀스 생성을 위해 max_length에 큰 값을 지정
max_length = 128

input_txt = """In a shocking finding, scientist discovered \
a herd of unicorns living in a remote, previously unexplored
valley, in the Andes Mountains. Even more surprising to the \
researchers was the fact that the unicorns spoke perfect Engl
"""

input_ids = tokenizer(input_txt, return_tensors="pt")["input_
# genrate() 함수 이용하여 모델에게 입력 텐서를 주고 텍스트 생성 요청하기
output_greedy = model.generate(input_ids, max_length=max_leng
                               do_sample=False)
# 생성된 텍스트를 디코딩하여 출력
print(tokenizer.decode(output_greedy[0]))
```

```
In a shocking finding, scientist discovered a herd of unicorns living in a
remote, previously unexplored valley, in the Andes Mountains. Even more
surprising to the researchers was the fact that the unicorns spoke perfect
English.


"The unicorns were very intelligent, and they were very intelligent," said Dr.
David S. Siegel, a professor of anthropology at the University of California,
Berkeley. "They were very intelligent, and they were very intelligent, and they
were very intelligent, and they were very intelligent, and they were very
intelligent, and they were very intelligent, and they were very intelligent, and
they were very
```

아니 이쁘게 잘 나오는뎅... 안나온다 햇느ㅡㄴ데.....

```
In a shocking finding, scientist discovered a herd of unicorns living in a
remote, previously unexplored valley, in the Andes Mountains. Even more
surprising to the researchers was the fact that the unicorns spoke perfect
English.

The researchers, from the University of California, Davis, and the University of
Colorado, Boulder, were conducting a study on the Andean cloud forest, which is
home to the rare species of cloud forest trees.

The researchers were surprised to find that the unicorns were able to
communicate with each other, and even with humans.

The researchers were surprised to find that the unicorns were able
```

: 원래라면 이렇게 여기서 그리디 서치 디코딩의 주요 단점이 드러남

📢 그리디 서치 디코딩의 단점

- 그리디 서치 알고리즘은 반복적인 출력 시퀀스를 생성하는 경향이 있어서

  뉴스 기사로는 적절하지 않음. 이는 그리드 서치 알고리즘의 보편적인 문제로

  최적의 솔루션을 만들기 어려움.

- 디코딩 측면에서 보면, 확률이 높은 단어가 확률이 낮은 단어보다 먼저 등장하기

  때문에 전체적으로 확률이 높은 단어 시퀀스를 생성하지 못하기도 함.

NOTE_ 그리디 서치 디코딩은 다양성이 필요한 텍스트 생성 작업에는 거의 사용되지 않지만, 결정적이고 사실적으로 정확한 출력이 필요한 수식 등의 짧은 문장 생성에는 유용합니다.[5] 이런 작업을 위해 줄바꿈이 있는 "5 + 8 => 13 \n 7 + 2 => 9 \n 1 + 0 =>" 같은 형식의 입력 프롬프트를 제공해 GPT-2의 조건부 생성을 제어할 수 있습니다.

5.3 빔 서치 디코딩 (인기)

빔 서치는 각 스텝에서 확률이 가장 높은 토큰을 디코딩하는 대신,

확률이 가장 높은 상위 b개의 다음 토큰을 추적한다.

여기서 b는 빔 또는 불완전 가설의 개수.

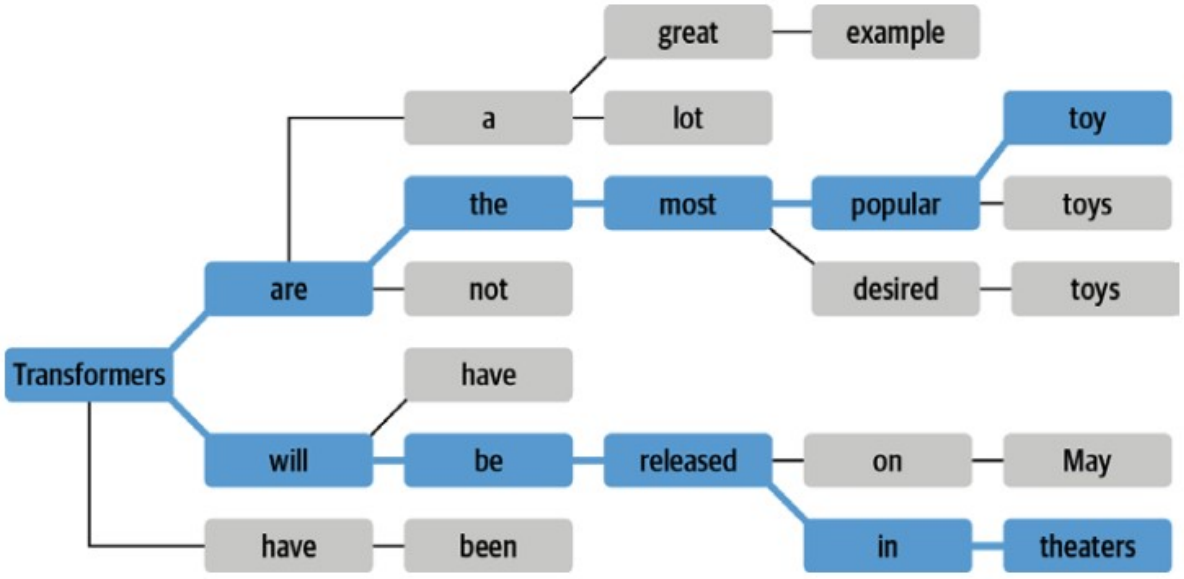


그림 5-4 b=2인 빔 서치

다음 빔 세트는 기존 세트에서 가능한 모든 다음 토큰을 확장하고 확률이 가장 높은

b개의 확장을 선택해 구성한다. 이 과정은 최대 길이나 EOS 토큰에 도달할 때까지 **반복**된다.

확률이 가장 높은 시퀀스는 **로그 확률**을 따라 b개 빔의 **순위**를 매겨 선택된다.

❓ **왜 확률이 아니라 로그 확률을 사용해 시퀀스 점수를 매길까?**

시퀀스 전체 확률 **P(y1, … , yt | x)**을 계산하려면 조건부 확률 **P(yt | y<t, x)의 곱**을 계산해야 하기 때문이다. 각 조건부 확률이 일반적으로 [0, 1] 범위 안에 속한 **작은 값**이므로 이를 곱해 얻은 전체 확률은 **언더플로**가 쉽게 발생한다.

즉, 컴퓨터가 이 계산의 결과를 더 이상 정확하게 표현할 수 없다는 뜻.

▼ 예시

```
#  t = 1024개의 토큰으로 이루어진 시퀀스에서 각 토큰의 확률이 0.5라 가정
0.5 ** 1024
```

: 이 시퀀스의 전체 확률은 매우 작은 수가 됨

```
5.562684646268003e-309
```

이는 수치적으로 불안정해 **언더플로**가 발생하지만, **로그 확률**을 계산하면 이를 피할 수 있다

▫ 로그 확률

결합 확률과 **조건부 확률**에 로그를 적용하면 로그의 곱셈 규칙에 따라 아래 식 획득

$$\log P(y_1, \ldots y_t \mid x) = \sum_{t=1}^{N} \log P(y_t \mid y_{<t}, x)$$

사실 그냥 앞서 본 조건부 확률 곱셈이 **로그 확률의 덧셈**으로 바뀐 것 뿐임.

▼ 로그 확률 예시

```
# 이전 예를 그대로 적용
import numpy as np

sum([np.log(0.5)] * 1024)
```

- 이 방식은 다루기 쉽고 더 작은 수에도 적용 가능함
-

서치 알고리즘은 **상대적 확률**만 비교하면 되므로 **로그 확률**을 사용해서도 비교 가능

▫ 로그 확률 비교하기

그리디 서치와 빔 서치로 생성한 텍스트의 로그 확률을 계산해

빔 서치가 전체 확률을 향상하는지 확인하기

▼ **하나의 토큰에 대한 로그 확률 계산 함수 만들기**

```
import torch.nn.functional as F

def log_probs_from_logtis(logits, labels):
  # 입력된 로짓에 대해 로그 소프트맥스 계산
  logp = F.log_softmax(logits, dim=-1)

  # 각 데이터 포인트에 대한 정답 라벨의 로그 확률을 추출
    # unsqueeze(2)를 사용하여 labels 텐서의 차원을 하나 확장하여 함수
  # 그 후 squeeze(-1)를 사용하여 결과 텐서의 크기를 줄임
  logp_label = torch.gather(logp, 2, labels.unsqueeze(2)).squ
  return logp_label
```

- `logits` : 모델의 출력값으로 소프트 맥스 함수를 통과하기 전의 값

  로짓은 확률 분포를 표현하는 값
- `labels` : 각 데이터 포인트에 대한 실제 라벨을 포함하는 텐서
- `torch.gather()` : 주어진 축(여기서는 2번째)에서 특정 인덱스에 해당하는 값을 추출

> 📢 🤗 트랜스포머스 모델은 입력 토큰이 주어지면 다음 토큰에 대한 정규화되지 않은 로짓 반환. 따라서 먼저 로짓을 정규화해서 시퀀스의 각 토큰을 위해
>
> 전체 어휘사전에 대한 확률 분포 만들고 시퀀스에 있는 토큰 확률**만** 선택

이 함수는 **하나**의 토큰에 대한 로그 확률을 제공하므로,

시퀀스 전체 로그 확률을 얻으려면 각 토큰의 로그 확률을 **더해야 함**

▼ 시퀀스의 전체 로그 확률 얻기

```
def sequence_logprob(model, labels, input_len=0):
  with torch.no_grad():
    output = model(labels)
    log_probs = log_probs_from_logtis(
        output.logtis[:, :-1, :], labels[:, 1:])

        # input_len 이후의 모든 로그 확률을 합산하여 시퀀스의 전체 로그
    seq_log_prob = torch.sum(log_prob[:, input_len:])
  return seq_log_prob.cpu().numpy()
```

- `input_len=0` : 이 값은 출력에서 무시할 초기 입력 토큰의 수를 의미.

  >> 모델이 입력 시퀀스를 생성하지 않았으므로 입력 시퀀스의 로그 확률은 무시.

📢 로짓과 레이블의 정렬이 중요함.

모델은 다음 토큰을 예측하기 때문에 첫 번째 레이블에 대한 로짓을 얻지 못함.

또 마지막 로짓에 대한 정답이 없기 때문에 마지막 로짓은 필요하지 않음

▼ OpenAI 프롬프트에서 그리디 서치로 만든 시퀀스의 로그 확률 계산

```
# 함수 호출하여 output_greedy 시퀀스에 대한 로그 확률 계산
# input_len은 입력 시퀀스의 길이로 설정
logp = sequence_logprob(model, output_greedy, input_len=len(i

# 시퀀스를 디코딩하여 텍스트로 출력
print(tokenizer.decode(output_greedy[0]))

print(f"\n로그 확률: {logp:.2f}")
```

```
In a shocking finding, scientist discovered a herd of unicorns living in a
remote, previously unexplored valley, in the Andes Mountains. Even more
surprising to the researchers was the fact that the unicorns spoke perfect
English.


"The unicorns were very intelligent, and they were very intelligent," said Dr.
David S. Siegel, a professor of anthropology at the University of California,
Berkeley. "They were very intelligent, and they were very intelligent, and they
were very intelligent, and they were very intelligent, and they were very
intelligent, and they were very intelligent, and they were very intelligent, and
they were very

로그 확률: -83.32
```

▼ 빔 서치로 생성한 시퀀스 로그 확률

```
# 빔 서치 활성화를 위해 generate() 함수에 num_beams 매개변수에 빔 개수
output_beam = model.generate(input_ids, max_length=max_length,
                             do_sample=False)

# 함수 호출하여 output_beam 시퀀스에 대한 로그 확률 계산
# input_len은 입력 시퀀스의 길이로 설정
logp = sequence_logprob(model, output_beam, input_len=len(input_ids[0]))

# 시퀀스를 디코딩하여 텍스트로 출력
print(tokenizer.decode(output_beam[0]))
```

- 빔 크기가 클수록 결과가 더 좋을 가능성이 높지만

  각 빔에 대해 병렬적으로 시퀀스를 생성하므로 생성 과정이 느려진다

```
In a shocking finding, scientist discovered a herd of unicorns living in a
remote, previously unexplored valley, in the Andes Mountains. Even more
surprising to the researchers was the fact that the unicorns spoke perfect
English.


The researchers, from the University of California, San Diego, and the
University of California, Santa Cruz, found that the unicorns were able to
communicate with each other in a way that was similar to that of human speech.


"The unicorns were able to communicate with each other in a way that was similar
to that of human speech," said study co-lead author Dr. David J.

로그 확률: -78.34
```

단순한 그리디 서치보다 빔 서치에서 더 높은 로그 확률 획득 (높을수록 좋다)

▫ n-그램 페널티

📢 빔 서치도 **텍스트가 반복**되는 문제가 있음

이를 해결하기 위해 no_repeat_ngram_size 매개변수로 **n-그램 페널티**를 부과한다.

그러면 지금까지의 n-그램을 추적해 이전에 보았던 n-그램을 생성하면

다음 토큰 확률이 0이 된다

▼ n-그램 페널티

```
# 생성된 시퀀스에서 반복되는 n-gram을 허용하지 않는 크기가 2라는 것
output_beam = model.generate(input_ids, max_length=max_lengt
                                do_sample=False, no_repeat_ngra
logp = sequence_logprob(model, output_beam, input_len=len(in
print(tokenizer.decode(output_beam[0]))
print(f"\n로그 확률: {logp:.2f}")
```

```
In a shocking finding, scientist discovered a herd of unicorns living in a
remote, previously unexplored valley, in the Andes Mountains. Even more
surprising to the researchers was the fact that the unicorns spoke perfect
English.


The researchers, from the University of California, San Diego, and the National
Science Foundation (NSF) in Boulder, Colorado, were able to translate the words
of the unicorn into English, which they then translated into Spanish.

"This is the first time that we have translated a language into an English
language," said study co-author and NSF professor of linguistics and
evolutionary biology Dr.

로그 확률: -101.87
```

: 반복적인 텍스트 생성을 막고 점수는 더 낮아졌지만 텍스트가 일관성을 유지한다

- n - 그램 페널티를 사용한 빔 서치는 확률이 높은 토큰에 초점을 맞추는 빔 서치와 반복을 줄이는 n - 그램 페널티의 균형을 잡는 좋은 방법임

- 사실적인 정확성을 요하는 요약, 기계 번역 같은 애플리케이션에 널리 사용

5.4 샘플링 방법

분야에 국한되지 않는 잡담이나 기사처럼 정확성보다 **다양성**이 중요할 때,
샘플링을 사용해 다양성을 늘리면서 반복을 줄이는 방법이 있음

가장 간단한 샘플링 방법은 각 타임스템 내에
모델이 출력한 **전체 어휘사전의 확률 분포**에서 **랜덤**하게 **샘플링**하는 것이다

$$P(y_t = w_i \mid y_{<t},\ x) = \text{softmax}(z_{t,i}) = \frac{\exp(z_{t,i})}{\sum_{j=1}^{|V|} \exp(z_{t,j})}$$

여기서 **| V |**는 **어휘사전의 크기**를 나타낸다. 소프트 맥스 함수를 적용하기 전에
로짓의 스케일을 조정하는 **온도 파라미터 T**를 추가하면 출력의 다양성이 쉽게 제어된다

$$P(y_t = w_i \mid y_{<t},\ x) = \frac{\exp(z_{t,i}/T)}{\sum_{j=1}^{|V|} \exp(z_{t,j}/T)}$$

▫ 토큰 확률에 미치는 온도의 영향

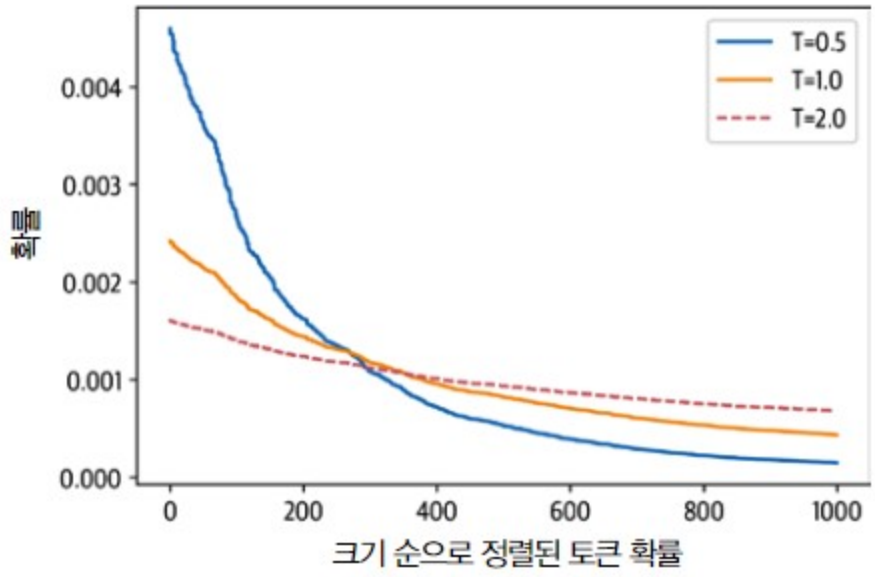


그림 5-5 세 가지 온도에서 랜덤하게 생성한 토큰 확률 분포

T 값을 바꾸면 확률 분포의 형태가 제어됨.

- T ≤ 1 일 때 이 분포는 원점 근처에서 정점에 도달하고 드문 토큰을 억제한다.
- T ≥ 1 일 때 이 분포는 평평해지고 각 토큰의 확률이 동일해진다.

▼ 온도가 생성되는 텍스트에 미치는 영향 알아보기 : 온도가 높을 때

```
# generate() 함수의 temperature 매개변수를 T=2로 지정해 샘플링 하기
output_temp = model.generate(input_ids, max_length=max_lengt
                             temperature=2.0, top_k=0)
print(tokenizer.decode(output_temp[0]))
```

- `top_k = N` : 모델이 다음 단어를 예측할 때 예측한 확률 값 중

  상위 N개의 단어만 고려하도록 하는 것을 의미

- 따라서 `top_k=0`은 모델이 예측한 확률 값 중 어떤 제한도 없이 **모든 단어를 고려**

```
In a shocking finding, scientist discovered a herd of unicorns living in a
remote, previously unexplored valley, in the Andes Mountains. Even more
surprising to the researchers was the fact that the unicorns spoke perfect
English.


Leave significant community history: actors assault MH buried plight felt gamb
Light SalSab homeworld Background info Lou forcibly taught its upon leaving
Sahulates Wally Green Chevroletasks god Tal Defensive Mobrequently
handsetlaneober ease DodCh Prayer button during GreekingSolution Hindu
occupational Oman contracted throwing Barnett likes friendly Rabbitg texts on
trending3 Creedater Conversion twelve Bluebirds particlesrhcz Thor Dale Dayton
Dennis agency threat encounters understands Kuro licences
```

온도가 높으면 횡설수설에 가까운 텍스트가 생성된다

: 드문 토큰이 강조되어 모델이 이상한 문법을 만들고 가짜 단어를 만들어냈음

▼ 온도가 생성되는 텍스트에 미치는 영향 알아보기 : 온도가 낮을 때

```
out_temp = model.generate(input_ids, max_length=max_length, 
                          temperature=0.5, top_k=0)
print(tokenizer.decode(output_temp[0]))
```

```
In a shocking finding, scientist discovered a herd of unicorns living in a
remote, previously unexplored valley, in the Andes Mountains. Even more
surprising to the researchers was the fact that the unicorns spoke perfect
English.


The scientist, who was not involved in the research, told the newspaper that the
unicorns spoke perfect English and that they were among the first to speak it.


"They were so clever, so clever that it was difficult to believe that they were
speaking English," he said.


The team found that the unicorns were not only able to communicate perfectly
with humans, but also with their human
```

이 텍스트가 더 일관성 있음.

: 항상 일관성(낮은 온도)과 다양성(높은 온도)의 균형점이 있기 때문에 상황에 맞게 조정하기

5.5 탑-k 및 뉴클리어스(탑-p) 샘플링

▫ 배경

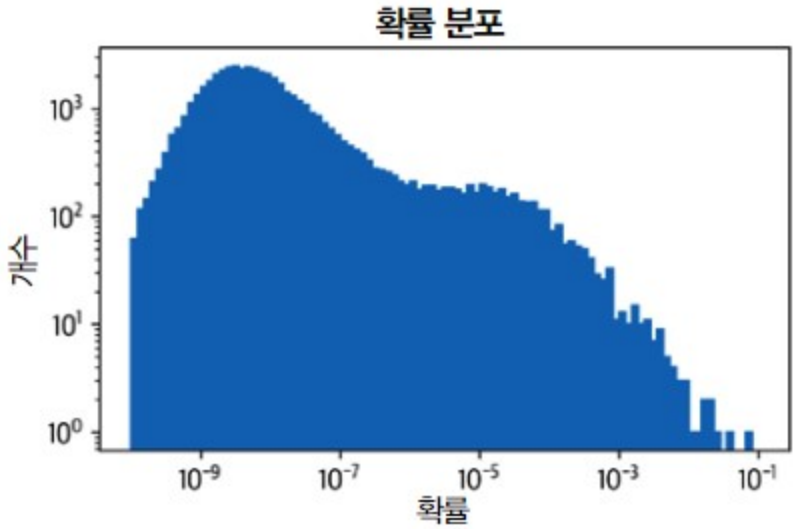
📢 일관성과 다양성의 균형을 조정하는 방법으로는 **어휘사전 분포를 잘라내는 방법**이 있다.

이 방법은 온도와 함께 다양성을 자유롭게 조정하지만, 더 제한된 범위에서

문맥상 매우 이상한 단어(즉, 확률이 낮은 단어)를 제외한다.

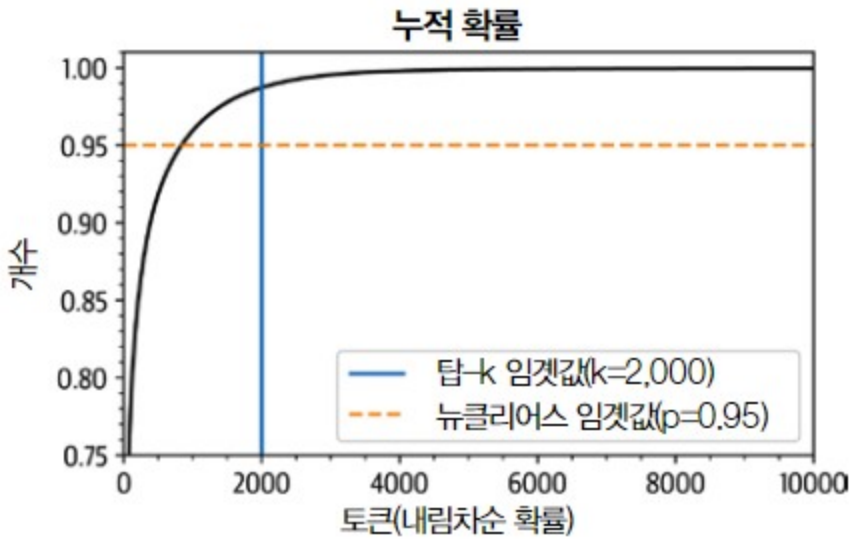
대표적인 방법으로는 탑-k 샘플링과 뉴클리어스 샘플링이 있다.

두 샘플링은 모두 각 타임스텝에서 샘플링에 사용할 토큰의 개수를 줄인다는 개념에 기초



T = 1에서 다음 토큰 예측의 확률 분포 히스토그램

- 10^(-8) 근처가 최고 정점이고 10^(-4) 근처에 조금 더 작은 제 2의 정점이 있음.
- 그 다음 확률이 10^(-2)와 10^(-1) 사이의 확률을 가진 토큰에서 급격히 줄어듦
- 그래프를 보면 확률이 가장 높은 토큰(10^(-1)에서 동떨어져 있는 막대)을 선택할 확률은 1/10



내림차순 확률로 토큰을 정렬한 후, 처음 10000개 토큰의 누적 합을 계산

- 이 곡선은 확률이 높은 토큰 중 하나를 선택할 확률을 나타냄

  예를 들어 확률이 가장 높은 1,000개 토큰 중 하나를 선택할 확률은 약 96%
- 확률이 빠르게 90%를 상회하지만 수천 개의 토큰이 지나야 100%에 가깝게 수렴 중임

📢 그래프를 보면 하위 8,000개 토큰 중 하나를 선택할 확률은 약 1/100

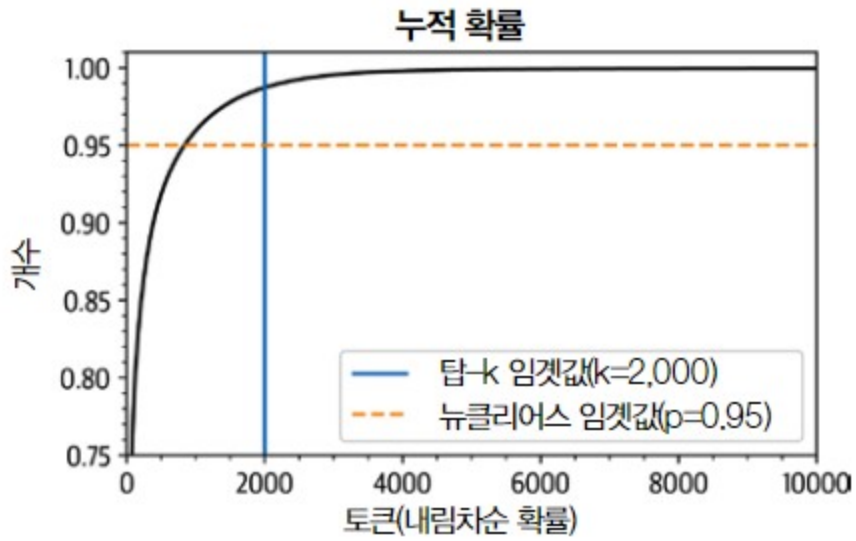
이는 텍스트를 생성할 때 토큰마다 한 번씩 샘플링하기 때문에 중요함

겨우 0.01 이더라도 수백 번 샘플링하게 되면 언젠가 희귀한 토큰을 선택할 가능성이 있음

이런 배경에서 탑-k와 탑-p 샘플링이 등장

▫ 탑-k 샘플링

탑-k 샘플링은 확률이 가장 높은 k개 토큰에서만 샘플링해서 확률이 낮은 토큰을 피함



내림차순 확률로 토큰을 정렬한 후, 처음 10000개 토큰의 누적 합을 계산

즉, 그림에서 탑-k 샘플링은 수직선을 정의하고 왼쪽에 있는 토큰에서만 샘플링을 하는 것

▼ 탑-k 샘플링하기

```
output_topk = model.generate(input_ids, max_length=max_lengt
                             top_k=50)
print(tokenizer.decode(output_topk[0]))
```

- `generate()` 함수는 매개변수 `top_k`를 제공

```
In a shocking finding, scientist discovered a herd of unicorns living in a
remote, previously unexplored valley, in the Andes Mountains. Even more
surprising to the researchers was the fact that the unicorns spoke perfect
English.


The researchers found that the only known language used by the animals was
Bengali. Researchers are unsure whether that language can be interpreted as
Chinese, but it would explain their strange speech patterns.


"The unicorns have the 'perfect' English, and we are not sure that that's what
they all say. The way they speak, we don't know. They don't say 'hello', we
```

지금까지 만든 텍스트 중 젤 갠챤음

▫ 탑-p 샘플링

📢 **k 값을 어떻게 정해야 할까?**

k값은 수동으로 선택해야 하고

실제 출력 분포에 상관없이 시퀀스의 각 선택에 동일하게 적용됨

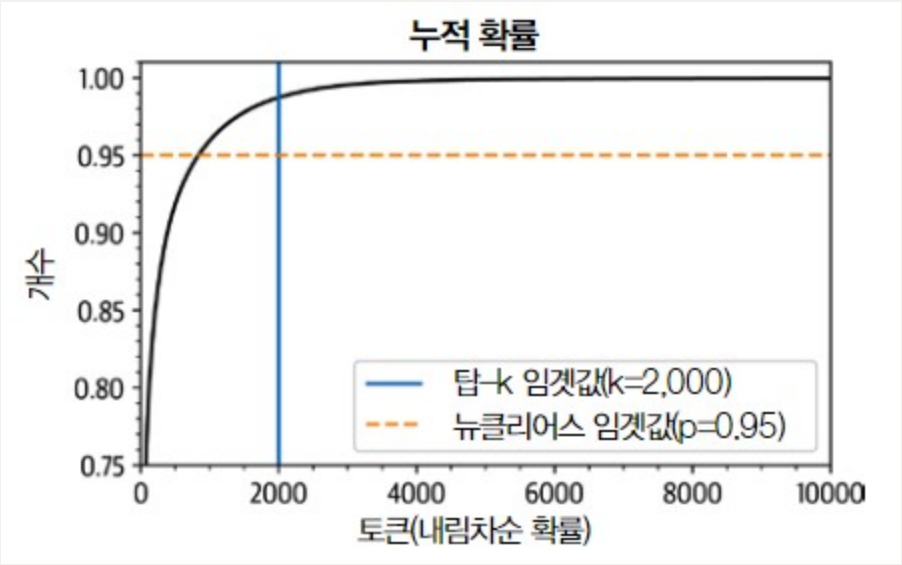
따라서 고정된 컷오프는 구릴 때도 있음 >> **동적인 컷오프** 적용하기

탑-p 샘플링은 고정된 컷오프 값을 선택하지 않고 어디서 컷오프 할 지 조건을 지정한다

: 이 조건은 선택한 특정 **확률 질량**에 도달할 때이다

▼ 예시

📢 예를 들어 이 값을 95%로 지정했다고 하자.



내림차순 확률로 토큰을 정렬한 후, 처음 10000개 토큰의 누적 합을 계산

그 다음 확률에 따라 내림차순으로 모든 토큰을 정렬하고

선택한 토큰의 확률 값이 95%에 도달할 때까지 이 리스트의 맨 위부터

토큰을 하나씩 추가한다. 그림을 보면 p 값은 누적 확률 그래프의 수평선에 해당.

이 수평선의 **아래**에 있는 토큰에서만 샘플링을 하게 된다.

출력 분포에 따라 (확률이 매우 높은) 하나의 토큰이 될 수도 있고

(확률이 비슷한) 백 개의 토큰이 될 수도 있음.

▼ 탑-p 샘플링하기

```
output_topp = model.generate(input_ids, max_length=max_lengt
                             top_p=0.90)
print(tokenizer.decode(output_topp[0]))
```

- `generate()` 함수는 매개변수 `top_p` 도 제공

```
In a shocking finding, scientist discovered a herd of unicorns living in a
remote, previously unexplored valley, in the Andes Mountains. Even more
surprising to the researchers was the fact that the unicorns spoke perfect
English.


"Our study is the first documented instance of a large-scale population-based
study of the diversity of languages that inhabit a region," the researchers
wrote in a release.

One of the most striking findings was the fact that the unicorns were able to
communicate with humans. The researchers discovered that the unicorns learned to
identify specific messages with their tongues.

"When the unicorns shared these
```

탑-p 샘플링도 일관성 있는 텍스트를 생성함

두 샘플링 방법을 연결하면 양 쪽의 장점을 모두 취할 수 있다

📢 ex) top_k = 50, top_p = 0.9로 지정하면

확률이 가장 높은 50개 토큰에서 확률 질량이 90%인 토큰 선택!

5.6 어떤 디코딩 방법이 최선일까?

최선의 방법은 주어진 텍스트 생성 작업의 특성에 따라 다르다

1. 수식이나 질문 답변 같은 정밀 작업을 수행하는 모델

온도를 낮추거나 확률이 가장 높은 답을 보장하기 위해 빔 서치와 함께 그리디 서치같은 방법 사용

2. 길고 창의적인 텍스트를 생성하는 모델

샘플링 방법으로 바꾸고 온도를 올리거나 탑-k와 탑-p 샘플링을 혼합해 사용