3주차 - 5.1~5.7절

5장 텍스트 생성

5.1 일관성 있는 텍스트 생성의 어려움

- 텍스트 생성의 어려움:
  1. 디코딩은 반복적으로 토큰을 생성하므로, forward pass 한번 통과하는 것보다 계산량이 많다.
  2. 생성된 텍스트의 품질과 다양성은 디코딩 방법과 하이퍼파라미터에 따라 달라짐.
- 디코더 기반 모델은 생성된 토큰들을 바탕으로 다음 토큰을 에측하는 방식으로 사전 훈련됨.
  - 조건부 확률의 곱으로 나타남:

$$P\left(y_1, \ldots, y_t \middle| \mathbf{x}\right) = \prod_{t=1}^{N} P\left(y_t \middle| y_{<t}, \mathbf{x}\right)$$

  - 다음 토큰 w_i에 대한 확률 분포:

$$P\left(y_t = w_i \middle| y_{<t}, \mathbf{x}\right) = \text{softmax}\left(z_{t,i}\right)$$

  - 확률이 가장 높은 시퀀스 선택:

$$\hat{\mathbf{y}} = \underset{\mathbf{y}}{\text{argmax}}\ P\left(\mathbf{y} \middle| \mathbf{x}\right)$$

    - $y$hat은 전체 시퀀스

5.2 그리디 서치 디코딩

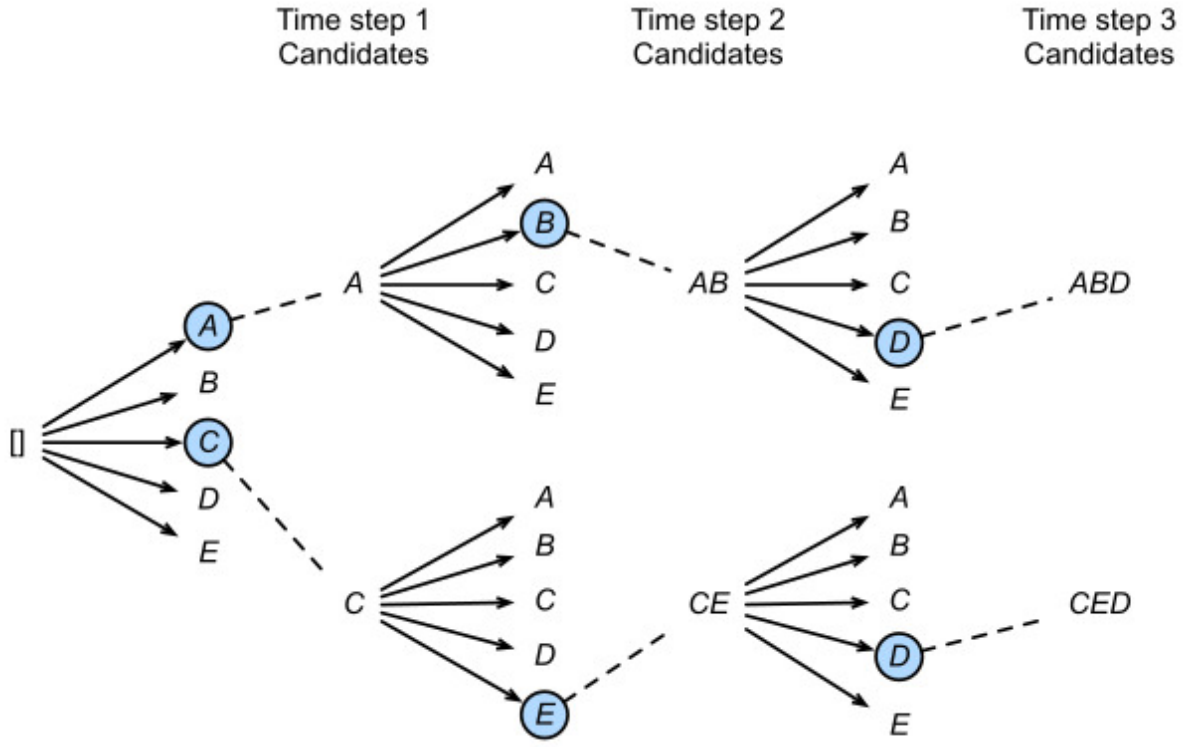
$$\hat{y}_t = \underset{y_t}{\operatorname{argmax}} P(y_t | y_{<t}, \mathbf{x})$$

```
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
import pandas as pd

device = "cuda"
model_name = 'gpt2-xl'
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name).to(d

input_text = 'Transformers are the'
input_ids = tokenizer(input_text, return_tensors='pt')['input_
iterations = []
n_steps = 8
choices_per_step = 5

with torch.no_grad():
    for _ in range(n_steps):
        iteration = dict()
        iteration['Input'] = tokenizer.decode(input_ids[0])
        output = model(input_ids=input_ids)
        next_token_logits = output.logits[0, -1, :]
        next_token_probs = torch.softmax(next_token_logits, d
        sorted_ids = torch.argsort(next_token_probs, dim=-1,
        for choice_idx in range(choices_per_step):
            token_id = sorted_ids[choice_idx]
            token_prob = next_token_probs[token_id].cpu().num
            token_choice = (
                f"{tokenizer.decode(token_id)} ({100 * token_
            )
```

```
            iteration[f'Choice {choice_idx + 1}'] = token_cho
        input_ids = torch.cat([input_ids, sorted_ids[None, 0,
        iterations.append(iteration)
```

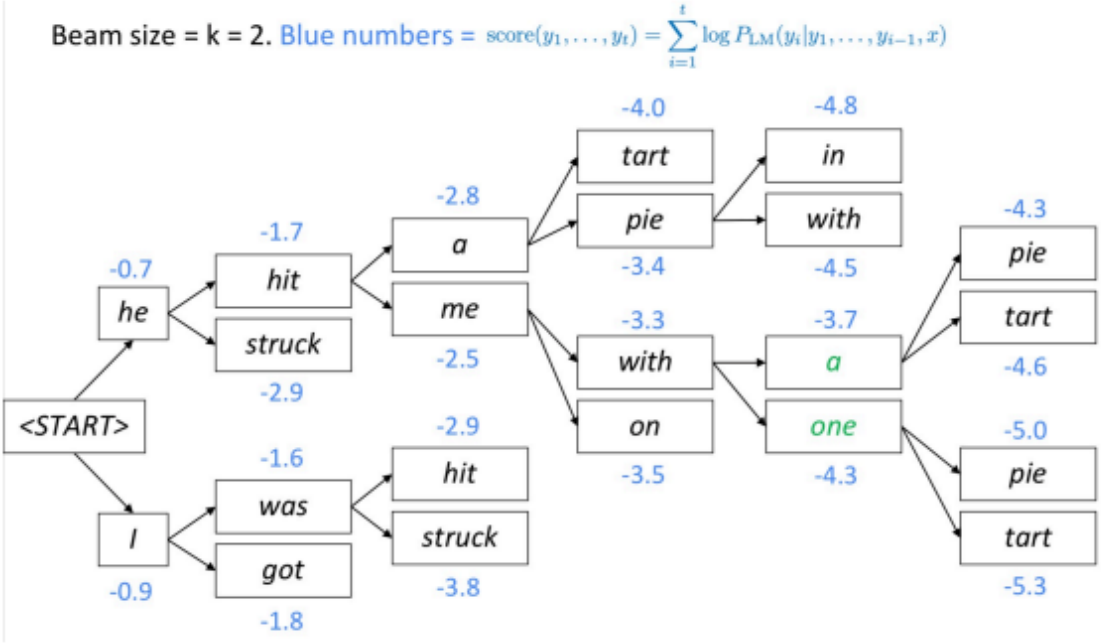
- 그리디 서치 디코딩의 단점:
  1. 반복적인 출력 시퀀스를 생성하는 경향이 있음
  2. 토큰을 하나씩 생성할 때마다 매 iteration에서 확률이 높은 단어가 먼저 등장하기 때문에, 전체적으로 확률이 높은 단어 시퀀스를 생성하지 못함.
     - 모델이 실수할 경우 돌이킬 수 없게 됨.
- 그리디 서치 디코딩이 유용한 생성:
  - 결정적이고 사실적으로 정확한 출력이 필요한 수식

5.3 빔 서치 디코딩



- 각 생성 스텝에서 확률이 가장 높은 상위 b개의 다음 토큰을 추적함
- b: beam 혹은 partial hypothesis
- beam search decoding 시에는 서로 다른 시점에 <EOS> 토큰이 생성이 되므로 각 hypothesis에서 <EOS> 토큰 만들 시 hypothesis를 종료함.

- 각 시점별로 유망한 빔 b개를 골라서 진행하는 방식
- 최종적으로 b개 중 가장 확률이 높은 선택을 함



$$score(y_1, ..., y_t) = logP_{LM}(y1, .., y_t|x)$$
$$= \sum_{i=1}^{t} logP_{LM}(y_i|y_1, ...y_{i-1}, x)$$

- 로그 확률을 사용하는 이유:
  - 각 조건부 확률이 0~1 사이의 값이기 때문에 반복해서 곱할 경우 underflow가 발생함.
  - underflow: 더 이상 정확하게 계산이 어려움.
  - 따라서 로그 확률을 이용해서 덧셈으로 바꿈. 이를 통해 수치적 불안정을 최소화함
- 생성된 토큰에 대한 로그 확률 계산하기:

```
import torch.nn.functional as F

def log_probs_from_logits(logits, labels):
    logp = F.log_softmax(logits, dim=-1)
```

```
    logp_label = torch.gather(logp, 2, labels, unsqueeze(2
    return logp_label
```

- `torch.gather` :
  - `torch.gather(input, dim, index, out=None, sparse_grad=False)`
  - input에서 dim차원 부분의 특정 index의 값만 추출하는 함수

```
def sequence_logprob(model, labels, input_len=0):
    with torch.no_grad():
        output = model(labels)
        log_probs = log_probs_from_logits(
            output.logits[:, :-1, :], labels[:, 1:])
        seq_log_prob = torch.sum(log_probs[:, input_len:])
    return seq_log_prob.cpu().numpy()
```

- `no_repeat_ngram_size`
  - 이전에 보았던 n 그램 생성 시 다음 토큰 확률이 0이 된다.
  - 반복을 줄이는 기능한다.

5.4 샘플링 방법

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

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

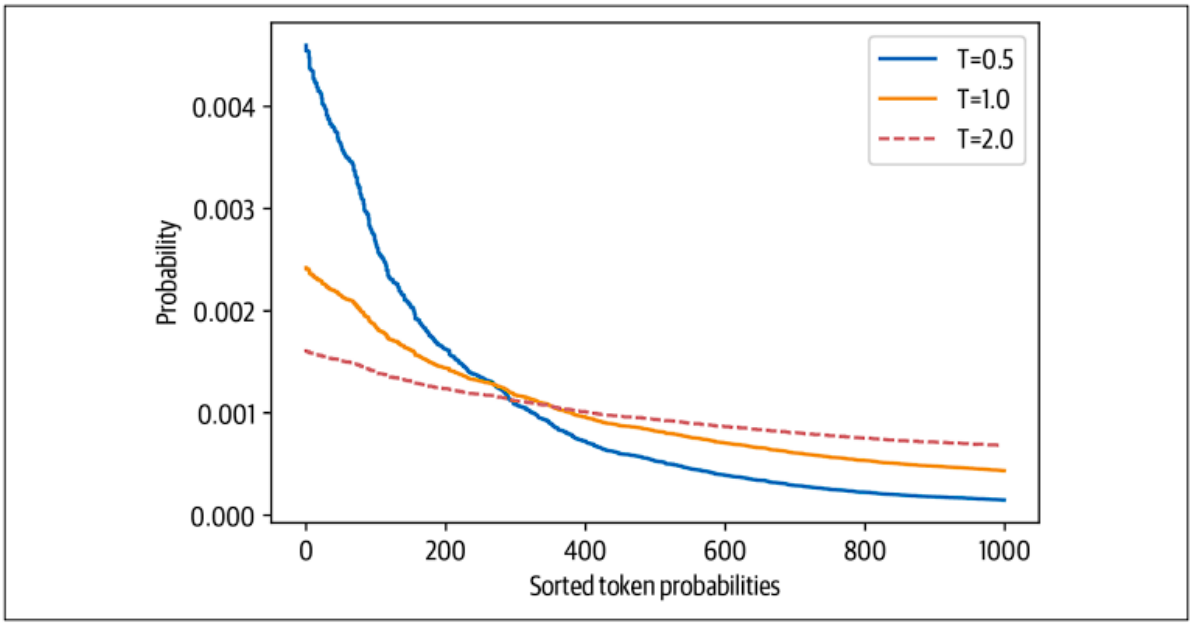
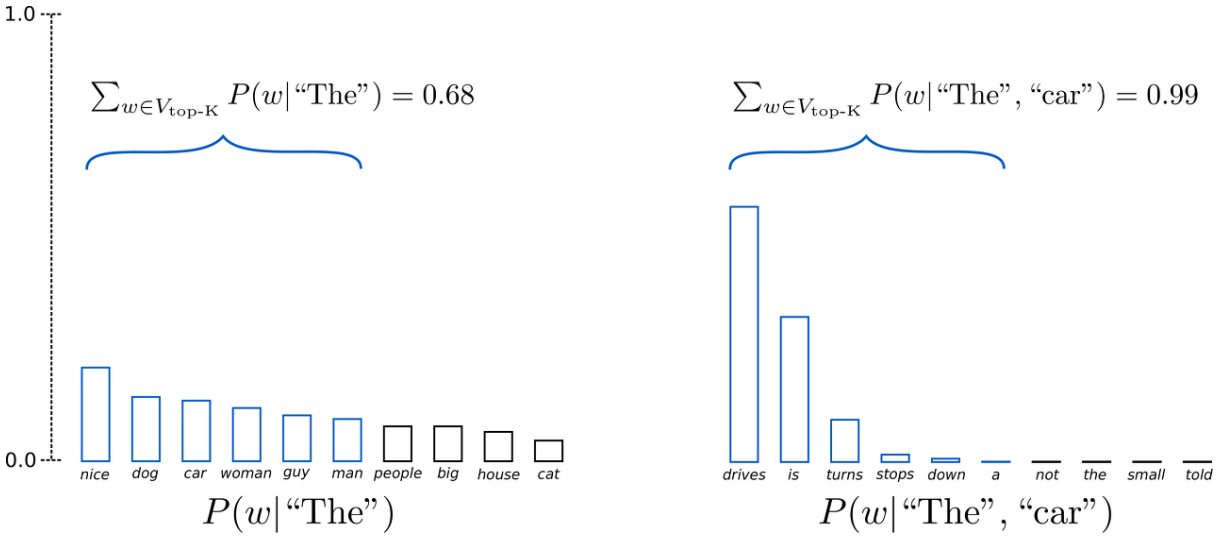


Figure 5-5. Distribution of randomly generated token probabilities for three selected temperatures

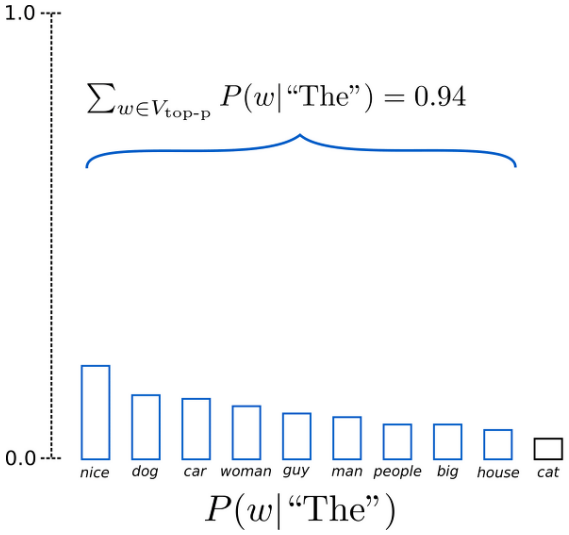
- temperature:
  - 0에 가까울수록 ㄴ자 형태로 변함. 확률이 높은 토큰의 확률이 더 높아짐
  - 커질수록 ㅡ자 형태로 변함. 확률이 낮은 토큰의 확률이 더 높아짐
  - top-p 샘플링과 관련 있는 것.

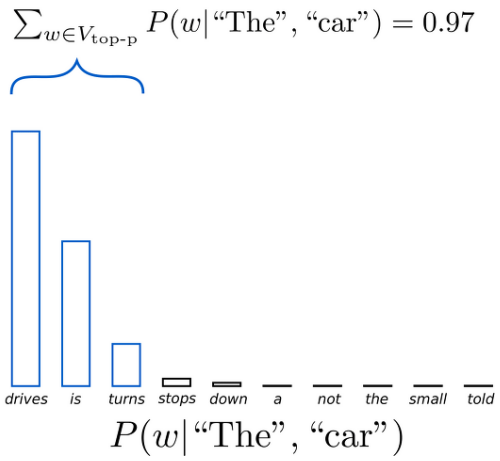
5.5 탑-k 및 뉴클리어스 샘플링



- top-k 샘플링
- 후보에 오를 만한 단어임에도 top-k에 들지 못할 수도 있음

- 확률이 낮은 단어여도 걸러지지 않을 수 있음





- top-p 샘플링 (뉴클리어스 샘플링)
- top-k 샘플링 보완