

언어 모델

단어 시퀀스에 확률을 부여하는 모델. 단어 시퀀스를 입력받아 해당 시퀀스가 얼마나 그럴듯한지 확률을 출력하는 모델. 따라서 한국어 말뭉치로 학습한 언어 모델은 자연스러운 한국어 문장에 높은 확률값을 부여함. 어떤 문장이 한국어스러운지 해당모델이 이해.

문장에서 l번째로 등장하는 단어를 w로 표현한다면, n개 단어로 구성된 문장이 해당 언어에서 등장할 확률, 즉 언어 모델의 출력은 다음 수식처럼 쓸 수 있음.

조건부 확률

$$\text{운전/난폭} = p(\text{난폭, 운전}) / p(\text{난폭})$$

‘난폭’이라는 단어와 ‘운전’이라는 단어가 동시에 나타날 결합 확률.

1. 순방향 언어모델.

임의의 단어 시퀀스가 해당 언어에서 얼마나 자연스러운지 이해하고 있는 언어모델을 구축. 그런데 조건부 확률의 정의에 따라 수식 3-4의 좌변과 우변이 같다는 사실을 알고 있으므로,

문장 앞부터 뒤로, 사람이 이해하는 순서대로 계산하는 모델을 순방향 언어모델이라고 한다. 헛, elmo같은 모델이 이런방식.

2. 역방향 언어모델.

문장 뒤부터 앞으로 계산하는 역방향 언어 모델. ELMO (embeddings from language Models)

$$P(w / \text{context})$$

이는 컨텍스트가 전제된 상태에서 특정단어(w)가 나타날 조건부 확률을 나타냄.

3. 마스크

마스크 언어모델은 학습 대상 문장에 빈칸을 만들어 놓고 해당 빈칸에 올 단어로 적절한 단어가 무엇일지 분류하는 과정 BERT가 대표적인 모델.

4. 스킵그램모델

단어 앞 뒤에 특정 범위를 두고, 이 범위내에 어떤 단어들이 올지 분류하는 과정으로 학습. 주변에 어제, 카페, 거기, 갔었어 하는 방식으로 나타날 확률을 각각 높입니다. 컨텍스트로 설정한 단어 주변에 어떤 단어들이 분포해 있는지. Word2Vec이 그 모델.

5. 언어모델의 유연성

실제로 최근에 제안되는 기법들은 프리트레이닝을 마친 딥러닝 계열 언어 모델을 바탕으로 할 때가 많습니다. 이 언어 모델의 최종 또는 중간 출력값을 가지고 다양한 테스트를 출력합니다. (출력값: 임베딩, 또는 리프레젠테이션)

6. 시퀀스 투 시퀀스

기계번역이란 어떤 언어의 토큰 시퀀스를 다른 언어의 토큰 시퀀스로 변환하는 과제임.

인코더 디코더구조

인코더는 소스 시퀀스의 정보를 압축해서 디코더로 보냄. 이를 인코딩

디코더는 인코더가 보내준 소스 시퀀스 정보를 받아서 타겟 시퀀스를 생성함.

트랜스포머의 최종 출력, 즉 디코더 출력은 타겟 언어의 어휘수만큼의 차원으로 구성된 벡터입니다. 이 벡터의 특징은 요솟(element)값이 모두 확률이라는 점,

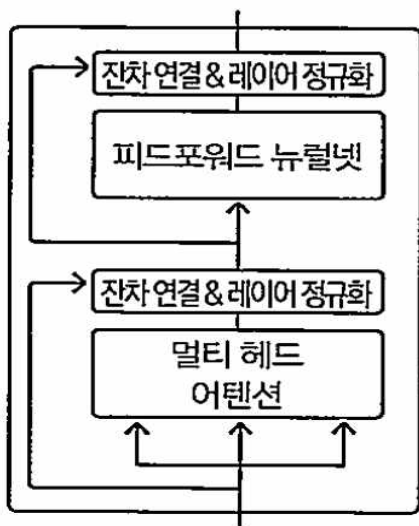
예를 들어 타겟 언어의 어휘가 총 3만개라고 가정해보면 디코더출력은 3만차원의 벡터, 3만개 각각의 확률이므로 0이상, 1이하의 값을 가지며 더하면 모두 1이 됩니다.

7. 트랜스포머 블록

다음 그림은 인코더 가운데 반복되는 요소를 떼어내 다시 나타낸 것이다.

이런 구조를 블록 또는 레이어라고 한다.

인코더는 이러한 블록 수십개를 쌓아서 구성합니다.



멀티헤드 어텐션
피드포워드 뉴럴 네트워크
잔차연결 및 레이어 정규화 등

3가지 요소로 구성되어 있다.

그림 3-15 트랜스포머 인코더 블록

디코더 쪽 블록의 구조도 인코더 블록과 본질적으로는 같다. 다만 마스크를 적용한 멀티 헤드 어텐션이 인코더와 다르다.

또 인코더가 보내 온 정보와 + 디코더 입력을 함께 이용해 멀티 헤드 어텐션을 수행하는 모듈이 추가되었습니다.

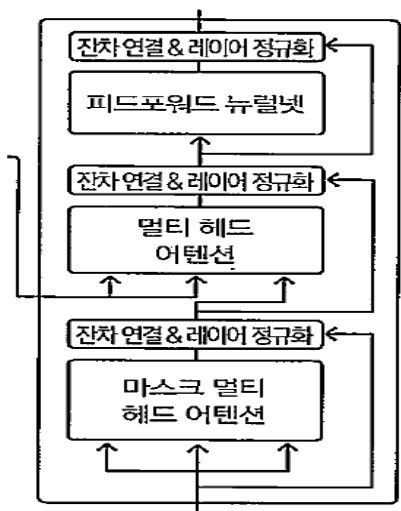


그림 3-16 트랜스포머 디코더 블록

인코더	디코더
소스 시퀀스 전체	타겟 시퀀스 전체
입력 임베딩	입력 임베딩
위치정보	위치정보
멀티헤드 어텐션	마스크 멀티헤드 어텐션
잔차연결 & 레이어 정규화	잔차연결 & 레이어 정규화
1. 피드포워드 뉴럴 네트워크	3. 멀티헤드 어텐션 (인코더에서 온 소스 추가)
2. 잔차연결 & 레이어 정규화	4. 잔차연결 & 레이어 정규화
	5. 피드포워드 뉴럴 네트워크
	6. 잔차연결 & 레이어 정규화
	7. 선형변환
	8. 소프트맥스
	9. 다음 토큰 확률 (타겟 시퀀스)

8. 셀프 어텐션 p77

시퀀스 입력에 의존하는 수행하는 기계학습방법의 일종. 시퀀스 요소 가운데 중요한 요소에 집중하고, 그렇지 않은 요소는 무시해 태스크 수행 성능을 끌어올림. 기계번역 과제에 처음 도입.

기계번역에 어텐션을 도입한다면 타겟언어를 디코딩할 때 소스언어의 단어 시퀀스 가운데 디

코딩에 도움이 되는 단어 위주로 취사선택해서 번역품질줄 끌어올림. 디코딩할 때 소스시퀀스 가운데 중요한 요소만 추림.

-> rnn -> 시간이 갈수록 정보압축문제가 발생함. 오래전 입력한 단어를 잊는다거나, 특정 단어 정보를 과도하게 반영해 전체 정보를 왜곡하는 경우가 자주 생김.

->어텐션 -> cafe라는 단어를 디코딩해야할 때 카페를 반드시 참조해야함. 어텐션이 없는 단순 rnn을 사용하면 워낙 초반에 입력된 단어라 모델이 잊었을가능성이 크고, 번역품질이 낮아짐.

디코더쪽 rnn에 어텐션을 추가하는 방식. 어텐션은 디코더가 타깃 시퀀스를 생성할 때 소스 시퀀스 전체에서 어떤 요소에 주목해야 할지를 알려주므로, 카페가 소스 시퀀스 초반에 등장하거나 소스 시퀀스의 길이가 길어지더라도 번역 품질이 떨어지는 것을 막을 수 있음.

거기 -> 카페 갔었어. 더 많이 반영.

카페라는 ㄴ 단어를 인코딩할 때, 거기, 갔었어. 의미가 더 많이 반영.

즉, 어텐션은 소스 시퀀스 전체 단어(어제, 카페...많더라) 들과 타깃 시퀀스 단어 하나하나 (cafe)사이를 연결하는데 사용.

어텐션은 rnn, 셀프 어텐션은 rnn없이 사용.

타깃언어의 단어를 1개 생성할 때 어텐션은 1회 수행하지만 셀프 어텐션은 인코더, 디코더 블록의 개수만큼 반복 수행.

11:35

9. 계산 예시. p81

셀프 어텐션은. 쿼리, 키, 벨류 3가지 요소가 서로 영향을 주고 받음. 트랜스포머 블록에는 문장 내 각 단어가 벡터 형태로 입력됨. 벡터란 숫자의 나열 정도.

블록 내에서 어떤 과정을 거쳐서 쿼리, 키, 벨류 3가지로 변환됨.

만일 입력되는 문장이 6단어였다면.

-> 쿼리 벡터 6개, 키벡터 6개, 벨류 벡터 6개 등 모두 18개가 됨.

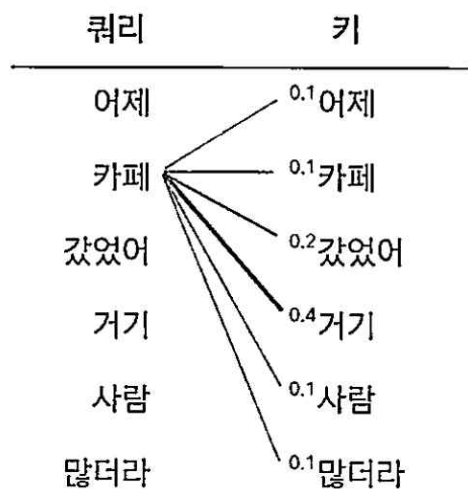


그림 3-22 셀프 어텐션 계산 예시

쿼리 단어 각각을 대상으로 모든 키 단어와 얼마나 유의적인 관계를 맺는지 그 합이 1인 확률 값으로 나타냄.

셀프 어텐션 모듈은 여기에 벨류 벡터들을 가중합(weight sum)하는 방식으로 계산을 마무리함. 카페벡터는 문장에 속한 각 단어와 카페 사이의 관계가 녹아 있음.

$$\mathbf{Z}_{\text{카페}} = 0.1 \times \mathbf{V}_{\text{어제}} + 0.1 \times \mathbf{V}_{\text{카페}} + 0.2 \times \mathbf{V}_{\text{갔었어}} + 0.4 \times \mathbf{V}_{\text{거기}} + 0.1 \times \mathbf{V}_{\text{사람}} + 0.1 \times \mathbf{V}_{\text{많더라}}$$

수식 3-6 셀프 어텐션 계산 예시

가중합: 복수의 수를 단순히 합하는 것이 아니라 각각에 어떤 가중치를 곱한 후, 이 곱셈 결과들을 합한 것을 가리킴.

셀프 어텐션을 블록 수 만큼 반복함.

10. 모든 입력과 출력

인코더 입력은 **소스 시퀀스의 입력 임베딩**에 위치정보를 더해서 만든다.
이때 인코더 입력은 소스 언어 문장의 토큰 인덱스 시퀀스가 됨.

인코더 입력층에서 만들어진 벡터 시퀀스가
최초 인코더 블록의 입력이 되며,
그 출력 벡터 시퀀스가 두 번째 인코더 블록의 입력이 됨.

출력층
디코더 출력-선형변화-소프트맥스-다음토큰확률

다음 타깃 언어의 어휘가 총 3만개라고 가정하면 이 벡터의 차원수는 3만이 되며,
3만개 요솟값을 모두 더하면 그 합은 1이 됨.

11. 셀프 어텐션 내부 동작

3-25를 보면 인코더에서 수행되는 셀프 어텐션의 입력은 이전 인코더 블록의 출력 벡터 시퀀스.

3-24의 단어 임베딩 차원수(d)가 4이고 인코더에 입력된 단어 개수가 3일 경우,
셀프 어텐션 입력은 다음 수식의 X와 같은 형태가 됨.

a a a a
b b b b
c c c c

a.b.c3개의 단어- 한단어는 임베딩 차원수가 4임.

-> 여기에 입력벡터 시퀀스(x)에 쿼리, 키, 벨류를 만들어주는 행렬(W)를 곱함.
3개의 벡터 시퀀스이니, 쿼리, 키, 벨류 각각 3개씩 총 9개의 벡터가 나옵니다.

$$\begin{bmatrix} a & a & a & a \\ b & b & b & b \\ c & c & c & c \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

$Q = X$ 곱하기 W_q

$K = X$ 곱하기 W_k

$V = X$ 곱하기 W_v

쿼리, 키, 벨류 만들기. W_q , W_k , W_v 는 기계번역을 가장 잘 수행하는 방향으로 학습 과정에서 업데이트 됨.

(2) 첫 번째 쿼리의 셀프 어텐션 출력값 계산하기.

단어 임베딩 차원수(d)가 4이고 인코더에 입력된 단어 개수가 3일 경우, 셀프 어텐션 입력은 다음 수식의 X와 같은 형태가 됨.	
a a a a b b b b c c c c	
<p>쿼리와 키를 곱한 후, 해당 행렬의 모든 요솟값을 키 차원수의 제곱근으로 나누고,</p> <p>이 행렬을 행 단위로 소프트맥스를 취해 스코어 행렬로 만들어 줌.</p> <p>이 스코어 행렬에 벨류를 행렬곱하면 셀프 어텐션 계산을 마칩.</p> <p>x 멀티헤드수.</p> <p>입력 단어수 2(a,b)개, 벨류 차원은 3, 헤드는 8개,</p> <p>2x 24 행렬</p>	
X	WO 행렬곱 (이것의 크기는 셀프어텐션 수행 결과 행렬의 수 x 목표 차원수) (24 x 목표 차원수)

->

첫 번째 쿼리를 벡터 x 모든 키 벡터에 전치를 취한 행렬(Ktranspose)를 곱함.
전치 (원래의 행렬의 행과 열을 교환해주는 것 뜻함)

첫 번째 쿼리 벡터와 첫 번째 키 벡터 사이의 문맥적 관계성이 녹아듬.

$$\begin{bmatrix} 1 & 0 & 2 \end{bmatrix} \times \begin{bmatrix} 0 & 4 & 2 \\ 1 & 4 & 3 \\ 1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 4 \end{bmatrix}$$

수식 3-17 첫 번째 쿼리 벡터에 관한 셀프 어텐션 계산 (1)

첫 번째 쿼리 x 키 벡터의 전치.

$$\text{softmax}\left(\begin{bmatrix} \frac{2}{\sqrt{3}} & \frac{4}{\sqrt{3}} & \frac{4}{\sqrt{3}} \end{bmatrix}\right) = \begin{bmatrix} 0.13613 & 0.43194 & 0.43194 \end{bmatrix}$$

수식 3-18 첫 번째 쿼리 벡터에 관한 셀프 어텐션 계산 (2)

다음 수식은 앞선 결과에 키 벡터의 차원수(dk=3)의 제곱근으로 나눈 후 소프트 맥스를 취하여 만든 벡터.

$$\begin{bmatrix} 0.13613 & 0.43194 & 0.43194 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 3 \\ 2 & 8 & 0 \\ 2 & 6 & 3 \end{bmatrix} = \begin{bmatrix} 1.8639 & 6.3194 & 1.7042 \end{bmatrix}$$

수식 3-19 첫 번째 쿼리 벡터에 관한 셀프 어텐션 계산 (3)

소프트맥스 벡터 x 벨류 벡터들을 계산.

12. 멀티헤드 어텐션

멀티 헤드 어텐션은 셀프 어텐션을 동시에 여러번 수행하는 걸 가리킴.

입력 단어수 2(a,b)개, 벨류 차원은 3, 헤드는 8개,

head-1			head-2			head-3			head-4			head-5			head-6			head-7			head-8		
a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a	a
b	b	b	b	b	b	b	b	b	b	b	b	b	b	b	b	b	b	b	b	b	b	b	b

2x24 행렬 (셀프어텐션 수행결과)

위에거 곱하기 x W0 행렬곱

(이것의 크기는 셀프어텐션 수행 결과 행렬의 수 x 목표 차원수)

(24 x 목표 차원수)

13. 인코더에서 수행하는 셀프 어텐션

트랜스포머의 학습과제가 한국어에서 영어라면, 인코더의 쿼리, 키, 벨류는 모두 한국어가 됨.

잘 수행된 셀프어텐션 결과.

어제 - 갔었어. 0.3

카페- 거기 0.4

14. 디코더에서 수행하는 셀프 어텐션

다른 것은 비슷.

멀티 헤드 어텐션이 다름.

쿼리	키
<s>	어제
I	카페
went	갔었어
to	거기
the	사람
cafe	많더라
:	

그림 3-34 학습 시 디코더에서 수행되는 셀프 어텐션(1)

트랜스포머 최종 출력은 카짓 시퀀시 각각에 대한 확률 분포, 모델이 한국어를 영어로 번역하는 태스크를 수행하고 있다면, 영어 문장의 다음 단어로 어떤 것이 적절한 것인지에 대한 확률.

input -> ouput
어제 -> I를 맞추고
카페 -> went
갔었어 ->

그래서 정답을 알려줄 수 있으므로, 미래정보를 셀프어텐션 계산에서 제외합니다.
멀티헤드 어텐션 계산시 제외 대상 단어들의 소프트맥스 확률이 0이 되도록 하여, 해당 단어 정보들이 무시되게 하는 방식으로 수행.
-> p97쪽 특이.

15. 잔차연결! (residual connection)

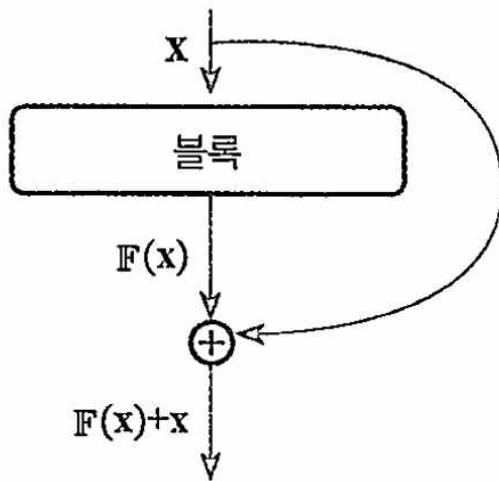


그림 3-49 잔차 연결 (1)

블록 계산이 계속될 때, 잔차 연결을 두는 것은 제법 효과가 있음.

블록 계산이 계속될 때 잔차 연결을 두는 것은 제법 큰 효과가 있습니다. 다음 그림의 왼쪽처럼 블록 연산을 3번 수행하고 블록과 블록 사이에 잔차 연결을 모두 적용했다고 가정해 봅시다. 그렇다면 모델은 사실상 그림의 오른쪽처럼 계산하는 형태가 됩니다.

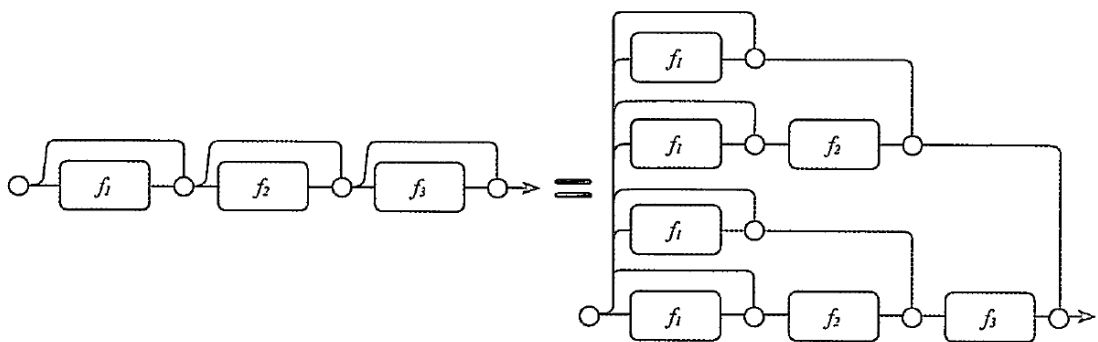


그림 3-50 잔차 연결 (2)

왼쪽이 잔차 연결이 없을 때는 차례대로 하는 한가지 방법만 존재했으나, 오른쪽처럼 잔차연결을 블록마다 설정해둌으로써 **모두 8가지의 새로운 경로가 생김.** 다시 말해서 다양한 관점에서 블록 계산을 수행하게 된다는 이야기임.

딥러닝 모델은 레이어가 많아지면 학습이 어려워지는 경향이 있음. 모델을 업데이트할 때, 그 레디언트가 전달되는 경로 같아지기 때문.

잔차연결은 모델 중간에 블록을 건너 뛰는 경로를 설정함으로써 학습을 쉽게 하는 효과가 있음.

16. 레이어 정규화. p107

layer normalization

미니배치의 인스턴스별로 평균을 빼주고 표준편차로 나눠 정규화를 수행하는 기법.

분모가 0이되는걸 방지하려고 더해주는 고정값이 있음 (보통 $1e-5$)로 설정

파이토치적 layerNorm객체는 이 두 값을 각각 1과 0으로 초기화합니다.

다시 말해서 학습 초기 레이어 정규화 수행은 배치 인스턴스의 평균을 빼고 표준편차로 나눈 결과에 1을 곱한 후 0을 더한다는 이야기. 이후 학습과정에서 테스트를 가장 잘 수행하는 방향으로 이 값들을 업데이트함.

• 코드 3-8 레이어 정규화 예시

```
import torch
input = torch.tensor([[1.0, 2.0, 3.0], [1.0, 1.0, 1.0]])
m = torch.nn.LayerNorm(input.shape[-1])
output = m(input)
```

배치의 첫 번째 데이터를 수식 3-27에 따라 정규화하면 그 결과는 $[-1.2247 \ 0.0 \ 1.2247]$ 이 됩니다. 같은 방식으로 배치의 두 번째 데이터를 정규화하면 $[0.0 \ 0.0 \ 0.0]$ 이 됩니다. 다음은 앞 코드의 output을 파이썬 콘솔에서 확인한 결과입니다.

17. Dropout.

각 요소값에 $1/(1-p)$ 를 곱하는 역할도 수행함.

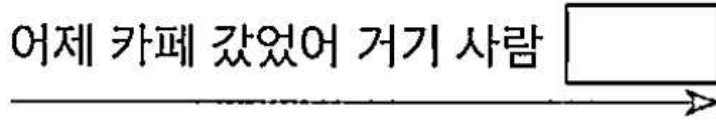
p가 0.2면, 살아남은 요소값에 각각 1.25를 곱하는셈. (특이사항)

1. 입력 임베딩과 최초 블록 사이,
2. 블록과 블록 사이.
3. 블록과 출력층 사이에 적용함. 보통 10%를 설정하는 것이 일반적

18. bert와 GPT 113쪽.

GPT

어제 카페 갔었어 거기 사람



BERT

어제 카페 갔었어 사람 많더라

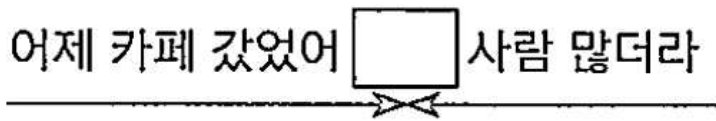


그림 3-54 GPT와 BERT의 프리트레인 방법

Gpt는 언어모델- 이전 단어들이 주어졌을 때, 다음 단어가 무엇인지 맞추는 과정에서 프리트레인. 일방향이다.

- 문장생성.
- 인코더만 사용.
- '풀문장' 어제 카페 갔었어. 거기 사람 많더라.

모든 과정을 거쳐서- 선형변환-소프트맥스까지 적용해 요숫값 각각이 확률이고, 대상 언어의 어휘 수 만큼 차원수를 가진 벡터가 되도록함.

'어제'라는 단어를 보고, '카페'를 맞춰야함. 정답 단어 이후의 단어들은 볼 수 없도록 0이 벨류값이 0이 되도록함.

'카페'의 요숫값 확률을 높이고, 나머지 단어의 벡터들의 확률은 낮춤.

ex

쿼리 - 어제, 카페, 갔었어 (어휘3번째까지 왔음)

키 - 어제, 카페, 갔었어 (앞에 나온 단어들 확률이 영향을 줌)

전체 벡터값 -다음 단어, '거기'에 대한 확률을 높이고 나머지는 낮춤.

Bert는

마스크 언어 모델. 문장 중간에 빈칸을 만들고 해당 빈칸에 어떤 단어가 적절한지 맞추는 과정에서 프리트레인 한다. 빈칸 안뒤 모두 문맥을 모두 살필 수 있다는 점에서 양방향이다.

- 문장의 의미를 추출하는데 강점.
- 디코더만 사용.

그림 3-63은 BERT의 셀프 어텐션을 나타낸 것입니다. 입력 단어 시퀀스가 어제 카페 갔었어 [MASK] 사람 많더라라고 가정해 보겠습니다. 그림에서 확인할 수 있듯이 BERT는 마스크 토큰 앞뒤 문맥을 모두 참고할 수 있습니다. 앞뒤 정보를 준다고 해서 정답을 미리 알려주는 것이 아니기 때문입니다.



그림 3-63 BERT의 셀프 어텐션

[MASK]라는 단어에 대응하는 BERT 마지막 레이어의 출력 결과에 선형 변환과 소프트맥스를 적용해 요솟값 각각이 확률이고 학습 대상 언어의 어휘 수만큼 차원 수를 가진 벡터가 되도록 합니다. 빈칸의 정답인 거기에 해당하는 확률은 높이고 나머지 단어의 확률은 낮아지도록 모델 전체를 업데이트합니다.



더 알아볼 것.

디스틸레이션(distillation), 쿼타이제이션(quantization), 프루닝(pruning, 파라미터 공유(weight sharing) 등이 그것임.

19. 121쪽부터