Attention이란?–원리부터 masking까지

RNN 기반의 시퀀스-투-시퀀스(Sequence-to-Sequence, seq2seq)(Sequence-to-Sequence, seq2seq)에는 다음과 같은 문제가 있다.

- hidden state에 정보를 저장하는데, capacity가 유한하여 전체 timestep의 정보를 담기 어렵다

즉, 문장의 길이가 길어진다면, 전체 문장의 정보를 담기에는 어려움이 있는 것이다.

이러한 문제를 해결할 수 있는것이 바로 attention이다.

Attention

Attention은 미분 가능한 key-value function이다.(Differentiable Key-Value Function)

다만, python의 dictionary처럼 Query가 Key와 완벽히 일치해야 Value를 반환하는 것이 아니라, Query와 Key의 유사도에 따라 Value를 반환한다.

즉, Key와 Query간의 유사도를 구해서, 유사도에 따라 Value를 averaging(weighted sum)하고, 이렇게 averaging된 value를 반환하는 것이다.

이러한 과정은 RNN계열 아키텍쳐의 hidden state의 한계로 인해 부족한 정보를 직접 encoder에 조회하여 예측에 필요한 정보를 얻어오는 과정이다.

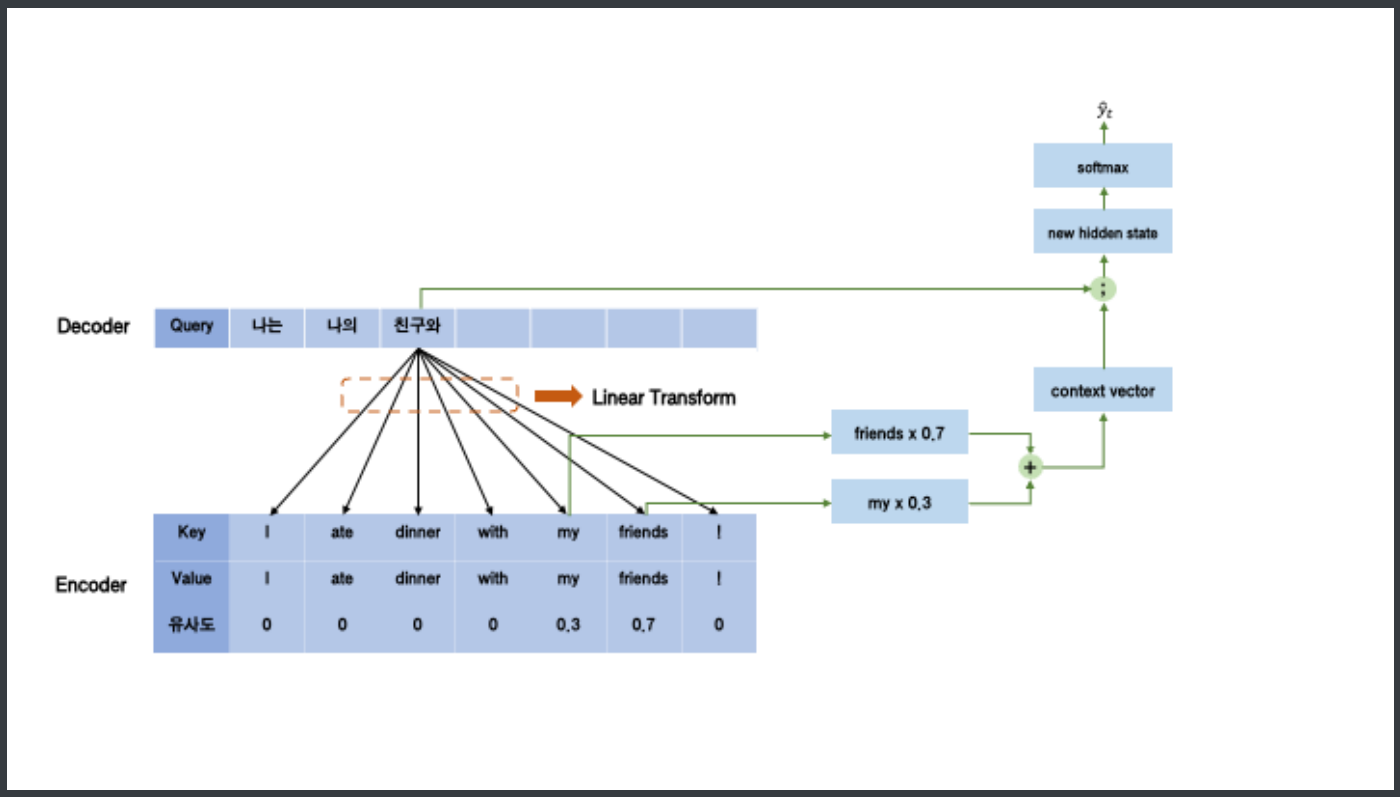
따라서, 예측에 필요한 정보를 잘 얻어오기 위해서는 Query를 잘 만들어내야 하는데, attention은 학습하면서 이 Query를 잘 만들어내는 과정을 학습한다.

시퀀스-투-시퀀스(Sequence-to-Sequence, seq2seq)에서의 Attention의 각 요소들은 다음과 같다.

- Query : 현재 timestep의 decoder output
- Keys : 각 timestep별 encoder output
- Values : 각 timestep별 encoder output

예시 그림과 함께 자세히 살펴보자

다음 예시와 함께 attention의 작동 순서를 하나씩 따라가보겠다.



1. decoder로부터 "친구와" 라는 output이 나옴
2. 해당 decoder의 output을 linear transform 한 뒤 encoder의 모든 timestep에 Query로 넣음 -> 이 때, linear transform의 학습이 잘 되어야 함
3. 각 timestep별 Key와 Query(여기서는 decoder의 output "친구와")의 유사도를 구함
4. 계산된 유사도를 각 timestep의 value에 곱해서 더함 => context vector
5. decoder의 output(Query)와 4번에서 만든 context vector를 concat

2번 과정에서, linear transform이라는 것이 눈에 띈다. linear transform이라는 것은 과연 무엇일까?

linear transform이란, decoder의 현재 hidden state를 보고, 필요한 정보를 보다 잘 얻어주는 Query로 변환해주는 역할을 한다.

예를 들어, 흔히들 많이 하는 구글링을 할 때, 누구는 구글링을 잘 하고 반면 잘 하지 못하는 사람도 있다. 구글은 그대로인데, 이 두 사람의 차이는 무엇일까?

바로, 구글 검색창에 검색어를 어떻게 입력하느냐의 차이이다. 같은 정보를 원해도 검색창에 검색어를 입력하는 방식이 다르면 얻을 수 있는 정보도 달라지는 것이다.

이처럼 linear transform은 encoder라는 구글에 현재 timestep의 decoder output을 검색하고자 할 때, 보다 정보를 잘 가져올 수 있도록 구글링을 잘 하게 만들어주는 역할인 셈이다.

이러한 attention을 수식과 함께 살펴보도록 하자

$$
\begin{gathered}
w = \text{softmax}(h_t^{dec} \cdot W_a \cdot h_{1:m}^{enc\,T}) \\
c = w \cdot h_{1:m}^{enc}, \\
\text{where } c \in \mathbb{R}^{\text{batch\_size x 1 x hidden\_size}} \text{ is a context vector, and} \\
W_a \in \mathbb{R}^{\text{hidden\_size x hidden\_size}}
\end{gathered}
\tag{3}
$$

$$
\begin{gathered}
w = \text{softmax}(h_t^{dec} \cdot W_a \cdot h_{1:m}^{enc\,T}) \\
c = w \cdot h_{1:m}^{enc}, \\
\text{where } c \in \mathbb{R}^{\text{batch\_size x 1 x hidden\_size}} \text{ is a context vector, and} \\
W_a \in \mathbb{R}^{\text{hidden\_size x hidden\_size}}
\end{gathered}
$$

W_a는 위에서 언급한 attention을 위한 linear transformation이다. 이 W_a의 shape은 (hidden size, hidden size)이다.

timestep t에서 decoder의 hidden state인 h_t^{dec}의 shape은 (batch size, 1, hidden size)이다.

$h_{1:m}^{enc\,T}$는 encoder의 hidden state이고, shape의 경우 (batch size, m , hidden size)를 transpose 했기 때문에 (batch size, hidden size, m)이 된다.

따라서, softmax 안의 연산을 수행하면 다음과 같은 과정을 거친다

(batch size, 1, hidden size) x (hidden size, hidden size) x (batch size, hidden size, m)

= {(batch size, hidden size) x (hidden size, hidden size)} x (batch size, hidden size, m)

= (batch size, hidden size) x (batch size, hidden size, m)

= (batch size, 1, hidden size) x (batch size, hidden size, m)

= (batch size, 1, m)

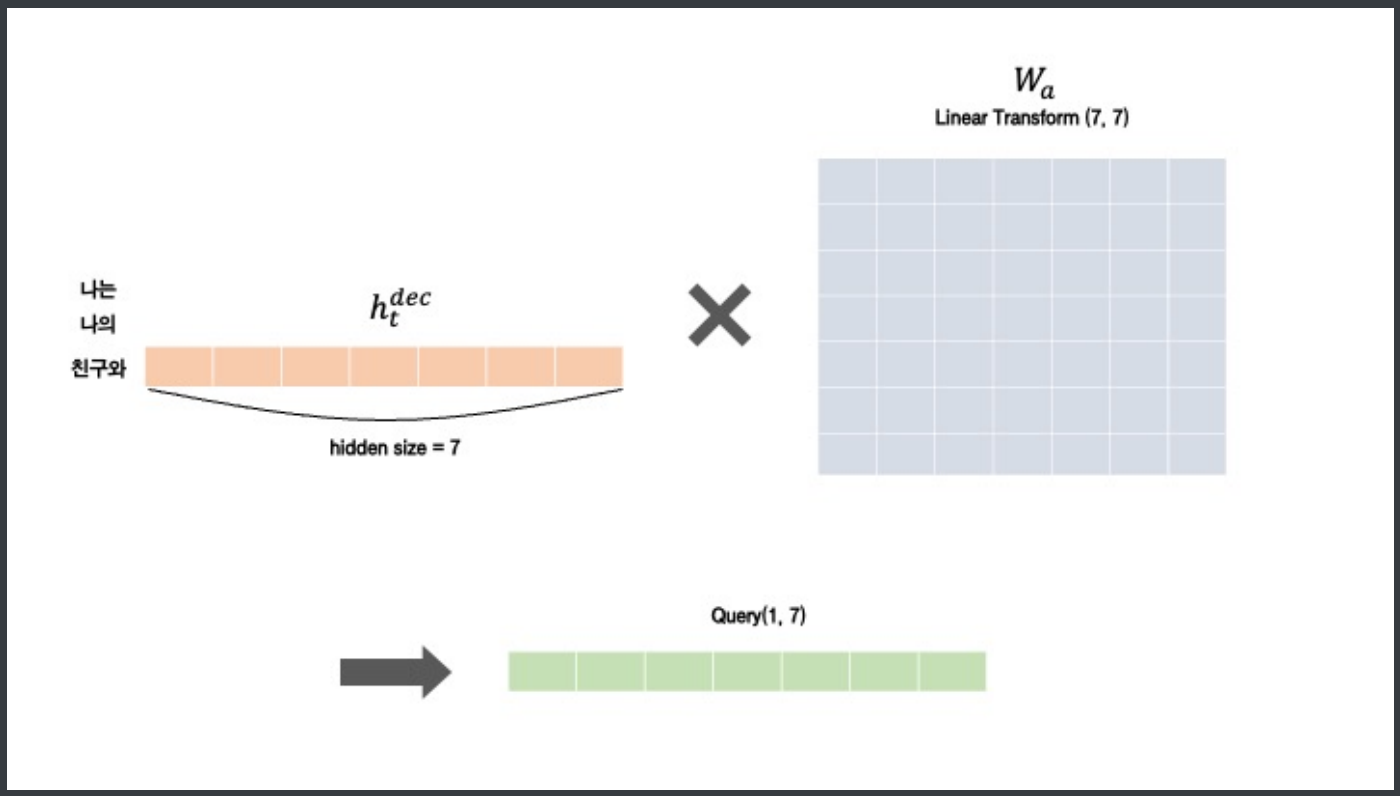
이 과정은, batch 내의 문장 별 해당 timestep(단어) t의 hidden state -> (h_t^{dec})가

linear transform을 거쳐 (W_a)

batch 내부의 encoder의 전체 timestep의 hidden state($h_{1:m}^{enc\,T}$) 와 곱해지고, softmax를 적용하는 것인데,

이는 batch 내부의 문장 별 해당 timestep의 encoder의 각각의 timestep에 대한 가중치인 것이다.

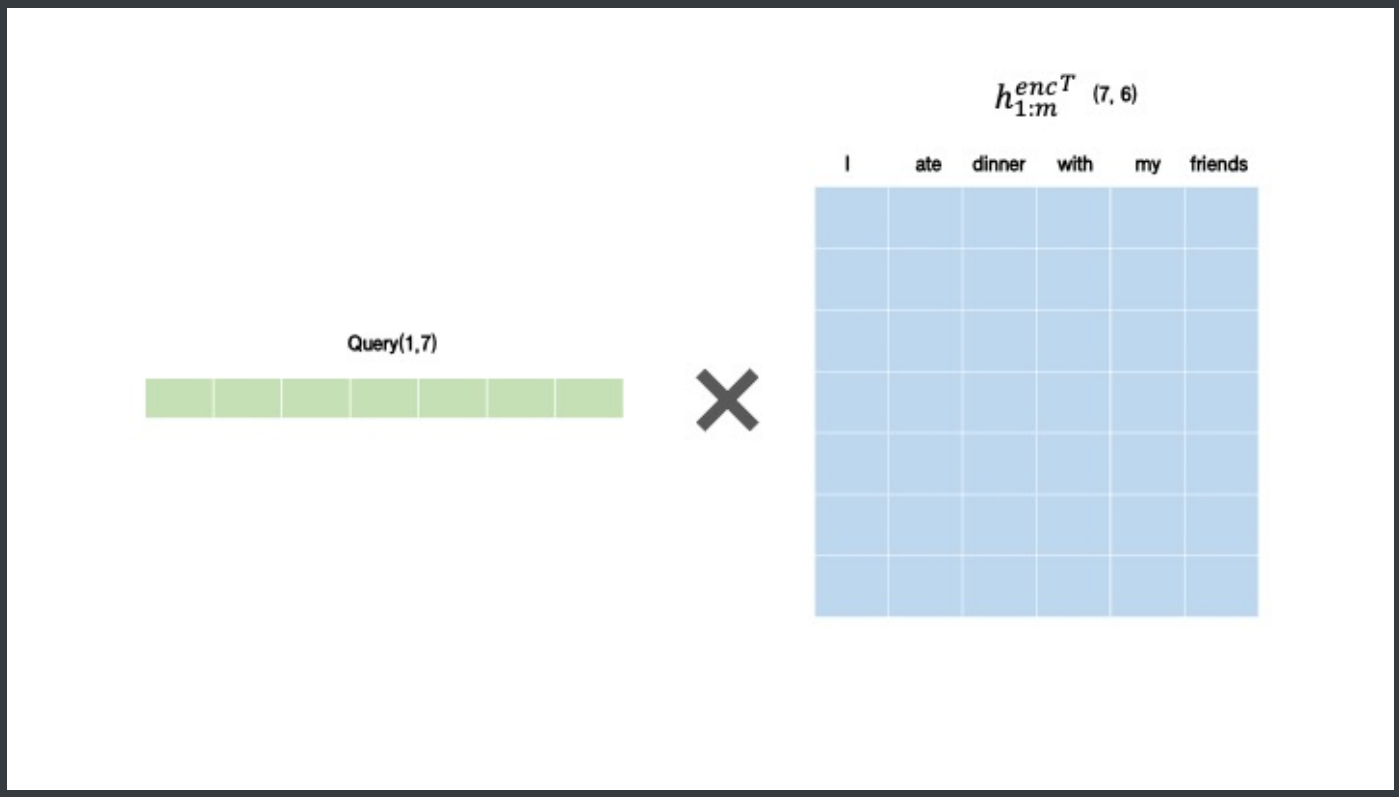
그림과 함께 알아보자 (그림에서는 batch size는 생략하였습니다)



우선 hidden size가 7이라고 가정해보자. decoder에서, 특정 timestep t에서의 hidden state h_t^{dec}이 그림에서 보일 것이다.

(옆에 있는 나는, 나의, 친구와는 이해를 돕기 위해 적어놓았다

이를 W_a에 곱해주면서 linear transform을 실행한다. 결과적으로 shape (1, 7)의 Query가 생성되었다. **(위에서의 과정 2)**



이렇게 생성된 Query를 인코더의 배치 안에서의 문장에 대한 hidden state의 전치 행렬, $h_{1:m}^{enc^T}$에 곱해 준다.

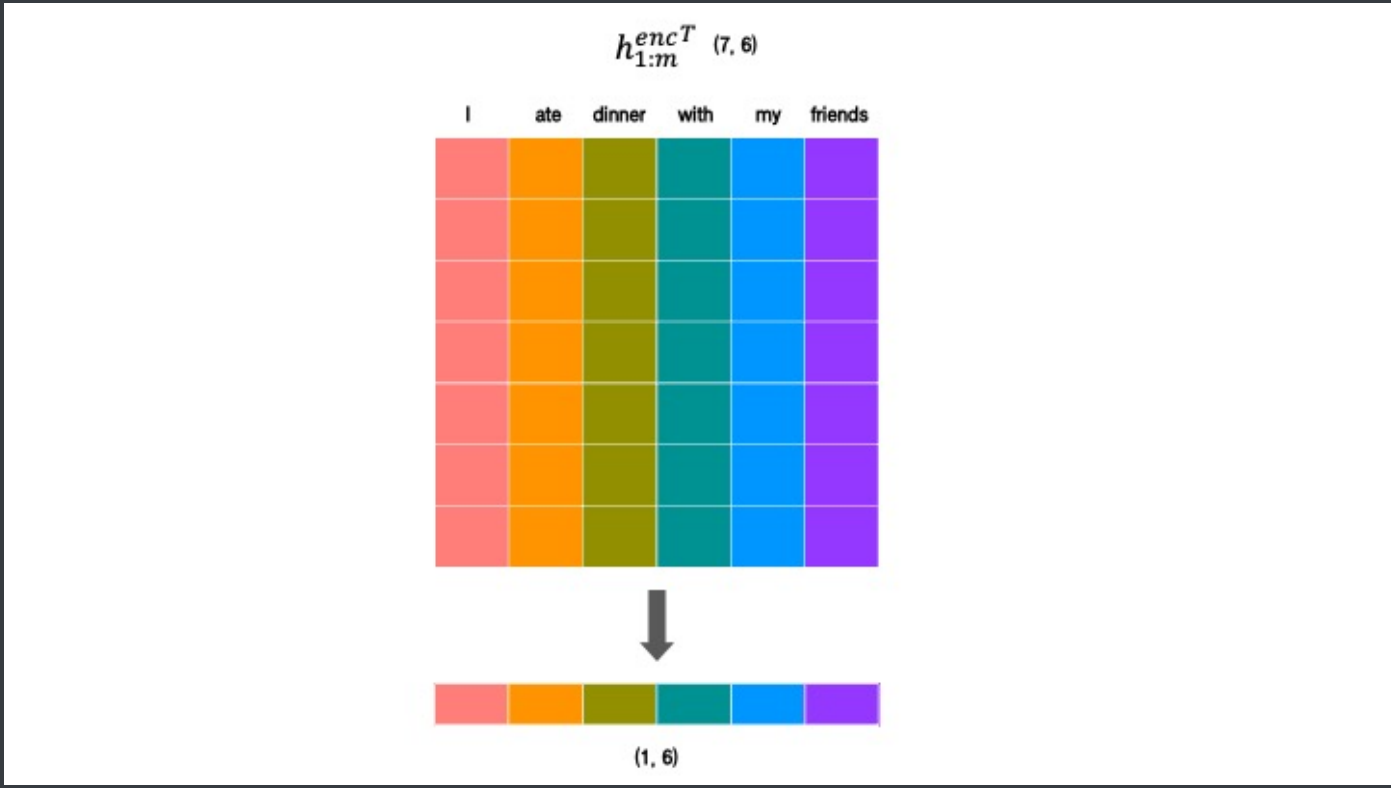
여기서의 m은 6으로, 배치 안의 문장의 단어 수이다.

(아까와 마찬가지로 각 단어들은 이해를 돕기 위해 적어놓았다.)

이렇게 되면, "친구와" 부분의 hidden state가 linear transform되어 Query가 되었고, 해당 Query는 $h_{1:m}^{enc^T}$과 행렬곱이 진행되면서 encoder의 각 timestep별 vector들과 내적 연산(dot product)이 실행된다.

내적 연산을 통해 두 벡터의 유사도를 구할 수 있는데, 이 말인즉슨, 문장 안에 있는 단어(timestep)와 Query의 단어가 얼마나 비슷한지를 구할 수 있는 것이다. (위에서의 과정 3)

그러면 다음과 같은 결과가 나온다



이렇게 각 단어(timestep) 별 유사도가 담긴 vector에 softmax를 씌워줌으로써, 우리는 드디어 batch 내부의 문장 별 해당 timestep의 encoder의 각각의 timestep에 대한 가중치 벡터w를 알 수 있게 된다

이후, 이렇게 계산된 가중치 w를 각 timestep의 value에 곱해서 더하여(weighted sum) context vector인 c를 만들어준다.

(위에서의 과정 4)

계산된 context vector c의 shape은 (batch size, 1, m) x (batch size, m, hidden size) = (batch size, 1, hidden size)가 된다

$$\tilde{h}_t^{dec} = \tanh([h_t^{dec}; c] \cdot W_{\text{concat}})$$
$$\hat{y}_t = \text{softmax}(\tilde{h}_t^{dec} \cdot W_{\text{gen}}) \quad (4)$$
$$\text{where } W_{\text{concat}} \in \mathbb{R}^{(2 \text{ x hidden\_size}) \text{ x hidden\_size}} \text{ and } W_{\text{gen}} \in \mathbb{R}^{\text{hidden\_size x } |\text{V}|}$$

context vector c는 decoder의 hidden state와 concat하게 된다 ($[h_t^{dec}; c]$)

이때의 shape은 concat을 하였으니 (batch size, 1, hidden sizex2)가 된다

이후, 원래의 hidden size로 차원 축소를 해주기 위해 W_{concat}을 곱해주고, 여기에 탄젠트-하이퍼볼릭 함수를 적용시키면서 $\tilde{h}_t^{dec}$를 얻게 된다 **(위에서의 과정 5)**

이후 seq2seq의 generator에 해당 $\tilde{h}_t^{dec}$를 넣어주고, softmax를 취해주면서 길었던 과정이 끝나게 된다.

Masking

이렇게 attention을 사용하면 기존보다 좋은 성능을 기대할 수 있다.

그러나, 바로 attention을 적용하기에는 아직 문제가 남아있다. 하나의 batch가 있을 때, 안에 들어있는 문장들의 길이는 천차만별이다.

문장의 길이가 다르기 때문에 짧은 문장의 경우, <PAD>로 채워지는데, 이로 인해 문제가 발생하게 된다.

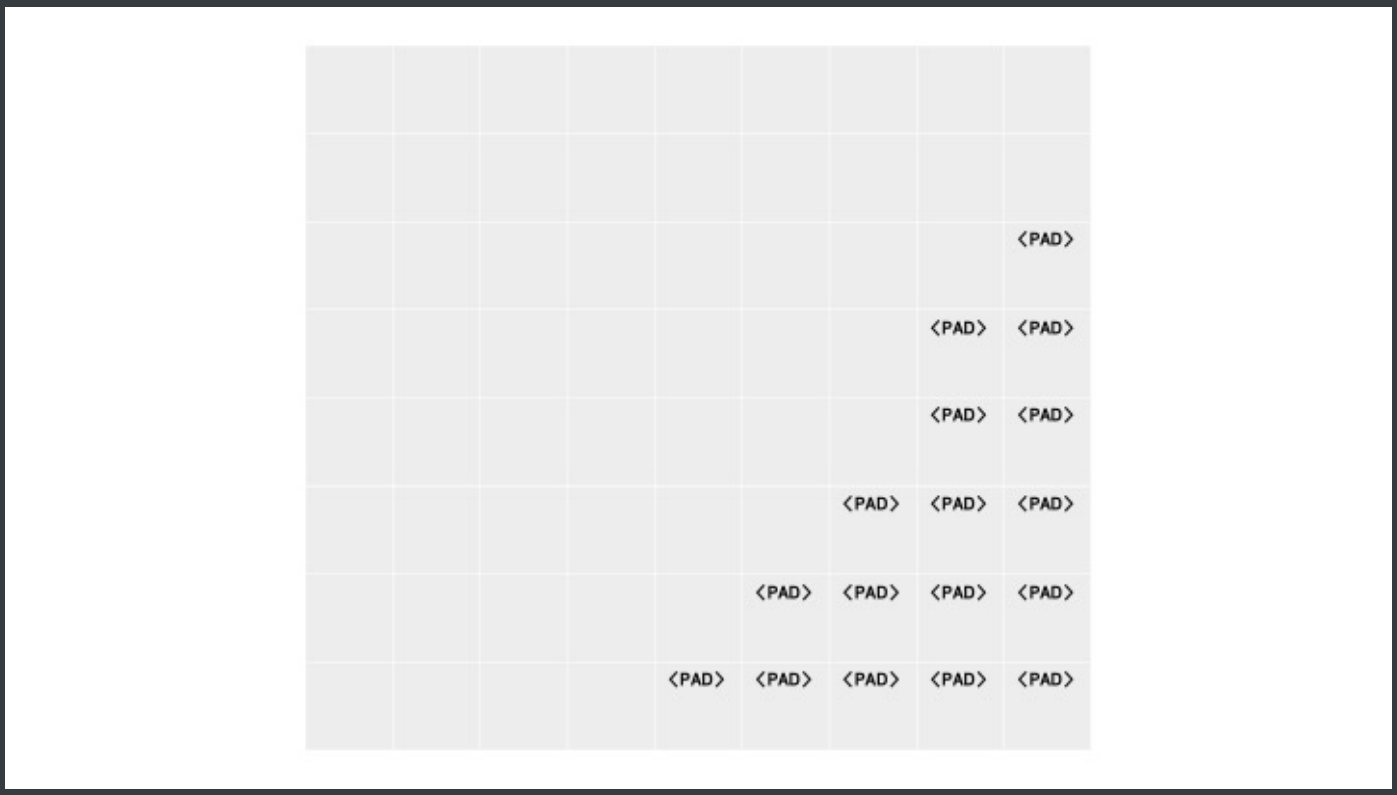
저는	지금	학교에	와	있습니다	.	〈PAD〉	〈PAD〉	〈PAD〉	〈PAD〉
배가	몹시	고프군요	혹시	먹을	것이	남아	있을지	궁금합니다	.
저는	XXX	입니다	.	〈PAD〉	〈PAD〉	〈PAD〉	〈PAD〉	〈PAD〉	〈PAD〉
키보드	스위치는	체리	흑축을	가장	선호합니다	.	〈PAD〉	〈PAD〉	〈PAD〉
운영체제는	애플의	맥을	좋아합니다	.	〈PAD〉	〈PAD〉	〈PAD〉	〈PAD〉	〈PAD〉

우선 우리는 softmax의 성질에 대해 알아보고 넘어가야한다. 유사도를 구할 때, 결국 마지막에 softmax 함수를 적용하게 되는데, **softmax 함수의 특성 상, input으로 음의 무한대가 들어가지 않으면 출력값으로 0이 나오기 힘들다.**

이러한 특성 때문에, <PAD>가 들어간 부분들에 대해 유사도가 0.01, 0.001과 같은 매우 작은 값들로 계산된다면, context vector를 구하기 위해 weighted sum을 할 때 <PAD>에 대한 정보가 조금이지만 context vector에 들어가게 된다.

따라서, 이러한 현상을 방지하기 위해 <PAD>가 있는 위치에는 음의 무한대값을 넣어줘서, 최종적으로 softmax를 취하면 0이 나오게끔 하여 필요없는 정보가 context vector에 들어가는 것을 방지할 것이다. 이를 masking이라고 한다.

그림과 함께 살펴보자



다음과 같이 하나의 batch 안에 <PAD>가 이런 형태로 분포해있다고 가정해보자.

<PAD>의 형태를 미리 알고있는 상태로, 그 모양대로 1이 채워져 있는 mask를 생성한다.

이후, Query와 Key를 곱한 결과가 나오면 이 결과에 Mask 형태로 음의 무한대 값으로 치환해준다. 이렇게 음의 무한대 값으로 치환해주면, 결과적으로 softmax를 적용시킨 context vector에는 <PAD>가 있던 자리에 0이 들어가게 됨으로써, 필요없는 정보를 차단할 수 있다.