Transformer & Attention

Sang Yup Lee

Transformer

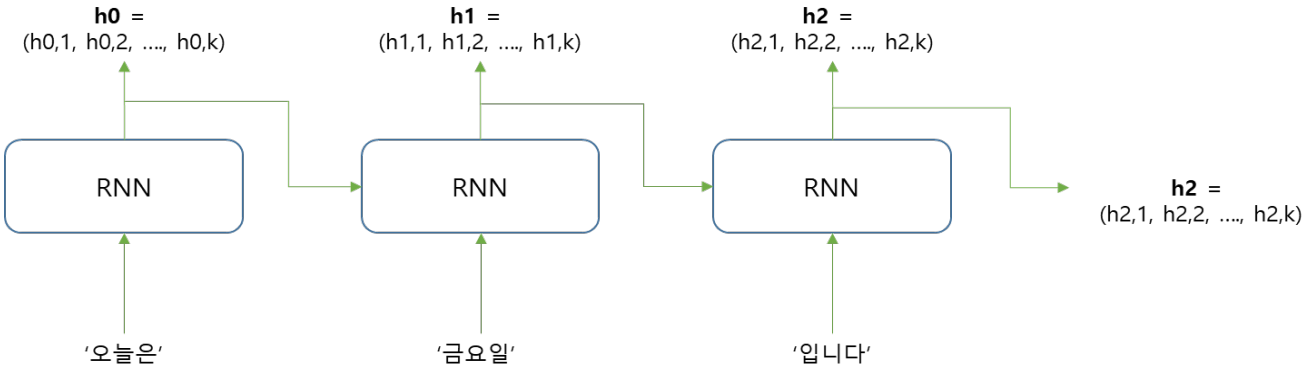
- 소개
  - 2017년에 Google에서 제안한 attention 기반의 encoder-decoder 알고리즘
    - 순환신경망 기반의 방법이 아니라 attention 사용
    - 주요 applications:
      - BERT (Bidirectional Encoder Representations from Transformers)
      - GPT (Generative Pre-trained Transformer)
      - BART (Bidirectional and Auto-Regressive Transformers)
  - Transformer를 이해하기 위해서는 attention으로 먼저 이해하는 것이 필요

Vaswani et al. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008).

Attention

Attention

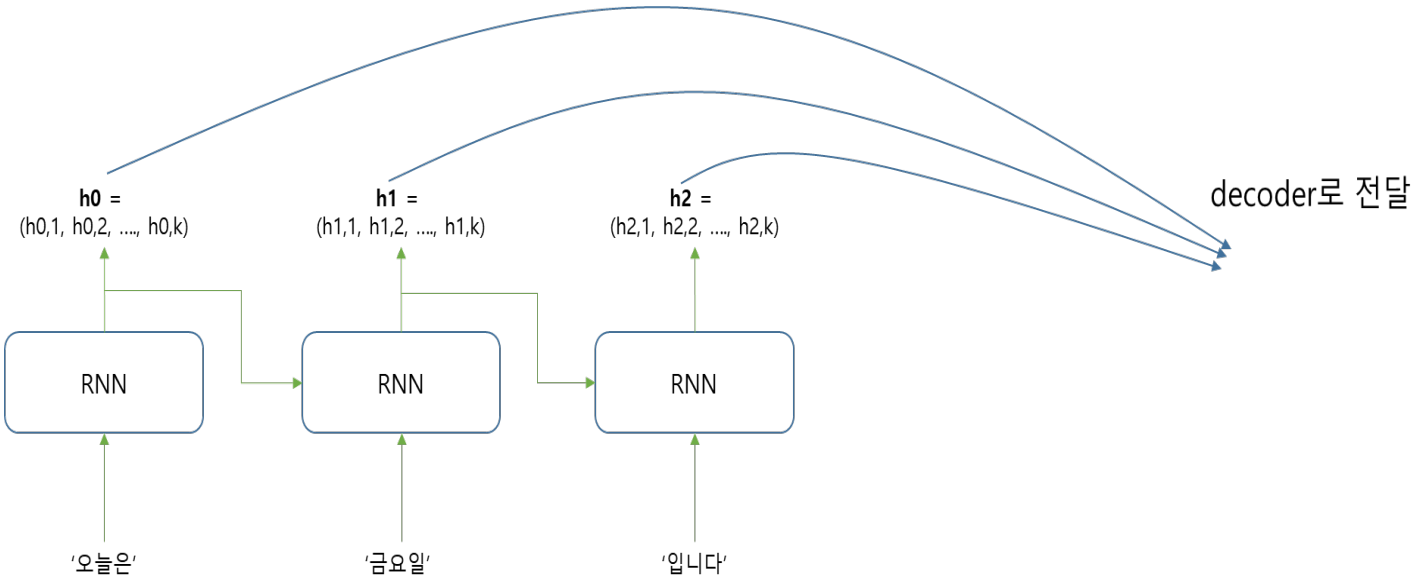
- Why was it proposed?
  - 순환신경망 기반의 seq2seq 모형이 갖는 문제점을 보완하기 위해
  - 순환신경망 기반의 seq2seq 의 주요한 문제점?
    - 입력된 sequence data에 대해서 하나의 고정된 벡터 정보(마지막 hidden state)만을 decoder로 전달한다는 것



그렇게 되면 입력된 모든 단어들의 정보가 제대로 전달되지 못한다는 문제 발생
특히, 입력된 단어가 많은 경우 앞쪽에서 입력된 단어들의 정보는 전달이 거의 안되는 문제 발생

Attention

- 그렇다면 어떻게 하면 되는가?
  - Encoder 부분에서 생성되는 각 단어에 대한 hidden state 정보를 모두 decoder로 전달
  - 예: '오늘음 금요일 입니다' ⇒ 'Today is Friday'로 번역하는 경우



Attention

- 그렇다면 encoder 부분에서 전달된 모든 단어들에 대한 hidden states를 어떻게 사용하는가?
  - Decoder 부분은 언어 모형의 역할 수행
    - 즉, 이전 단어들의 정보를 사용해서 다음 단어가 무엇인지를 예측
    - Attention 기반의 decoder는 새로운 단어를 예측할 때 encoder 부분에서 넘어온 단어들의 모든 hidden states 정보를 사용
    - 예측하고자 하는 단어와 관련이 더 많은 (encoder 부분에 입력된) 단어에 더 많은 주의(attention)을 기울이게 됨 (즉, 관련이 높은 단어의 정보를 더 많이 사용해서 단어를 예측, 이렇게 하면 더 정확하게 예측 가능)
      - 이를 attention 이라고 함
    - 신경을 쓰는 정도 (attention을 주는 정도)를 가중치로 표현

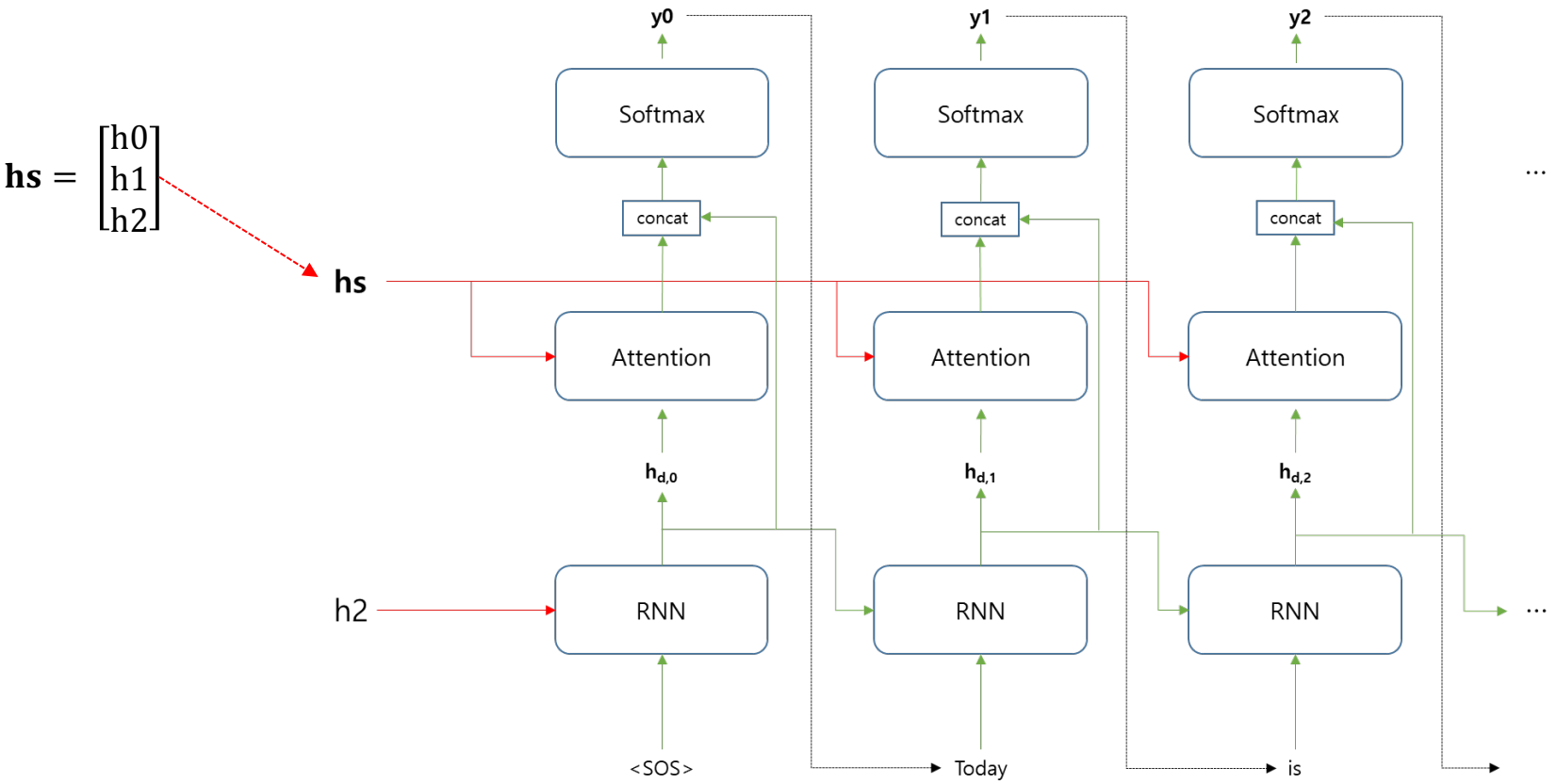
Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Attention

- 예: '오늘은 금요일입니다' ⇒ 'Today is Friday'
  - encoder에서 출력되는 값
    - 입력되는 단어들 (즉, ' 오늘은 ' , ' 금요일 ' , ' 입니다')에 대한 hidden state 정보
      - h0, h1, h2
      - $\mathrm{hs} = \begin{bmatrix} \mathrm{h0} \\ \mathrm{h1} \\ \mathrm{h2} \end{bmatrix}$
  - decoder에서 하는 일
    - 보다 정확한 번역을 위해서 encoder에서 전달된 hs의 각 hidden state에 가중치 부여
    - 즉, decoder층에서의 예측하고자 하는 단어와 더 많은 관련있는 encoder 단어에 가중치를 더 많이 부여

Attention

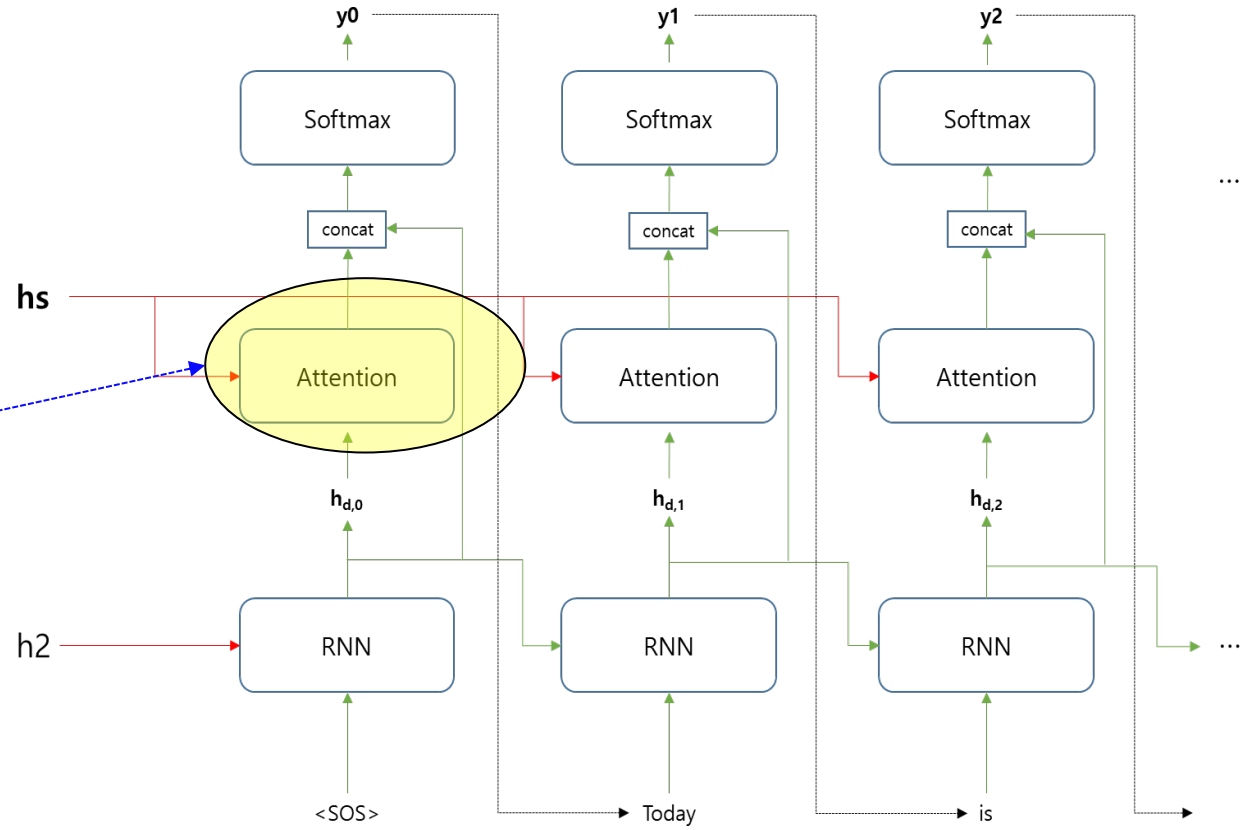
- Attention을 포함한 decoder 구조



Attention

- 'Today'를 예측하는 경우

$$\mathbf{hs} = \begin{bmatrix} h0 \\ h1 \\ h2 \end{bmatrix}$$



이 attention 층에서는
어떠한 일이 일어나는가?

예측하고자 하는 단어가
'Today' 이기 때문에
'Today'에 대응하는
입력단어인 '오늘은' 단어에
대한 hidden state에 가장
큰 가중치를 준다

Attention

- Example: 'Today'를 예측하는 경우

'오늘은'
'금요일'
'입니다'

$$\begin{bmatrix} h0 \\ h1 \\ h2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 & 2 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

예측을 하고자 하는 'today'와 관련이 제일 높은 '오늘은'에 제일 큰 가중치

- 가중치의 예
  - $\begin{bmatrix} h0 \\ h1 \\ h2 \end{bmatrix} \begin{matrix} *0.8 \\ *0.1 \\ *0.1 \end{matrix}$
  - $h0 * 0.8 = (0.8 \;\; 0 \;\; 0 \;\; 0.8 \;\; 1.6)$
  - $h1 * 0.1 = (0.1 \;\; 0 \;\; 0 \;\; 0.1 \;\; 0.1)$
  - $h2 * 0.1 = (0.1 \;\; 0.1 \;\; 0 \;\; 0 \;\; 0.1)$
- Attention 층에서 출력되는 값은? => 위의 결과 벡터들을 더한 벡터
- $(0.8 \;\; 0 \;\; 0 \;\; 0.8 \;\; 1.6) + (0.1 \;\; 0 \;\; 0 \;\; 0.1 \;\; 0.1) + (0.1 \;\; 0.1 \;\; 0 \;\; 0 \;\; 0.1) = (1.0 \;\; 0.1 \; 0 \;\; 0.9 \;\; 1.8)$

Attention

- 가중치의 계산
  - 가중치는 hs의 각 hidden state와 decoder에 예측하고자 하는 단어에 대한 hidden state와의 유사도를 이용해 계산
  - hidden state 간의 유사도를 계산 => (일반적으로) 내적 연산
  - decoder 부분의 첫번째 RNN층에서 출력되는 'Today' 단어를 예측하는데 사용되는 hidden state => $\mathbf{h_{d,0}}$
    - $\mathbf{h_{d,0}}$ = (1 0 0 0 2)
  - h0, h1, h2와 hd,0과의 내적 연산
    - $(1\ 0\ 0\ 1\ 2) \cdot (1\ 0\ 0\ 0\ 2) = 1 + 4 = 5$
    - $(1\ 0\ 0\ 1\ 1) \cdot (1\ 0\ 0\ 0\ 2) = 1 + 2 = 3$
    - $(1\ 1\ 0\ 0\ 1) \cdot (1\ 0\ 0\ 0\ 2) = 1 + 2 = 3$
    - 이 값들을 attention score라고 함
      - Attention score의 값이 클수록 관련도가 크다는 것을 의미

Attention

- Attention score functions used in literature

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

<Source> https://lilianweng.github.io/posts/2018-06-24-attention/

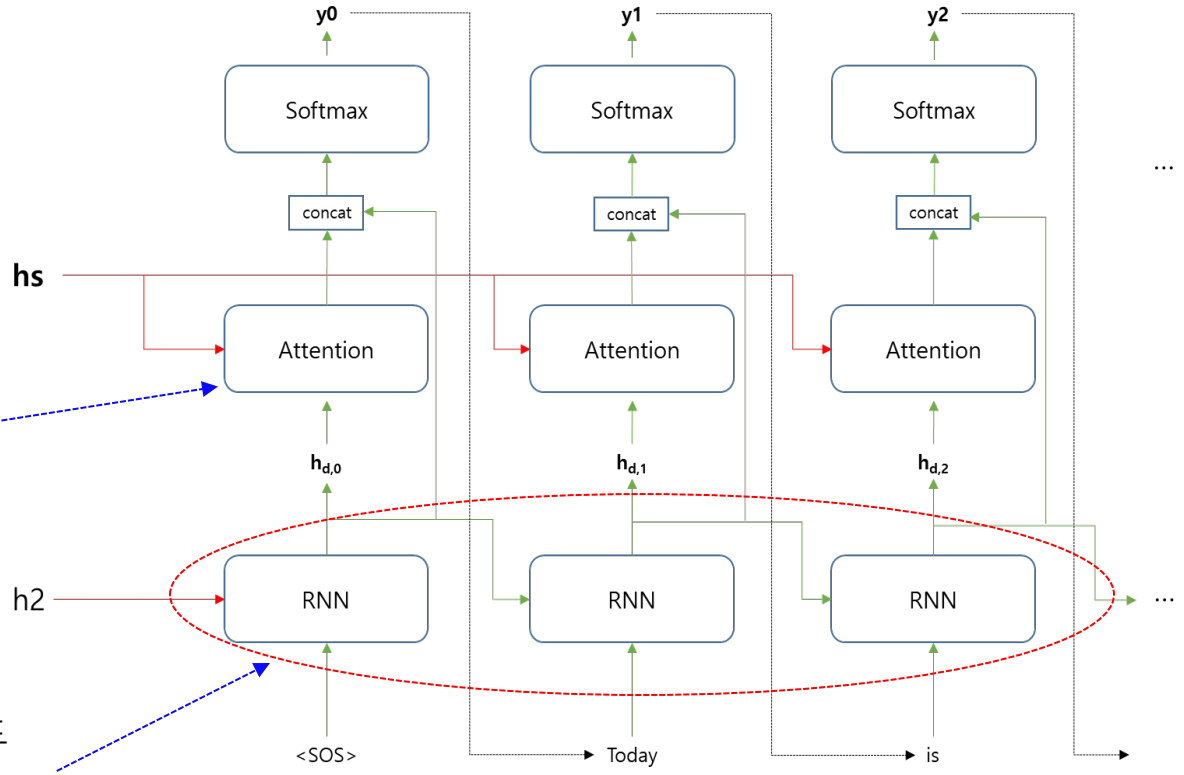
Attention

- 가중치의 계산 (cont'd)
  - Attention score를 이용해서 가중치를 계산
  - 가중치는 확률값으로 표현
  - 확률값을 계산하기 위해서 attention score에 softmax()를 적용
    - Attention score가 5인 경우에 대한 가중치는
      - $e^5 / (e^5+e^3+e^3) \approx 0.8$
    - Attention score 3에 대한 가중치는
      - $e^3 / (e^5+e^3+e^3) \approx 0.1$
- 최종적으로 출력되는 값
  - Attention에서 출력되는 값과 RNN 층에서 출력되는 값 간의 이어붙이기 (concatenation)
  - 즉, $\mathrm{Concat}\big((1.0\ \ 0.1\ 0\ \ 0.9\ \ 1.8),(1\ \ 0\ \ 0\ \ 0\ \ 2)\big)$

Attention

- Attention을 이용한 decoder 구조

$$\mathbf{hs} = \begin{bmatrix} h0 \\ h1 \\ h2 \end{bmatrix}$$



이러한 attention을 encoder-decoder attention 이라고 함

RNN층을 사용할수도 있고 사용하지 않을 수도 있음

Self-attention

Self-attention

- (Encoder-decoder) Attention과의 차이
  - Attention은 encoder-decoder 모형에서 보통 decoder 에서 encoder에서 넘어오는 정보에 가중치를 주는 식으로 작동
  - Self-attention은 **입력된 텍스트 데이터 내에 존재하는 단어들 간의 관계**를 파악하기 위해 사용
    - 관련이 높은 단어에 더 많은 가중치를 주기 위해 사용
  - 인코더(또는 디코더)에서 사용 (인코더 기준으로 설명)
  - 인코더에 입력된 시퀀스 데이터에 존재하는 토큰들의 관계를 파악하기 위해 사용
  - 텍스트 데이터의 경우
    - 입력된 문서에서 사용된 단어들의 관계를 파악하기 위해 사용
    - 이러한 관계 정보를 파악하면, 문서의 전체적인 특성을 더 잘 파악할 수 있음

Self-attention

- Simple example
  - 입력 sequence = 'Today is Monday'
  - 각 단어에 대한 embedding vector (가정)
    - $\begin{bmatrix} e0 \\ e1 \\ e2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix}$
  - 예: 'Monday' 에 대한 self-attention 결과를 얻고자 하는 경우
  - 순서 (앞에서 설명한 attention 과 유사)
    - Attention score 계산 => 'Monday' 에 대한 embedding vector와 다른 단어들의 벡터들(자기자신포함)간의 내적 연산
    - 가중치의 값 계산 => Attention score를 softmax()에 적용해서 계산

Self-attention

- Simple example (cont'd)
  - 'Monday'에 대한 attention score

$$(1\ 0\ 0\ 1\ 1) \bullet (1\ 1\ 0\ 0\ 1) = 2$$
$$(1\ 0\ 0\ 1\ 0) \bullet (1\ 1\ 0\ 0\ 1) = 1$$
$$(1\ 1\ 0\ 0\ 1) \bullet (1\ 1\ 0\ 0\ 1) = 3$$

  - 가중치 계산

$$e^2 / (e^2 + e^1 + e^3) \approx 0.25$$

$$e^1 / (e^2 + e^1 + e^3) \approx 0.09$$

$$e^3 / (e^2 + e^1 + e^3) \approx 0.66$$

각 단어의 embedding vector에 가중치를 곱한 후, 더해서 self-attention의 결과값 도출

Attention in Transformer

Attention in Transformer

- Transformer에서의 self-attention (Encoder-decoder attention 도 유사하게 동작)
  - 앞의 예에서는 입력된 단어들의 임베딩 정보를 그대로 사용하여 attention score와 가중치를 구한다고 설명
  - Transformer의 self-attention은 입력 받은 단어들 중에서 어떠한 단어에 더 많은 가중치를 줘야 하는지 파악하기 위해서 각 단어들에 대한 Query, Key, Value 라고 하는 서로 다른 3개의 벡터들을 사용
    - Key, Value 벡터들은 사전 형태의 데이터 의미: key는 단어의 id와 같은 역할, value는 해당 단어에 대한 구체적 정보를 저장하는 역할을 한다고 생각
    - Query 벡터는 유사한 다른 단어를 찾을 때 사용되는 (질의) 벡터라고 생각 가능

Attention in Transformer

- Transformer에서의 self-attention (or attention)
  - 작동 순서
    - 단계1: 입력된 간 단어들에 대해서 Query, Key, Value 벡터를 계산
      - 이때 각각의 가중치 행렬이 사용됨
    - 단계2: Attention score 계산
      - Query를 이용해서 각 Key 들하고의 유사한 정도를 계산 ⇒ 내적 연산
    - 단계3: Attention score를 이용하여 가중치 계산
      - Softmax() 함수를 적용
    - 단계4: 가중치를 Value 벡터에 곱한다.
    - 최종 결과물
      - 가중치가 곱해진 Value 벡터들의 합

Attention in Transformer

- Query, Key, Value 벡터 구하기
  - 별도의 가중치 행렬 (weights matrix)을 사용
  - 입력된 각 단어의 임베딩 벡터와 가중치 행렬의 곱을 통해 Q, K, V 벡터를 구함.
  - Example
    - Embedding vector size = 5
    - Q, K, V vector size = 3 인 경우
    - 각 가중치 행렬의 크기는?

Query 가중치 행렬

$$\begin{bmatrix} W^Q_{0,0} & W^Q_{0,1} & W^Q_{0,2} \\ W^Q_{1,0} & W^Q_{1,1} & W^Q_{1,2} \\ W^Q_{2,0} & W^Q_{2,1} & W^Q_{2,2} \\ W^Q_{3,0} & W^Q_{3,1} & W^Q_{3,2} \\ W^Q_{4,0} & W^Q_{4,1} & W^Q_{4,2} \end{bmatrix}$$

Key 가중치 행렬

$$\begin{bmatrix} W^K_{0,0} & W^K_{0,1} & W^K_{0,2} \\ W^K_{1,0} & W^K_{1,1} & W^K_{1,2} \\ W^K_{2,0} & W^K_{2,1} & W^K_{2,2} \\ W^K_{3,0} & W^K_{3,1} & W^K_{3,2} \\ W^K_{4,0} & W^K_{4,1} & W^K_{4,2} \end{bmatrix}$$

Value 가중치 행렬

$$\begin{bmatrix} W^V_{0,0} & W^V_{0,1} & W^V_{0,2} \\ W^V_{1,0} & W^V_{1,1} & W^V_{1,2} \\ W^V_{2,0} & W^V_{2,1} & W^V_{2,2} \\ W^V_{3,0} & W^V_{3,1} & W^V_{3,2} \\ W^V_{4,0} & W^V_{4,1} & W^V_{4,2} \end{bmatrix}$$

Attention in Transformer

- Query, Key, Value 벡터 구하기 (cont'd)
  - Today = (1 0 0 1 1), is = (1 0 0 1 0), Monday = (1 1 0 0 1)
  - Today에 대해서 Attention 작업을 수행하는 경우
  - $\text{Query 가중치 행렬} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix}$
  - 단어1 (즉, Today)에 대한 Query 벡터 구하기
    - $[1\ 0\ 0\ 1\ 1] \times \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} = [2\ 0\ 2]$

Attention in Transformer

- Attention score 구하기: 단어1에 대해서
  - 단어 1에 대한 Query 벡터: $[2\ \ 0\ \ 2]$
  - 단어 1에 대한 Key 벡터: $[3\ \ 2\ \ 1]$
  - 단어 2에 대한 Key 벡터: $[2\ \ 2\ \ 0]$
  - 단어 3에 대한 key 벡터: $[2\ \ 1\ \ 2]$
  - 내적 연산
    - $[2\ \ 0\ \ 2] \bullet [3\ \ 2\ \ 1] = 2\times3 + 2\times1 = 8$
    - $[2\ \ 0\ \ 2] \bullet [2\ \ 2\ \ 0] = 4$
    - $[2\ \ 0\ \ 2] \bullet [2\ \ 1\ \ 2] = 8$
  - Softmax() 적용
    - softmax([8 4 8]) = [0.5 0 0.5]
      - $e^8 / (e^8+e^4+e^8) \approx 0.5$
      - $e^4 / (e^8+e^4+e^8) \approx 0.0$

Key 가중치 행렬

$$\begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Attention in Transformer

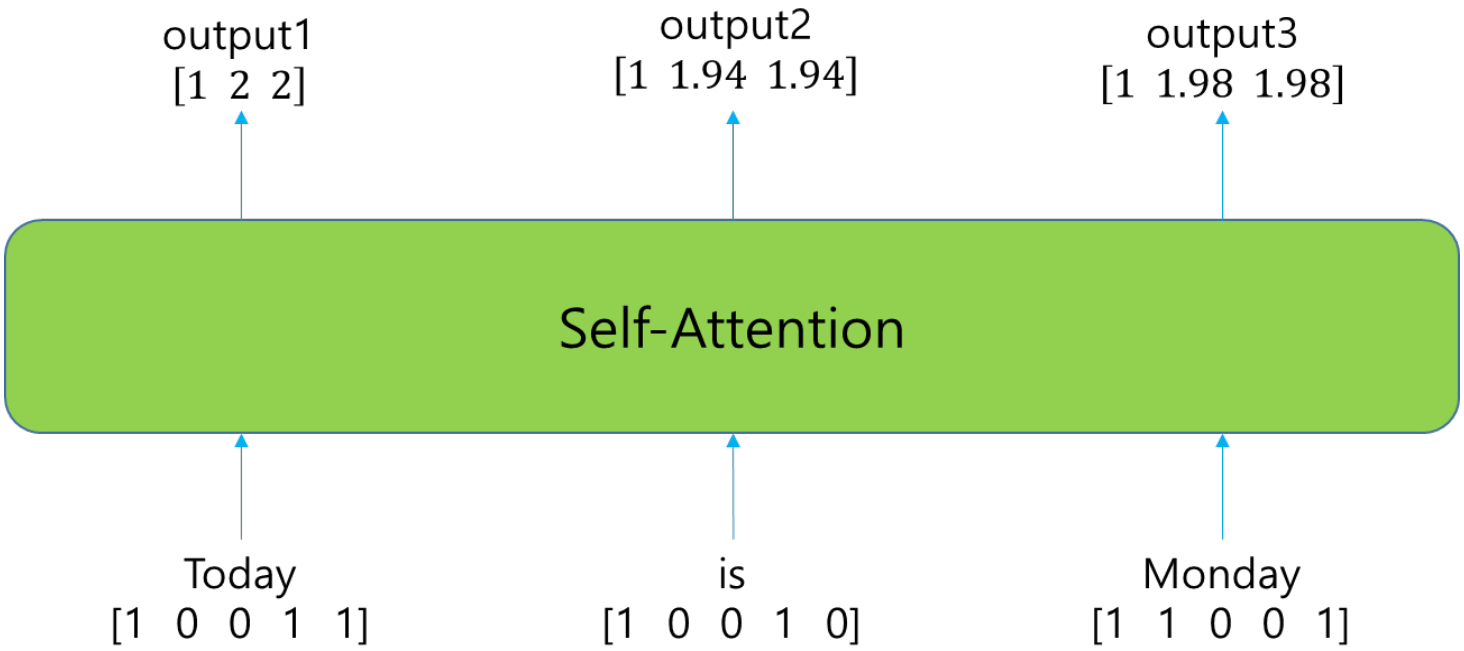
- Value vectors에 가중치 곱하기
  - 단어1의 value 벡터: $[1\ \ 2\ \ 2]$
  - 단어2의 value 벡터: $[1\ \ 1\ \ 1]$
  - 단어3의 value 벡터: $[1\ \ 2\ \ 2]$
  - $0.5\times[1\ \ 2\ \ 2] = [0.5\ \ 1\ \ 1]$
  - $0.0\times[1\ \ 1\ \ 1] = [0\ \ 0\ \ 0]$
  - $0.5\times[1\ \ 2\ \ 2] = [0.5\ \ 1\ \ 1]$
  - 최종 결과물: 각 벡터의 합
    - $[0.5\ \ 1\ \ 1] + [0\ \ 0\ \ 0] + [0.5\ \ 1\ \ 1] = [1\ \ 2\ \ 2]$
- 동일한 작업을 단어2, 3에 대해서 수행

Value 가중치 행렬

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

Attention in Transformer

- 최종 결과물



Attention in Transformer

- 수식으로 표현하기
  - $\text{Attention}(Q, K, V) = \text{Softmax}(QK^T)V$
    - Q는 Query 벡터들에 대한 행렬,
    - K는 Key 벡터들에 대한 행렬,
    - V는 Value 벡터들에 대한 행렬
    - K^T는 K 행렬의 전치행렬 (Transpose)
    - Q와 K^T의 곱하기는 Query 벡터들과 Key 벡터들의 내적연산을 의미

Attention in Transformer

- $\mathrm{Attention}(\mathrm{Q},\mathrm{K},\mathrm{V}) = \mathrm{Softmax}\left(\mathrm{QK}^{\mathrm{T}}\right)\mathrm{V}$ (cont'd)
  - 앞의 예 (cont'd)

$$\mathrm{Q} = \begin{bmatrix} 2 & 0 & 2 \\ 1 & 0 & 1 \\ 1 & 0 & 2 \end{bmatrix}, \mathrm{K} = \begin{bmatrix} 3 & 2 & 1 \\ 2 & 2 & 0 \\ 2 & 1 & 2 \end{bmatrix}, \mathrm{V} = \begin{bmatrix} 1 & 2 & 2 \\ 1 & 1 & 1 \\ 1 & 2 & 2 \end{bmatrix}$$

$$, \mathrm{K}^{\mathrm{T}} = \begin{bmatrix} 3 & 2 & 2 \\ 2 & 2 & 1 \\ 1 & 0 & 2 \end{bmatrix}$$

첫번째 단어에 대한 attention score

$$\mathrm{QK}^{\mathrm{T}} = \begin{bmatrix} 2 & 0 & 2 \\ 1 & 0 & 1 \\ 1 & 0 & 2 \end{bmatrix} \times \begin{bmatrix} 3 & 2 & 2 \\ 2 & 2 & 1 \\ 1 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 8 & 4 & 8 \\ 4 & 2 & 4 \\ 5 & 2 & 6 \end{bmatrix}$$

Attention in Transformer

- $\text{Attention}(\text{Q}, \text{K}, \text{V}) = \text{Softmax}(\text{QK}^{\text{T}})\text{V}$ (cont'd)
  - 앞의 예 (cont'd)

$$\text{Softmax}(\text{QK}^{\text{T}}) = \text{Softmax}\left(\begin{bmatrix} 8 & 4 & 8 \\ 4 & 2 & 4 \\ 5 & 2 & 6 \end{bmatrix}\right) \approx \begin{bmatrix} 0.5 & 0 & 0.5 \\ 0.47 & 0.06 & 0.47 \\ 0.27 & 0.01 & 0.72 \end{bmatrix}$$

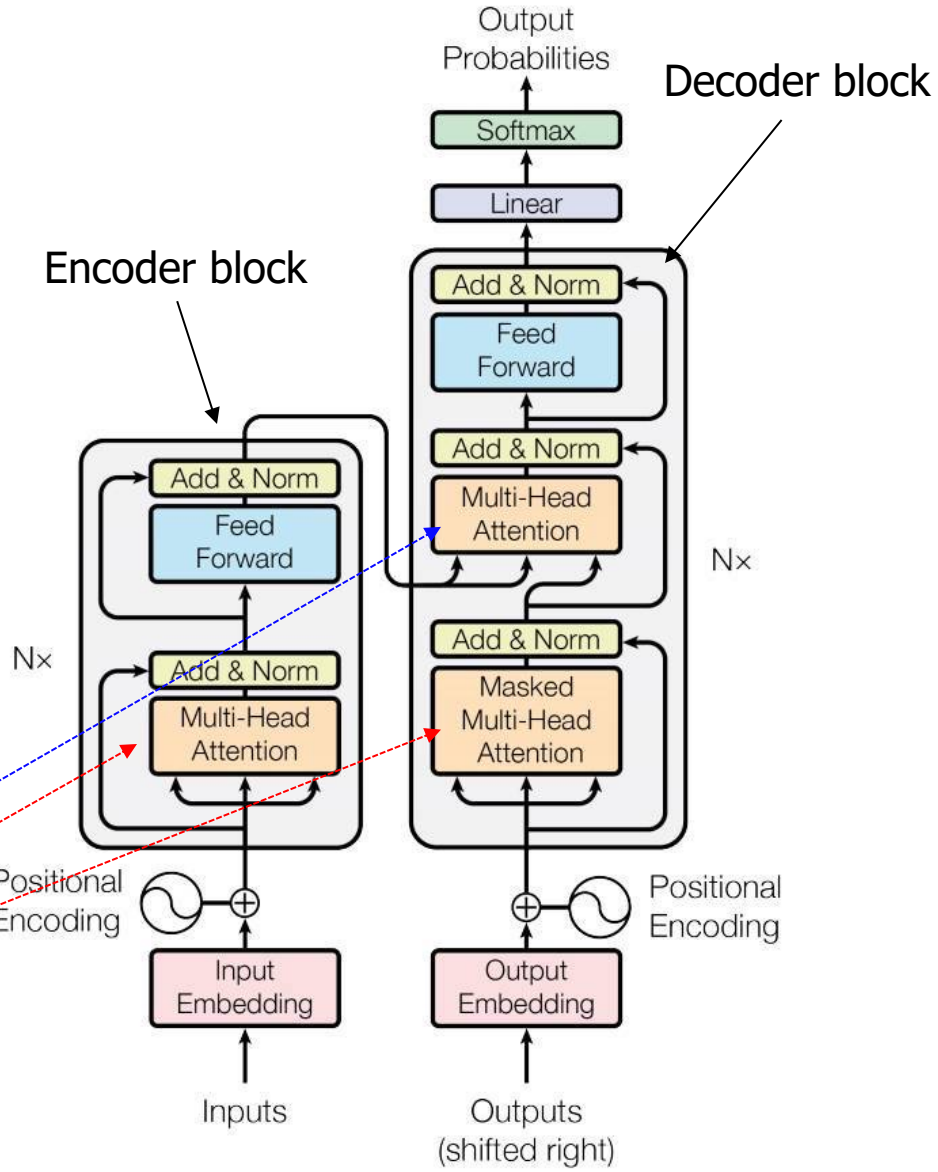
$$\text{Attention}(\text{Q}, \text{K}, \text{V}) = \text{Softmax}(\text{QK}^{\text{T}}) \times \text{V}$$
$$\approx \begin{bmatrix} 0.5 & 0 & 0.5 \\ 0.47 & 0.06 & 0.47 \\ 0.27 & 0.01 & 0.72 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 2 \\ 1 & 1 & 1 \\ 1 & 2 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 2 \\ 1 & 1.94 & 1.94 \\ 1 & 1.99 & 1.99 \end{bmatrix}$$

Transformer

Transformer

- Attention 기반의 encoder-decoder 모형
- 전체적인 구조
  - Encoder block과 decoder block을 여러개 (N개, 해당 논문[1]에서는 6개) 쌓아서 encoder 부분과 decoder부분을 생성
  - Decoder 부분은 언어모형으로 작동

중요 components
- Encoder-decoder attention
- Self-attention



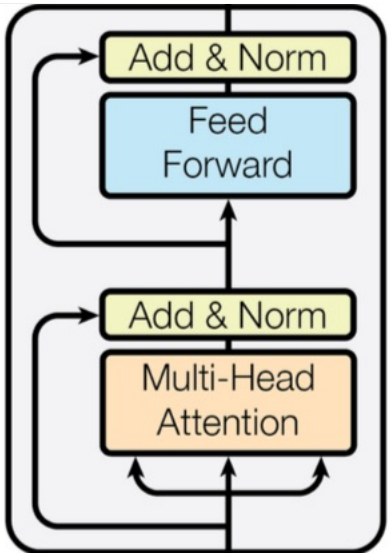
[1] Vaswani et al. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008).

Encoder 부분

Transformer

- Encoder 부분
  - Encoder block
    - 이를 여러 개 사용 (N=6)
  - Multi-Head Attention
    - 여기서의 Attention은 Self-attention을 의미, 즉, 입력된 sequence data에 대한 self-attention 적용
    - Multi-head는 self-attention을 여러 개 적용했다는 것을 의미

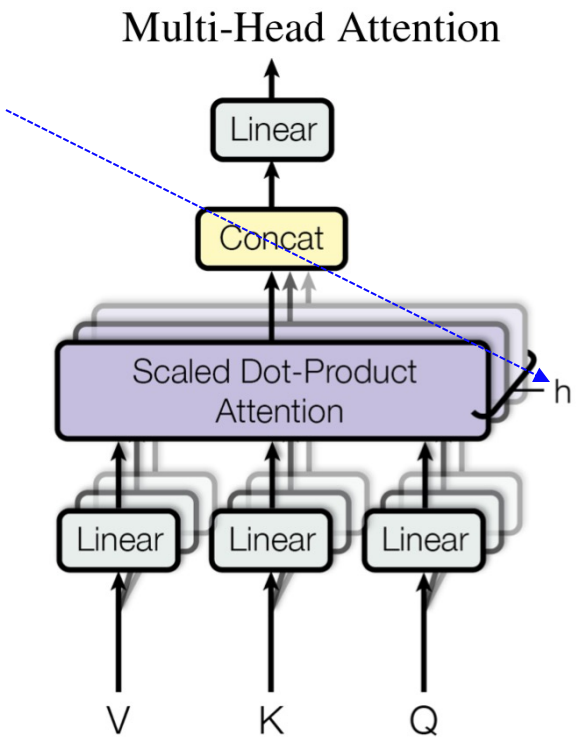
N x

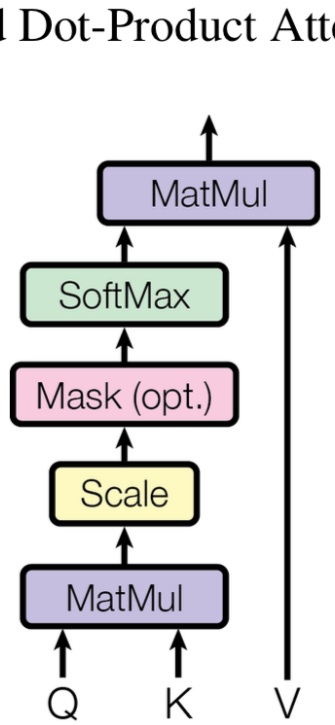


Transformer

- Details of Multi-head attention

h = # of heads
해당 논문에서는 h=8





Transformer

- Equation of scaled dot-product attention
  - $\text{Attention}(\text{Q}, \text{K}, \text{V}) = \text{Softmax}\left(\frac{\text{QK}^{\text{T}}}{\sqrt{d_k}}\right)\text{V}$
    - d_k=Key vector의 크기 (in the paper, 64)
- Equation of multi-head attention
  - $\text{MultiHead}(\text{Q}, \text{K}, \text{V}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_\text{h})W^O$,where $\text{head}_\text{i} = \text{Attention}(\text{QW}_\text{i}^\text{Q}, \text{KW}_\text{i}^\text{K}, \text{VW}_\text{i}^\text{V})$
  - Transformer에서 h = 8
  - Q, K, V를 바로 사용하지 않고 linear projection을 한번 더 해줬음
  - Q, K, V에 속한 query, key, value 벡터들의 차원은 d_{model} (=512)으로 동일
  - $\text{W}_\text{i}^\text{Q}, \text{W}_\text{i}^\text{K}, \text{W}_\text{i}^\text{V}$ 의 가중치 행렬을 이용해서 특정한 차원의 크기가 다른 query, key, value 벡터로 변환
  - query, key 벡터의 차원은 d_k, value 벡터의 차원은 d_v로 표현 ⇒ $d_k = d_v = 64$ 로 동일하게 설정

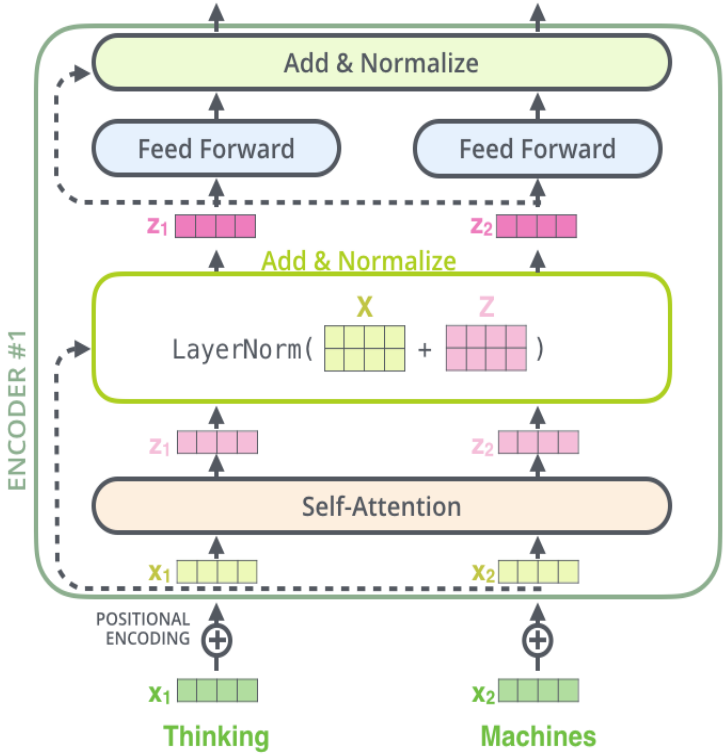
512x64

Transformer

- Equation of multi-head attention (cont'd)
  - Transformer에서 h = 8
  - 사용 이유
    - 주목해야 하는 다른 단어가 무엇인지를 더 잘 파악
    - 각 단어가 갖고 있는 (문맥적) 특성을 더 잘 표현
  - 각 self-attention의 결과물 이어붙이기 (concat)
    - MSA에서 출력되는 각 토큰의 결과물: 64*8 = 512
  - 그 다음 다시 한번 Linear
    - W^O: 512x512

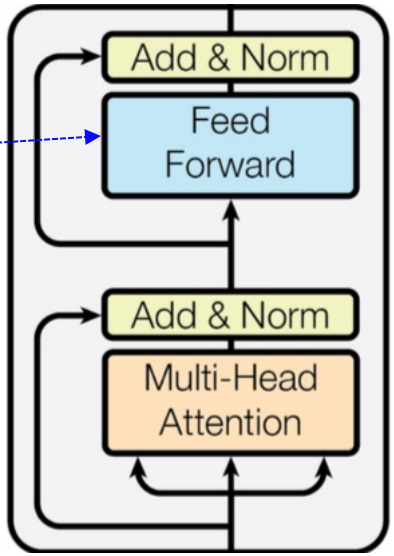
Transformer

- Add & Norm layer
  - Add: 이 부분에서는 self-attention 층이 출력하는 값과 self-attention에 입력된 값을 더함
    - ResNes에서 사용한 skip connection 과 유사
  - Layer normalization: Add 연산 결과에 layer norm 적용, 이는 각 관측치에 대해 하나의 레이어에 존재하는 노드들의 입력값들의 정보를 이용해서 표준화하는 것을 의미



<출처: http://jalammar.github.io/illustrated-Transformer/>

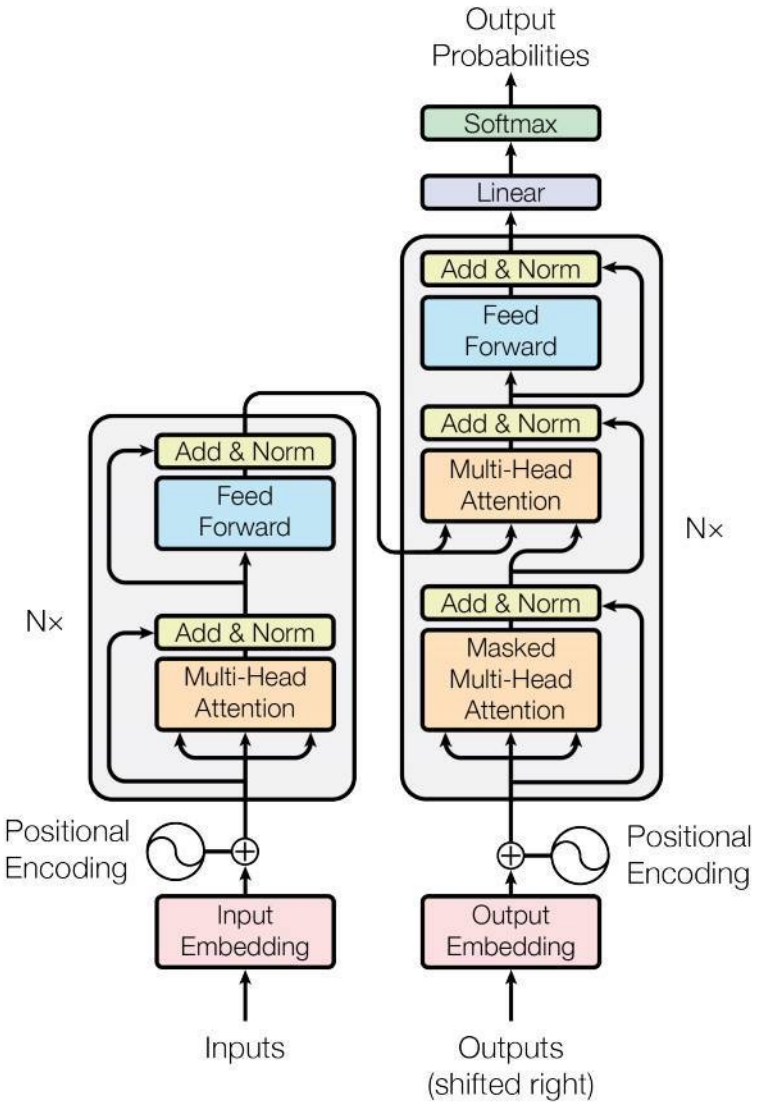
Transformer



- Position-wise feed-forward network
  - 2개의 fully connected layer (혹은 dense layer)로 구성
  - 이를 token 마다 (즉, position 마다) 적용
  - 가중치 공유
  - 첫번째 layer에 ReLU 활성화함수 사용
  - $\mathrm{FFN}(x) = \max(0, xW_1 + b_1)\,W_2 + b_2$
    - x는 첫번째 FCL에 입력되는 입력 벡터를 의미
    - W_1는 입력벡터와 첫번째 FCL 사이의 가중치 행렬
    - W_2는 첫번째 FCL과 두번째 FCL 사이의 가중치 행렬

Transformer

- 위치정보 임베딩 (Positional embedding)
  - Transformer 모형에서는 단어들의 embedding 정보만을 사용하는 것이 아니라 (단어들이 갖는 입력된 시퀀스 데이터 내에서의 위치 (position) 정보도 사용
  - 위치 정보를 사용하게 되면, 단어들 간의 위치를 파악함으로써 단어들 간의 상대적인 거리를 파악
  - 이를 사용하는 주된 이유는, Transformer는 RNN이나 LSTM와 같은 순환신경망 구조 (단어들이 순서대로 입력)를 사용하지 않고 attention 방법을 사용하기 때문
  - 단어들이 갖는 (상대적인) 위치 정보를 반영하기 위해서 위치 정보를 반영하는 벡터를 사용하는데 이를 positional embedding 벡터



Transformer

- 위치정보 임베딩 (Positional embedding) (cont'd)

최종 embedding	(e0,0, e0,1, …, e0,n)	(e1,0, e1,1, …, e1,n)	(e2,0, e2,1, …, e2,n)
	=	=	=
Positional embedding	(p0,0, p0,1, …, p0,n)	(p1,0, p1,1, …, p1,n)	(p2,0, p2,1, …, p2,n)
	+	+	+
원래의 embedding	(x0,0, x0,1, …, x0,n)	(x1,0, x1,1, …, x1,n)	(x2,0, x2,1, …, x2,n)
입력단어	단어0	단어1	단어2

Transformer

- 위치정보 임베딩 (Positional embedding) (cont'd)
  - How to compute?
    - sin(), cos() 함수 사용
    - 아래 공식 사용

$$PE_{(i,j)} = \sin\left(\frac{i}{10000^{j/emb\_dim}}\right), \qquad \text{j가 짝수인 경우}$$

$$PE_{(i,j)} = \cos\left(\frac{i}{10000^{j-1/emb\_dim}}\right), \qquad \text{j가 홀수인경우}$$

    - 여기서 $PE_{(i,j)}$는 단어 i의 positional embedding vector의 위치 j의 원소값을 의미

Transformer

- 위치정보 임베딩 (Positional embedding) (cont'd)
  - 예) ['Today', 'is', 'Monday']에 대한 positional embedding 벡터
    - emb_dim = 512 인 경우

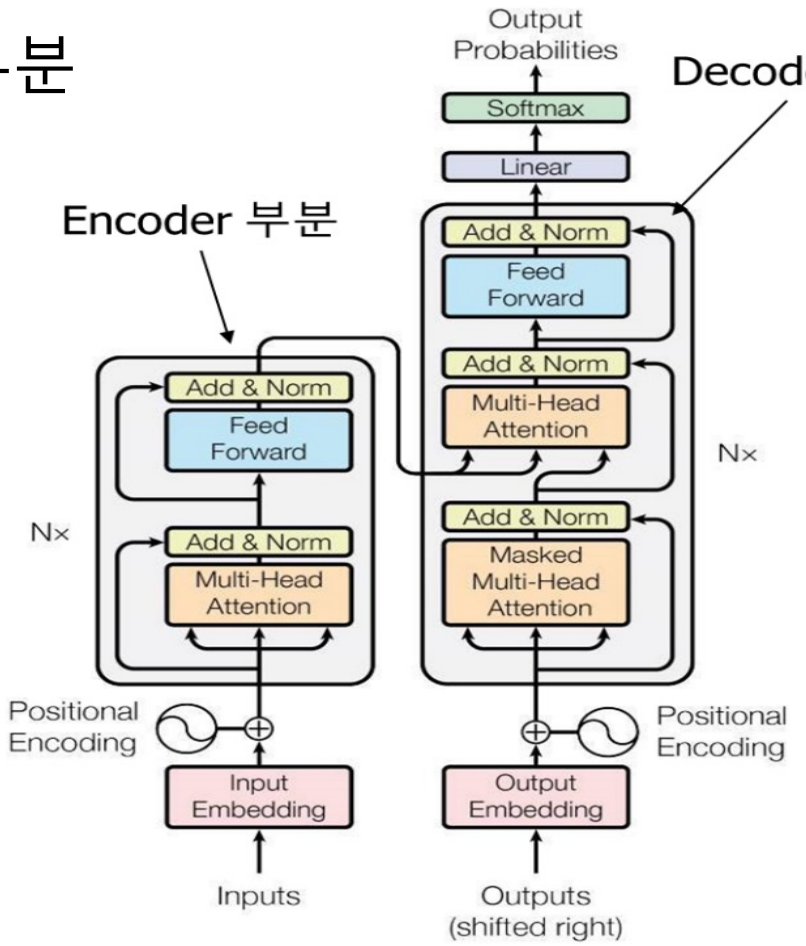
$$
\begin{array}{c}
 \\
Today\ (i=0) \\
is \quad (i=1) \\
Monday\ (i=2)
\end{array}
\begin{array}{c}
\begin{array}{cccc} j=0 & & j=1 & \end{array} \\
\begin{bmatrix}
sin\left(\frac{0}{10000^{\frac{0}{512}}}\right) & cos\left(\frac{0}{10000^{\frac{0}{512}}}\right) & \cdots \\
sin\left(\frac{1}{10000^{\frac{0}{512}}}\right) & cos\left(\frac{1}{10000^{\frac{0}{512}}}\right) & \cdots \\
sin\left(\frac{2}{10000^{\frac{0}{512}}}\right) & cos\left(\frac{2}{10000^{\frac{0}{512}}}\right) & \cdots
\end{bmatrix}
\end{array}
$$

    - 이는 학습되는 것이 아니라, 주어진 공식에 따라 미리 계산된다.

Decoder 부분

Transformer

- Decoder 부분



6개의 decoder block 사용

Transformer

- Decoder 부분
  - 작동 방식
    - seq2seq의 decoder와 비슷한 역할
      - 즉, encoder에 입력된 sequence에 대한 또 다른 sequence data를 출력
      - 텍스트의 경우, 언어 모형이라고 생각할 수 있음
- Decoder block
  - 전체적인 구조는 encoder block과 유사
  - 주요 차이: 두 가지 종류의 attention 사용
    - Masked self-attention
    - Encoder-decoder attention

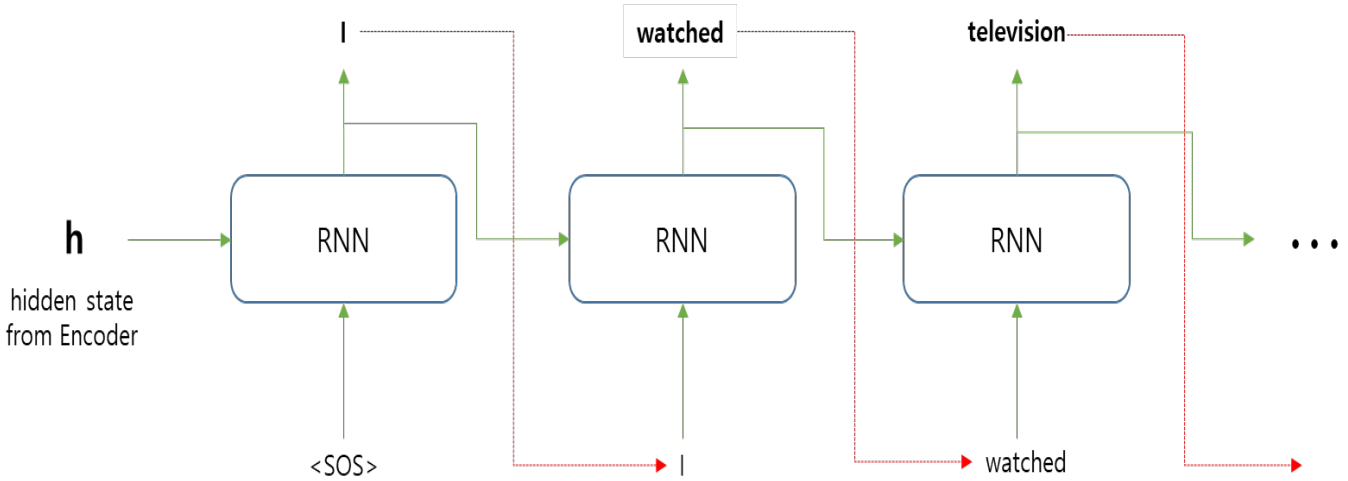
Transformer

- Masked self-attention
  - Encoder의 self-attention과 약간 다르게 작동 ⇒ 이해하기 위해서는 학습의 단계에서 Transformer의 decoder 부분이 어떻게 작동하는지를 먼저 알아야 함
  - Teacher forcing 방법 사용
    - Decoder는 언어모형의 역할을 하지만, **학습의 단계**에서는 모형이 현재 단계까지 예측한 단어들의 정보를 사용하여 다음 단어를 예측하는 것이 아니라, 정답 데이터 정보를 이용해서 각 단계의 단어들을 예측하는 것
    - 모형이 예측한 단어들의 정보를 이용해서 다음 단어를 예측하는 경우에는 이전 단어들에 대한 예측이 잘못되면 그 다음 단어에 대한 예측이 제대로 될 수 없기 때문
    - 예: '오늘은 금요일 입니다'을 'Today is Friday'로 번역하는 경우
      - 첫단어를 'Today' 라고 예측하지 못하면, 그 다음 단어인 'is'를 제대로 예측할 수 없다.
      - 이를 위해 학습에서는 실제 정답 데이터를 사용하여 학습한다.
      - 앞에 <SOS> 토큰을 더해서 사용 ⇒ 논문에서는 right shifted output이라고 표현

Transformer

- Teacher forcing (cont'd)
  - 예: RNN 기반의 seq2seq
    - 'I 나는 어제 골프를 쳤다'를 'I played golf yesterday'로 번역하는 경우

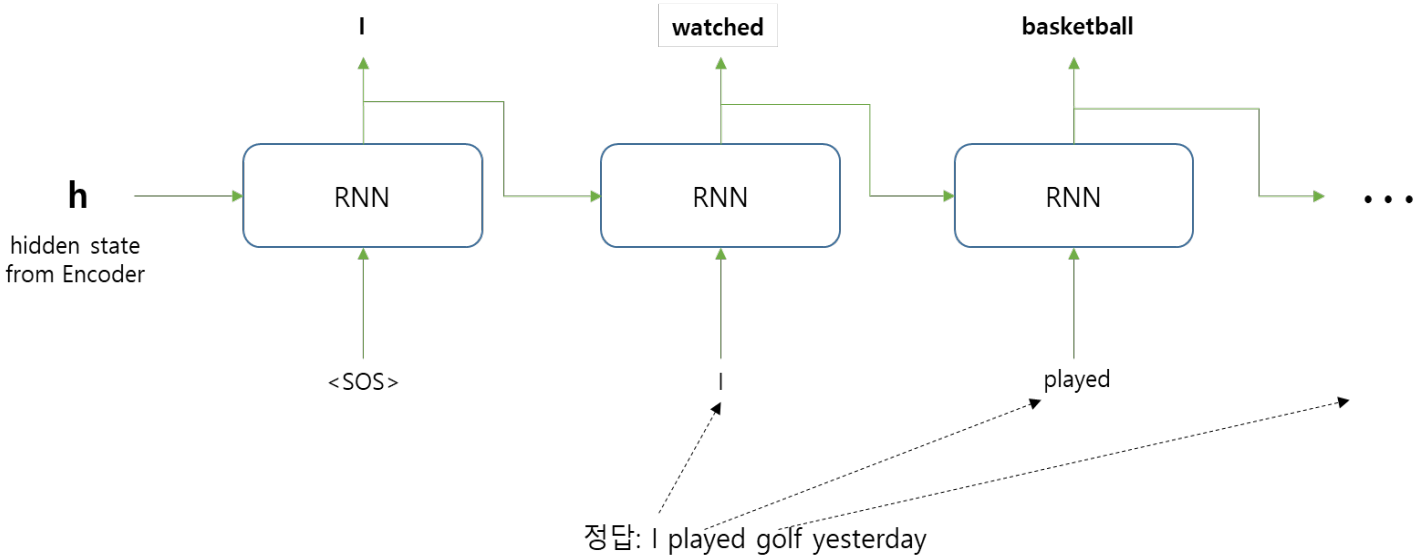
정답



<Teacher forcing을 사용하지 않는 경우, 즉, 예측된 단어 정보를 이용해서 그 다음 단어를 예측하는 경우>

Transformer

- Teacher forcing (cont'd)
  - 이러한 문제를 방지하기 위해 teacher forcing 방법 사용
  - 모형에서 예측한 단어를 이용해서 다음 단어를 예측하는 것이 아니라, 모형에서 예측한 단어가 무엇인지와는 상관없이 정답 단어를 이용해서 다음 단어를 예측하는 방식



Transformer

- Masked self-attention (cont'd)
  - 예: '오늘은 금요일 입니다'를 'Today is Friday'로 번역하는 경우
    - <SOS>, 'Today', 'is', 'Friday' 을 학습 데이터로 사용하여 각 단계에서의 단어를 예측
    - 각 토큰의 embedding 정보와 positional embedding 정보를 사용해서 query, key, value 벡터를 구하고 attention score를 계산하게 됨
    - 다음과 같이 attention score가 구해졌다고 가정

	<SOS>	This	is	Friday
<SOS>	7	2	2	2
This	1	6	2	4
is	1	2	8	1
Friday	1	4	2	6

단어 'This'에 대한 attention score

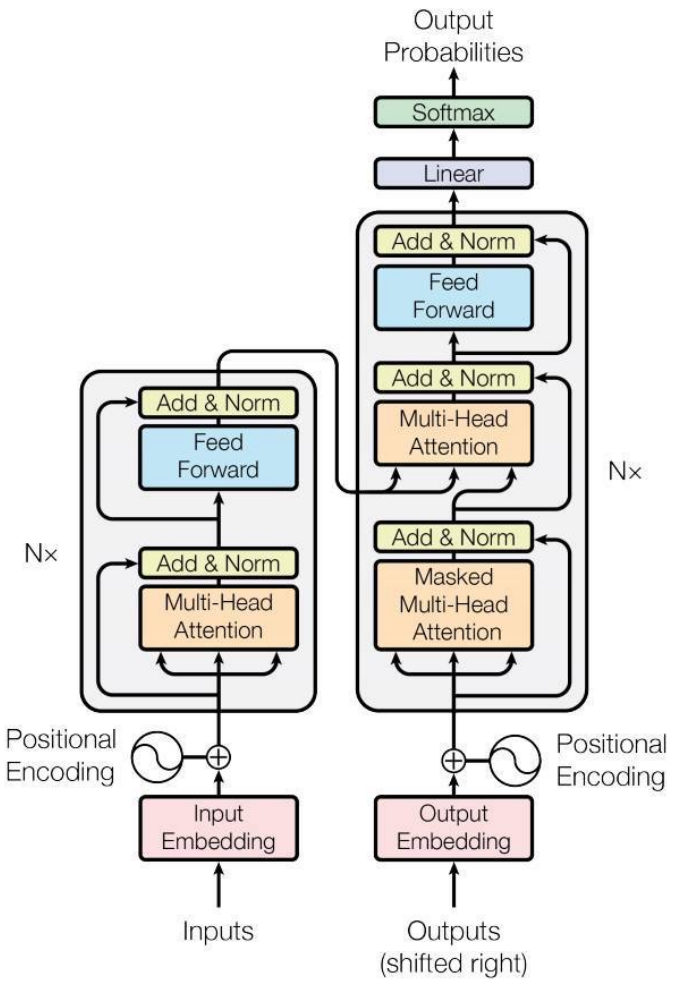
Transformer

- Masked self-attention (cont'd)
  - Decoder 는 언어 모형으로 작동하기 때문에 특정 단어의 attention score 중에서 자기 자신 다음에 나오는 단어에 대한 정보를 사용할 수 없음
  - 예: 두번째 단어(즉, is)까지의 정보를 사용해서 세번째 단어를 예측하는 경우
    - 이때는, is 단어에 대한 self-attention 결과를 사용해야 함
    - decoder 는 언어모형이기 때문에 is 다음에 나오는 단어들의 정보를 사용할 수 없음 (is 다음에 나오는 단어들은 아직 예측이 되지 않아 그 정보를 알지 못하는 것)
    - 즉, is 까지의 단어/토큰인 <SOS>, This, is 에 대한 정보만을 사용해서 attention score를 구하고, 그 정보를 이용해서 계산된 가중치를 value 벡터에 곱해서 attention 결과물을 얻어야 함
    - 이를 위해서 기준이 되는 단어 이후에 나오는 단어들에 대한 attention score를 오른쪽과 같이 -∞로 대체 ⇒ softmax() 함수의 값이 0이 됨
    - 따라서 value 벡터에 0이 곱해짐

	<SOS>	This	is	Friday
<SOS>	7	-∞	-∞	-∞
This	1	6	-∞	-∞
is	1	2	8	-∞
Friday	1	4	2	6

Transformer

- Encoder-decoder attention
  - Self-attention과 마찬가지로 query, key, value 벡터들을 사용
    - query 벡터는 decoder 부분에 입력된 단어에 대한 query 벡터
      - 디코더 부분에 입력된 단어의 query 벡터를 계산하기 위해서 encoder-decoder attention 밑에 있는 Add & Norm 층이 출력하는 결과를 이용
    - key와 value 벡터는 encoder 부분에서 (encoder 부분에 입력된) 각 단어에 대한 값으로 전달
      - Encoder 부분의 마지막 encoder block에서 출력하는 각 단어들에 대한 hidden state 정보를 사용해서 계산
    - 작동하는 방식은 앞에서 설명한 self-attention과 동일



Transformer

- 학습
  - 학습 데이터
    - WMT 2014 English-German dataset consisting of about 4.5M sentence pairs
      - Byte Pair Encoding
    - WMT 2014 English-French dataset consisting of 36M sentence pairs
      - WordPiece Encoding
  - Optimizer
    - Adam optimizer with $\beta_1 = 0.9, \beta_2 = 0.98$
    - Dropout 적용 (10%)

Transformer

- 성능

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

Transformer

- Model variations

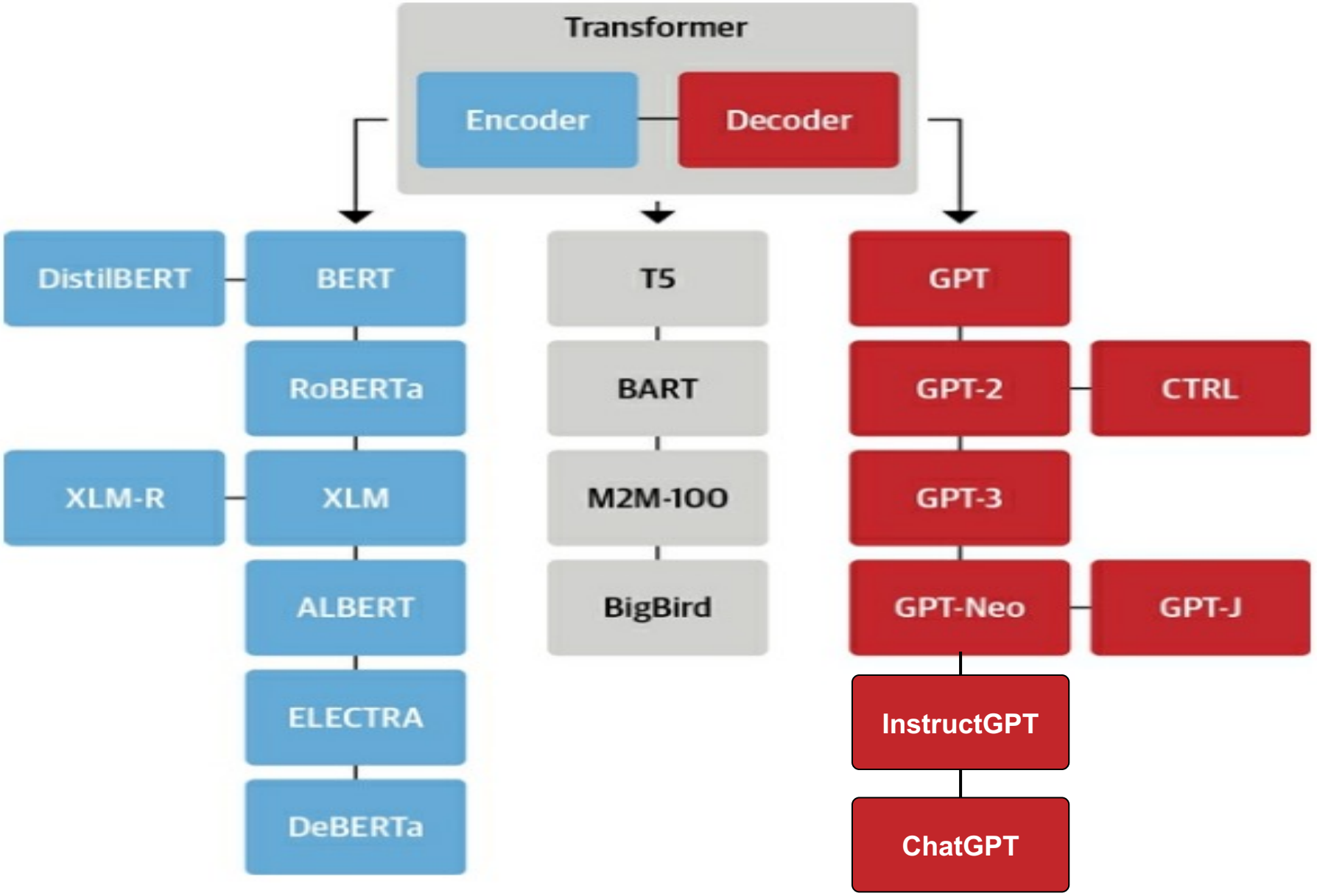
	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

Transformer

- Encoder block을 사용한 감성분석
  - English
    - Transformer_imdb_example.ipynb 참고
  - 한글
    - Transformer_sentiment_Korean.ipynb 참고

Transformer 응용

- 대표적 모형들
  - BERT (Bidirectional Encoder Representations from Transformers)
    - Transformer의 encoder 부분만을 사용
    - 주요 사용 용도
      - 문서/단어 임베딩, 문서 분류, Q&A, 단어들 간의 관계 추출
  - GPT (Generative Pre-trained Transformer)
    - Transformer의 decoder 부분만을 사용
    - 주요 사용 용도
      - 텍스트 생성
  - BART (Bidirectional and Auto-Regressive Transformers)
    - Transformer의 encoder와 decoder 모두 사용
    - 주요 사용 용도
      - 텍스트 요약 등
  - T5 (Text-to-Text Transfer Transformer)
    - Transformer의 encoder와 decoder 모두 사용



<Source: Tunstall et al. (2022). NLP with Transformers>