

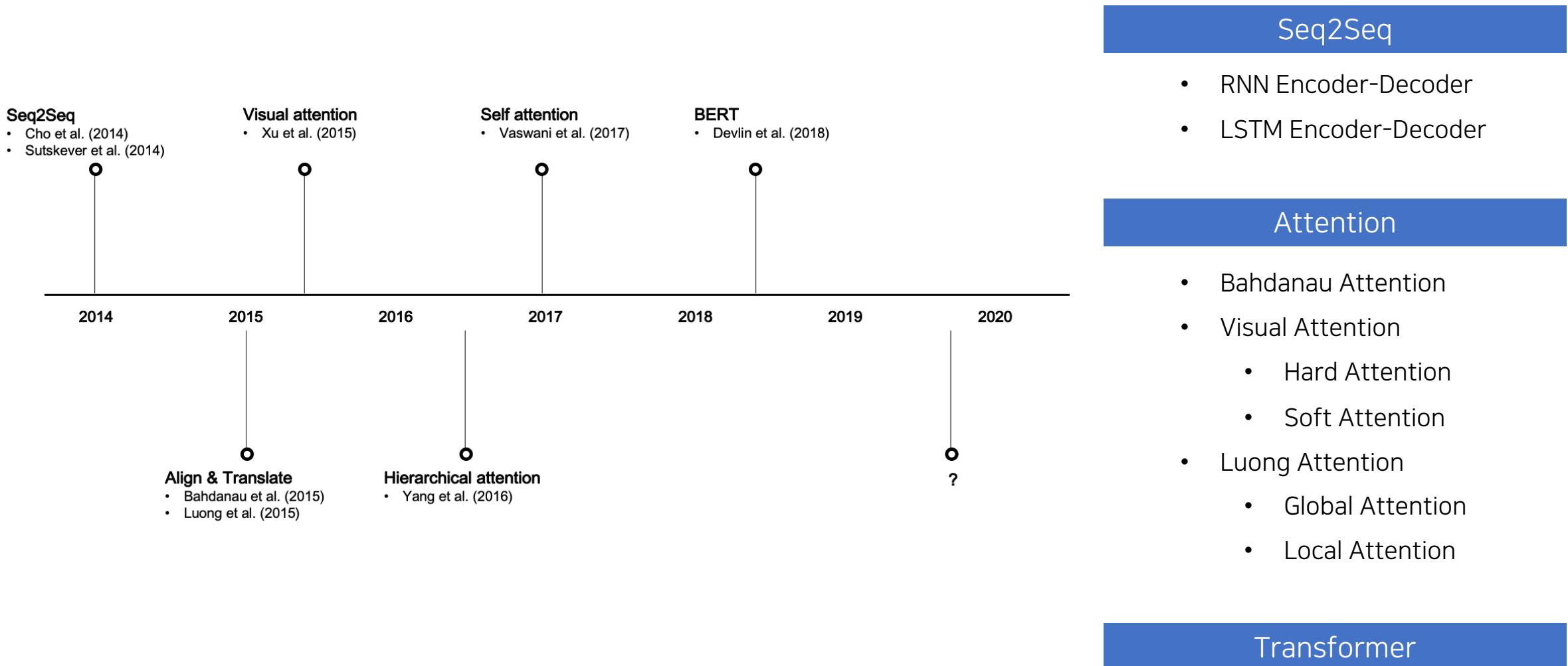
# Attention Timeline 정리

---

2020/10/6

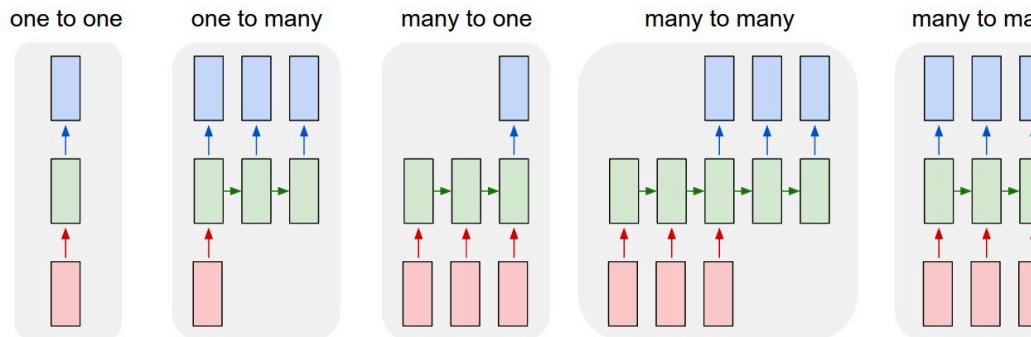
배현진

# 0. Timeline

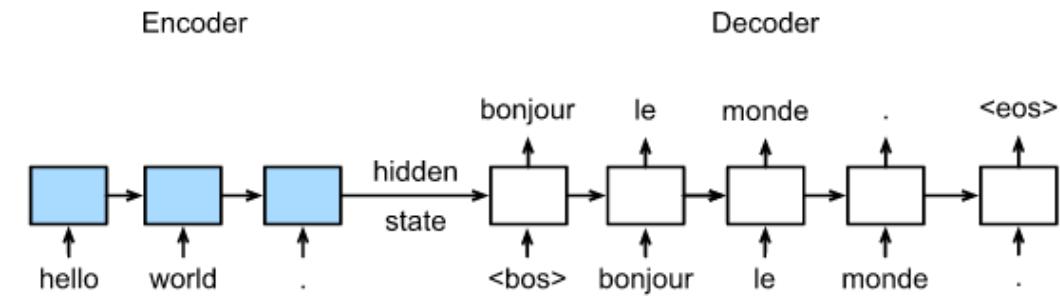


# 1. Seq2Seq

- 대부분의 텍스트 데이터는 **Sequence**로 구성됨.
  - eg) 단어: sequence of characters, 문장: sequence of words, 문단: sequence of sentences ...
- **Seq2Seq**: 한 시계열 데이터를 다른 시계열 데이터로 변환하는 기법
  - input sequence(source)를 output sequence(target)으로 변환하는 기법(source 길이 ≠ target 길이)
  - Encoder와 Decoder, 두 부분으로 구성되어 **하나의 sequence를 다른 sequence로 맵핑**하는 모델



Seq2Seq

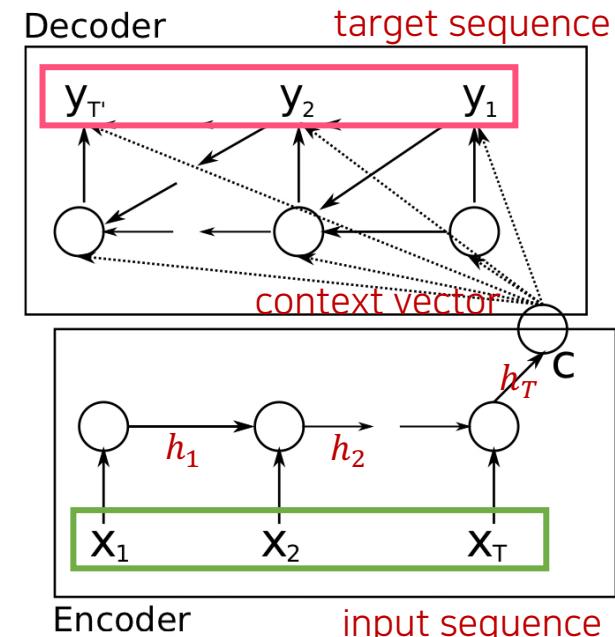


# 1. Seq2Seq

Cho et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation(2014)

- **RNN Encoder-Decoder 모델:**

- 두개의 RNN을 사용해
  - 1) 하나로는 **입력 시퀀스를 고정 길이 벡터로 Encode**하고
  - 2) 다른 하나로는 **고정 길이의 벡터를 출력 시퀀스로 Decode**하도록 학습시키는 신경망 네트워크 구조를 고안
- **인코더**: 입력 시퀀스의 각 단어를 연속적으로 읽을 수 있는 RNN으로 구성
  - 단어를 읽으며 시각 t의 은닉 상태  $h_t$ 를 계산:  $h_t = f(h_{t-1}, x_t)$
  - 전체 input sequence를 다 읽고 난 뒤 마지막 RNN의 은닉 상태( $h_T$ )
    - 전체 시퀀스에 대한 요약, **문맥 벡터 c**가 된다.



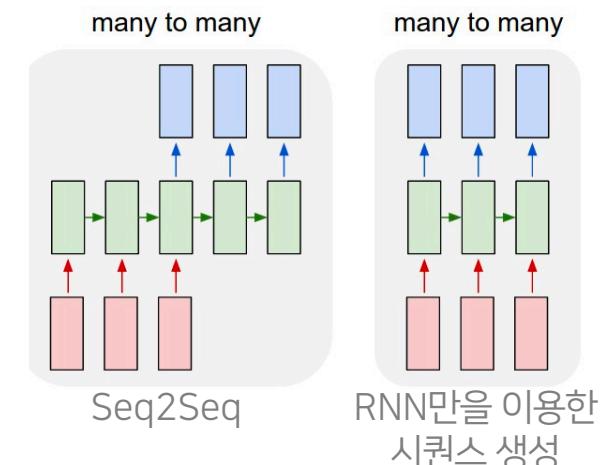
- **디코더**: 은닉 상태  $h_t$ 에 대해 타깃 단어  $y_t$ 를 예측할 수 있는 RNN으로 이뤄진 구조

- RNN: 현재 시각의 입력과 이전 시각의 은닉 상태를 이용해 target을 만들어냄  $h_t = f(h_{t-1}, x_t)$

- RNN 인코더-디코더:

- 이전 시각의 출력 단어와 은닉 상태, 문맥 벡터  $\rightarrow$  현재 시각의 출력 단어와 은닉 상태

$$h_t = f(h_{t-1}, y_{t-1}, c) \quad P(y_t | y_{t-1}, y_{t-2}, \dots, y_1, c) = g(\mathbf{h}_{\langle t \rangle}, y_{t-1}, c)$$



# 1. Seq2Seq

Sutskever et al. Sequence to Sequence Learning with Neural Networks(2014)

- **LSTM Encoder-Decoder 모델**

- 기존의 RNN 인코더-디코더 모델과 다른 점:

- 1. 인코더와 디코더를 **LSTM**으로 구성

- Long range temporal dependency에 강한 LSTM을 사용하면 RNN을 사용한 기존 모델보다 **긴 시퀀스에 잘 동작**할 것!

- 2. input 문장의 **순서를 reverse**

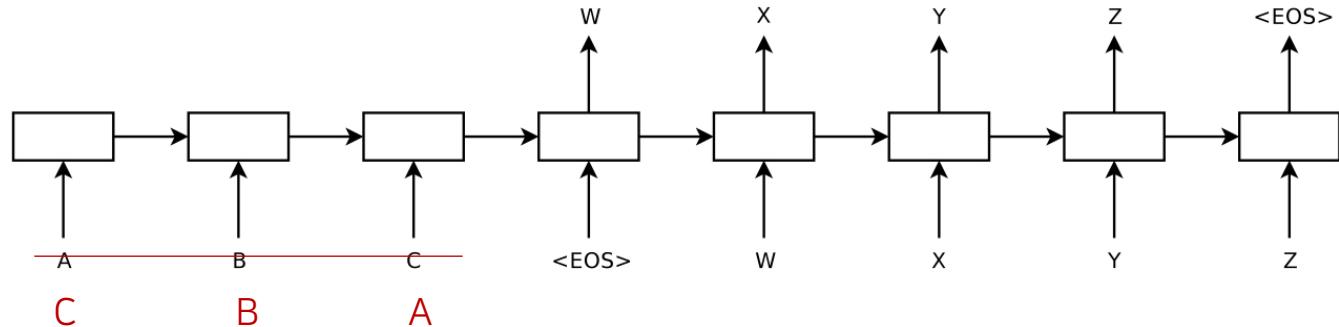
- 문장 (a, b, c) => 문장 (a, β, γ)로 번역한다고 할 때, input 문장의 순서를 바꿔 (c, b, a)로 입력

- a는 a와, b는 β와 거리상 가까워지게 되므로 input과 output 사이의 **관계를 더 잘 포착**할 수 있다.

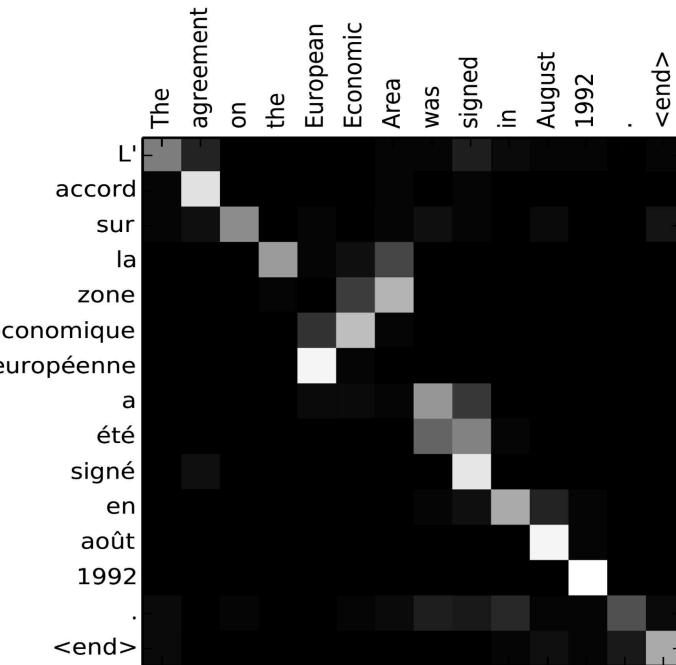
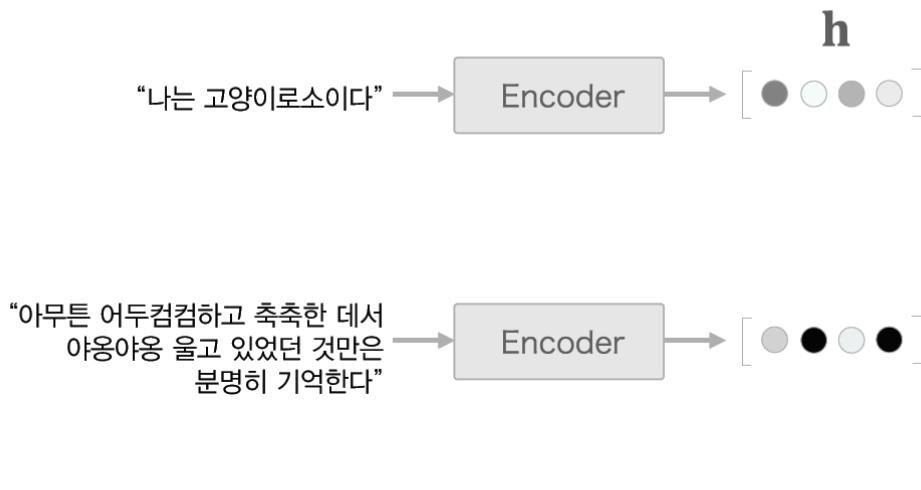
- 구조는 기존과 마찬가지로 입력 시퀀스와( $x_1, \dots, x_T$ )와 그에 대응하는 출력 시퀀스( $y_1, \dots, y_{T'}$ )에 대한 조건부 확률  $p(y_1, \dots, y_{T'} | x_1, \dots, x_T)$ 를 학습

- 1. LSTM의 마지막 은닉 상태를 이용해 입력 시퀀스로부터 고정 길이 벡터  $v$ 를 얻어낸다.

- 2. 디코더의 첫 은닉 상태(initial hidden state)를  $v$ 로 두고 현재 시각의 타깃 단어를 조건부확률을 이용해 계산한다.



## 2. Attention



- Seq2Seq:
  - 시계열 데이터를 다루기 위해 input sequence를 고정 길이 벡터로 맵핑해 디코더에 전달
  - 입력 문장의 길이에 상관 없이 문장의 모든 정보를 항상 같은 길이의 벡터에 밀어 넣어야 하기 때문에 문제가 발생
- Attention:
  - 입력 문장에서 가장 관련된 정보가 집중되어 있는 부분을 찾아 이를 고려해 출력 시퀀스를 만들어내는 기법
  - "저는 맥주로 하겠습니다."를 "I'd like a beer"로 번역 시, 'beer'를 예측할 때 '맥주'에 주목하도록 한다.

## 2. Attention

Bahdanau et al. Neural Machine Translation by Jointly Learning to Align and Translate(2015)

- Encoder-Decoder Model with Attention Mechanism

- Encoder

- Bidirectional RNN을 사용해 단어 앞 뒤에 대한 정보를 모두 포함

$$h_j = [\vec{h}_j^\top; \overleftarrow{h}_j^\top]^\top$$

- Decoder

- 입력 문장 전체를 고정 길이의 문맥 벡터로 인코딩하는 것이 아니라 각 시각마다 고유한 문맥 벡터를 가지고 hidden state를 계산

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

- context vector  $c_i$

- 입력 문장에 대해 인코더가 맵핑하는 annotation sequence  $h_j$ , 즉 인코더의 모든 은닉 상태를 종합해 만든다.

- 각 annotation  $h_i$ 는 전체 입력 시퀀스에서  $i$ 번째 단어에 대한 다른 단어의 focus 정보를 포함
- 이 annotation들에 대해 가중합을 계산해 문맥 벡터를 계산

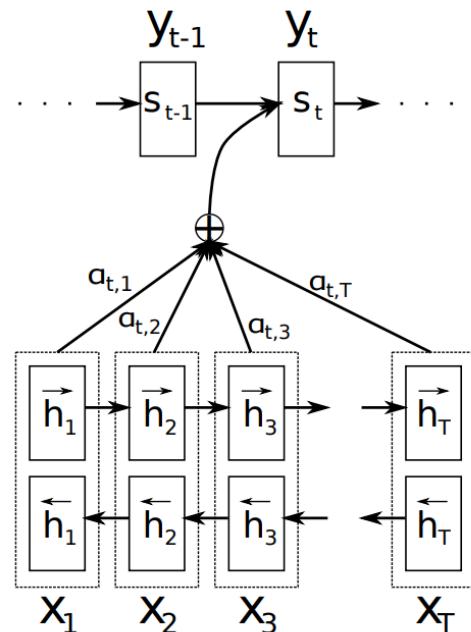
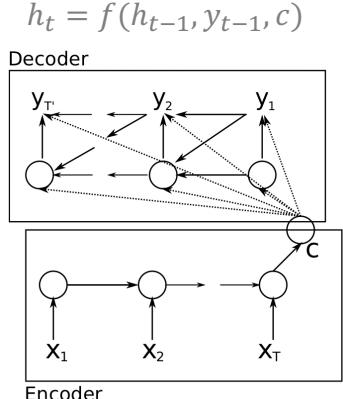
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \quad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})} \quad e_{ij} = a(s_{i-1}, h_j)$$

- Attention Score  $e_{ij}$ :

- 현재 디코더의 시점인  $i$ 에서 단어를 예측하기 위해, 인코더의 모든 은닉 상태 각각이 디코더의 현 시점(아직  $i$ 번째 단어 예측 전이기 때문에  $i-1$ )의 은닉 상태와 얼마나 유사한지를 판단

- 다음과 같은 조건부 확률 식을 이용해 target 단어를 만들어 낸다.

$$p(y_i | y_1, \dots, y_{i-1}, x) = g(y_{i-1}, s_i, c_i)$$



## 2. Attention

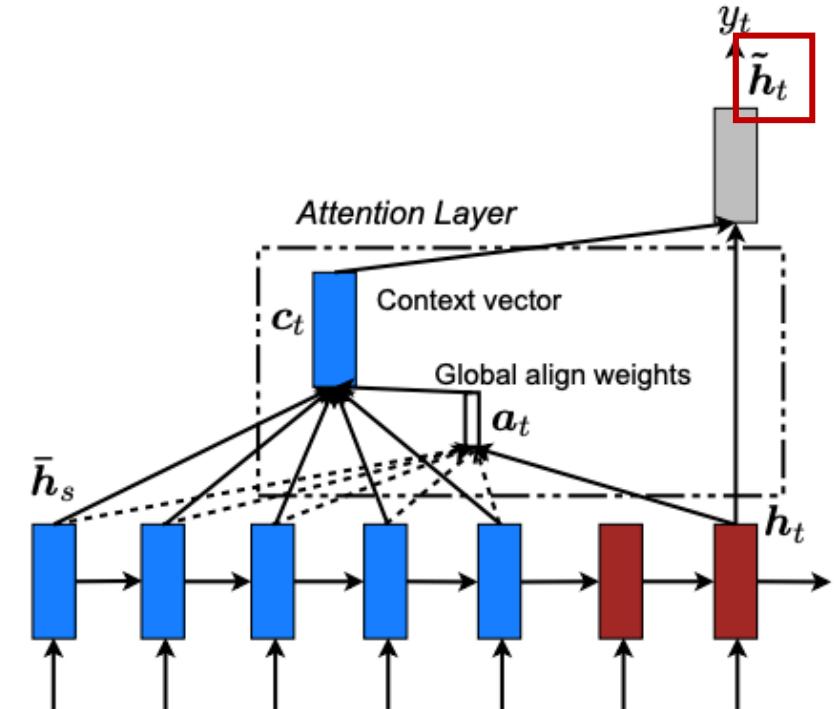
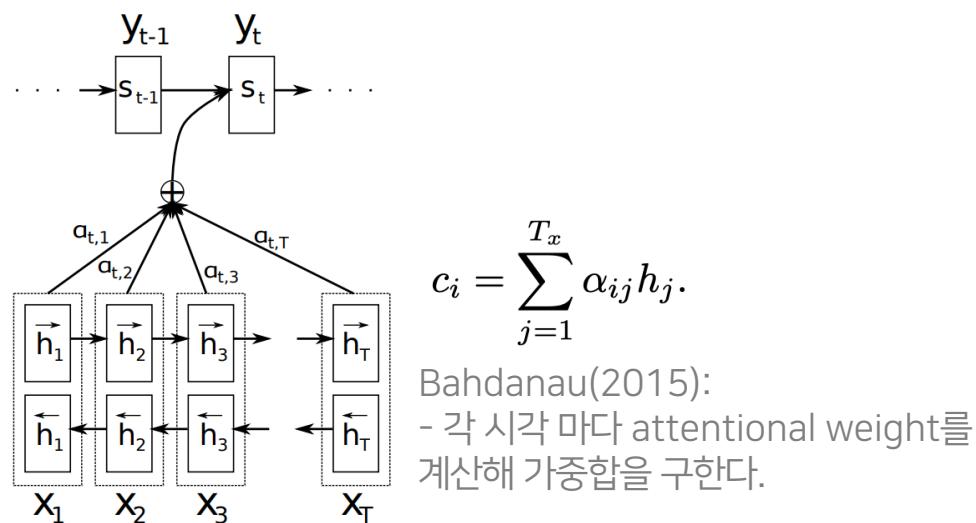
Luong et al. Effective Approaches to Attention-based Neural Machine Translation(2015)

### Luong Attention

#### 1) attention vector $\tilde{h}_t$ 도입(input feeding)

- 문맥 벡터와 디코더의 은닉 상태를 함께 고려해 attention 계산하기 위한 벡터
- 시각 t에서  $h_t$ 와  $c_t$ 를 concatenation 레이어 이용해 합친다.
- 이 벡터를 softmax 레이어로 전달해 타깃 단어를 예측

$$\tilde{h}_t = \tanh(\mathbf{W}_c[c_t; h_t]) \quad p(y_t|y_{<t}, x) = \text{softmax}(\mathbf{W}_s \tilde{h}_t)$$



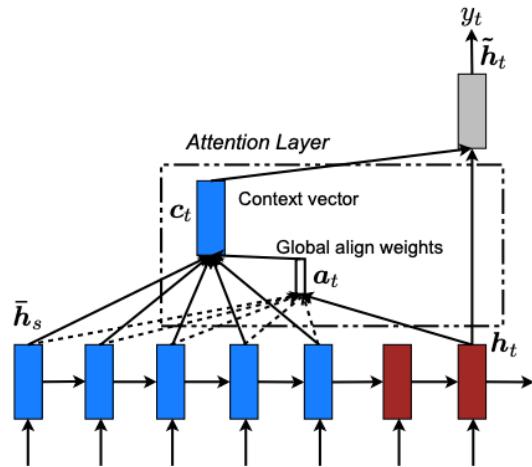
## 2. Attention

Luong et al. Effective Approaches to Attention-based Neural Machine Translation(2015)

### 2) Global Attention VS Local Attention

- Global: Source 문장의 모든 단어를 고려해 문맥 벡터를 만들어 냄
- Local: 각 시각마다 Source 문장의 일부(subset)을 골라 그 단어들만을 고려

#### Global Attention

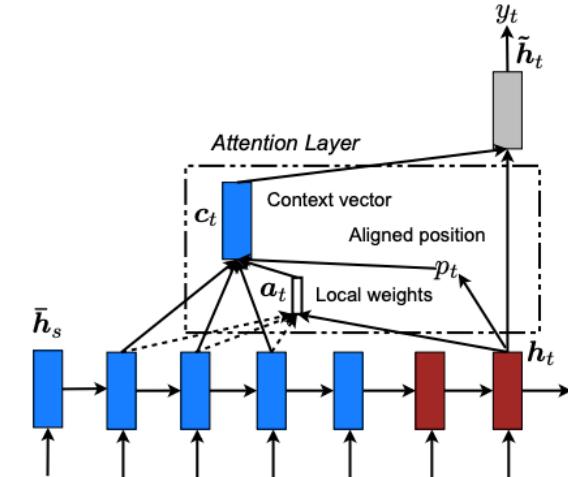


- 인코더의 모든 은닉 상태를 고려해 문맥 벡터 만듦!

$$\begin{aligned} \mathbf{a}_t(s) &= \text{align}(\mathbf{h}_t, \bar{\mathbf{h}}_s) \\ &= \frac{\exp(\text{score}(\mathbf{h}_t, \bar{\mathbf{h}}_s))}{\sum_{s'} \exp(\text{score}(\mathbf{h}_t, \bar{\mathbf{h}}_{s'}))} \end{aligned}$$

- Alignment vector  $a_t$ : 현재 은닉 상태  $\mathbf{h}_t$ 와 source의 은닉 상태(annotation)  $\bar{\mathbf{h}}_s$ 를 고려해 생성
- 모든 단어를 고려하기 때문에 계산량 증가 + 긴 문장을 다루는데 문제점

#### Local Attention



- 시각 t에 대해 alignment position  $p_t$ 를 계산
- $[p_t - D, p_t + D]$ 에 해당하는 source의 은닉 상태에 대해 가중평균을 구해 문맥벡터를 만듦
- $p_t$ 를 구하는 방법
  - Predictive alignment
    - $p_t$ 에 가까운 단어일수록 높은 attention 주기 위해  $p_t$  기준으로 정규분포 계산
  - $p_t = S \cdot \text{sigmoid}(\mathbf{v}_p^\top \tanh(\mathbf{W}_p \mathbf{h}_t))$
  - $\mathbf{a}_t(s) = \text{align}(\mathbf{h}_t, \bar{\mathbf{h}}_s) \exp\left(-\frac{(s - p_t)^2}{2\sigma^2}\right)$
- Monotonic alignment
  - Global attention의  $a_t$  구하는 식과 동일

$\mathbf{W}_p, \mathbf{v}_p$ : 모델 파라미터  
S: source 문장 길이  
 $\sigma = D/2$

## 2. Attention

Yang et al. Hierarchical Attention Networks for Document Classification(2016)

- Hierarchical Attention Networks

- 문서의 계층적 구조(단어 → 문장 → 문서)에서 착안한 계층적 attention 구조 가지는 모델
- 2중 Attention 구조를 이용해 단어 벡터에서 시작해 문서 레벨의 벡터를 만듦.
- 구조 (i: 문서내 문장의 순서)

- Word Encoder

- 각 문장을 이루고 있는 단어를 임베딩 행렬을 이용해 임베딩해준다.  $x_{it} = W_e w_{it}, t \in [1, T]$
- 이때 bidirectional GRU 이용 → 단어 양쪽의 정보를 모두 포함한 단어 annotation을 만듦!  $\vec{h}_{it} = \overrightarrow{\text{GRU}}(x_{it}), t \in [1, T]$   
 $\overleftarrow{h}_{it} = \overleftarrow{\text{GRU}}(x_{it}), t \in [T, 1]$ .

- Word Attention

- 어떤 문장의 의미를 결정하는데 모든 단어가 기여하는 것은 아님
  - Attention을 사용해 문장 의미를 결정하는데 중요한 역할을 하는 단어만을 추출
  - 그 representation을 종합해 문장 벡터를 만듦  $s_i = \sum_t \alpha_{it} h_{it}$

$$u_{it} = \tanh(W_w h_{it} + b_w)$$

$$\alpha_{it} = \frac{\exp(u_{it}^\top u_w)}{\sum_t \exp(u_{it}^\top u_w)}$$

- Sentence Encoder

- bidirectional GRU를 이용해 sentence vector를 인코딩  $\vec{h}_i = \overrightarrow{\text{GRU}}(s_i), i \in [1, L]$ ,
- $h_i$ : i번째 문장과 그 주변의 문장에 대한 요약 포함  $\overleftarrow{h}_i = \overleftarrow{\text{GRU}}(s_i), i \in [L, 1]$ .

- Sentence Attention: 문서 내에서 중요한 문장을 알아내기 위해 Attention 사용

- 문장 level 문맥 벡터를 이용해 문장 중요도를 계산  $u_i = \tanh(W_s h_i + b_s)$ .
- 그렇게 계산된  $v$ 는 document vector, 즉 문서 내 문장들의 정보를 담고 있는 벡터

$$\alpha_i = \frac{\exp(u_i^\top u_s)}{\sum_i \exp(u_i^\top u_s)},$$

$$v = \sum_i \alpha_i h_i,$$

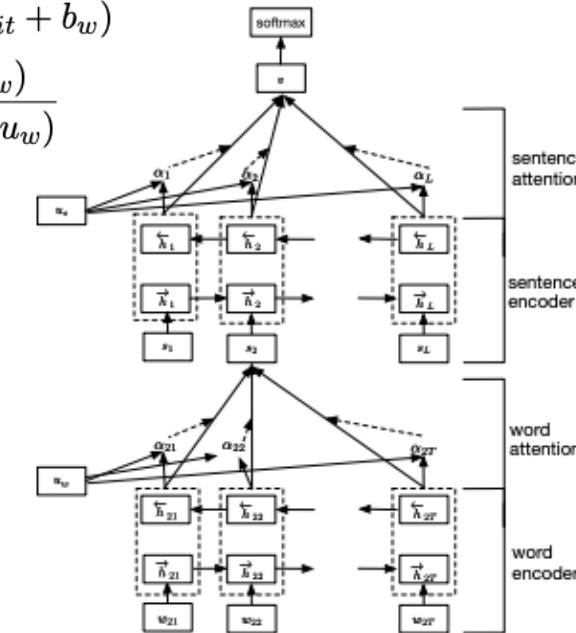


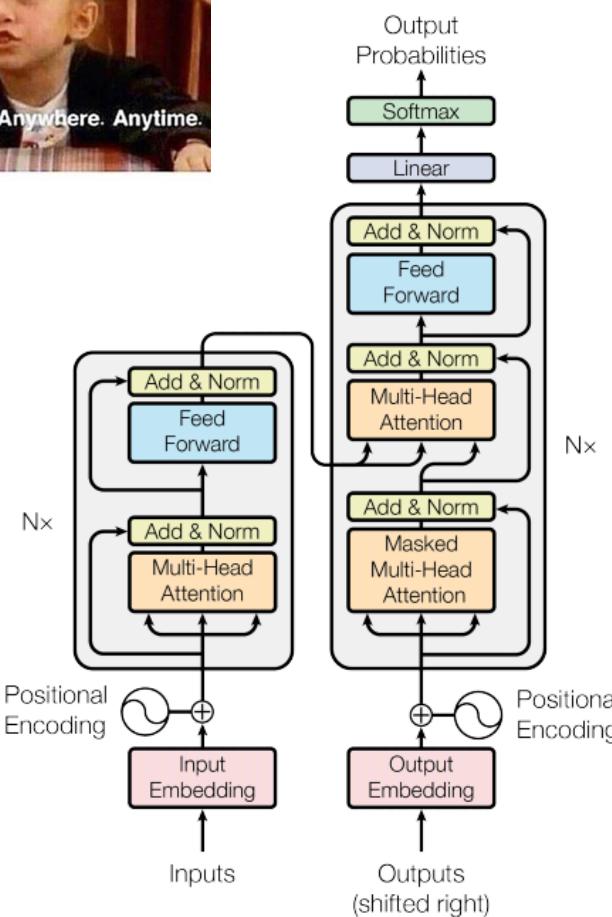
Figure 2: Hierarchical Attention Network.

When he asks you how much  
attention do you need???



### 3. Transformer

Vaswani et al. Attention Is All You Need(2017)

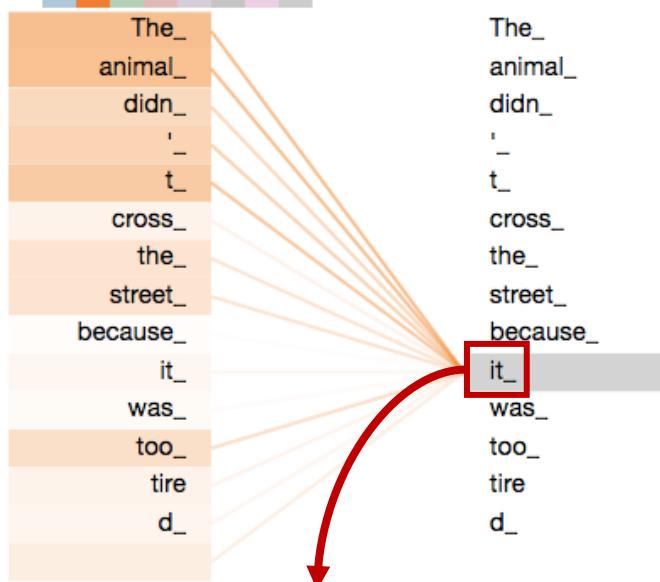


- **RNN:**

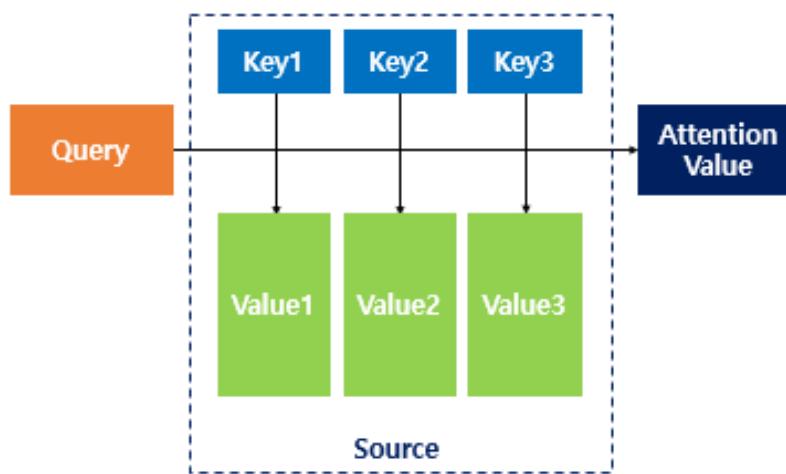
- input과 output 시퀀스를 구성하는 단어의 position을 따라 계산을 수행
- 각 시각마다 단어를 나열(align)해 직전 시간의 은닉 상태  $h_{t-1}$ 와 현재 시각의 입력 단어를 이용해 현재 시각의 은닉 상태인  $h_t$ 를 알아냄.
- 시간에 따라 이전 시각의 계산한 결과를 이용해 순서대로 계산하는 Recurrent 모델은 **병렬처리를 할 수 없음.**
- 시퀀스의 길이에 따른 성능 저하 문제를 해결하기 위해 Attention이 고안되었지만 대부분의 **Attention 또한 Recurrent 모델을 사용.**

- **Transformer**

- RNN을 없애고 self-attention을 이용해 sequence - sequence 계산을 할 수 있도록 만든 구조
- **병렬처리가 가능!**

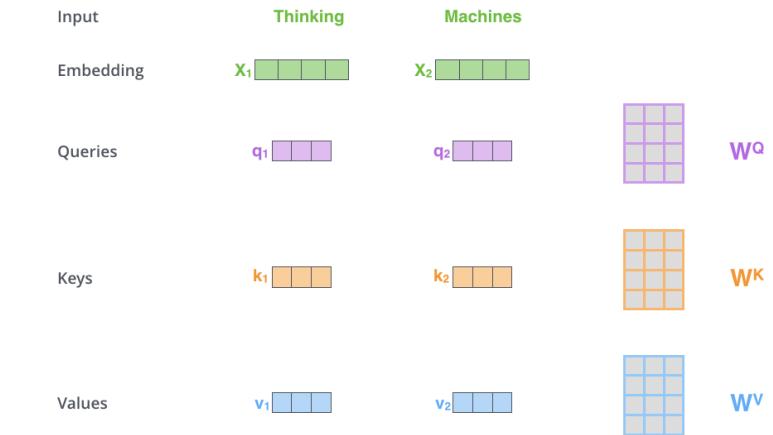
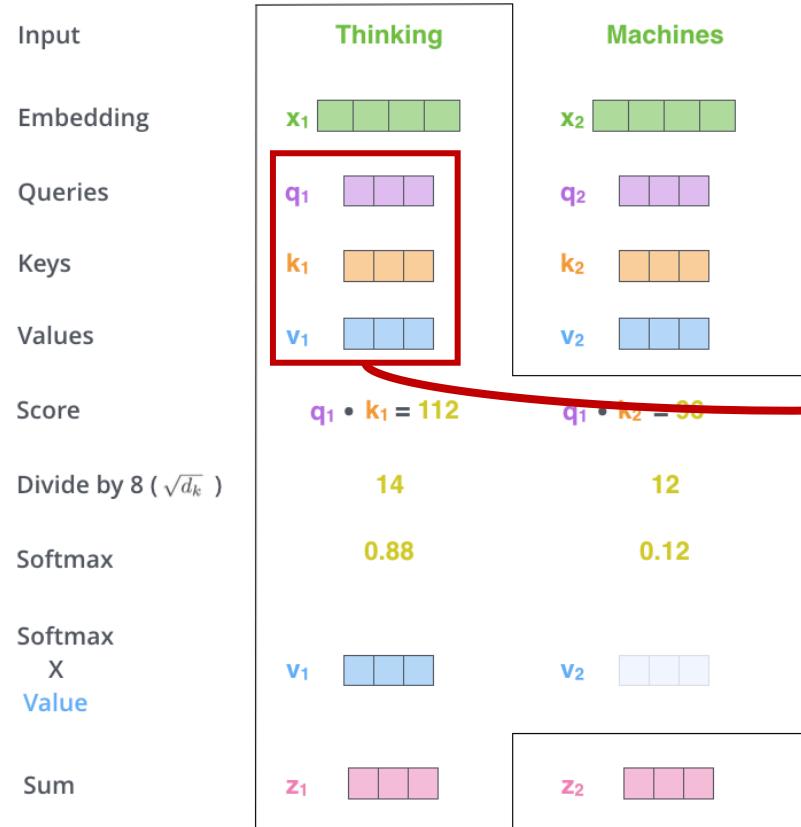


- 문장 안에서 'it'이 가리키는 단어는 무엇인가?
- self-attention을 이용해 문장 내 단어 사이의 의미 관계를 파악한다!

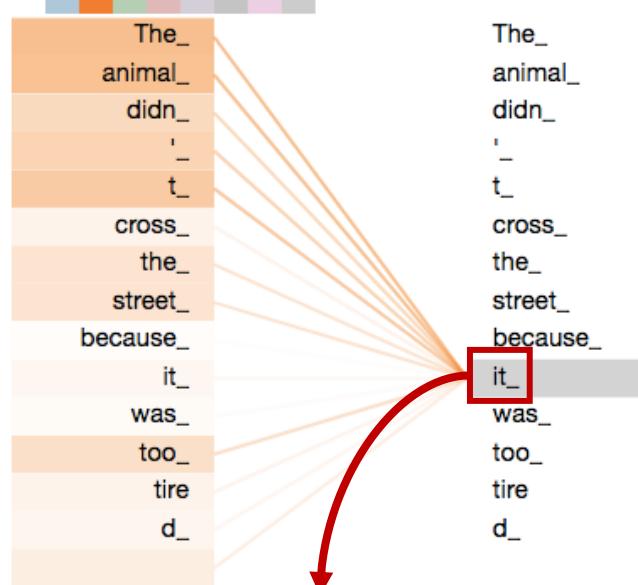


# 3. Transformer

- Self Attention



- 입력 임베딩에 queries, keys, values weight를 곱해 queries, keys, values를 구한다!
  - weight는 학습을 통해 구할 수 있음.



- 문장 안에서 'it'이 가리키는 단어는 무엇인가?
- self-attention을 이용해 문장 내 단어 사이의 의미 관계를 파악한다!

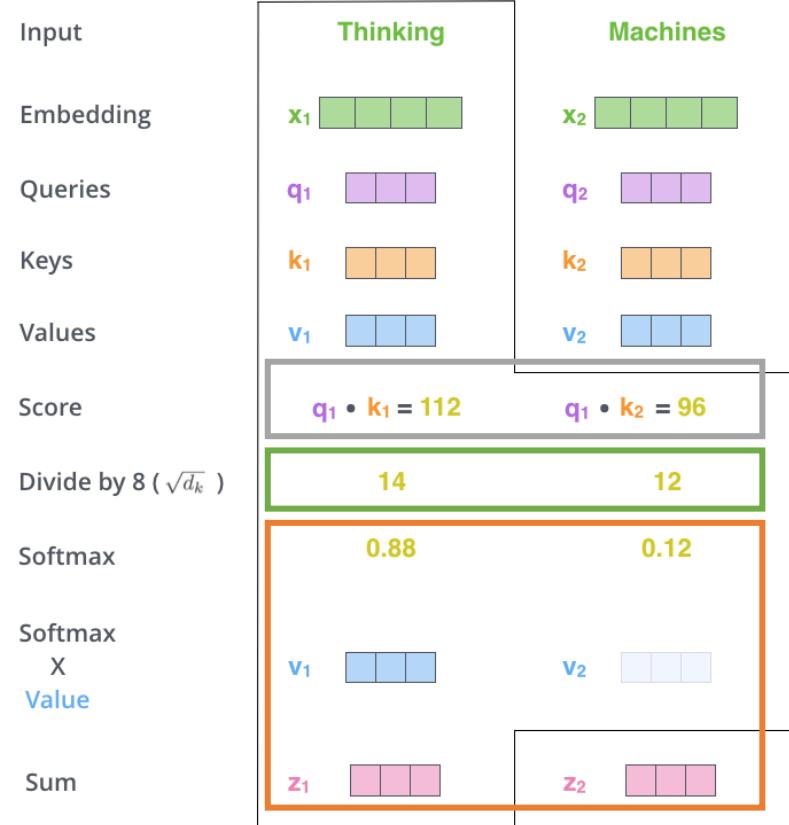
$$\begin{array}{c}
 X \quad \quad \quad W^Q \quad \quad \quad Q \\
 \left[ \begin{array}{ccccc} \text{green} & \text{green} & \text{green} & \text{green} & \text{green} \end{array} \right] \times \left[ \begin{array}{ccccc} \text{purple} & \text{purple} & \text{purple} & \text{purple} & \text{purple} \end{array} \right] = \left[ \begin{array}{ccccc} \text{purple} & \text{purple} & \text{purple} & \text{purple} & \text{purple} \end{array} \right] \\
 X \quad \quad \quad W^K \quad \quad \quad K \\
 \left[ \begin{array}{ccccc} \text{green} & \text{green} & \text{green} & \text{green} & \text{green} \end{array} \right] \times \left[ \begin{array}{ccccc} \text{orange} & \text{orange} & \text{orange} & \text{orange} & \text{orange} \end{array} \right] = \left[ \begin{array}{ccccc} \text{orange} & \text{orange} & \text{orange} & \text{orange} & \text{orange} \end{array} \right] \\
 X \quad \quad \quad W^V \quad \quad \quad V \\
 \left[ \begin{array}{ccccc} \text{green} & \text{green} & \text{green} & \text{green} & \text{green} \end{array} \right] \times \left[ \begin{array}{ccccc} \text{blue} & \text{blue} & \text{blue} & \text{blue} & \text{blue} \end{array} \right] = \left[ \begin{array}{ccccc} \text{blue} & \text{blue} & \text{blue} & \text{blue} & \text{blue} \end{array} \right]
 \end{array}$$

}

모든 과정은 단어 벡터로 따로 수행하는 것이 아니라 query / key / value를 행렬로 묶어서 일괄적으로 계산

# 3. Transformer

- Self Attention



- score 계산

- 어떤 단어에 대해 그 단어와 문장 내 모든 단어들에 대해 점수를 계산
- 현재 단어의 쿼리 벡터와 점수를 매기려는 단어의 키 벡터를 내적해 점수를 얻는다.

- Scaling

- 점수를  $\sqrt{d_k}$ 로 나눔 ( $d_k$ : key 벡터 크기)
- $d_k$ 가 커질수록 내적을 계산하는 규모가 커지기 때문에 softmax 함수의 gradient가 매우 작아지게 됨.
- $\sqrt{d_k}$ 로 나눠줌으로써 일정한 gradient를 가질 수 있도록 만들어준다!

- Softmax

- 현재 위치의 단어 인코딩이 각 단어의 표현을 얼마나 가질지를 결정한다.
- 물론 현재 위치의 단어에 대한 점수가 가장 높지만, 현재 단어와 관련되어 다른 단어에 대한 정보를 얼마나 가지고 갈지를 계산
- Softmax로 구한 값을 각 단어별 value 값에 곱해 전부 더한다.

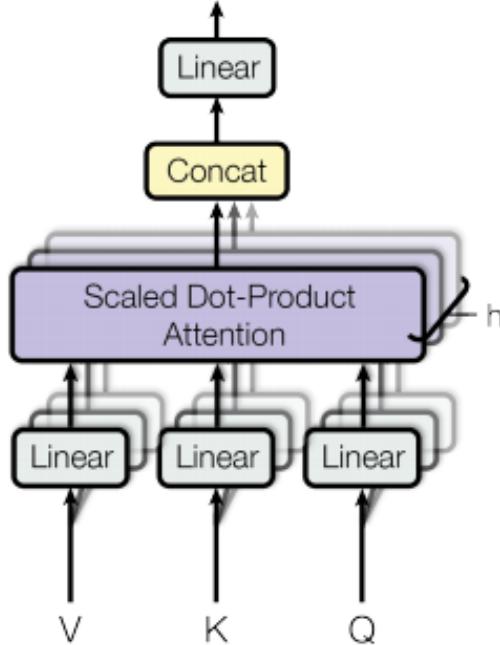
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

### 3. Transformer

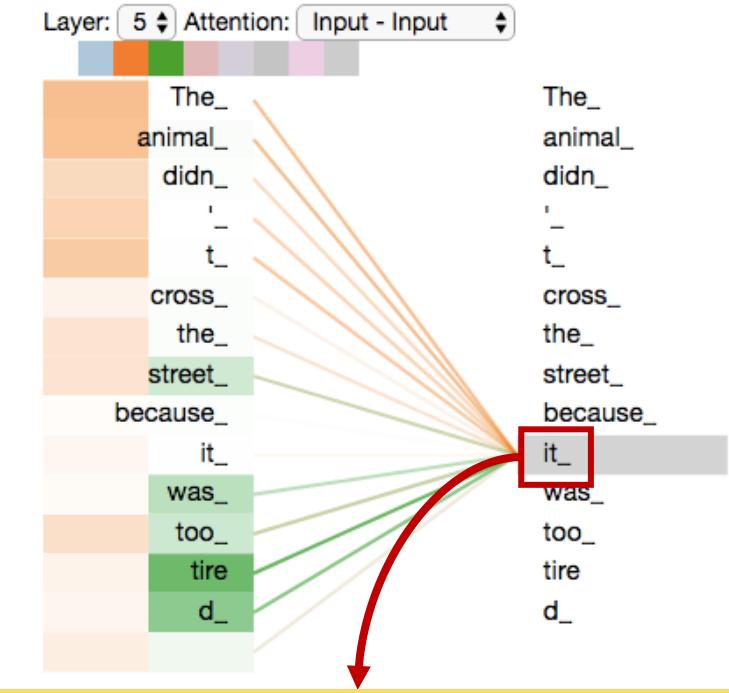
Vaswani et al. Attention Is All You Need(2017)

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where  $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$



- **Multi-Head**
- $d_{model}$ 의 keys / values / queries로 attention을 계산하는 것이 아니라  $h$ 차원으로 나눠서 계산
  - 하나의 attention보다 여러 번의 attention을 병렬적으로 수행시 성능이 더 좋다.
- 각 stage마다 다른 representation를 만들어냄
  - 각 subspace마다 다른 위치에 attend 된 표현을 얻을 수 있다.



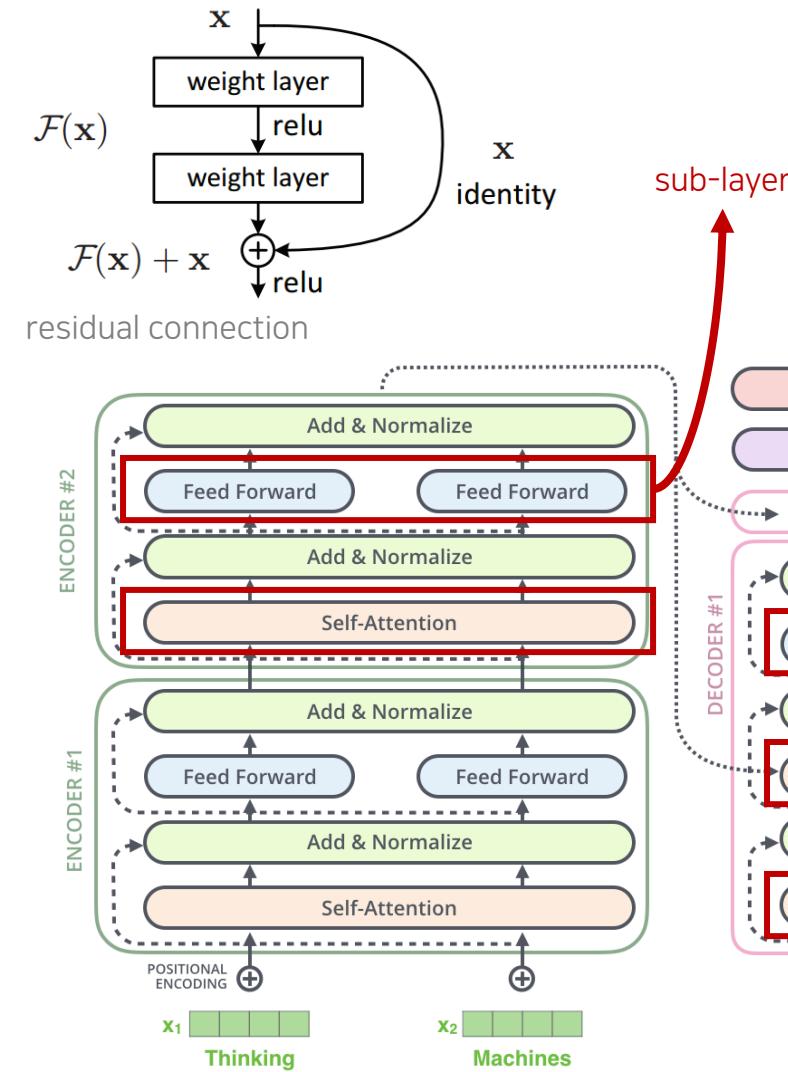
첫번째 head: 'it'과 'animal'의 연관도를 높게 계산

두번째 head: 'it'과 'tired'의 연관도를 높게 계산

→ 각 Attention Head가 전부 다른 시각에서 보고  
있기 때문이다!

# 3. Transformer

Vaswani et al. Attention Is All You Need(2017)



## 인코더

- 6개의 레이어로 구성, 각 레이어는 두개의 sub-layer로 구성
  - multi-head self-attention
  - position-wise fully connected feed forward network
- sub-layer는 residual connection과 layer normalization으로 연결
- 각 sub-layer의 output: LayerNorm( $x + \text{Sublayer}(x)$ )

## 디코더

- 6개의 레이어로 구성, 각 레이어는 3개의 sub-layer로 구성
  - multi-head attention
  - position-wise fully connected feed forward network
  - masked multi-head attention
- 인코더와 마찬가지로 sub-layer들은 residual connection과 layer normalization으로 연결

# 3. Transformer

Vaswani et al. Attention Is All You Need(2017)

## Feed Forward

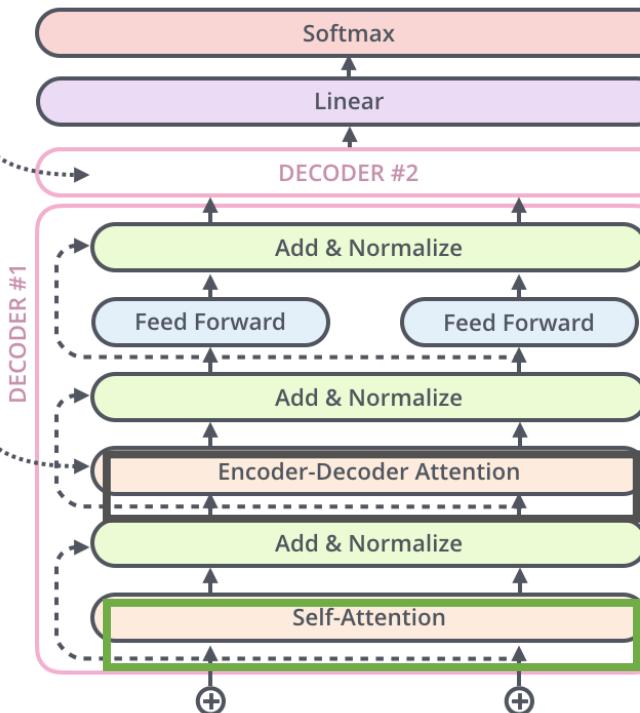
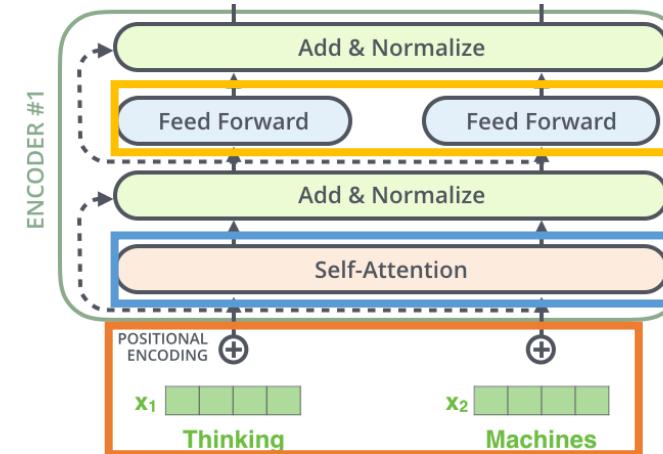
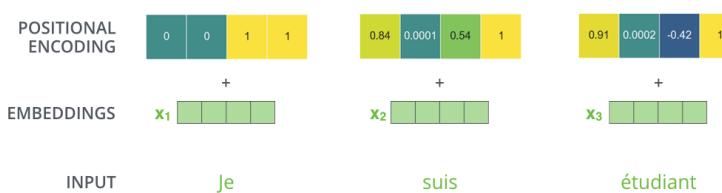
- 각 위치의 단어마다 독립적으로 적용되어 결과 출력

## self-attention layers

- keys, values, queries는 이전 시각의 인코더의 output을 가지고 계산(이전 시각의 output = 현재 시각의 input)

## 임베딩

- Word Embedding: input 문장을 임베딩
- Positional Embedding: 각 단어에 위치 정보를 알려주기 위해 임베딩 벡터에 positional embedding을 더해 인코더 입력으로 사용
  - RNN: 시간 순으로 단어가 순차적으로 입력
  - Transformer: 단어를 순차적으로 입력 받지 않음
    - 따라서 단어의 위치 정보를 직접 알려주어야 한다.



## encoder-decoder attention layers

- query: 이전 시각의 디코더 레이어로부터
- key, value: 인코더의 결과값
- 이런 구조는 디코더에서 입력 시퀀스의 모든 부분에 attend할 수 있게 해준다.
  - 기존의 seq2seq의 attention 매커니즘 구조를 따온 것!
- self-attention X

## masked self-attention

- 기존의 인코더-디코더 모델에서처럼 이전 시각의 정보를 현재 시각의 예측에 사용하도록 만들기 위해 masking 사용
- 현재 시각 이전(t-1)에 생성된 output만 참조하기 때문에 현재 시각 이후의 key들에 대해 mask!

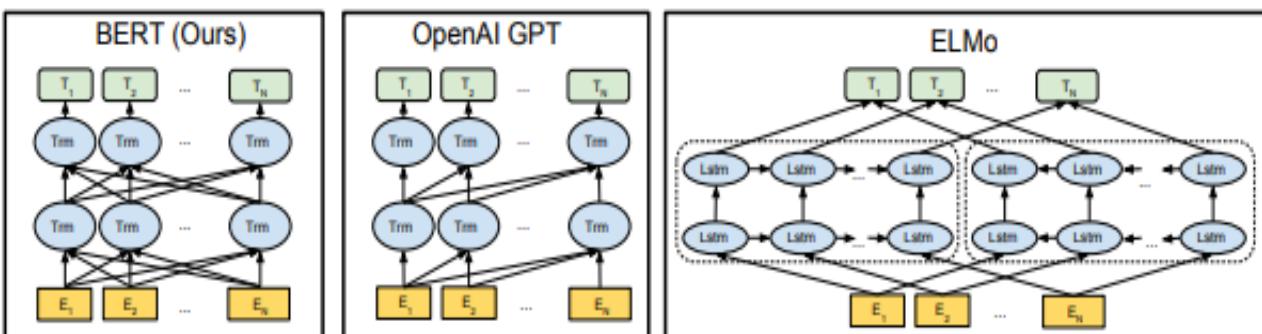
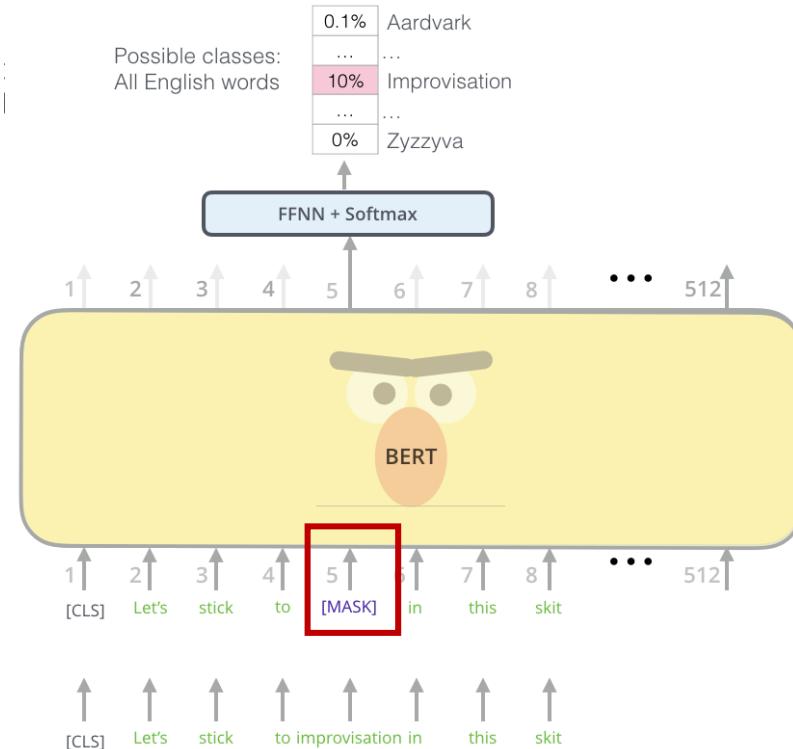


# 3. Transformer

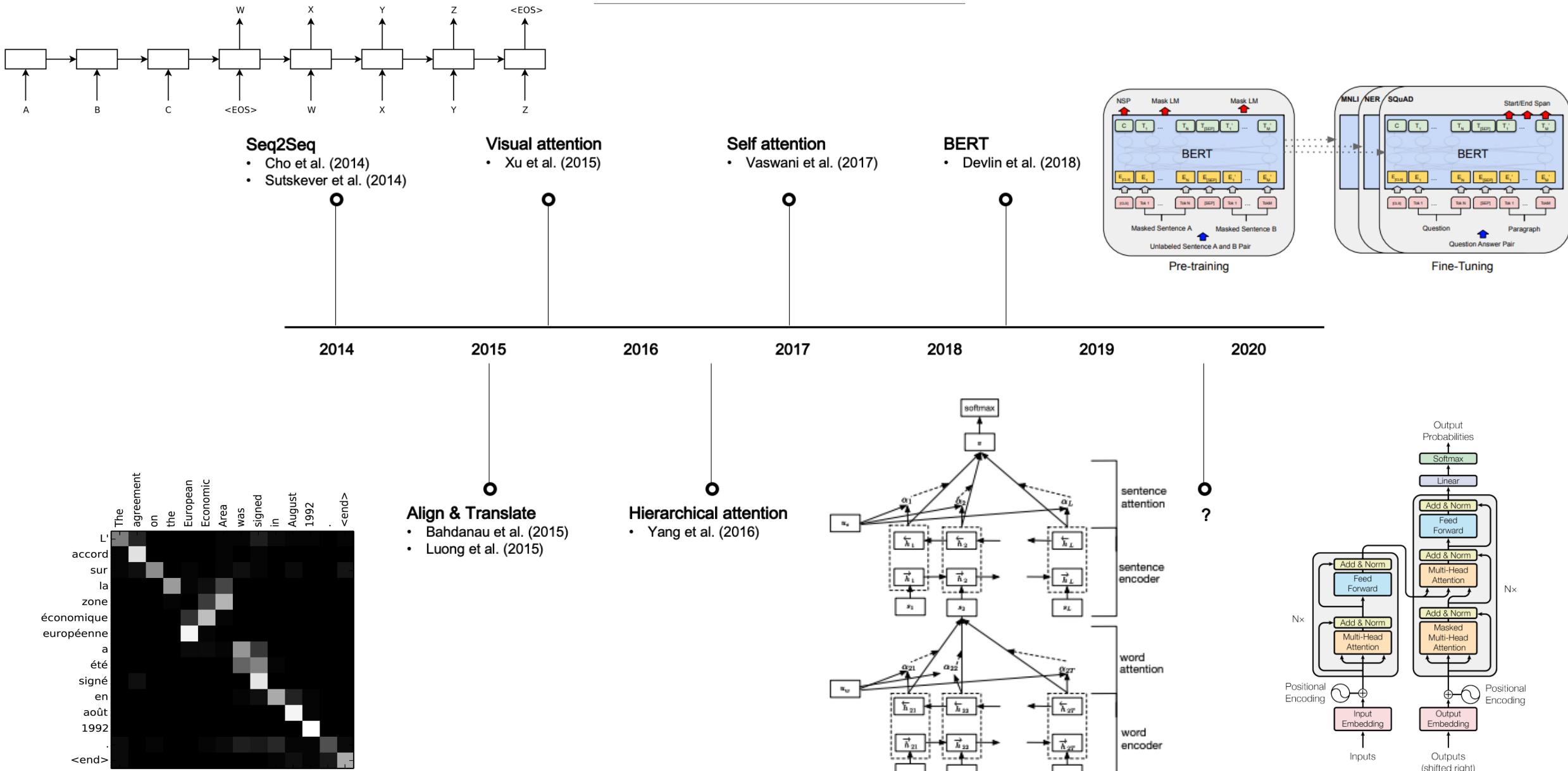
## BERT(Bidirectional Encoder Representations from Transformers)

- Transformer의 인코더를 사용
  - Transformer의 인코더는 문장의 모든 단어를 이용해 계산하는 self-attention을 사용
  - 따라서 다음 단어를 예측해야하는 언어 모델 task에서 이를 사용하기엔 부적절
  - 랜덤으로 15%의 단어를 [MASK]를 통해 마스킹 해 모델이 그 단어를 맞추도록 학습
- Why Transformer encoder?
  - 대부분의 기존의 언어모델은 단방향(left-to-right / right-to-left) 구조
    - 과거의 정보에만 attend할 수 있기 때문에 pretrained 된 정보를 온전히 이용하지 못함
    - 양방향 문맥이 필요한 task의 경우 약점으로 작용
  - ELMO: 양방향 LSTM을 이용해 단순히 앞뒤 은닉 상태를 concatenation하기 때문에 shallow한 정보만을 가지게 됨
  - Transformer를 이용해 한번에(bi-directional) input 정보를 인코딩

Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding(2019)



# 4. Conclusion



## 5. Reference

---

1. Cho, K., van Merriënboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. *Learning phrase representations using RNN encoder-decoder for statistical machine translation*(2014)
2. Sutskever, I., Vinyals, O., and Le, Q. *Sequence to sequence learning with neural networks*(2014)
3. Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*(2015)
4. Minh-Thang Luong, Hieu Pham, Christopher D. Manning. *Effective Approaches to Attention-based Neural Machine Translation*(2015)
5. Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, Yoshua Bengio. *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*(2015)
6. Zichao Yang , Diyi Yang , Chris Dyer , Xiaodong He , Alex Smola , Eduard Hovy. *Hierarchical Attention Networks for Document Classification*(2016)
7. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin. *Attention Is All You Need*(2017)
8. Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*(2018)