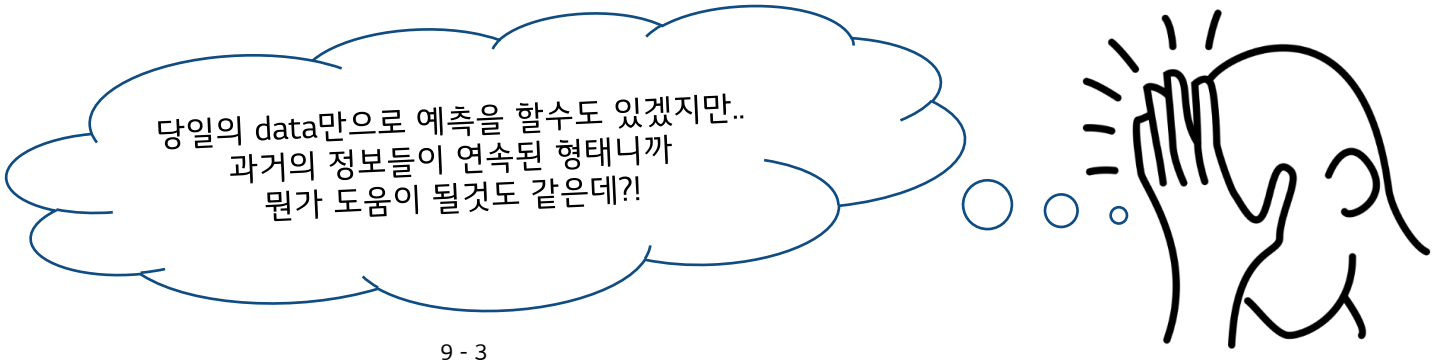
M9. Recurrent Neural Network

RNN

- Recurrent unit
- Gradient vanishing/exploding
- LSTM/GRU
- Attention
- 부록 : Transformer

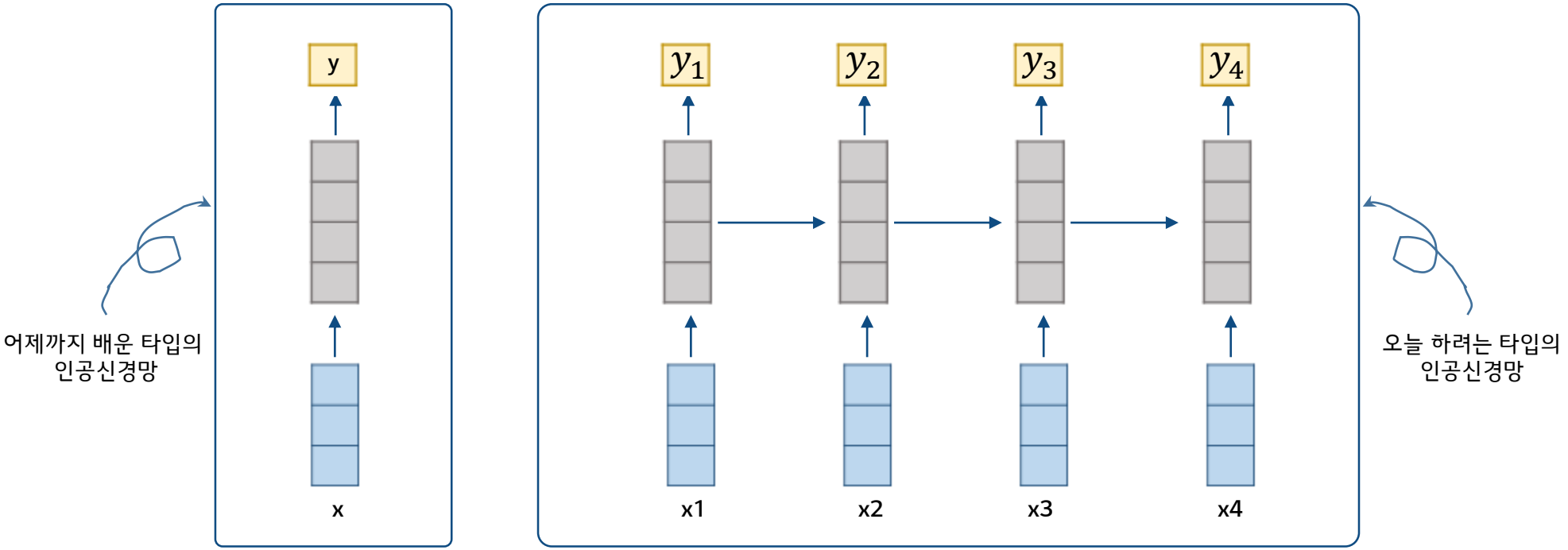
EXAMPLE : 어제 주가, 어제 KOSPI, 뉴스 언급량을 사용해 오늘의 주가를 예측해보자

Date	03-01	03-02	03-03	03-04	t
어제 주가	1000	1010	1200	1115	x_t
어제 KOSPI	2211	2220	2219	2200	
뉴스 언급량	90	100	98	80	
오늘 주가	1010	1200	1115	?	y_t



EXAMPLE : 어제 주가, 어제 KOSPI, 뉴스 언급량을 사용해 오늘의 주가를 예측해보자

Date	03-01	03-02	03-03	03-04	
					t
어제 주가	1000	1010	1200	1115	x_t
어제 KOSPI	2211	2220	2219	2200	
뉴스 언급량	90	100	98	80	
오늘 주가	1010	1200	1115	?	y_t

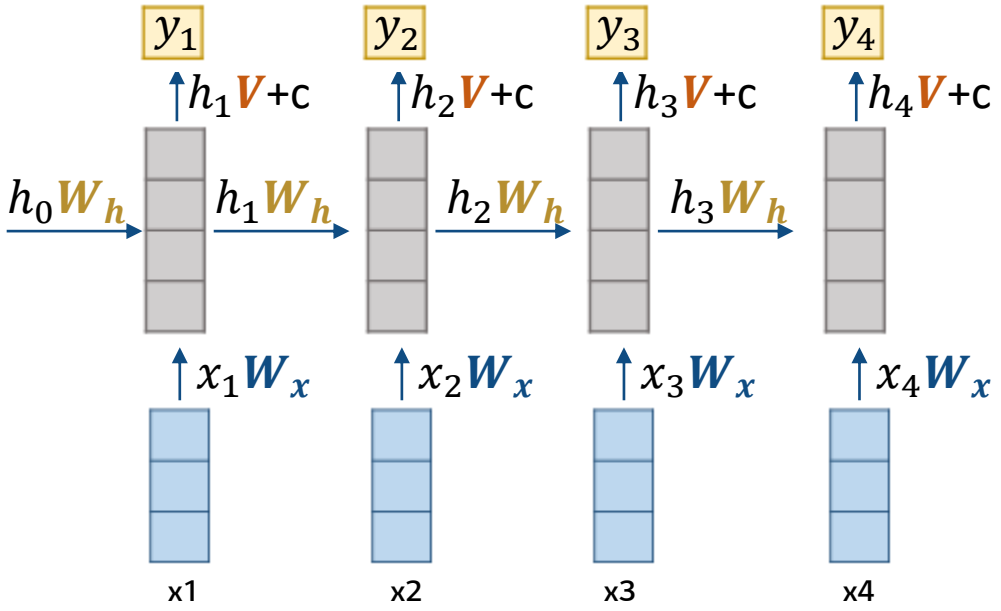


EXAMPLE : 어제 주가, 어제 KOSPI, 뉴스 언급량은 사용해 오늘의 주가를 예측해보자

Date	03-01	03-02	03-03	03-04	
					t
어제 주가	1000	1010	1200	1115	x_t
어제 KOSPI	2211	2220	2219	2200	
뉴스 언급량	90	100	98	80	
오늘 주가	1010	1200	1115	?	y_t

RNN

- 매 시점 데이터를 처리할 때마다 동일한 파라미터(W_x, W_h, V, b , c) 공유
- 임의 길이의 sequential 입력 처리 가능



현 시점 t의 정보(hidden)는

현 시점 t의 input과

이전 시점 t-1의 정보(hidden)도 구성

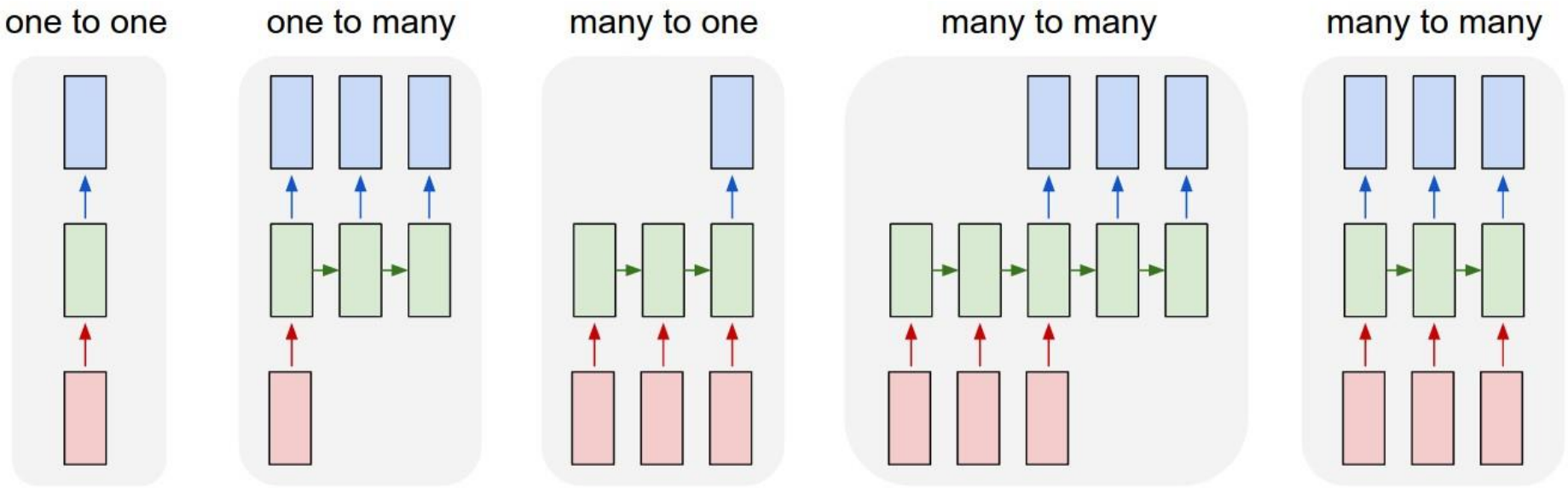
$$h_t = \tanh(x_t W_x + h_{t-1} W_h + b)$$

(3x4) (4x4)

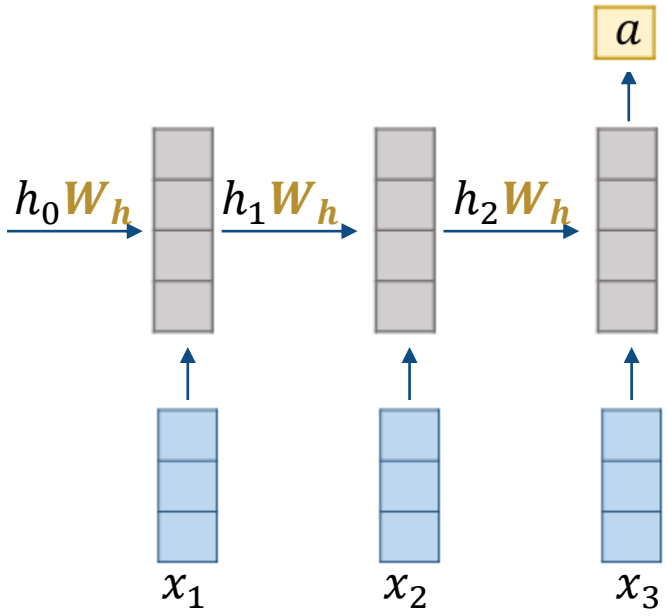
$$y_t = h_t^{(4x1)} V + c$$

Recurrent Neural Network

RNN은 입력과 출력의 길이가 유연하기 때문에 다양한 모델을 설계할 수 있다



Backpropagation Through Time(BPTT) : RNN의 파라미터 업데이트하기



$$C = d(y, a)$$

$$a_t = h_t \boldsymbol{V} + c$$

$$h_t = \tanh(x_t \boldsymbol{W_x} + h_{t-1} \boldsymbol{W_h} + b)$$

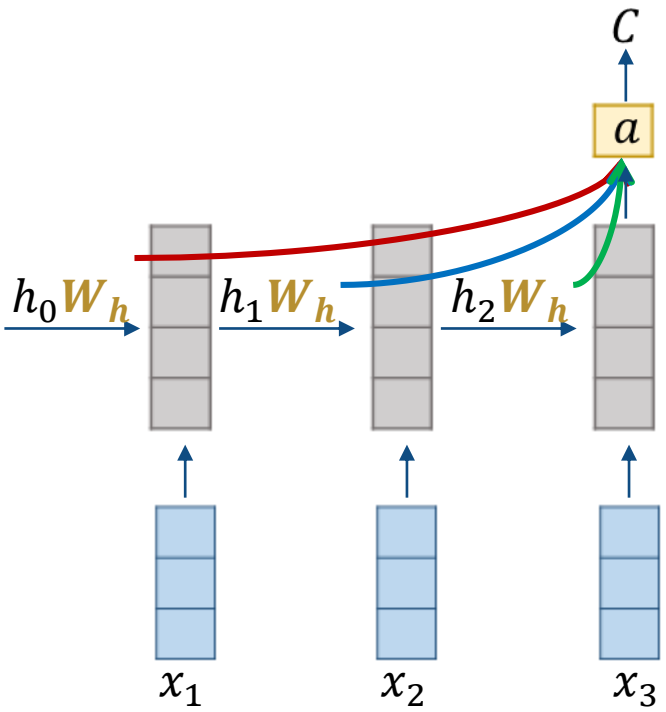
$\boldsymbol{W_h}$를 업데이트 해보자!

1일차 경사하강법 … 기억나시나요?!

REMIND :

$$w_{j+1} \leftarrow w_j - \alpha \frac{\partial C(w)}{\partial w_j}$$

Backpropagation Through Time(BPTT) : RNN의 파라미터 업데이트하기



$$C = d(y, a)$$

$$a_t = h_t \boldsymbol{V} + c$$

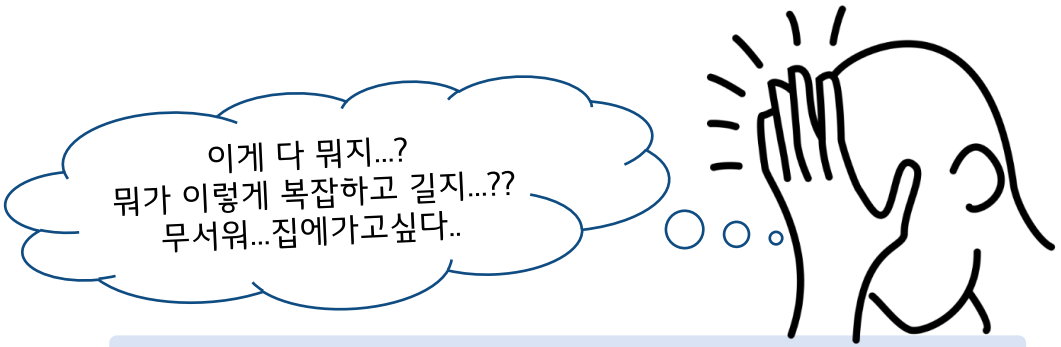
$$h_t = \tanh(\underbrace{x_t \boldsymbol{W_x} + h_{t-1} \boldsymbol{W_h} + b}_{q_t})$$

W_h를 업데이트하기 위해, gradient를 구해보자!

$$\frac{\partial C}{\partial W_h} = \frac{\partial C}{\partial h_3}\frac{\partial h_3}{\partial q_3}\frac{\partial q_3}{\partial W_h} +$$

$$\frac{\partial C}{\partial h_3}\frac{\partial h_3}{\partial q_3}\frac{\partial q_3}{\partial h_2}\frac{\partial h_2}{\partial q_2}\frac{\partial q_2}{\partial W_h} +$$

$$\frac{\partial C}{\partial h_3}\frac{\partial h_3}{\partial q_3}\frac{\partial q_3}{\partial h_2}\frac{\partial h_2}{\partial q_2}\frac{\partial q_2}{\partial h_1}\frac{\partial h_1}{\partial q_1}\frac{\partial q_1}{\partial W_h}$$



RNN에서는 timestep마다 weight를 공유,

영향을 받은 모든 시간에서의 Loss를 더해주어야 한다

Gradient 정보가 점점 사라지거나 증폭되는 Gradient vanishing / exploding 현상

$$\frac{\partial C}{\partial W_h} = \frac{\partial C}{\partial h_3}\frac{\partial h_3}{\partial q_3}\frac{\partial q_3}{\partial W_h} + \frac{\partial C}{\partial h_3}\frac{\partial h_3}{\partial q_3}\frac{\partial q_3}{\partial h_2}\frac{\partial h_2}{\partial q_2}\frac{\partial q_2}{\partial W_h} + \frac{\partial C}{\partial h_3}\frac{\partial h_3}{\partial q_3}\frac{\partial q_3}{\partial h_2}\frac{\partial h_2}{\partial q_2}\frac{\partial q_2}{\partial h_1}\frac{\partial h_1}{\partial q_1}\frac{\partial q_1}{\partial W_h}$$

여러 timestep을 고려하는 과정에서
Gradient가 증폭될 수 있음
(Gradient Exploding)

과거의 정보일수록 tanh미분값을 여러 번 곱하게 됨

→ 과거의 정보를 반영하는 부분의 gradient가 거의 0에 가까워짐 (Gradient Vanishing)

→ 현재와 먼 과거의 정보일수록 정보 의존성 감소

→ 즉, 현시점에서의 결정은 최근 내용위주로만 잘 반영, 오래된 시점의 내용은 거의 다 망각

Gradient vanishing / exploding 문제의 해결

Gradient Vanishing

LSTM, GRU 등 변형된 Recurrent unit 활용

- 먼 과거의 정보도 잘 잊어버리지 않도록 변형
- 각 timestep의 입력들에 다른 가중치를 적용, 중요한 인풋의 가중치를 높임

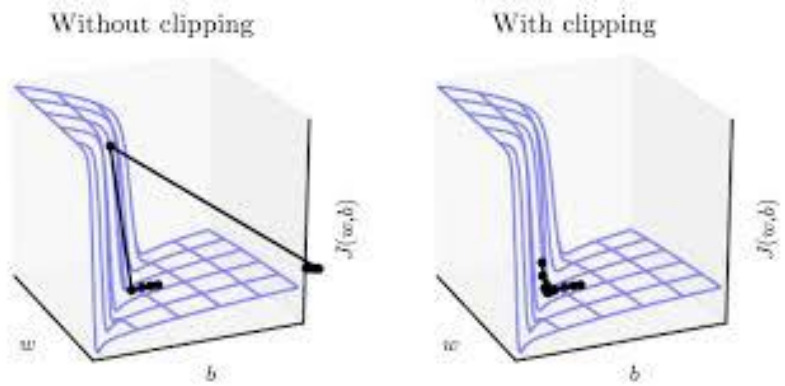
Attention 메커니즘 활용

Gradient Exploding

Gradient Clipping 활용

- Gradient가 너무 커질 경우, 지정된 상한선을 넘지 않도록 유지

  → 파라미터를 너무 큰 폭으로 update하지 않도록 방지



RNN 구조의 장단점

RNN 장점 (+)

- 가변 길이의 입력 처리 가능
- 전 timestep에 걸쳐 파라미터 공유 → shared representations

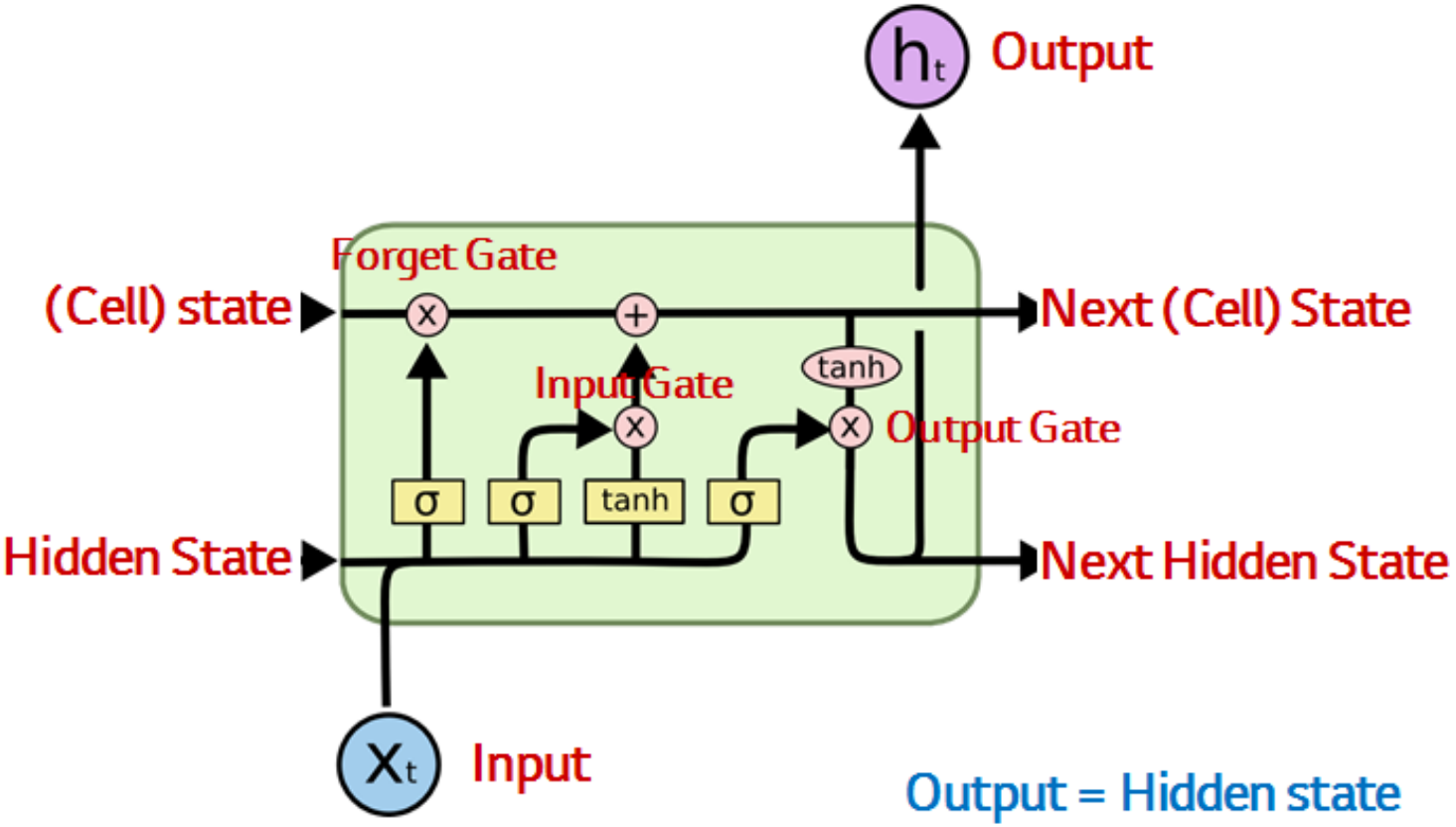
  → 긴 입력 값이 들어와도 모델 사이즈가 증가하지 않음
- (이론적으로) 많은 이전 단계의 정보를 현재의 timestep에 적용 가능

RNN 단점 (-)

- 이전 timestep이 모두 계산되어야 현재 timestep 계산가능 → 다소 "느림"
- Gradient vanishing / exploding 현상
- (실질적으로) 많은 이전 timestep의 정보를 활용할 수 있는 모델이 아님

  → long-term dependency

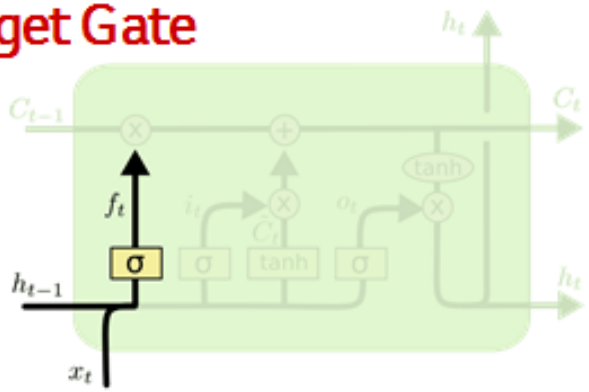
LSTM (Long Short Term Memories) unit



무섭고 복잡하게 생겼습니다만... 중요한 것은 Gate라는 것만 기억합시다!

LSTM (Long Short Term Memories)

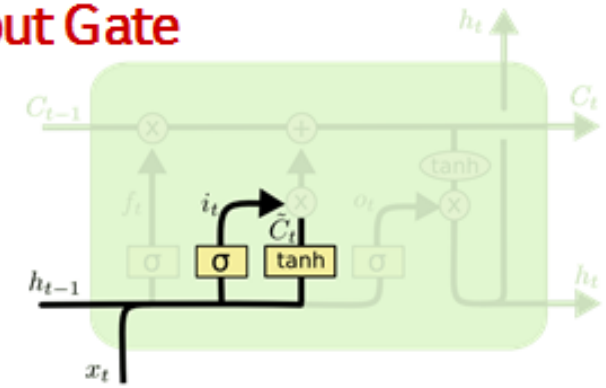
Forget Gate



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

어떤 정보를 버릴지 결정

Input Gate



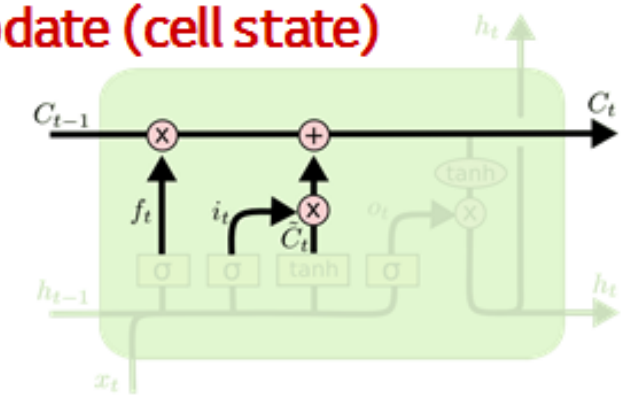
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

새로 들어온 정보 중 어떤 정보를 얼마나 반영할 지 결정

LSTM (Long Short Term Memories)

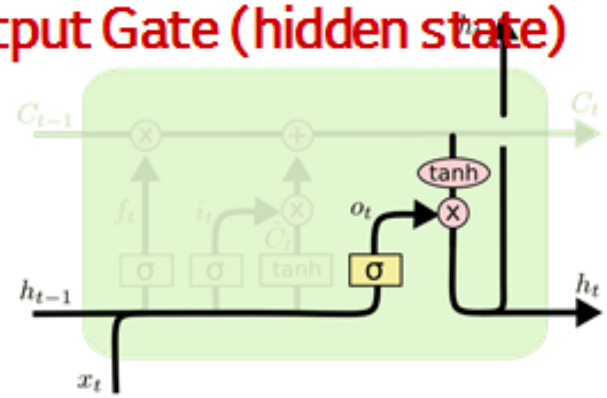
Update (cell state)



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

앞서 결정된 대로, 기존 정보와 새 정보의 반영 정도에 따라 업데이트

Output Gate (hidden state)

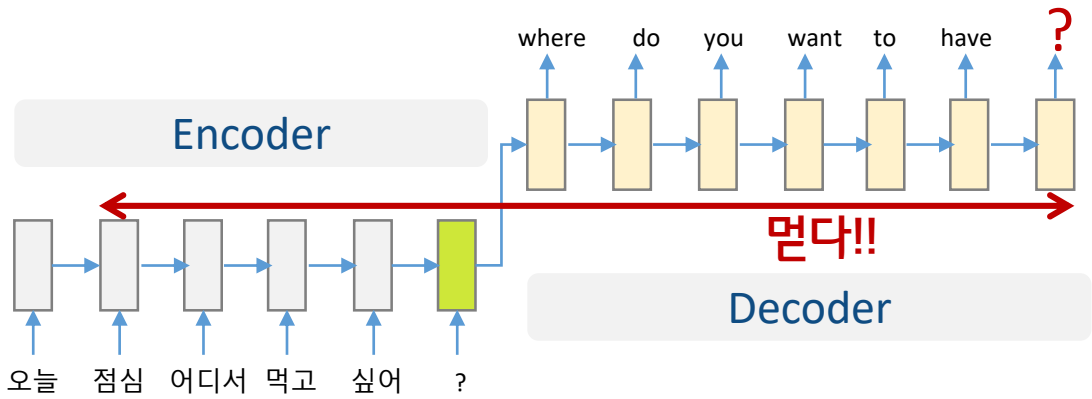


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

업데이트된 정보를 얼마나 반영하여 output으로 내보낼지 결정

Attention : 중요한 부분의 정보에 더 집중한 representation 만들기

- RNN을 번역 과제에 활용해보자 (feat. many-to-many)



- 인풋 문장을 읽은 RNN 인코더의 마지막 hidden은 문장의 전반적인 문맥 정보를 압축하여 담고 있음
- 그런데, 이 hidden은 먼 과거의 덜 큰 정보일수록 정보를 굉장히 압축해놓음 (long term dependency 문제)
- 따라서 위의 그림과 같이 마지막 hidden에만 의존해 전체 문장을 번역하려다 보면 원본에 어떤 내용이 있었는지 잊어버릴 수 있음

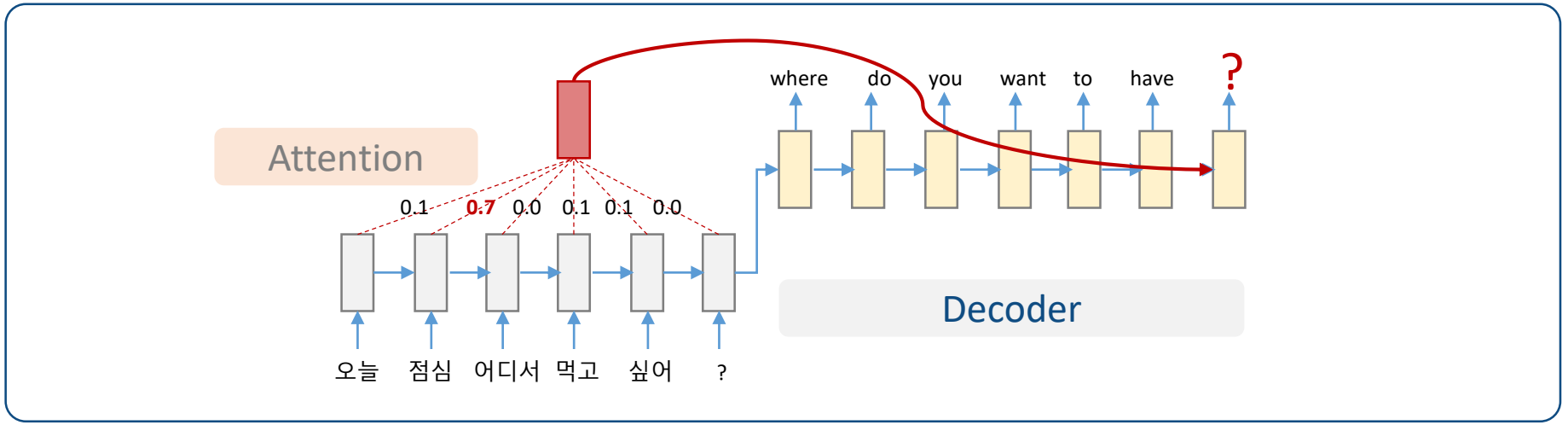
Motive

각 단어를 번역(decoding)할 때
원본 문장에서 중요했던 단어와 문맥을 참고하면 나아지지 않을까?

Attention : 중요한 부분의 정보에 더 집중한 representation 만들기

- Attention은 이용해 문맥 정보 알려주기

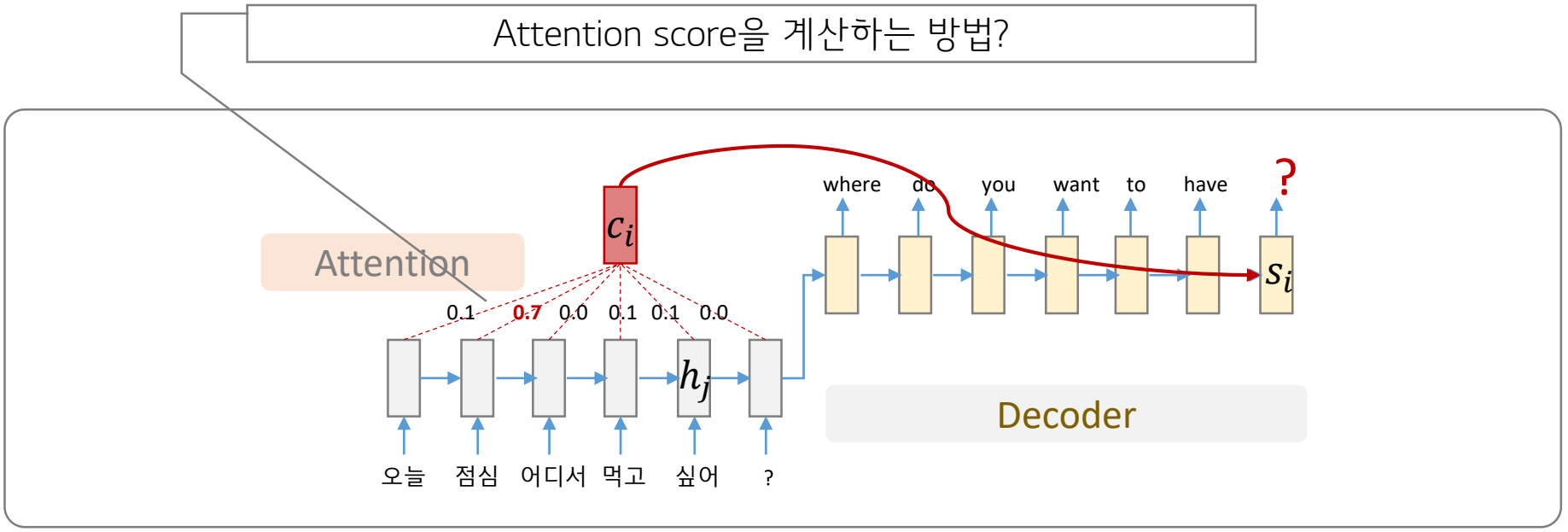
  → 각 단어를 번역할 때 원본에서 중요하게 봐야 할 단어와 문맥을 참고하게 만들자



- 각 step에서 디코딩할 때 중요한 단어에 대해 집중한 feature를 생성해 RNN hidden에 추가해줌.
- 예를 들어 위에서 ? 에 들어갈 단어를 번역 할 때는 원본 문장 중 <점심>에 집중하는 것이 좋으니 이에 해당하는 가중치 0.7도 가장 높게 계산된 벡터를 생성, 번역에 사용하면 <lunch>라는 올바른 단어를 꺼낼 수 있을 것
- 가중치에 해당하는 **attention score**은 직전에 사용한 RNN 디코더 hidden과 input hidden과의 관련도 등으로 계산하기 때문에, 각 time step에서 다른 값을 가짐.

  → 즉, 사람의 개입 없이 모델이 스스로 집중해서 봐야 할 포인트를 찾는다!

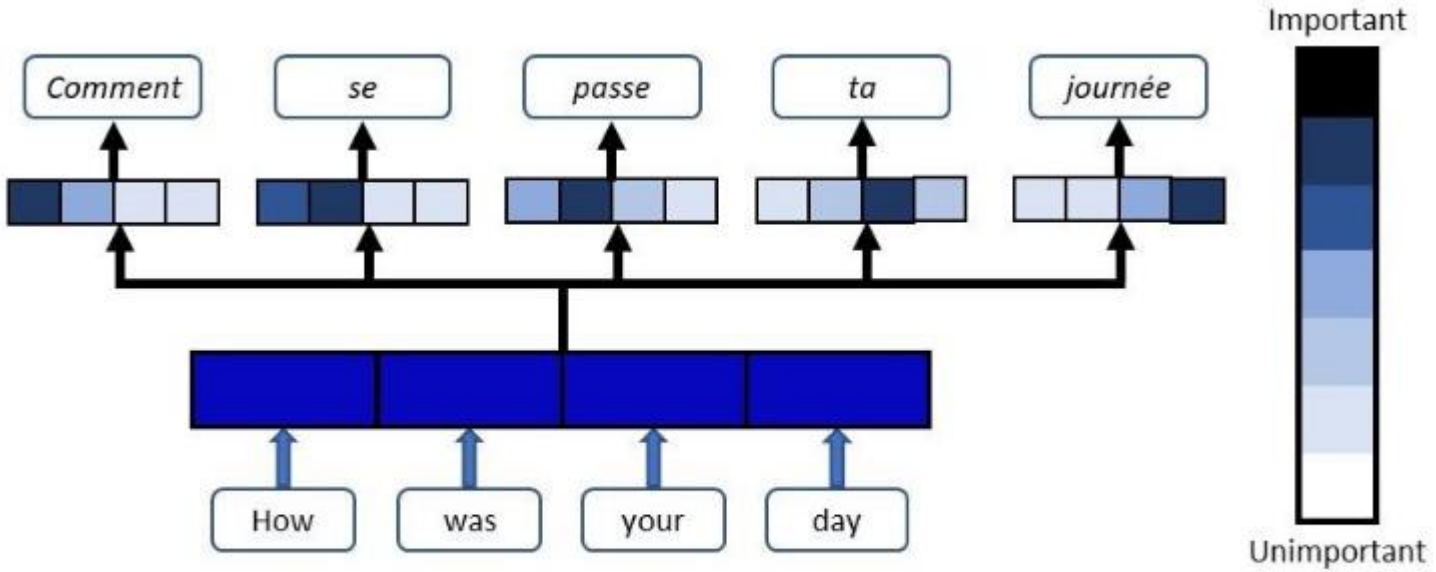
Attention : 중요한 부분의 정보에 더 집중한 representation 만들기



- 디코딩하는 i번째 타임스텝 직전의 hidden은 s_{i-1},
- Attention 대상 도큰 중 j번째에 대한 hidden은 h_j 라고 할 때, i번째 타임스텝의 hidden은 다음과 같이 구한다.

잠깐!! 수식이 복잡해 보인다고 포기하지 마시라!
attention score를 구하는 방법은 이 외에도 매우 다양하니 복잡한 수식은 외우려 할 필요 없다!
중요한 점은 한번의 추론마다 직전의 time 뿐 아니라 과거의 모든 인코딩들을 고려한다는 점!

Attention : 중요한 부분의 정보에 더 집중한 representation 만들기

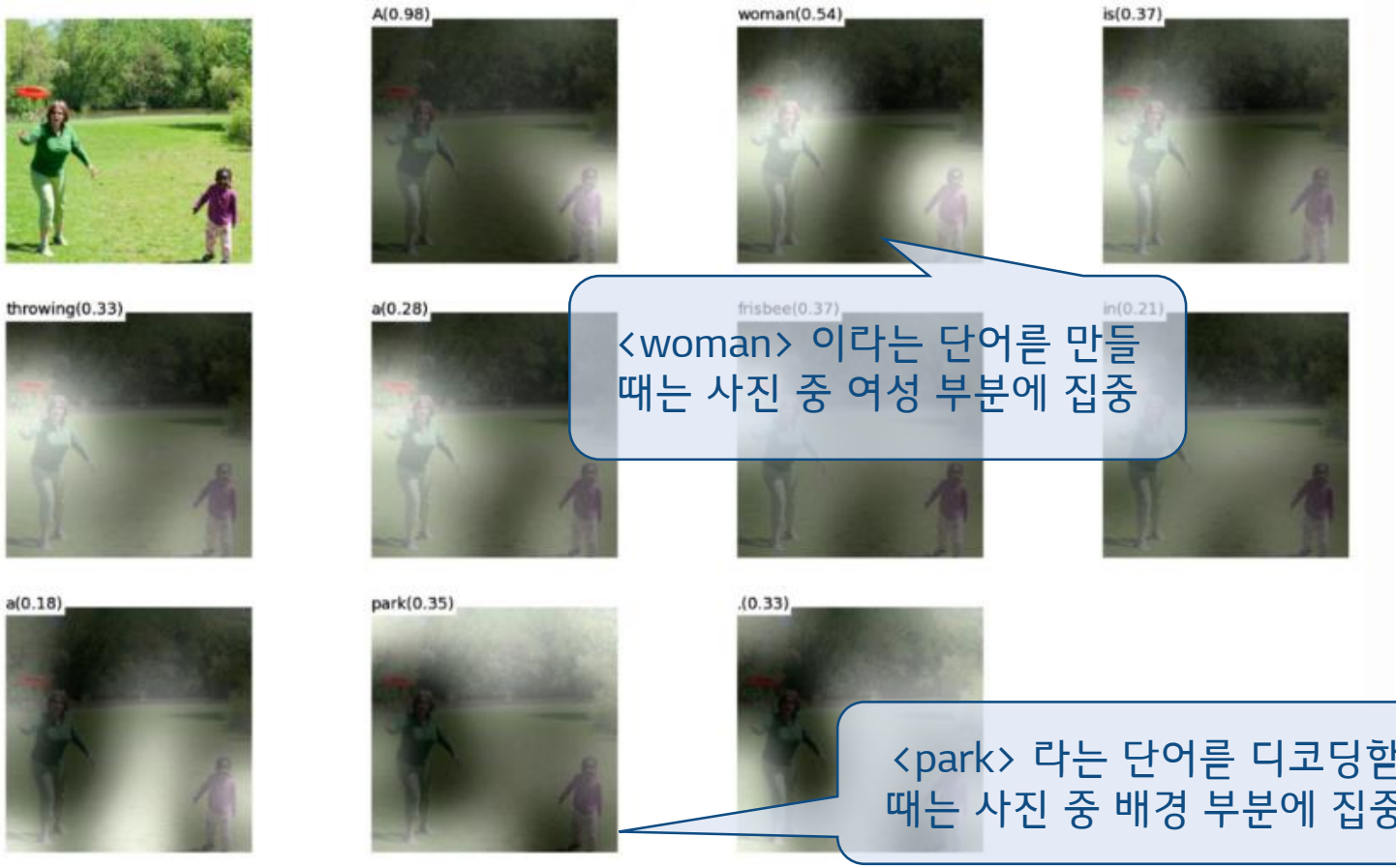


Weights are assigned to input words at each step of the translation

Attention

- 번역 과제에서 고안된 개념이긴 하나, input의 hidden들 중 현재 timestep에서 중요한 부분에 가중치를 줘서 representation을 만들겠다는 attention의 개념은 이미지처리 등에서도 사용할 수 있음

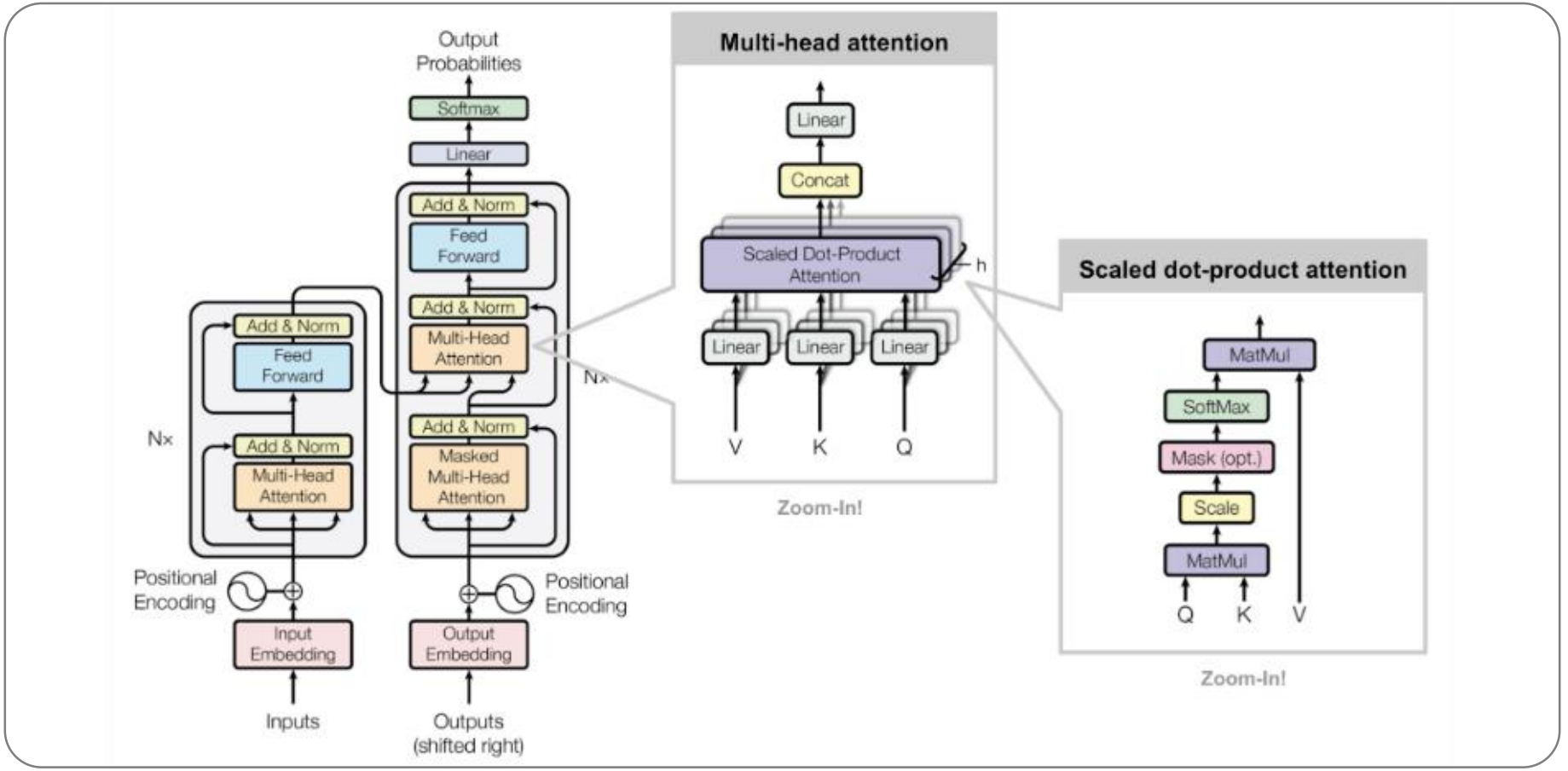
<Show and Tell>



부록. Transformer

문장 시권스를 인코딩하는 새로운 접근법

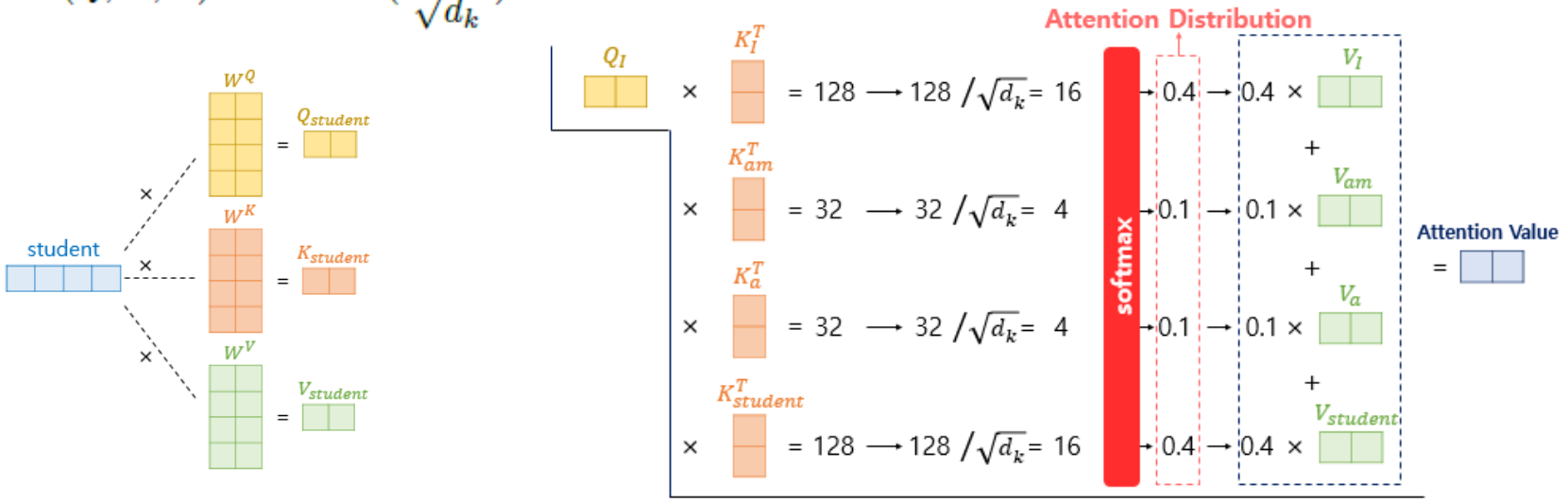
- Transformer 구조를 제안한 “Attention is All you Need”는 2017년에 발표된 가장 흥미로운 논문 중 하나!
- Transformer에서는 **Self attention**을 사용해 Recurrent Unit 없이도 문장을 모델링할 수 있다.
- 핵심은 multi-head self-attention에서 **사용하는 scaled dot-product attention**



Scaled dot-product attention

- Self attention은 인풋 시권스 전체에 대해 attention을 계산해 각 도큰의 representation을 만들어가는 과정으도, 업데이트된 representation은 문맥 정보를 가지고 있다.
- 예를 들어 “아이유는 1993년에 태어났다. 그녀는 최근에 드라마 호텔 델루나에 출연했다” 라는 인풋에 대해 self-attention을 적용하면 “그녀”에 해당하는 representation은 “아이유”에 대한 정보를 담게 된다.

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



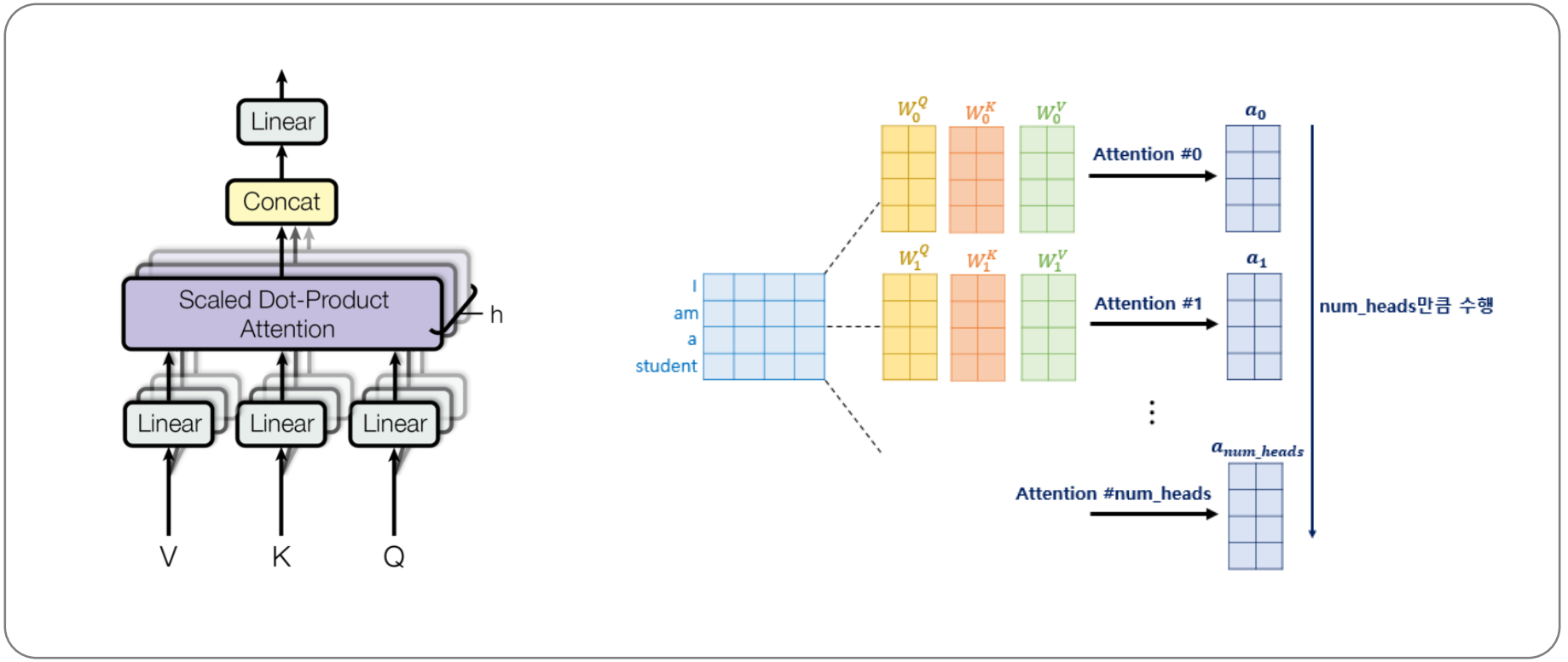
Attention의 대상이 되는 도큰들은 key와 value, attention 하는 도큰은 query도 변환 (행렬곱)

Attention 가중치는 query에 대해 각 key들의 가중치는 scale된 벡터 내적에 softmax를 취해서 구함.

가중치를 이용해 value를 가중합하여 query의 representation을 업데이트

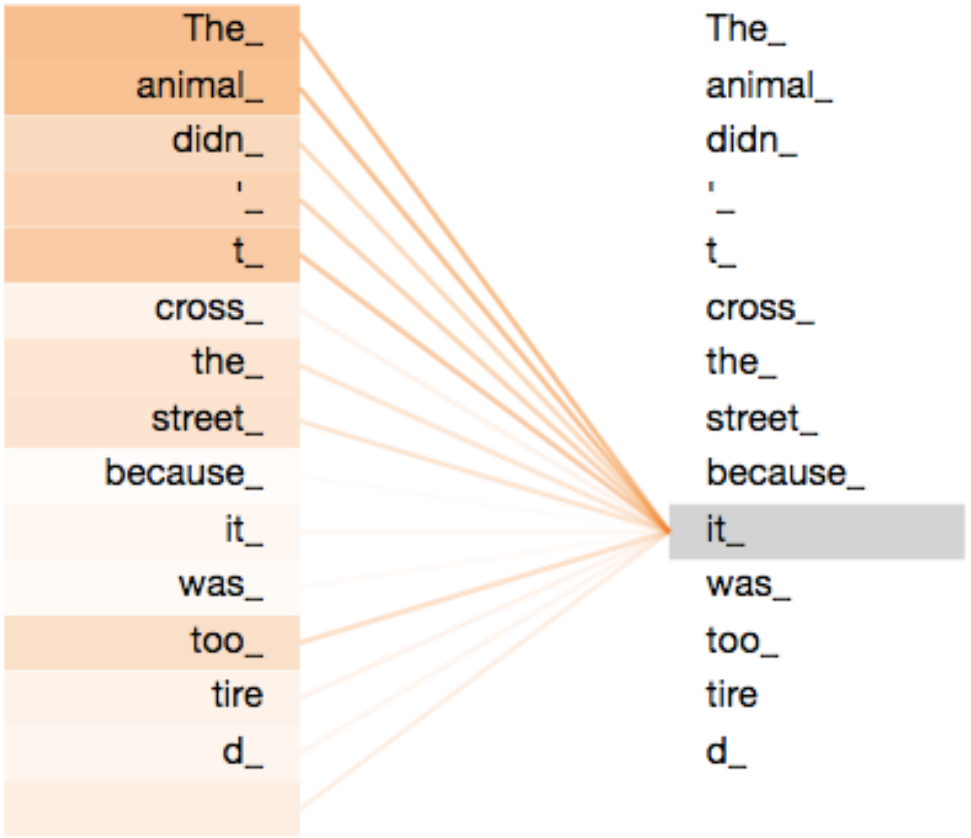
Multi head attention

- Scaled dot-product attention은 한 번에 계산하는 것이 아니라 여러 개의 head를 이용해 계산함.
- 즉, 같은 attention 계산 과정을 여러 번 반복하여 그 결과를 concat하여 최종 attention score을 계산
- 이는 CNN filter을 여러 장 사용함으로써 이미지에 있는 다양한 특성을 포착하는 것처럼, 토큰 사이의 다양한 관계를 포착하기 위함임.



Transformer Self-attention example

- “The animal didn’t cross the street because it was too tired”
- 라는 문장에 Transformer 구조를 이용해 self attention 적용



“it” 이라는 대명사에 대해 self-attention은 계산한 결과 it이 지칭하는 ‘the animal’ 도큰에 대한 score이 높게 나타남.

Transformer 활용 언어모델 : BERT

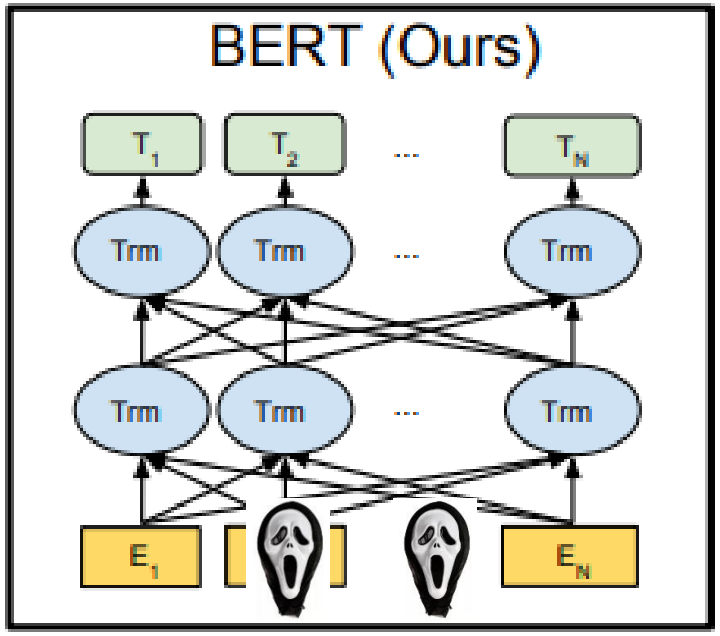
Bidirectional Encoder Representations from Transformers (a.k.a BERT)

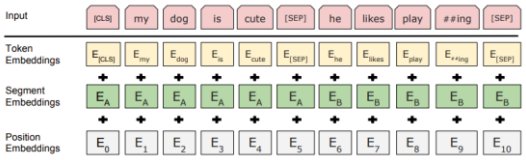
- Model 득징
  - Bi-directional
  - Transformer 구조를 여러 층 활용
  - 다양의 corpus도 사전학습

- 두 가지 사전학습 과제 수행시킨 뒤 fine-tuning
  - Masked Language Model
  - Next Sentence Prediction

사전학습 과제 1: Masked Language Model

- 가려진 단어를 맞추는 과제를 해결함으로써 주변 맥락에 따른 단어의 의미 학습

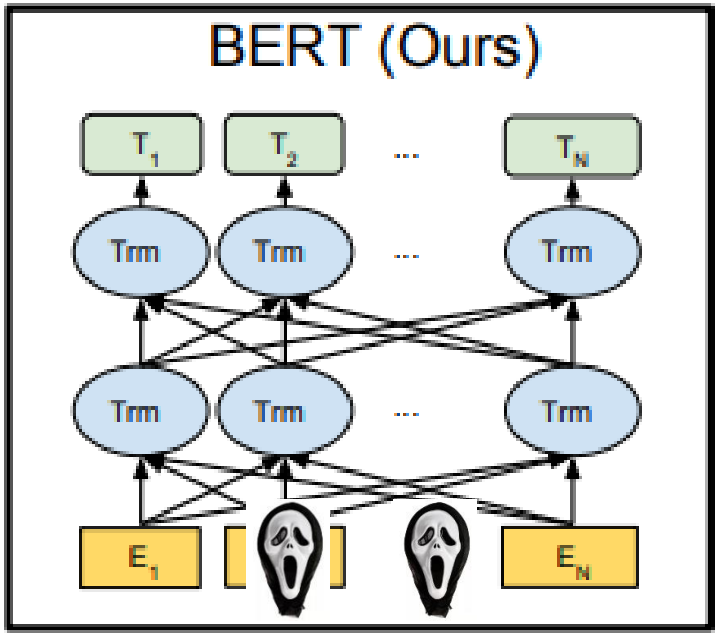


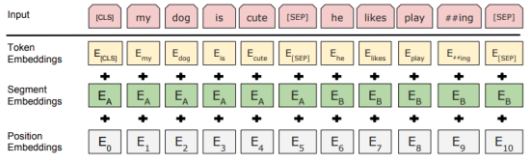


my dog is [MASK] →  → my dog is hairy

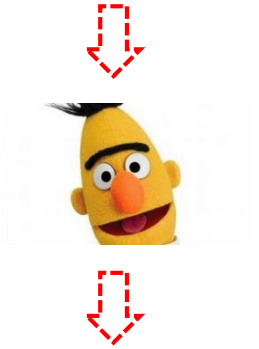
사전학습 과제 2: : Next Sentence Prediction

- 제시된 두 문장이 이어진 문장인지 아닌지를 맞추는 과제를 수행





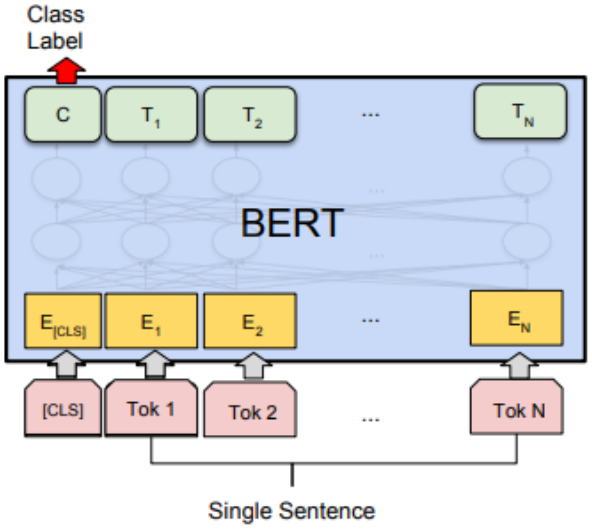
문장 1 : 모자를 쓴 남성이 장바구니를 들고 마트에 갔다.
문장 2 : 그 남자는 우유를 세 동 집어들었다



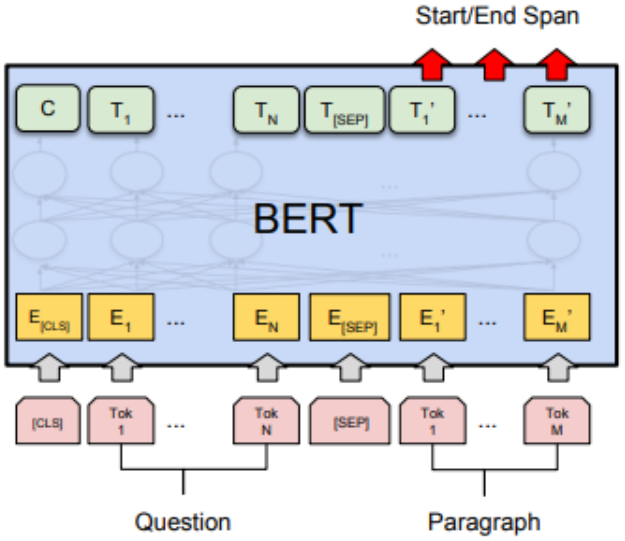
연결된 문장 맞음

BERT Fine-tuning

- 주어진 과제 유형에 따라서 마지막 output layer만 변경하여 간단하게 fine-tuning



(b) Single Sentence Classification Tasks: SST-2, CoLA



(c) Question Answering Tasks: SQuAD v1.1

Transformer 활용 언어모델 : BERT

BERT : 방대한 양의 데이터로 수행한 사전학습 과제의 힘

- 사전 학습 과제
  - 40 epoch, 1,000,000 iterations
  - $BERT_{base}$: 4 Cloud TPUs(=16 TPU chips)
  - $BERT_{large}$: 16 Cloud TPUs(= 64 TPU chips)
  - 엄청난 자원을 사용하여 4일 내내 학습

- Fine-tuning
  - 3~4 epoch만 추가 수행
  - 추가학습은 조금만 수행해도 좋은 성능!!! → 사전학습의 위력

- 11개의 Natural Language Processing task에서 State-Of-The-Art 달성 !