

# Recurrent Neural Network

Hyunjoong Kim

[soy.lovit@gmail.com](mailto:soy.lovit@gmail.com)

[github.com/lovit](https://github.com/lovit)

# Language model & Recurrent Neural Network

# language model

---

- Language model 은 단어열의 확률 분포 모델입니다.
  - $m$  개의 단어로 구성된 문장의 확률을 표현합니다.
    - $sent = [w_1, w_2, \dots, w_m]$
    - $p(sent) = p(w_1, w_2, \dots, w_m)$

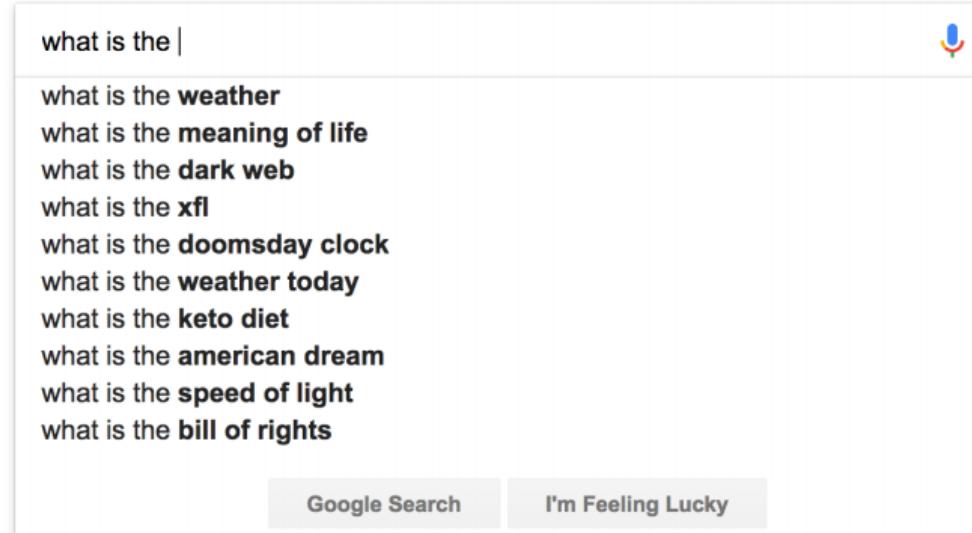
# language model

---

- Language model 을 이용하면
  - 문장이 올바른 문장인지 비문인지를 판단할 수 있습니다.
  - 새로운 문장을 생성할 수도 있습니다.
  - 표절 (plagiarism) 을 찾을 수도 있습니다.

# language model

---



# Classical language model

---

- n-grams 을 이용하는 language model 이 제안되었습니다.
  - $p(w_m | w_1, w_2, \dots, w_{m-1}) = p(w_m | w_{m-2:m-1})$
  - $p(s) = p(w_1, w_2, \dots, w_m) = p(w_1) \times p(w_2 | w_1) \times p(w_3 | w_{1:2}) \dots$

# Classical language model

---

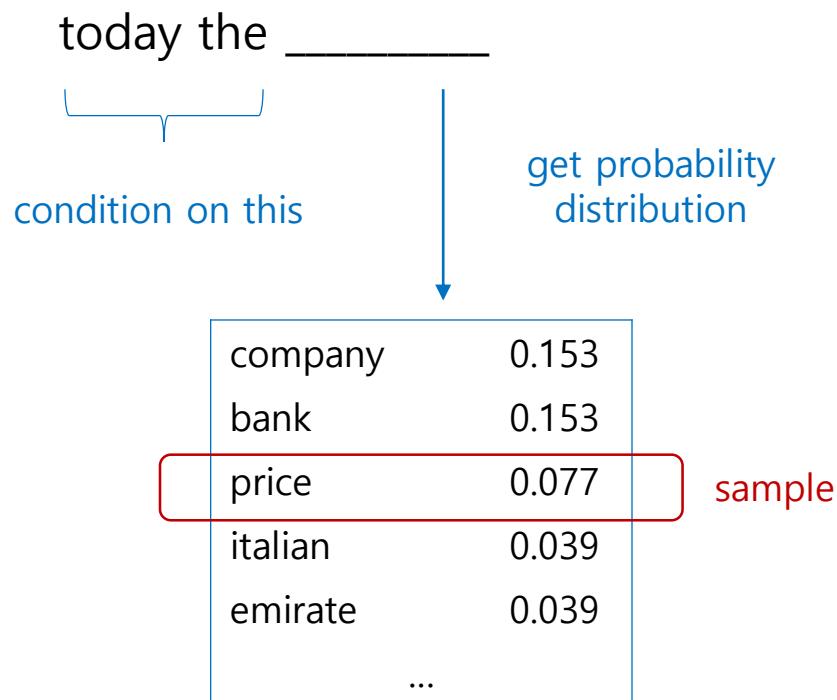
- $p(w_i | w_{i-n+1:i-1})$  은 빈도수로 정의됩니다.

- $$p(w_i | w_{i-n+1:i-1}) = \frac{\#(w_{i-n+1:i})}{\#(w_{i-n+1:i-1})}$$

- $$p(cat | this, is) = \frac{\#(this, is, cat)}{\#(this, is)}$$

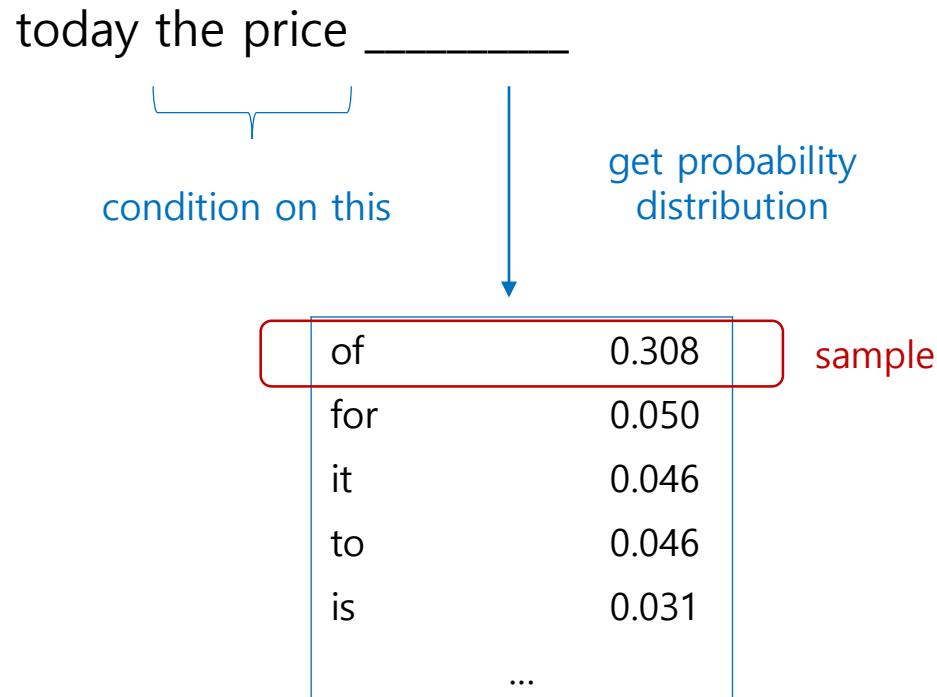
# Sentence generation

- $p(w_i | w_{i-2:i-1})$  가 높은  $w_i$  를 골라서 문장을 생성합니다.



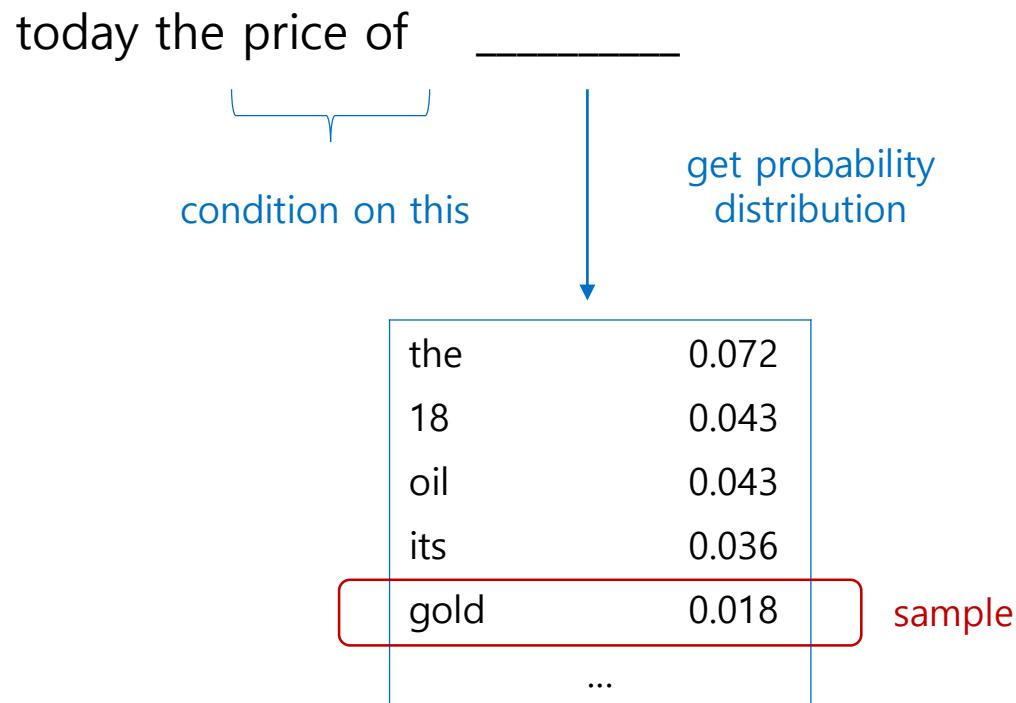
# Sentence generation

- $p(w_i | w_{i-2:i-1})$  가 높은  $w_i$  를 골라서 문장을 생성합니다.



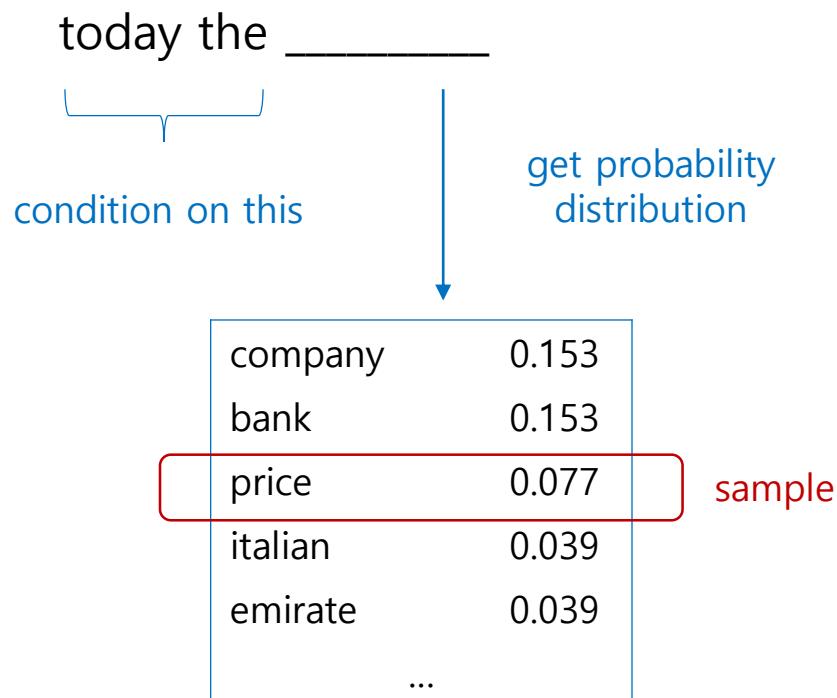
# Sentence generation

- $p(w_i | w_{i-2:i-1})$  가 높은  $w_i$  를 골라서 문장을 생성합니다.



# Sentence generation

- Beam search 는 매 step마다 가장 확률이 높은 k 개의 후보를 남겨둡니다.

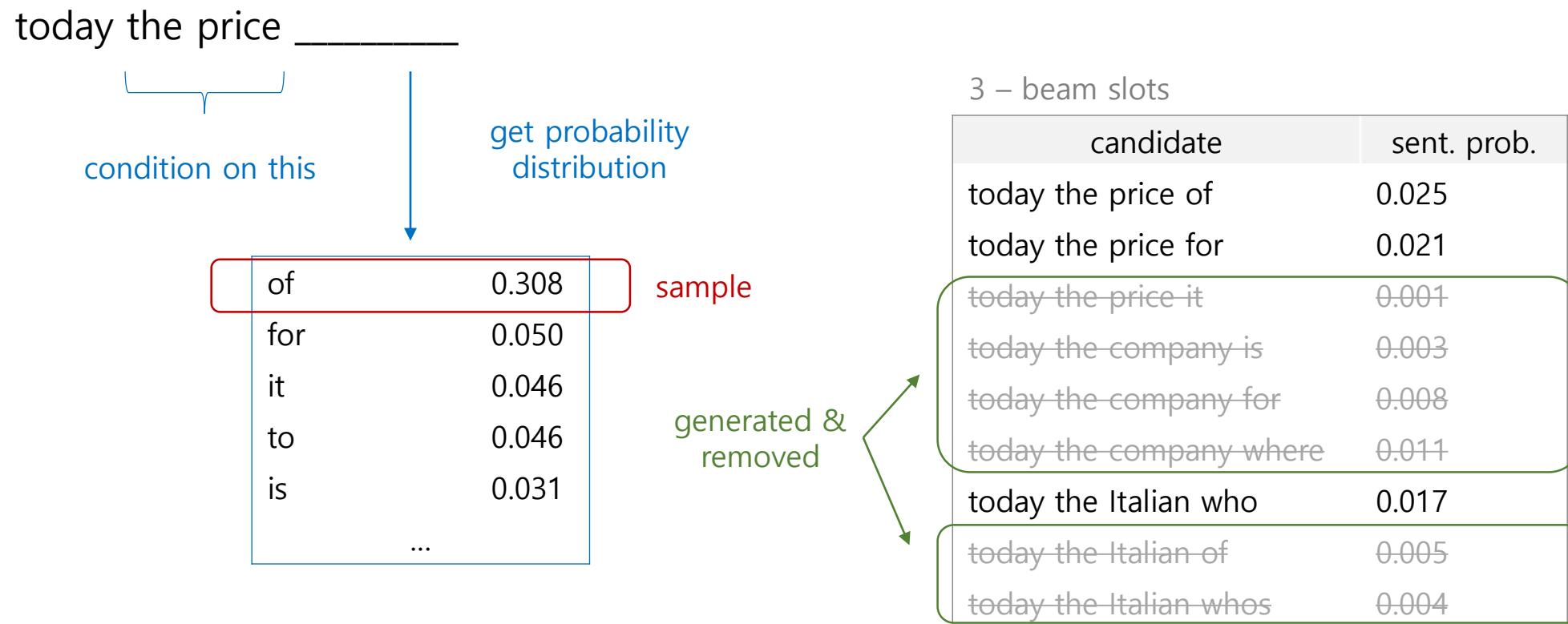


3 – beam slots

candidate	sent. prob.
today the price	0.153
today the company	0.153
today the italian	0.077

# Sentence generation

- Beam search 는 매 step마다 가장 확률이 높은 k 개의 후보를 남겨둡니다.



# Classical language model

---

- n-gram 을 이용하는 language model 은 크게 세 가지 한계가 있습니다.
  - 한 번도 보지 못했던 단어열의 확률은 0 입니다.
    - 충분히 말이 되는 문장이더라도 이전에 보지 못했으면 비문입니다.
    - 단어 간의 유사도가 없습니다.
      - 'this is cat' 을 보았어도 'this is dog' 이라는 문장의 확률이 0 일 수 있습니다.
    - n 이 조금만 커도 n-gram 의 경우의 수가 매우커집니다.

# Classical language model

---

- smoothing 처음 본 n-gram 의 확률을 0 보다 크게 만듭니다.
  - Additive (Laplace) smoothing
    - $p(w_i | w_{i-n+1:i-1}) = \frac{\# w_{i-n+1:i} + \alpha}{\# w_{i-n+1:i-1} + \alpha d}$  로 정의합니다.
    - 처음 보는 단어라 하더라도 0이 아닌 확률이 생깁니다.
    - 그 외에도 다양한 smoothing 방법이 제안되었습니다.

# Classical language model

---

- Back-off 는 길이가 짧은 n-grams 을 함께 context 로 이용합니다.

$$\bullet \quad p(w_4|w_1, w_2, w_3) = \alpha_1 \times \frac{\#(w_{1:4})}{\#(w_{1:3})} + \alpha_2 \times \frac{\#(w_{2:4})}{\#(w_{2:3})} + \alpha_3 \times \frac{\#(w_{3:4})}{\#(w_3)}$$

- Katz's back-off model

$$\bullet \quad P(w_i | w_{i-n+1}, \dots, w_{i-1}) = \begin{cases} \frac{\#(w_{i-n+1:i})}{\#(w_{i-n+1:i-1})} & \text{if } (w_{i-n+1:i}) > k \\ P(w_i | w_{i-n+2}, \dots, w_{i-1}) & \text{otherwise} \end{cases}$$

# Classical language model

---

- 그러나  $n$  이 클 때의  $n$ -grams 의 개수가 지나치게 커지는 문제와  
'cat', 'dog' 의 유사도를 반영하지 못하는 문제는 해결하기 어려웠습니다.

# Feed-forward network based language model

---

- Bengio (2003) 의 neural probabilistic language model (NPLM) 은 classical language model 이 풀기 어려운 두 종류의 문제를 겪냥한 neural network 기반 language model 입니다.

**First**, it is not taking into account contexts farther than 1 or 2 words, **second** it is not taking into account the "similarity" between words. For example, having seen the sentence "The cat is walking in the bedroom" in the training corpus should help us generalize to make the sentence "A dog was running in a room" almost as likely, simply because "dog" and "cat" (resp. "the" and "a", "room" and "bedroom", etc...) have similar semantic and grammatical roles.

(Bengio et al., 2003)

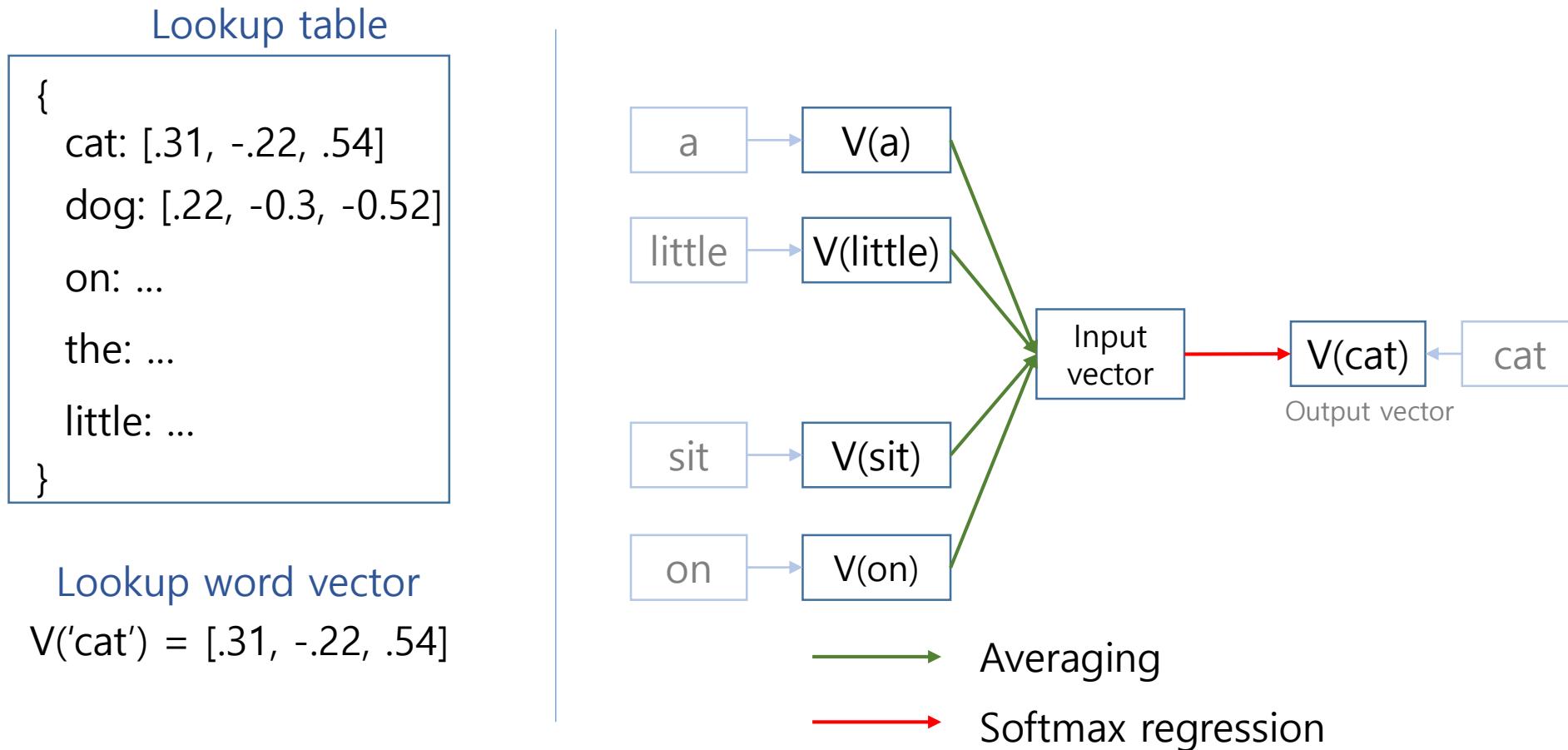
# Feed-forward network based language model

---

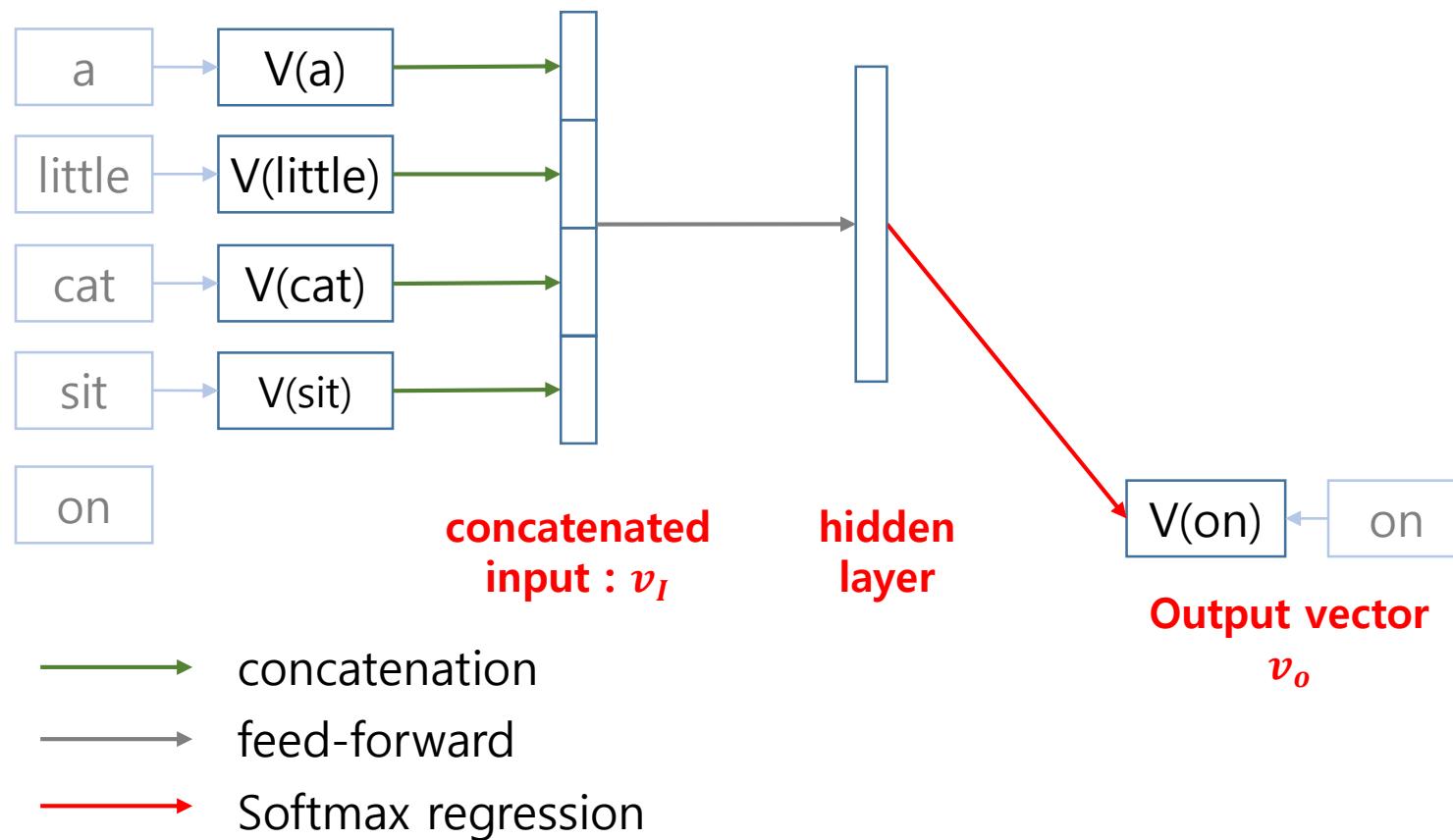
- Neural probabilistic, feed-forward network based language model 은 word embedding (e.g. Word2Vec) 의 시작이기도 합니다.
  - 학습 결과 'dog' 과 'cat' 의 벡터가 비슷하게 학습됩니다.

# Word2Vec

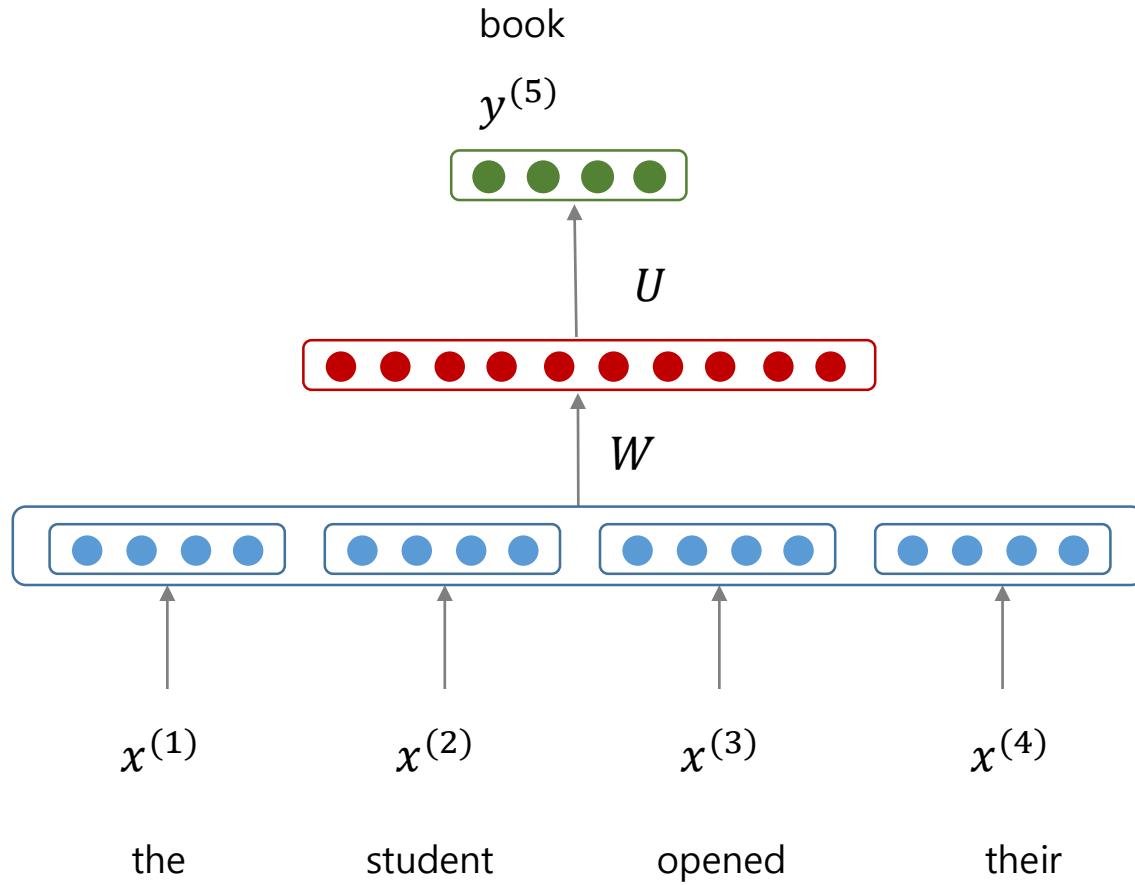
- Word2Vec은  $|V|$  개의 classes 를 예측하는 Softmax regression 입니다.



# Feed-forward network based language model



# Feed-forward network based language model



output distribution

$$\hat{y} = \text{softmax}(Uh)$$

hidden layer

$$h = f(We)$$

concatenated word embedding

$$e = [e^{(1)}; e^{(2)}; e^{(3)}; e^{(4)}]$$

# Feed-forward network based language model

---

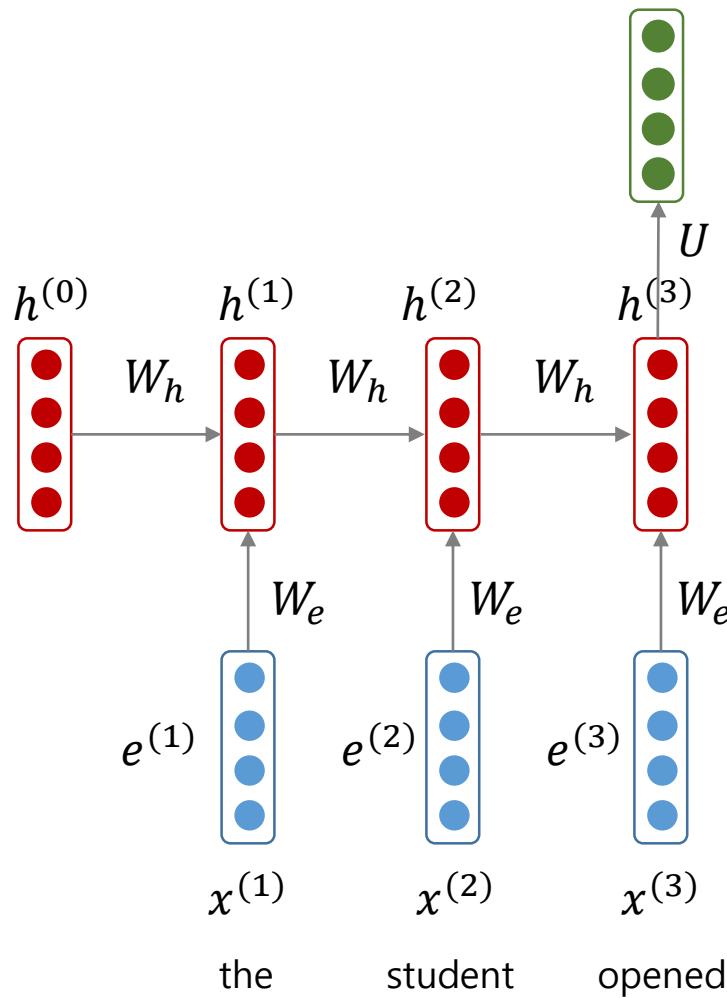
- Bengio et al., (2003) 에서 제안된 모델은 단어 간의 유사성을 표현하며, n-grams 을 hidden layer 의 output 으로 표현함으로써 모델의 크기를 가볍게 만들었습니다.
- 그러나 제한된 길이의 context 밖에 이용하지 못합니다.
  - $n - 1$  개의 단어만을 이용합니다.
  - 문장 전체의 문맥을 이용하는 language model 을 만들고 싶습니다.

# Recurrent neural network based language model

---

- Recurrent neural network 는 임의의 길이의 context 를 hidden states 로 이용할 수 있습니다.
- 앞에 등장하였던 단어들의 정보를 압축하여 하나의 hidden states vector,  $h^{(t)}$ 로 표현합니다.

# Recurrent neural network based language model



output distribution

$$\hat{y} = \text{softmax}(Uh)$$
$$p(y^{(t)} | y_{1:t-1})$$

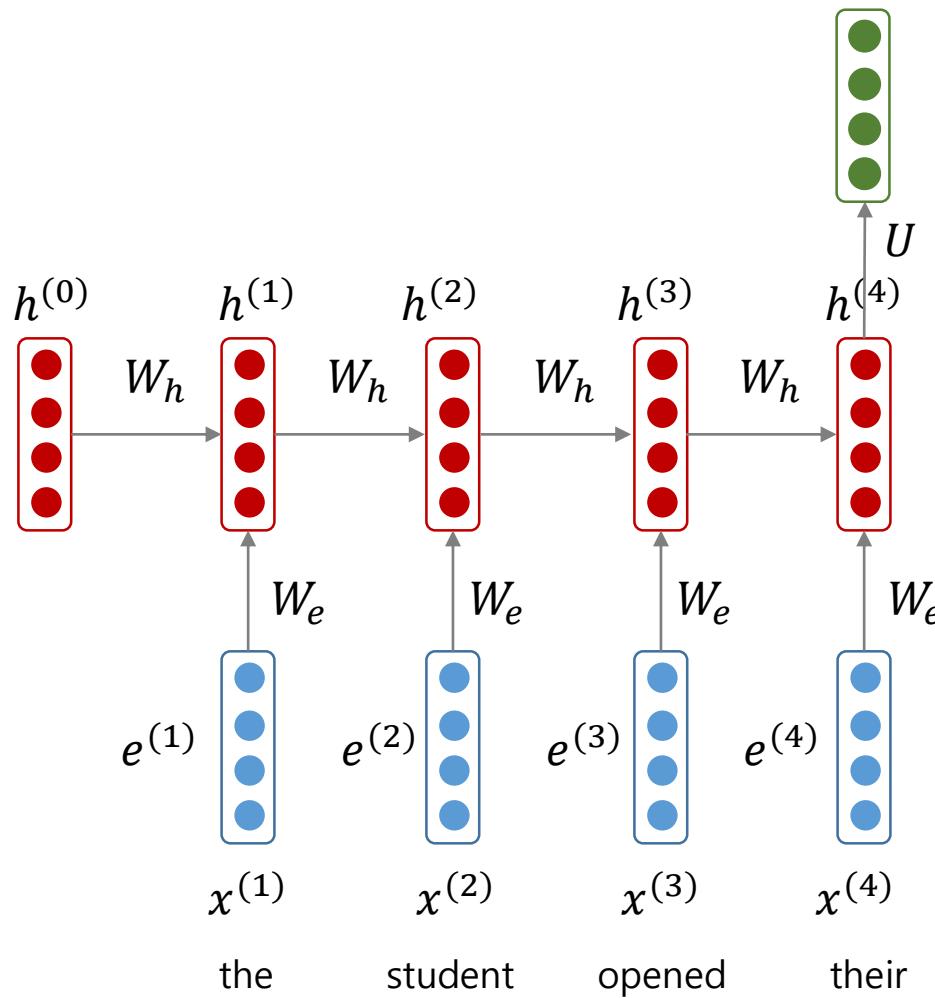
hidden layer

$$h^{(t)} = f(W_h \cdot h^{(t-1)} + W_e e^{(t)})$$
$$h^{(0)} : \text{initial hidden state}$$

word embedding

$$e^{(t)} = E \cdot x^{(t)}$$

# Recurrent neural network based language model



output distribution

$$\hat{y} = \text{softmax}(Uh)$$
$$p(y^{(t)}|y_{1:t-1})$$

hidden layer

$$h^{(t)} = f(W_h \cdot h^{(t-1)} + W_e e^{(t)})$$
$$h^{(0)} : \text{initial hidden state}$$

word embedding

$$e^{(t)} = E \cdot x^{(t)}$$

# Recurrent neural network based language model

---

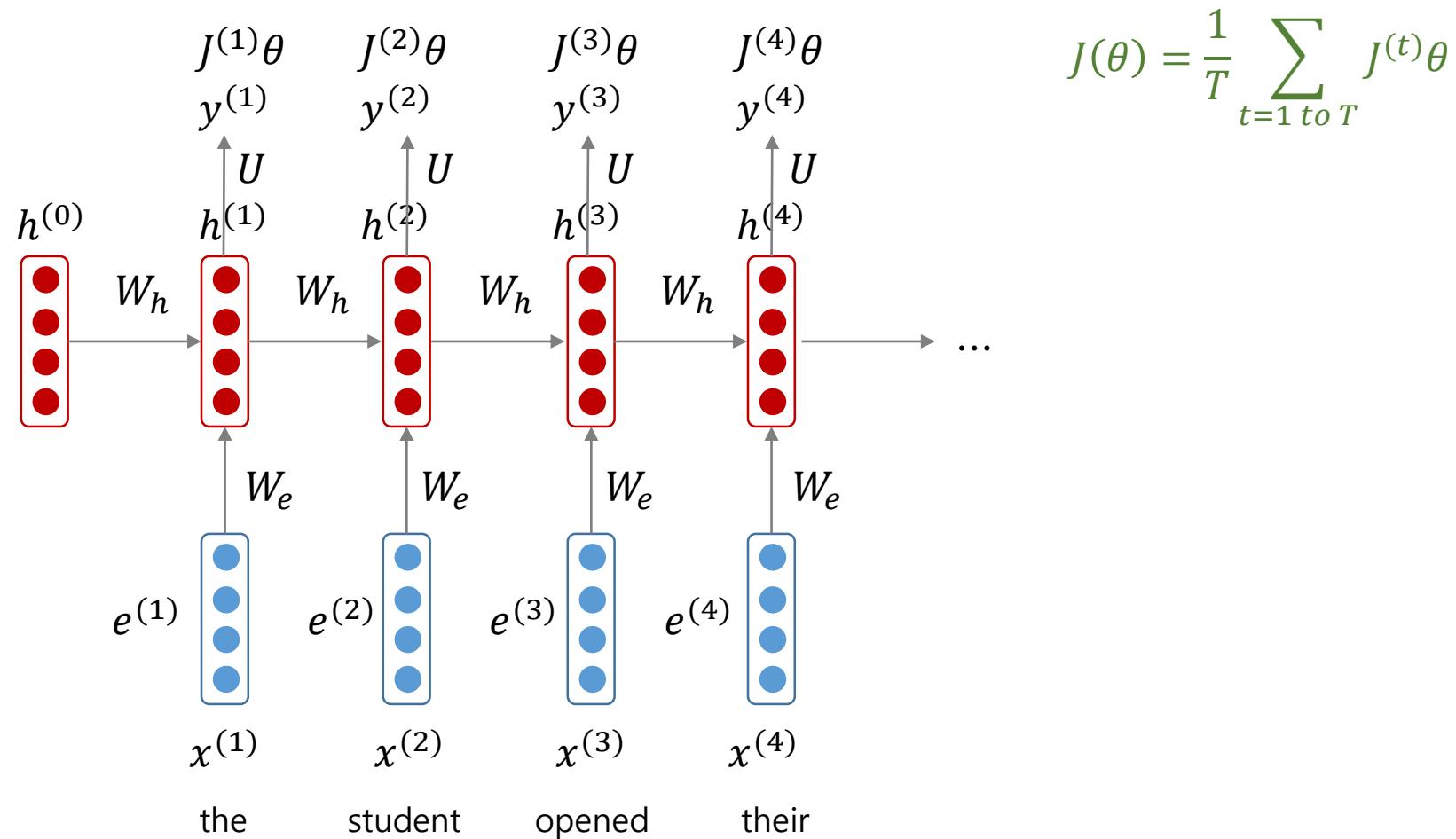
- RNN 은 이전 단계의 hidden states 와 현재 단계의 input vector 를 합쳐 현재 단계의 hidden layer 의 input 으로 이용합니다.
  - 이전까지의 단어에 의한 맥락과 현재 단어의 정보를 모두 이용하여 현재의 맥락을 표현합니다.

$$h^{(t)} = f(W_h \cdot h^{(t-1)} + W_e e^{(t)})$$

$$f([W_h; W_e] \cdot [h^{(t-1)}; e^{(t)}])$$

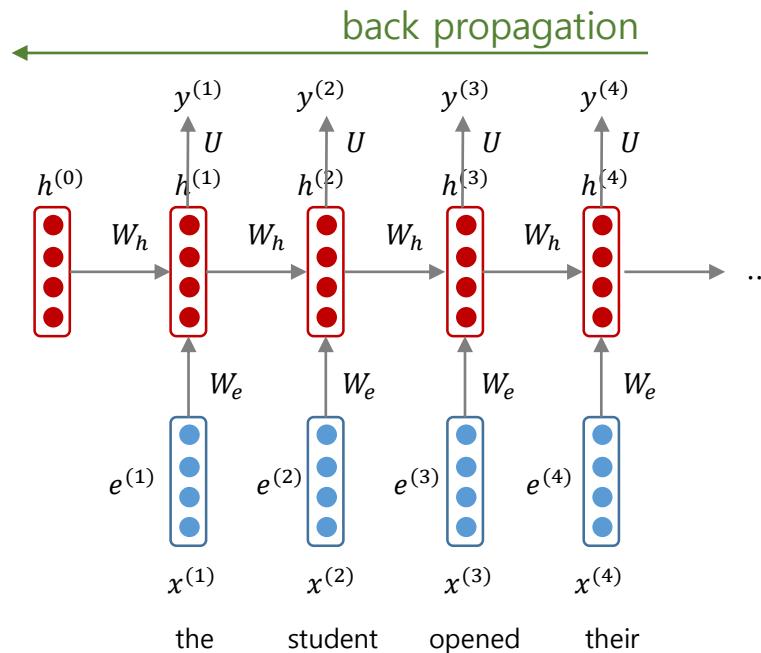
# Recurrent neural network based language model

- RNN 의 loss 는 outputs 의 모든 loss 의 합입니다.



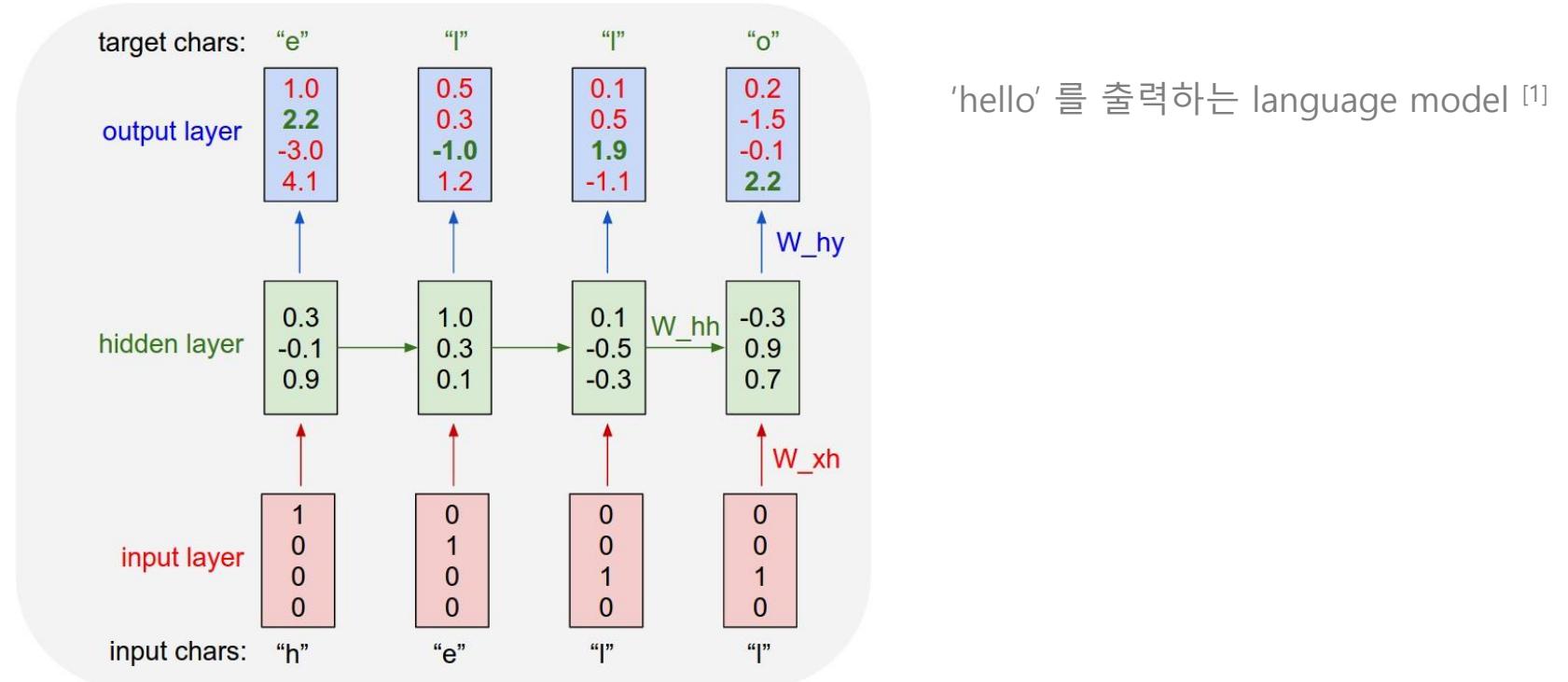
# Recurrent neural network based language model

- Back Propagation Through Time (BPTT) 는 하나의 sequence 전체에 대한 loss 를 모두 계산한 다음 수행하는 back propagation 입니다.
  - 멀리 떨어진 단어 간의 연관성까지도  $[W_h; W_e]$  에 학습되도록 만듭니다.



# Character – level, RNN based language model

- RNN 의 input 단위를 ‘단어’ → 글자’로 바꾸면 character level 의 language model 을 만들 수 있습니다.

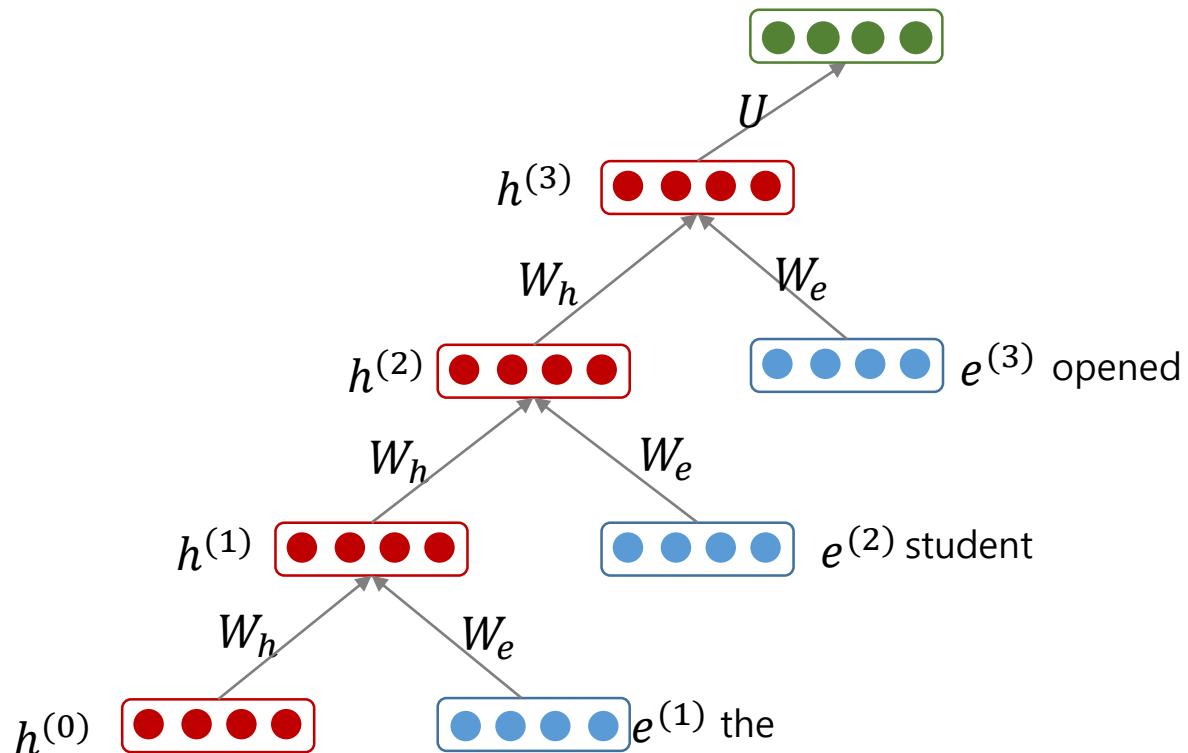


[1] <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

[2] Kim, Y., Jernite, Y., Sontag, D., & Rush, A. M. (2016, February). Character-Aware Neural Language Models. In AAAI (pp. 2741-2749).

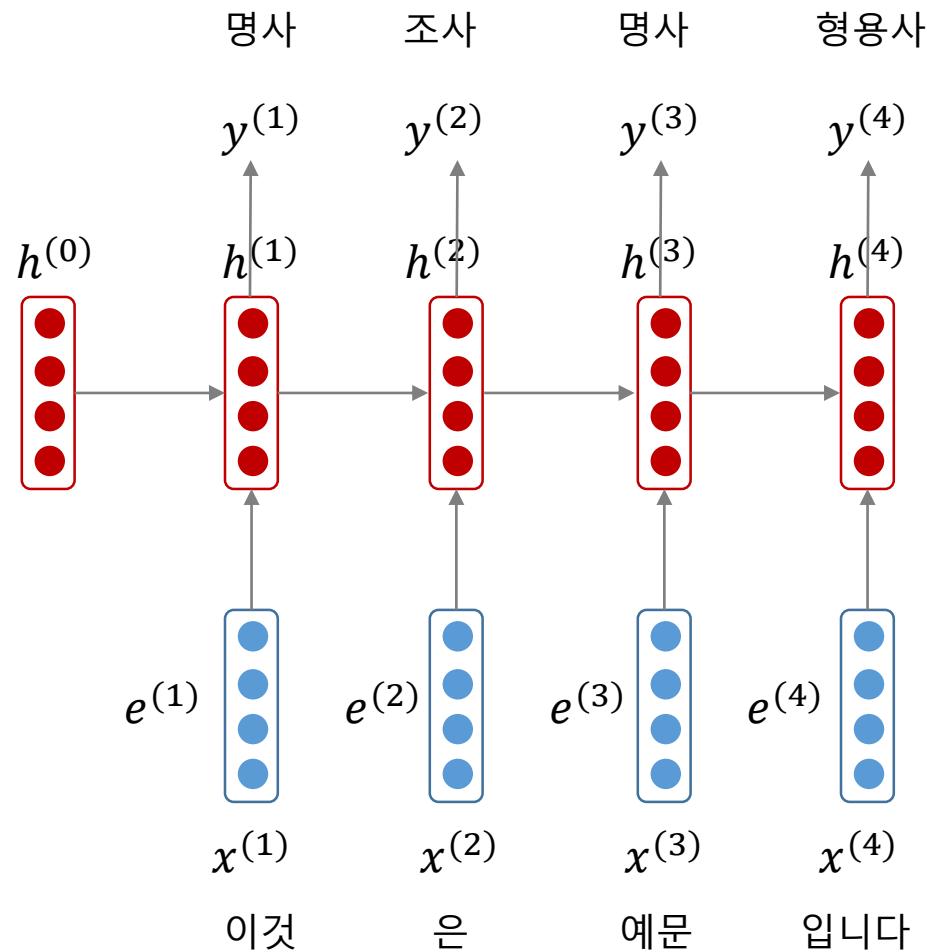
# Recurrent neural network as Feed-forward neural network

- RNN 은 Feed-forward NN 의 한 종류입니다.
  - 문장의 길이가 가변적인 문제를 구조적으로 해결하였습니다.



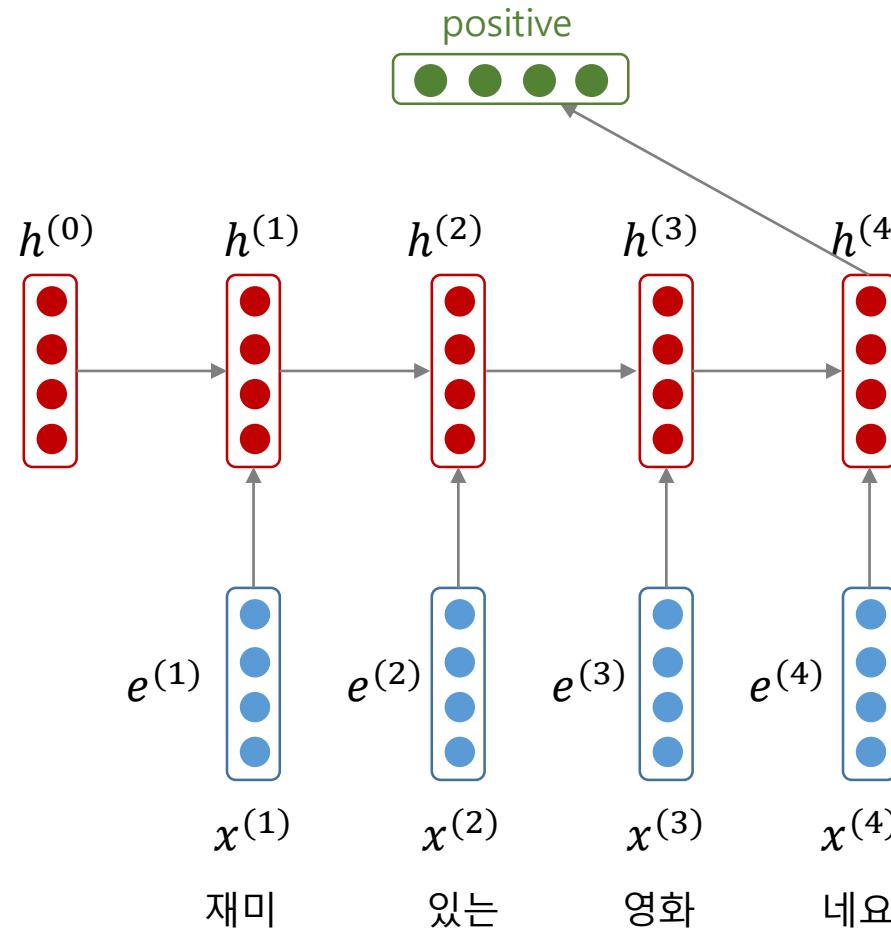
# Recurrent neural network based tagging

- RNN 은 sequential labeling 에 이용될 수 있습니다.



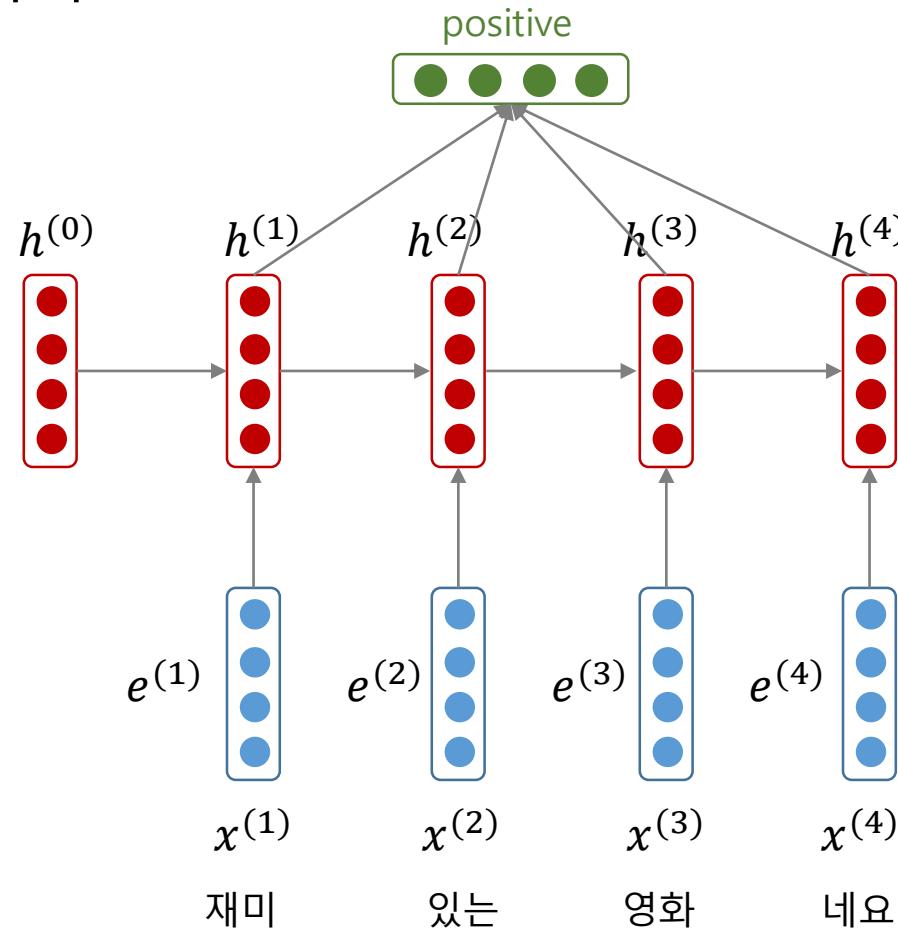
# Recurrent neural network based sentence classification

- 마지막 hidden states vector 를 이용하여 sentence classification 을 할 수 있습니다.



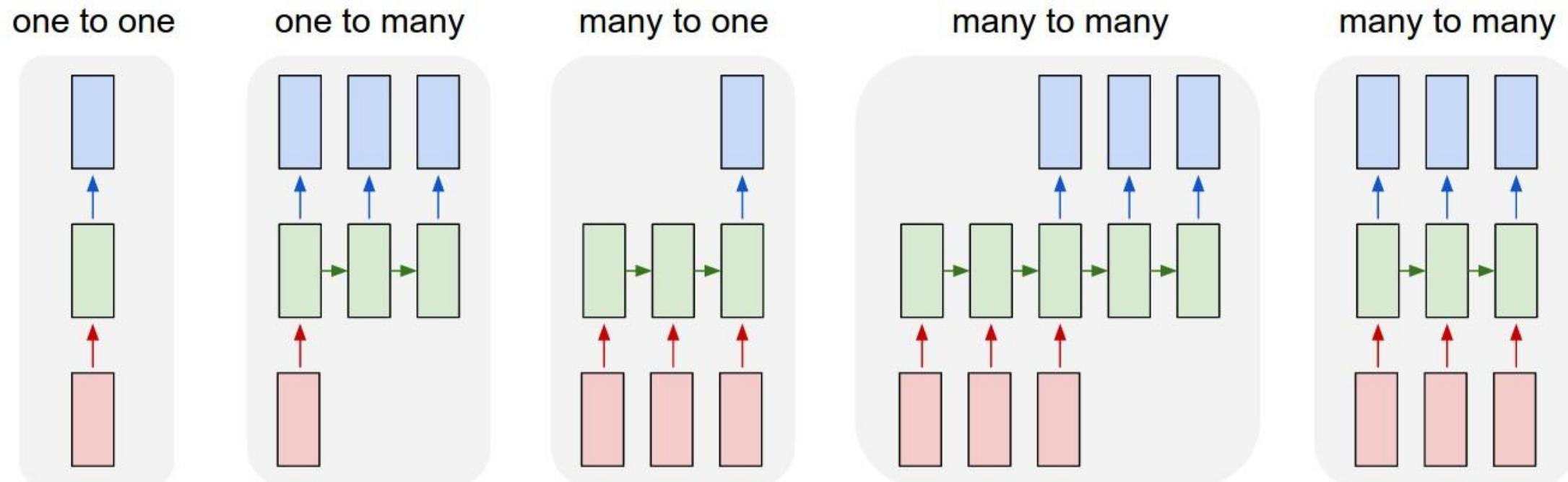
# Recurrent neural network based sentence classification

- Hidden vectors 의 element-wise max or mean 을 이용하여 문장 벡터를 만들 수도 있습니다.



# Recurrent neural network

- RNN 은 input 과 output 의 개수 / 순서에 따라 여러 형태로 디자인할 수 있습니다.



# Recurrent neural network

---

- Convolutional neural network 는 locality 라는 데이터의 성질을 반영한 neural network 구조입니다.
  - Locality 는 이미지 데이터의 특징입니다.
  - NLP 에서는 n-grams 의 역할을 합니다.

# Recurrent neural network

---

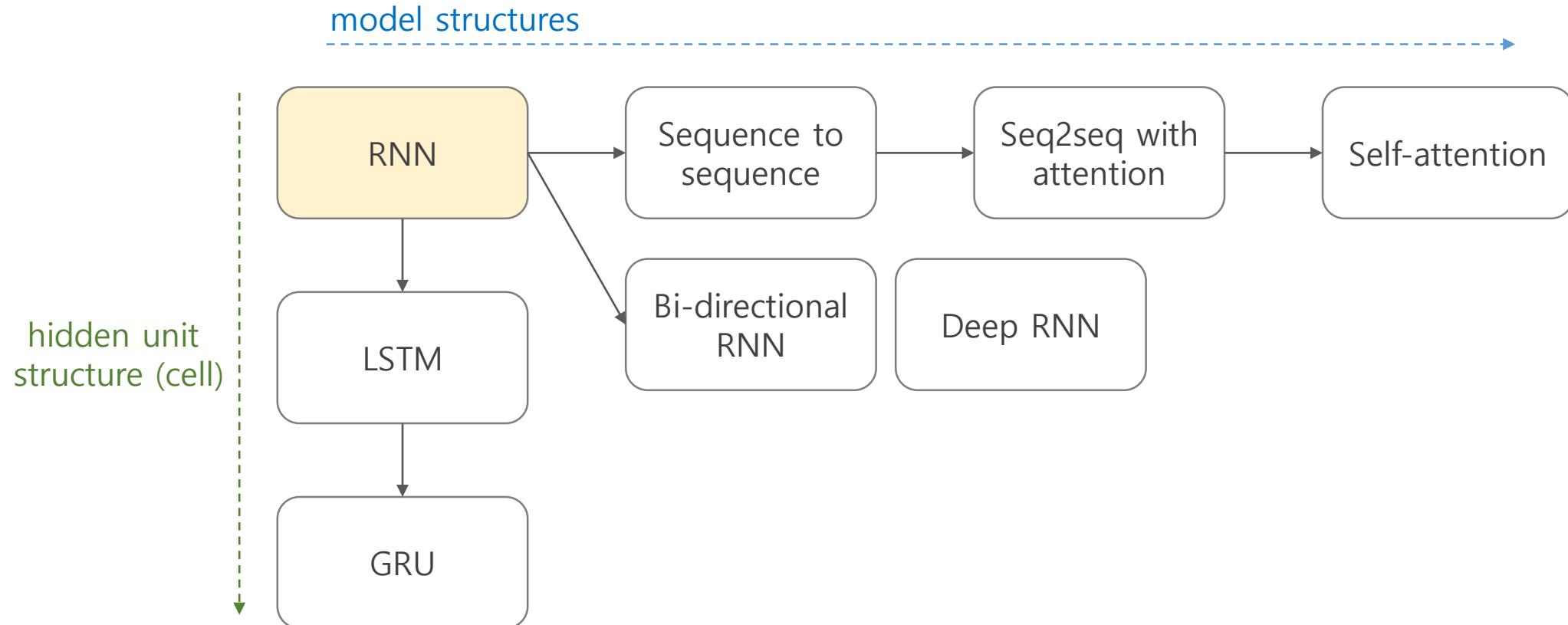
- Recurrent neural network 는 sequence 라는 텍스트 데이터의 특징을 반영한 neural network 입니다.
  - Sequence 의 길이가 가변적입니다.
  - 단어/문맥/문장의 의미는 문장에 존재하는 단어들을 모두 고려해야 합니다.
  - 한정된 windows 안의 정보만을 이용하는 feed-forward neural network 보다 더 자유롭게 정보를 이용할 수 있습니다.

# Recurrent neural network

---

- 그러나 RNN 의 기본 구조만으로는 여러 문제들이 발생합니다.  
최근 RNN 의 발전은 이러한 문제점들을 해결하는 과정이기도 합니다.

# Recurrent neural network



# Hidden layer unit

# Gated Recurrent Unit (GRU)

---

- RNN 은 gradient descent vanishing problem 이 발생합니다.
  - RNN 은 멀리 떨어진 단어 간의 연관성도 학습하려는 모델입니다.  
(language model 에서의  $x^{(1)}$  과  $x^{(6)}$  처럼)
  - $x^{(1)}$  과  $x^{(6)}$  이 연결되기 위해서는 여러 번의 gradient 를 거칩니다.

$$y^{(6)} = g(Uh^{(6)})$$

$$h^{(6)} = f(W_h \cdot h^{(5)} + W_e e^{(6)})$$

...

$$h^{(1)} = f(W_h \cdot h^{(0)} + W_e e^{(1)})$$

# Gated Recurrent Unit (GRU)

---

- RNN 은 gradient descent vanishing problem 이 발생합니다.
  - 본래는 long dependency 를 학습하기 위해 제안된 모델이지만, 사실상 local dependency 밖에 학습할 수 없습니다.
- 근본적인 이유는
  - (1) input 정보를 선택적으로 hidden 에 넣지 못하였으며
  - (2) 불필요한 문맥을 버리지 못했기 때문입니다.

# Gated Recurrent Unit (GRU)

---

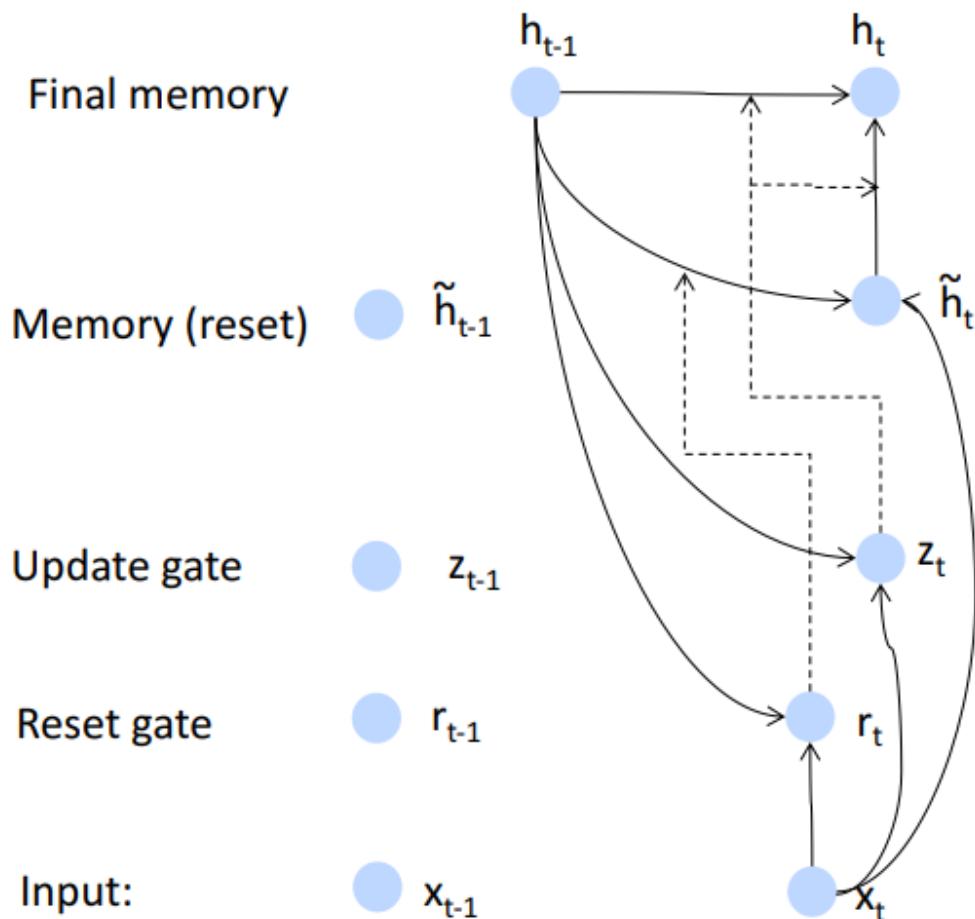
- LSTM, GRU 는 hidden states 에 필요한 정보만을 저장하여 long dependency 를 더 잘 학습하기 위한 방법입니다.

# Gated Recurrent Unit (GRU)

---

- Gated Recurrent Unit (GRU) 는 두 종류의 gate 를 이용합니다.
  - update gate :  $z_t = \sigma(W^{(z)} \cdot x_t + U^{(z)} \cdot h_{t-1})$
  - reset gate :  $r_t = \sigma(W^{(r)} \cdot x_t + U^{(r)} \cdot h_{t-1})$
  - new memory content :  $\tilde{h}_t = \tanh(W \cdot x_t + r \circ U \cdot h_{t-1})$
  - final memory :  $h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$

# Gated Recurrent Unit (GRU)



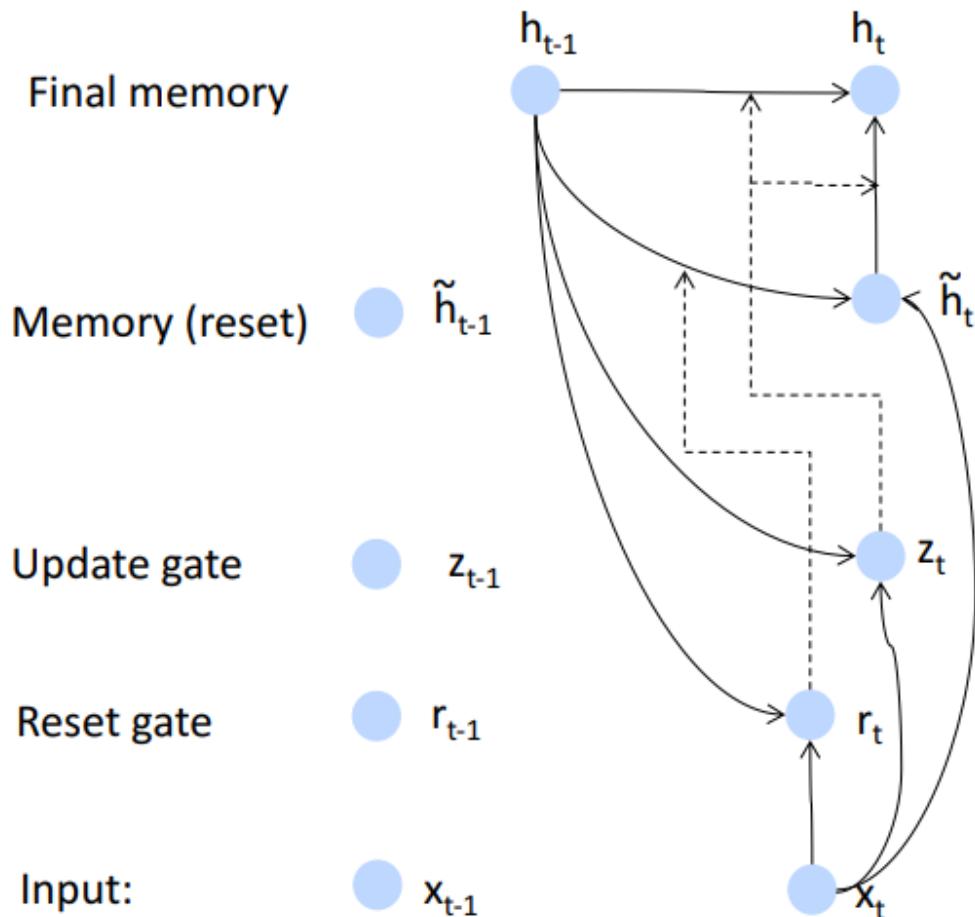
$$z_t = \sigma(W^{(z)} \cdot x_t + U^{(z)} \cdot h_{t-1})$$

$$r_t = \sigma(W^{(r)} \cdot x_t + U^{(r)} \cdot h_{t-1})$$

$$\tilde{h}_t = \tanh(W \cdot x_t + r \circ U \cdot h_{t-1})$$

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

# Gated Recurrent Unit (GRU)



$$z_t = \sigma(W^{(z)} \cdot x_t + U^{(z)} \cdot h_{t-1})$$

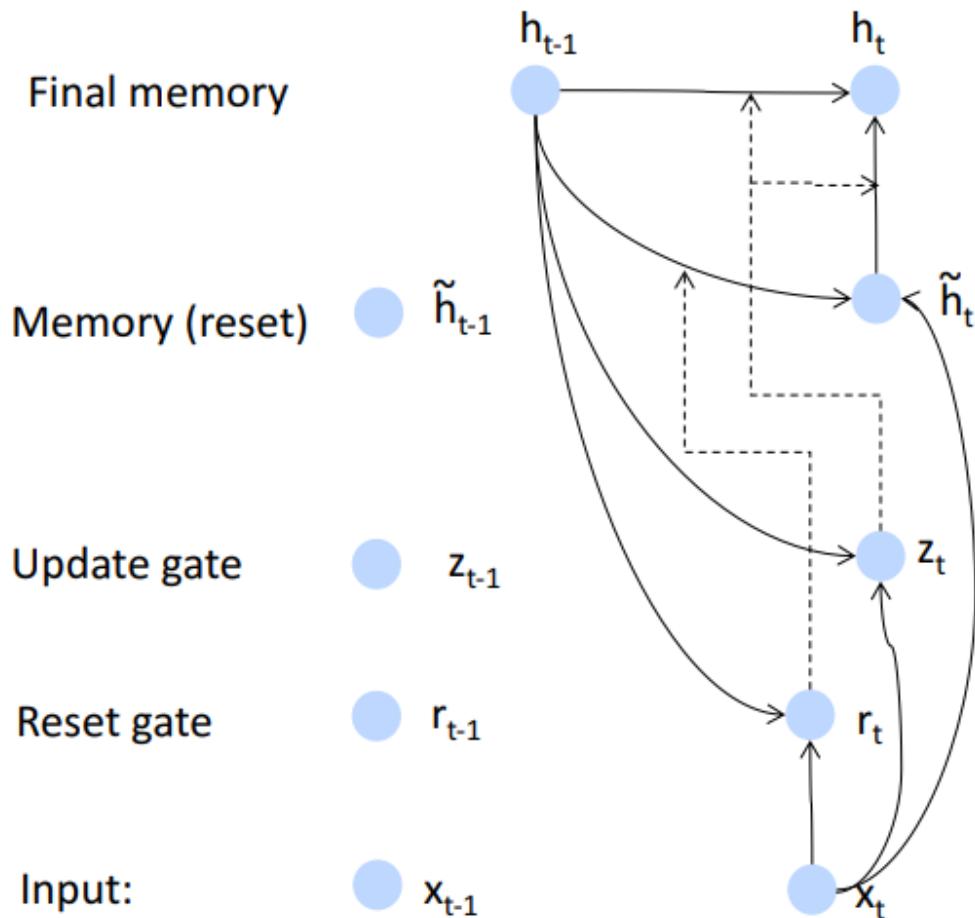
$$r_t = \sigma(W^{(r)} \cdot x_t + U^{(r)} \cdot h_{t-1})$$

$$\tilde{h}_t = \tanh(W \cdot x_t + r \circ U \cdot h_{t-1})$$

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

- $r_t$  이 0 과 비슷하면 이전 hidden state 의 정보를 거의 이용하지 않습니다.
- 대신  $x_t$  를 이용하여 현재의 state 를 표현합니다.

# Gated Recurrent Unit (GRU)



$$z_t = \sigma(W^{(z)} \cdot x_t + U^{(z)} \cdot h_{t-1})$$

$$r_t = \sigma(W^{(r)} \cdot x_t + U^{(r)} \cdot h_{t-1})$$

$$\tilde{h}_t = \tanh(W \cdot x_t + r \circ U \cdot h_{t-1})$$

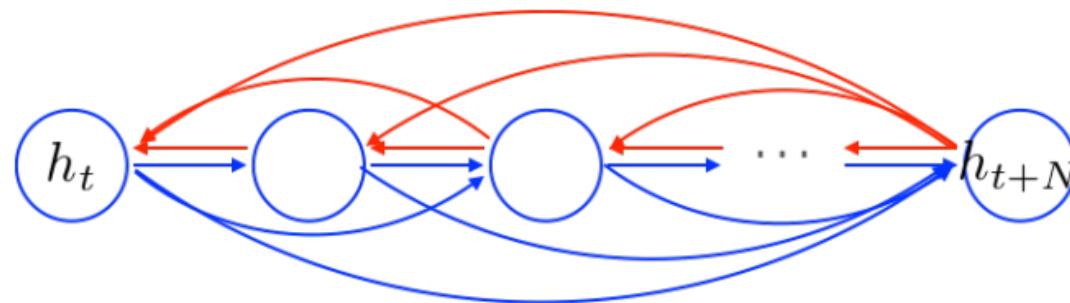
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

- $z_t$  이 0 과 비슷하면 새로 만들어진 memory content  $\tilde{h}_t$  를 현재의 state 로 이용합니다.
- $z_t$  이 1 에 가까우면 이전의 memory 를 그대로 이용합니다 ( $x_t$  를 무시합니다)

# Gated Recurrent Unit (GRU)

---

- GRU 나 LSTM 과 같은 gated 방법이 long dependency 를 학습 할 수 있는 이유는 “아마도” gates 에 의하여 멀리 떨어진 step 간에 shortcut 이 생기기 때문일 것이라 짐작합니다.
  - Perhaps we can create *adaptive* shortcut connections.
  - Let the net prune unnecessary connections *adaptively*.



# Long Short-Term Memory (LSTM)

---

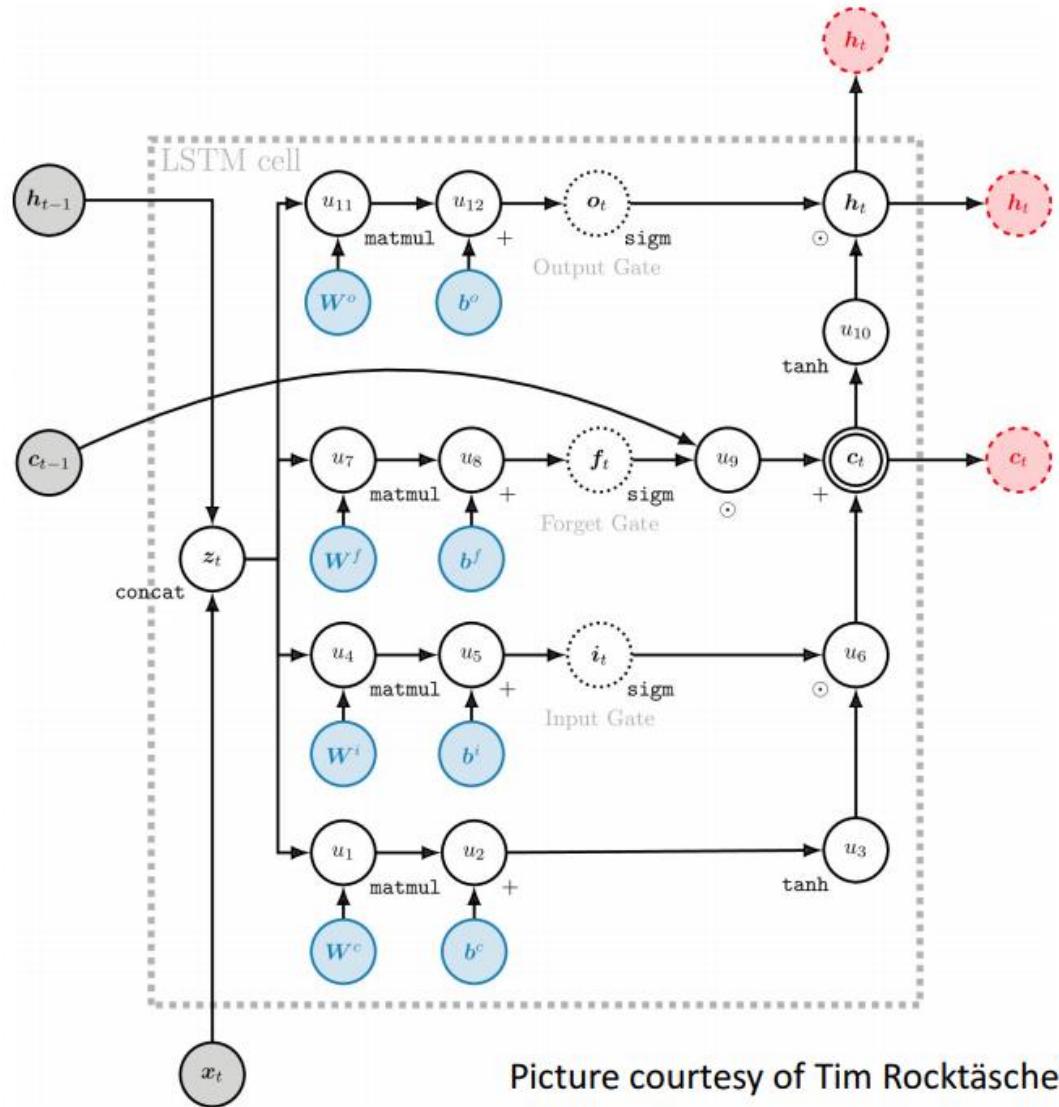
- Long Short-Term Memory (LSTM) 은 GRU 보다 먼저 제안된 방법입니다.
  - GRU 의 목표가 “비싼 계산 비용이 드는 LSTM 의 기능은 유지하면서 빠르게 학습할 수 있는 가벼운 cell 을 만드는 것” 이었습니다.

# Long Short-Term Memory (LSTM)

---

- LSTM 은 세 가지 gates 로 이뤄져 있습니다.
  - input gate :  $i_t = \sigma(W^{(i)} \cdot x_t + U^{(i)} \cdot h_{t-1})$
  - forget gate :  $f_t = \sigma(W^{(f)} \cdot x_t + U^{(f)} \cdot h_{t-1})$
  - output gate :  $o_t = \sigma(W^{(o)} \cdot x_t + U^{(o)} \cdot h_{t-1})$
- new memory cell content :  $\tilde{c}_t = \tanh(W^{(c)} \cdot x_t + U^{(c)} \cdot h_{t-1})$
- final memory cell :  $c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$
- final hidden state :  $h_t = o_t \circ \tanh(c_t)$

# Long Short-Term Memory (LSTM)



$$i_t = \sigma(W^{(i)} \cdot x_t + U^{(i)} \cdot h_{t-1})$$

$$f_t = \sigma(W^{(f)} \cdot x_t + U^{(f)} \cdot h_{t-1})$$

$$o_t = \sigma(W^{(o)} \cdot x_t + U^{(o)} \cdot h_{t-1})$$

$$\tilde{c}_t = \tanh(W^{(c)} \cdot x_t + U^{(c)} \cdot h_{t-1})$$

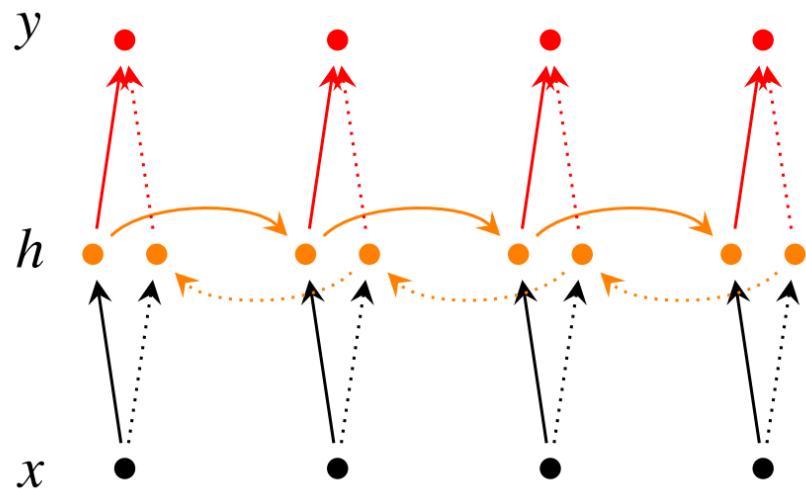
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

$$h_t = o_t \circ \tanh(c_t)$$

# Structures

# Bidirectional RNN

- 문맥을 표현할 때에는 앞/뒤의 단어를 모두 살펴보는 것이 좋습니다.
  - Bidirectional RNN 은 두 개의 독립적인 RNN 의 hidden states 를 concatenation 하여 최종 hidden states vector 로 이용합니다.

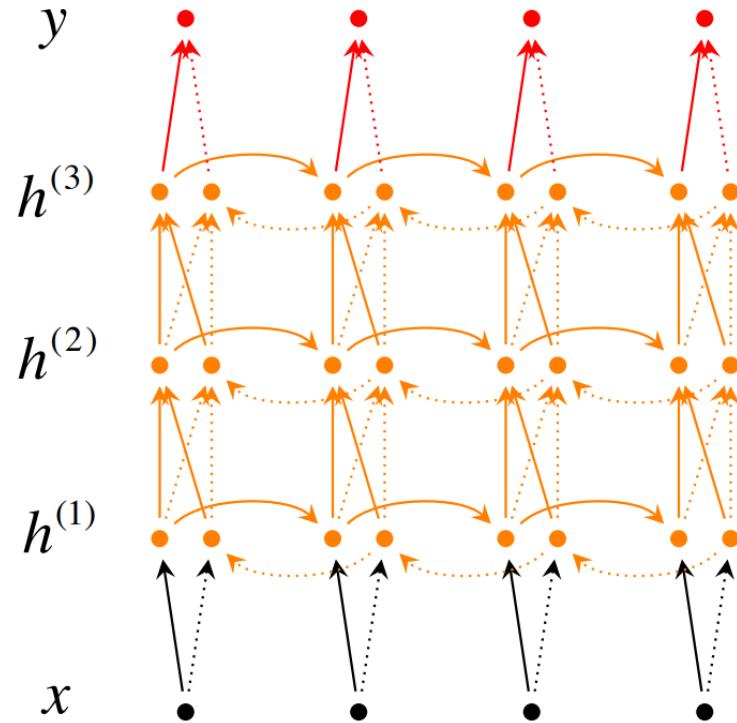


$$\begin{aligned}\vec{h}_t &= f(\vec{W} \cdot x_t + \vec{V} \cdot \vec{h}_{t-1}) \\ \overleftarrow{h}_t &= f(\overleftarrow{W} \cdot x_t + \overleftarrow{V} \cdot \overleftarrow{h}_{t-1})\end{aligned}$$

$$y_t = g(U[\vec{h}_t; \overleftarrow{h}_t])$$

# Deep RNN

- 복잡한 sequence pattern 을 학습하기 위하여 여러 층의 hidden layers 를 쌓을 수도 있습니다.



$$\vec{h}_t^{(i)} = f \left( \vec{W}^{(i)} \cdot h_t^{(i-1)} + \vec{V}^{(i)} \cdot \overrightarrow{h}_{t-1}^{(i-1)} \right)$$

$$\overleftarrow{h}_t^{(i)} = f \left( \overleftarrow{W}^{(i)} \cdot h_t^{(i-1)} + \overleftarrow{V}^{(i)} \cdot \overleftarrow{h}_{t-1}^{(i-1)} \right)$$

$$y_t = g \left( U \left[ \overrightarrow{h}_t^{(L)} ; \overleftarrow{h}_t^{(L)} \right] \right)$$

## ELMo (Embedding from Language Models)

---

- ELMo 는 2 Layer LSTM 을 이용하는 word embedding 입니다.
  - 각 layers 의 hidden vectors 결합을 embedding vector 로 이용합니다.
- ELMo 는 단어의 문맥을 표현할 수 있습니다.
  - Word2Vec 은 단어가 문맥과 상관없이 고정된 벡터를 지닙니다.
  - 앞/뒤, 문장 전체의 단어를 고려하는 hidden vectors 를 단어 벡터로 이용함으로써 문맥을 표현할 수 있습니다.

# ELMo (Embedding from Language Models)

- 같은 단어 ‘play’ 가 문맥에 따라 다른 벡터로 표현됩니다.

Source		Nearest Neighbors
GloVe	play	playing, game, games, played, players, plays, player, Play, football, multiplayer
biLM	Chico Ruiz made a spectacular <span style="border: 1px solid red;">play</span> on Alusik 's grounder {...}	Kieffer , the only junior in the group , was commended for his ability to hit in the clutch , as well as his all-round excellent <span style="border: 1px solid red;">play</span> .
	Olivia De Havilland signed to do a Broadway <span style="border: 1px solid red;">play</span> for Garson {...}	{...} they were actors who had been handed fat roles in a successful <span style="border: 1px solid red;">play</span> , and had talent enough to fill the roles competently , with nice understatement .

Table 4: Nearest neighbors to “play” using GloVe and the context embeddings from a biLM.

## ELMo (Embedding from Language Models)

---

- 단어  $t_k$  의 임베딩 벡터는 모든 hidden vectors 의 선형 결합입니다.
  - 결합 비율  $s_j$  는 목적에 따라 다릅니다.
  - $\gamma$  는 안정적 학습을 위한 상수입니다.
  - $ELMo_k = \gamma \sum_{j=0}^L s_j h_{k,j}$

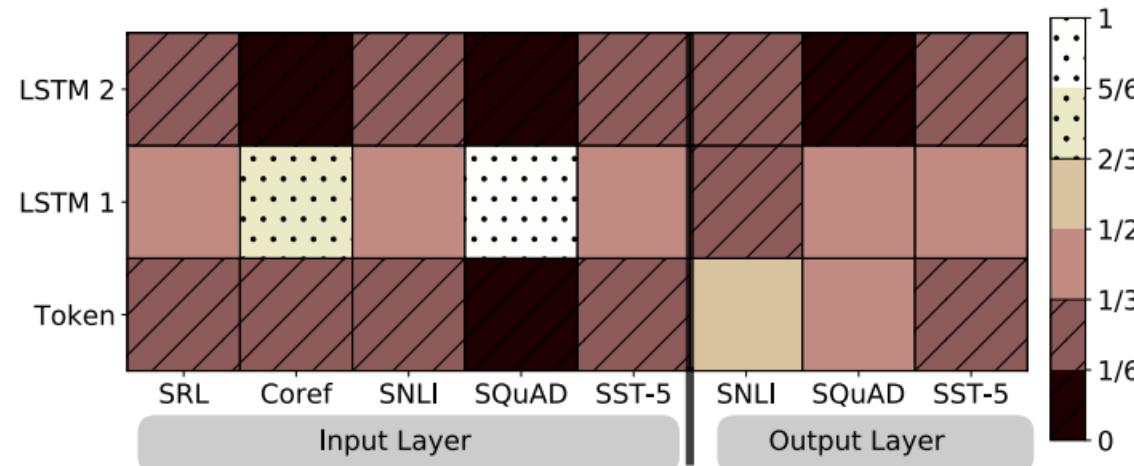
# ELMo (Embedding from Language Models)

- 문맥을 반영하는 단어 벡터를 이용하면 성능 향상이 됩니다.

TASK	PREVIOUS SOTA		OUR BASELINE	ELMO + BASELINE	INCREASE (ABSOLUTE/ RELATIVE)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

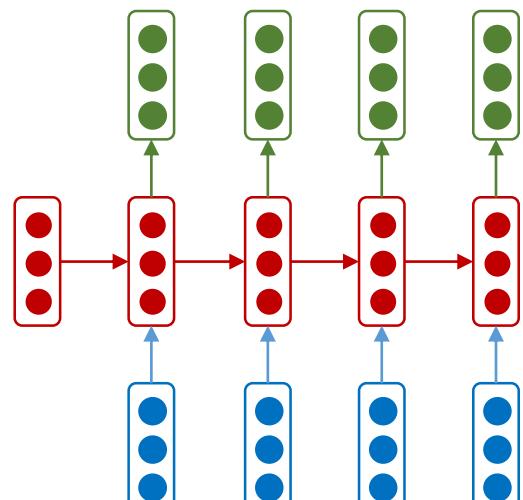
# ELMo (Embedding from Language Models)

- Tasks마다 유용한 정보가 다릅니다.
  - Semantic 한 작업에서는 higher layers 가, 품사 판별같은 작업에서는 lower layers 가 도움이 됩니다.

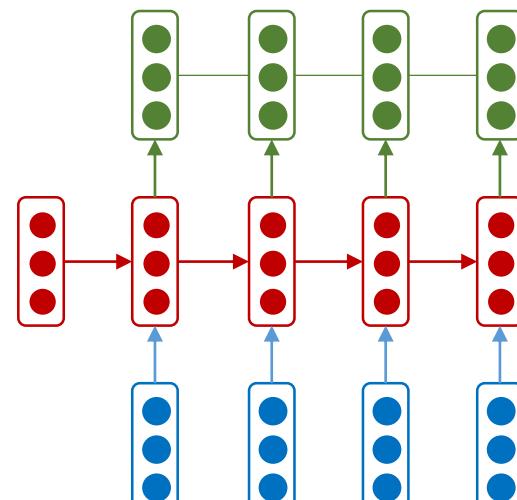


# LSTM-CRF

- RNN 은 각 output  $y_i$  가 서로 독립적으로 생성됩니다.
  - 품사 판별과 같은 작업에서는  $y_{i-1}$  과  $y_i$  간에 상관성이 존재합니다.
  - LSTM-CRF 는  $y_{i-1}$  가  $y_i$  를 생성하는데 영향을 주도록 모델링합니다.



LSTM (RNN) structure



LSTM CRF structure

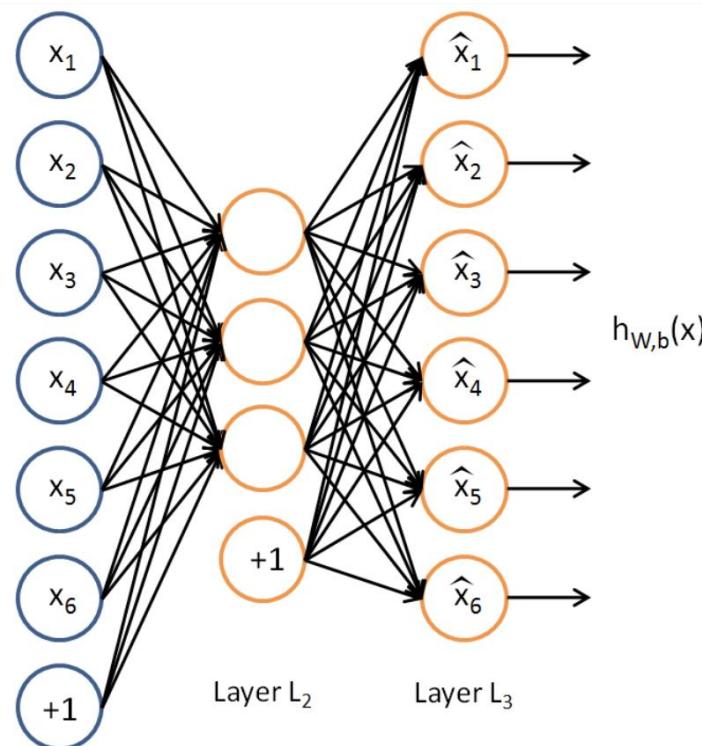
## LSTM-CRF

---

- LSTM-CRF 는  $f(y_{1:n} | x_{1:n})$  에 transition score 를 더하여 학습합니다.
  - LSTM :  $\text{maximize } \sum_{t=1}^T f_\theta(y_t, x)$
  - LSTM-CRF :  $\text{maximize } \sum_{t=1}^T [A(y_{t-1}, y_t) f_\theta(y_t, x)]$   
where  $A(y_{t-1}, y_t)$  : transition score
- Transition score 는  $y_{t-1}$  다음에  $y_t$  로 태깅할 점수입니다.

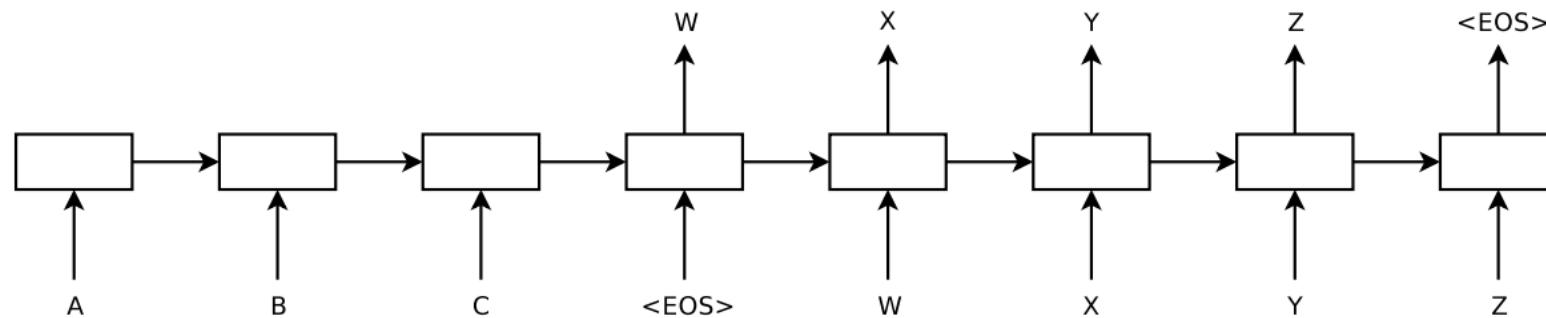
# Encoder – decoder

- Autoencoder 는 벡터 형식의 input 에 대한 encoder – decoder 입니다.



# Sequence to sequence

- Seq2seq 은 두 개의 RNN 을 이용하여 sequence 를 encoding / decoding 하는 모델 구조 입니다.



Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

# Sequence to sequence

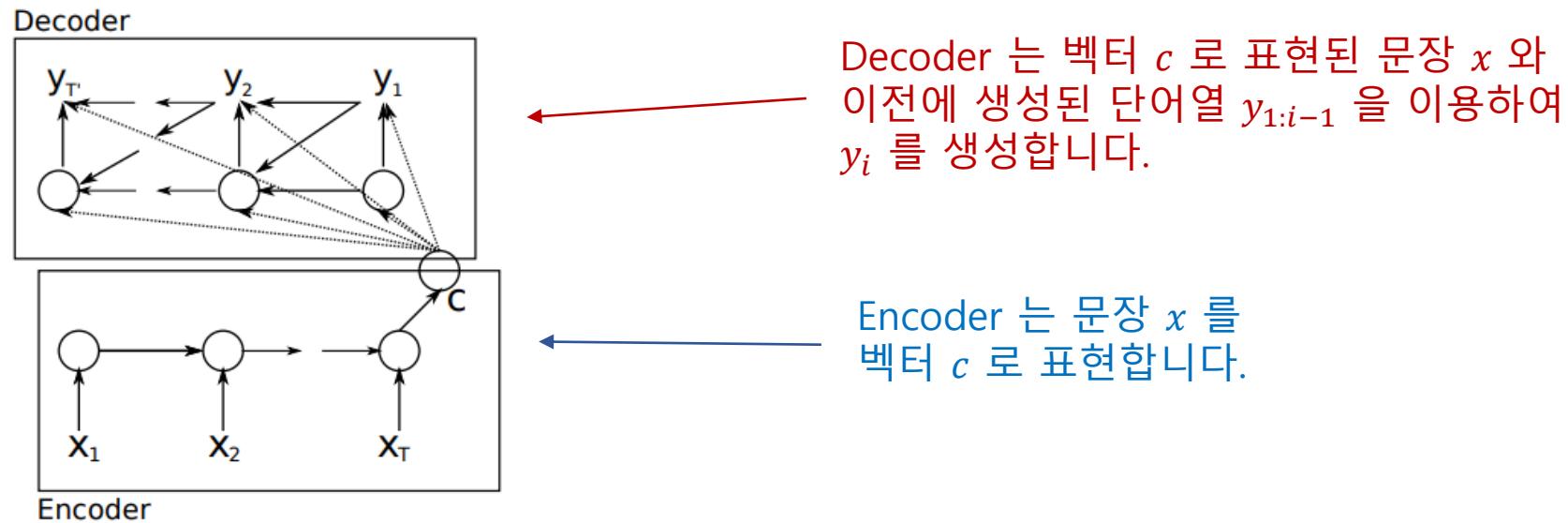
---

- Seq2seq 은 주어진  $x$  에 대하여  $y$  가 복원될 확률을 최대화 합니다.
  - $\max_{\theta} \frac{1}{N} \sum_{n=1 \text{ to } N} \log p_{\theta}(y_n | x_n)$
  - $kor \rightarrow eng$  로의 번역의 경우에는  $kor(x)$  의 벡터가  $eng(y)$  의 context 로 고정되어야 합니다.

# Sequence to sequence

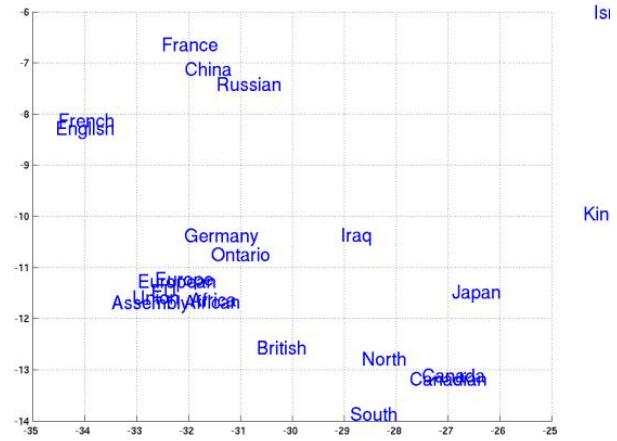
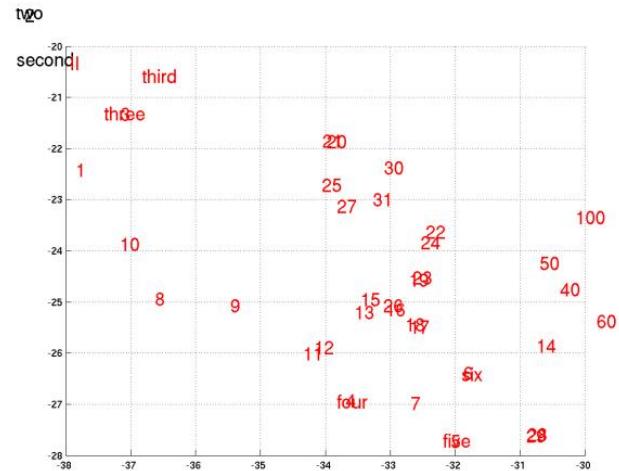
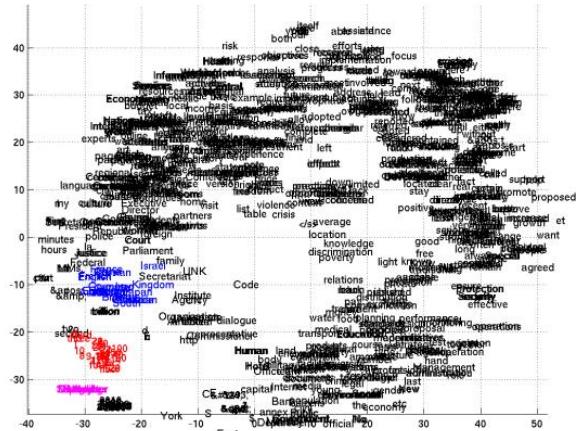
- Encoder 의 output 은 context 역할을 합니다.

$$p(y | x) = \prod_i p(y_i | y_{1:i-1}, x) = \prod_i p(y_i | y_{1:i-1}, c)$$



An illustration of the proposed RNN Encoder-Decod

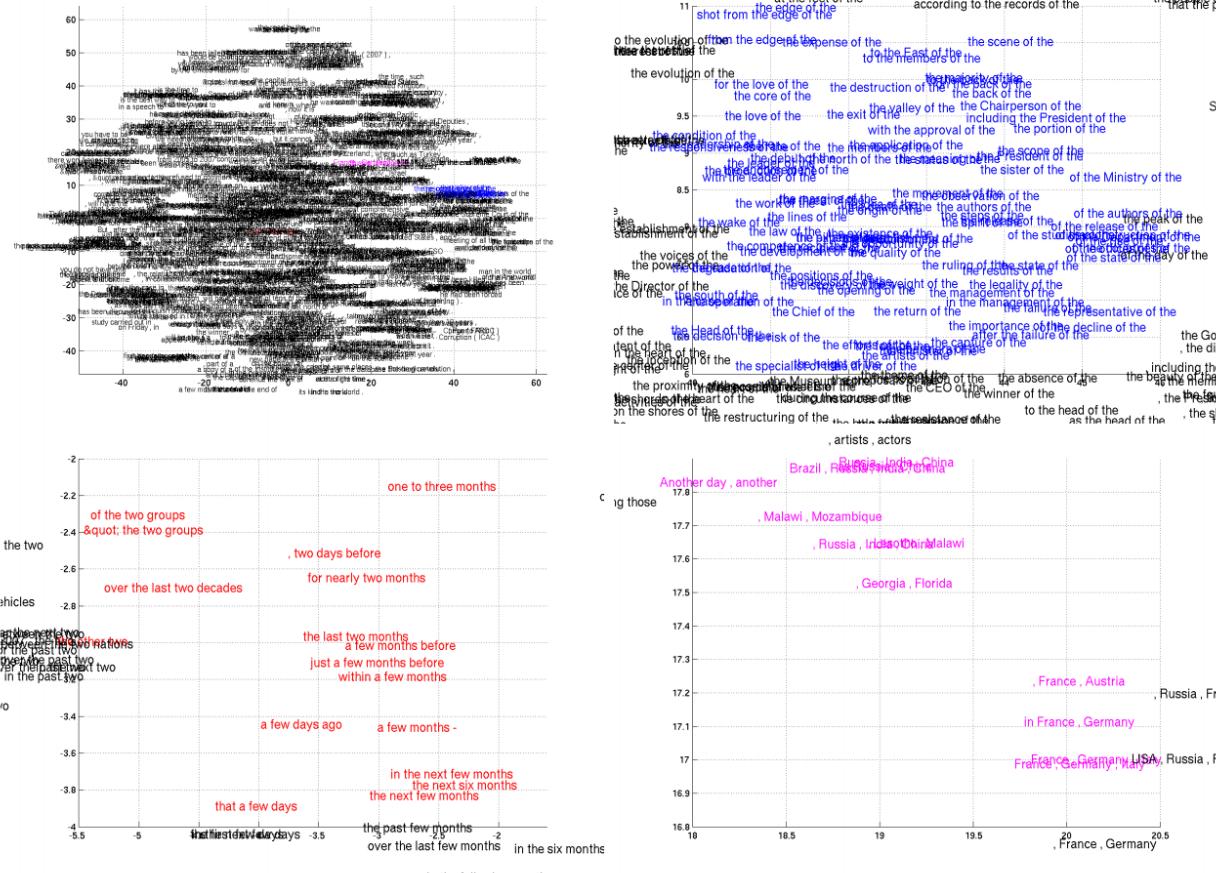
# Sequence to sequence



2-D embedding of the learned **word representation**. The top left one shows the full embedding space, while the other three figures show the zoomed-in view of specific regions (color-coded)

- Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

# Sequence to sequence



2-D embedding of the learned **phrase representation**. The top left one shows the full representation space (1000 randomly selected points), while the other three figures show the zoomed-in view of specific regions (color-coded)

- Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

# Attention mechanism

# Sequence to sequence with Attention

---

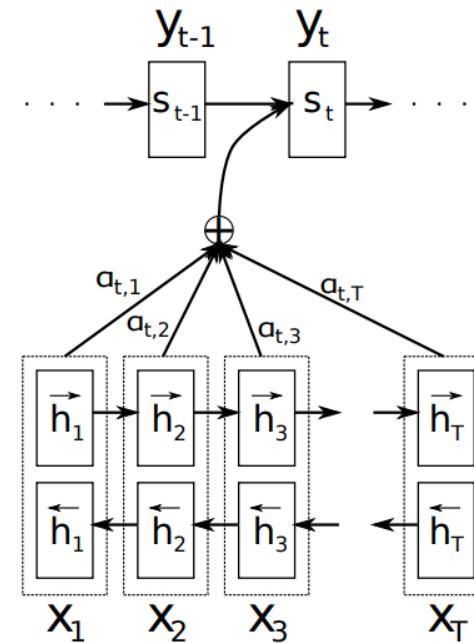
A potential issue with this encoder–decoder approach is that a neural network needs to be able to compress **all the necessary information of a source sentence into a fixed-length vector.**

**Instead**, it encodes the **input sentence into a sequence of vectors** and **chooses** a subset of these vectors adaptively while decoding the translation. This frees a neural translation model from having to squash all the information of a source sentence, regardless of its length, into a fixed-length vector.

Neural machine translation by jointly learning to align and translate (Bahdanau, D., Cho, K., & Bengio, Y. 2014).

# Sequence to sequence with Attention

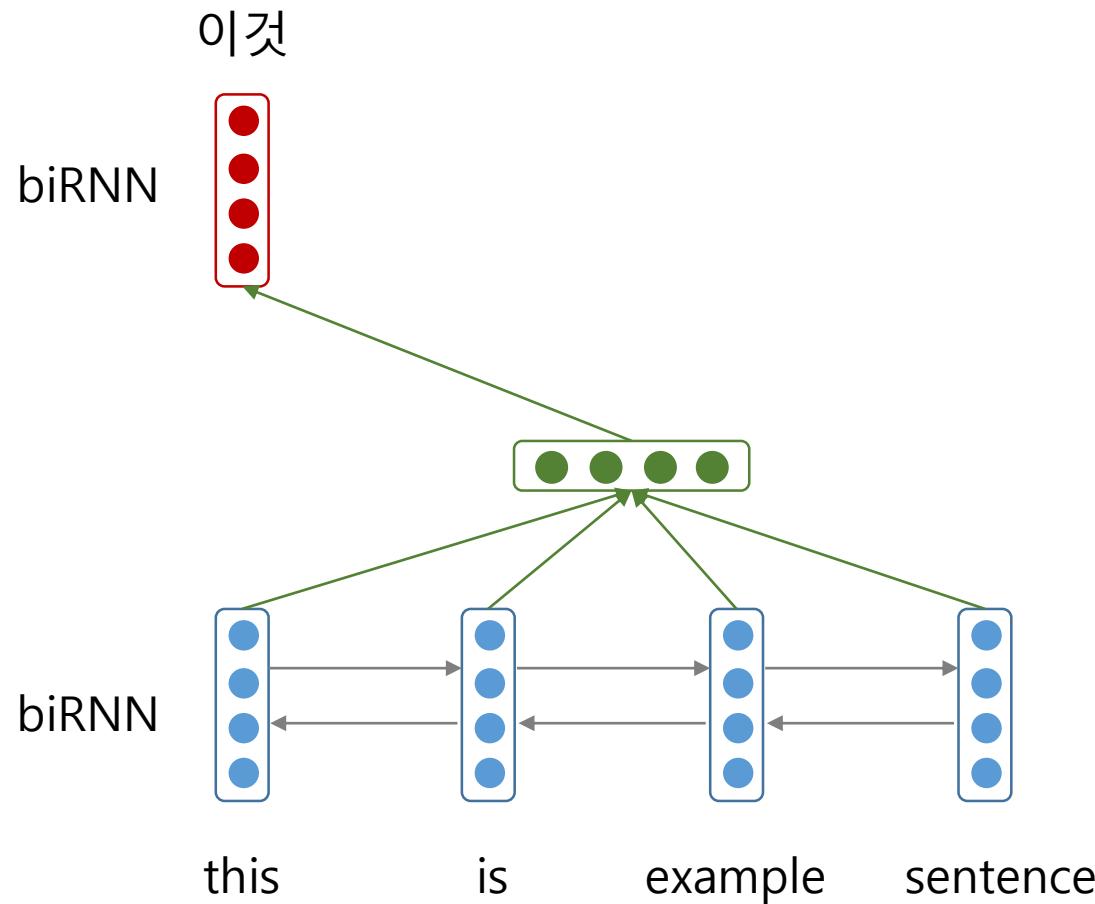
- seq2seq 는 고정된 context 를 이용합니다.
  - $p(y | x) = \prod_i p(y_i | y_{1:i-1}, c)$
- seq2seq + attention 에서는 각  $y_i$  마다 다른 context 를 이용합니다.
  - 출력될 단어마다 서로 다른 context 를 이용할 수 있습니다.
  - $p(y | x) = \prod_i p(y_i | c_t, x)$



The graphical illustration of the proposed model trying to generate the  $t$ -th target word  $y_t$  given a source sentence  $(x_1, x_2, \dots, x_T)$

# Sequence to sequence with Attention

without attention



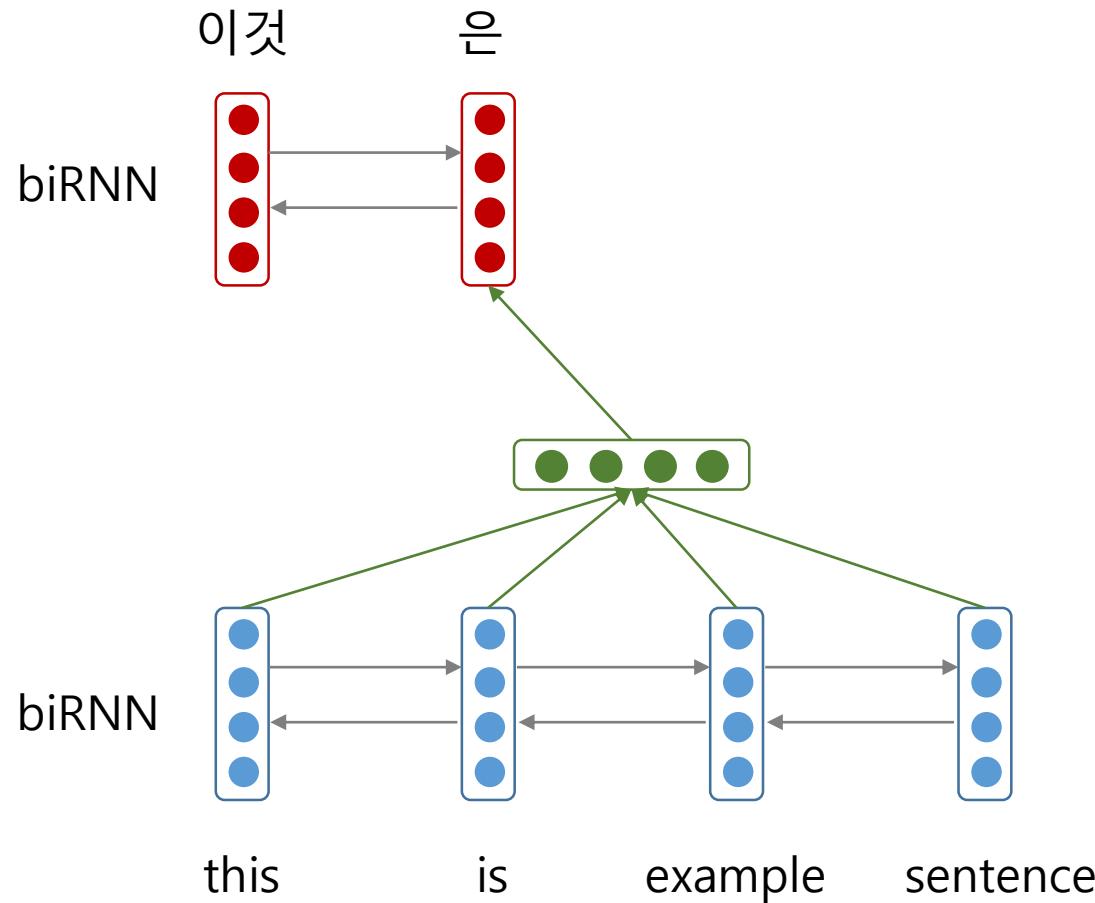
fixed context vector

$$p(y_i | y_{1:i-1}, x) = g(y_{i-1}, s_i, \mathbf{c})$$

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

# Sequence to sequence with Attention

without attention



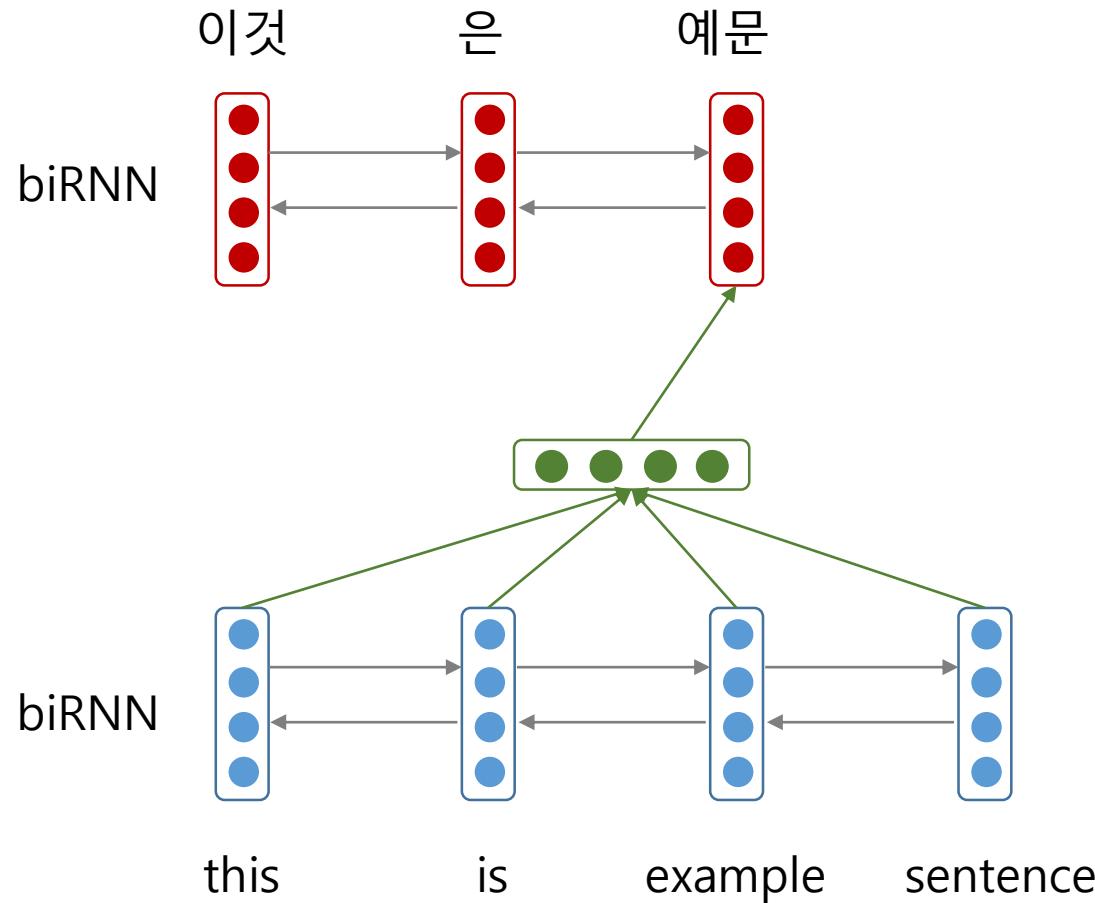
fixed context vector

$$p(y_i | y_{1:i-1}, x) = g(y_{i-1}, s_i, \mathbf{c})$$

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

# Sequence to sequence with Attention

without attention



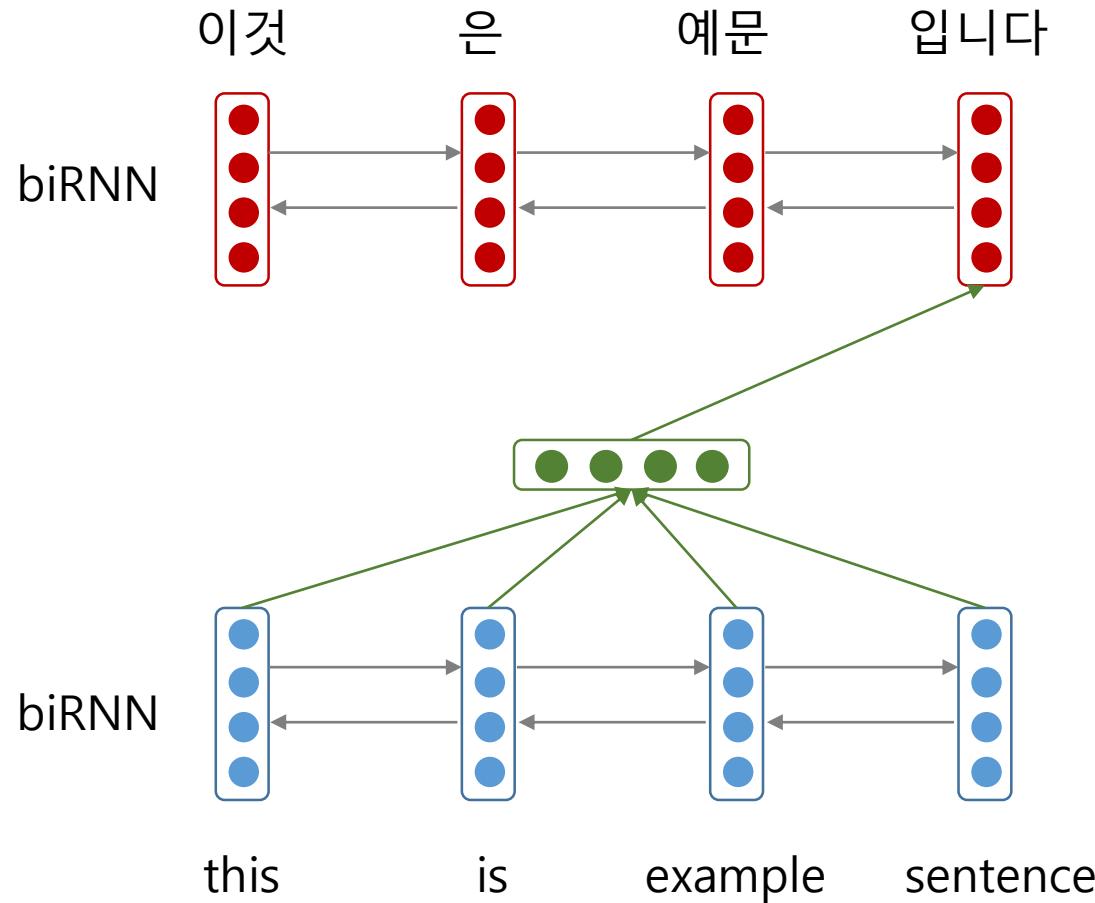
fixed context vector

$$p(y_i | y_{1:i-1}, x) = g(y_{i-1}, s_i, \mathbf{c})$$

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

# Sequence to sequence with Attention

without attention



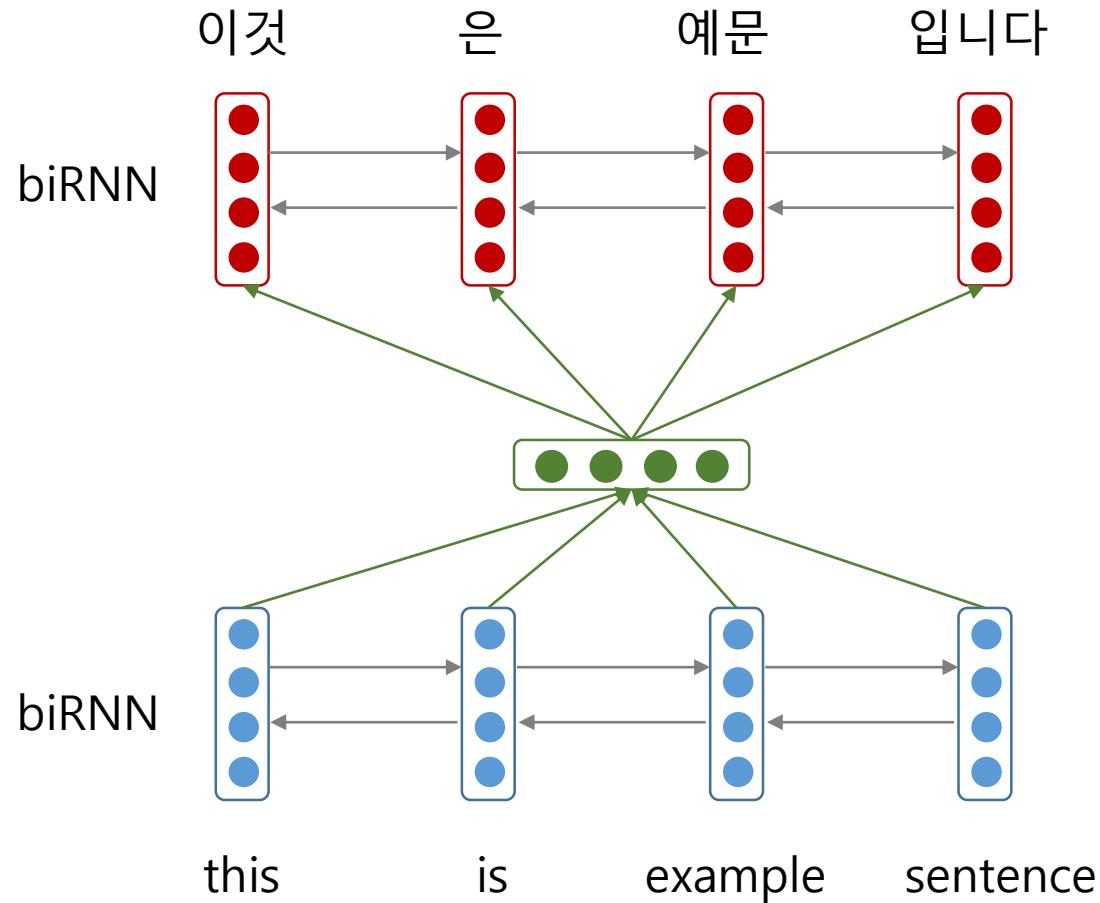
fixed context vector

$$p(y_i | y_{1:i-1}, x) = g(y_{i-1}, s_i, \mathbf{c})$$

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

# Sequence to sequence with Attention

without attention



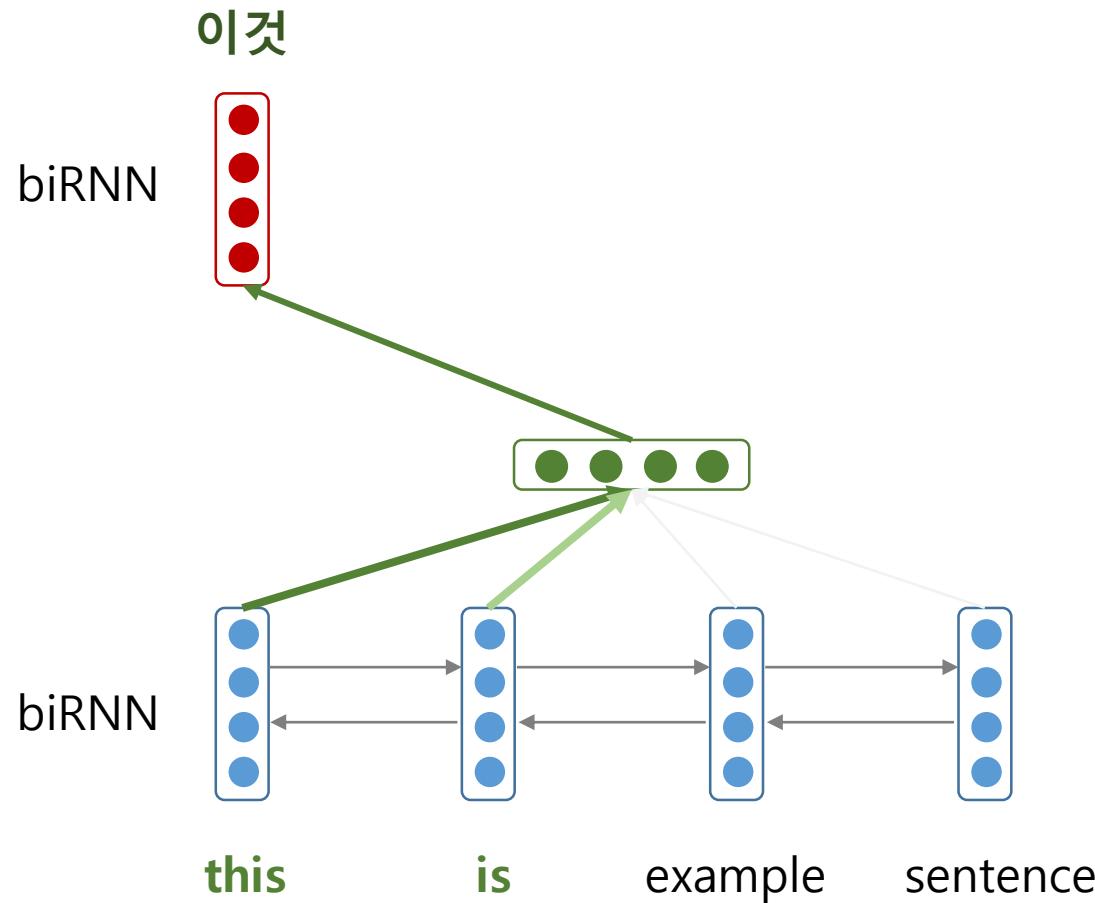
fixed context vector

$$p(y_i | y_{1:i-1}, x) = g(y_{i-1}, s_i, \mathbf{c})$$

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

# Sequence to sequence with Attention

with attention



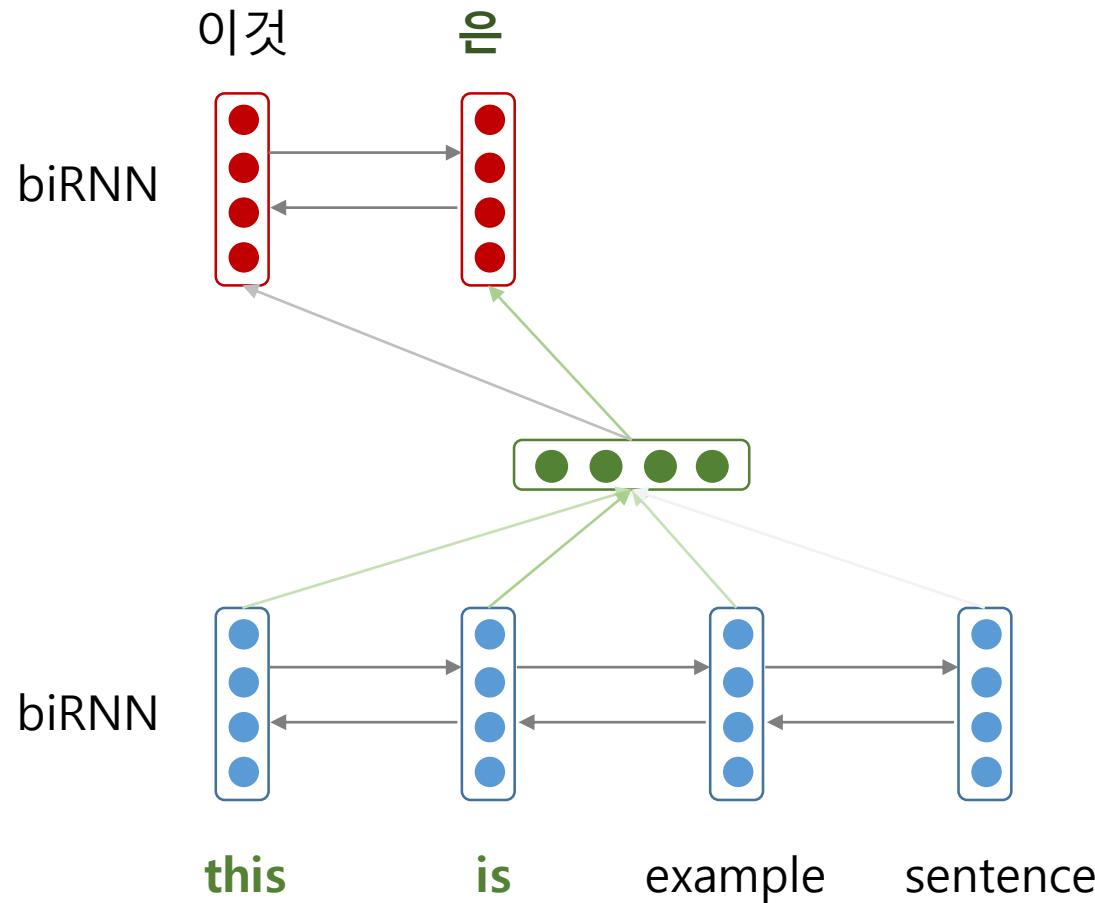
$y_i$  dependent context vector

$$p(y_i | y_{1:i-1}, x) = g(y_{i-1}, s_i, \mathbf{c}_i)$$

$$s_i = f(s_{i-1}, y_{i-1}, \mathbf{c}_i)$$

# Sequence to sequence with Attention

with attention



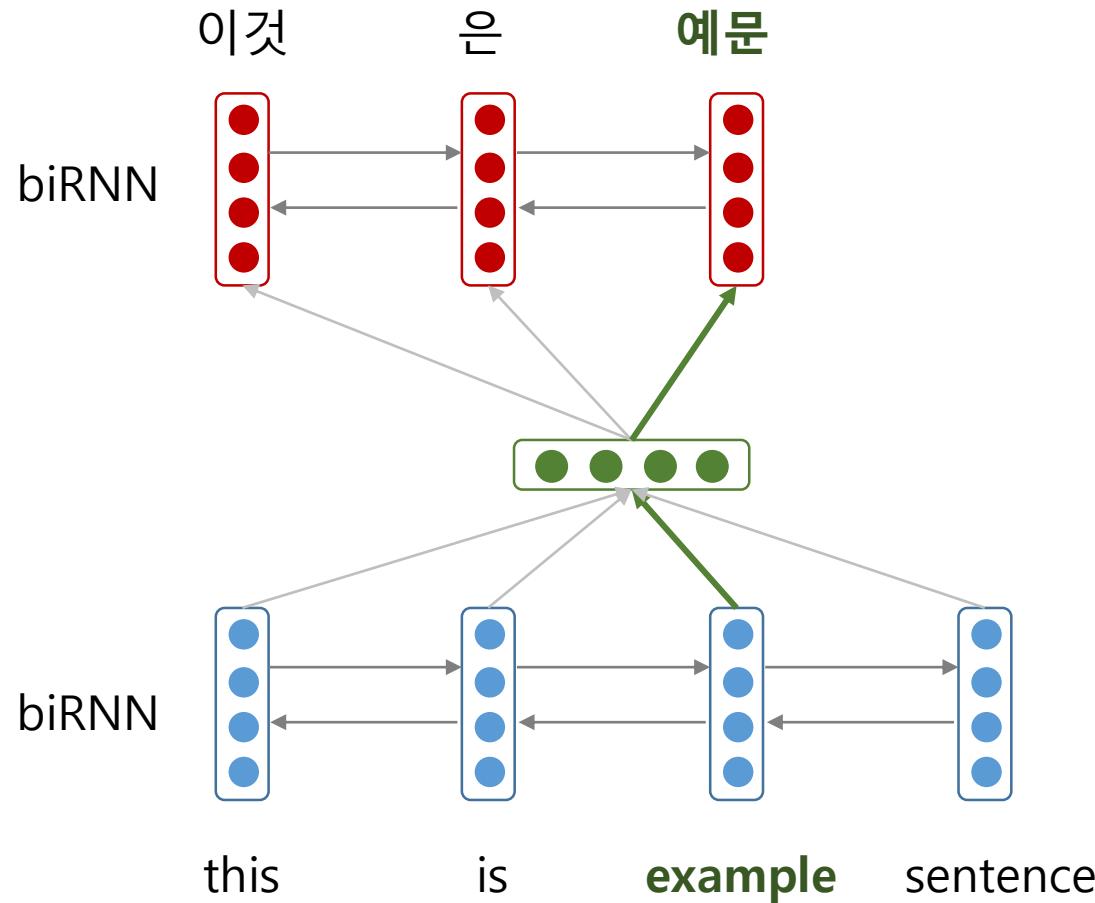
$y_i$  dependent context vector

$$p(y_i | y_{1:i-1}, x) = g(y_{i-1}, s_i, \mathbf{c}_i)$$

$$s_i = f(s_{i-1}, y_{i-1}, \mathbf{c}_i)$$

# Sequence to sequence with Attention

with attention

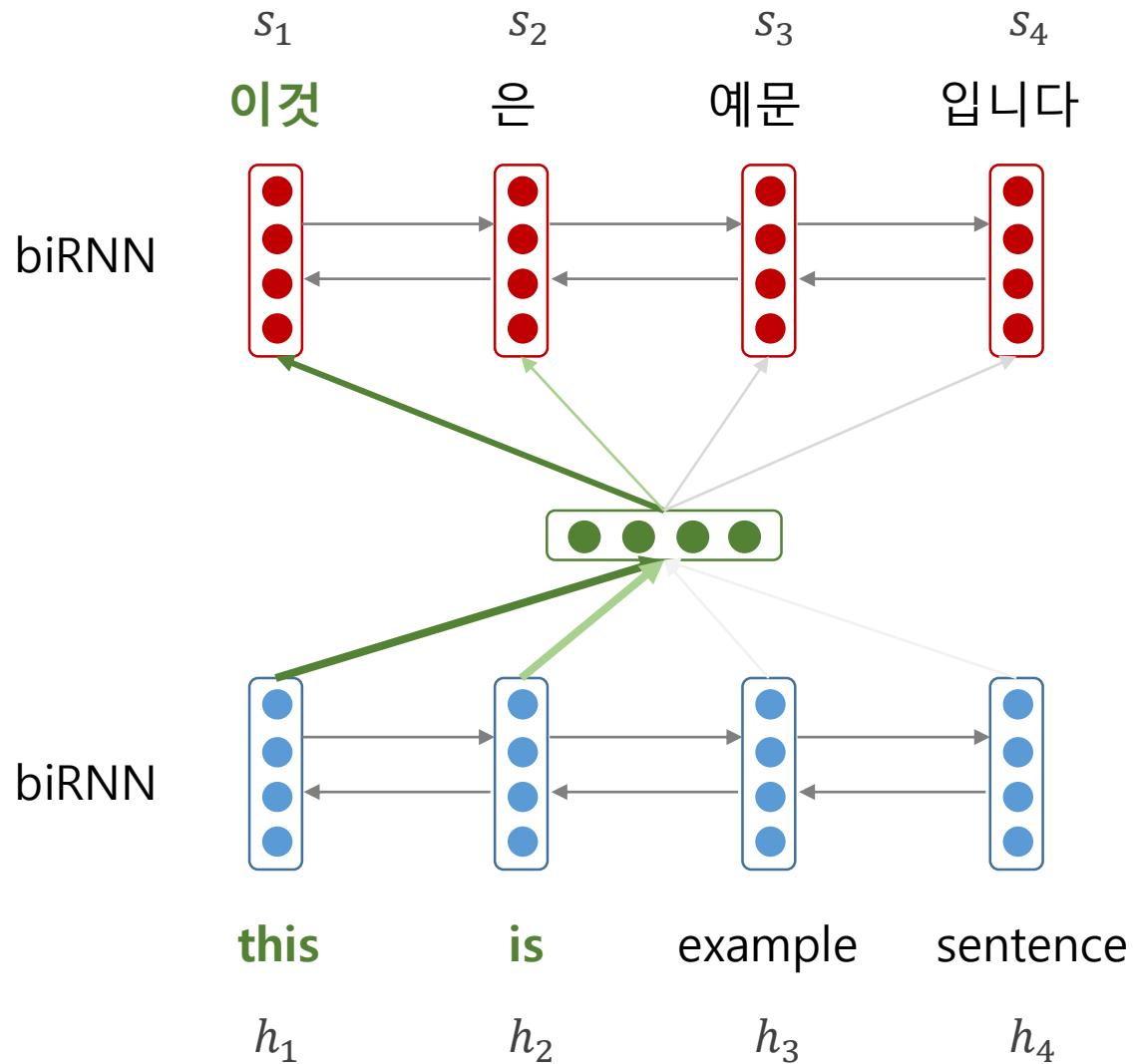


$y_i$  dependent context vector

$$p(y_i | y_{1:i-1}, x) = g(y_{i-1}, s_i, \mathbf{c}_i)$$

$$s_i = f(s_{i-1}, y_{i-1}, \mathbf{c}_i)$$

# Sequence to sequence with Attention



$$p(y_i | y_{1:i-1}, x) = g(y_{i-1}, s_i, \mathbf{c}_i)$$

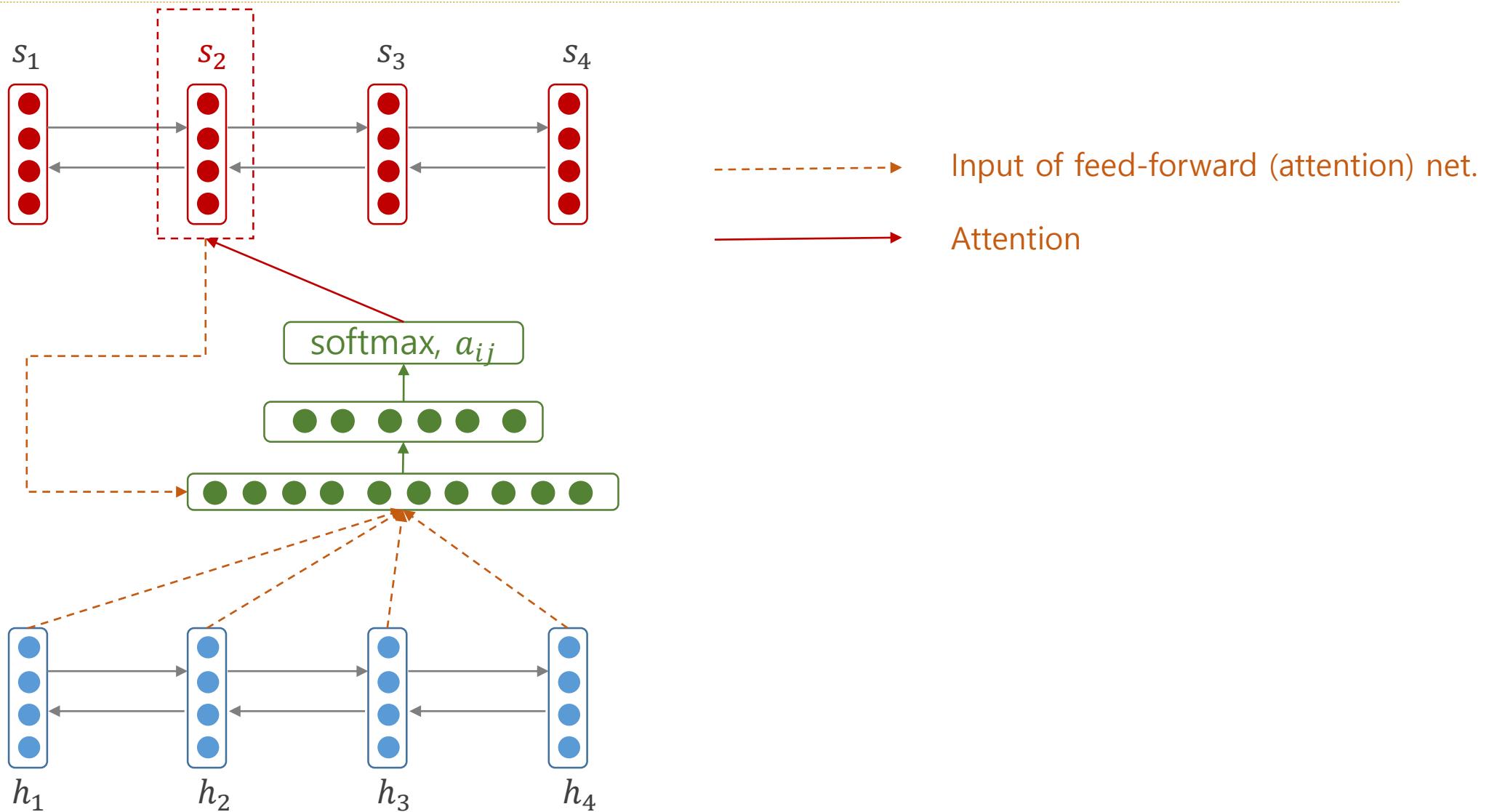
$$s_i = f(s_{i-1}, y_{i-1}, \mathbf{c}_i)$$

$$\mathbf{c}_i = \sum_{j=1 \text{ to } T_x} \alpha_{ij} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1 \text{ to } T_x} \exp(e_{ik})}$$

$$e_{ij} = a(s_{i-1}, h_j), \quad a : \text{feed-forward network}$$

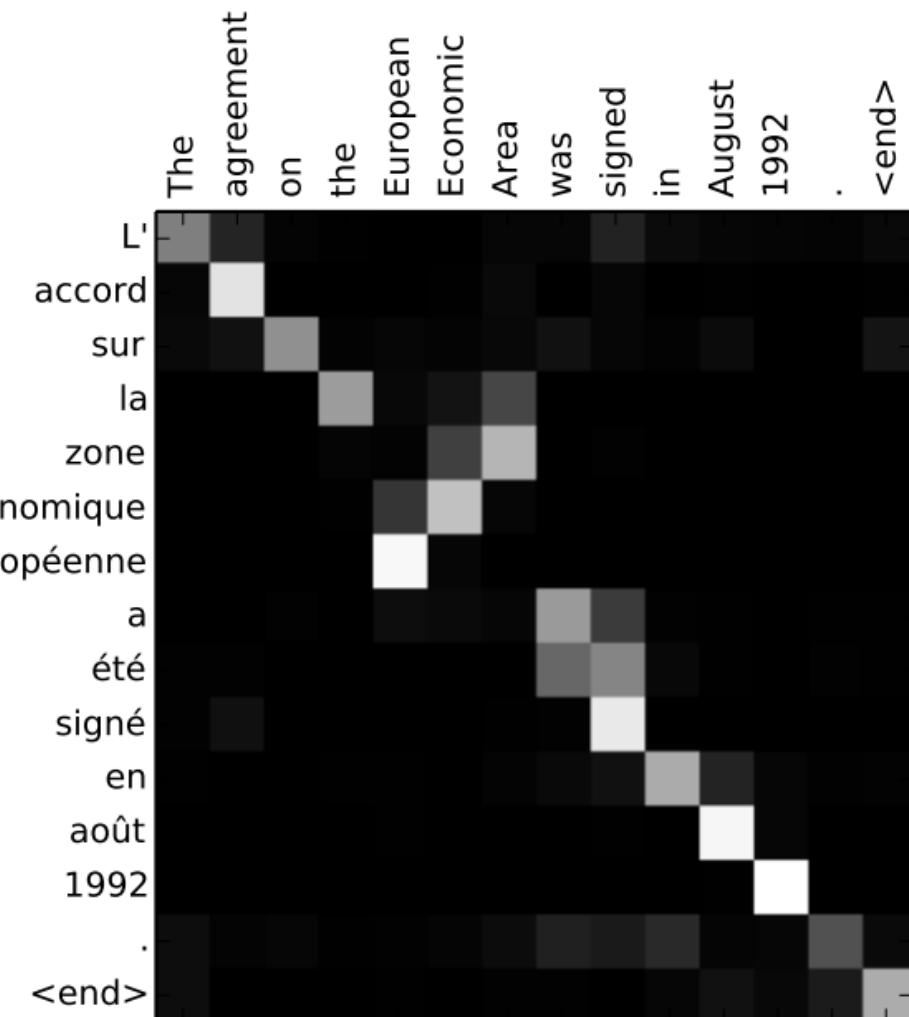
# Sequence to sequence with Attention



# Sequence to sequence with Attention

- Attention  $a_{ij}$  에는  $(x_i, y_j)$  의 상관성이 학습되어 있습니다.
- 모델의 학습된 결과를 해석할 수 있습니다.

Four sample alignments found by RNNsearch-50. The x-axis and y-axis of each plot correspond to the words in the source sentence (English) and the generated translation (French), respectively. Each pixel shows the weight  $\alpha_{ij}$  of the annotation of the  $j$ -th source word for the  $i$ -th target word



# Sequence to sequence with Attention

---

- Attention 은 key  $k$  와 query  $q$  간의 상관성을 계산하는 함수입니다.
  - $a = f(k, q)$
  - additive attention :  $f(x_i, q) = w^T \sigma(W^{(1)}x_i + W^{(2)}q)$
  - multiplicative attention :  $f(x_i, q) = \langle W^{(1)}x_i, W^{(2)}q \rangle$
  - additive 가 더 좋은 성능을 보이지만, multiplicative 가 효율적이며, multiplicative 를 이용할 경우 attention weight 가 더 sparse 합니다.

# Sequence to sequence with Attention

---

- Attention 은 key  $k$  와 query  $q$  간의 상관성을 계산하는 함수입니다.
  - $a = f(k, q)$
  - seq2seq + attention.에서도 additive attention 을 이용합니다.
    - $e_{ij} = a(s_{i-1}, h_j) = f(W^{(1)}s_{i-1} + W^{(2)}h)$
    - Additive attention 은 feed forward network 입니다.
      - $f(W^{(1)}s_{i-1} + W^{(2)}h) = f\left(\left[W^{(1)}; W^{(2)}\right]^T [s_{i-1}; h]\right)$

# Attention in image captioning

- Image captioning에서도 attention은 이용됩니다.
  - 모델의 해석에도 이용됩니다.

Figure 3. Examples of attending to the correct object (white indicates the attended regions, *underlines* indicated the corresponding word)



A woman is throwing a frisbee in a park.



A dog is standing on a hardwood floor.



A stop sign is on a road with a mountain in the background.



A little girl sitting on a bed with a teddy bear.



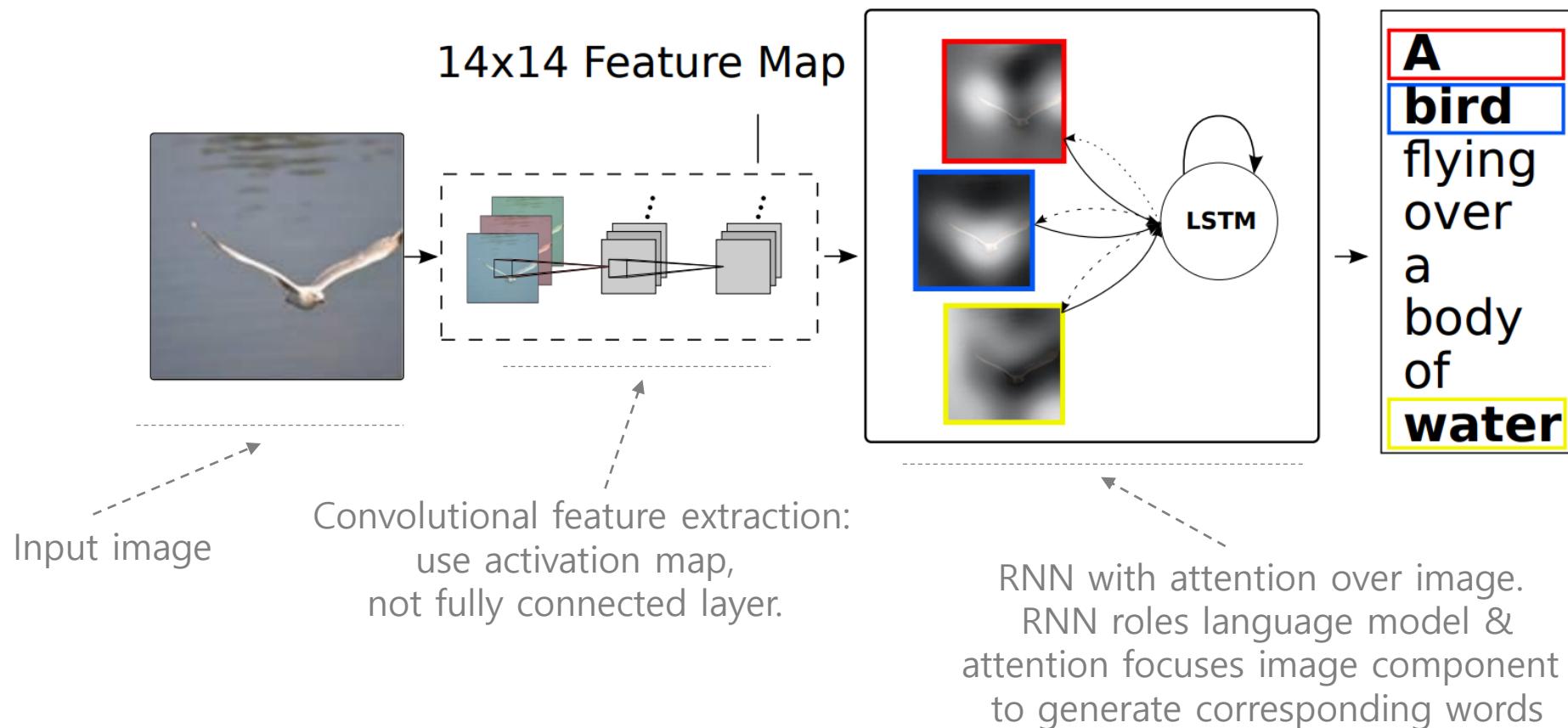
A group of people sitting on a boat in the water.



A giraffe standing in a forest with trees in the background.

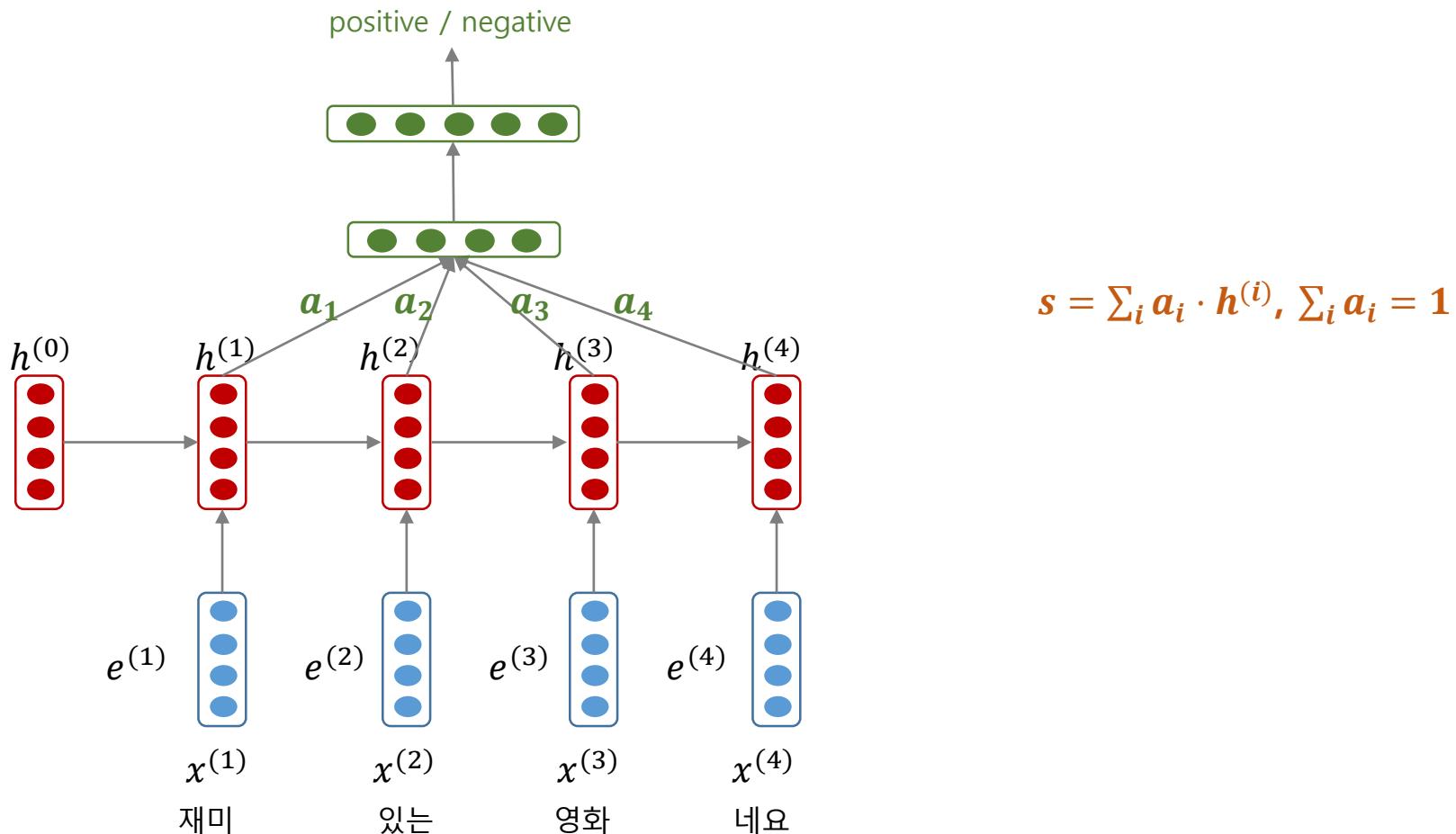
# Attention in image captioning

- Image captioning에서도 attention은 이용됩니다.



# Recurrent neural network with self attentive manner

- Attention 을 이용하여 hidden vectors 의 weight 를 다르게 부여합니다.

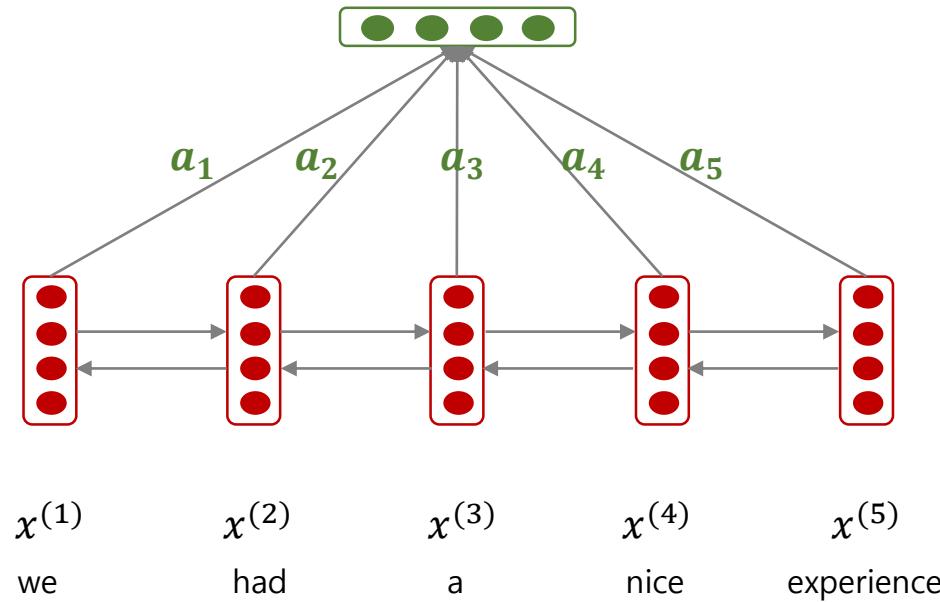


# Recurrent neural network with self attentive manner

---

- 문장 분류에 필요한 정보 (단어)는 일부입니다.
  - 마지막 hidden vector 를 문장의 repr. 로 이용하면 RNN 이 정보를 기억하는 모든 작업을 해야합니다.
  - 중요한 정보들을 선택하여 그들의 선형 조합으로 문장을 표현합니다.
- BiLSTM 의 hidden vectors 의 가중 평균으로 문장의 repr. 을 학습합니다.
  - Hidden vector 는 단어 수준이 아닌 문맥 수준의 정보를 표현합니다.
  - (attention) weights 는 문장의 중요 성분 해석을 도와줍니다.

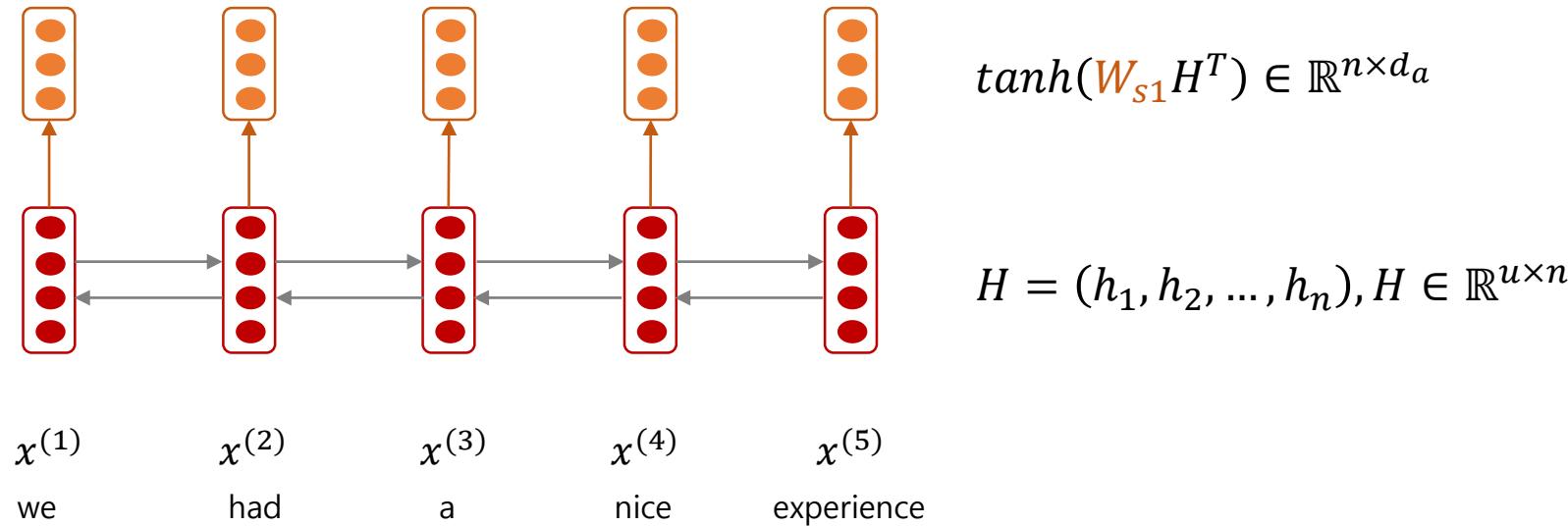
# Recurrent neural network with self attentive manner



- Lin et al., (2017) 은 BiLSTM 을 이용하여 단어 개수가  $n$  인 문장에 대한  $(n \times h)$  의 벡터를 학습합니다.  
$$H = (h_1, h_2, \dots, h_n)$$
- Hidden vectors 에 대한 attention 을 이용하여  $h$  차원의 문장 벡터를 학습합니다.
- $$a = softmax(w_{s2} \cdot \tanh(W_{s1} H^T))$$

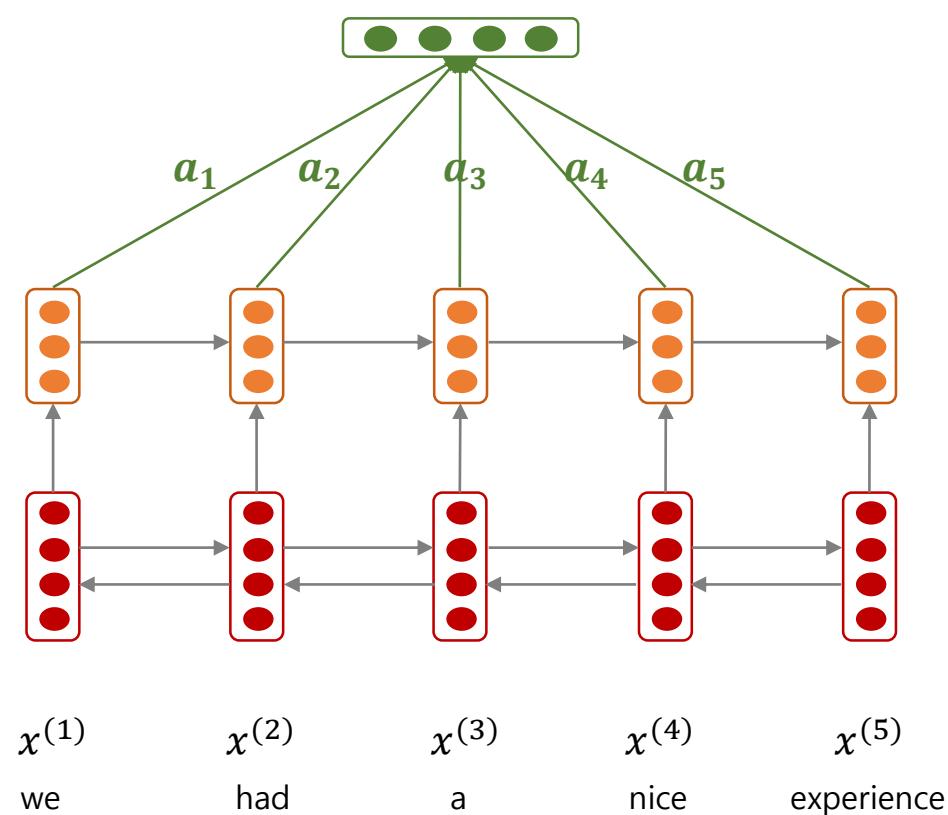
# Recurrent neural network with self attentive manner

- 각 hidden vector 는  $W_{s1}$  이 곱해져 공간 변환이 됩니다.  
(논문에서는  $h = 300 \times 2, d_a = 350$ )
- Hidden space 가 attention space 로 변합니다.
- Hyper-tangent 에 의하여  $[-1, 1]$  로 scaling 됩니다.



# Recurrent neural network with self attentive manner

- Attention space에서의 repr.에  $w_{s2}$ 를 곱하여 각 시점의 attention weight를 계산합니다.



$$s = \sum_i a_i \cdot h_i, \quad s \in \mathbb{R}^h$$

$$a = \text{softmax}(\tanh(W_{s1}H^T) \cdot w_{s2}) \in \mathbb{R}^{n \times 1}$$

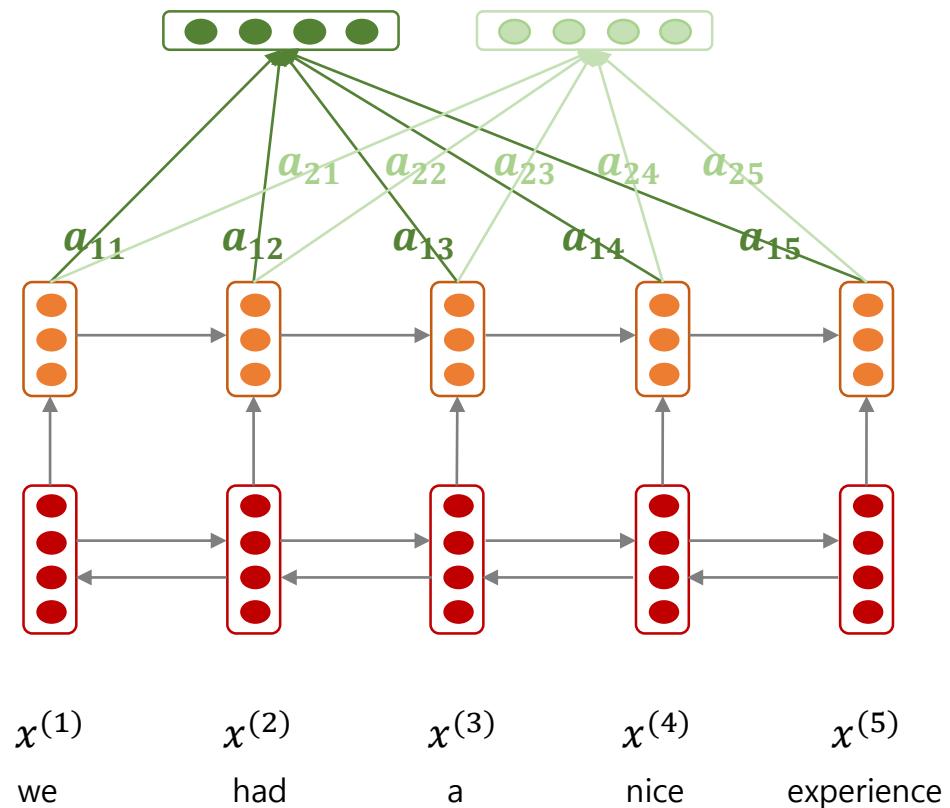
where  $w_{s2} \in \mathbb{R}^{d_a}$

$$\tanh(W_{s1}H^T) \in \mathbb{R}^{n \times d_a}$$

$$H = (h_1, h_2, \dots, h_n), H \in \mathbb{R}^{h \times n}$$

# Recurrent neural network with self attentive manner

- $w_{s2}$  는 attention aspect 를 의미합니다.
- 다양한 관점으로 문장에서 중요한 부분을 선택합니다.
- $d_a$  차원의 벡터를  $r$  개 이용합니다. (논문  $r = 30$ )



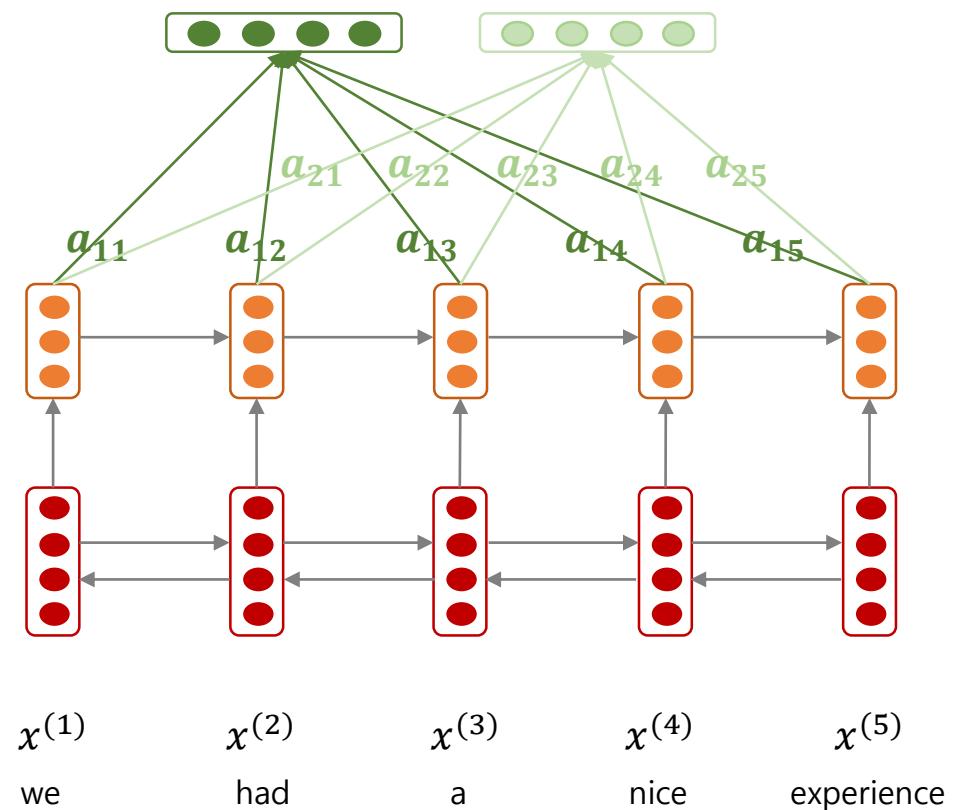
$$A = \text{softmax}(\tanh(W_{s1}H^T) \cdot W_{s2}) \in \mathbb{R}^{n \times r}$$

where  $W_{s2} \in \mathbb{R}^{d_a \times r}$

$$s_j = \sum_i a_{ji} \cdot h_i, \quad s_j \in \mathbb{R}^h$$

# Recurrent neural network with self attentive manner

- $A \in \mathbb{R}^{n \times r}$  에는 중복된 관점이 존재할 수 있습니다.
- $W_{s2}$  의 각 column 이 서로 다르길 원합니다
  - 각 column 은 softmax 이므로 normal vector 입니다.
  - $\|AA^T - I\|_F^2$  를 regularization term 으로 이용합니다.

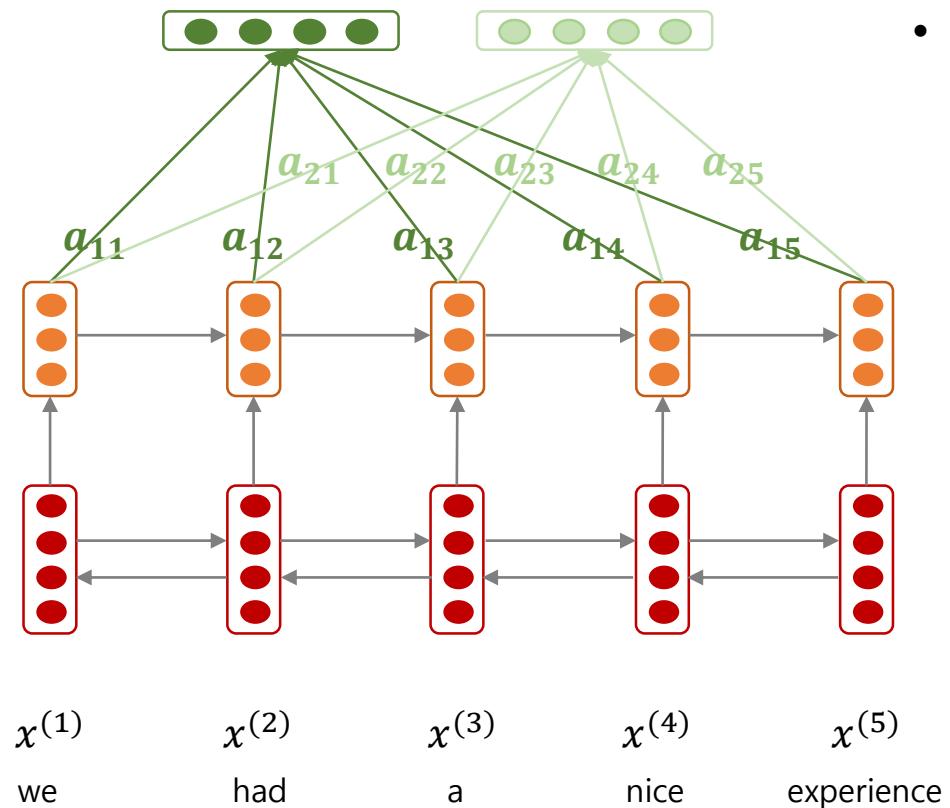


$$A = \text{softmax}(\tanh(W_{s1}H^T) \cdot W_{s2}) \in \mathbb{R}^{n \times r}$$

$$s_j = \sum_i a_{ji} \cdot h_i, \quad s_j \in \mathbb{R}^h$$

# Recurrent neural network with self attentive manner

- $A$  를 만드는 함수는 2 layers feed-forward network 입니다.
- FC 를 이용하여 다양한 관점으로 문장에서 필요한 정보를 선택하는 attentions 을 학습합니다.
- Attention weight matrix 를 이용하여 문장에서의 중요 성분을 시각적으로 확인할 수 있습니다.



$$A = \text{softmax}(\tanh(W_{s1}H^T) \cdot W_{s2}) \in \mathbb{R}^{n \times r}$$

# Recurrent neural network with self attentive manner

---

- i really enjoy Ashley and Ami salon she do a great job be friendly and professional I usually get my hair do when I go to MI because of the quality of the highlight and the price the price be very affordable the highlight fantastic thank Ashley i highly recommend you and ill be back
- love this place it really be my favorite restaurant in Charlotte they use charcoal for their grill and you can taste it steak with chimichurri be always perfect Fried yucca cilantro rice pork sandwich and the good tres lech I have had.The desert be all incredible if you do not like it you be a mutant if you will like diabeetus try the Inca Cola
- this place be so much fun I have never go at night because it seem a little too busy for my taste but that just prove how great this restaurant be they have amazing food and the staff definitely remember us every time we be in town I love when a waitress or waiter come over and ask if you want the cab or the Pinot even when there be a rush and the staff be run around like crazy whenever I grab someone they instantly smile acknowlegde us the food be also killer I love when everyone know the special and can tell you they have try them all and what they pair well with this be a first last stop whenever we be in Charlotte and I highly recommend them
- great food and good service .... what else can you ask for everything that I have ever try here have be great
- first off I hardly remember waiter name because its rare you have an unforgettable experience the day I go I be celebrate my birthday and let me say I leave feel extra special our waiter be the best ever Carlos and the staff as well I be with a party of 4 and we order the potato salad shrimp cocktail lobster amongst other thing and boy be the food great the lobster be the good lobster I have ever eat if you eat a dessert I will recommend the cheese cake that be also the good I have ever have it be expensive but so worth every penny I will definitely be back there go again for the second time in a week and it be even good ..... this place be amazing

Reviews of Yelp 5 starred

# Hierarchical Attention Network (HAN)

---

- 문서는 문장으로, 문장은 단어로 구성되어 있습니다.
  - 문장 분류에 모든 단어가 동일한 중요도를 지니지 않듯이, 문서 분류에도 모든 문장이 동일한 중요도를 지니지 않습니다.
- Self-attention 을 통해 문서 분류에 필요한 문장과 단어만을 선택적으로 이용하여 문서의 representation 를 학습합니다.

# Hierarchical Attention Network (HAN)

---

First, since documents have a hierarchical structure, we likewise construct a document representation by building representation of sentences and then aggregating these into a document representation.

Second, different words and sentences in a documents are differentially informative.

Third, the importance of words and sentences are highly context dependent, i.e. the same word or sentence may be differentially important in different context.

# Hierarchical Attention Network (HAN)

- Encoder 에는 BiGRU, Attention 에는 1 layer 를 이용합니다.

- Word encoder for sentence  $s_i$

$$\overrightarrow{h_{it}} = \overrightarrow{GRU}(x_{it}), \overleftarrow{h_{it}} = \overleftarrow{GRU}(x_{it})$$

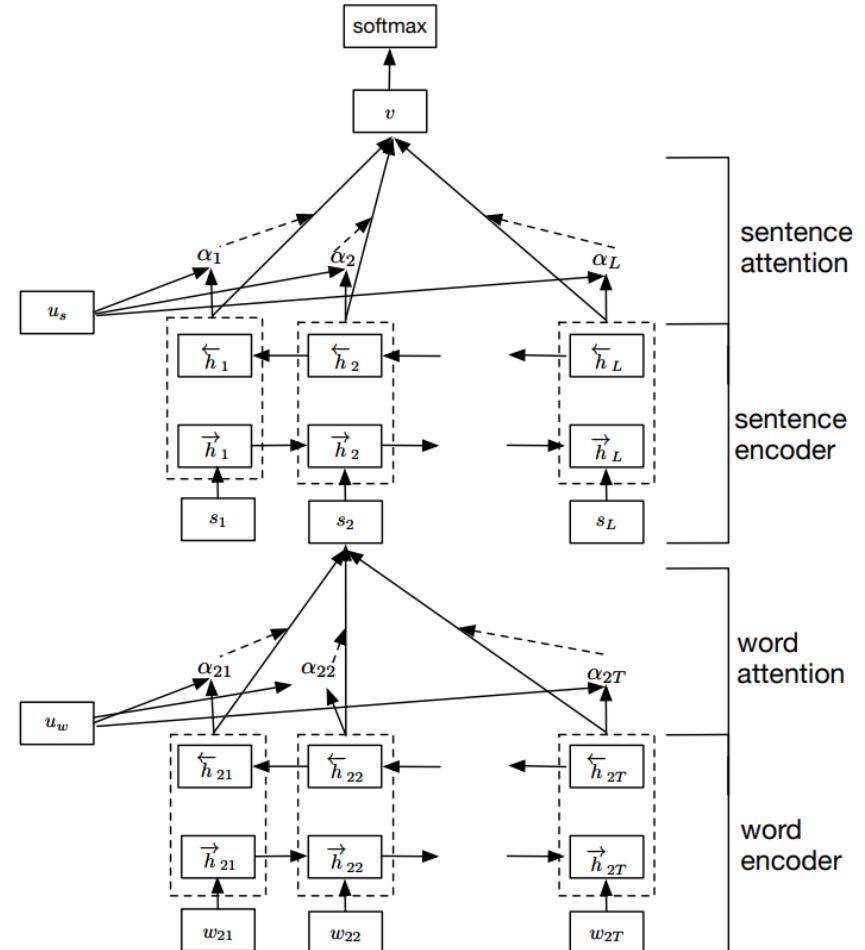
- Word attention

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

$$a_{it} = \frac{\exp(u_{it}^T u_w)}{\sum_t \exp(u_{it}^T u_w)}$$

$$s_i = \sum_t a_{it} h_{it}$$

$u_w$  : word-level context vector. randomly initialized & jointly learned.



# Hierarchical Attention Network (HAN)

- Sentence  $s_i$  를 마치 단어처럼 이용합니다. 문장에 흐름이 있습니다.
- Sentence encoder for sentence  $i$

$$\vec{h}_i = \overrightarrow{GRU}(s_i), \overleftarrow{h}_i = \overleftarrow{GRU}(s_i)$$

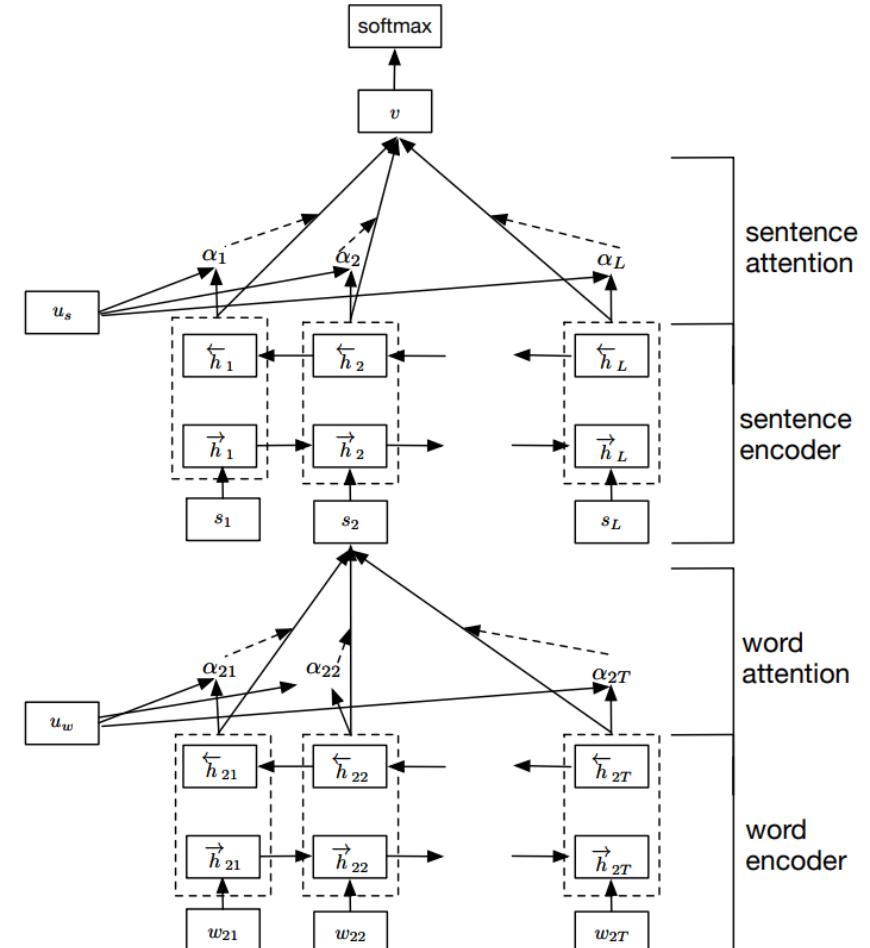
- Word attention

$$u_i = \tanh(W_s h_i + b_w)$$

$$a_i = \frac{\exp(u_i^T u_s)}{\sum_i \exp(u_i^T u_s)}$$

$$v = \sum_t a_i h_i$$

$u_w$  : sent-level context vector. randomly initialized & jointly learned.



# Hierarchical Attention Network (HAN)

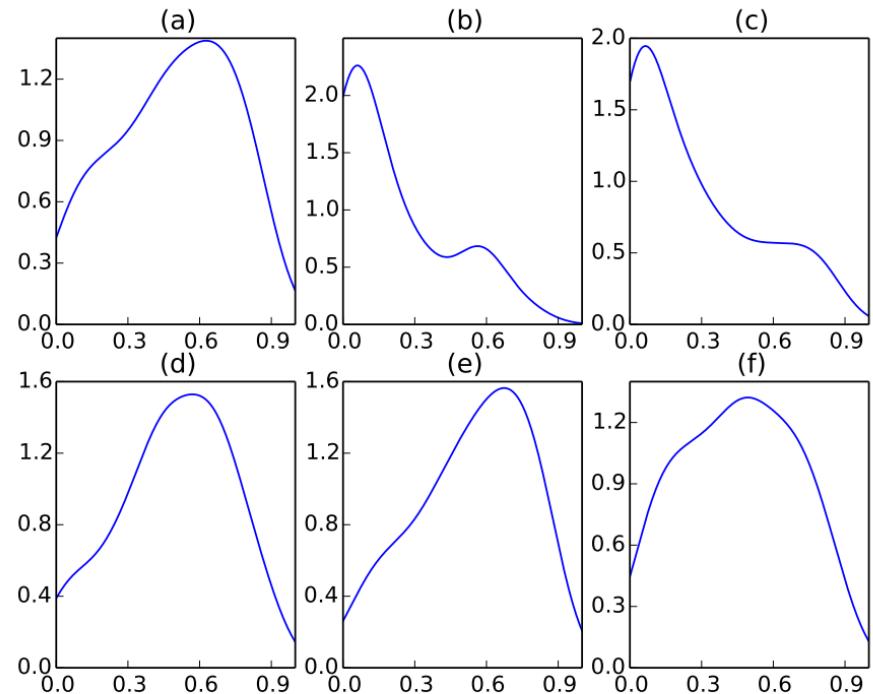
	Methods	Yelp'13	Yelp'14	Yelp'15	IMDB	Yahoo Answer	Amazon
<b>Zhang et al., 2015</b>	BoW	-	-	58.0	-	68.9	54.4
	BoW TFIDF	-	-	59.9	-	71.0	55.3
	ngrams	-	-	56.3	-	68.5	54.3
	ngrams TFIDF	-	-	54.8	-	68.5	52.4
	Bag-of-means	-	-	52.5	-	60.5	44.1
<b>Tang et al., 2015</b>	Majority	35.6	36.1	36.9	17.9	-	-
	SVM + Unigrams	58.9	60.0	61.1	39.9	-	-
	SVM + Bigrams	57.6	61.6	62.4	40.9	-	-
	SVM + TextFeatures	59.8	61.8	62.4	40.5	-	-
	SVM + AverageSG	54.3	55.7	56.8	31.9	-	-
	SVM + SSWE	53.5	54.3	55.4	26.2	-	-
<b>Zhang et al., 2015</b>	LSTM	-	-	58.2	-	70.8	59.4
	CNN-char	-	-	62.0	-	71.2	59.6
	CNN-word	-	-	60.5	-	71.2	57.6
<b>Tang et al., 2015</b>	Paragraph Vector	57.7	59.2	60.5	34.1	-	-
	CNN-word	59.7	61.0	61.5	37.6	-	-
	Conv-GRNN	63.7	65.5	66.0	42.5	-	-
	LSTM-GRNN	65.1	67.1	67.6	45.3	-	-
<b>This paper</b>	HN-AVE	67.0	69.3	69.9	47.8	75.2	62.9
	HN-MAX	66.9	69.3	70.1	48.2	75.2	62.9
	HN-ATT	<b>68.2</b>	<b>70.5</b>	<b>71.0</b>	<b>49.4</b>	<b>75.8</b>	<b>63.6</b>

# Hierarchical Attention Network (HAN)

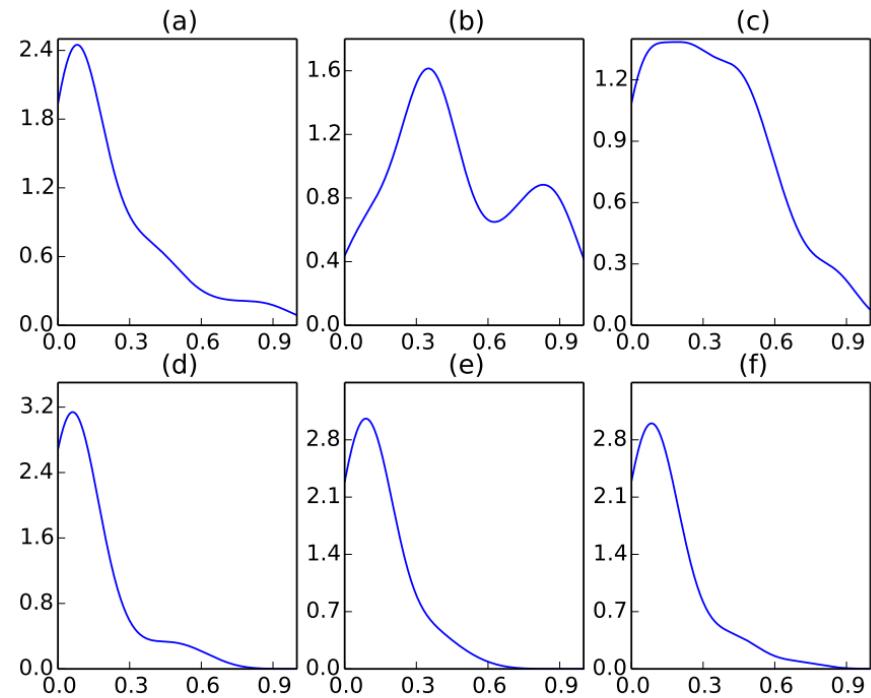
---

From Table, we can see that **neural network** based methods that **do not explore hierarchical document structure**, such as LSTM, CNN-word, CNN-char have **little advantage over traditional methods** for large scale (in terms of document size) text classification.

# Hierarchical Attention Network (HAN)



**Figure 3:** Attention weight distribution of *good*. (a) — aggregate distribution on the test split; (b)-(f) stratified for reviews with ratings 1-5 respectively. We can see that the weight distribution shifts to *higher* end as the rating goes higher.



**Figure 4:** Attention weight distribution of the word *bad*. The setup is as above: (a) contains the aggregate distribution, while (b)-(f) contain stratifications to reviews with ratings 1-5 respectively. Contrary to before, the word *bad* is considered important for poor ratings and less so for good ones.

# Hierarchical Attention Network (HAN)

GT: 4 Prediction: 4

pork belly = delicious .  
scallops ?  
i do n't .  
even .  
like .  
scallops , and these were a-m-a-z-i-n-g .  
fun and tasty cocktails .  
next time i 'm in phoenix , i will go  
back here .  
highly recommend .

GT: 0 Prediction: 0

terrible value .  
ordered pasta entree .  
. \$ 16.95 good taste but size was an  
appetizer size .  
. no salad , no bread no vegetable .  
this was .  
our and tasty cocktails .  
our second visit .  
i will not go back .

Figure 5: Documents from Yelp 2013. Label 4 means star 5, label 0 means star 1.

GT: 1 Prediction: 1

why does zebras have stripes ?  
what is the purpose or those stripes ?  
who do they serve the zebras in the  
wild life ?  
this provides camouflage - predator  
vision is such that it is usually difficult  
for them to see complex patterns

GT: 4 Prediction: 4

how do i get rid of all the old web  
searches i have on my web browser ?  
i want to clean up my web browser  
go to tools > options .  
then click " delete history " and "  
clean up temporary internet files . "

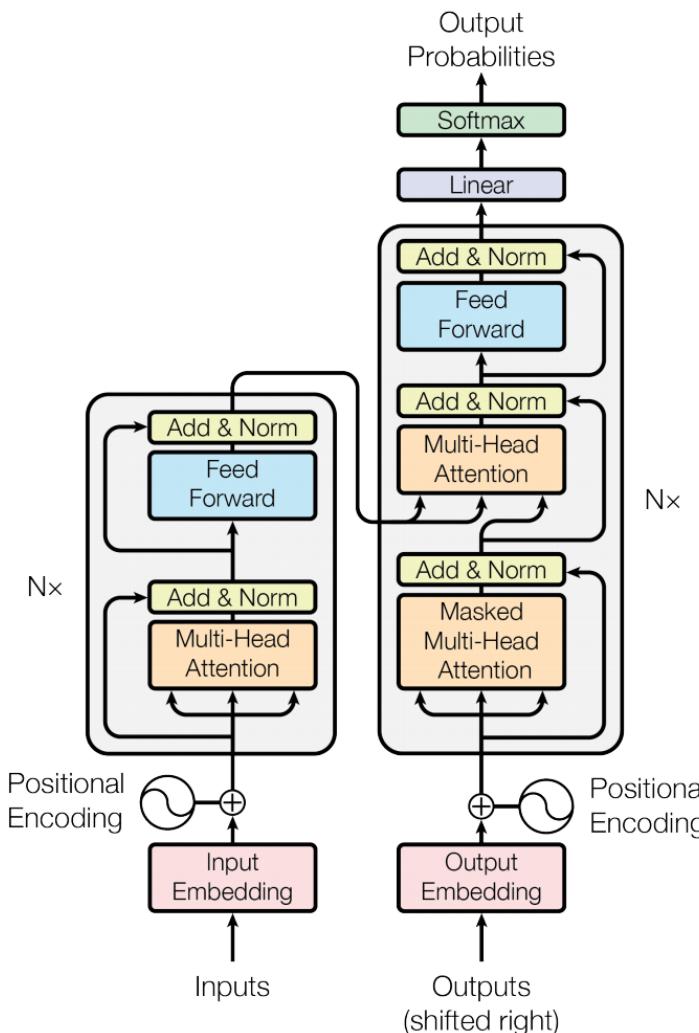
Figure 6: Documents from Yahoo Answers. Label 1 denotes Science and Mathematics and label 4 denotes Computers and Internet.

# Transformer (Attention is all you need)

---

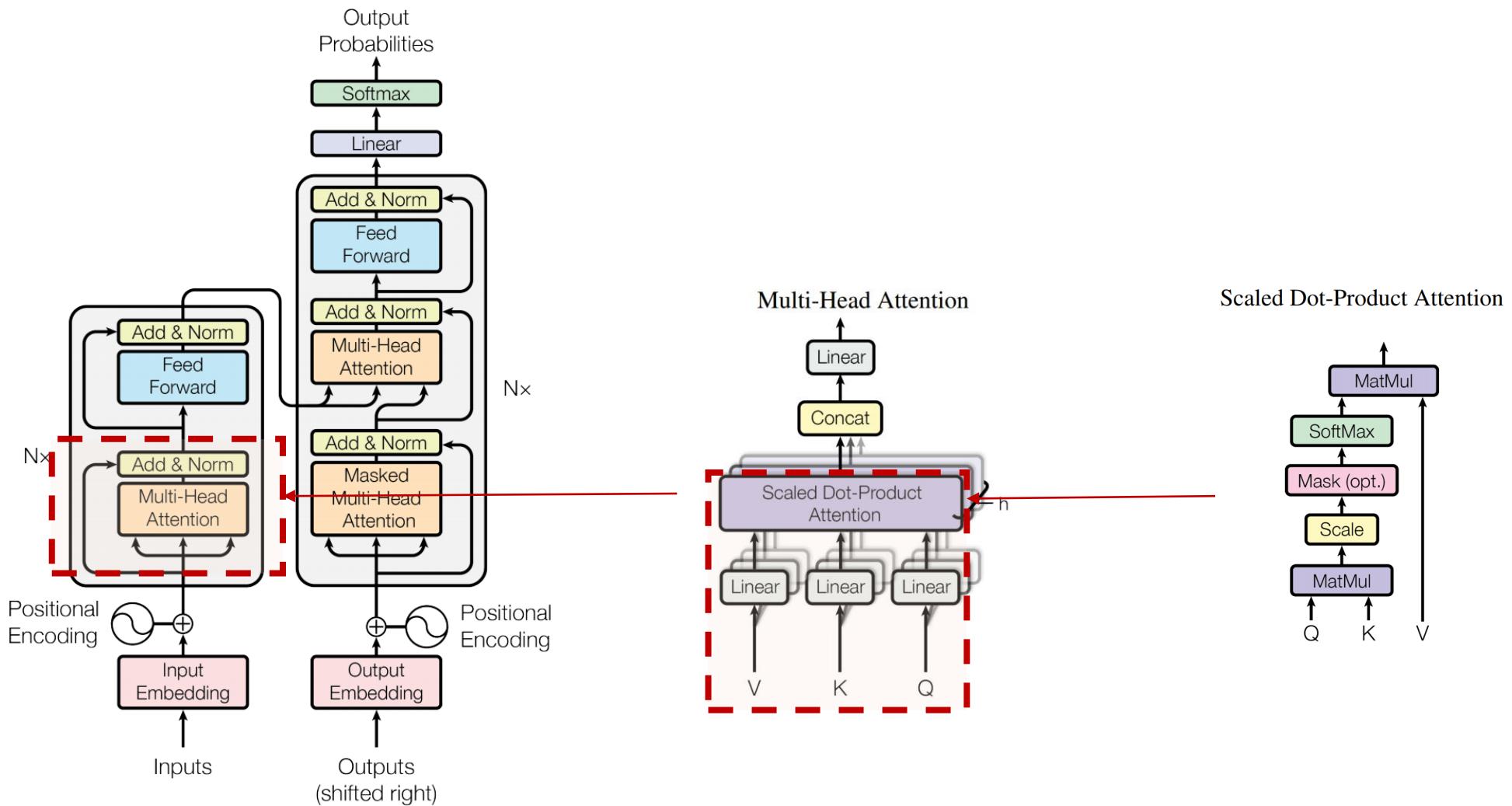
- Why self-attention
  - RNN 계열은 각 layer 별로 parameters 가 많습니다.
  - RNN 계열은  $h_i$  계산을 위하여  $h_{i-1}$  가 필요합니다. 순차적인 계산이 필요하므로 병렬화 (parallelization) 이 어렵습니다.
  - 여전히 LSTM 도 long dependency 는 잘 학습되지 않습니다.
    - 두 단어가 연결되기 위한 path 가 깁니다.

# Transformer (Attention is all you need)



- Encoder, decoder, attention network 모두 feed forward neural network 를 이용하여 구성합니다.
- Encoder, decoder 는 6 개의 transformer layer 로 이뤄졌습니다.

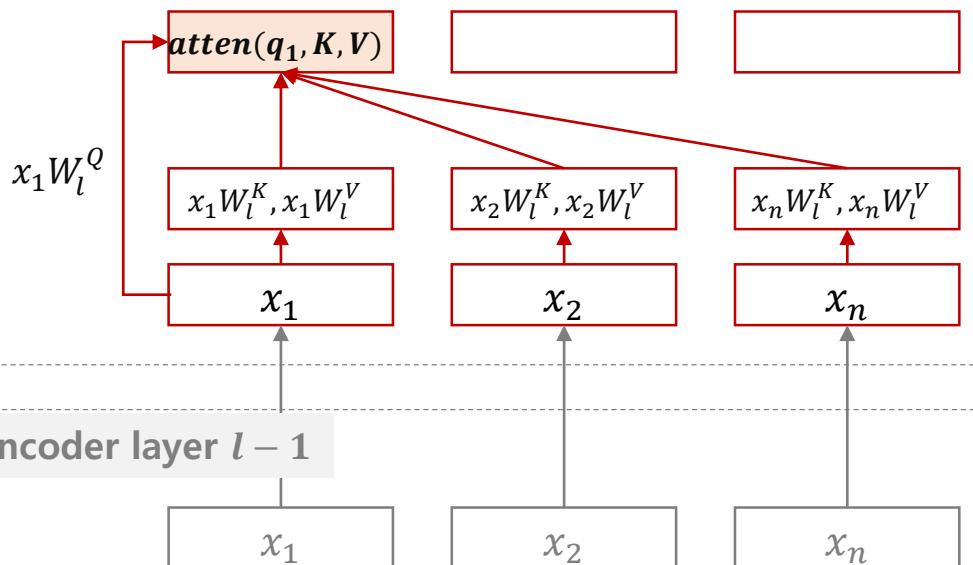
# Transformer (Attention is all you need)



# Transformer (Attention is all you need)

- Layers 의 output 은  $d_{model}$  차원의  $n$  개의 벡터열 입니다.  
( $d_{model} = 512$ ,  $n$  : 문장의 단어 개수)
- $layer_l$  의  $i$  번째 output 을 만들기 위하여  $x_i W_l^Q$  를 query 로  
 $x_i W_l^K, x_i W_l^V$  를 key, value 로 이용합니다.
- $W_l^Q, W_l^K, W_l^V$  를 통하여 input 의 공간을 변화시킵니다.

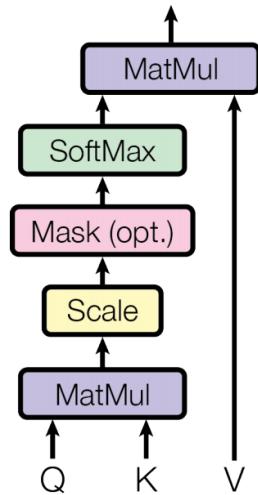
Encoder layer  $l$



Encoder layer  $l - 1$

# Transformer (Attention is all you need)

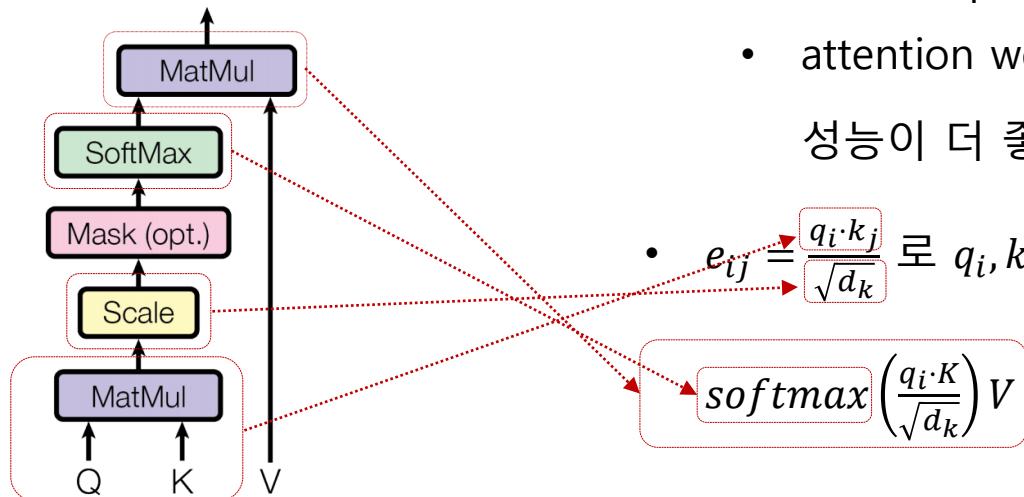
Scaled Dot-Product Attention



- $atten(x_i W_l^Q, x_j W_l^K, x_j W_l^V)$  를 잠시 동안  $atten(q, k, v)$  라 합니다.
- Query 는 현재 처리해야 하는 단어이며, key 는 query 와의 유사도 계산에 이용할 정보, value 는 key 에 해당하는 실제 값입니다.
- HAN 이나 seq2seq + atten 에서는 key, value 모두  $h_i$  입니다.
  - attention weight 와 average 에 이용하는 정보를 나눠 학습하면 성능이 더 좋아지기 때문에  $(W_l^Q, W_l^K, W_l^V)$  로 정보를 나눕니다.
- $e_{ij} = \frac{q_i \cdot k_j}{\sqrt{d_k}}$  로  $q_i, k_j$  의 유사도를 계산합니다.
$$\text{softmax}\left(\frac{q_i \cdot K}{\sqrt{d_k}}\right)V : i$$
 번째 output 을 계산합니다.

# Transformer (Attention is all you need)

Scaled Dot-Product Attention



- $\text{atten}(x_i W_l^Q, x_j W_l^K, x_j W_l^V)$  를 잠시 동안  $\text{atten}(q, k, v)$  라 합니다.
- Query 는 현재 처리해야 하는 단어이며, key 는 query 와의 유사도 계산에 이용할 정보, value 는 key 에 해당하는 실제 값입니다.
- HAN 이나 seq2seq + atten 에서는 key, value 모두  $h_i$  입니다.
  - attention weight 와 average 에 이용하는 정보를 나눠 학습하면 성능이 더 좋아지기 때문에  $(W_l^Q, W_l^K, W_l^V)$  로 정보를 나눕니다.
- $e_{ij} = \frac{q_i \cdot k_j}{\sqrt{d_k}}$  로  $q_i, k_j$  의 유사도를 계산합니다.
- $\text{softmax}\left(\frac{q_i \cdot K}{\sqrt{d_k}}\right) V$ : 으로  $i$  번째 output 을 계산합니다.

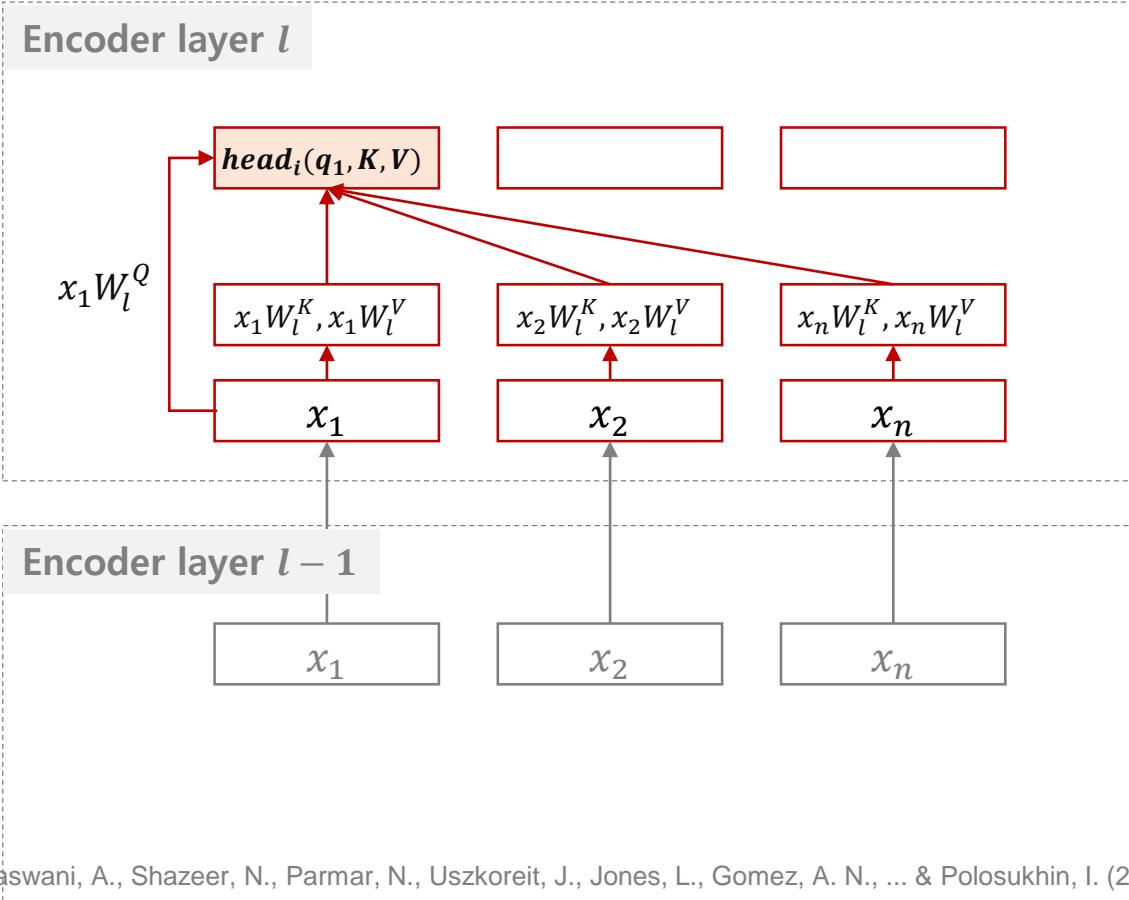
# Transformer (Attention is all you need)

---

- 하나의 attention 보다는, 작은 attention 여러 개를 합쳐 이용하면 더 정교한 모델링이 가능합니다.
  - $\text{multihead}(Q, K, V) = \text{concat}(\text{head}_1, \dots \text{head}_h)W_l^O$
  - $d_{model} = 512$  를  $h = 8$  개의 부분 공간으로 나눠 각각 attention 을 학습합니다.
  - 마치 여러개의 conv. 필터처럼 서로 다른 관점으로 문장을 해석하는 역할을 합니다.

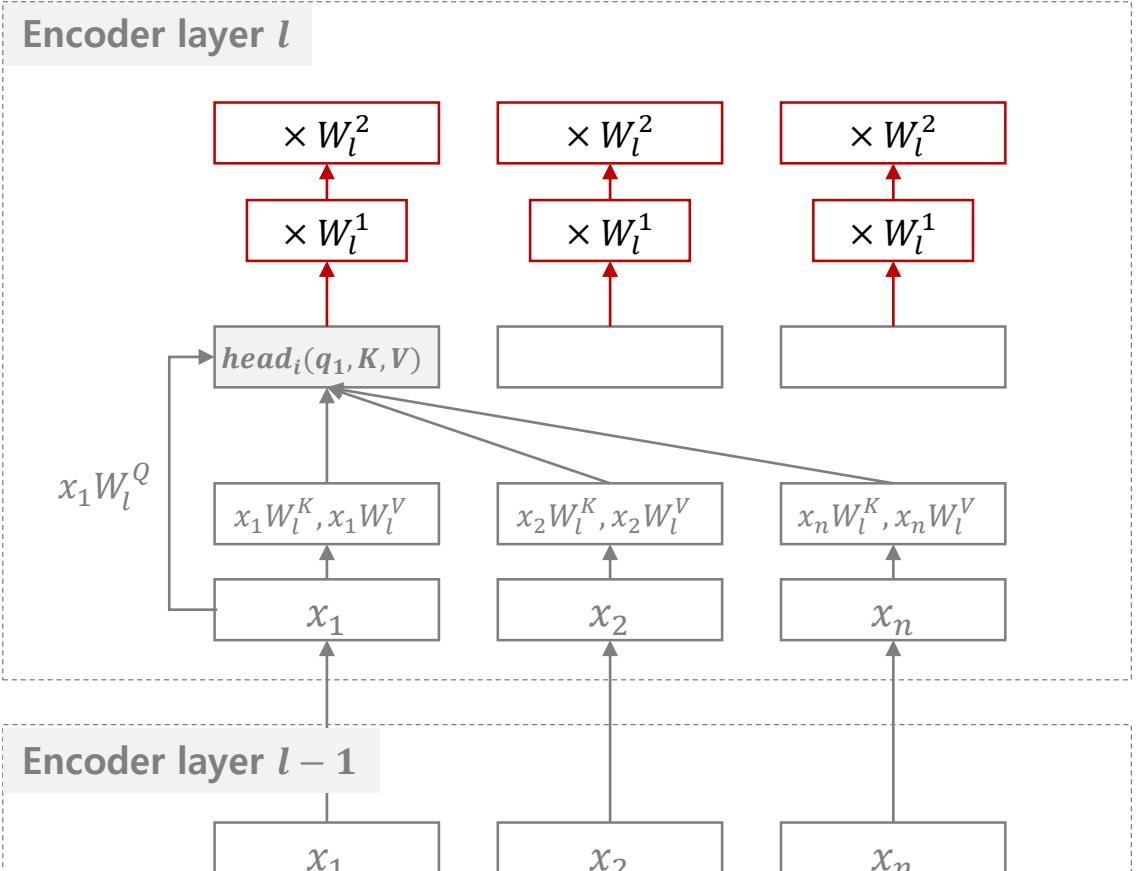
# Transformer (Attention is all you need)

- $\text{multihead}(Q, K, V) = \text{concat}(\text{head}_1, \dots, \text{head}_h)W_l^O$
- $\text{head}_i(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$
- $W_l^Q, W_l^K \in R^{d_m \times d_k}$   
 $W_l^V \in R^{d_m \times d_v}$   
 $W_l^O \in R^{hd_v \times d_m}$



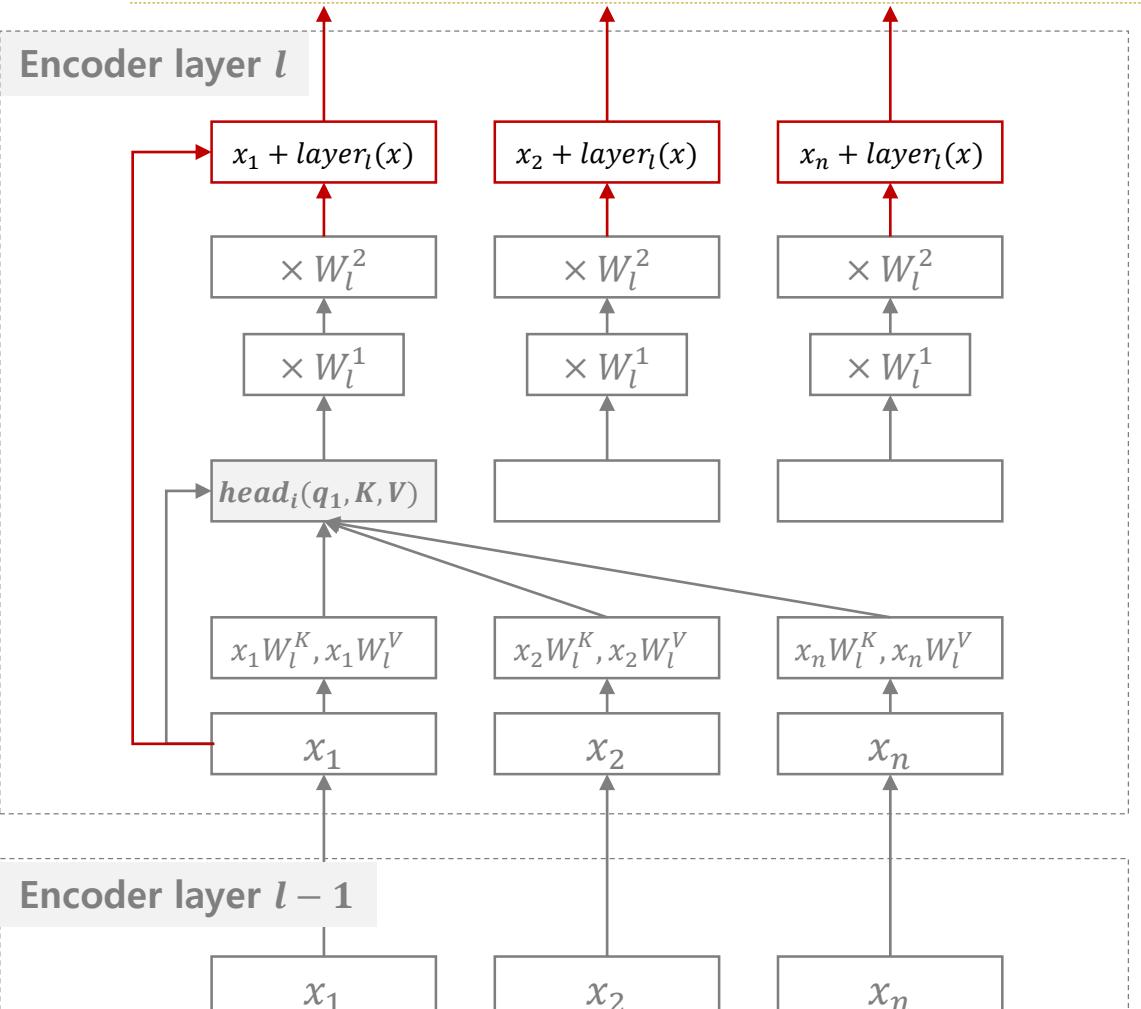
# Transformer (Attention is all you need)

- Multi-head attention 의 결과에 2 layer feed-forward net 을 거쳐 각 단어의 representation  $x_i$  의 공간을 변화시킵니다.



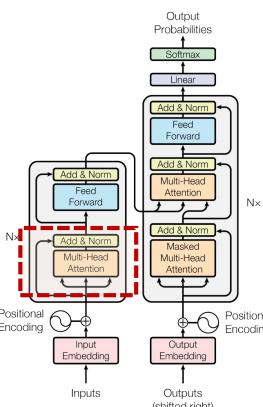
$$FFN(x_i) = \max(0, x_i W_1 + b_1)W_2 + b_2$$

# Transformer (Attention is all you need)

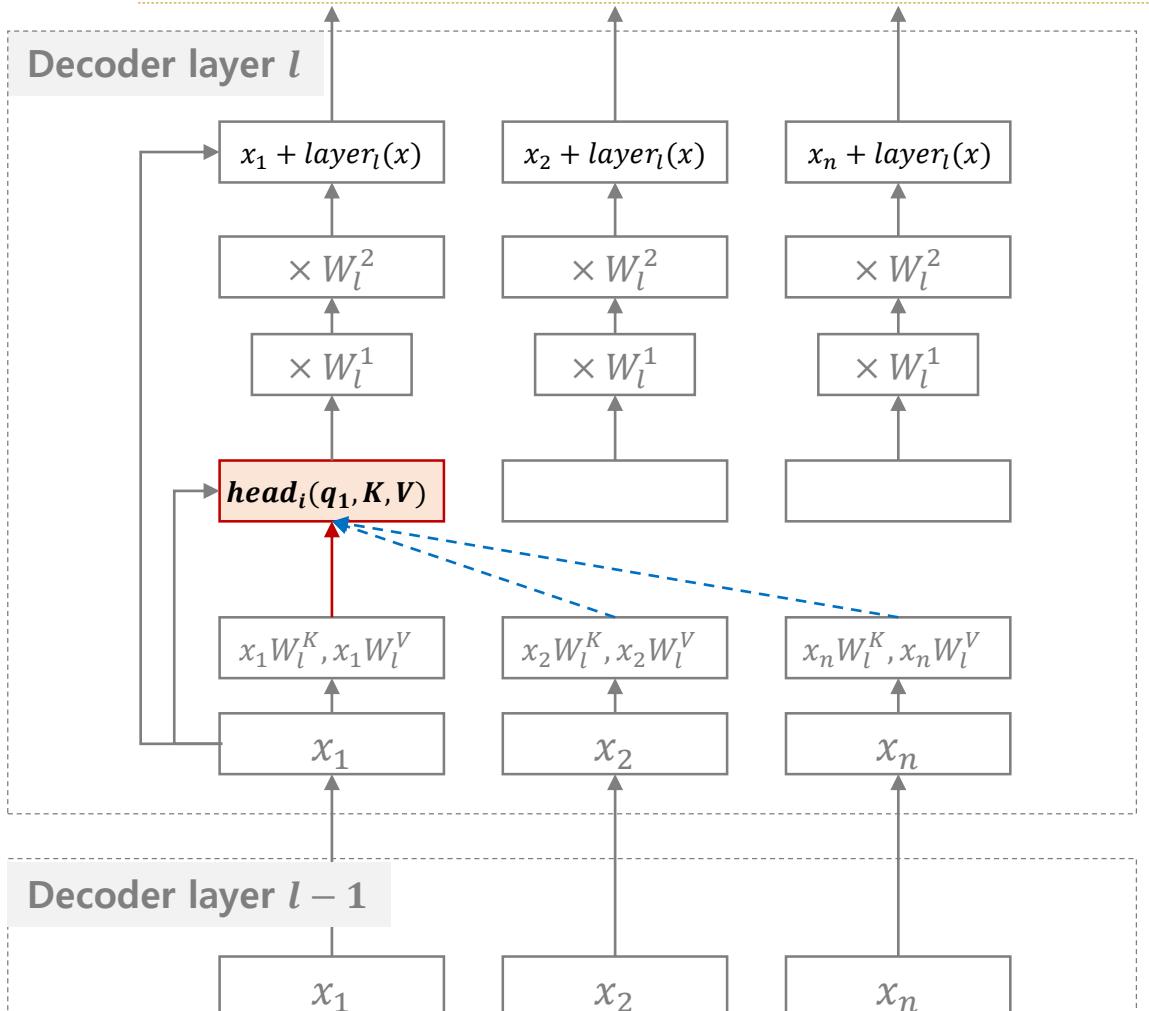


- Residual connection 을 통하여 input sequence 에 정보를 추가합니다.

$$x \leftarrow x + sublayer(x)$$

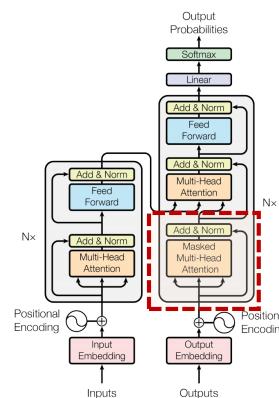


# Transformer (Attention is all you need)



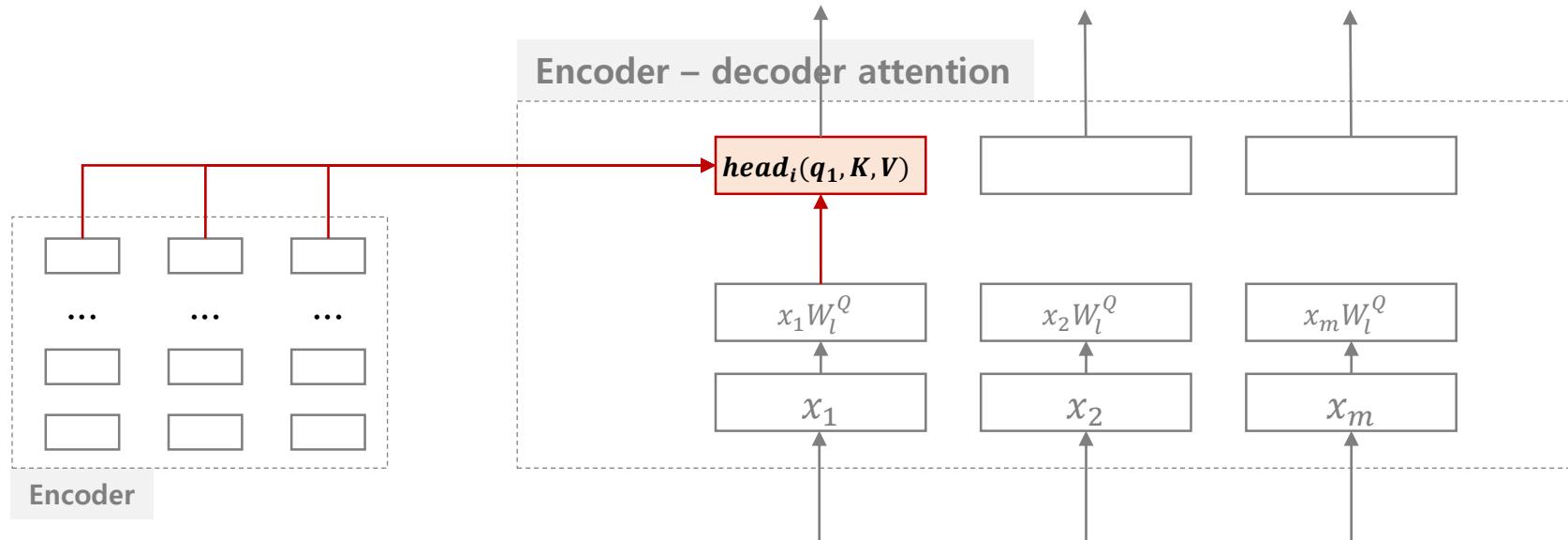
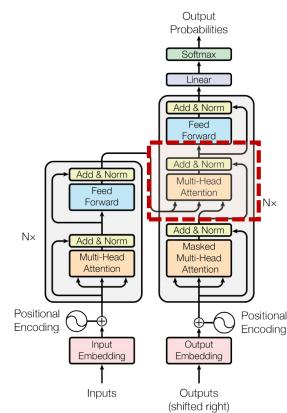
- Decoder layer 에서는  $q_i$  의 attention 에  $\{k_j: v_j\}, j \leq i$  만 이용 가능합니다.

- 단어 생성시에는 이전에 생성된 단어만 이용합니다.



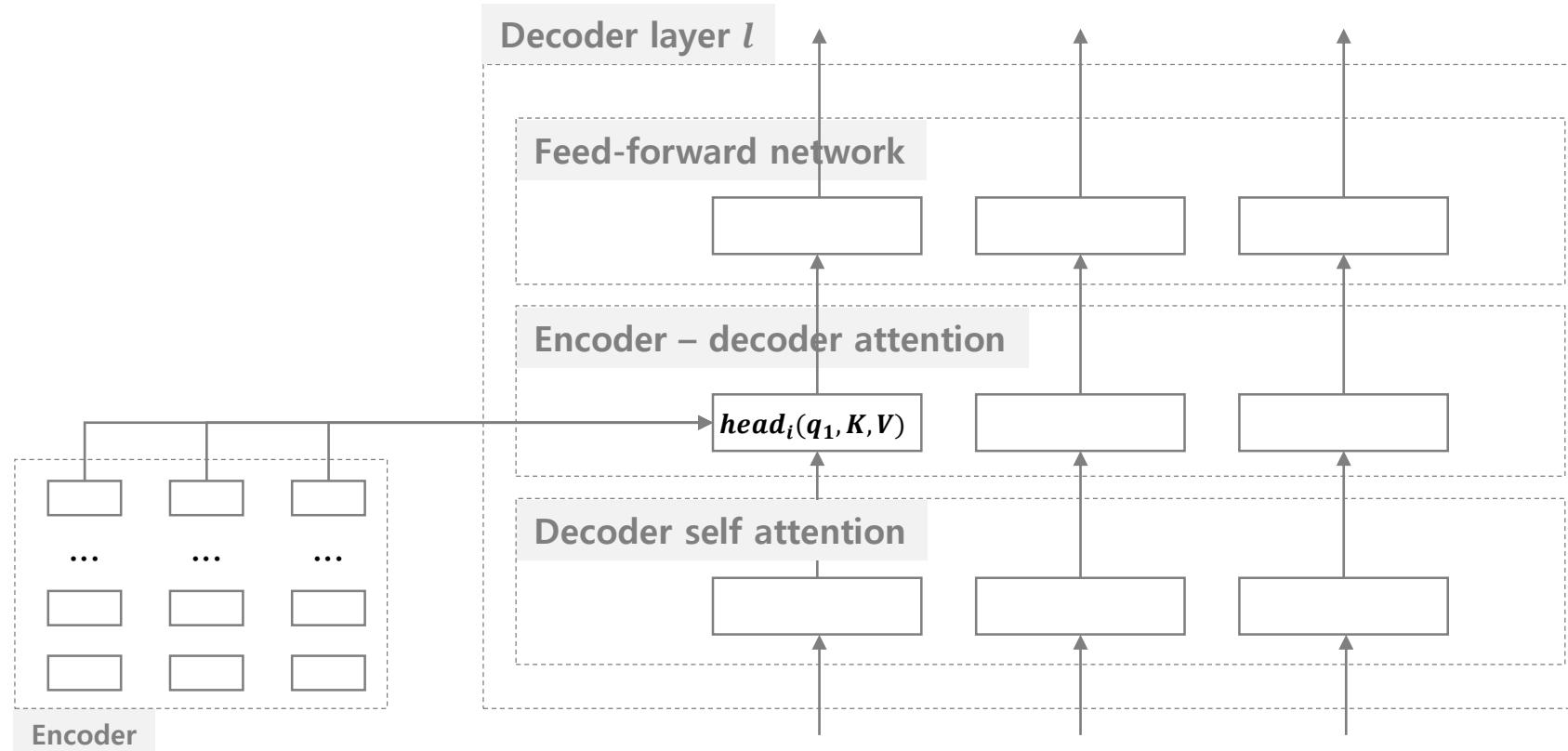
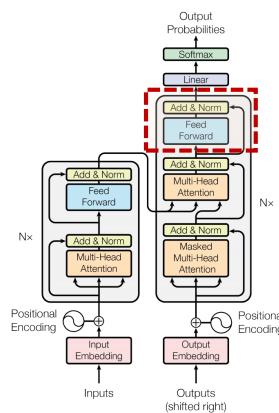
# Transformer (Attention is all you need)

- Last encoder layer output 과의 attention 을 통하여 input sequence 정보를 이용합니다.



# Transformer (Attention is all you need)

- Decoder 의 self attention layer 와 encoder – decoder attention 의 결과가 더해진 뒤, feed forward net 을 통과합니다.



# Transformer (Attention is all you need)

---

- 계산량이 훨씬 줄었음에도 BLUE 기준 성능이 향상되었습니다.

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

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	<b>41.29</b>	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	<b><math>3.3 \cdot 10^{18}</math></b>	
Transformer (big)	<b>28.4</b>	<b>41.0</b>	$2.3 \cdot 10^{19}$	

# Transformer (Attention is all you need)

Component	Note
Multi-head attention	<ul style="list-style-type: none"><li>같은 크기의 공간을 이용한다면 단일한 attention 보다, 여러 개의 작은attention 의 결과를 합쳐 (concatenation) 쓰는 것이 더 좋습니다.</li><li>마치 CNN model 에서의 여러 개의 convolution filter 와도 비슷합니다.</li><li>각 head 마다 병렬적으로 계산이 가능합니다.</li></ul>
Encoder – decoder attention	<ul style="list-style-type: none"><li>Encoder – decoder 사이의 attention 으로, sequence to sequence 에서의 attention 과 같은 역할을 합니다.</li></ul>
Feed-forward layer	<ul style="list-style-type: none"><li>2 layers feed-forward net 로 attention(q, K, V) 결과를 선형변환 합니다.</li></ul>
Positional encoding	<ul style="list-style-type: none"><li>Transformer 는 단어의 위치 정보를 전혀 이용하지 않기 때문에 input vector 에 position vector 를 더하여 입력합니다.</li></ul>
Residual connection	<ul style="list-style-type: none"><li><math>x \leftarrow x + sublayer(x)</math></li><li>attention 을 거친 뒤 input 과 더함으로써, <i>sublayer</i> 를 통하여 정보를 추가합니다.</li></ul>

# Transformer (Attention is all you need)

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

- $n > d$  이지 않다면 self-attention 이 RNN 계열보다 계산량이 적으며,
- feed-forward network 를 이용하기 때문에 병렬 계산이 쉽습니다.
- $attention(Q, K, V)$  에 의하여 두 단어가 직접 연결 됩니다 ( $O(1)$ )

# Transformer (Attention is all you need)

- 서로 상관이 있는 단어들이 높은 attention 을 지닙니다.



# Transformer (Attention is all you need)

---

Encoding meaning, not word

After **starting** with representations of **individual words** or even pieces of words,  
they **aggregate information from surrounding words**  
**to determine the meaning** of a given bit of language in context.

For example, deciding on the most likely meaning and appropriate representation of the word  
“**bank**” in the sentence “I arrived at the bank after crossing the...”  
requires knowing if the sentence **ends in “... road.” or “... river.”**

# Transformer (Attention is all you need)

---

- 여러 개의 networks 를 한 번에 학습합니다.
- 학습용 병렬 데이터가 있다면 end – to – end training 이 가능합니다.

# Transformer (Attention is all you need)

---

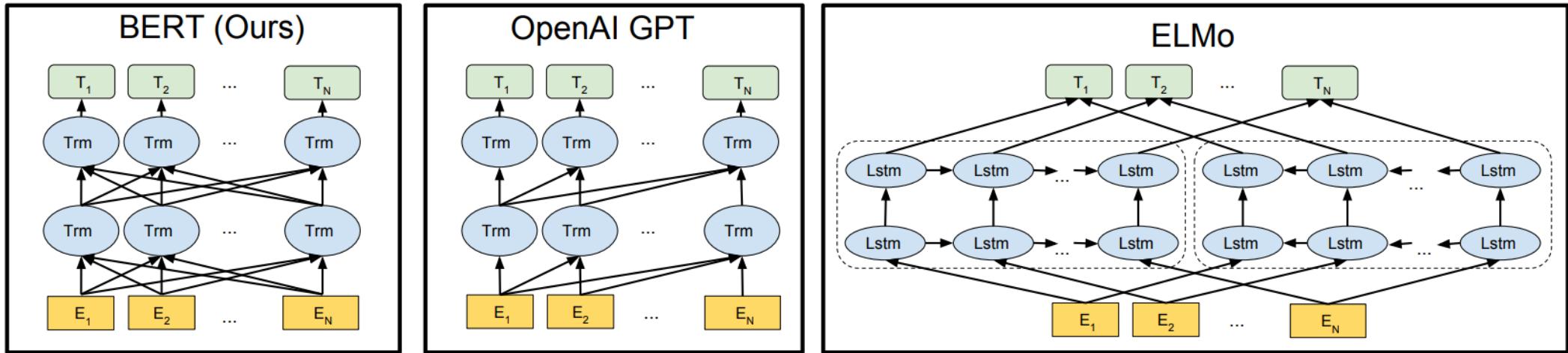
- <http://nlp.seas.harvard.edu/2018/04/03/attention.html>

# BERT

---

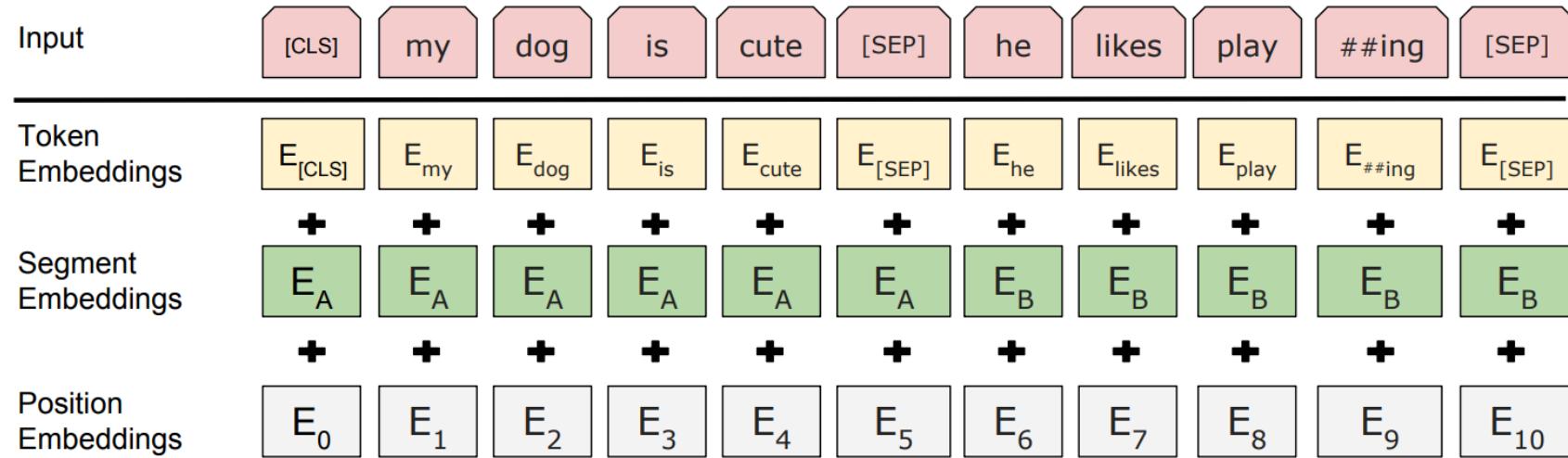
- Transformer 를 이용하는 language model 로, 다양한 작업의 pre-trained model 로 이용됩니다.
  - Bidirectional Encoder Representations from Transformers
  - Masked LM 방식을 이용하여 pre-training 을 수행한 뒤,
  - 각각의 tasks 에 알맞게 짧은 시간 fine-tuning 을 합니다.

# BERT



- ELMo 는 hidden vector 를 feature 로 직접 이용합니다.
- OpenAI 의 GPT 는 transformer 를 이용하지만, 단방향 학습을 합니다.
- BERT 는 masked LM 방식으로 양방향 학습이 가능합니다.

# BERT



- Input 으로 (1) word embedding, (2) position embedding, (3) sentence embedding ( $E_A$ ,  $E_B$ )
- 문장의 시작 부분에 [CLS], 문장의 끝 부분에 [SEP] special token 을 이용합니다.
- 첫 문장에는  $E_A$ , 둘째문장에는  $E_B$  embedding vectors 를 더하는데, 이는 문장에 대한 position embedding 역할을 합니다. ( $E_A$ ,  $E_B$  도 special token 입니다)

# BERT

---

- Masked LM 은 BERT 의 학습 방식으로, 15 % tokens 을 임의로 masking 한 뒤, 이를 예측하는 문제를 학습합니다 (pre-training)
  - 단어가 [mask] 라는 special token 으로 치환됩니다.
  - 15 % 로 선택된 token 중, 80 % [mask] 로 치환 / 10 % 다른 단어 / 10 % 실제 단어를 남겨둡니다.
  - Word embedding vector 가 각각 [mask] / 다른 단어 / 실제 단어에서 look-up 됩니다.
  - 모든 단어를 [mask] 로 변환하면 그 또한 bias 가 생기기 때문입니다.

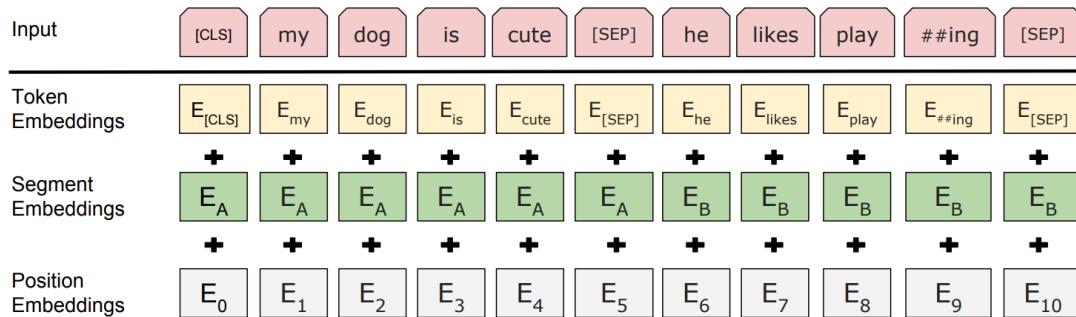
# BERT

---

- Masked LM 은 BERT 의 학습 방식으로, 15 % tokens 을 임의로 masking 한 뒤, 이를 예측하는 문제를 학습합니다 (pre-training)
  - sent = "my dog is hairy", → selected "hairy"
    - with 80 %, "my dog is [mask]"
    - with 10 %, "my dog is apple"
    - with 10 %, "my dog is hairy"
  - Model predict whether  $x(4)$  is "hairy"

# BERT

- 연속된 두 문장을 이용하여 masked LM 을 학습합니다.
  - NLP tasks 중에는 두 개의 문장을 이용하는 경우들이 있습니다 (Q&A)
  - (negative sampling 처럼) 앞의 문장으로 뒤의 문장을 예측하는 pre-training tasks 도 학습합니다. (pos : neg = 50 % : 50 %)

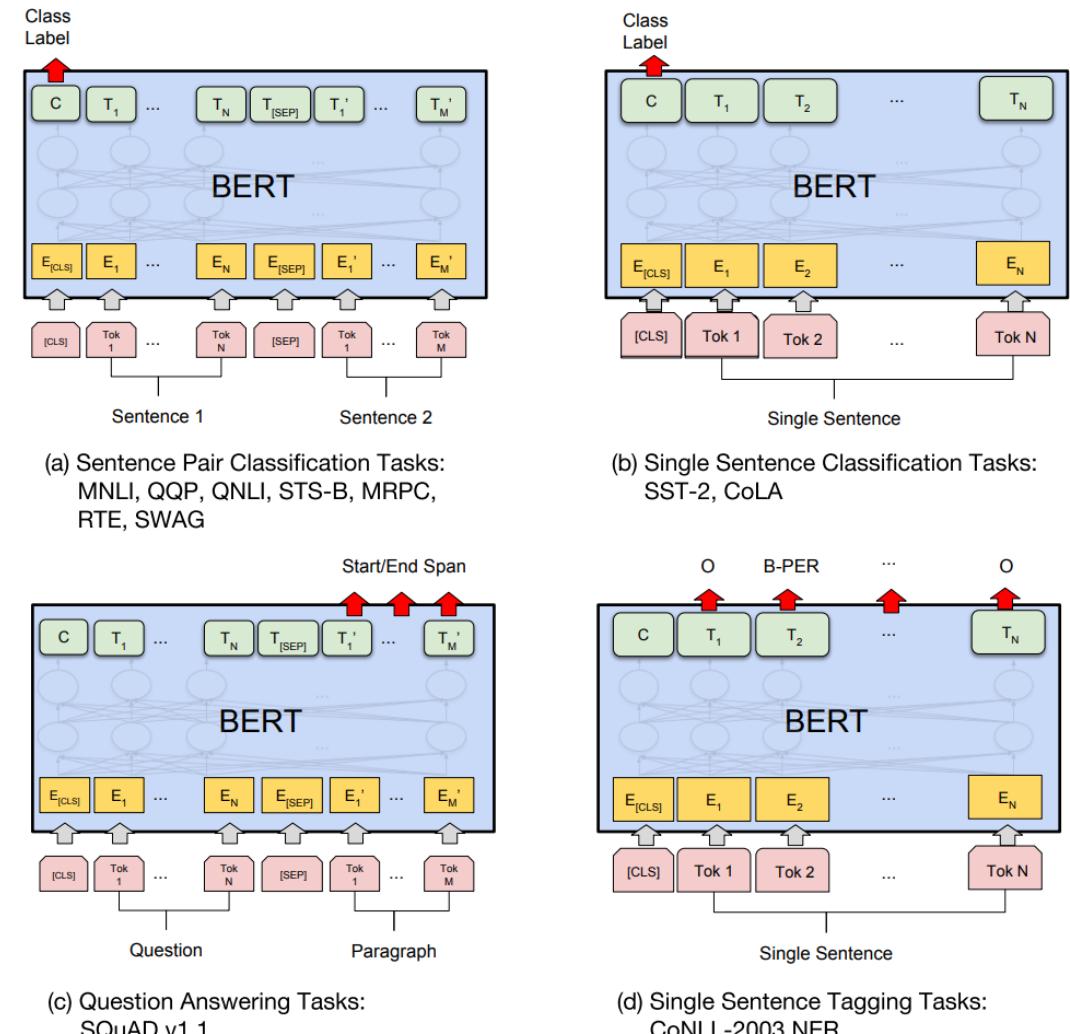


Input = [CLS] the man went to [MASK] store [SEP]  
he bought a gallon [MASK] milk [SEP]  
Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]  
penguin [MASK] are flight ##less birds [SEP]  
Label = NotNext

# BERT

- tasks 별로 다른 정보를 이용합니다.
  - Sentence classification / similarity:
    - output of [CLS] in last layer
  - Q&A:
    - outputs of sent B in last layer
  - Sequential labeling:
    - output of sent A in last layer



# BERT

---

- 각 문장별로 15 % 의 token 만 pre-training 에 이용하므로, 여러 번의 batch 를 돋니다 (100,000 epochs)
- Parameters 의 양이 매우 큽니다.
  - $\text{BERT}_{\text{BASE}} = 110\text{M}$     $\text{BERT}_{\text{LARGE}} = 340\text{M}$
  - AlphaGo policy network = 4.6M / ResNet-50 = 25M \*
- Base / Large 각각 4TPU / 16TPU 로 4 일 학습하였습니다.

# BERT

---

- Wikipedia, BooksCorpus 와 같은 일반적인 데이터로 pre-training 을 한 뒤, 단일모델로 11 개의 NLP tasks 에서 state of the art 를 이룹니다.
  - 대량의 데이터에 대한 거대한 pre-training model 을 학습하여 대부분의 tasks 에 공유하여 사용합니다.

- 
- <http://docs.likejazz.com/bert/>
  - <http://jalammar.github.io/illustrated-bert/>

# Deep learning in machine translation

---

One of the main successes of deep learning is due to the effectiveness of recurrent networks for language modeling and their application to speech recognition and machine translation.

However, in other cases including several text classification problems, it has been shown that deep networks do not convincingly beat the prior state of the art techniques.

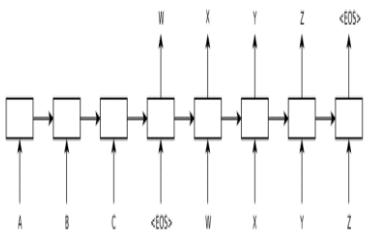
## NMT 에서의 딥러닝의 변화

---

- 기계번역 (Neural, Machine Translation) 은 딥러닝 모델링이 적용되어 성공적으로 상품화된 분야입니다.
  - 실험적으로 성공한 분야는 많지만, 상품화까지 진행되는 분야는 적습니다.
  - 기계번역 분야의 변화를 살펴보면 NLP + 딥러닝의 발전 방향을 예상할 수 있습니다.

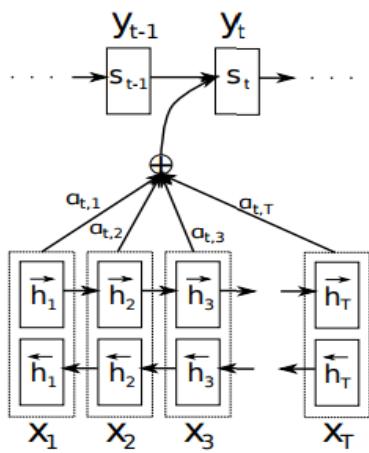
# NMT 에서의 딥러닝의 변화

seq2seq



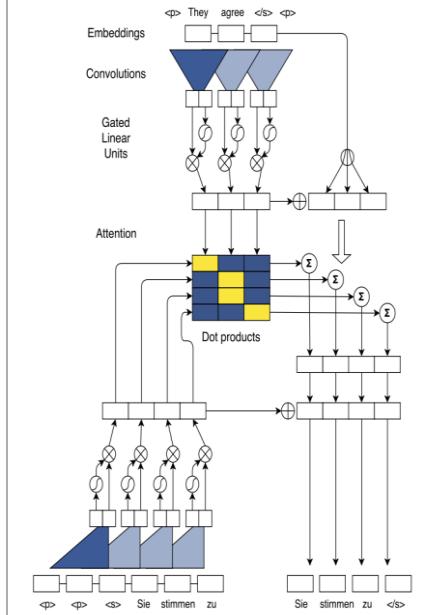
(2013)

Seq2seq + Attention



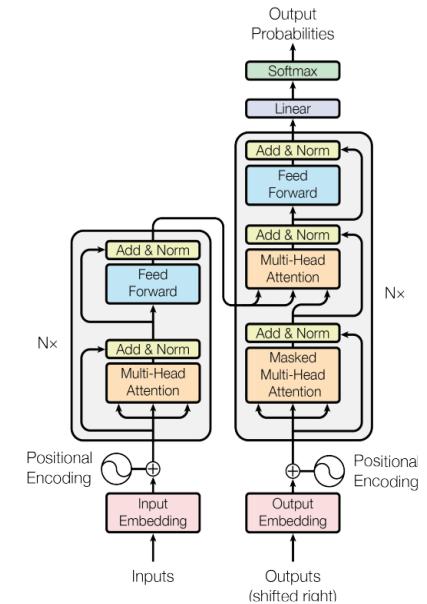
(2014)

CNN seq2seq



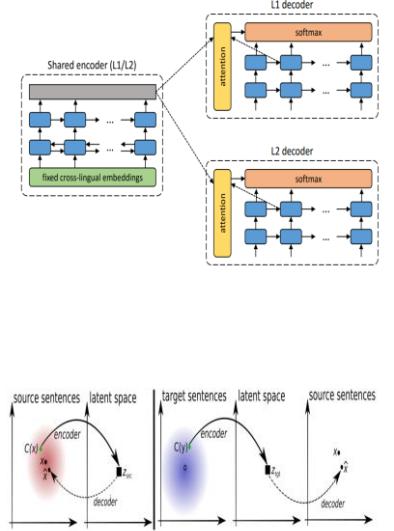
(2017)

Self attention



(2017)

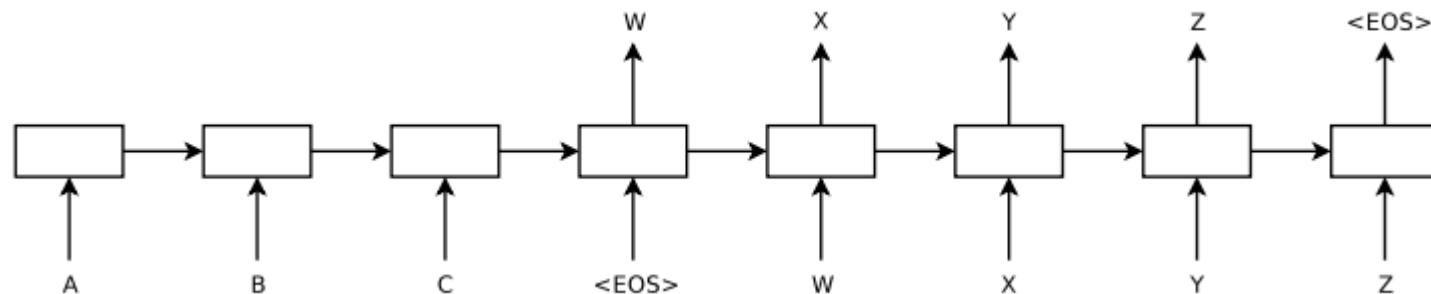
Unsupervised machine translation



(2018)

# NMT 에서의 딥러닝의 변화

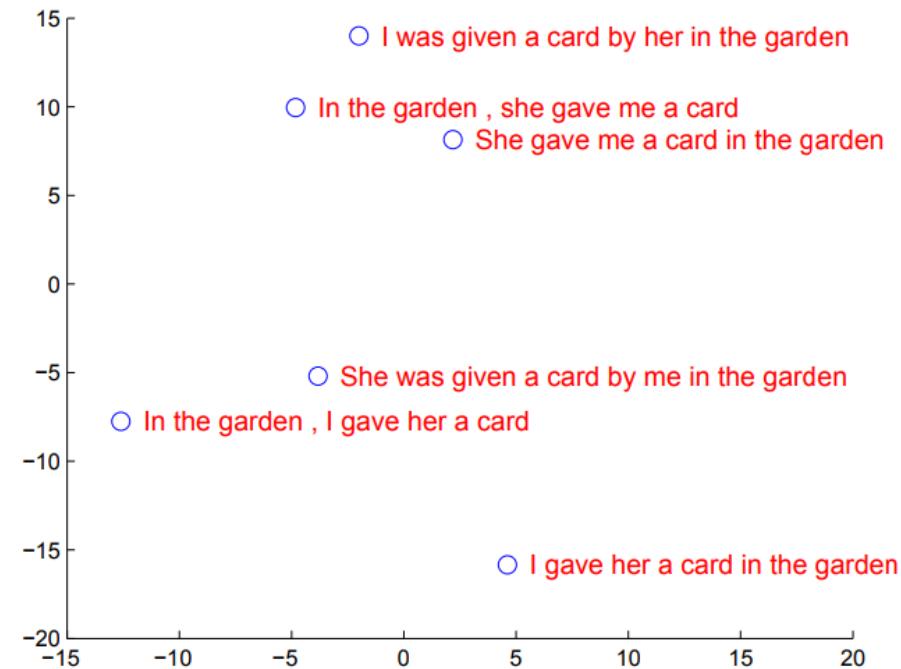
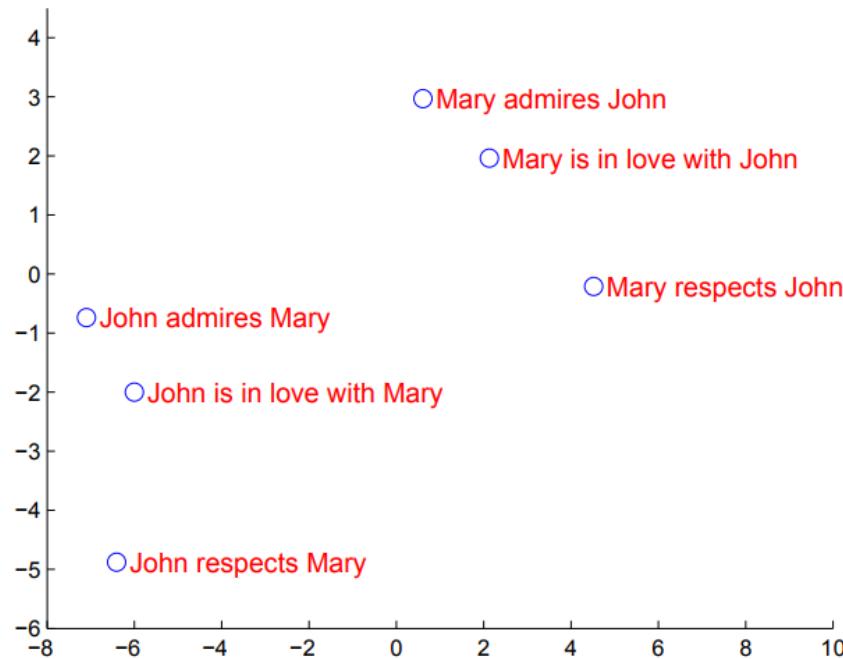
- Seq2seq 에서 RNN 을 이용한 Encoder – decoder 구조가 제안되었습니다.
  - 긴 문맥을 반영하는 것이 목적입니다.



"WXYZ" = translate("ABC")

# NMT 에서의 딥러닝의 변화

- RNN Encoder – Decoder 구조는 문장의 의미적 유사성을 학습합니다.



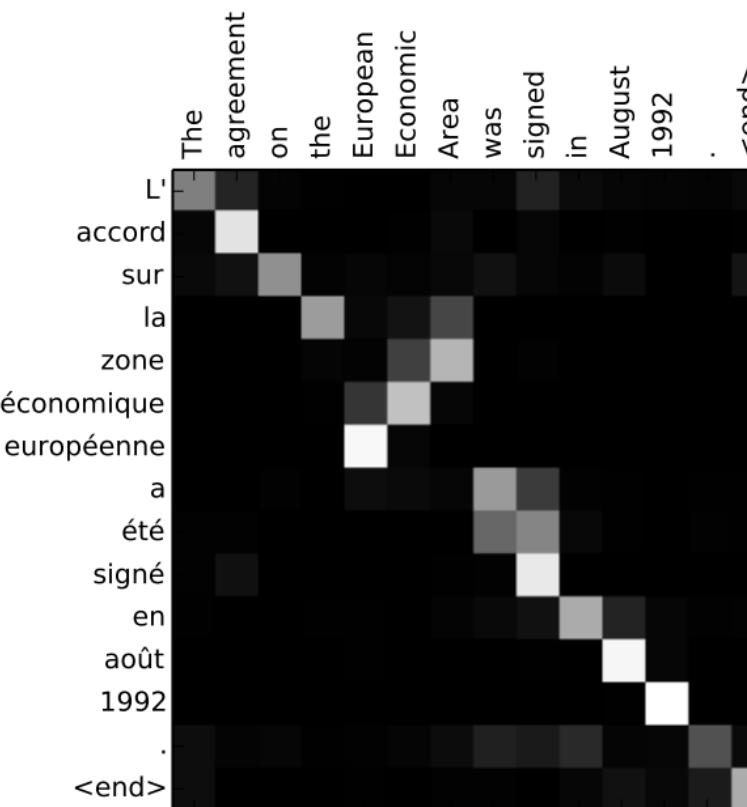
# NMT 에서의 딥러닝의 변화

---

- Seq2seq 은 한 문장을 하나의 fixed length vector  $c$  로 표현합니다.
  - “하나의 문장의 정보를 하나의 벡터로 표현하는 것이 옳은가” 의문을 제기
- Seq2seq + Attention 에서는
  - 하나의 문장을 “단어 벡터열”로 표현하며,
  - $y_i$  를 생성할 때,  $c$  가 아닌,  $h = [h_1, \dots, h_T]$  를 조합하여 매 단어마다 서로 다른 문맥을 이용합니다.

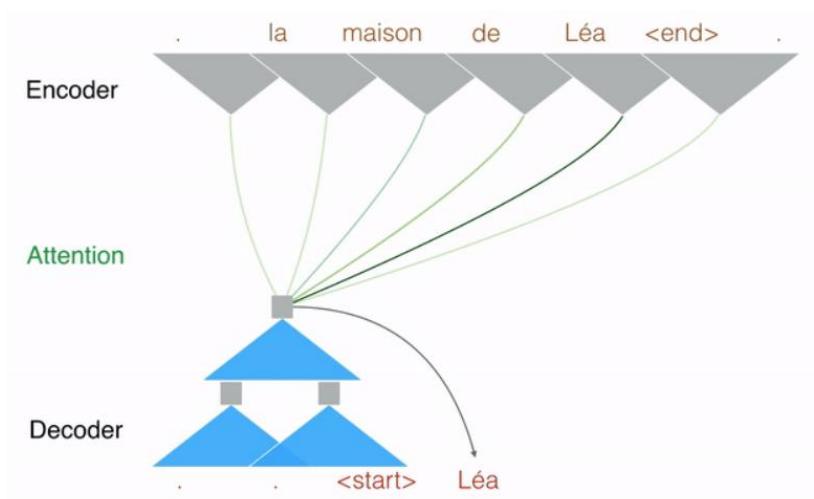
# NMT 에서의 딥러닝의 변화

- Attention 은 상황에 필요한 정보만을 선택할 수 있습니다



# NMT 에서의 딥러닝의 변화

- Convolutional Seq2seq 은 Encoder – decoder 에 CNN 을 이용합니다.
  - Recurrent networks 는 순차적으로 단어를 읽기 때문에 병렬화가 어렵습니다.
- Multi-step attention 을 이용함으로써, 한 층의 layer 로 표현하기 어려운 복잡한 의미 (문맥)도 표현합니다.



# NMT 에서의 딥러닝의 변화

- Self – attention 은 다시 feed-forward 를 sequential modeling 에 이용합니다.
  - RNN 의 장점인 가변적인 문맥 인식의 능력을 지니면서도의
  - feed-forward networks 의 장점인 계산량과 병렬화를 이용합니다.
  - Attentions 을 통하여 문맥적인 의미를 이해합니다.

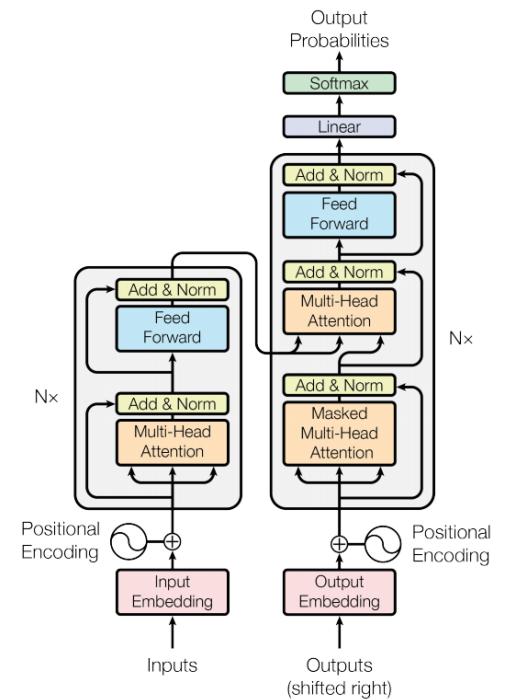
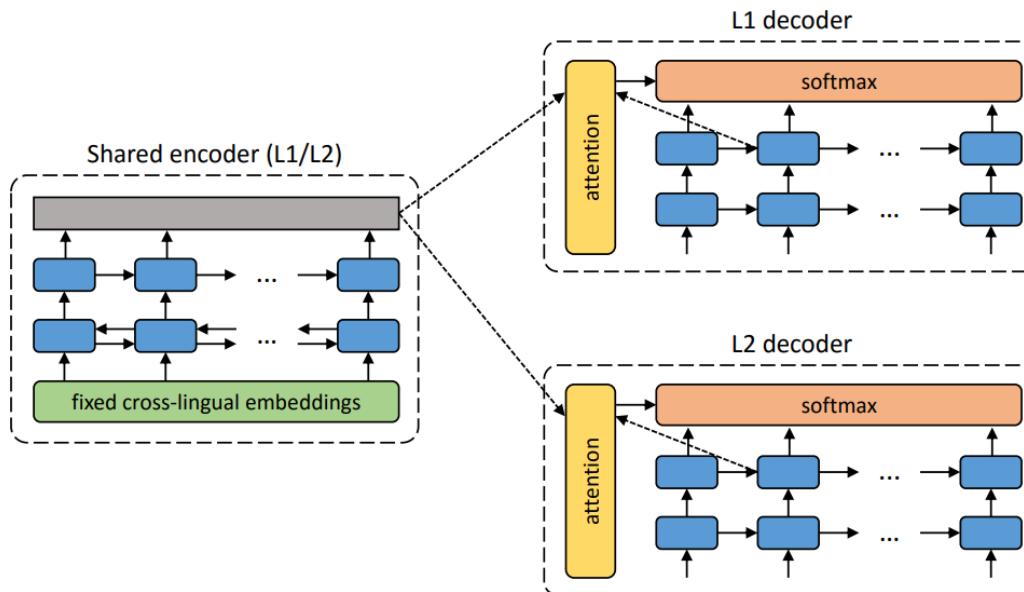


Figure 1: The Transformer - model architecture.

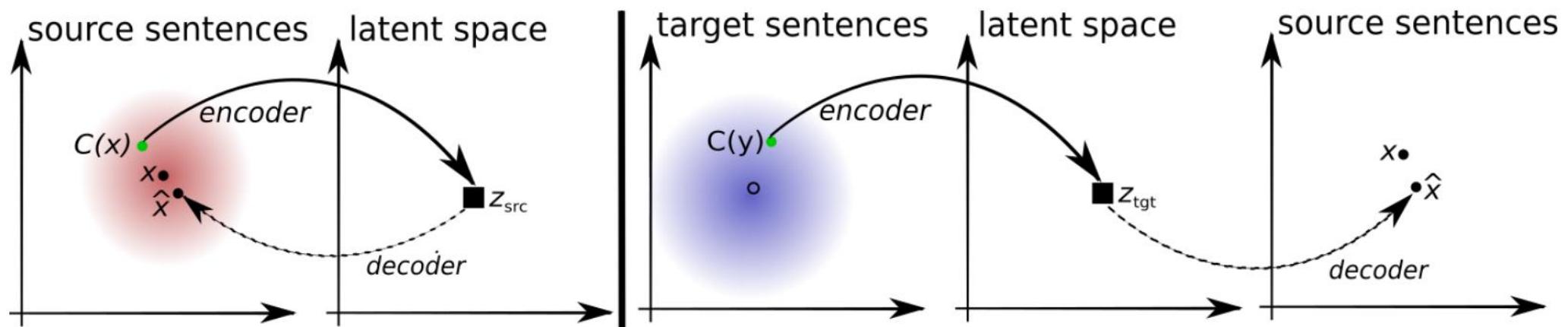
# NMT 에서의 딥러닝의 변화

- ICLR 2018 에 parallel corpus 를 이용하지 않는 NMT 가 제안되었습니다.
  - L1 은 denoising encoder – decoder system 을 이용합니다.
  - L2 에서는 L1 으로 문장을 번역합니다.



# NMT 에서의 딥러닝의 변화

- Facebook에서도 ICLR 2018에 parallel corpus를 이용하지 않는 NMT를 제안했습니다.
  - 두 언어가 공통된 latent space를 share합니다.
  - Source  $\rightarrow$  latent space  $\rightarrow$  target sentence로 번역됩니다.



# NMT 에서의 딥러닝의 변화

---

- RNN 은 feed-forward NN 과 달리 문장 전체의 정보를 이용합니다.
- LSTM 은 long dependency 를 잘 학습합니다.
- Seq2Seq 은 입력문장의 정보를 압축하여 번역에 이용합니다.
- Seq2seq + attention 은 필요한 정보를 선택하여 번역에 이용합니다.
- Self – attention 은 문장 전체의 문맥을 고려하는 feed-forward 방식입니다.
- 병렬 말뭉치에 의존하지 않기 위한 방법들이 제안되고 있습니다.

# Reference

---

- <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>
- <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- <https://distill.pub/2016/augmented-rnns/>
- <http://web.stanford.edu/class/cs224n/>
- <https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html>
- [https://pytorch.org/tutorials/intermediate/seq2seq translation tutorial.html](https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html)
- <https://www.youtube.com/user/mrkebi>