Recurrent Neural Network

Hyunjoong Kim

soy.lovit@gmail.com

github.com/lovit

Language model &
Recurrent Neural Network

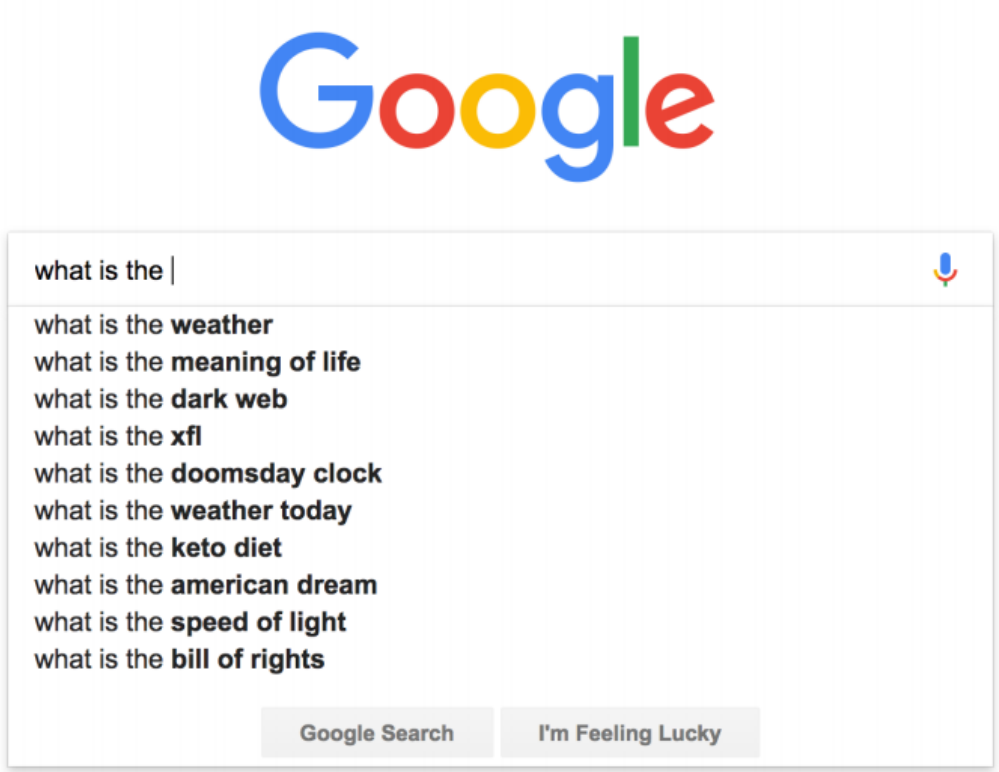
language model

- 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



http://web.stanford.edu/class/cs224n/

Classical language model

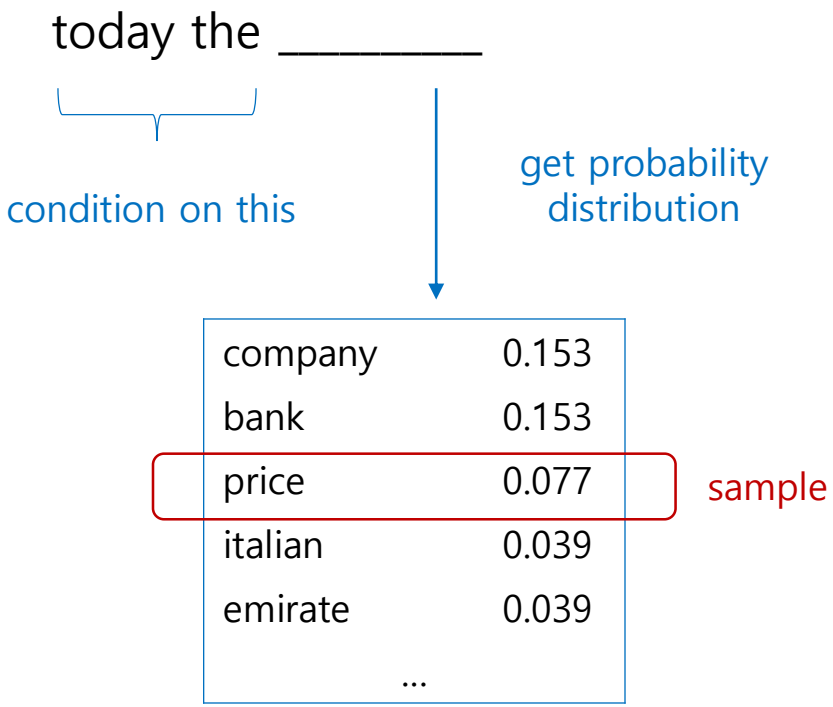
- n-grams 을 이용하는 language model 이 제안되었습니다.
  - $p(w_m|\, w_1, w_2, \ldots w_{m-1}) = p(w_m|\, w_{m-2:m-1})$
  - $p(s) = \ p(w_1, w_2, \ldots, w_m) = p(w_1) \times p(w_2|\, w_1) \times p(w_3|w_{1:2}) \ldots$

Classical language model

- $p(w_i | w_{i-n+1:i-1})$ 은 빈도수로 정의됩니다.
  - $p(w_i | w_{i-n+1:i-1}) = \frac{\#(w_{i-n+1:i})}{\#(w_{i-n+1:i-1})}$
  - $p(cat \mid this, is) = \frac{\#\,(this,is,cat)}{\#\,(this,is)}$

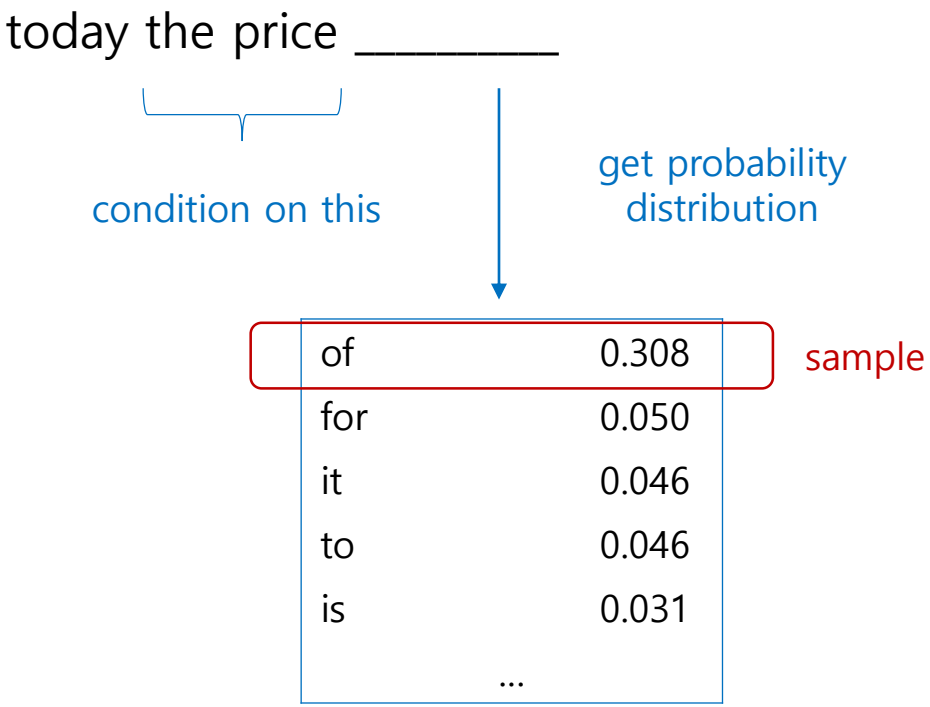
Sentence generation

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



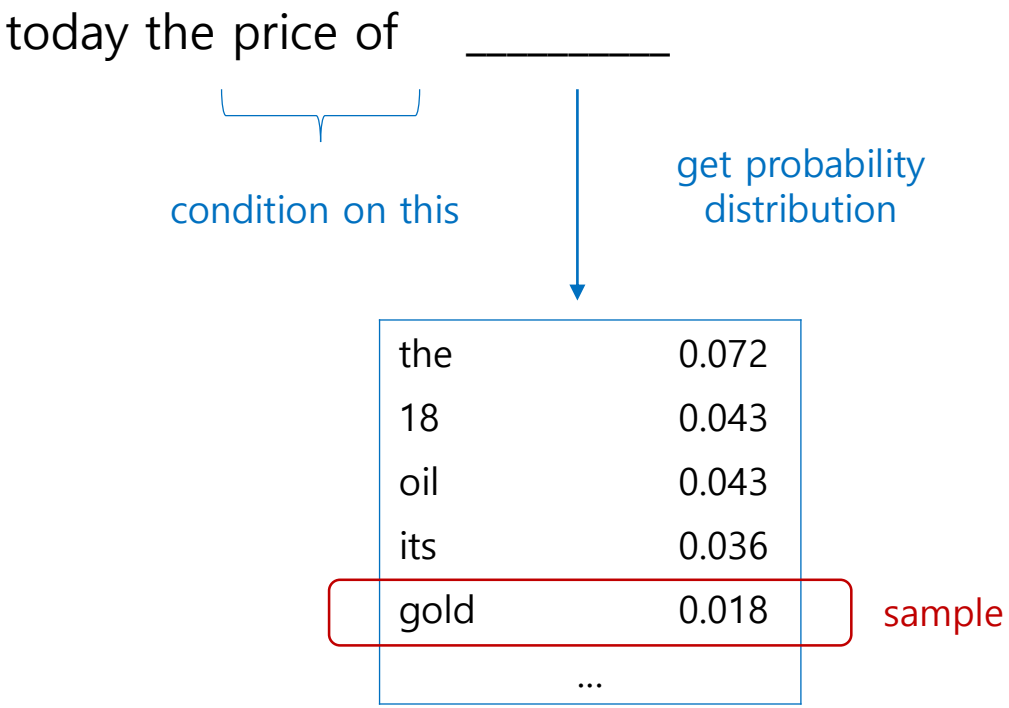
Sentence generation

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



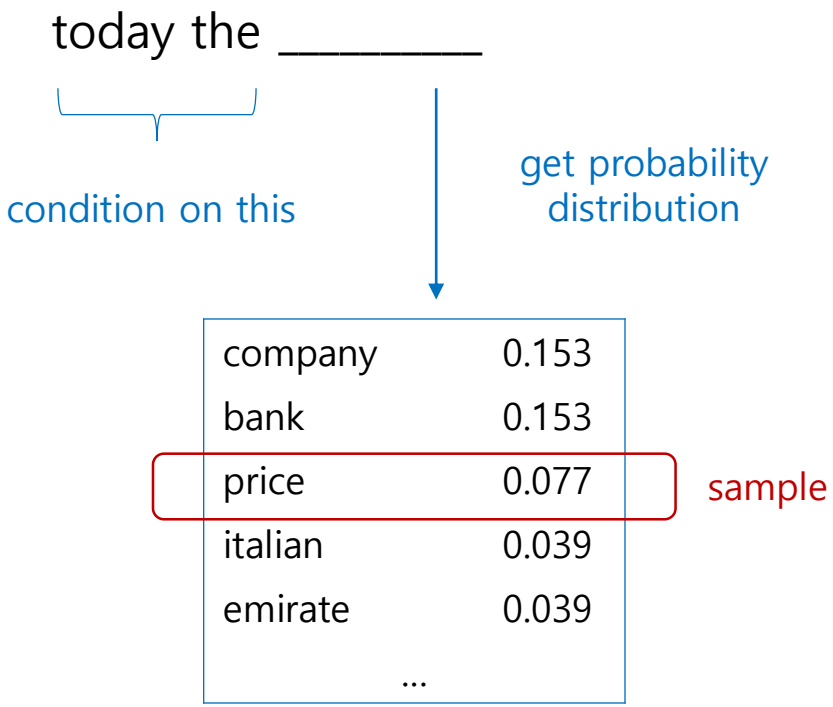
Sentence generation

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



Sentence generation

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

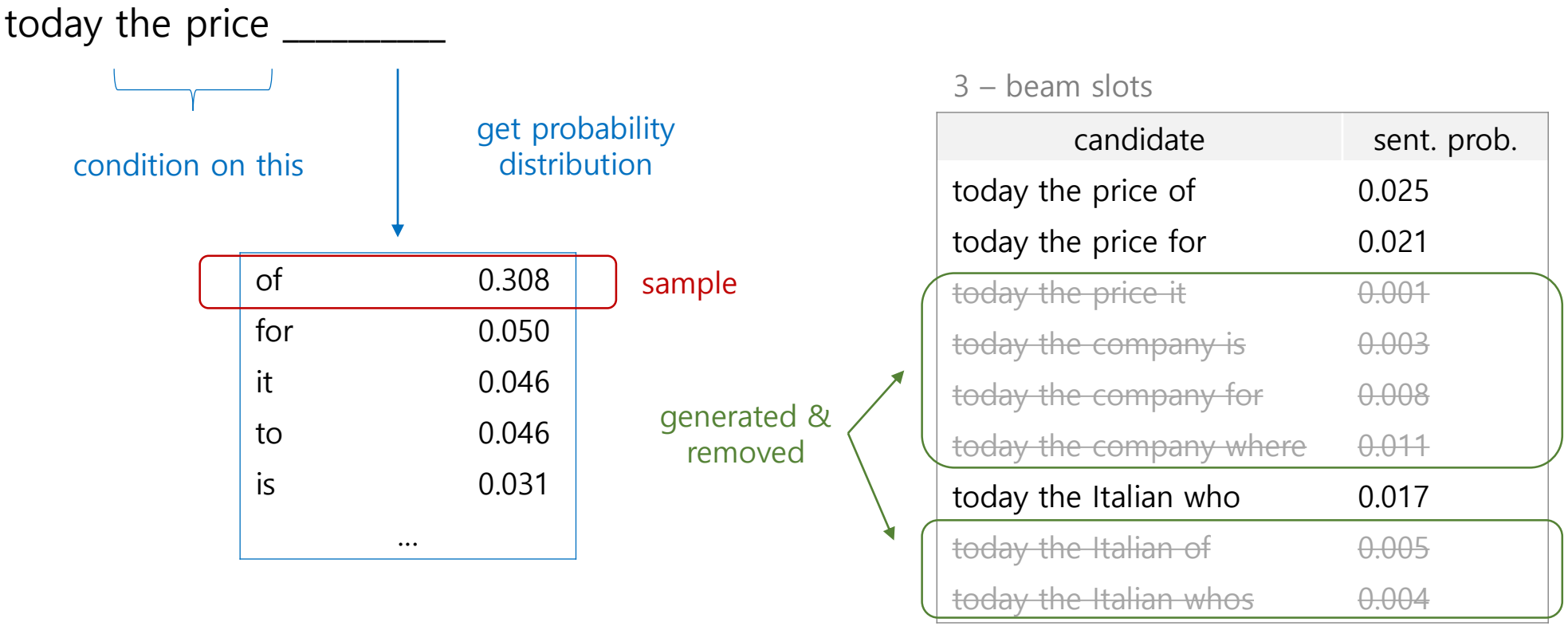


3 – beam slots

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

Sentence generation

- Beam search 는 매 step 마다 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 0 이 아닌 확률이 생깁니다.
  - 그 외에도 다양한 smoothing 방법이 제안되었습니다.

Classical language model

- Back-off 는 길이가 짧은 n-grams 을 함께 context 로 이용합니다.
  - $p(w_4|w_1, w_2, w_3) = \alpha_1 \times \frac{\#(w_{1:4})}{\#(w_{1:3})} + \alpha_2 \times \frac{\#(w_{2:4})}{\#(w_{2:3})} + \alpha_3 \times \frac{\#(w_{3:4})}{\#(w_3)}$

- Katz's back-off model
  - $P(w_i|\, w_{i-n+1}, \ldots w_{i-1}) = \begin{cases} \frac{\#(w_{i-n+1:i})}{\#(w_{i-n+1:i-1})} \text{ if } (w_{i-n+1:i}) > k \\ P(w_i|\, w_{i-n+2}, \ldots, w_{i-1}) \quad otherwise \end{cases}$

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 겨냥한 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)*

Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, *3*(Feb), 1137-1155.

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

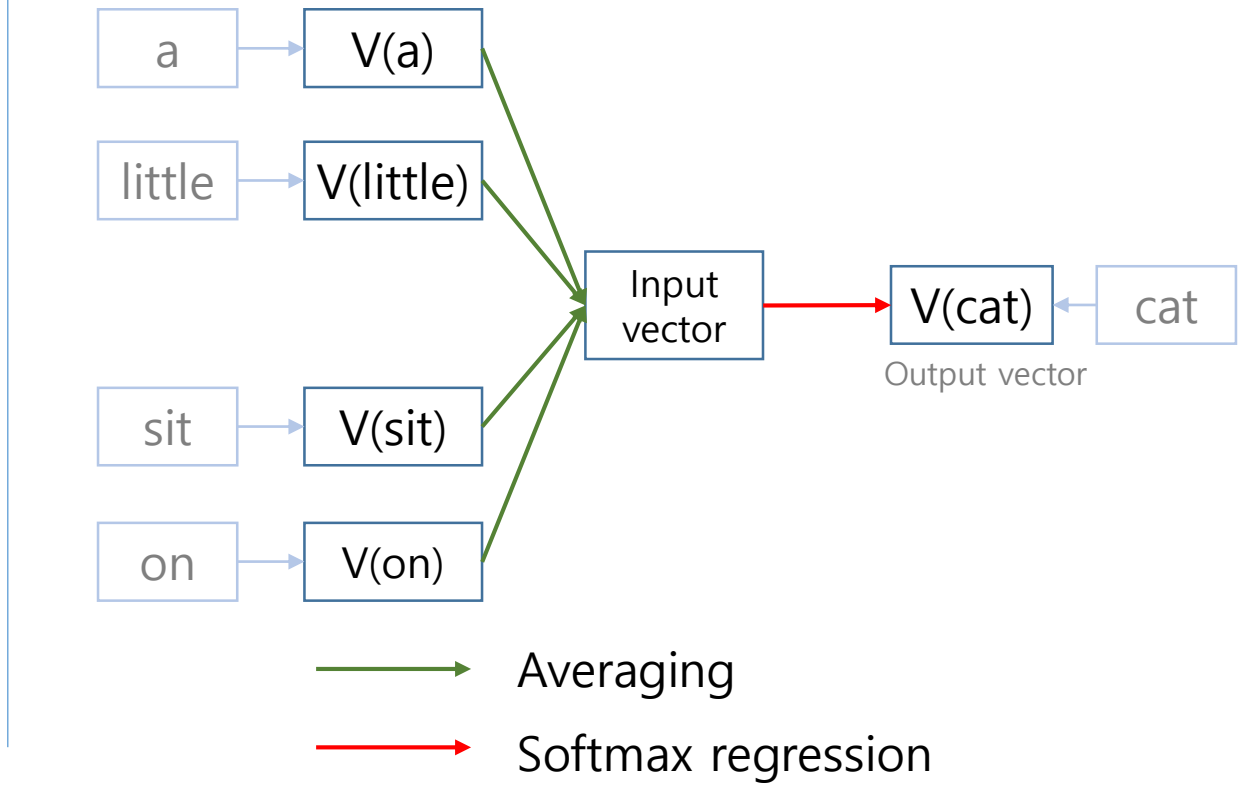
- Word2Vec은 |V| 개의 classes 를 예측하는 Softmax regression 입니다.

Lookup table

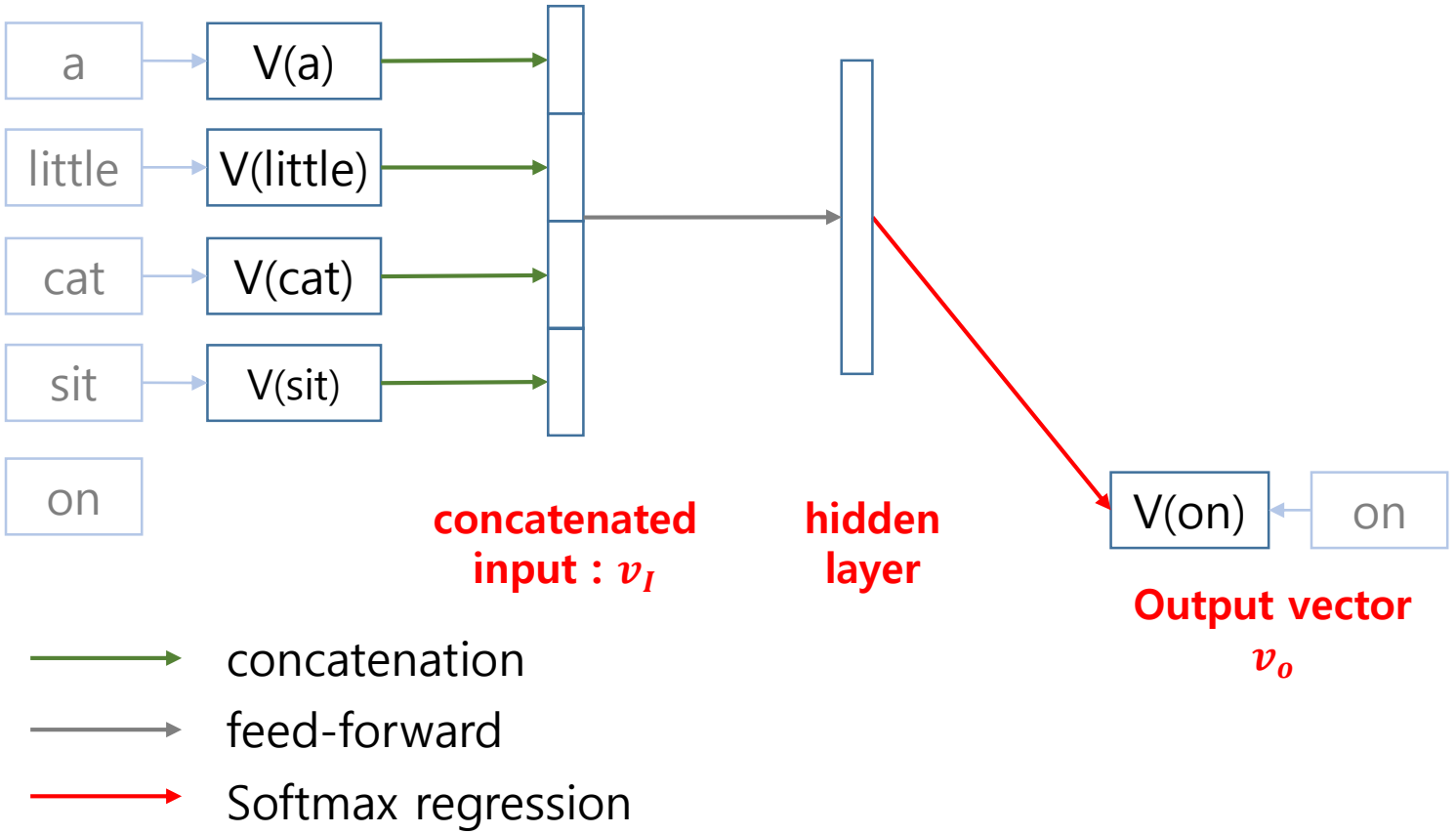
```
{
  cat: [.31, -.22, .54]
  dog: [.22, -0.3, -0.52]
  on: …
  the: …
  little: …
}
```

Lookup word vector

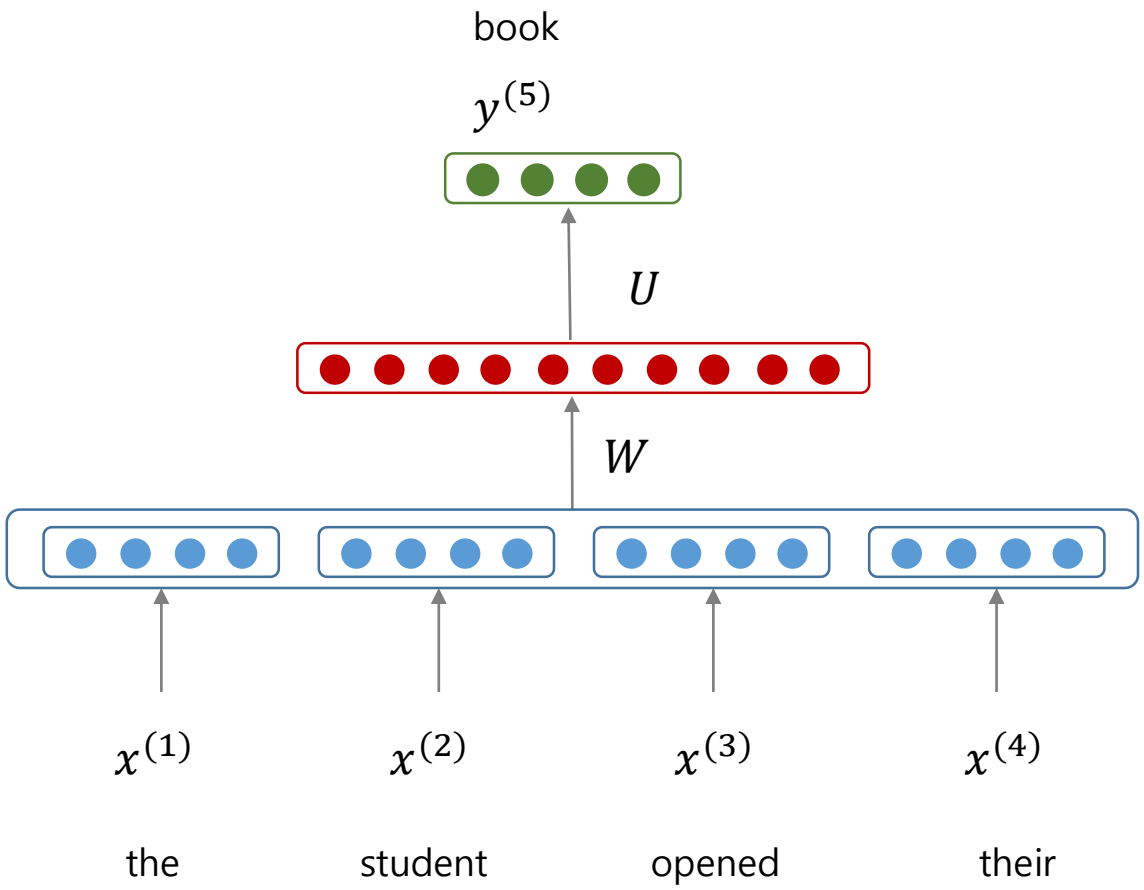
V('cat') = [.31, -.22, .54]



Feed-forward network based language model



Feed-forward network based language model



output distribution

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

hidden layer

$$h = f(We)$$

concatenated word embedding

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

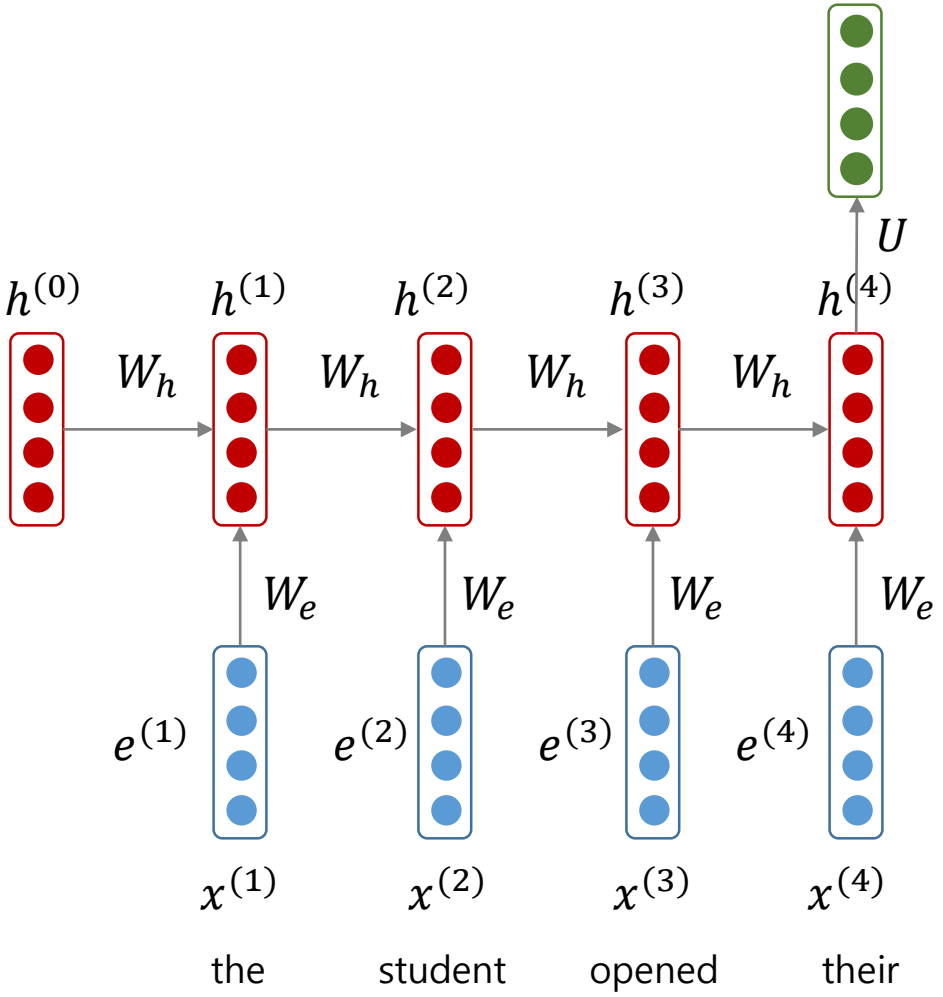
Feed-forward network based language model

- 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

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

$p\left(y^{(t)}\middle|y_{1:t-1}\right)$

hidden layer

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

$h^{(0)}$: initial hidden state

word embedding

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

Recurrent neural network based language model

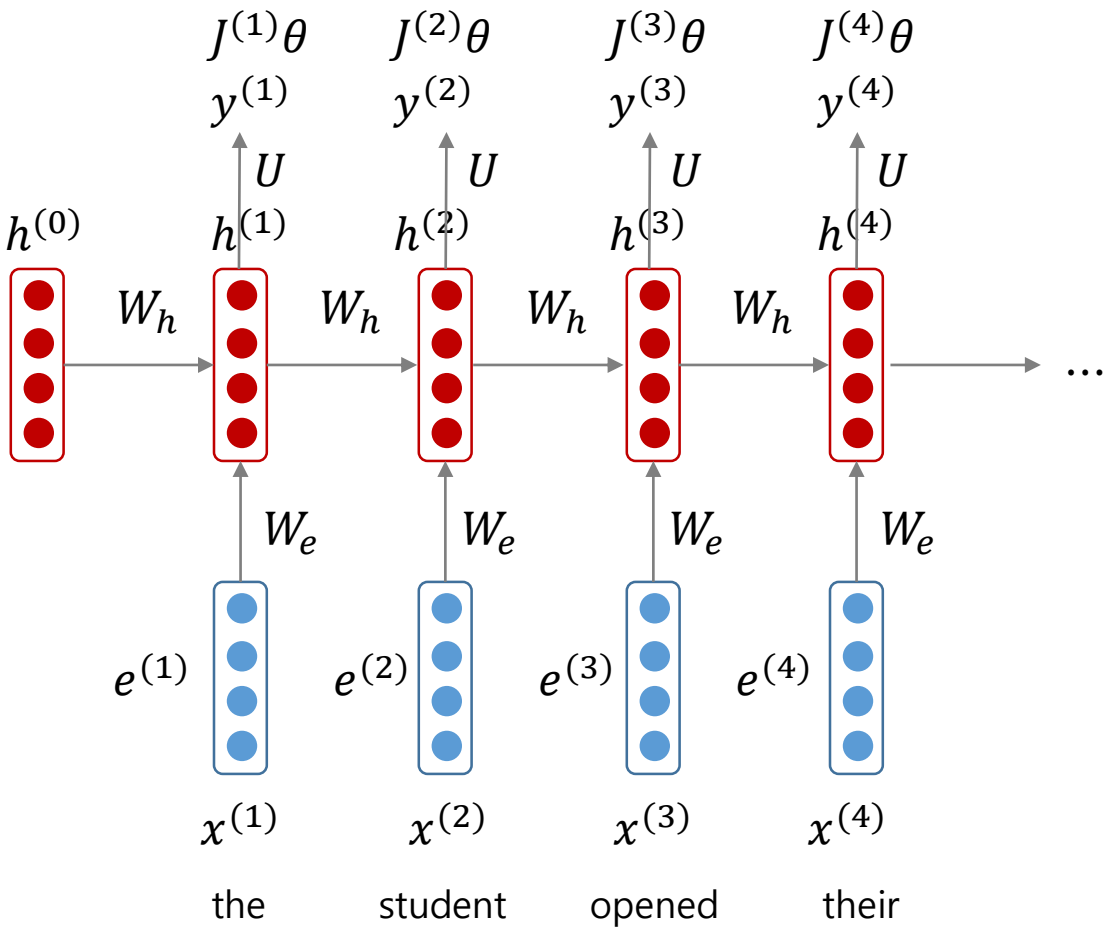
- RNN 은 이전 단계의 hidden states 와 현재 단계의 input vector 를 합쳐 현재 단계의 hidden layer 의 input 으로 이용합니다.
  - 이전까지의 단어에 의한 맥락과 현재 단어의 정보를 모두 이용하여 현재의 맥락을 표현합니다.

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

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

Recurrent neural network based language model

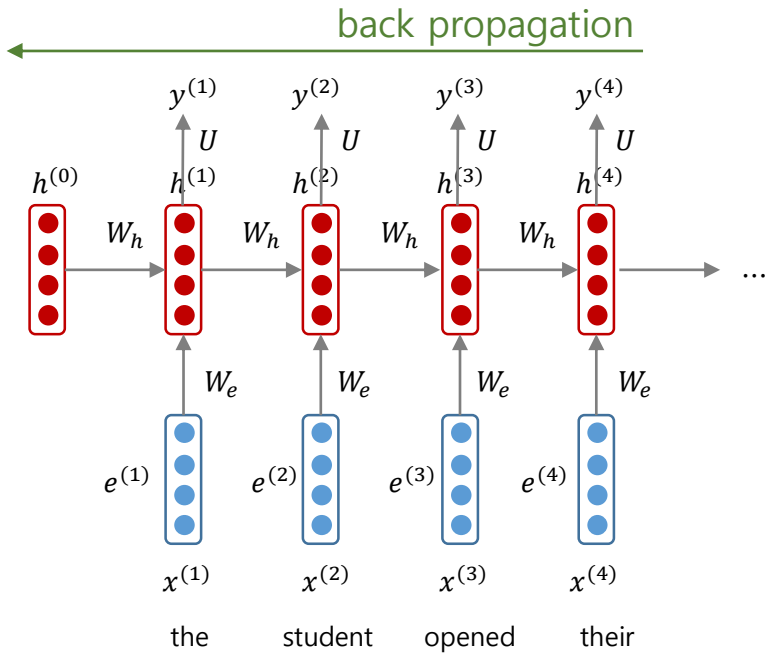
- RNN 의 loss 는 outputs 의 모든 loss 의 합입니다.



$$J(\theta) = \frac{1}{T} \sum_{t=1 \; to \; T} J^{(t)}\theta$$

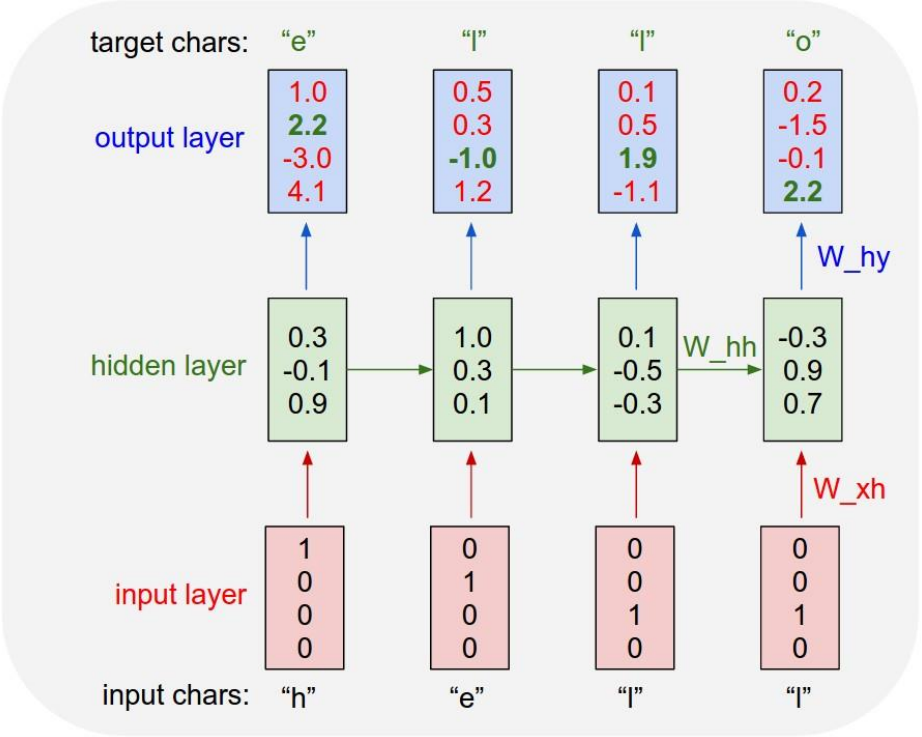
Recurrent neural network based language model

- 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.

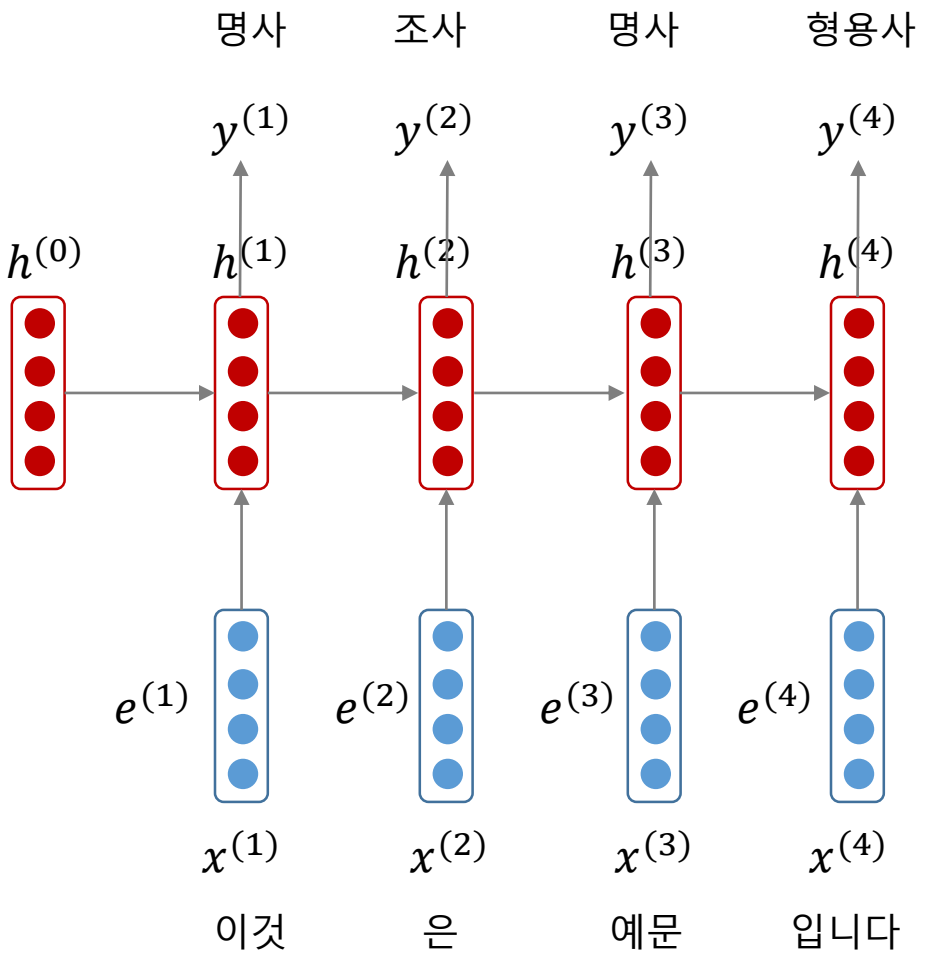


'hello' 를 출력하는 language model [1]

[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).

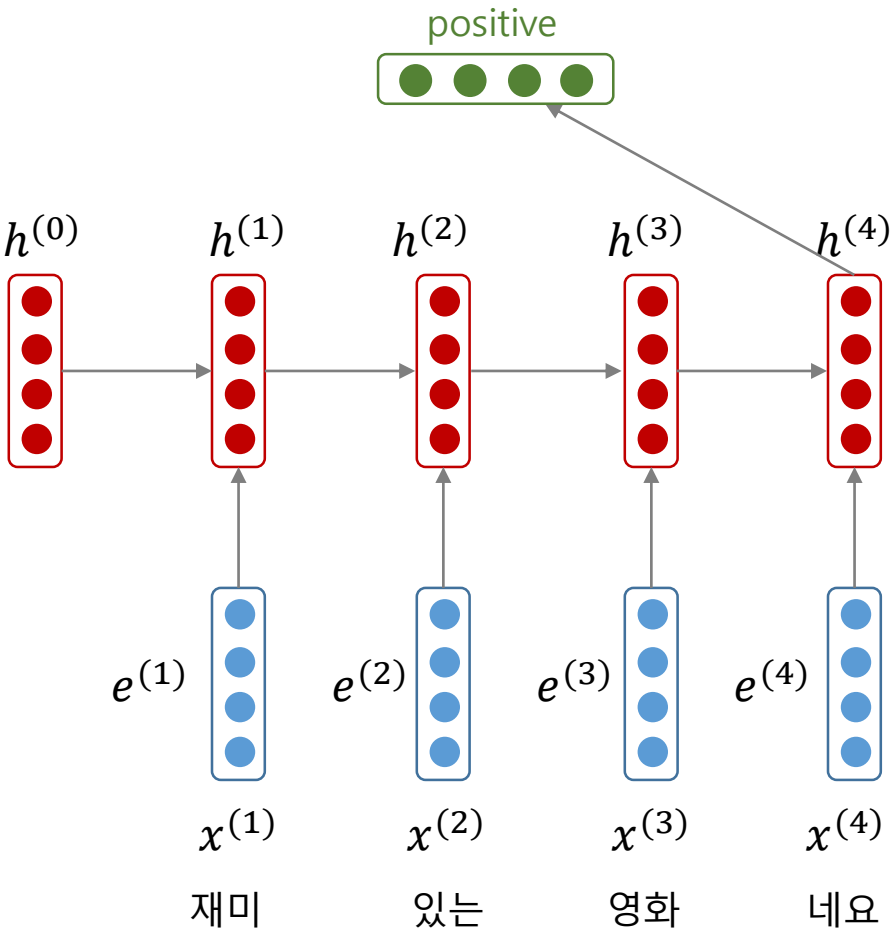
Recurrent neural network based tagging

- RNN 은 sequential labeling 에 이용될 수 있습니다.



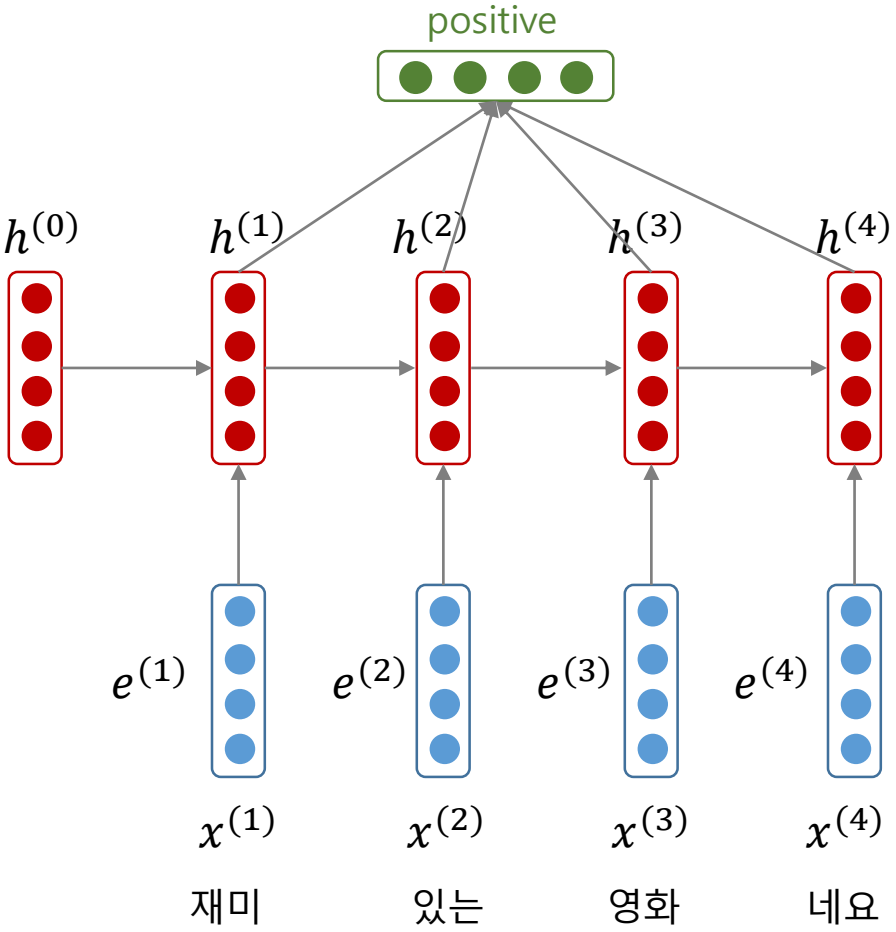
Recurrent neural network based sentence classification

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



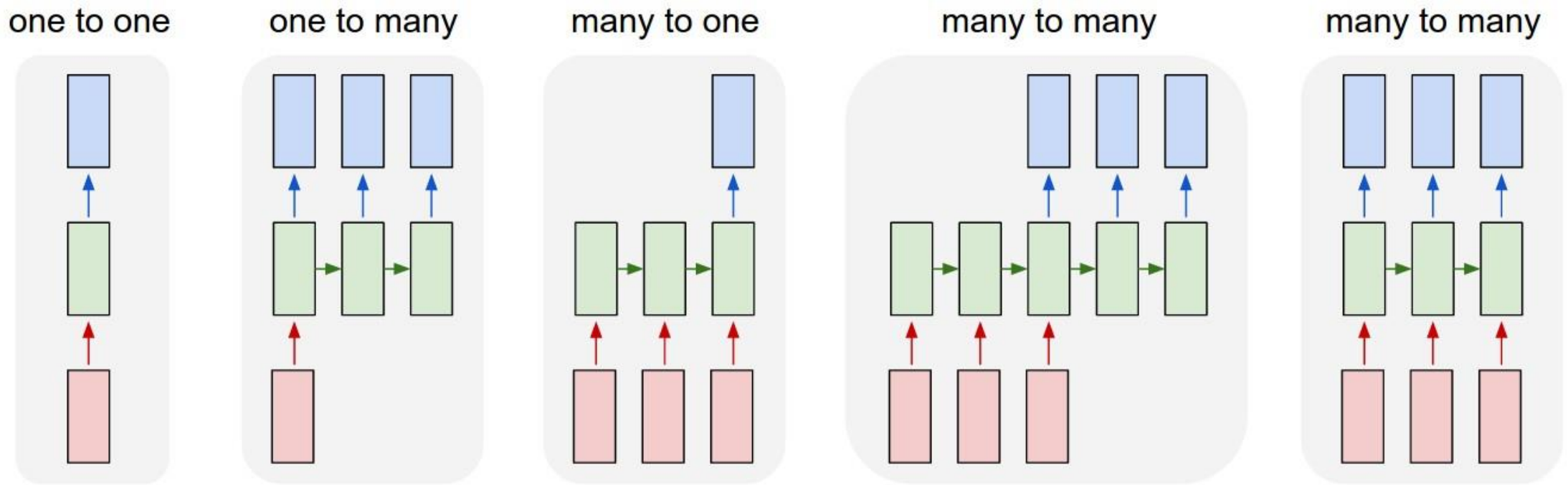
Recurrent neural network based sentence classification

- 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.



http://karpathy.github.io/2015/05/21/rnn-effectiveness/

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

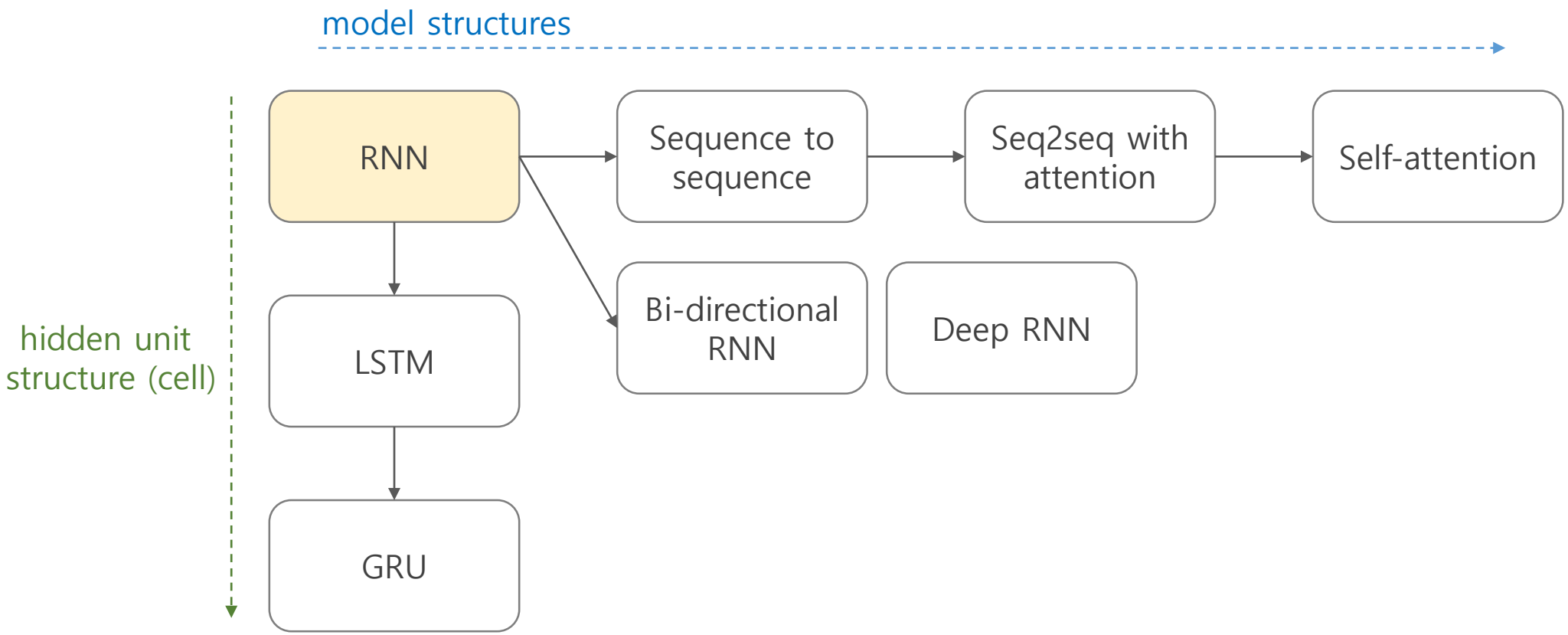
- Recurrent neural network 는 sequence 라는 텍스트 데이터의 특징을 반영한 neural network 입니다.
  - Sequence 의 길이가 가변적입니다.
  - 단어/문맥/문장의 의미는 문장에 존재하는 단어들을 모두 고려해야 합니다.
  - 한정된 windows 안의 정보만을 이용하는 feed-forward neural network 보다 더 자유롭게 정보를 이용할 수 있습니다.

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.

$$y^{(6)} = g\left(Uh^{(6)}\right)$$

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

$$\dots$$

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

- 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*.

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.

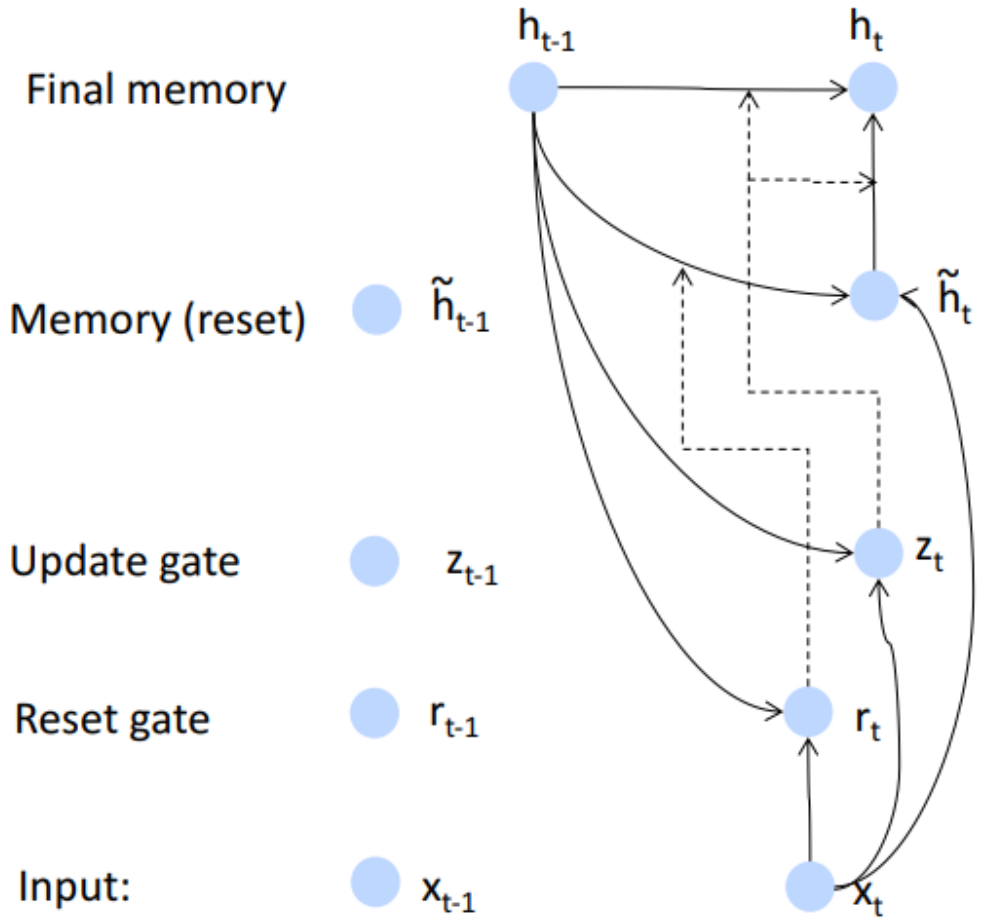
Gated Recurrent Unit (GRU)

- 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 : $z_t = \sigma\left(W^{(z)} \cdot x_t + U^{(z)} \cdot h_{t-1}\right)$
  - reset gate : $r_t = \sigma\left(W^{(r)} \cdot x_t + U^{(r)} \cdot h_{t-1}\right)$
  - new memory content : $\widetilde{h_t} = \tanh(W \cdot x_t + r \circ U \cdot h_{t-1})$
  - final memory : $h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \widetilde{h_t}$

Gated Recurrent Unit (GRU)



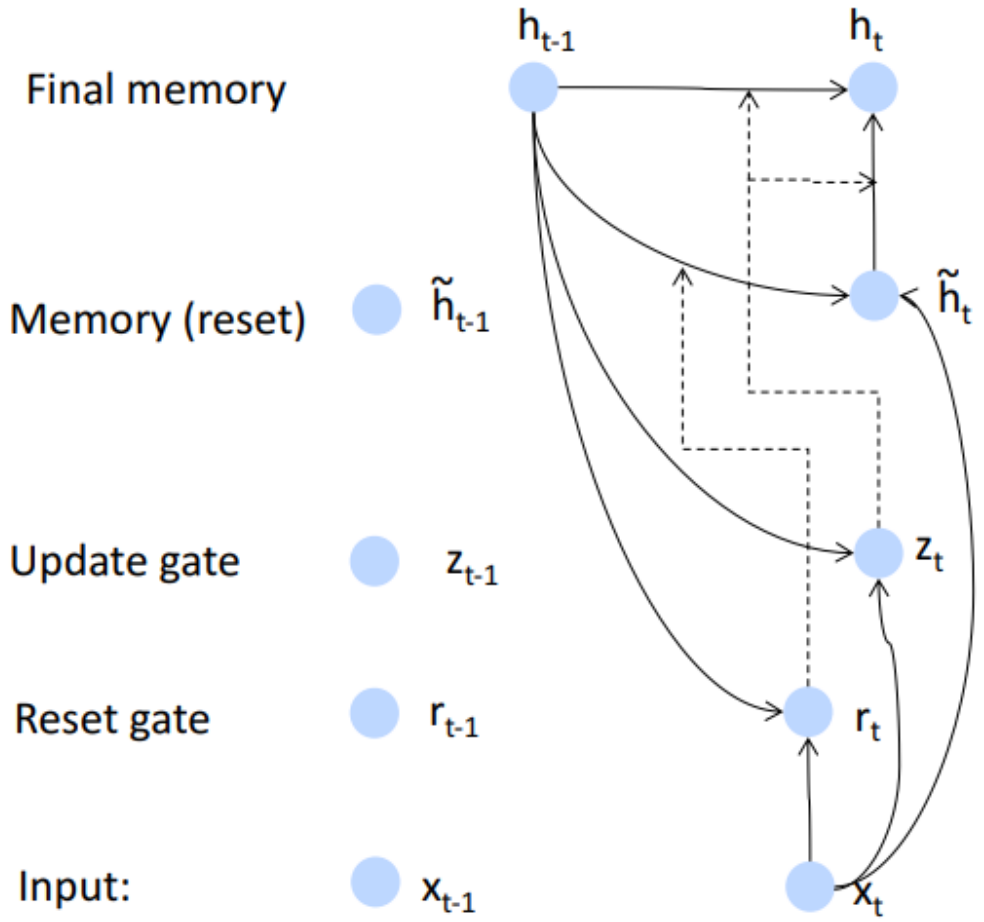
$$z_t = \sigma\left(W^{(z)} \cdot x_t + U^{(z)} \cdot h_{t-1}\right)$$

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

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

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

Gated Recurrent Unit (GRU)



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

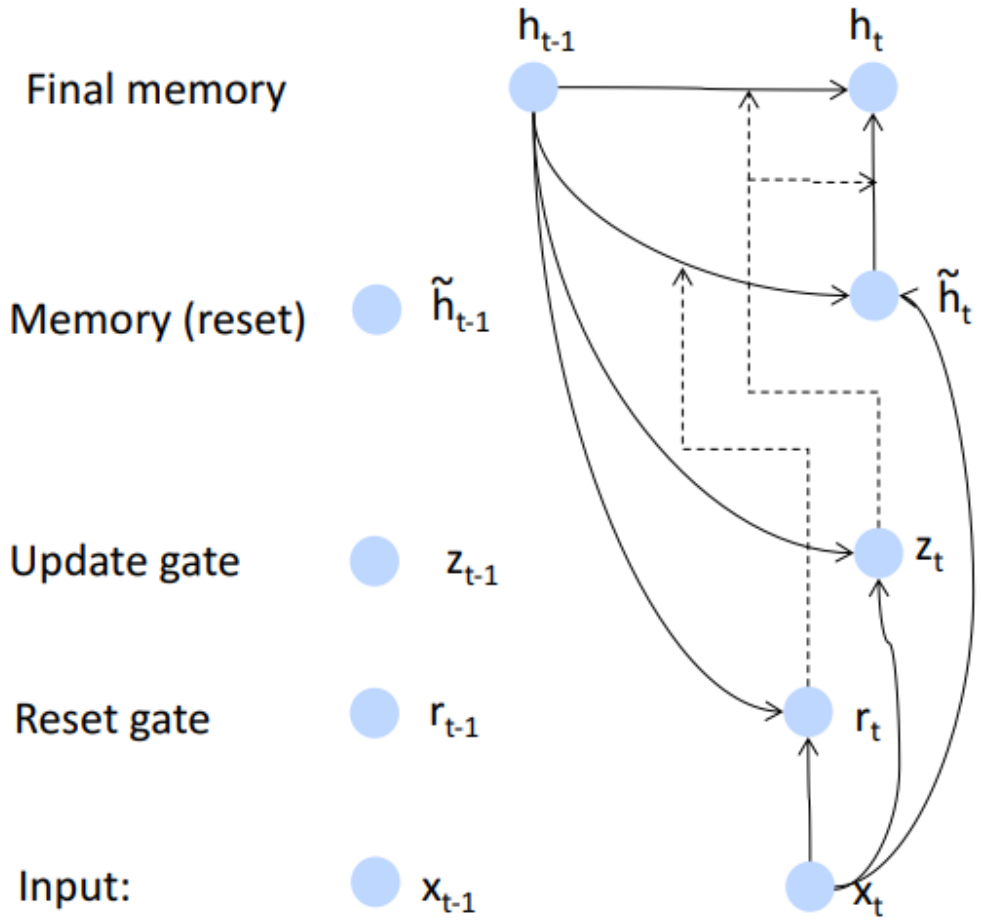
$$r_t = \sigma\left(W^{(r)} \cdot x_t + U^{(r)} \cdot h_{t-1}\right)$$

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

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

- 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)



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

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

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

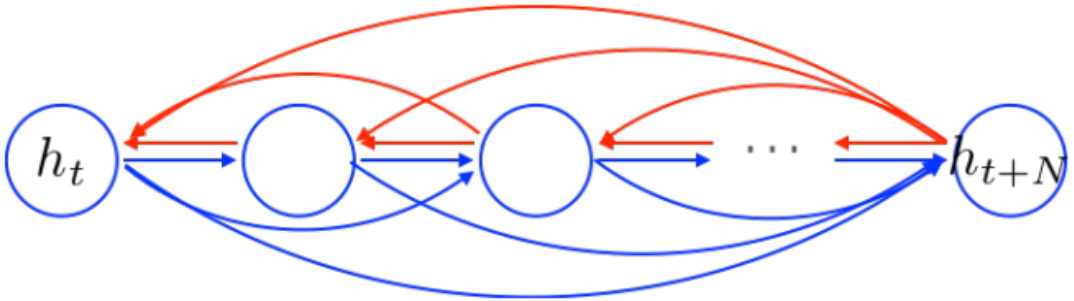
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h_t}$$

- 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.

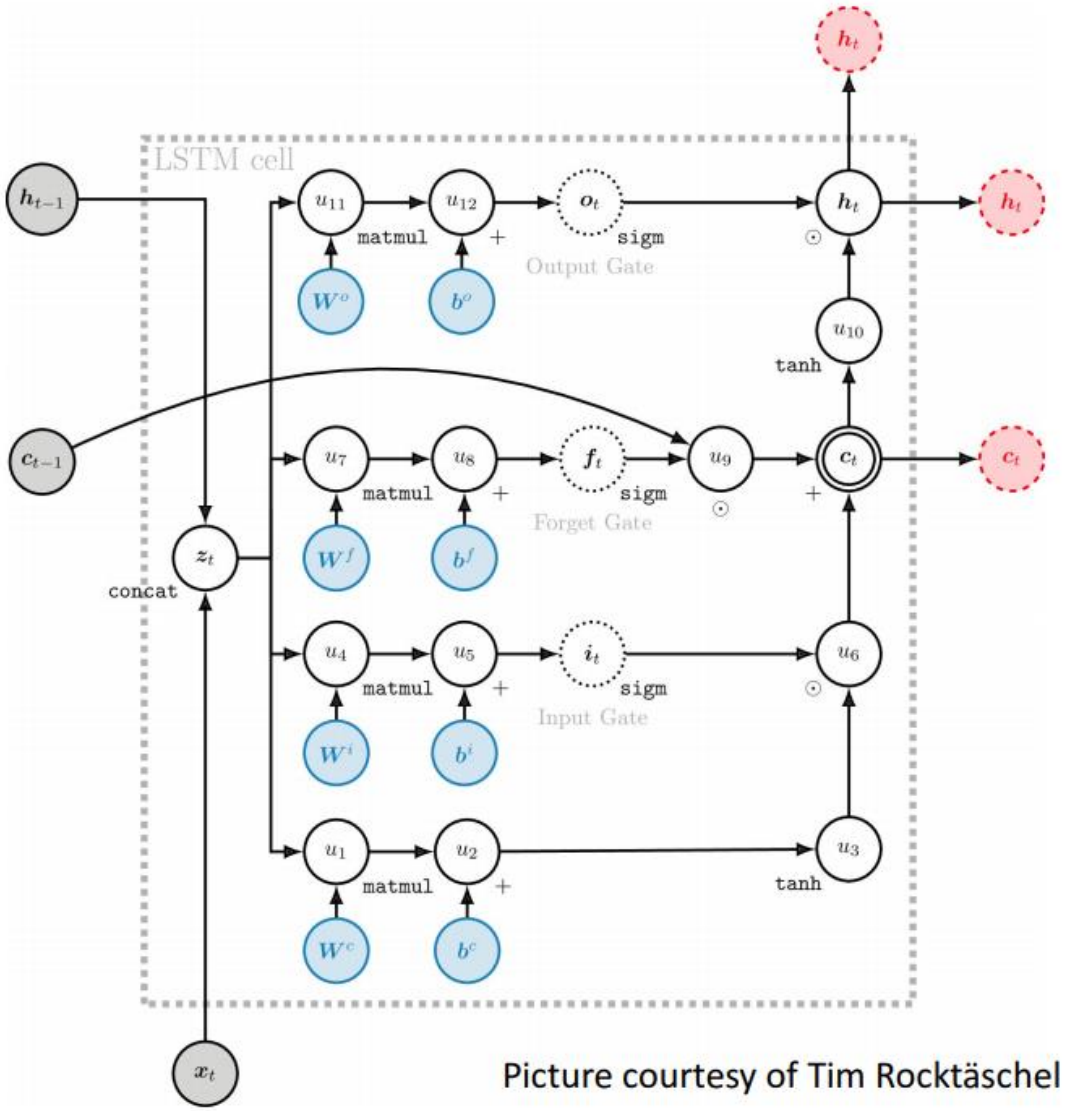
Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780.

Long Short-Term Memory (LSTM)

- LSTM 은 세 가지 gates 로 이뤄져 있습니다.
  - input gate : $i_t = \sigma\left(W^{(i)} \cdot x_t + U^{(i)} \cdot h_{t-1}\right)$
  - forget gate : $f_t = \sigma\left(W^{(f)} \cdot x_t + U^{(f)} \cdot h_{t-1}\right)$
  - output gate : $o_t = \sigma\left(W^{(o)} \cdot x_t + U^{(o)} \cdot h_{t-1}\right)$

  - new memory cell content : $\widetilde{c_t} = \tanh\left(W^{(c)} \cdot x_t + U^{(c)} \cdot h_{t-1}\right)$
  - final memory cell : $c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c_t}$
  - final hidden state : $h_t = o_t \circ \tanh(c_t)$

Long Short-Term Memory (LSTM)



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

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

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

$$\widetilde{c_t} = \tanh\left(W^{(c)} \cdot x_t + U^{(c)} \cdot h_{t-1}\right)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \widetilde{c_t}$$

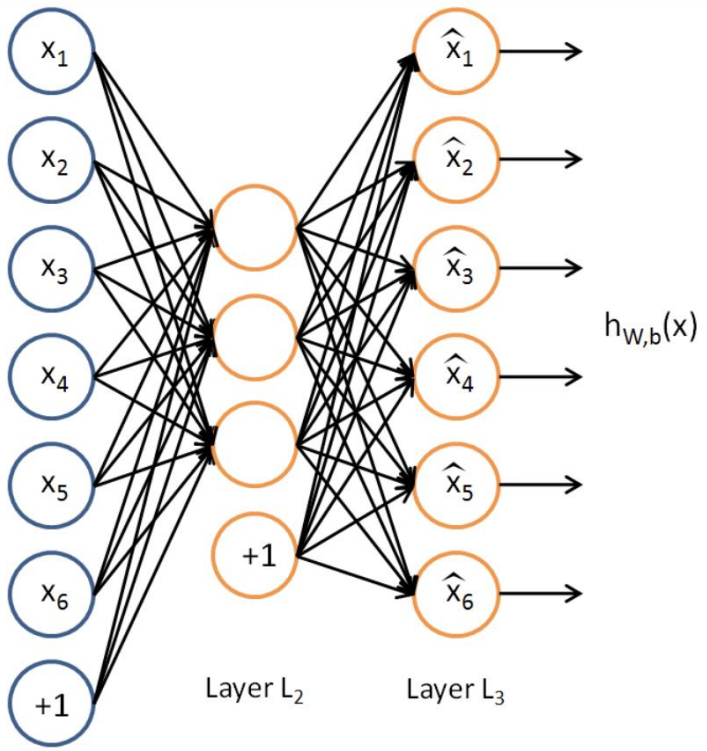
$$h_t = o_t \circ \tanh(c_t)$$

http://web.stanford.edu/class/cs224n/

Structures

Encoder – decoder

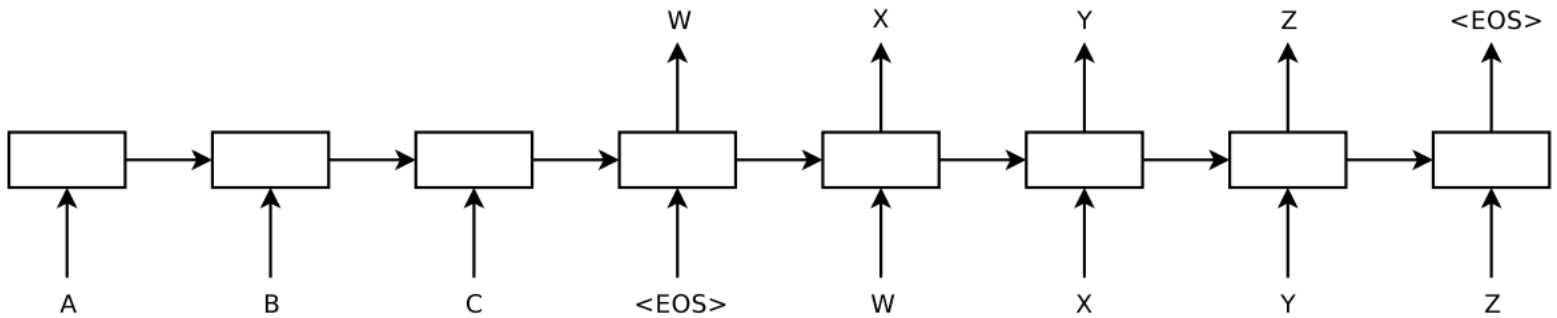
- Autoencoder 는 벡터 형식의 input 에 대한 encoder – decoder 입니다.



http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/

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.

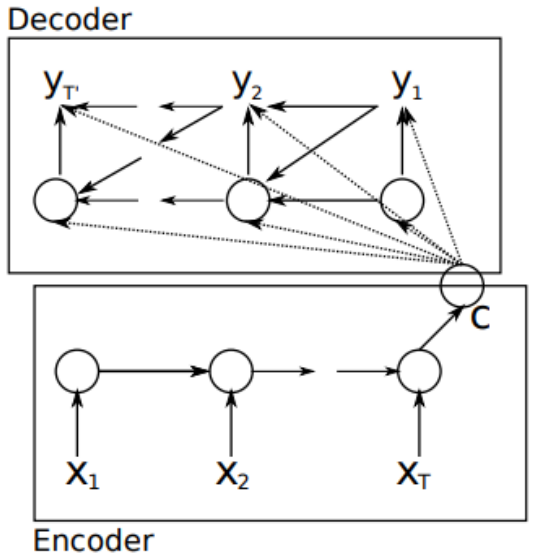
Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Advances in neural information processing systems (pp. 3104-3112).

Sequence to sequence

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

Sequence to sequence

- Encoder 의 output 은 context 역할을 합니다.
  - $p(y \mid x) = \prod_i p(y_i \mid y_{1:i-1}, x) = \prod_i p(y_i \mid y_{1:i-1}, c)$



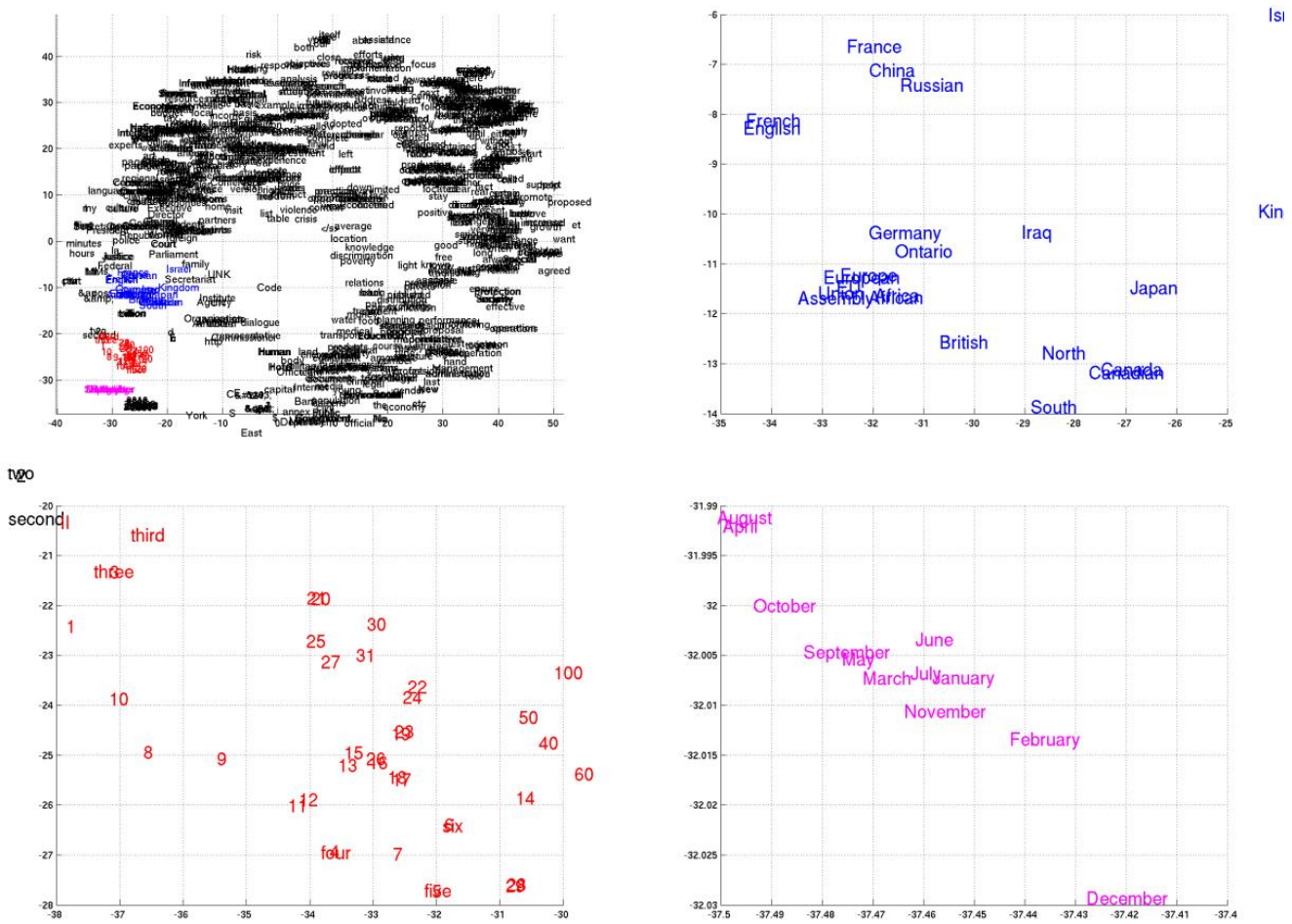
Decoder 는 벡터 c 로 표현된 문장 x 와 이전에 생성된 단어열 $y_{1:i-1}$ 을 이용하여 y_i 를 생성합니다.

Encoder 는 문장 x 를 벡터 c 로 표현합니다.

An illustration of the proposed RNN Encoder–Decod

- 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 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)

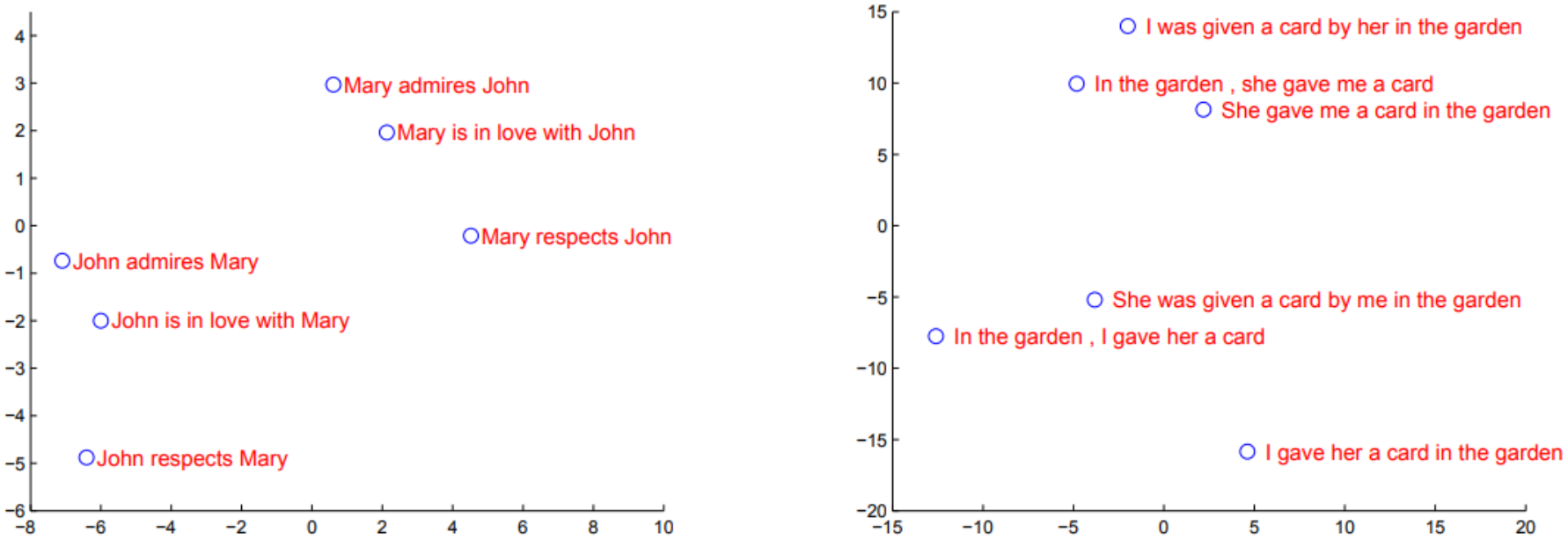
- Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

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*.

Sequence to sequence



The figure shows a 2-dimensional PCA projection of the LSTM hidden states that are obtained after processing the phrases in the figures.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Advances in neural information processing systems (pp. 3104-3112).

Sequence to sequence with Attention

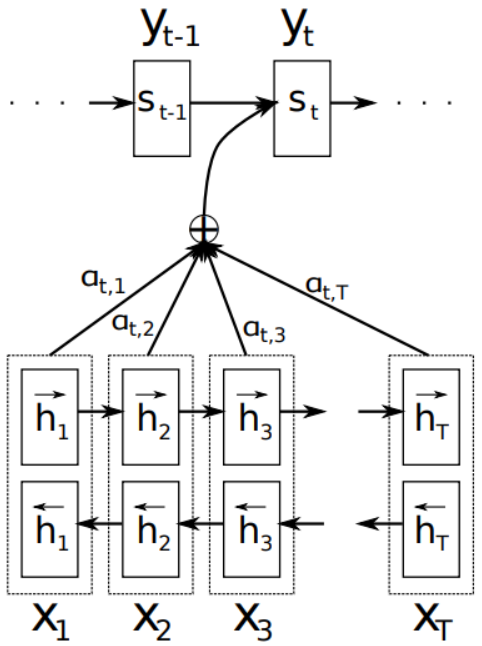
A potential issue with this encoder–decoder approach is that a neural network needs to be able to compress all the necessary information of a source sentence into a fixed-length vector.

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

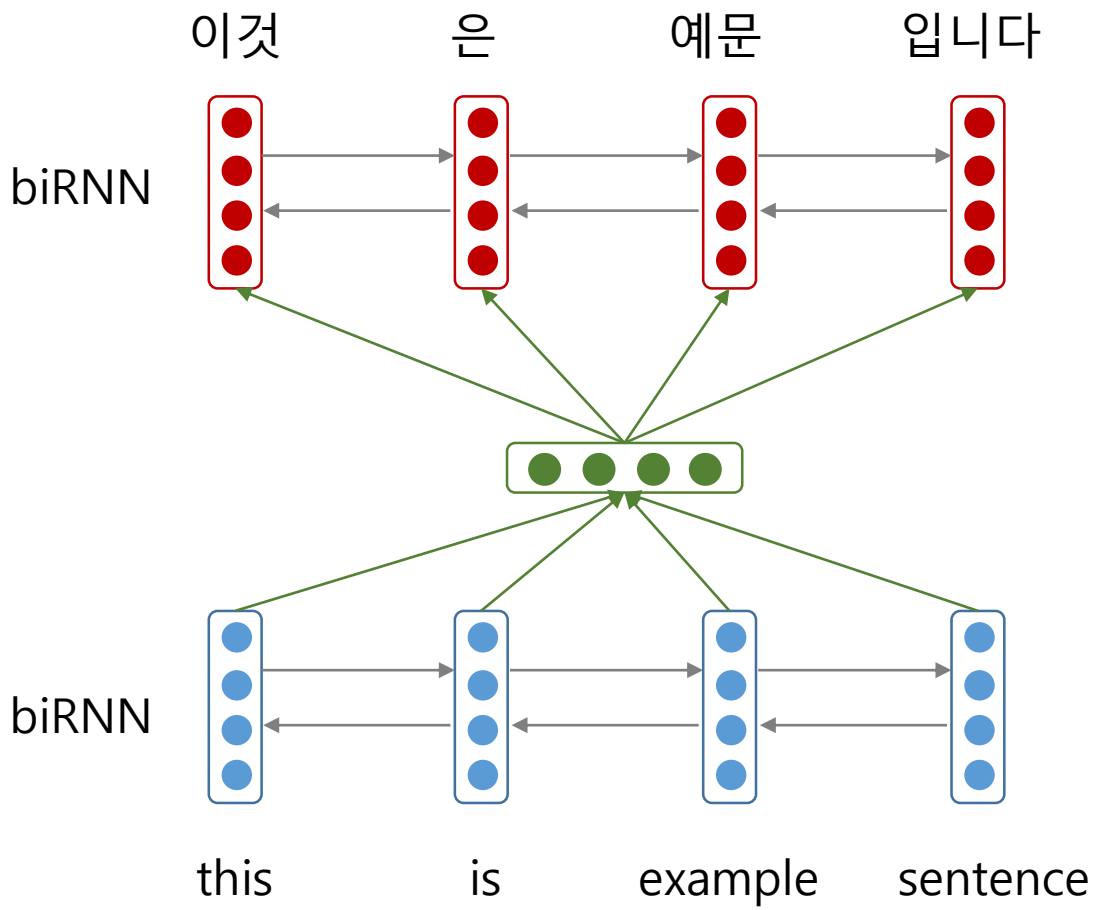


The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots x_T)$

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Sequence to sequence with Attention

without attention



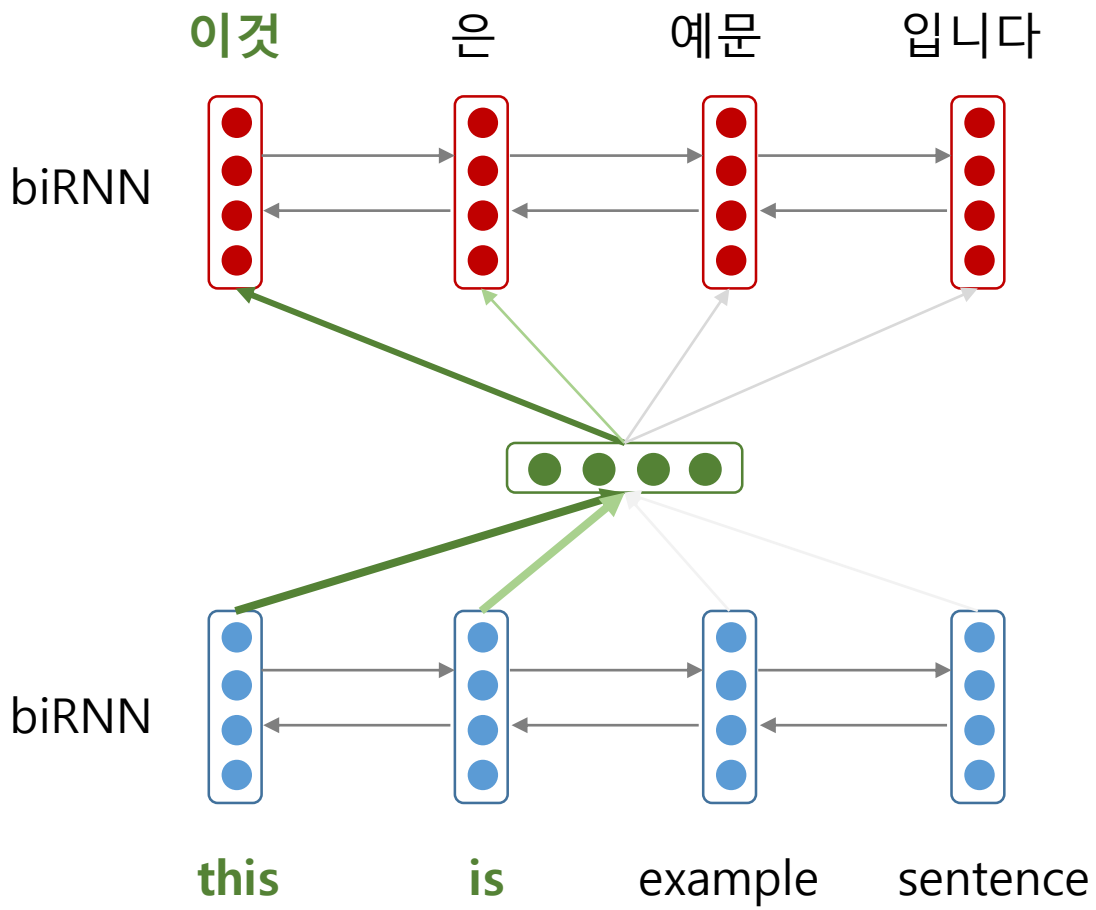
fixed context vector

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

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

Sequence to sequence with Attention

with attention



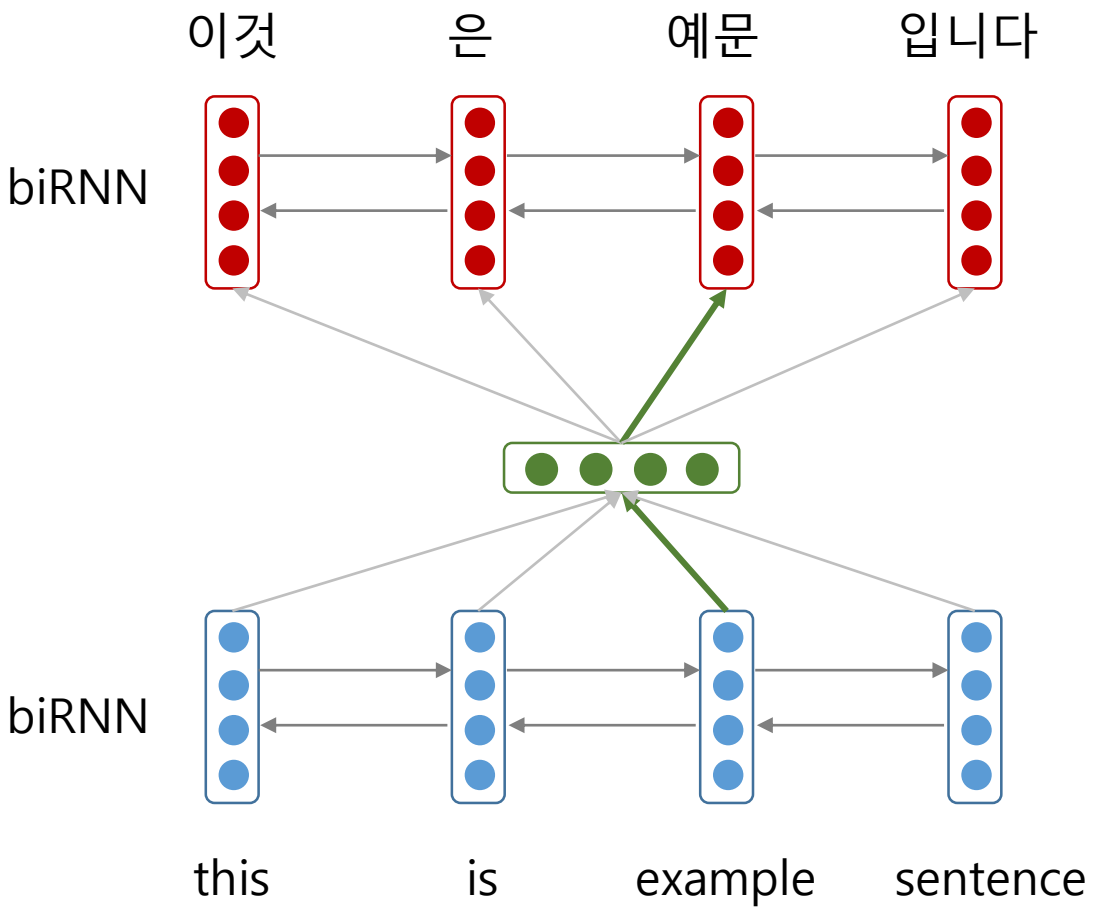
y_i dependent context vector

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

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

Sequence to sequence with Attention

with attention

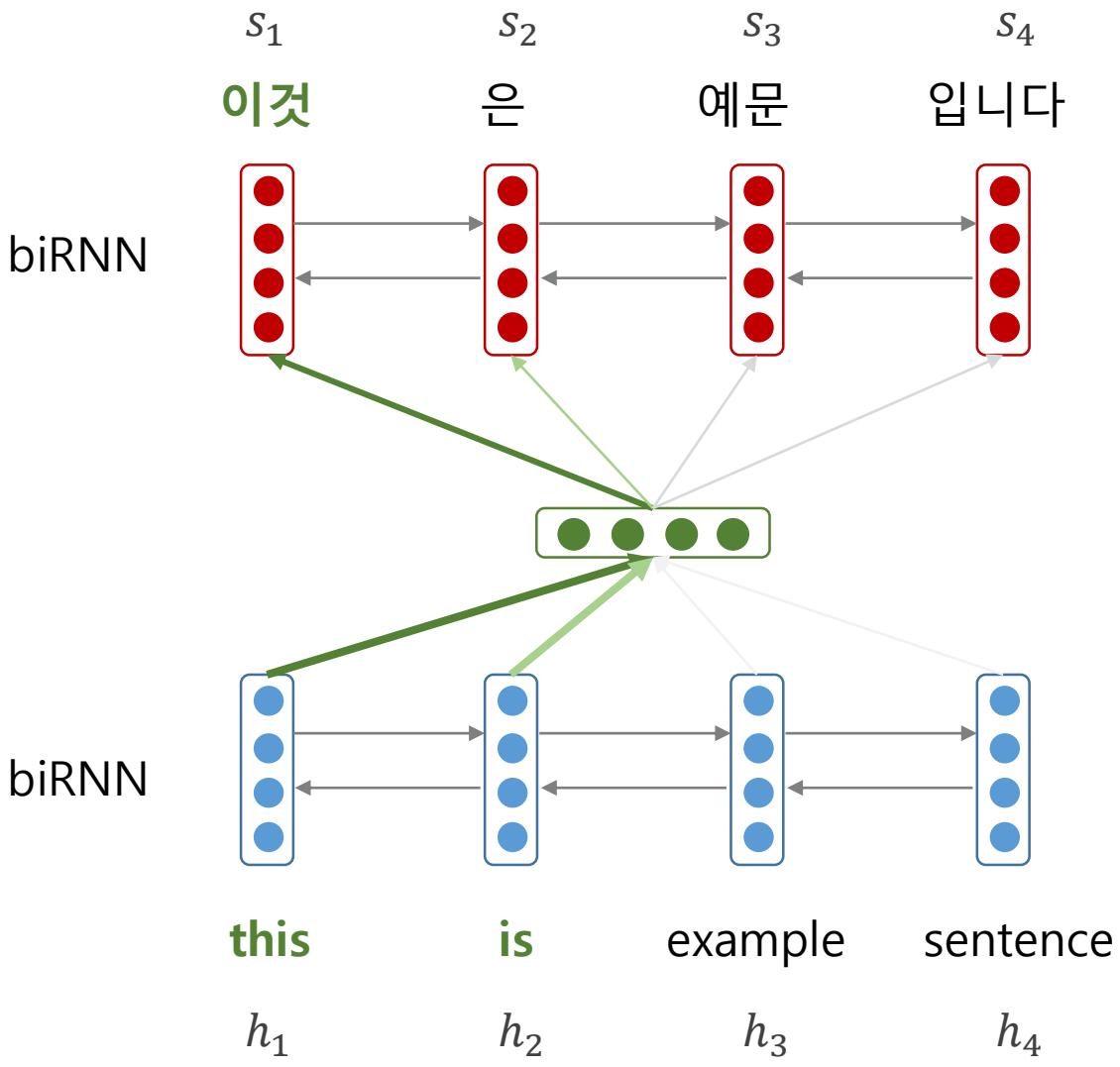


y_i dependent context vector

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

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

Sequence to sequence with Attention



y_i dependent context vector

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

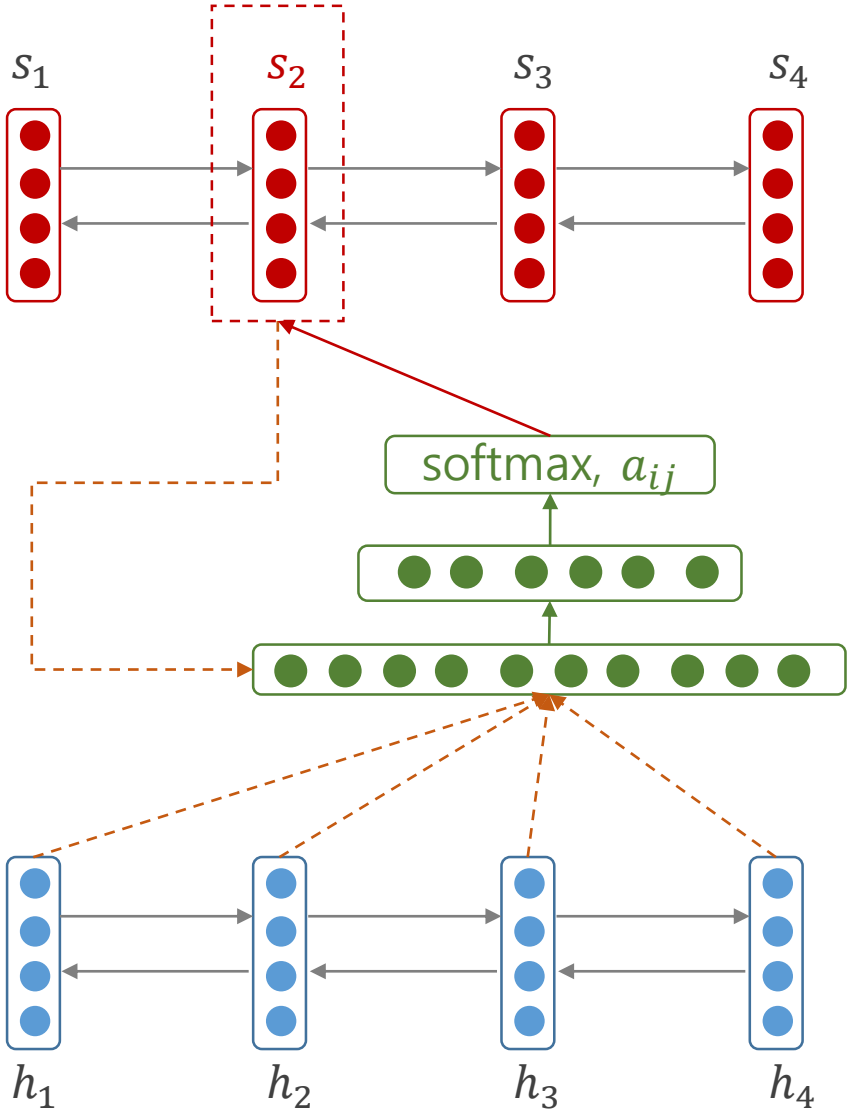
$$s_i = f(s_{i-1}, y_{i-1}, \boldsymbol{c_i})$$

$$\boldsymbol{c_i} = \sum_{j=1\ to\ T_x} \alpha_{ij} h_j$$

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

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

Sequence to sequence with Attention

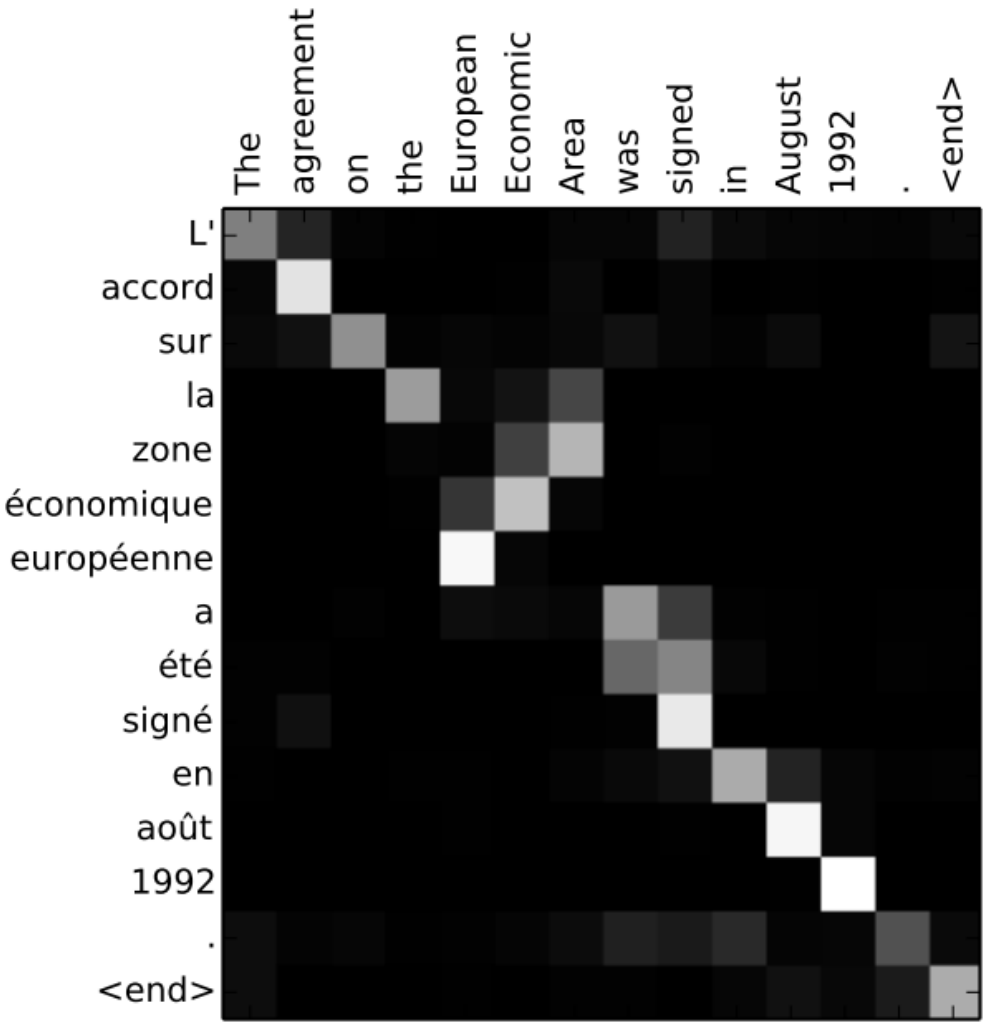


Input of feed-forward (attention) net.

Attention

Sequence to sequence with Attention

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



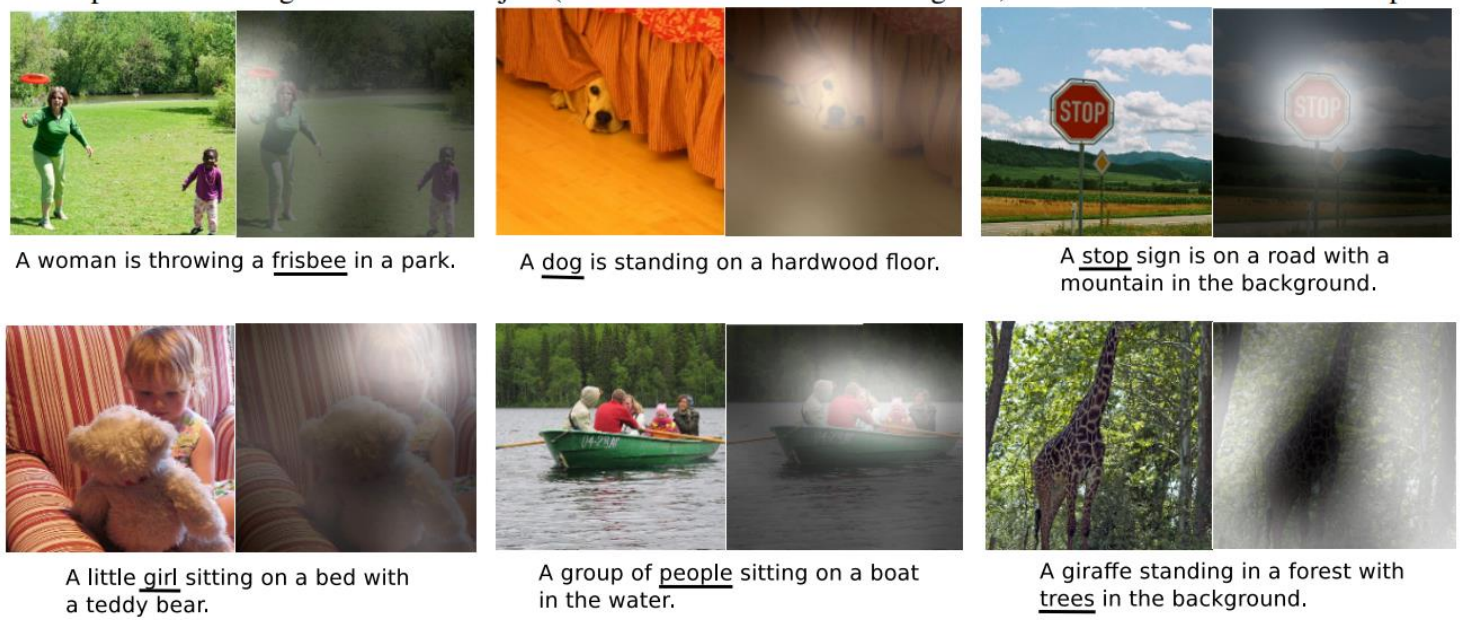
Four sample alignments found by RNNsearch-50. The x-axis and y-axis of each plot correspond to the words in the source sentence (English) and the generated translation (French), respectively. Each pixel shows the weight α_{ij} of the annotation of the j-th source word for the i-th target word

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Attention in image captioning

- Image captioning 에서도 attention 은 이용됩니다.
  - 모델의 해석에도 이용됩니다.

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

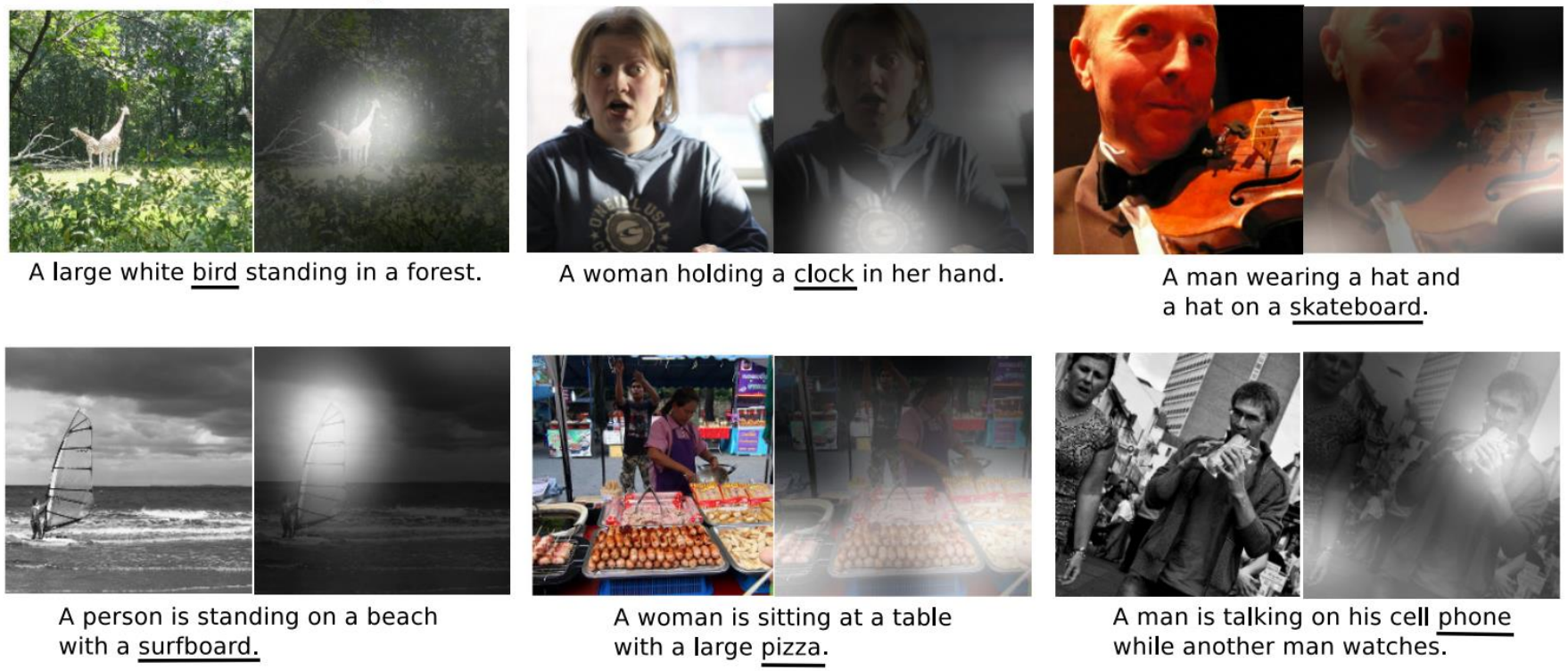


- Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., ... & Bengio, Y. (2015, June). Show, attend and tell: Neural image caption generation with visual attention. In International Conference on Machine Learning (pp. 2048-2057).

Attention in image captioning

- 물론 언제나 완벽하지는 않습니다.

Figure 5. Examples of mistakes where we can use attention to gain intuition into what the model saw.



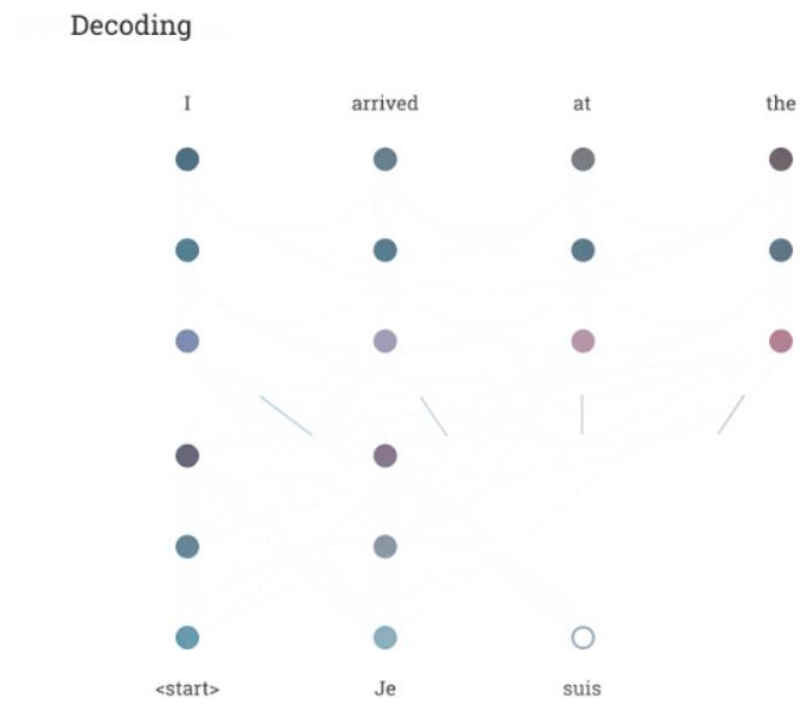
- Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., ... & Bengio, Y. (2015, June). Show, attend and tell: Neural image caption generation with visual attention. In International Conference on Machine Learning (pp. 2048-2057).

Self-attention (Attention is all you need)

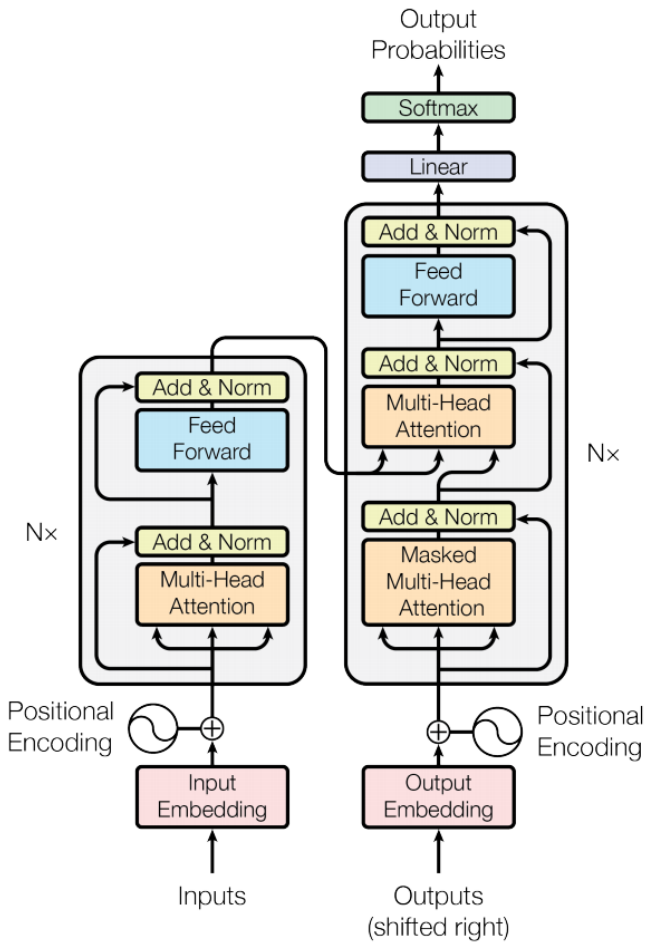
- Seq2seq + attention 에서는 encoder, decoder 는 RNN 을, attention network 는 feed-forward network 를 이용하였습니다.
- Transformer 라는 이름으로 제안된 self-attention model 에서는 encoder 와 decoder 까지 feed-forward network 를 이용합니다.
  - Encoding word/sentence → “Encoding meaning”
  - Sequential 하게 hidden 을 계산하는 것과 달리 feed-forward network 는 병렬화가 가능합니다.

- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*(pp. 6000-6010).

Self-attention (Attention is all you need)



Animation from google research blog

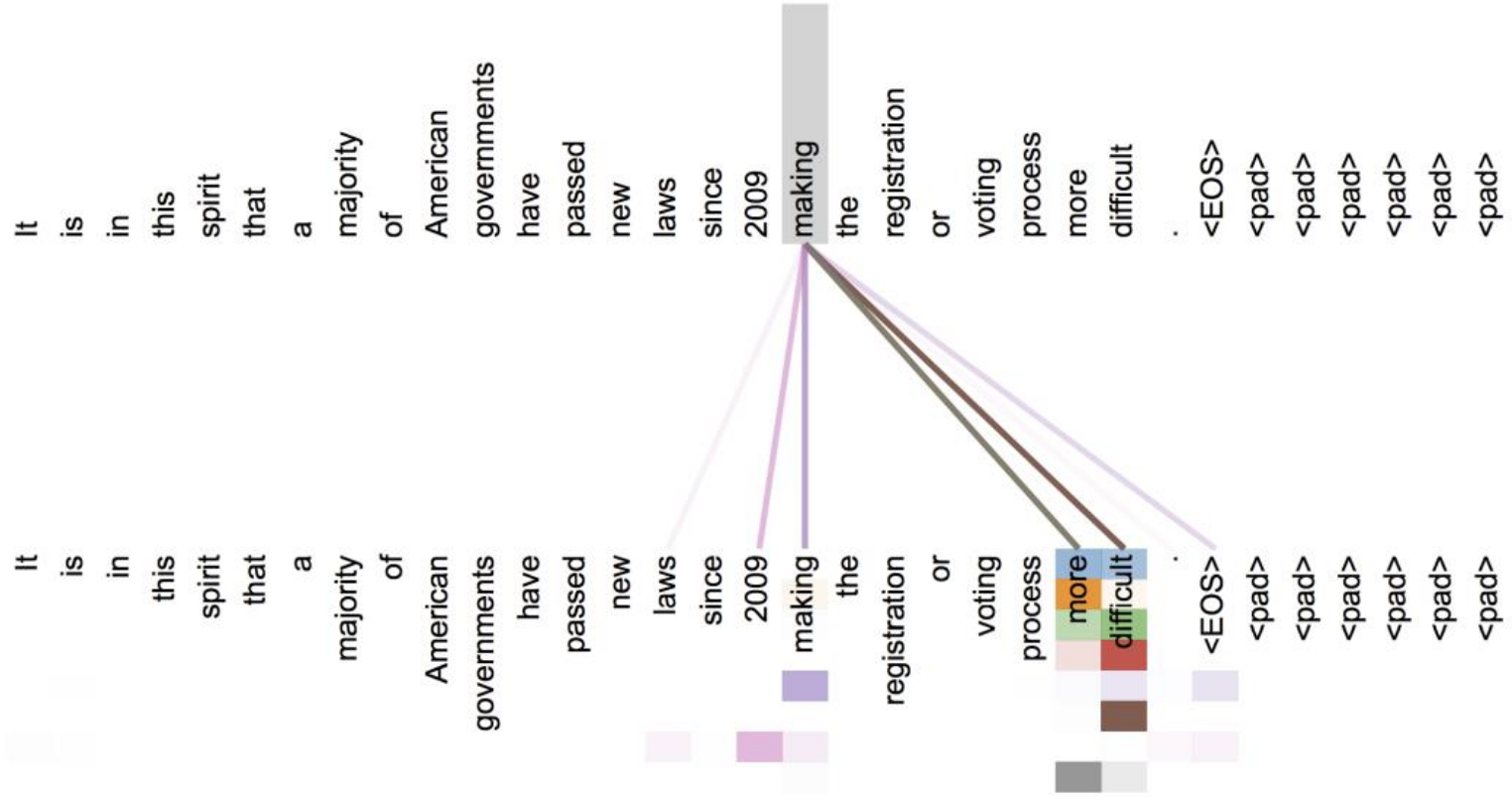


Transformer model architecture

[1] https://research.googleblog.com/2017/08/transformer-novel-neural-network.html
[2] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*(pp. 6000-6010).

Self-attention (Attention is all you need)

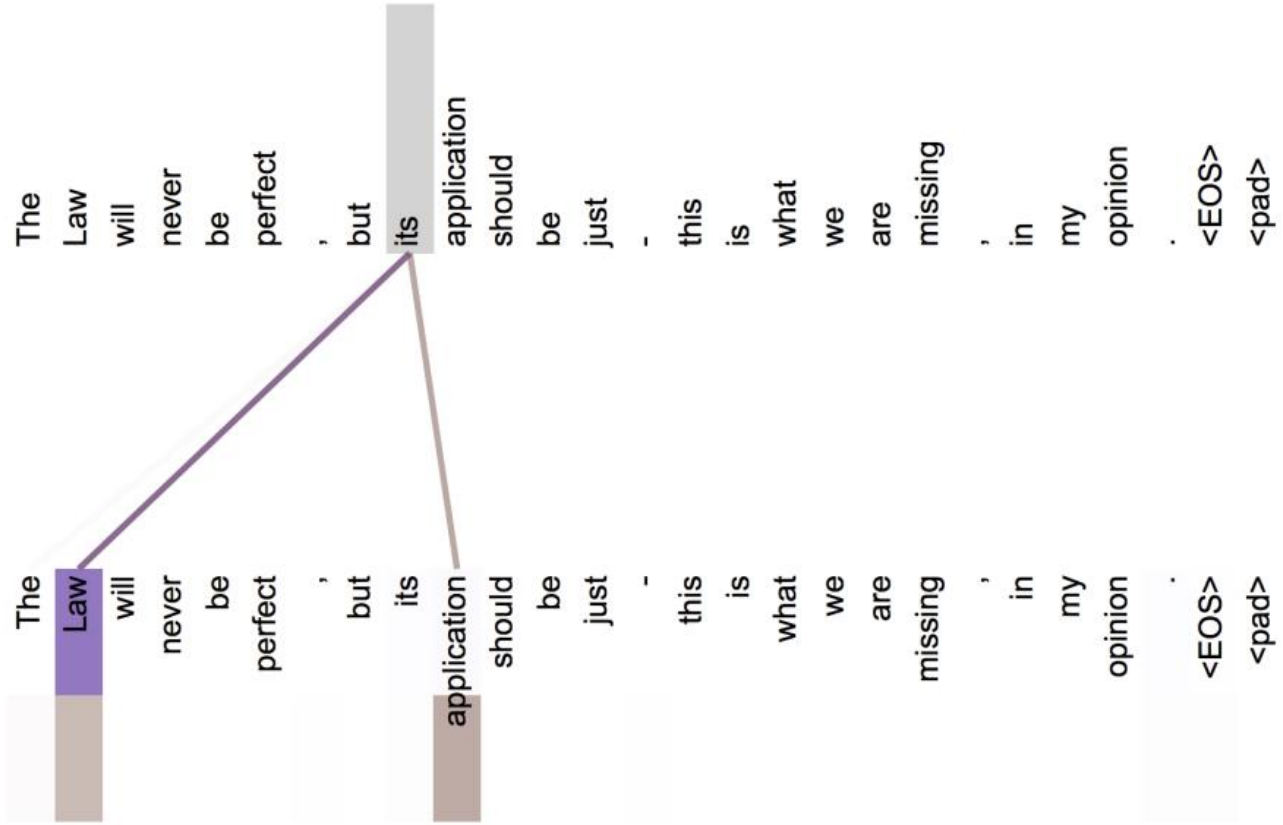
- 서로 상관이 있는 단어들이 높은 attention 을 지닙니다.



http://web.stanford.edu/class/cs224n/

Self-attention (Attention is all you need)

- 서로 상관이 있는 단어들이 높은 attention 을 지닙니다.



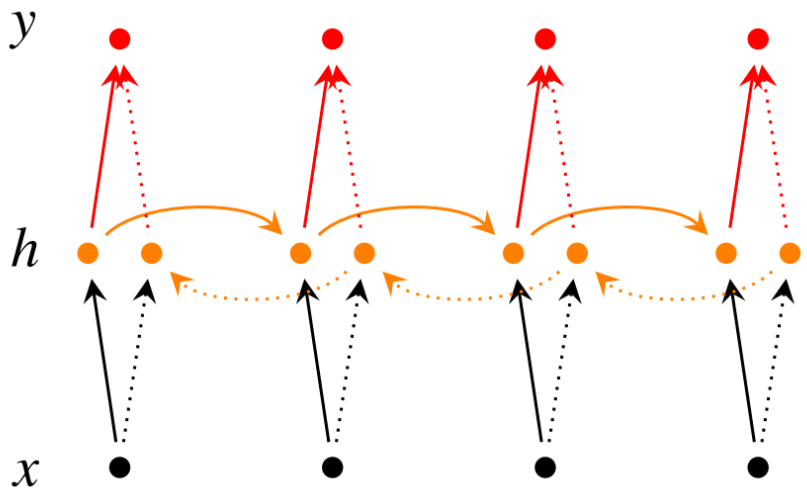
http://web.stanford.edu/class/cs224n/

Self-attention (Attention is all you need)

- Attention model 은 end – to – end 로 word embedding, attention mechanism 까지 모두 데이터 기반으로 학습한다는 장점이 있습니다.

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.

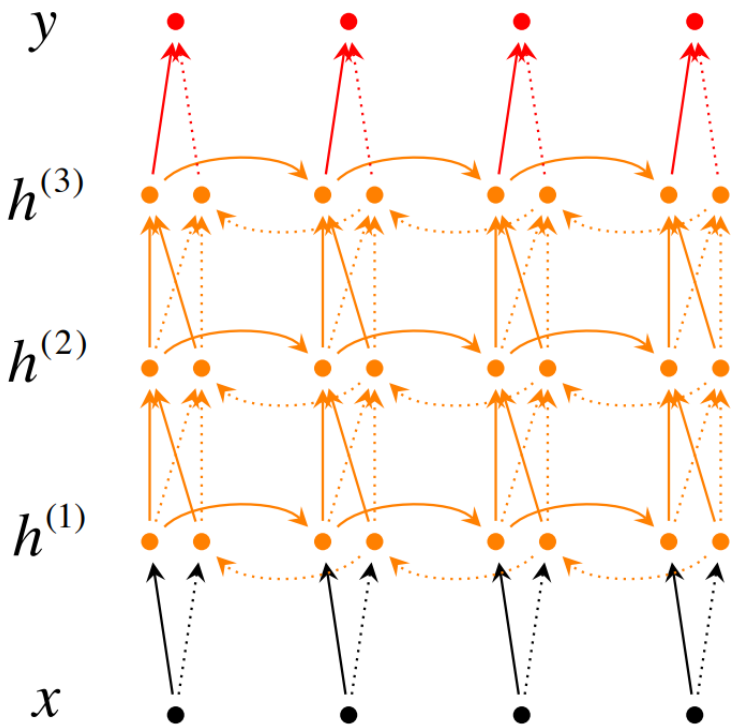


$$\overrightarrow{h_t} = f\left(\overrightarrow{W} \cdot x_t + \overrightarrow{V} \cdot \overrightarrow{h_{t-1}}\right)$$
$$\overleftarrow{h_t} = f\left(\overleftarrow{W} \cdot x_t + \overleftarrow{V} \cdot \overleftarrow{h_{t-1}}\right)$$

$$y_t = g\left(U\left[\overrightarrow{h_t}; \overleftarrow{h_t}\right]\right)$$

Deep RNN

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



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

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

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

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
- Alex Graves Ph.D thesis, https://www.cs.toronto.edu/~graves/phd.pdf