딥 러 닝 세 미 나

ToBig's 7기 최희정

Deep NLP

Natural Language Processing with Deep Learning

Contents

Unit 01 | NLP Task

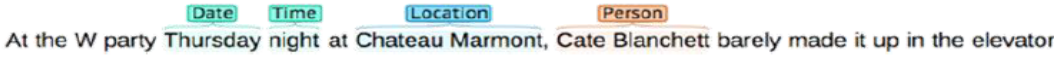
최근 20년간 자연어 처리 문제

응용

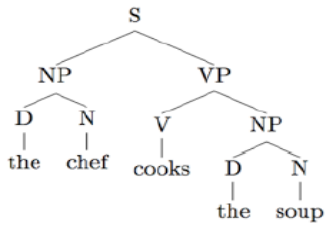
- 문장내에서 '이해하기 '
- 문장외에서 정보 '찾기'
- Q/A
- 번역
- 감성분석
- 주어복원
- 주제분류
- 검색
- 자연어이해
- ...
- ...
- 대화

NLP Stack

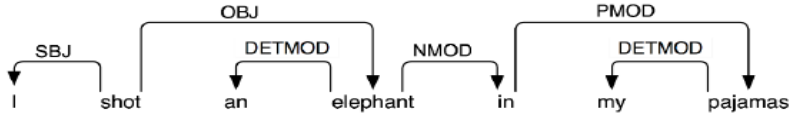
- 개체명분석



- 구문분석



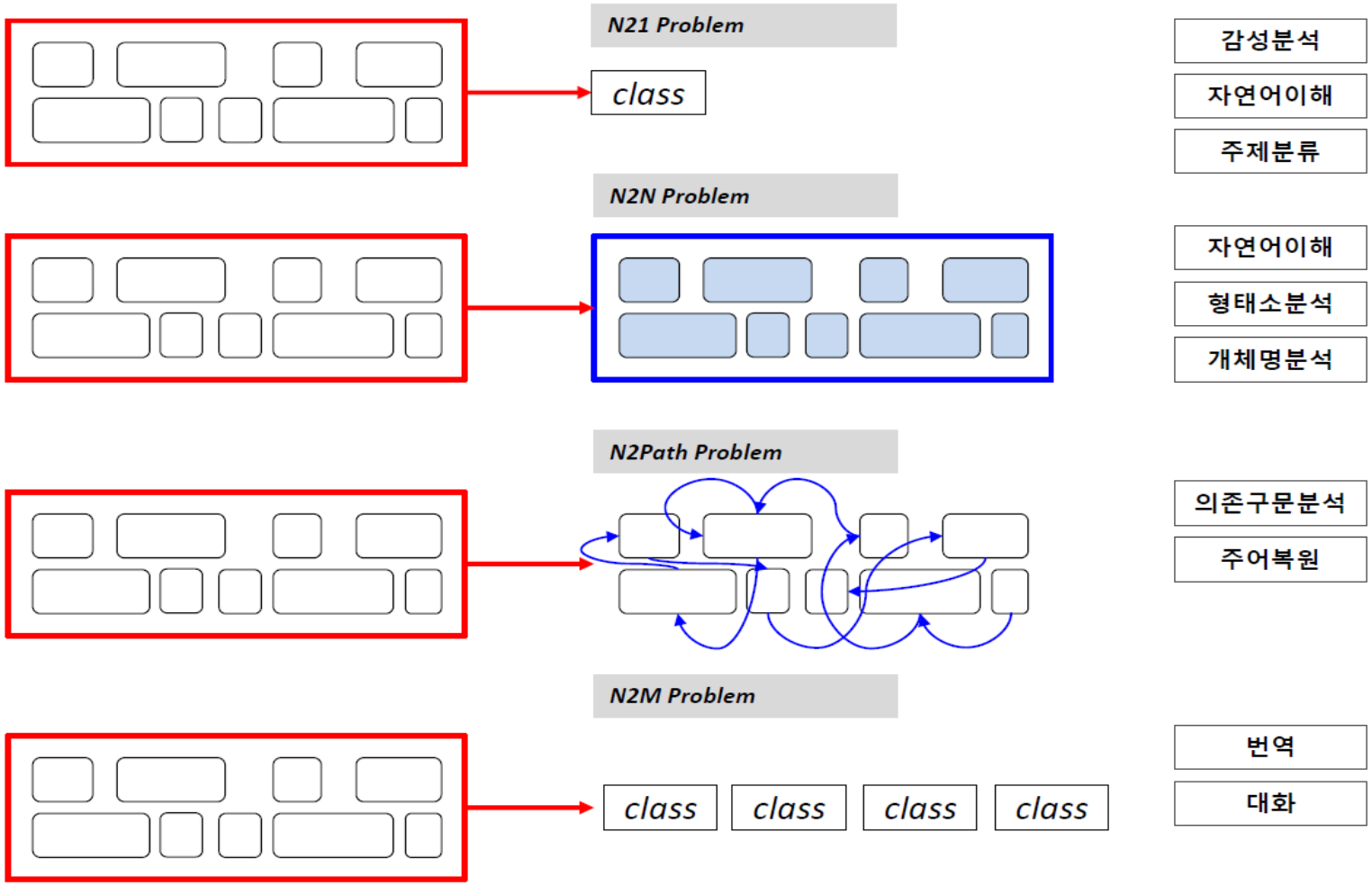
- 의존구문분석



- 형태소분석: Ex) 밥/NNG+먹/VV+기/ETN 싫/VA+대/EF
- 정규화: Ex) 채널 삼백번 틀어줘 → 채널 300번 틀어줘
- 띄어쓰기: Ex) 무한도전좀 보여 줘 → 무한도전 좀 보여줘

Unit 01 | NLP Task

Problem Abstraction



Unit 02 | Word Representation

word representation

: 컴퓨터가 words를 인식하기 위해 필요한 벡터화(embedding) 과정

1) Naive representation: one-hot vector

2) Predictive representation: Word2Vec

3) Task-based representation

1. Naive representation

One-hot vector

: 해당 단어의 사전상 위치에는 1, 나머지 위치에는 0을 넣어 dimension이 단어사전의 size와 동일하고, 0과 1만을 원소로 가지는 vector로 단어를 벡터화하는 방법

ex) data: [I like NLP, NLP is fun]

-> 단어사전: [I, like, NLP, is, fun] -> I = $[1\ 0\ 0\ 0\ 0]^T$, NLP = $[0\ 0\ 1\ 0\ 0]^T$

-> But, 이러한 word representation은 1)sparse하고 2)모든 단어들 간의 내적이 0이므로 3)단어간 의미상 유사도 측정이 불가능하다.

단어의 주변을 보면 그 단어를 안다.

You shall know a word by the company it keeps.

-- 언어학자 J.R. Firth (1957)

2. Predictive representation

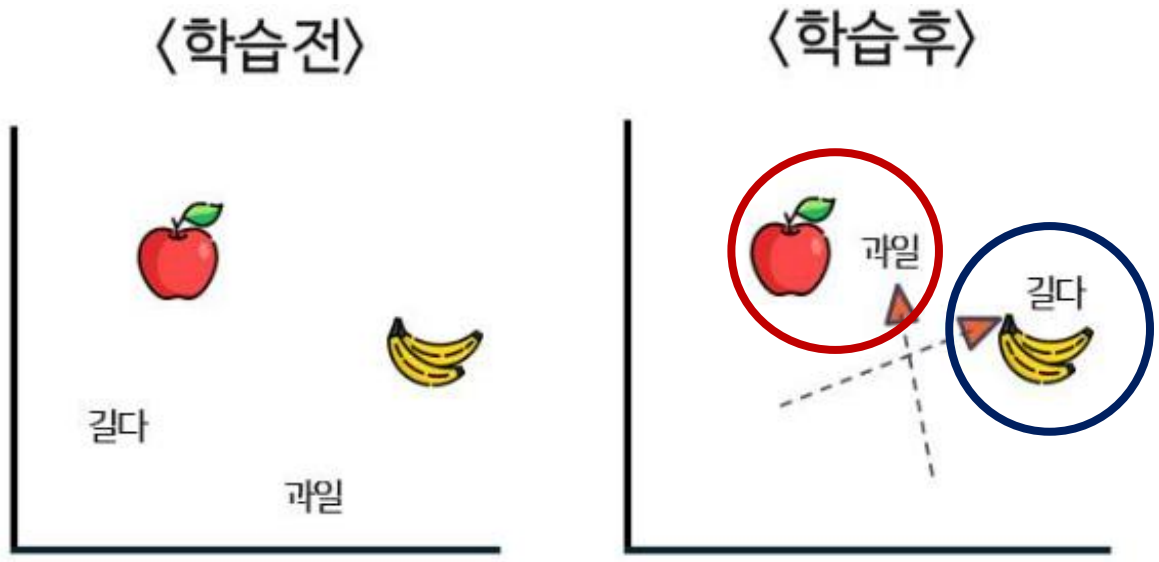
Word2Vec

: Distributional Hypothesis의 아이디어를 바탕으로 (center word, context words)을 이용해
서로를 예측하는 과정을 통해 embedding을 학습하는 Neural Network Model

1) CBOW
: center word의 앞뒤 N/2개의 context words(총 N개)를 Input으로 center word를 예측하는 model

2) Skip-gram
: center word를 input으로 center word의 앞뒤 N/2개의 context words(총 N개)를 예측하는 model

Word2Vec



-> 함께 자주 등장하는 단어들은 유사도가 높으므로 함께 자주 등장하는 단어일수록 가까이 embedding

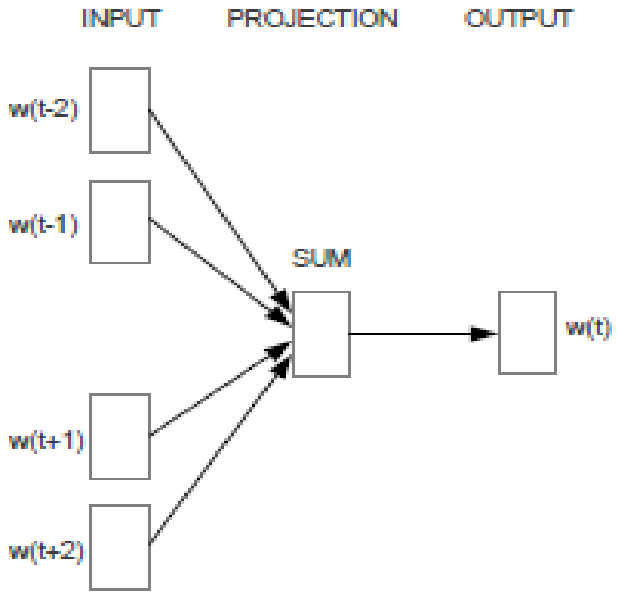
-> '사과'는 '과일'이랑 '바나나'는 '길다'랑 더 가까이 embedding

"집 앞 편의점에서 아이스크림을 사 먹었는데,
__ 시려서 너무 먹기가 힘들었다."

-> context words를 이용해 center word를 예측하자!

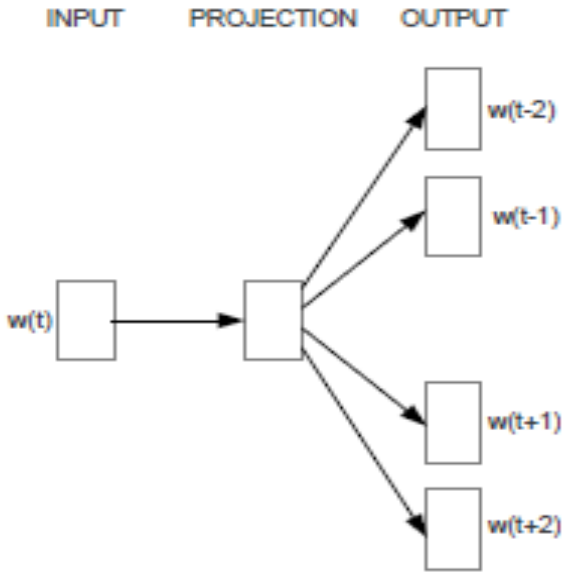
Word2Vec

1) CBOW (context words -> center word 예측)



CBOW

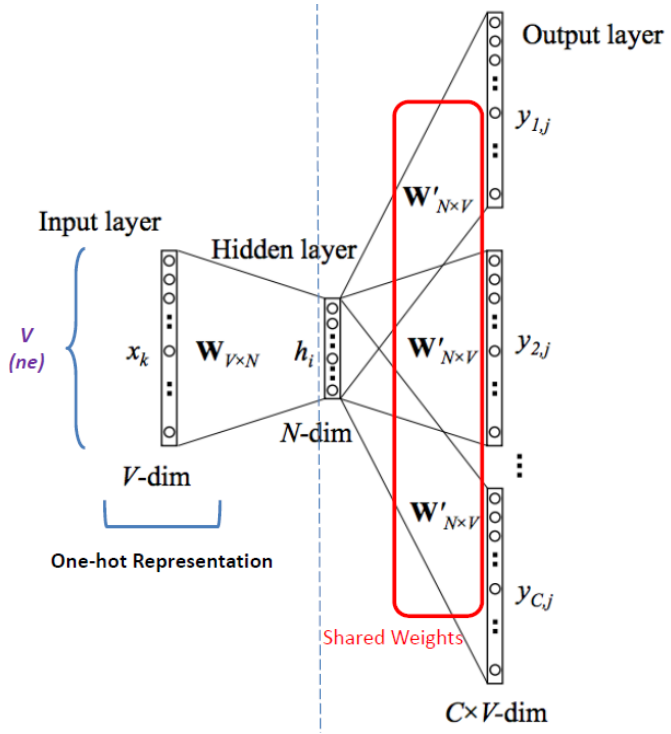
2) Skip-gram (center word -> context words 예측)



Skip-gram

Word2Vec

2) Skip-gram (center word -> context words 예측)



notation) V: 단어사전의 dim, N: 원하는 word vector의 dim, C/2: window-size

a) Input -> Hidden
: $\boldsymbol{h_i}$(Nx1) $= \boldsymbol{W^T}$(NxV) $*$ $\boldsymbol{x_k}$(Vx1)

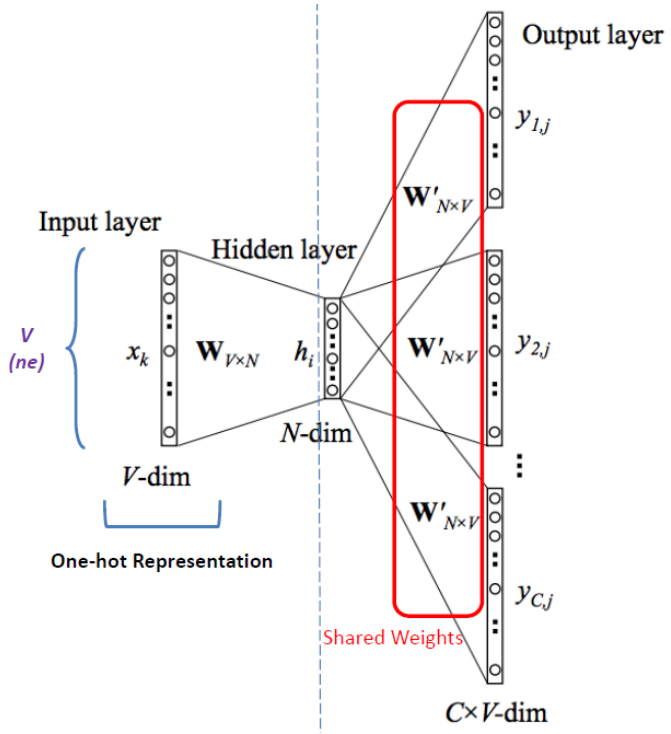
b) Hidden -> Output
: $\boldsymbol{y_j}$(Vx1) $= (\boldsymbol{W`})^T$(VxN) $*$ $\boldsymbol{h_i}$(Nx1)

c) Output -> Softmax
: $\boldsymbol{y_j}$(Vx1)에 softmax를 취해 center word가 등장했을 때, 각 단어가 등장할 확률 계산

d) Softmax -> Loss
: c)의 결과와 true context words의 one-hot vector간 cross-entropy 계산
(∵ context words의 등장 여부만 판단하므로 분류 모델로 볼 수 있음)

Word2Vec

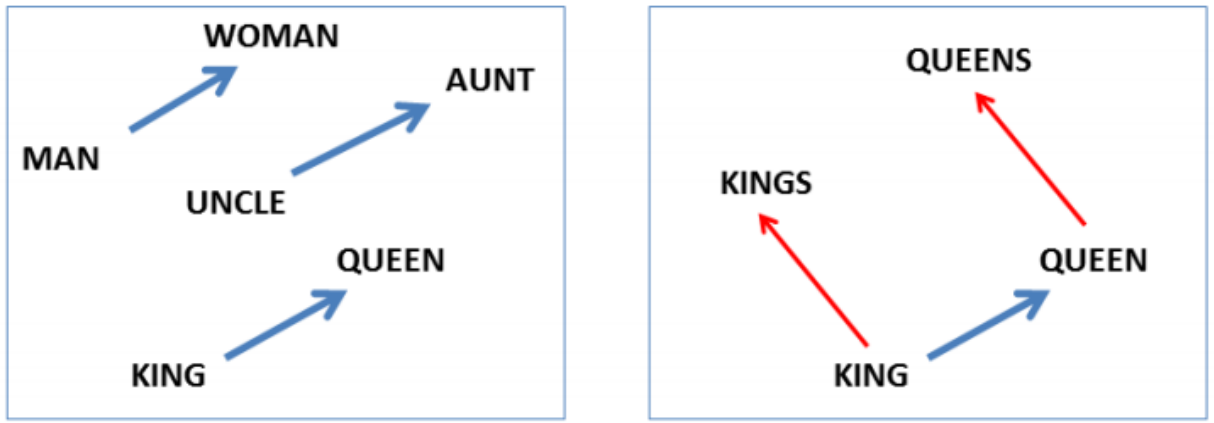
2) Skip-gram (center word -> context words 예측)



- 최종적으로 학습해야 하는 가중치: Input -> Hidden 가중치 W(VxN)
- 학습된 가중치 W(VxN) 의 각 row가 각 단어에 대한 word vector
  : 처음에 설정한 N차원의 word vector
- Output은 word vector간의 내적 값이므로 Skip-gram은 center word & context words 간의 코사인유사도를 학습하는 모델이라고 볼 수 있음
  : one-hot vector와 다르게 단어간 의미상 유사도 측정 가능

Word2Vec

2) Skip-gram (center word -> context words 예측)



(Mikolov et al., NAACL HLT, 2013)

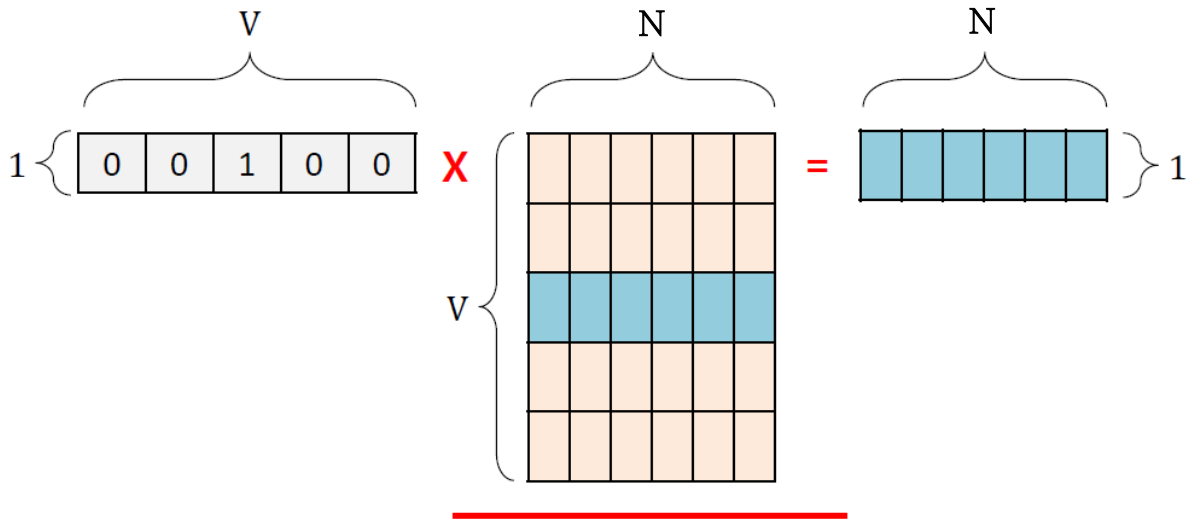
Vector Representation 을 통한 추론의 예시.

- Word analogy task(word vector간 연산)
  : QUEEN = KING – MAN + WOMAN
  (∵ MAN – WOMAN = KING – QUEEN)
  -> 단어의 의미적 학습 가능

Word2Vec

Look-up table

: 학습된 word vector를 table로 정리해 table에서 필요한 word를 찾아서 사용하는 방법



- Word2Vec에서 학습된 word vector의 matrix W(VxN)를 Look-up table로 설정
- word를 embedding할 때마다 word에 해당하는 one-hot vector에 Look-up을 행렬곱을 해 해당 word vector를 참조해 embedding 하는 방식

3. Task-based representation

From random initialization to Word vector

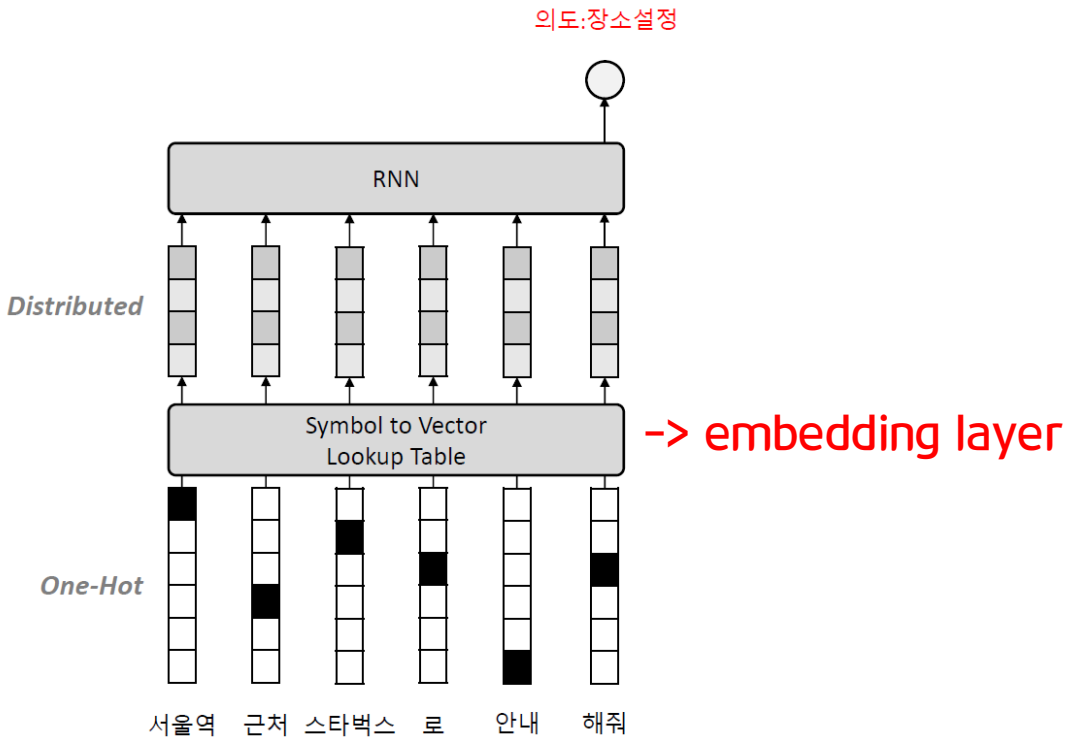
: Task model에 embedding layer를 추가해 Task-Loss에 대한 back-propagation을 embedding layer까지 진행해 word vector를 학습하는 방법

-> task를 기반으로 word vector를 학습하므로 word의 일반적인 의미가 아닌 task에 적합한 word의 의미를 학습함

ex) good: task에 따라 '좋은' 또는 '상품'의 의미를 지닌 word vector로 학습됨

3. Task-based representation

From random initialization to Word vector

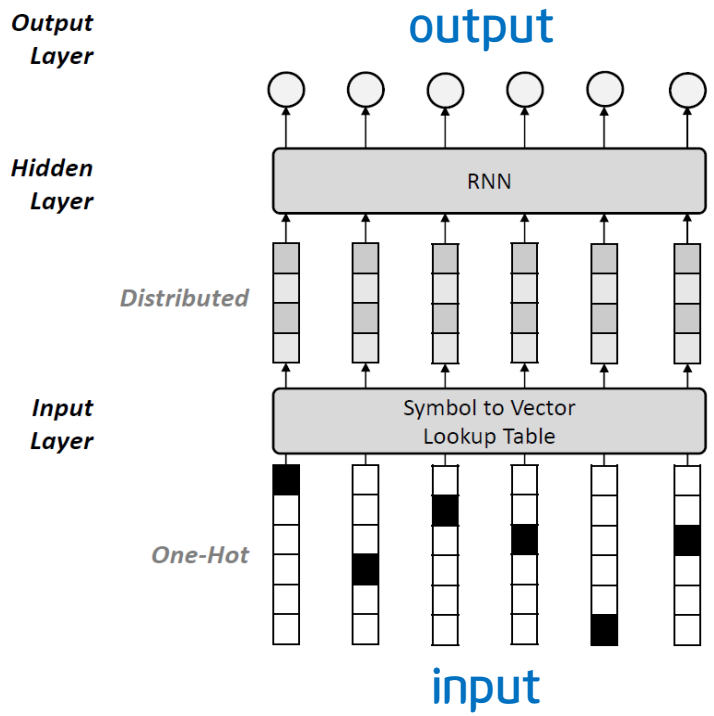


-> embedding layer

- embedding layer의 Look-up table의 초기값을 random으로 설정
- 분류 모델의 Loss로부터의 back-propagation을 통해 Look-up table을 update하는 방식으로 word vector를 학습함

Sequence to Sequence

: Seq2Seq는 RNN의 가장 발전된 형태의 모델로 LSTM, GRU 등 RNN cell을 길고 깊게 쌓아 sequence 데이터를 처리하는 데 특화된 모델



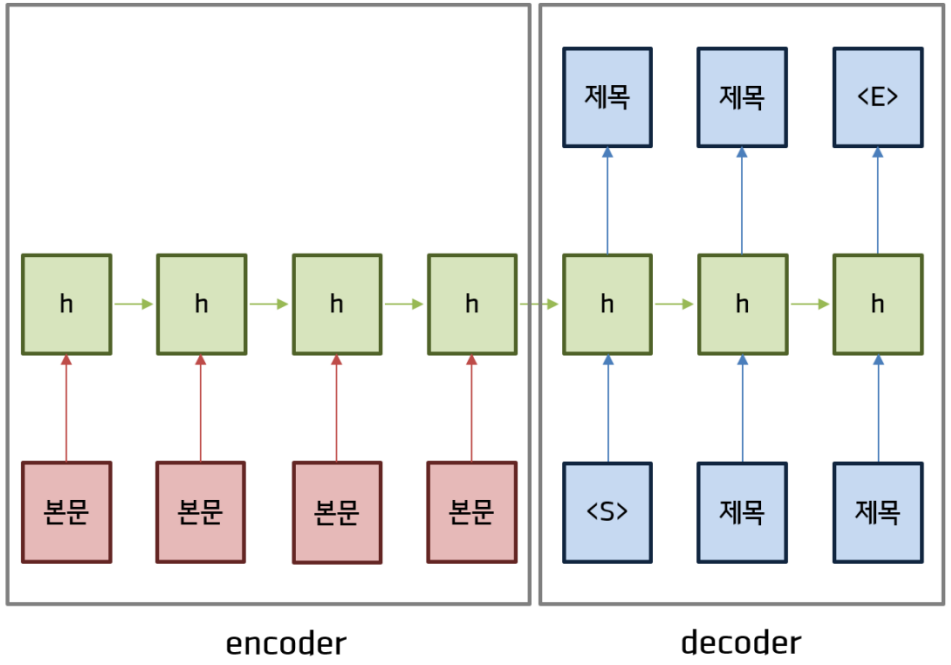
Decoder: encoder의 압축 정보를 받아 output 생성

Encoder: input 정보 압축

Embedding: input의 각 word를 벡터화

Sequence to Sequence

- Encoder & Decoder

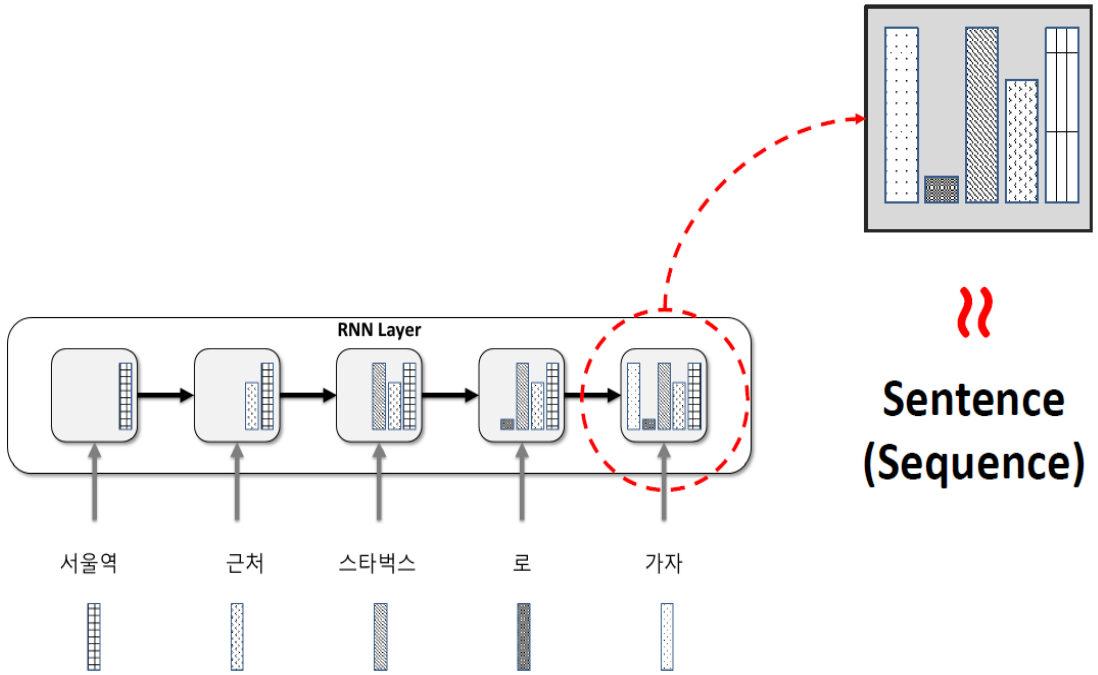


- data: Encoder input, Decoder input, target

token	용도
<UNK>	기준점(threshold)보다 빈도수가 낮은 단어를 대체하는 token
<PAD>	encoding size & decoding size보다 size가 작을 경우 채워주는 token
<S>	decoder input의 처음에 붙이는 token
<E>	target의 마지막에 붙이는 신호

Sequence to Sequence

- Encoder & Decoder

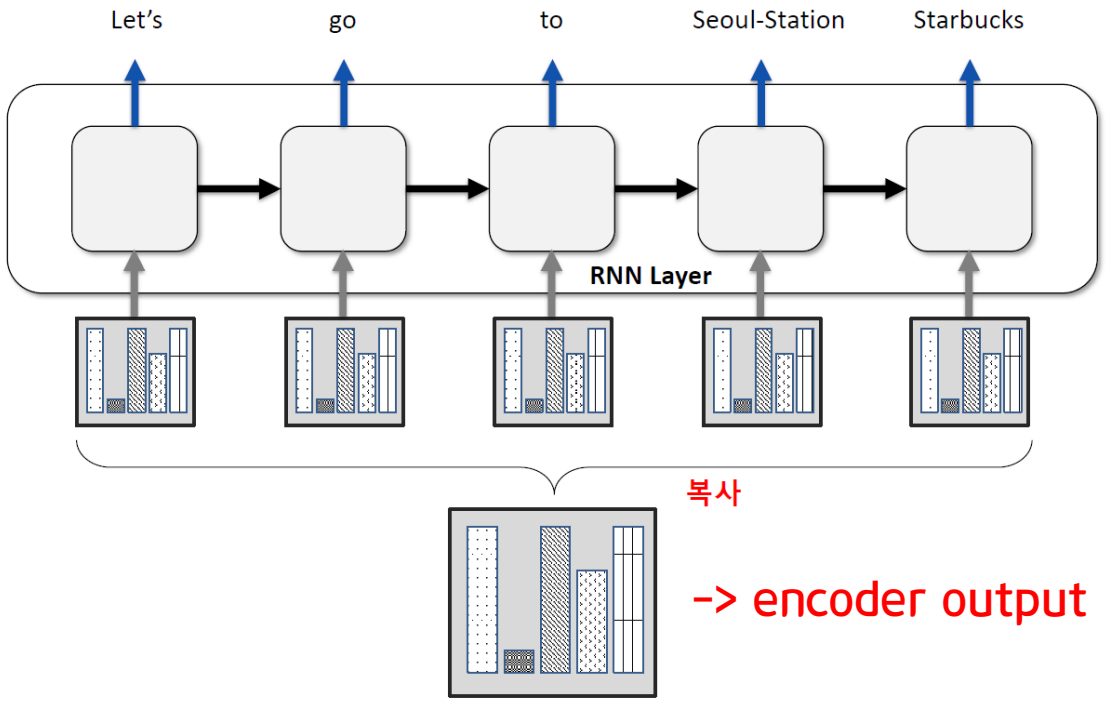


- Encoder
  : RNN layer를 거치면서 input의 정보를 압축한 sequence vector 생성

- Encoder input
  : fixed-size의 sequence(∵행렬 연산)
  -> 길이가 긴 sequence는 특정부분을 잘라내고
  길이가 짧은 sequence는 <PAD>를 붙여줌

Sequence to Sequence

- Encoder & Decoder

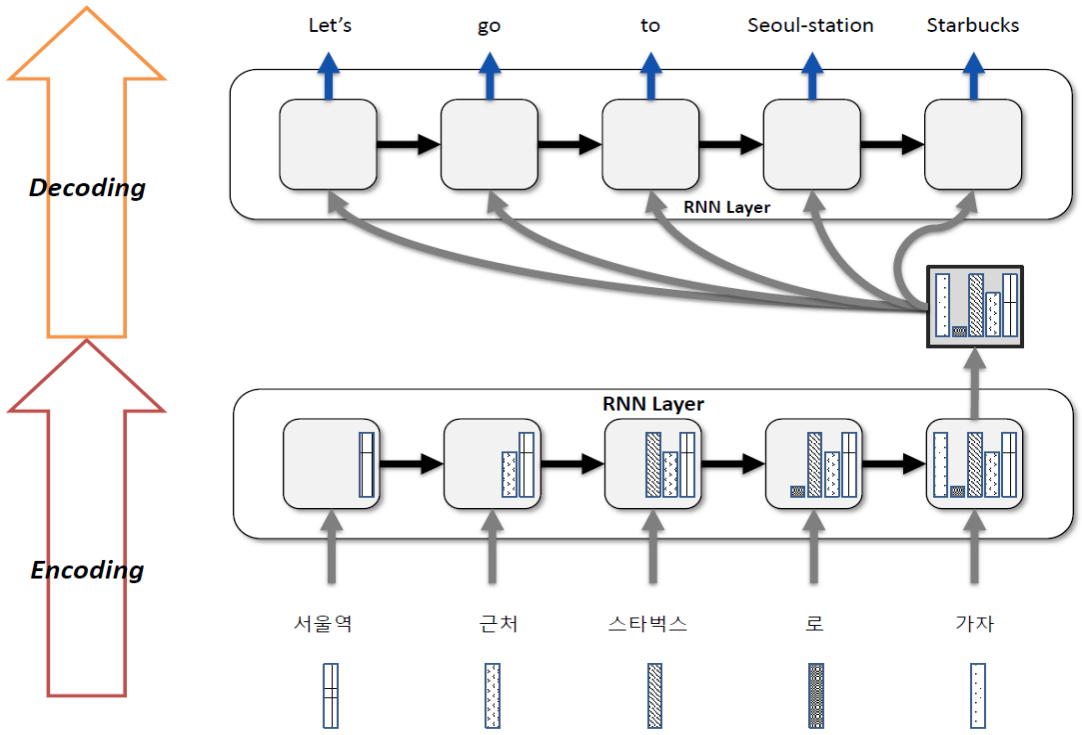


- Decoder
  : Encoder의 압축 정보와 Decoder의 input을 받아 최종 output 생성

- Decoder input:
  train: <S> + 지정된 길이의 제목 sequence
  test: <S> (이후 input은 예측된 단어)
  -> Decoder의 시작을 알리기 위해 <S> 이용

Sequence to Sequence

- Encoder & Decoder



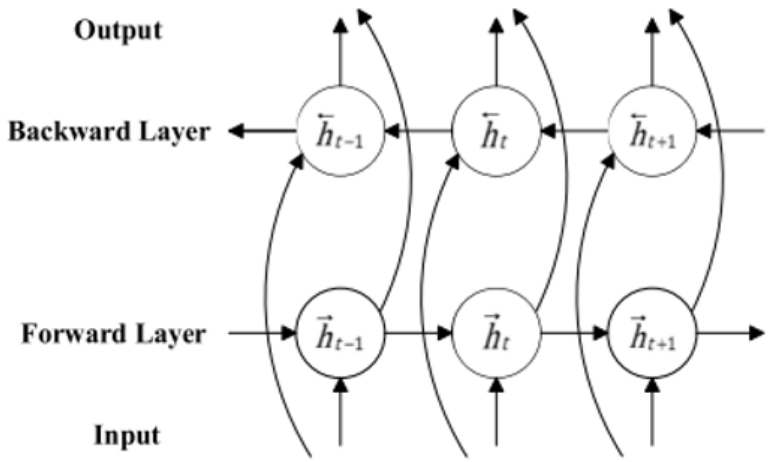
- target
  : fixed-size의 sequence + <E>
  -> Decoder의 끝을 알리기 위해 <E> 이용

- Loss
  : Decoder output과 target간의 cross-entropy

Sequence to Sequence

- **BiDirectional RNN**

  : input sequence의 forward & backward 방향을 모두 고려하는 RNN



bidirectional

- Forward Layer

  : input sequence를 순방향으로 넣는 RNN layer

$$\overrightarrow{h_t} = f(\overrightarrow{W_{xh}} x_t + \overrightarrow{W_{hh}} \overrightarrow{h_{t-1}})$$

- Backward Layer

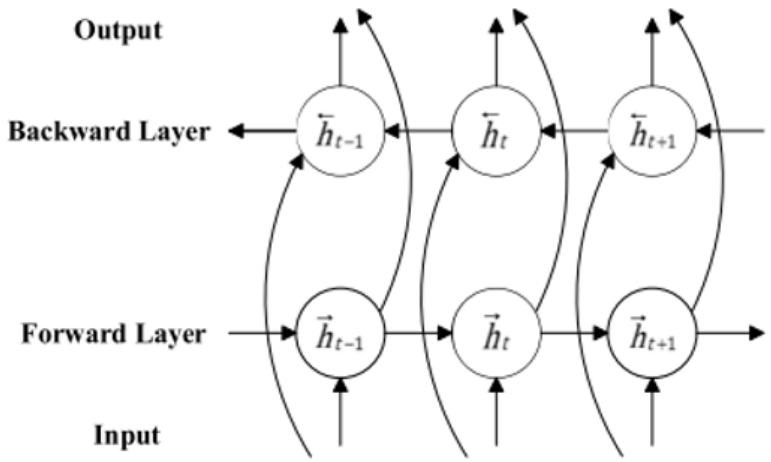
  : input sequence를 역방향으로 넣는 RNN layer

$$\overleftarrow{h_t} = f(\overleftarrow{W_{xh}} x_t + \overleftarrow{W_{hh}} \overleftarrow{h_{t-1}})$$

Sequence to Sequence

- **BiDirectional RNN**

  : input sequence의 forward & backward 방향을 모두 고려하는 RNN



bidirectional

- Output
  : Forward Layer & Backward Layer의 hidden 모두 반영

$$y_t = g(W_{hy}[\overrightarrow{h_t}; \overleftarrow{h_t}])$$

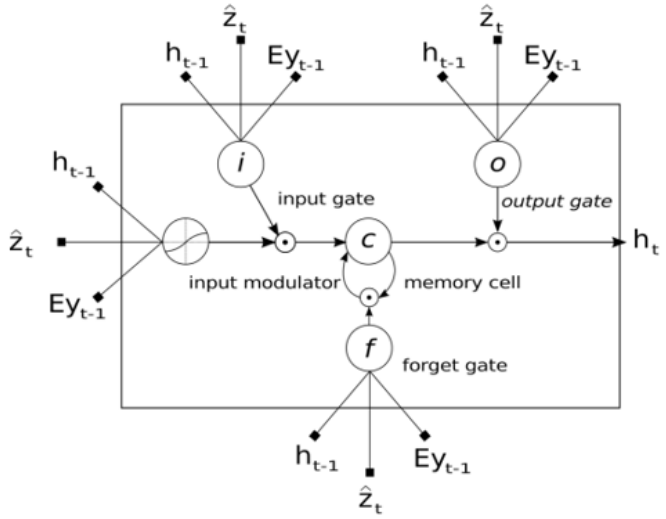
- Bidirectional RNN은 양방향 정보를 모두 반영하므로 기존 RNN보다 성능이 향상됨

- 일반적으로 영어에서는 forward directional RNN보다 backward directional RNN의 성능이 더 좋음

Sequence to Sequence

- **Attention mechanism**

  : Attentional decoder는 decoding시 매 time-step 별로 새로 생성될 토큰을 결정할 때 source sequence 중 가장 관련이 많은 token을 결정한 후 그 정보를 활용하는 구조



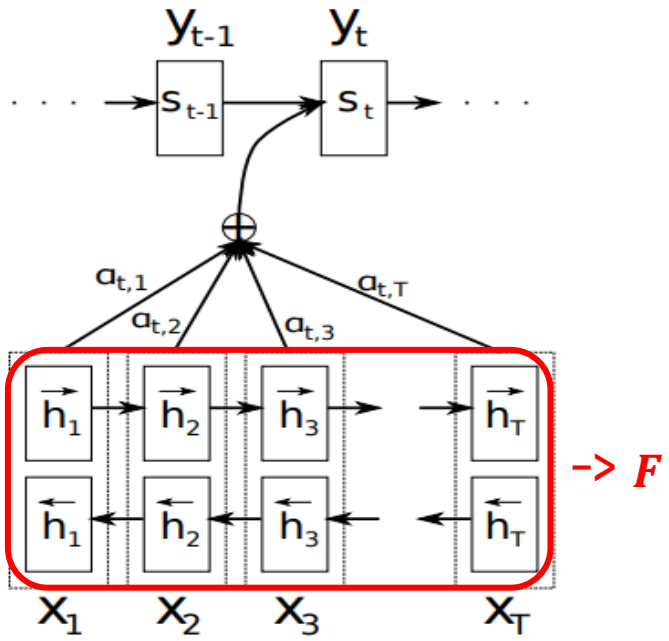
attention

- 문장 길이가 길고 층이 깊으면, encoder가 압축해야 할 정보가 너무 많아서 정보 손실이 일어나고, decoder는 encoder가 압축한 정보를 초반 예측에만 사용하는 경향을 보임
- 따라서 encoder-decoder 사이에 bottle-neck 문제가 발생함
- 이를 보완하기 위해 decoder 예측 시 가장 의미 있는 encoder 입력에 주목하게 만드는 attention mechanism 이용

Sequence to Sequence

- **Attention mechanism**

  : Attentional decoder는 decoding시 매 time-step 별로 새로 생성될 토큰을 결정할 때 source sequence 중 가장 관련이 많은 token을 결정한 후 그 정보를 활용하는 구조



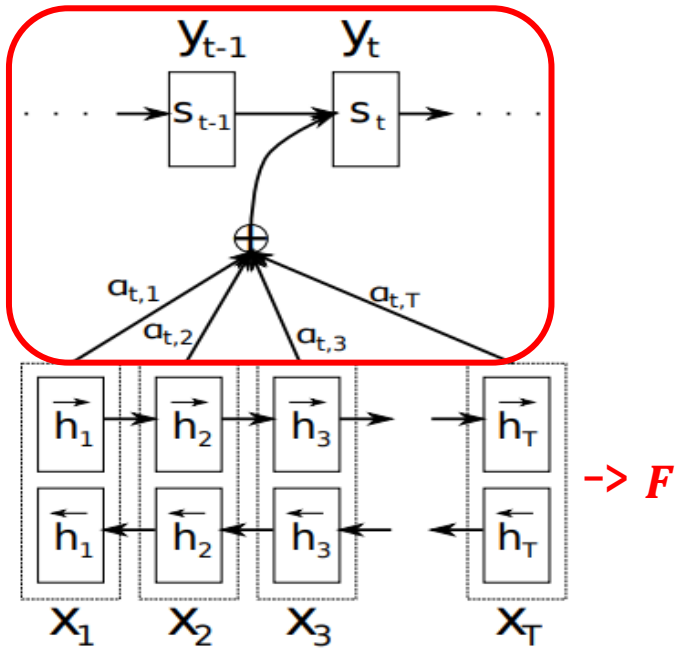
- Encoder와 Decoder에 서로 관련된 단어가 존재한다면 해당 단어들에 대해 encoder의 output과 decoder의 input은 유사할 것이라는 아이디어를 기반으로 함
  (ex. '서울역'의 encoder output ≒ 'seoul-station'의 decoder input)

- Encoder는 기존 Seq2Seq와 동일하며, Encoder의 양방향 hidden state $h_j = \begin{bmatrix} \overrightarrow{h_j} \\ \overleftarrow{h_j} \end{bmatrix}$를 j번째 열벡터로 하는 행렬 F를 생성함

Sequence to Sequence

- **Attention mechanism**

  : Attentional decoder는 decoding시 매 time-step 별로 새로 생성될 토큰을 결정할 때 source sequence 중 가장 관련이 많은 token을 결정한 후 그 정보를 활용하는 구조



-> F

- e_{ij}: decoder의 input s_{i-1}과 encoder의 output h_j의 유사도

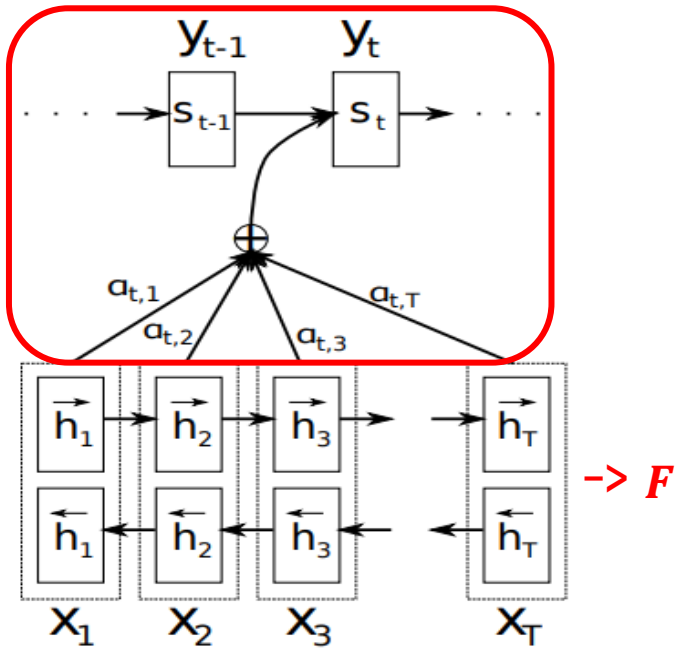
$$e_{ij} = a(s_{i-1}, h_j)$$

이 때, a는 유사도를 측정하는 alignment model로 주로
1)$F^T V s_{i-1}$ 2)$v^T tanh(WF + V s_{i-1})$와 같은 형태를 이용함

Sequence to Sequence

- **Attention mechanism**

  : Attentional decoder는 decoding시 매 time-step 별로 새로 생성될 토큰을 결정할 때 source sequence 중 가장 관련이 많은 token을 결정한 후 그 정보를 활용하는 구조



-> F

- α_{ij}: e_{ij}에 softmax를 취한 유사도의 확률 값

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

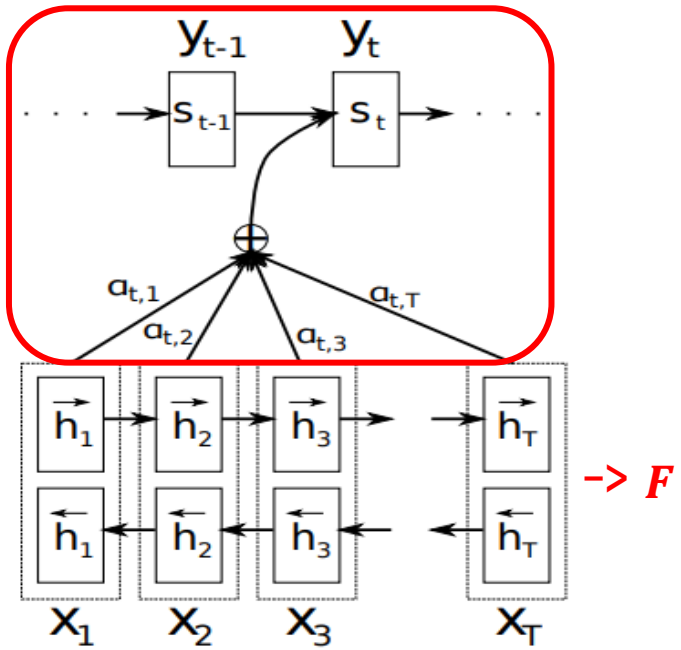
- α_i: i번째 단어를 예측할 때 쓰이는 attention vector

$$\alpha_i = [\alpha_{i1}, \alpha_{i2}, \dots, \alpha_{iT_x}]$$

Sequence to Sequence

- **Attention mechanism**

  : Attentional decoder는 decoding시 매 time-step 별로 새로 생성될 토큰을 결정할 때 source sequence 중 가장 관련이 많은 token을 결정한 후 그 정보를 활용하는 구조



-> F

- c_i: i번째 단어를 예측할 때 쓰이는context vector

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j = F\alpha_i$$

- decoder의 input s_{i-1}와 context vector c_i의 concatenation으로 s_i 생성

Evaluation

BLEU (Bilingual Evaluation Understudy)

: 사람이 번역한 reference와 machine translation과의 n-gram 매칭을 통해 측정하는 평가지표로 machine translation에서 등장한 n-gram이 사람이 번역한 reference에도 똑같이 등장한 횟수가 많을수록 높은 평가지표이다.

제약조건

: 1) 각 n-gram에 대해서 reference에 등장한 n-gram의 매칭은 1회만 유효함
2) 짧은 문장일수록 매칭이 쉬우므로 문장길이에 대한 패널티를 부여함

Unit 04 | Evaluation

BLEU

Reference (human) translation:
The U.S. island of Guam is maintaining a high state of alert after the Guam airport and its offices both received an e-mail from someone calling himself the Saudi Arabian Osama bin Laden and threatening a biological/chemical attack against public places such as the airport .

Machine translation:
The American [?] international airport and its the office all receives one calls self the sand Arab rich business [?] and so on electronic mail , which sends out ; The threat will be able after public place and so on the airport to start the biochemistry attack , [?] highly alerts after the maintenance.

step1) n의 최댓값 k을 설정

step2) k 이하의 n-gram에 대해 p_n을 계산

$$p_n = \frac{\#\ matched\ n-grams}{\#\ n-grams\ in\ candidate\ translation}$$

step3) 간결성 패널티 β 계산

$$\beta = e^{\min(0,1-\frac{len(ref)}{len(MT)})}$$

step4) BLEU 계산(n이 클수록 맞추기 어려우므로 확률 값을 더 크게 반영)

$$BLEU = \beta \prod_{n=1}^{k} p_n^{1/2^n}$$

BLEU

한계점

: 1) BLEU는 여러가지 스코어의 곱으로 이루어져 있기 때문에 문장단위에서는 0의 점수를 가질 확률이 높으므로 corpus 단위로만 평가 가능하다.

2) 실제로는 사람마다 번역한 결과가 다르기 때문에 번역에는 여러가지 reference가 존재하는데, BLEU는 하나의 reference만을 기준으로 score를 측정한다.

3) BLEU는 문법구조, 유의어 등을 반영하지 못하므로 BLEU가 낮다고 나쁜 번역이라고 보기 어렵다.

들어주셔서 감사합니다.