9월 16일 세미나

목차

발표할 내용 대표적인 자연어 처리 딥러닝 모델

이유 : KDD의 논문 내용을 이해하기 위해서는 Transformer의
개념을 이해해야 하기 때문에..

2014 : Seq2Seq

2015 : Seq2Seq + Attention

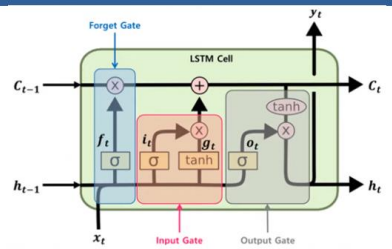
2017 : Transformer

2017 ~ : GPT1, 2, 3, Bert

Review

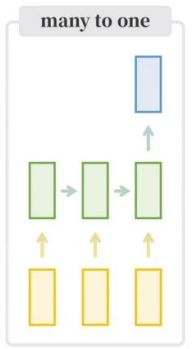
LSTM layer에 대한 이해가 필수.

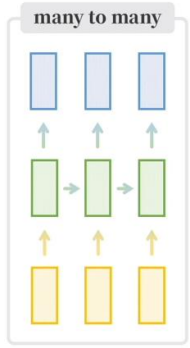
LSTM



1. **Forget Gate :** 과거 정보를 잊기 위한 게이트
2. **Input Gate :** 현재 정보를 추가하기 위한 게이트
3. **Output Gate :** 최종 결과를 위한 게이트

LSTM 사용모델 (대표 2가지)



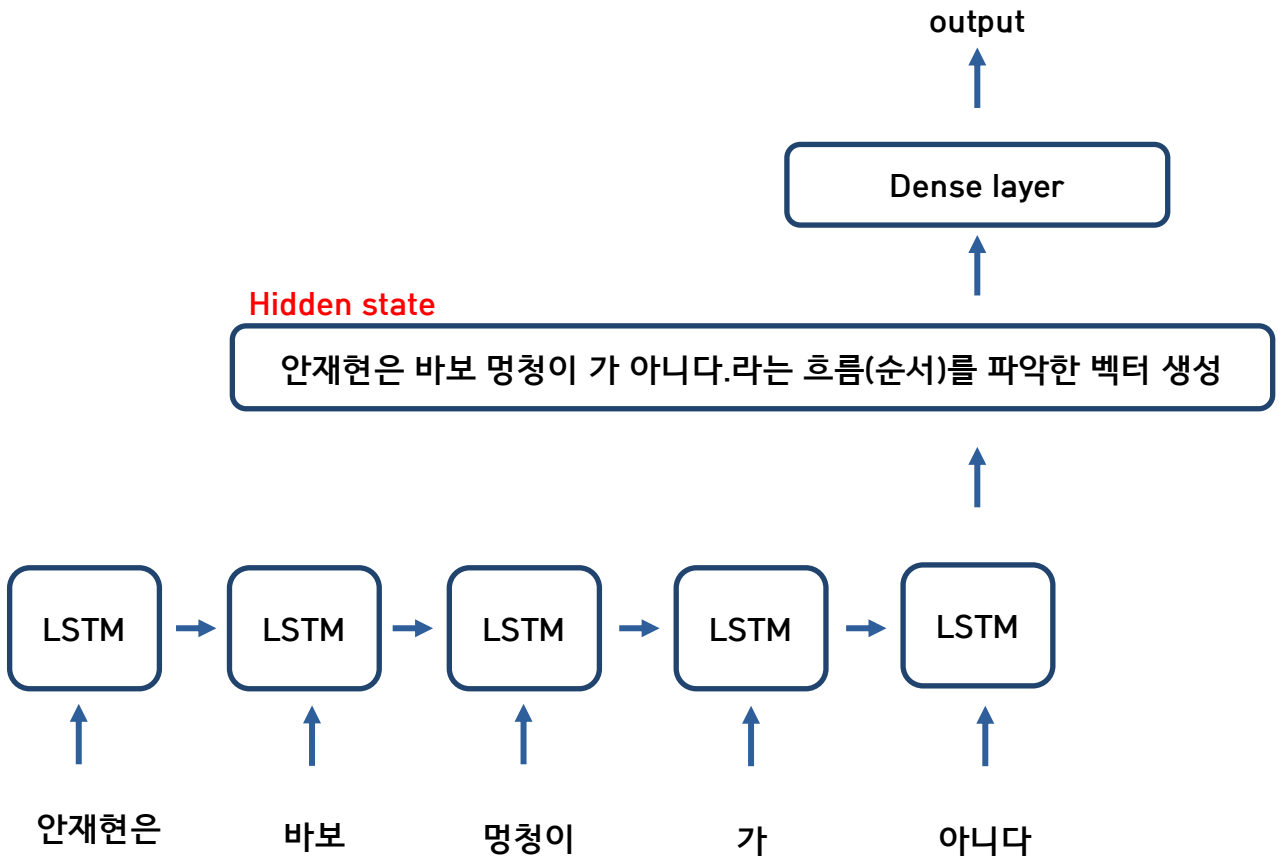


Many to One : 시계열 데이터 분석 및 문장의 감정 분석 등

Many to Many : 언어번역 모델 및 챗 봇 등

Many to One

LSTM layer의 결과 값으로는 Hidden state가 출력됩니다.




```
x = tf.random.uniform(shape=(1, 5, 1))
```

```
lstm = LSTM(32, return_sequences = False)(x)
```

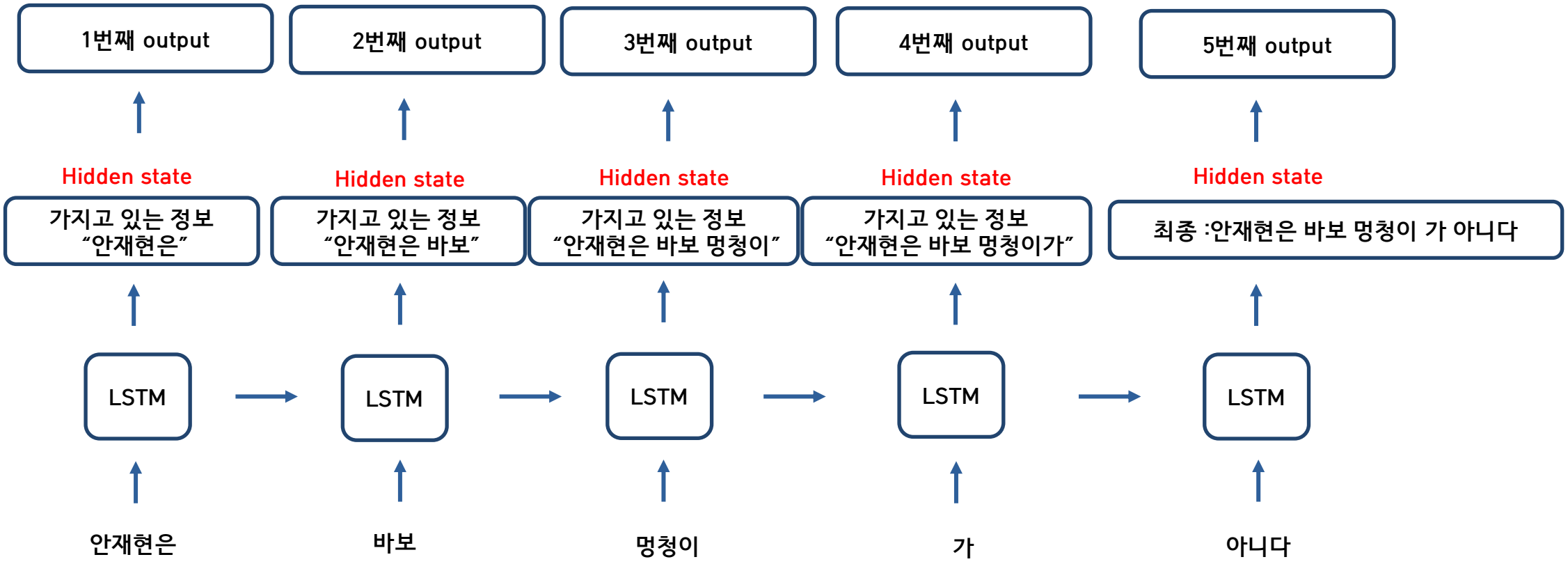
```
lstm.shape
```

```
TensorShape([1, 32])
```

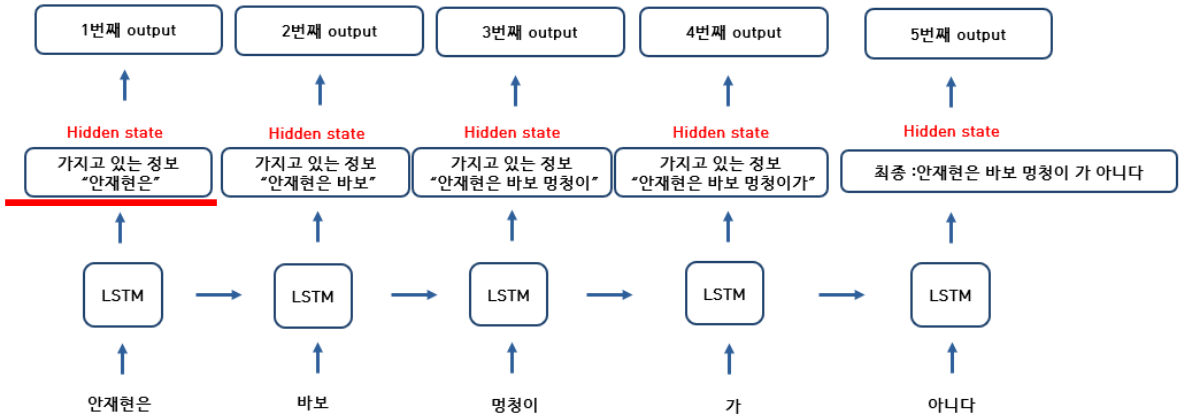
```
lstm[0] # hidden state 정보
```

```
<tf.Tensor: shape=(32,), dtype=float32, numpy=
array([-0.05833721, -0.05061596,  0.03891953, -0.01145083,  0.06062743,
       -0.02227054, -0.06765839,  0.0310909 , -0.04489772,  0.08720598,
        0.02180265,  0.04883013, -0.06752123, -0.02138126,  0.04067618,
       -0.07620423, -0.07380977, -0.04170011,  0.05346873, -0.03556101,
        0.03211616,  0.05330011, -0.04550693,  0.08088022, -0.06204802,
        0.01169802, -0.03131113, -0.03850553,  0.01424779,  0.0723258 ,
        0.06402884,  0.04098319], dtype=float32)>
```

Many to Many


1번째 output
2번째 output
3번째 output
4번째 output
5번째 output
Hidden state
Hidden state
Hidden state
Hidden state
Hidden state
가지고 있는 정보 "안재현은"
가지고 있는 정보 "안재현은 바보"
가지고 있는 정보 "안재현은 바보 멍청이"
가지고 있는 정보 "안재현은 바보 멍청이가"
최종 :안재현은 바보 멍청이 가 아니다
LSTM
LSTM
LSTM
LSTM
LSTM
안재현은
바보
멍청이
가
아니다

Many to Many




```
x = tf.random.uniform(shape=(1, 5, 1))
```

```
lstm = LSTM(32, return_sequences = True)(x)
```

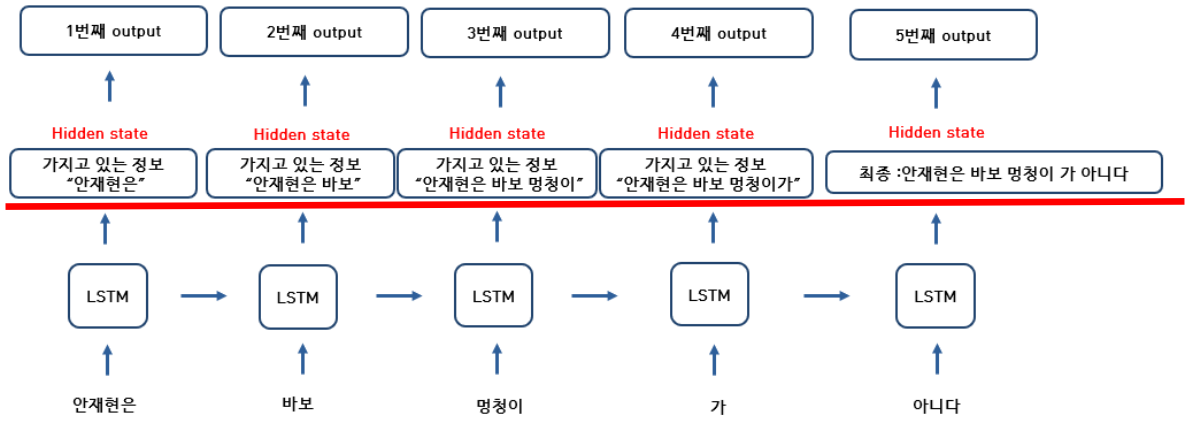
```
lstm.shape
```

```
TensorShape([1, 5, 32])
```

```
lstm[0][0] # hidden state 정보
```

```
<tf.Tensor: shape=(32,), dtype=float32, numpy=
array( 0.00195432,  0.01709786, -0.01057209,  0.01041461,  0.01410812,
      -0.00611006, -0.00461479, -0.01641238, -0.00324372,  0.0112311 ,
      -0.00430766, -0.01171554,  0.00314369,  0.01189991,  0.01276057,
      -0.00359455, -0.01821861,  0.00569498, -0.0124941 ,  0.01729859,
      -0.01596324,  0.00788764, -0.01137581,  0.02060312, -0.00429557,
      -0.01391818, -0.00782552,  0.00644801, -0.00865948,  0.00149803,
      -0.00289152, -0.01506324], dtype=float32)>
```

Many to Many




```
lstm[0] # hidden state 정보
```

```
<tf.Tensor: shape=(5, 32), dtype=float32, numpy=
array([[ 0.00195432,  0.01709786, -0.01057209,  0.01041461,  0.01410812,
        -0.00611006, -0.00461479, -0.01641238, -0.00324372,  0.0112311 ,
        -0.00430766, -0.01171554,  0.00314369,  0.01189991,  0.01276057,
        -0.00359455, -0.01821861,  0.00569498, -0.0124941 ,  0.01729859,
        -0.01596324,  0.00788764, -0.01137581,  0.02060312, -0.00429557,
        -0.01391818, -0.00782552,  0.00644801, -0.00865948,  0.00149803,
        -0.00289152, -0.01506324],
       [ 0.00643059,  0.05230677, -0.02970994,  0.03286283,  0.04048819,
        -0.01831999, -0.01640013, -0.04799915, -0.01065593,  0.03885581,
        -0.01276772, -0.0348695 ,  0.01045726,  0.03730418,  0.03991703,
        -0.01192827, -0.05444446,  0.01833103, -0.03784417,  0.0544773 ,
        -0.04836419,  0.02433733, -0.03173381,  0.06628548, -0.01267091,
        -0.04465014, -0.02596248,  0.01796034, -0.02723905,  0.00365761,
        -0.01155271, -0.04521547],
       [ 0.00847752,  0.04996704, -0.0279263 ,  0.03171166,  0.03303511,
        -0.02102872, -0.01886425, -0.05000684, -0.01264043,  0.03869578,
        -0.00943664, -0.04007478,  0.0119991 ,  0.04376892,  0.03450331,
        -0.01667585, -0.06458636,  0.01905481, -0.04158842,  0.0548035 ,
        -0.05740071,  0.02654798, -0.02561309,  0.07403766, -0.00984163,
        -0.05405014, -0.03051407,  0.014162  , -0.02261876,  0.00123873,
        -0.01653843, -0.04107218],
       [ 0.01095233,  0.06691842, -0.03568148,  0.04397456,  0.04104413,
        -0.02839891, -0.02732792, -0.06288272, -0.01835349,  0.05464178,
        -0.01085064, -0.0522504 ,  0.01625937,  0.05808708,  0.04695264,
        -0.02227727, -0.08543221,  0.02793356, -0.05405536,  0.07458265,
        -0.07552743,  0.03540804, -0.03024762,  0.09892067, -0.01181274,
        -0.07504166, -0.04207195,  0.01735519, -0.0286809 ,  0.0016781 ,
        -0.02433511, -0.05610375],
       [ 0.01342819,  0.08020437, -0.04100993,  0.05322834,  0.04461775,
        -0.03555999, -0.03519996, -0.07385727, -0.02377721,  0.06713522,
        -0.01064887, -0.06365588,  0.02008392,  0.07182936,  0.05499301,
        -0.02812848, -0.10490367,  0.03533391, -0.06506652,  0.08852547,
        -0.09206016,  0.04323209, -0.03152005,  0.12075984, -0.01249185,
        -0.09407034, -0.05361676,  0.01870184, -0.03146604,  0.00150971,
```

Review

MANY TO MANY 모델을 언어 번역 처리에 적용하려고 보니?..

다음과 같은 문제점이 발생 ..

1. I am Jaehyun -> 나는 재현이야

-> 세 가지의 입력이 두가지의 출력으로 내보내는 것이 힘듬

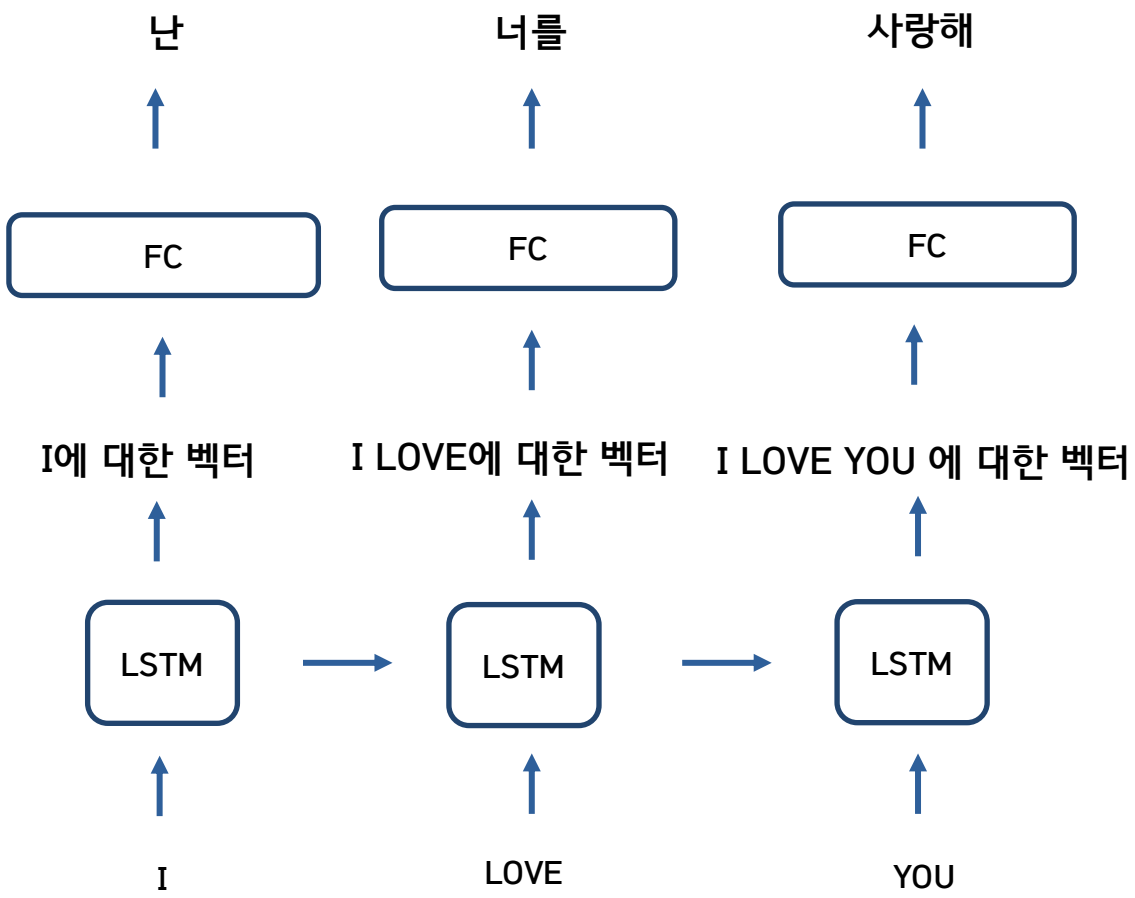
2. S V O(미국) -> S O V(한국)

Ex) I LOVE YOU -> 난 널 사랑해

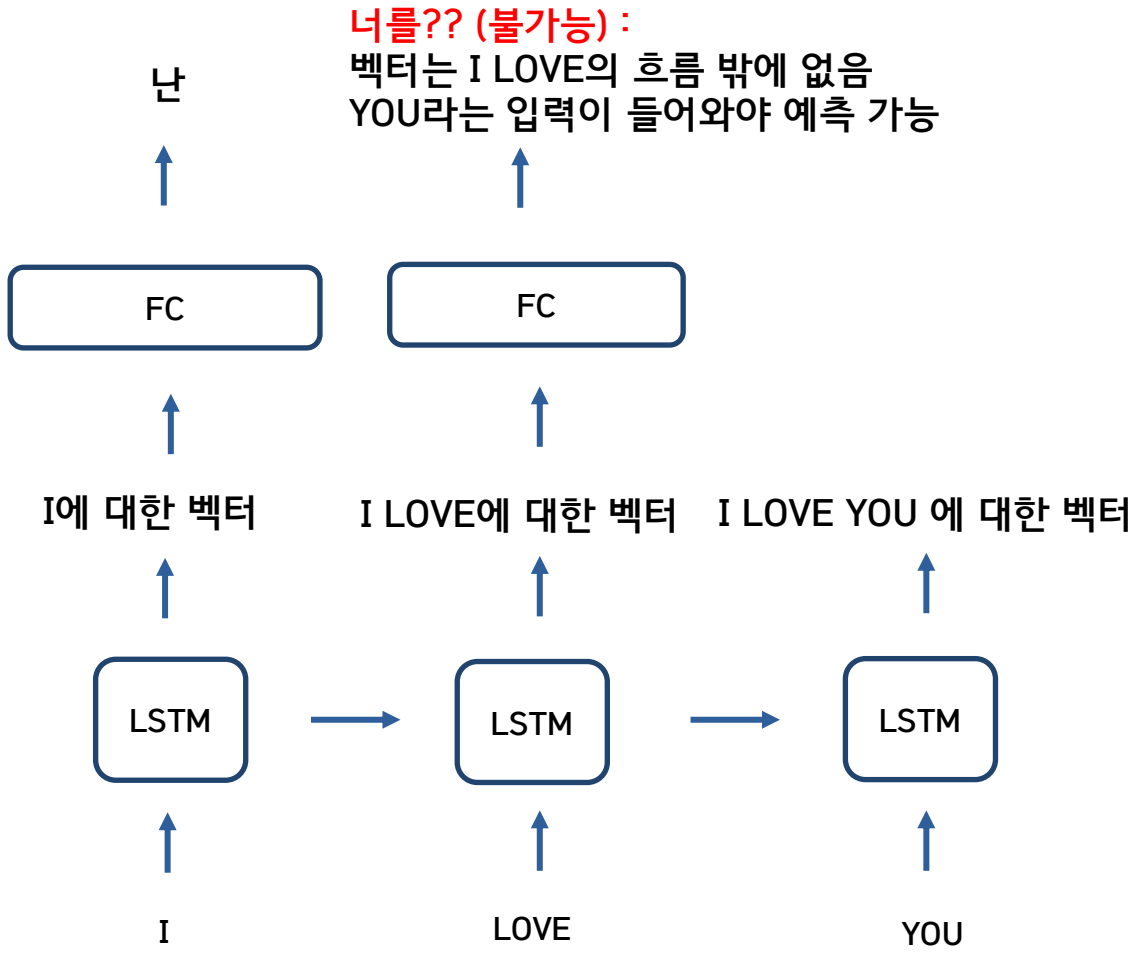
-> 어순이 맞지 않아서, 이전에 들어온 입력 데이터가 다음 데이터를 예측할 수 없음.

Many to many를 번역 모델로 사용

원하는 상황



실제 상황



논문

위에서 언급한 many to many 모델의 약점들을 보완해서 나온 것 -> seq2seq

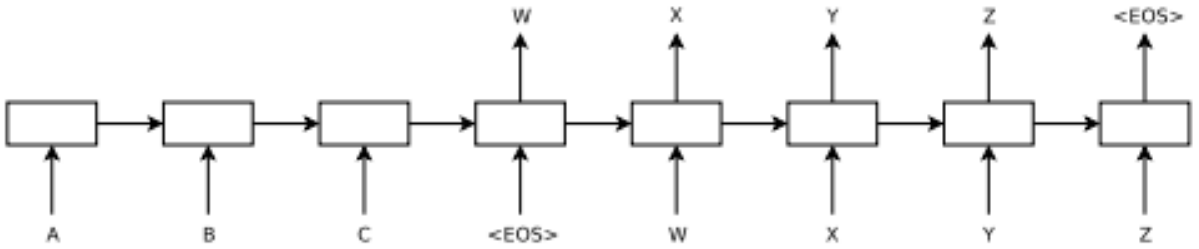
논문에서는 번역모델로 테스트를 진행합니다.

Sequence to Sequence Learning with Neural Networks

Ilya Sutskever
Google
ilyasu@google.com

Oriol Vinyals
Google
vinyals@google.com

Quoc V. Le
Google
qvl@google.com



Seq2Seq

Seq2Seq?
새로운 개념의 도입? (X)
기존 개념의 활용(O)

1. Encoder : RNN(LSTM, GRU)의 순환 신경망을 사용하여 Context vector를 얻는 곳

2. Context vector : Encoder(LSTM)에 나온 최종 hidden state 값
(문맥 벡터)

3. Decoder : Context vector와 RNN(LSTM, GRU)의 순환 신경망을 사용하여 최종 Output 출력

Seq2Seq

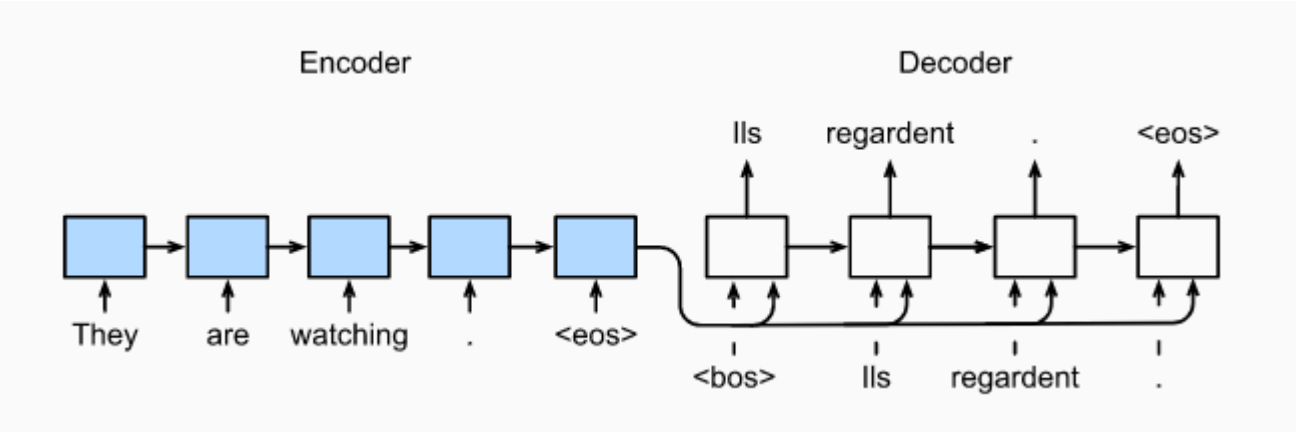
Seq2Seq는 자연어 처리 모델

문제점을 해결할 수 있는 방안(간단)

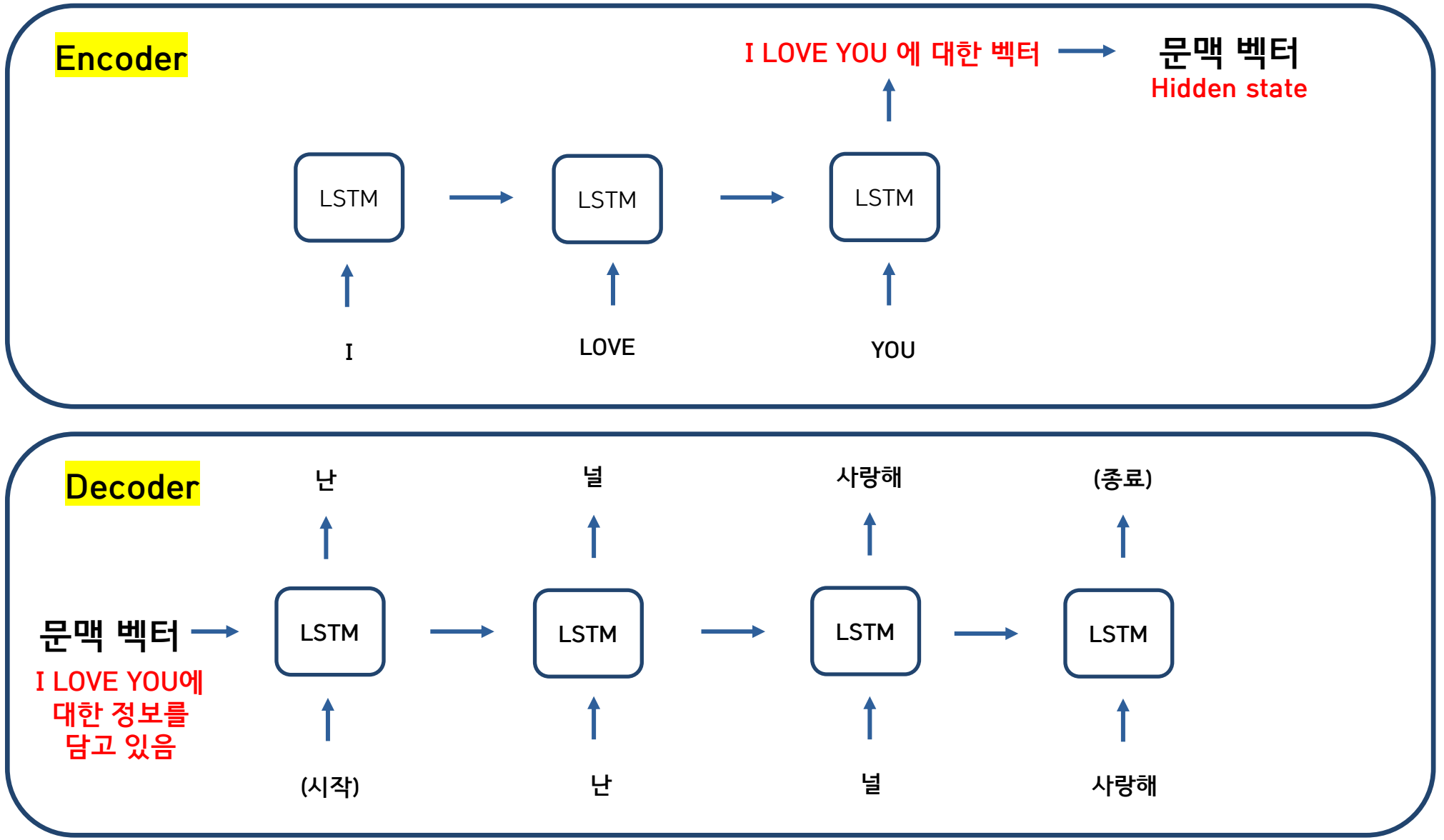
Many to One(Encoder)과 Many to Many (Decoder)를 연결해서 모델을 만들어보자

Many to One은 입력 데이터의 흐름을 가지고 있는 최종 벡터 값을 출력하는 모델

Many to Many는 각 입력에 따른 hidden state를 출력 해주는 모델



구조


Encoder
I LOVE YOU 에 대한 벡터
문맥 벡터
Hidden state
LSTM
LSTM
LSTM
I
LOVE
YOU
Decoder
난
널
사랑해
(종료)
문맥 벡터
I LOVE YOU에
대한 정보를
담고 있음
LSTM
LSTM
LSTM
LSTM
(시작)
난
널
사랑해

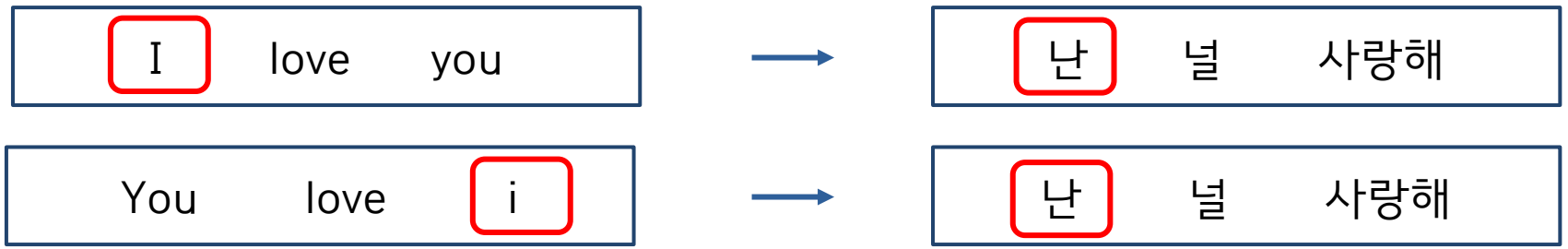
논문

논문에서의 추가적인 내용

1. RNN을 기준으로 설명하였으며, 이는 LSTM으로 변경하여
긴 시퀀스의 문장들도 해석이 가능해진다.

2. 문장의 입력을 반대로 넣을 경우 더 성능이 좋은 것을 확인할 수 있었다.

그 이유로서 보통 학습을 진행할 때를 생각해보면 됨



(RNN 계열의 특성)

컨텍스트 벡터를 생각해볼 때 가장 최근에 입력된 정보가 비교적 기억이 더 잘 남아 있으므로

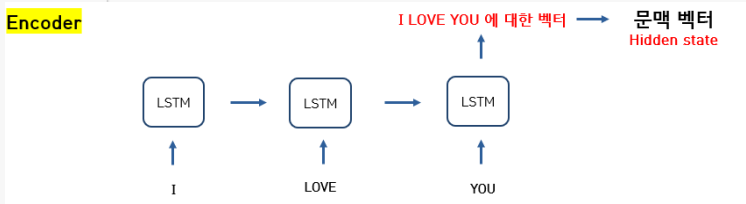
출력의 첫 부분이 컨텍스트 벡터와 연관성이 더 존재하기 때문..

논문

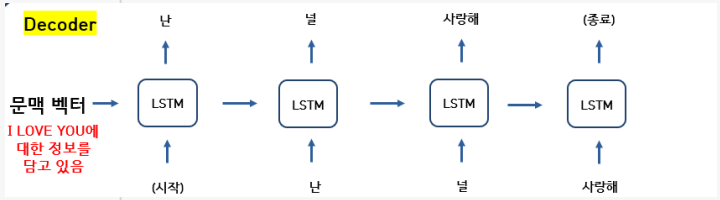
논문에서의 군집화를 통한 결과물

코드 구현

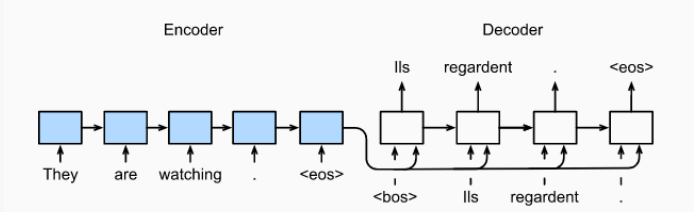
```python
class Encoder(tf.keras.Model):
  def __init__(self, units):
    super(Encoder, self).__init__()
    self.lstm = LSTM(units, return_state = True, return_sequences = True)

  def call(self, inputs):
    x, hidden_state, cell_state = self.lstm(inputs)
    return [hidden_state, cell_state] # LSTM은 RNN과 다르게 hidden_state, cell_state를 인풋으로 내놓음
```




```python
class Decoder(tf.keras.Model):
  def __init__(self, units):
    super(Decoder, self).__init__()
    self.lstm = LSTM(units, return_state = True, return_sequences = True)
    self.attention = Attention()
    self.dense = Dense(1, activation = 'sigmoid')

  def call(self, inputs, initial_state): # initial_state = 문맥 벡터
    x, hidden, cell = self.lstm(inputs, initial_state = initial_state) # 헷갈리면 안되는 것. x와 hidden_state는 동일한 값임
    key_value = tf.
    x = self.dense(x)
    return x
```




```python
class Seq2Seq(tf.keras.Model):
  def __init__(self, units):
    super(Seq2Seq, self).__init__()
    self.encoder = Encoder(units)
    self.decoder = Decoder(units)

  def call(self, inputs):
      encoder_inputs, decoder_inputs = inputs # inputs[0] -> encoder_inputs(입력) inputs[1] -> decoder_inputs(출력)
      context_vector = self.encoder(encoder_inputs) # 2가지가 들어옴 (hidden, cell)
      decoder_outputs,decoder_hidden,decoder_cell = self.decoder(inputs = decoder_inputs, initial_state = context_vector) # cell state와 hidden state가 들어감
      return decoder_outputs # shape 맞추는 것만 맞추면 돌아감
```



Attention

이러한 seq2seq 모델 또한 문제점이 존재 합니다.
바로 고정적인 크기를 가진 문맥 벡터 입니다.

따라서 그 2015년에 나온 새로운 개념 Attention이 추가 됩니다.
Seq2seq에 Attention의 적용은 큰 변화를 가지고 오게 됩니다.

Published as a conference paper at ICLR 2015

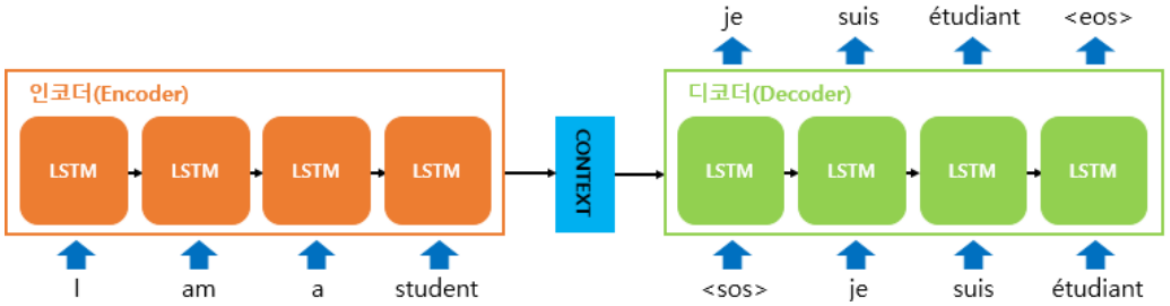
Attention개념을 최초로 소개한 논문

NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

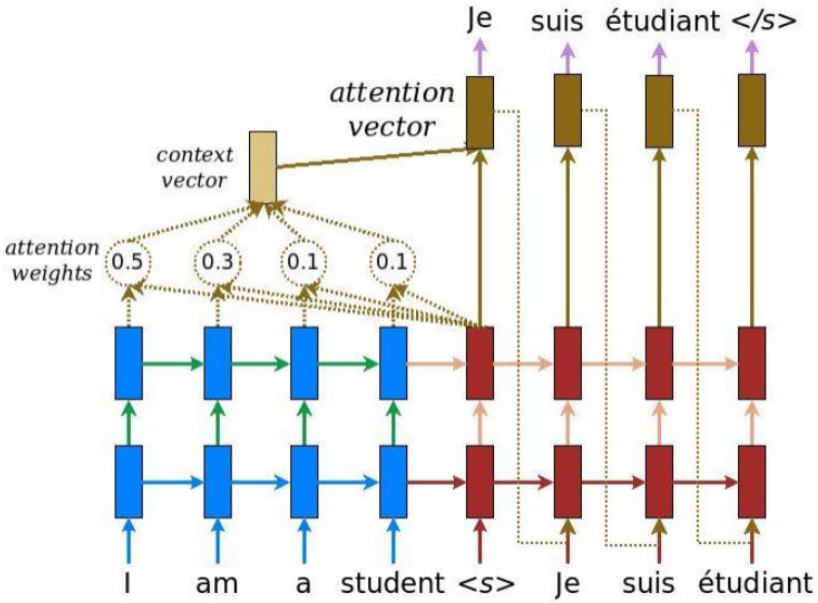
KyungHyun Cho **Yoshua Bengio***
Université de Montréal

모델 비교



Seq2seq

고정된 컨텍스트 벡터를 가지고 있음



Seq2seq + Attention 모델

출력에 따른 동적인 컨텍스트 벡터 선정

Attention

어텐션 자체에 대한 개념을 한번 잡고 갑시다.

어텐션 매커니즘은 제가 집중하고 싶은 부분을 선정한다면, 해당 부분을 위주로 재구성 하는 것 입니다.

집중하고 싶은 부분 : 커피

집중하고 싶은 부분을 위주로 재구성한 벡터를 만들게 됨
-> 자연어 처리에서의 적용은 Seq2seq + Attention에서 볼 수 있습니다.

Attention

구현해야할 부분

1. 어떤 방식으로 현재 출력해야 할 부분과 입력에 가장 관련된 부분를 찾을 수 있을까

2. 가장 관련된 부분을 찾더라도 이를 어떤 식으로 사용해야 할까

이를 해결하기 위한 내용들이 Attention 논문에 나오게 됩니다.

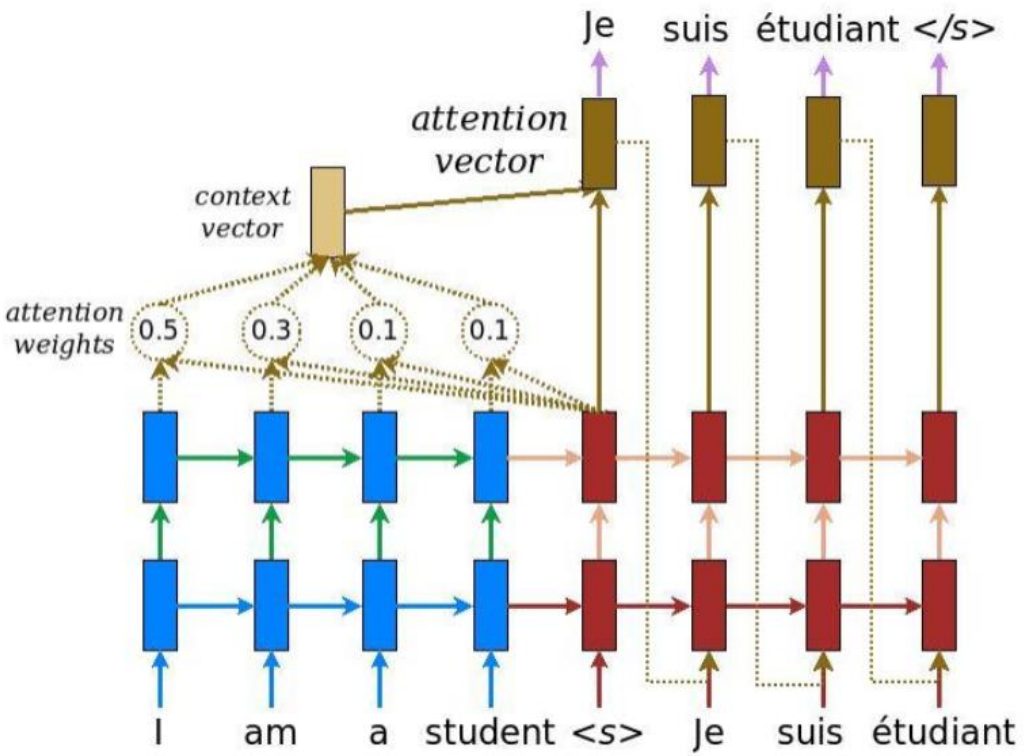
또한 다양하게 발전된 Attention은 여러가지 존재합니다.
성능 차이는 거의 없으나, 계산 방식에 다른 차이입니다. (교육 속도는 차이 남)

대표적으로

1. 루옹 어텐션과 2. 바다나우 어텐션이 존재합니다.
(종류가 많습니다.)

Attention

1. 루옹 어텐션



attention weights : 출력하고 하는 단어와 입력된 단어가 얼마나 연관(상관관계)가 존재하는가를 확률 값으로 나타낸 것

context vector : 출력과 가장 연관성이 존재하는 것을 attention weights로 알게 되었고, 각각의 attention weights와 입력의 hidden state들의 곱의 가중합 입니다.

attention vector : 위에서 구한 context vector와 출력하고자 하는 단어의 hidden state의 concatenate 연산을 통해 나온 벡터입니다.

$$\alpha_{ts} = \frac{\exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'=1}^{S} \exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)} \quad \text{[Attention weights]} \quad (1)$$

$$\boldsymbol{c}_t = \sum_s \alpha_{ts} \bar{\boldsymbol{h}}_s \quad \text{[Context vector]} \quad (2)$$

$$\boldsymbol{a}_t = f(\boldsymbol{c}_t, \boldsymbol{h}_t) = \tanh(\boldsymbol{W}_c[\boldsymbol{c}_t; \boldsymbol{h}_t]) \quad \text{[Attention vector]} \quad (3)$$

Attention

가정 : I am Jaehyun -> 나는 재현입니다.

1. 어떤 방식으로 현재 출력해야 할 부분과 입력에 가장 관련된 부분를 찾을 수 있을까

의문점에 대한 해결

1. 실제로 서로 연관성이 있는 벡터임을 확인하는 과정은 각각의 hidden state를 비교하며 알게 됩니다.

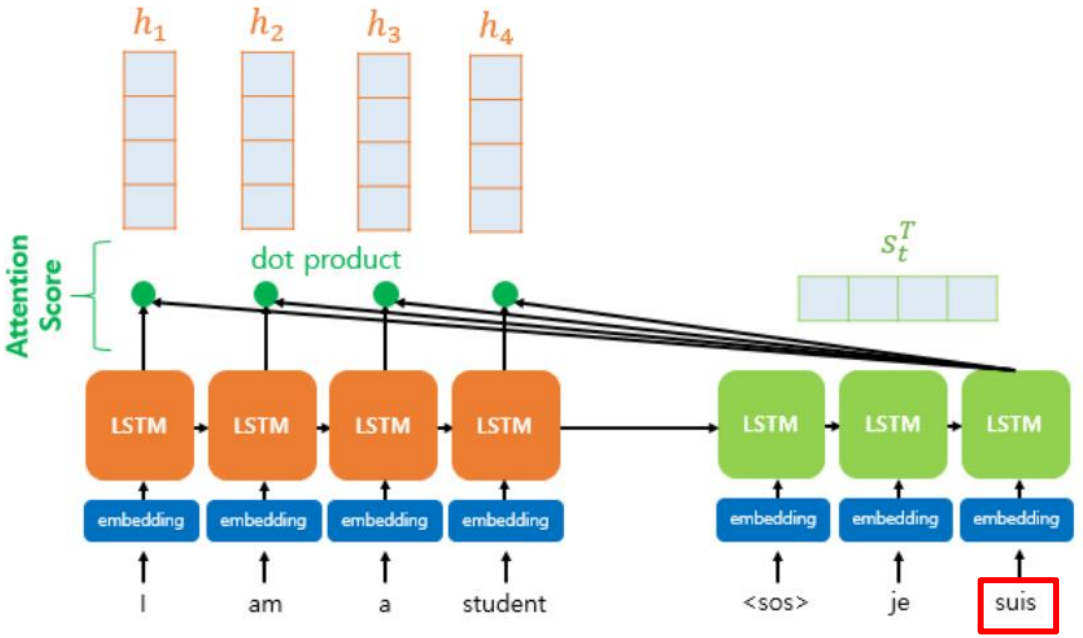
LSTM layer의 특징으로는 들어오는 입력 값의 출력이 hidden state로 나오게 됩니다.
결국 "I" 라는 하나의 입력이 들어오더라도 이는 "I"에 대한 hidden state가 나오게 됩니다.

Attention 연산에서 서로 비교하는 부분은 각 hidden state입니다.

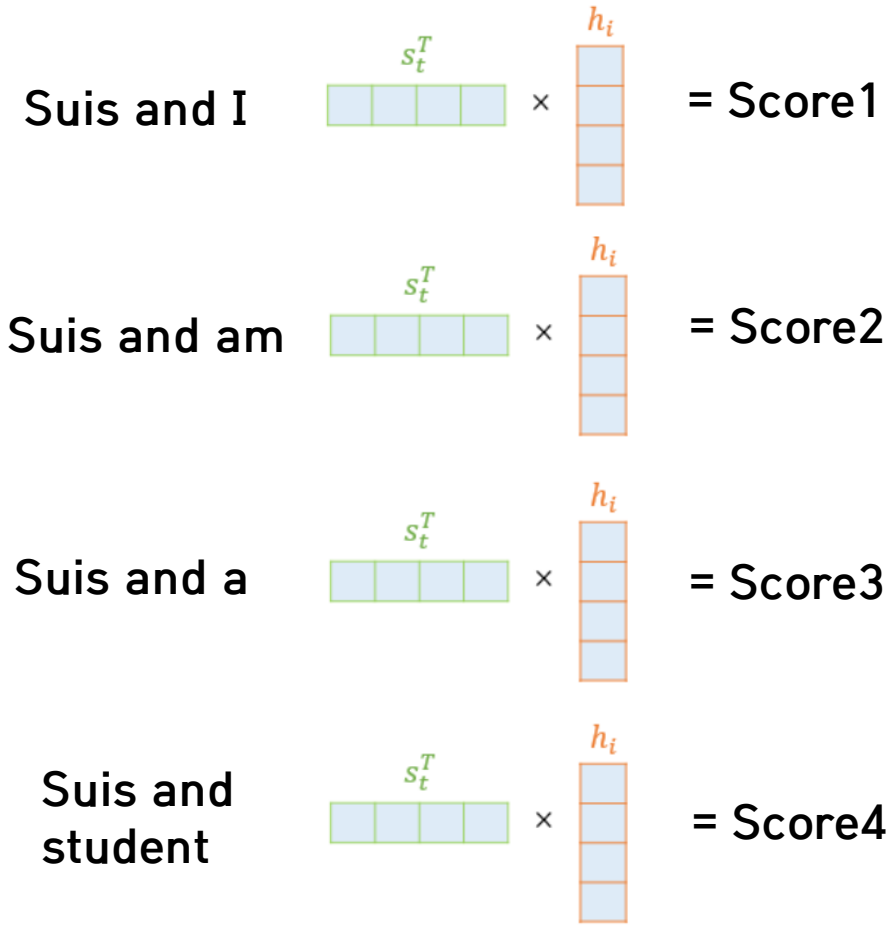
결국 " 나 " 라고 하는 디코더 입력 “hidden state”와
인코더의 ‘I’, ‘am’, ‘Jaehyun’의 hidden state를 서로 비교하여 가장 관련된 것을 말해줍니다.

Attention

서로 비교하는 과정은 내적으로 이루어져 있습니다. (닷 프로덕트)

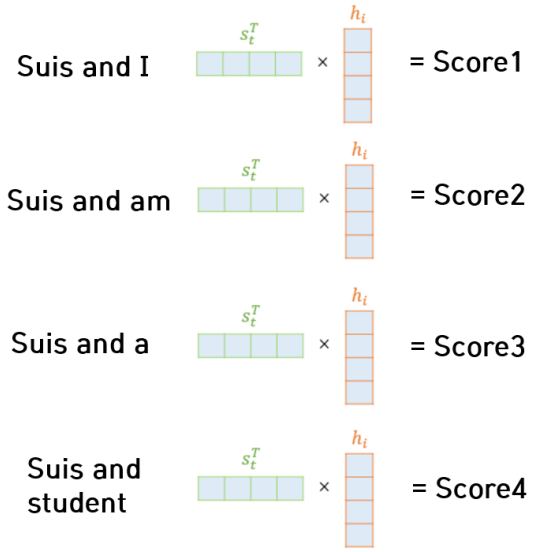


주의 : Context vactor 를 LSTM에 넣지 않음



그림자료 : 딥러닝을 사용한 자연어 처리 입문

Attention



$$\alpha_{ts} = \frac{\exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'=1}^{S} \exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)} \quad \text{[Attention weights]} \quad (1)$$

Softmax(Score1 | Score2 | Score3 | Score4)

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

attention weights

0.02	0.04	0.03	0.89

attention weights

		각 입력에 대한 hidden state					
0.02		I	0.04	0.2	0.3	0.1	0.008 0.004 0.006 0.002
0.04		am	0.2	0.1	0.5	0.3	0.008 0.004 0.02 0.012
0.03	X	a	0.1	0.4	0.71	0.44	0.003 0.012 0.021 0.013
0.89		student	0.23	0.54	0.11	0.78	0.204 0.480 0.097 0.694

=

Attention

0.008	0.004	0.006	0.002
+			
0.008	0.004	0.02	0.012
+			
0.003	0.012	0.021	0.013
+			
0.204	0.480	0.097	0.694

$$c_t = \sum \alpha_{ts} \bar{h}_s \qquad \text{[Context vector]} \qquad (2)$$

=

Context vector

0.223	0.5	0.144	0.721

(입력 전체의 정보를 포함하고 있음)

Concatenate

Context vector

0.223	0.5	0.144	0.721

Suis Hidden state (가정)

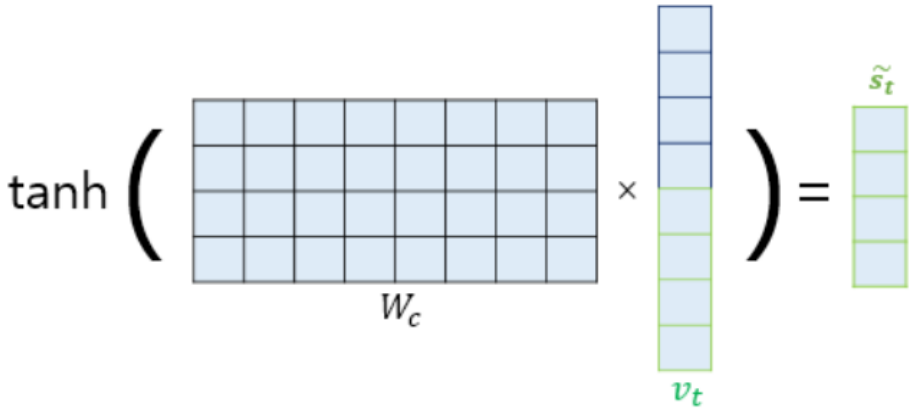
0.2	0.3	0.4	0.5

→

Context vector and Hidden state(Suis) Concatenate vector

0.223	0.5	0.144	0.721	0.2	0.3	0.4	0.5

Attention



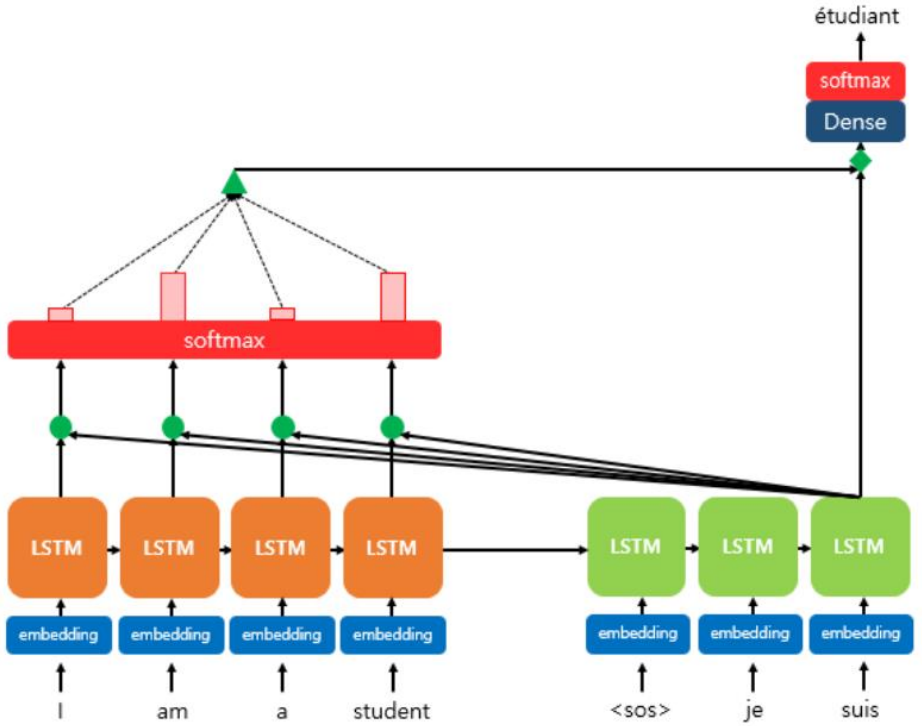
Concatenate vector (Vt) -> fully connected layer
= Attention vector

$$\boldsymbol{a}_t = f(\boldsymbol{c}_t, \boldsymbol{h}_t) = \tanh(\boldsymbol{W}_c[\boldsymbol{c}_t; \boldsymbol{h}_t]) \qquad \text{[Attention vector]} \qquad (3)$$

Attention vector -> fully connected layer(activation : softmax)

$$\hat{y}_t = \text{Softmax}\left(W_y \tilde{s}_t + b_y\right)$$

Seq2seq + Attention
최종 Model의 형태



Attention

1. 바다나우 어텐션

전체적인 흐름은 동일하며,

루 옹 어텐션과 다른 점

1. Attention Score를 구하는 과정에서 학습이 가능한 가중치 행렬을 사용합니다.

2. 구하려고 하는 디코더 입력의 이전 시점의 hidden state를 사용합니다.

3. Context Vector를 디코더 입력에 concat 해준다.

Attention

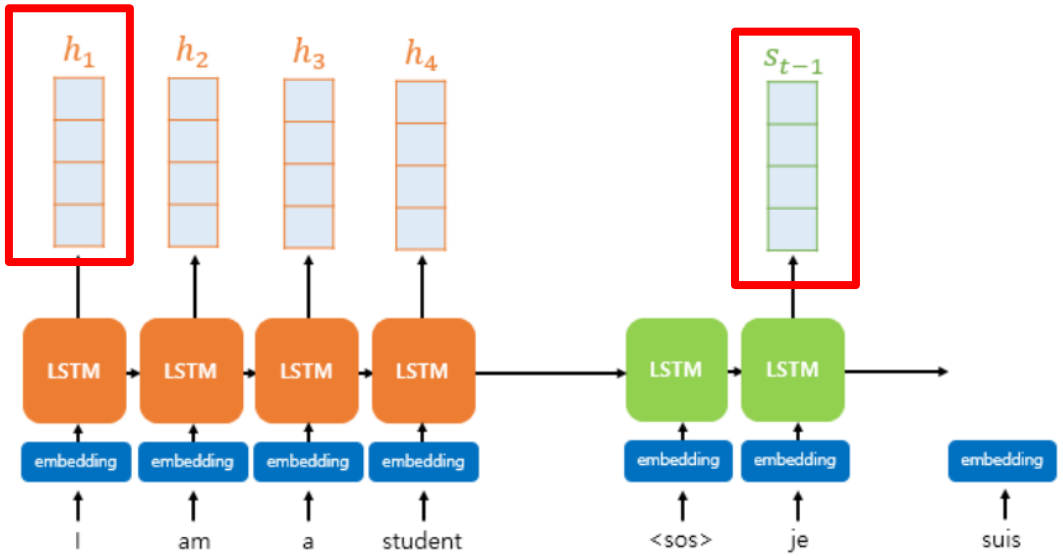
1. Attention Score를 구하는 과정에서 학습이 가능한 가중치 행렬을 사용합니다

$$score(s_{t-1},\ h_i) = W_a^T\ tanh(W_b s_{t-1} + W_c h_i)$$

S(t-1) = 이전 시점의 LSTM hidden state

H(i) = 인코더 부분의 LSTM hidden state

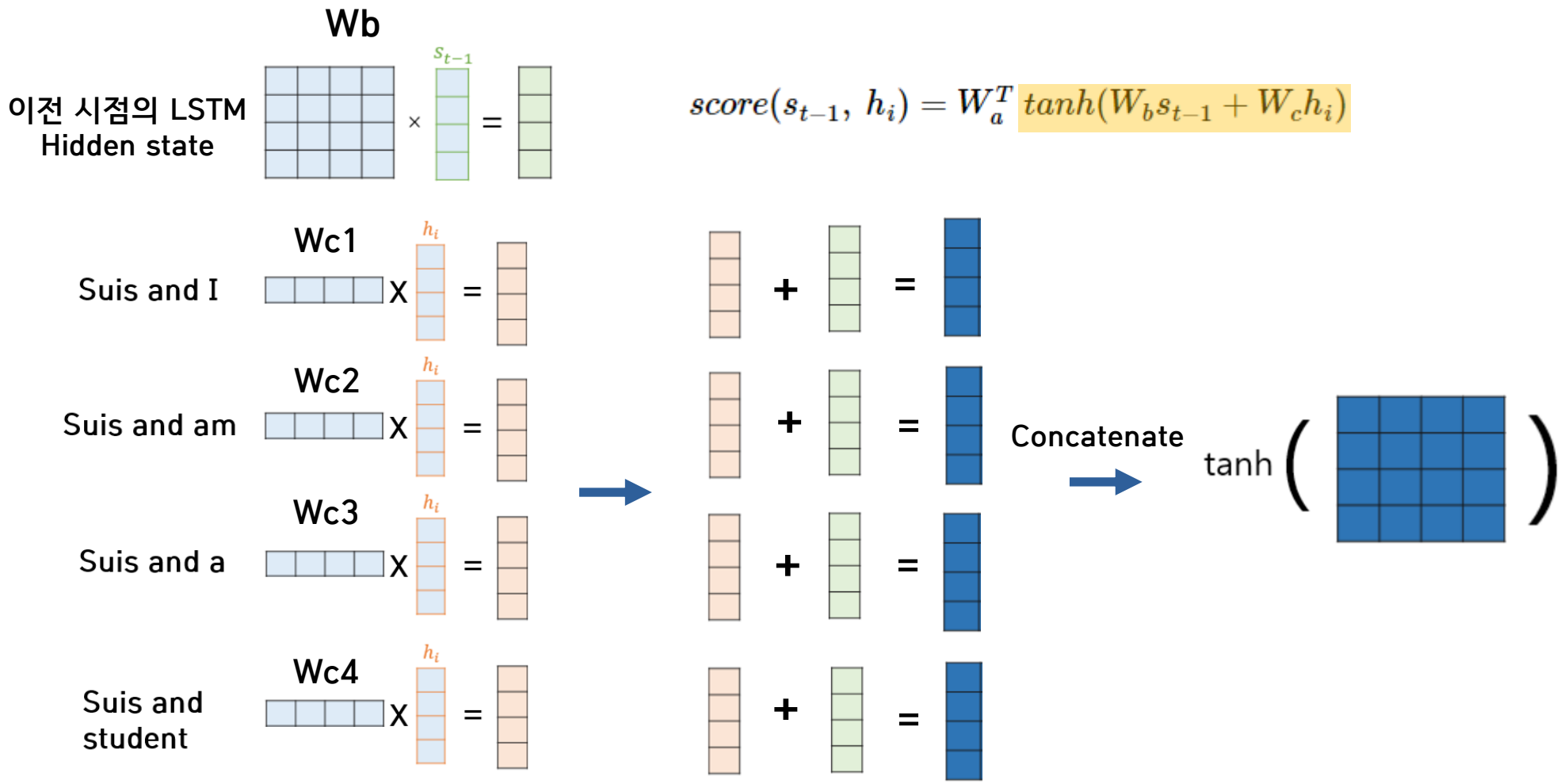
Wa, Wb, Wc : 학습하는 가중치 행렬



그림자료 : 딥러닝을 사용한 자연어 처리 입문

Attention

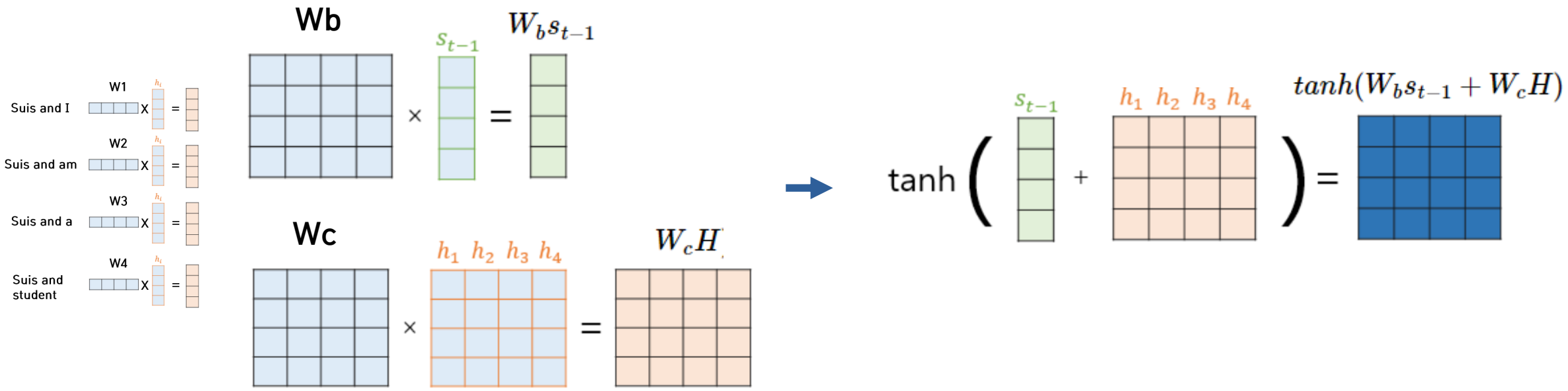
2. 구하려고 하는 디코더 입력의 이전 시점의 hidden state를 사용합니다.



그림자료 : 딥러닝을 사용한 자연어 처리 입문

Attention

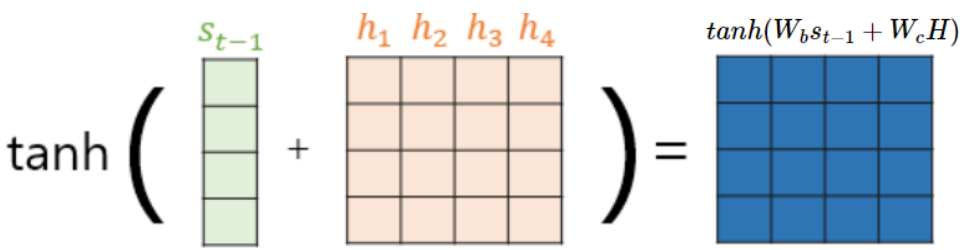
$$score(s_{t-1},\ H) = W_a^T\ tanh(W_b s_{t-1} + W_c H)$$



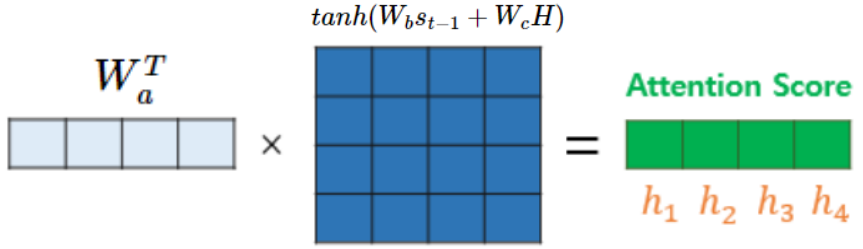
그림자료 : 딥러닝을 사용한 자연어 처리 입문

Attention

$$score(s_{t-1},\ H) = W_a^T\ tanh(W_b s_{t-1} + W_c H)$$



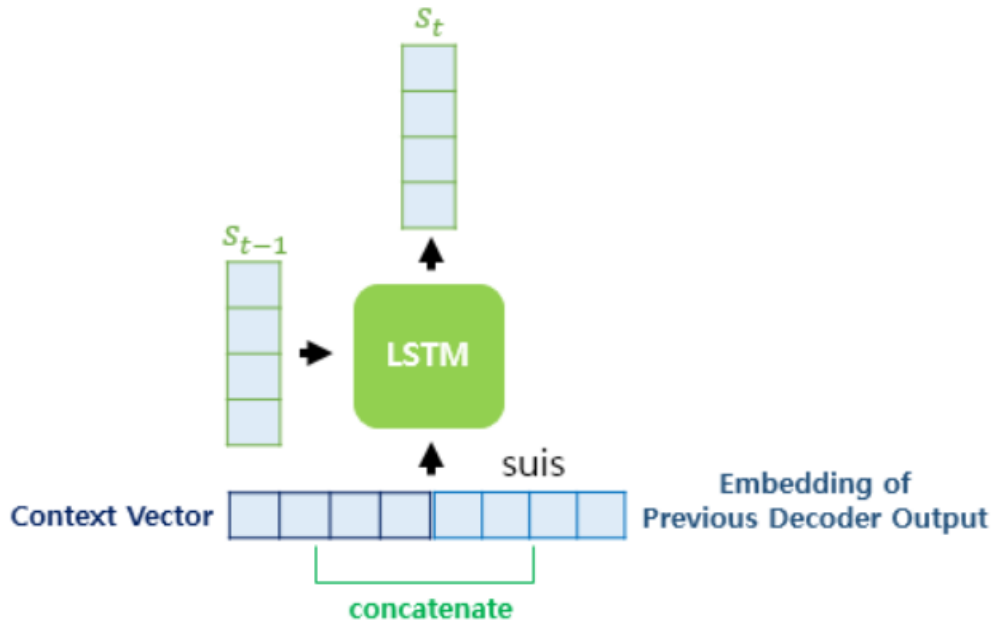
$$score(s_{t-1},\ H) = W_a^T\ tanh(W_b s_{t-1} + W_c H)$$



그림자료 : 딥러닝을 사용한 자연어 처리 입문

Attention

3. Context Vector를 디코더 입력에 concat 해준다.



(루 옹 어텐션과 비슷하게 입력 전체의 정보를 포함하고 있음)

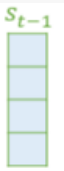
(루 옹 어텐션과 다른 점 Context vector를 LSTM의 입력으로 넣음)

그림자료 : 딥러닝을 사용한 자연어 처리 입문

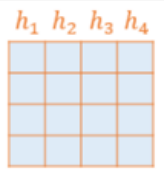
바다나우 어텐션 구현

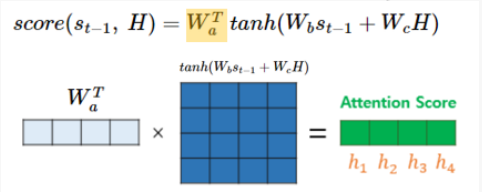
```
class BahdanauAttention(tf.keras.Model):
  def __init__(self, units):
    super(BahdanauAttention, self).__init__()
    self.Wc = Dense(units) #fully connected layer
    self.Wb = Dense(units)
    self.V = Dense(1)

  def call(self, values, query):
    # values : 인코더(입력)의 hidden state 행렬, query : 디코더 t-1시점의 LSTM hidden state
    decoder_hidden = tf.expand_dims(query, 1) # 계산을 위한 차원 맞춤


    score = self.V(tf.nn.tanh(
        self.Wc(values) + self.Wb(decoder_hidden)))



    attention_weights = tf.nn.softmax(score, axis=1) # sc



    context_vector = attention_weights * values



    context_vector = tf.reduce_sum(context_vector, axis=1)
    return context_vector, attention_weights
```

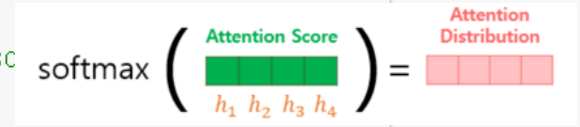
query

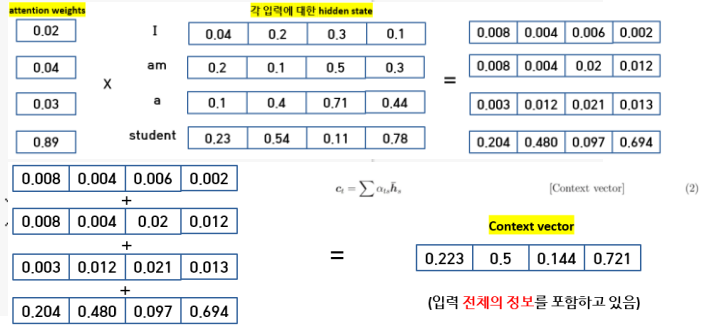


values









적용한 교육 결과

DACON

진행 중인 대회

[정규대회] 2021 농산물 가격예측 AI 경진대회

시계열 | 농넷 | 한국농수산식품유통공사 | 농산물 | NMAE

거래량, 가격의 데이터를 받아서, 농산물의 1주, 2주, 4주 후의 가격을 예측하는 대회

date	요일	배추_거래량(kg)	배추_가격(원/kg)	무_거래량(kg)	무_가격(원/kg)	양파_거래	양파_가격	건고추_거	건고추_가
2016-01-01	금요일	0	0	0	0	0	0	0	0
2016-01-02	토요일	80860	329	80272	360	122787.5	1281	3	11000
2016-01-03	일요일	0	0	0	0	0	0	0	0
2016-01-04	월요일	1422742.5	478	1699653.7	382	2315079	1235	699	4464
2016-01-05	화요일	1167241	442	1423482.3	422	2092960	1213	1112.6	4342
2016-01-06	수요일	1045507.5	442	1904372.1	409	1860569	1263	1672	7041
2016-01-07	목요일	1039925	448	1438990.2	428	1868012	1241	1191	3908
2016-01-08	금요일	942655	420	1563537.8	390	1483395	1218	672.4	4836
2016-01-09	토요일	689121	389	1723983.9	345	1107263	1210	5456.6	5112
2016-01-10	일요일	0	0	0	0	0	0	0	0
2016-01-11	월요일	1161028	398	1940798.7	336	1651530	1215	73	5712
2016-01-12	화요일	730460.1	431	1748847.8	309	1606583	1161	6321	5498
2016-01-13	수요일	642654	429	1416822	307	1211231	1166	758.8	6879
2016-01-14	목요일	777445.9	441	1412984	306	1313759	1148	1330	4350
2016-01-15	금요일	793655	449	1437415	307	1187834	1146	995	6643
2016-01-16	토요일	537832.1	454	1033808	306	990370.5	1150	92	5246

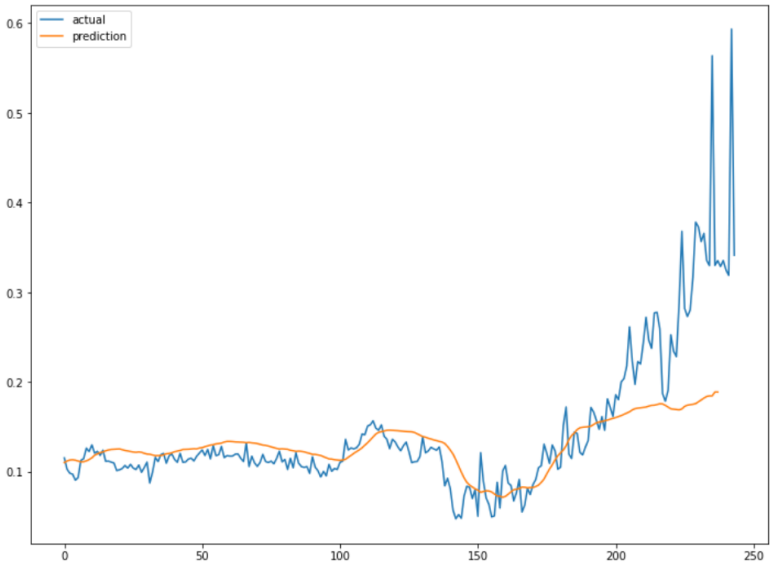
Data : 배추, 무, 양파, 건고추 등 21개의 채소 가격과, 거래량 등이 나와 있음

이 중 배추의 데이터만 가지고 와서 예측하는 모델을 비교해 볼 예정

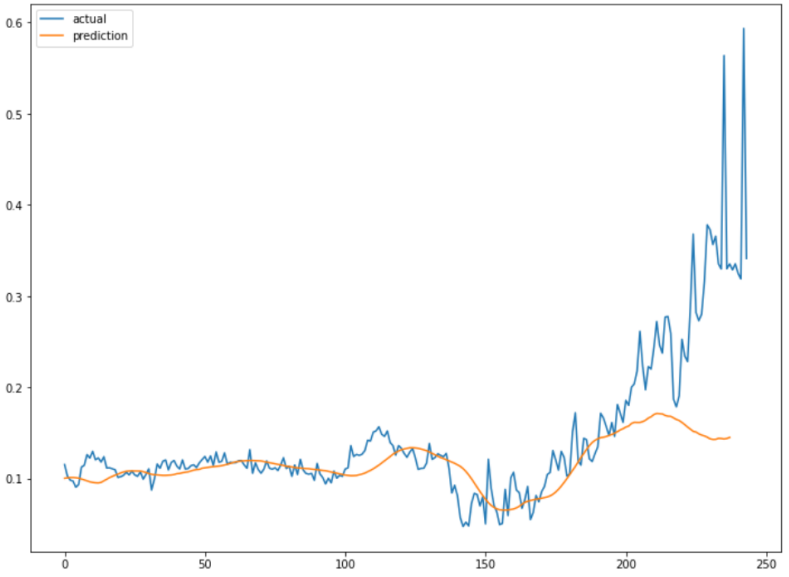
적용한 교육 결과

배추의 일주일 뒤 가격 예측

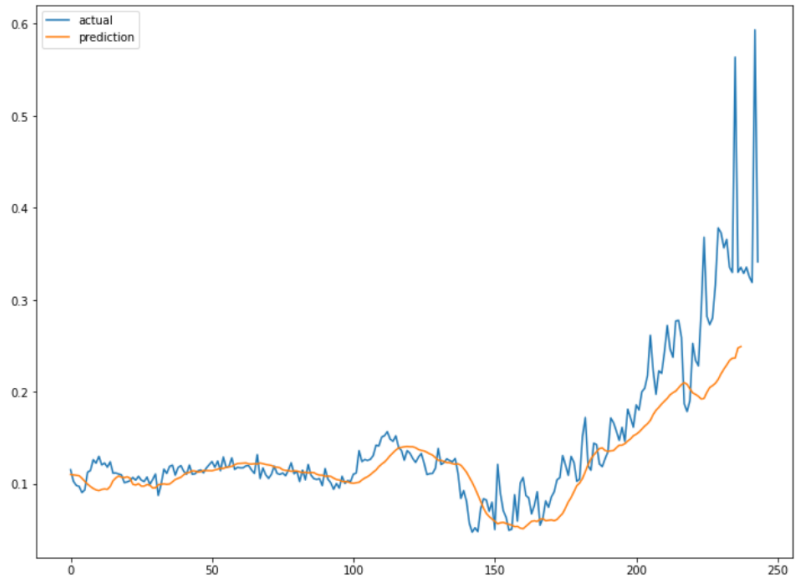
모델 1 LSTM



모델 2 LSTM + Attention



모델 3 CNN + 양방향 LSTM + Attention



적용한 교육 결과

감사합니다.