

NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

2021년 한국외대 자연어처리 과정 논문 발표

<A반 3조> 박주영(팀장), 강민영, 김세진, 박가연, 이성민, 조혁준

CONTENTS

01

Abstract

02

Background

03

Alignment Model

04

Code Implement

05

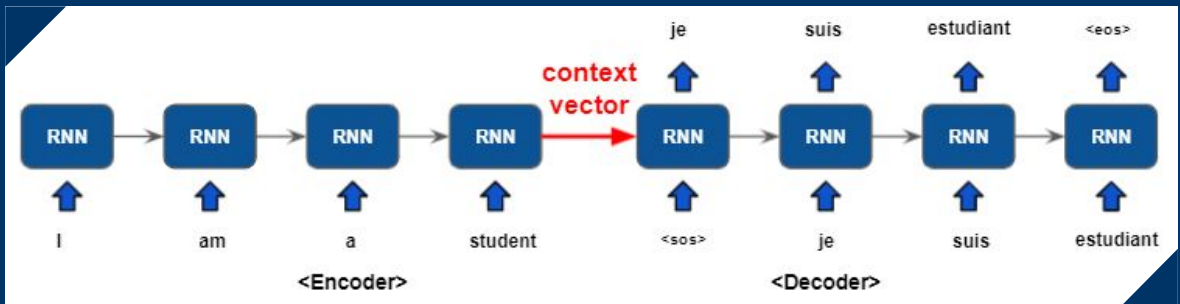
Experiment

01 ABSTRACT

본 논문에서 제안한 방식 요약



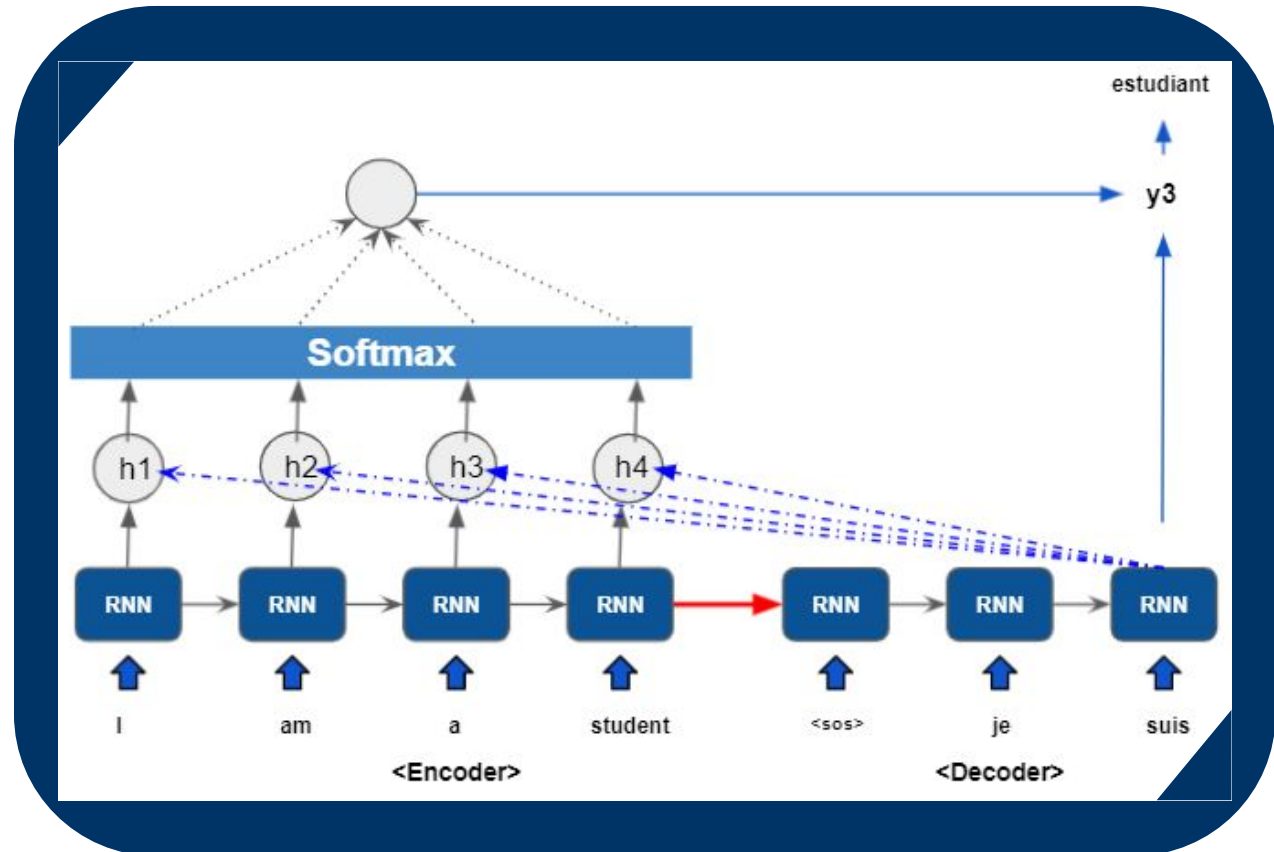
ABSTRACT



<RNN Encoder-Decoder 모델 구조>

NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

- Neural Machine Translation은 기계 번역 분야에서 새로 발견된 방법이다
- 기존 LSTM 기반의 Encoder-Decoder 모델 (Seq2Seq Model)은 Sequential Problem에 뛰어난 성능을 지닌다
- **BUT** 기존 Encoder-Decoder 아키텍처는 **Bottleneck 문제** 발생한다
- 본 논문에서는 이러한 문제를 완화할 수 있는 **Soft-Alignment**를 제안한다



<Alignment Model>

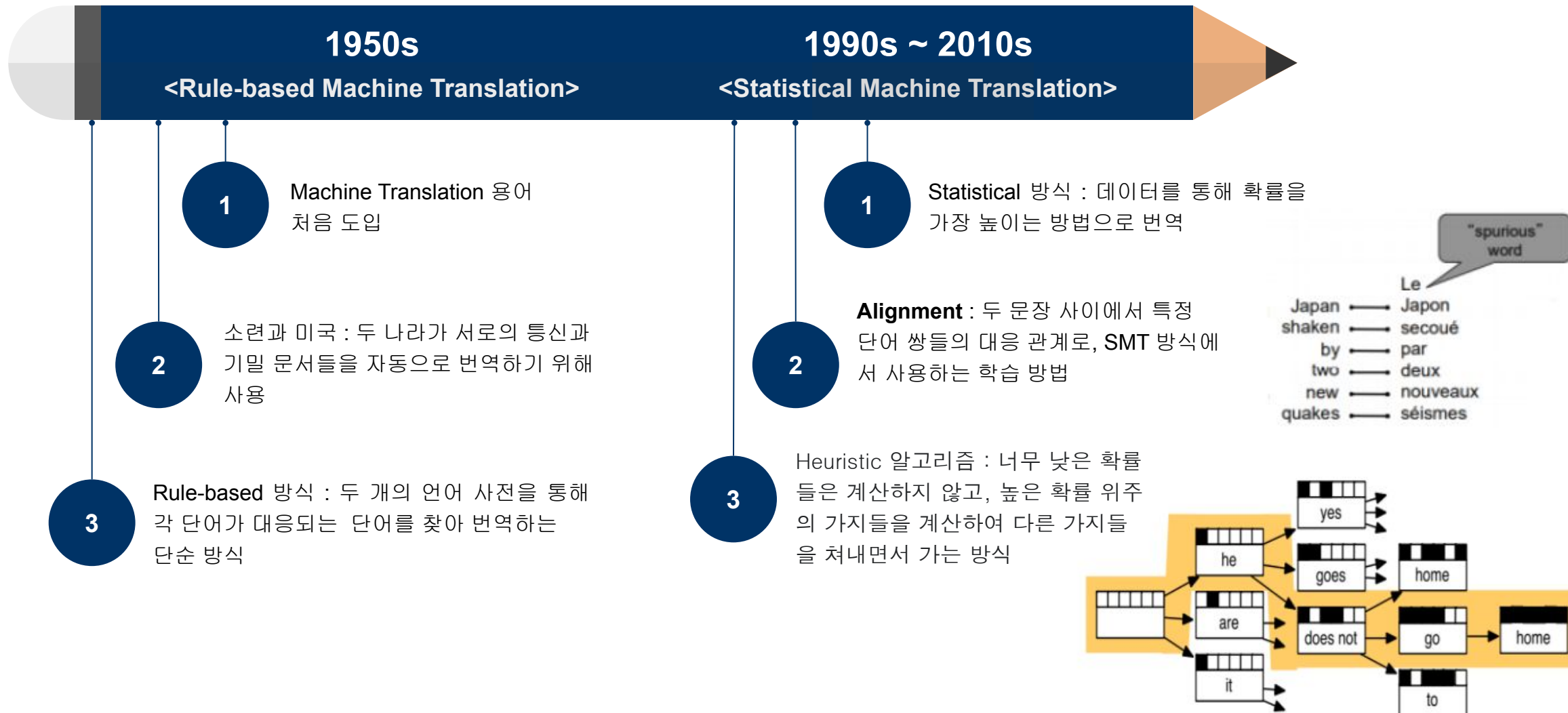
02 BACKGROUND

필요한 배경 지식과 Alignment Model이 필요한 이유

2. Alignment Model이 필요한 이유

2-1. Background

(1) 기계 번역(Machine Translation)



2. Alignment Model이 필요한 이유

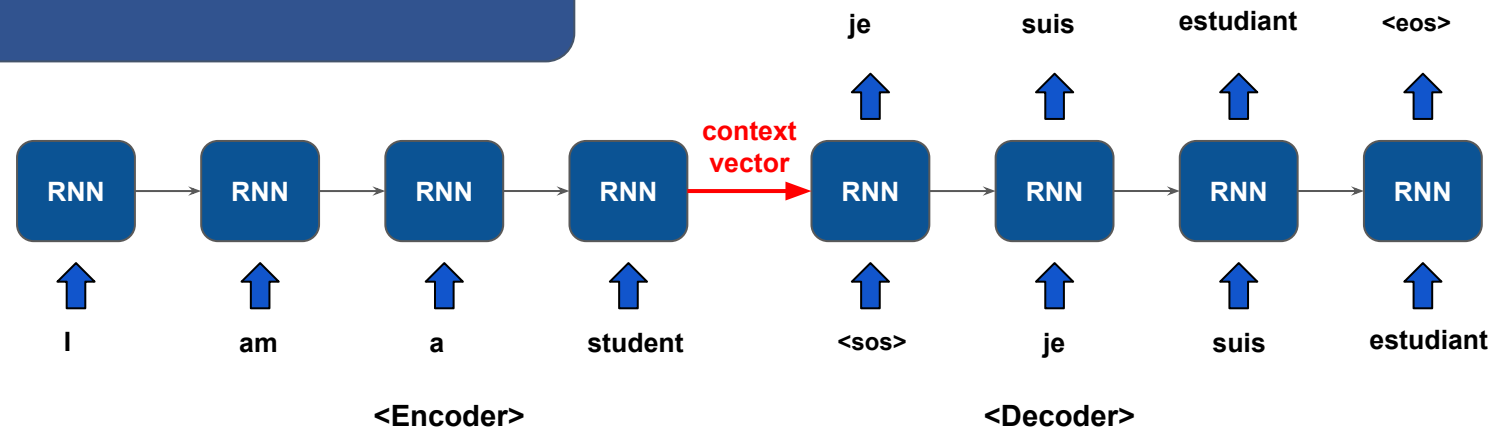
2-1. Background

(2) 확률론적 관점에서 기계 번역(Machine Translation)

- 기계 번역은 <확률적인 접근 방법>을 통해 수행된다
- Source Sentence x 가 주어졌을 때, 조건부 확률 $p(y|x)$ 를 최대화 하는 Target Sentence y 를 찾는다

$$\arg \max_y P(y | x)$$

Neural Machine Translation

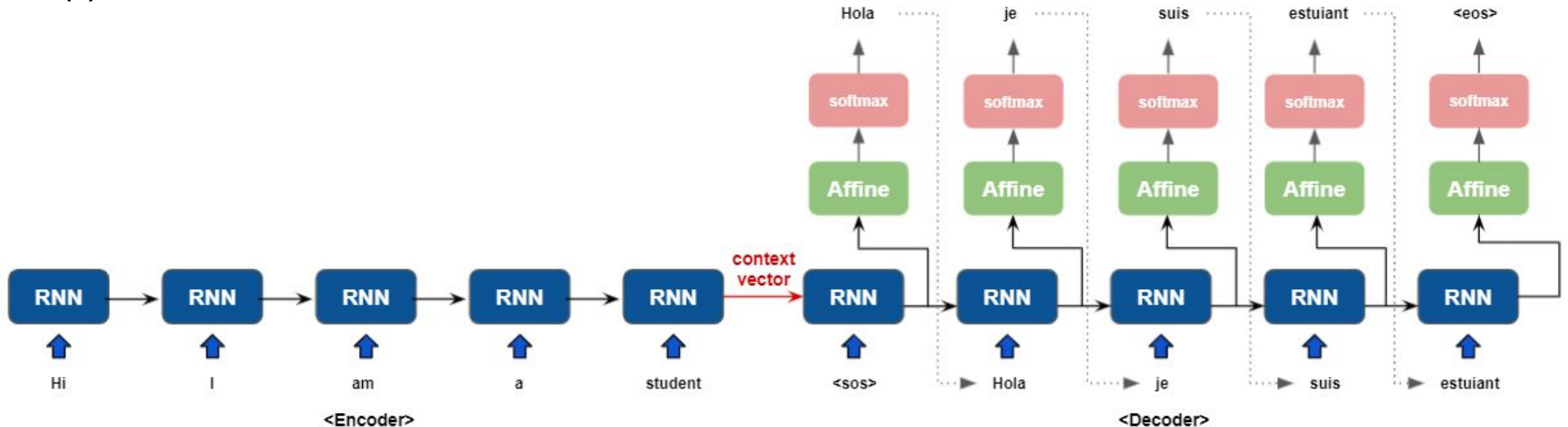


- NMT는 조건부 확률을 최대화 하기 위해 **번역 데이터 쌍**을 통해 학습한다
- 최근 NMT 모델은 2개의 RNN 모델인 **Encoder-Decoder** 구조를 사용한다

2. Alignment Model이 필요한 이유

2-1. Background

(3) RNN Encoder-Decoder 구조



<RNN Encoder-Decoder Architecture>

Sequence to Sequence Task 중 기계 번역(Machine Translation)에 많이 사용하는 모델

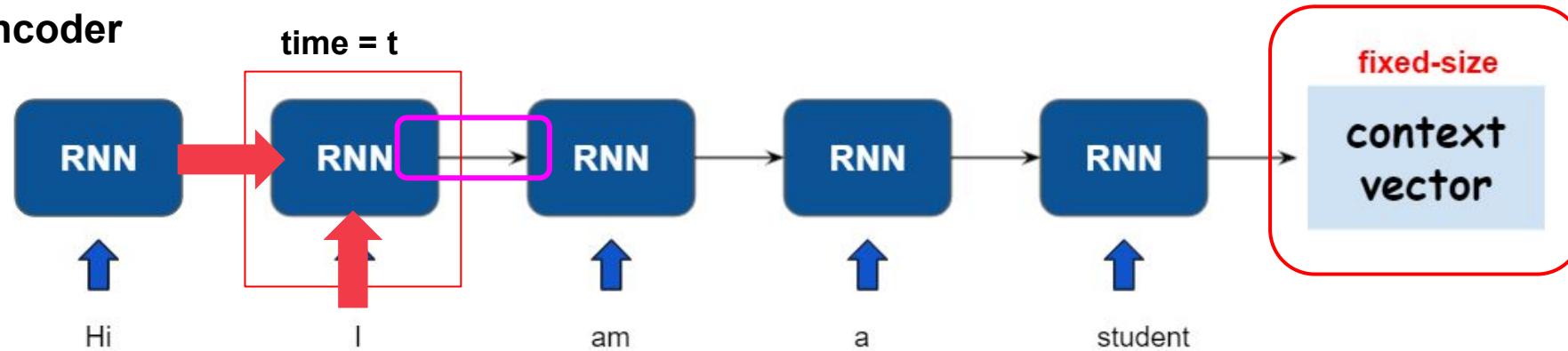
"Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation"

Cho, K., van Merriënboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. (2014a)

2. Alignment Model이 필요한 이유

2-1. Background

(3-1) RNN Encoder



<RNN Encoder Architecture>

Context vector

- 현재 시점을 t 라 할 때, RNN 셀은 $t-1$ 에서의 hidden state와 t 에서의 input data를 입력으로 받아, t 에서의 hidden state를 생성한다
- 만들어진 hidden state는 다음 시점인 $t+1$ 의 hidden state의 입력으로 전달된다
- 이렇게 모든 input 단어를 받으면 **Encoder RNN 셀의 마지막 시점 hidden state를 Decoder에 넘겨주는데, 이를 Context Vector라 한다**

Context vector에는 input sequence의 정보가 응축 (encoding) 되어 있다.

수식

$$(1) \ x = (x_1, \dots, x_{T_x})$$

▶ 각 time step에서 하나의 input data 존재

$$(2) \ h_t = f(x_t, h_{t-1})$$

▶ RNN 연산을 통해 $h_1 \sim h_t$ 의 hidden state 출력

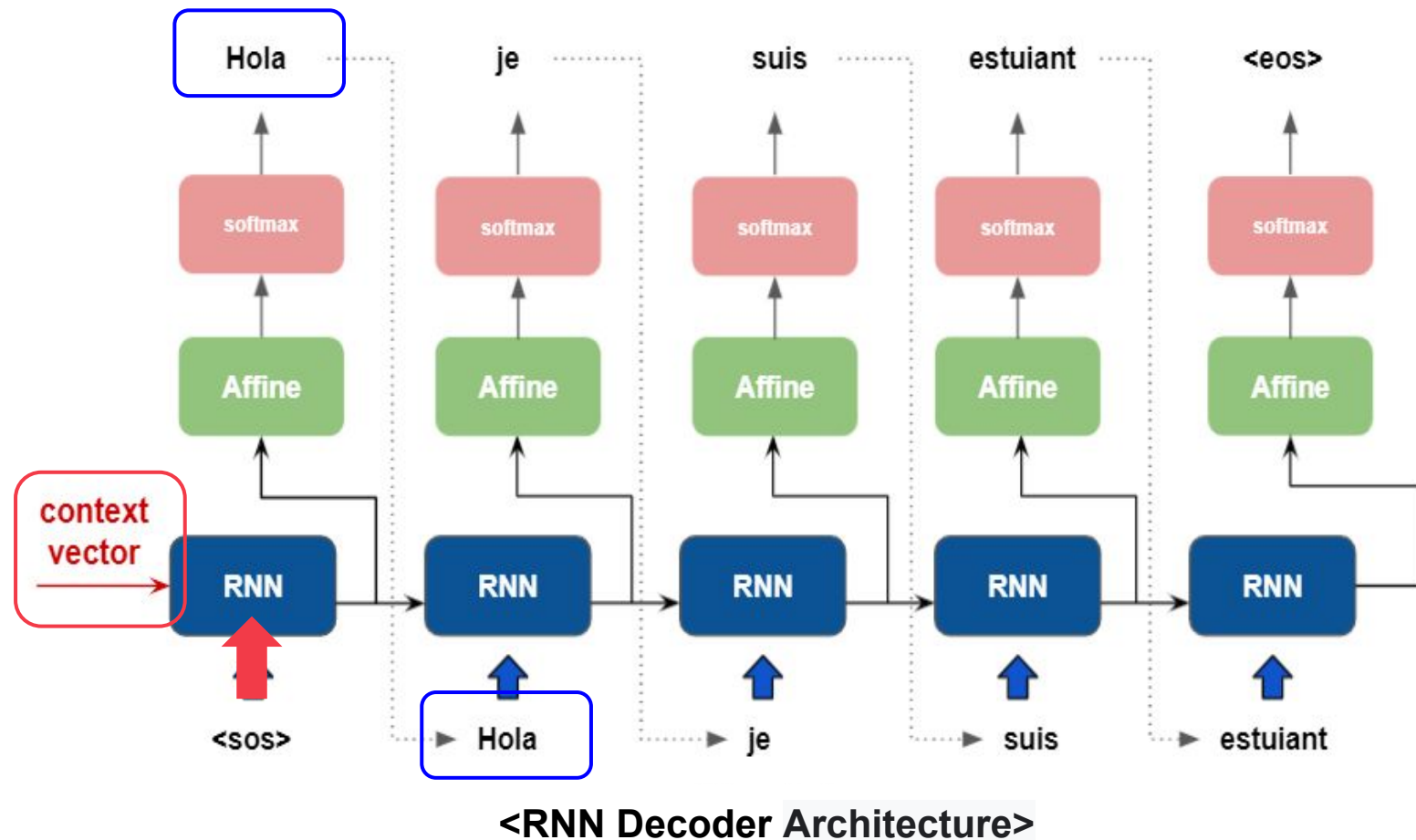
$$(3) \ c = q(\{h_1, \dots, h_{T_x}\})$$

▶ Encoder는 모든 hidden state의 결과를 하나의 context vector에 압축하여 정보를 저장

2. Alignment Model이 필요한 이유

2-1. Background

(3-2) RNN Decoder



Decoder

- **Decoder**는 전달 받은 **Context vector**를 자신의 첫 **hidden state** 입력 값으로 사용한다
- 이 hidden state와 첫 시작 토큰 **<sos>**를 시작으로 다음 time step에 나올 target word를 예측한다
- 그리고 이 예측된 단어는 다음 time step의 입력으로 들어가 **<eos>** 토큰을 만나기 전까지 계속 진행한다
- **Decoder**는 번역 **sequence**를 출력한다

수식

$$(1) \quad p(y) = \prod_{t=1}^T p(y_t || \{y_1, \dots, y_{t-1}\}, c)$$

▶▶ time step=t 일때, 이전 시점에서 나온 결과 값을 사용해 다음 나올 y 값을 예측

$$(2) \quad p(y_i || y_1, \dots, y_{i-1}, x) = g(y_{i-1}, s_i, c)$$

▶▶ i번째 target word가 y_i가 등장할 조건부 확률에 대한 수식

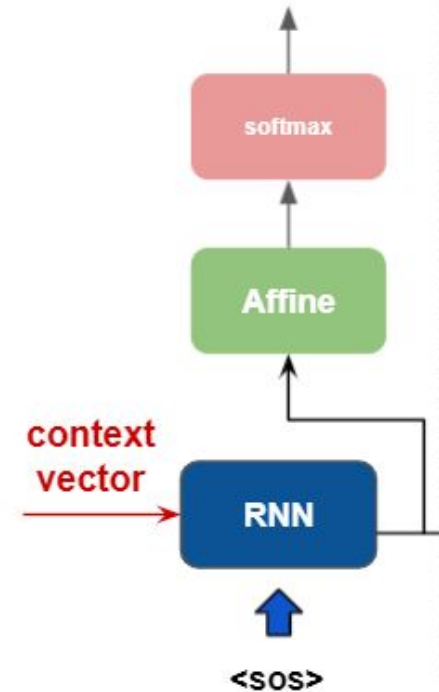
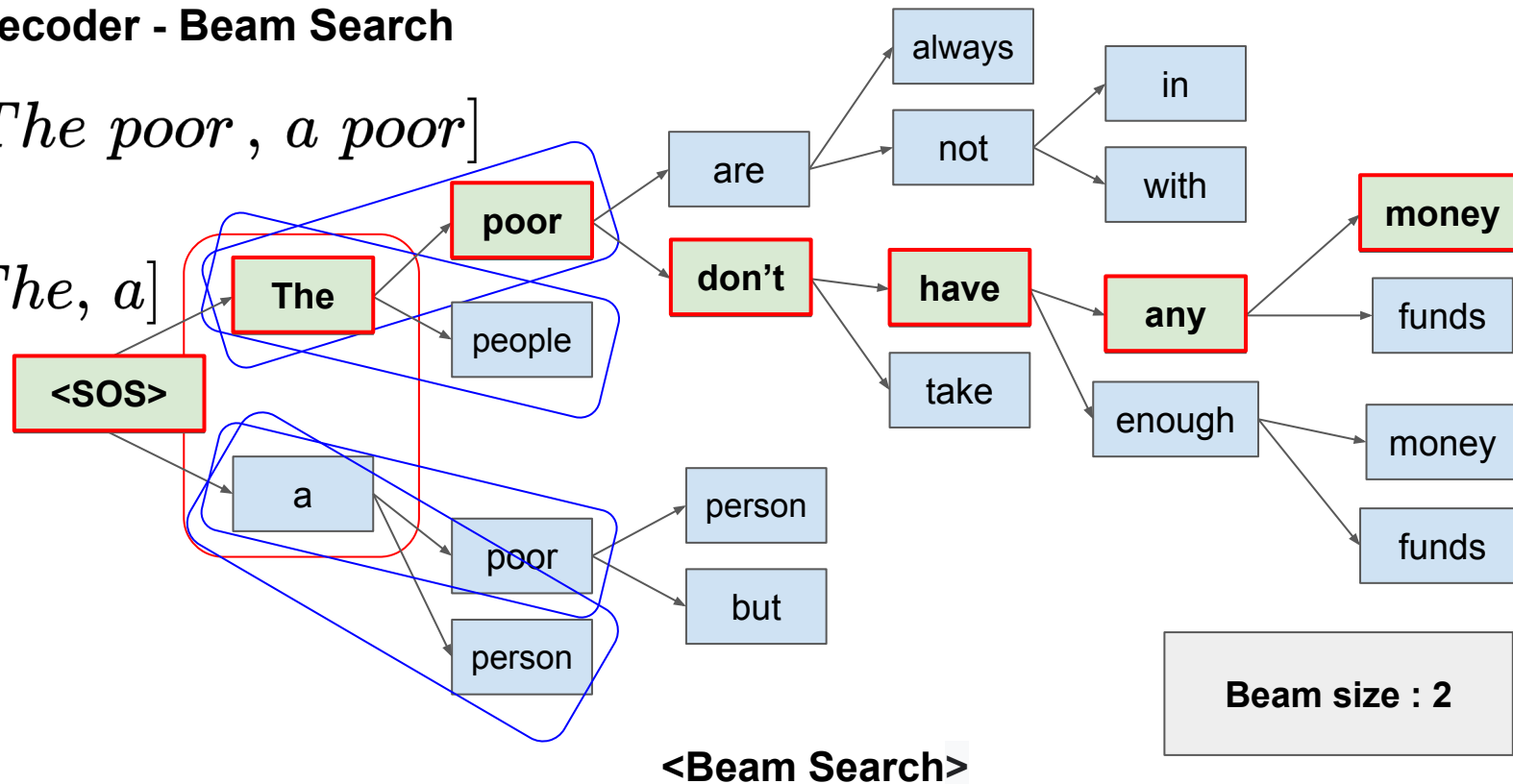
2. Alignment Model이 필요한 이유

2-1. Background

(3-3) RNN Decoder - Beam Search

$$P(y_1, y_2 || x) = [The\ poor, a\ poor]$$

$$P(y_1 || x) = [The, a]$$

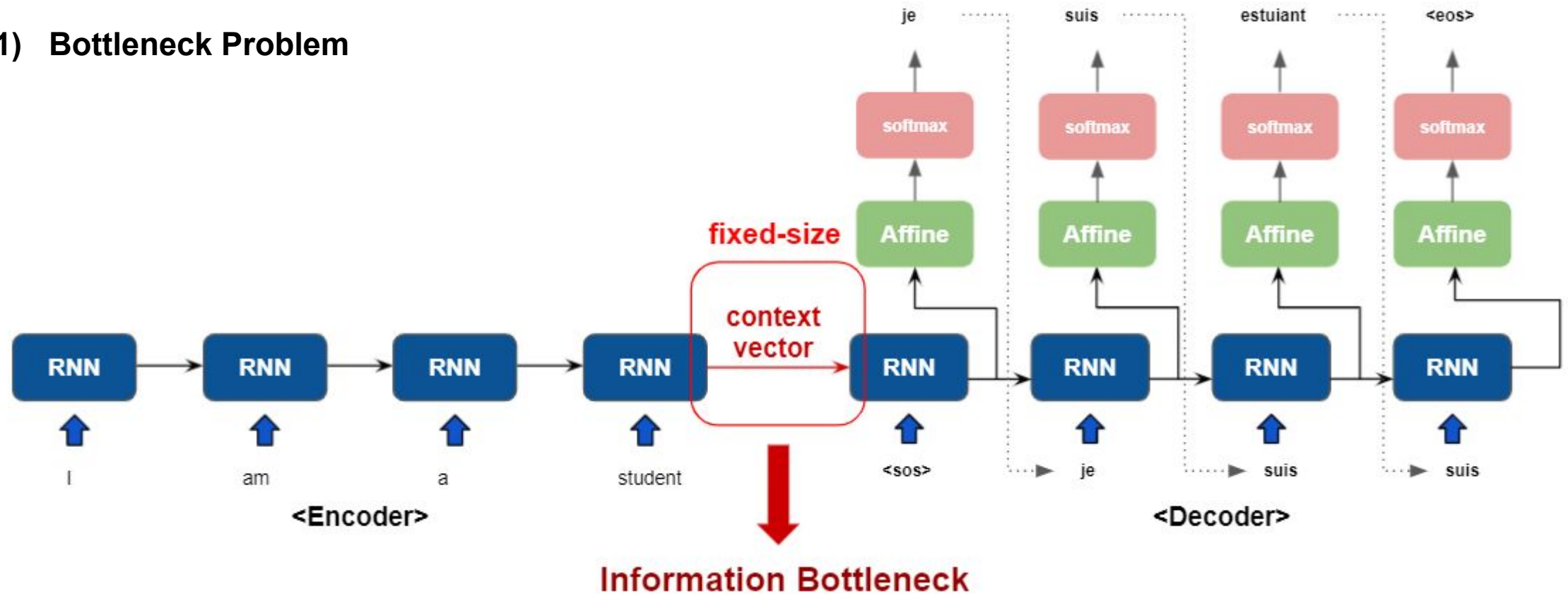


Beam Search는 매번 하나의 출력 값이 아닌 Beam 개수(k) 만큼 출력 값을 도출하여 다양성을 주고, 마지막에 가장 좋은 **Sequence**가 무엇인지 판단하도록 하는 **Decoding 방법**이다

2. Alignment Model이 필요한 이유

2-2. RNN Encoder-Decoder 한계 및 해결책

(1) Bottleneck Problem



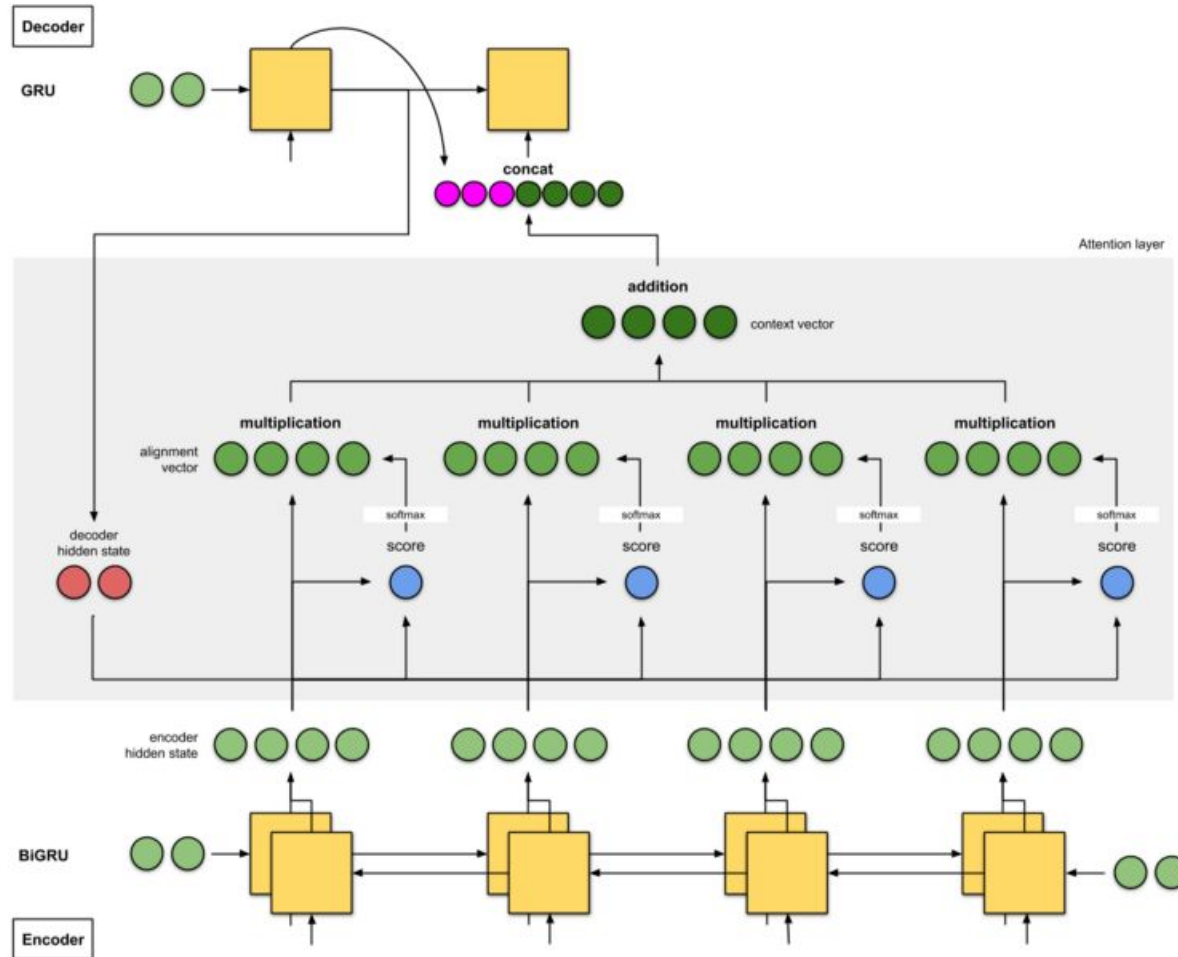
<Information Bottleneck>

- input sequence에 대한 모든 정보를 고정된 사이즈(fixed-size)인 context vector에 담는다
- 따라서 input sequence의 길이가 굉장히 길어지면, 이전 정보들이 유실될 가능성이 높아지기 때문에 정보 손실 문제점이 발생한다

2. Alignment Model이 필요한 이유

2-2. RNN Encoder-Decoder 한계 및 해결책

(1) Bottleneck Problem - Solved with Alignment Model



<Alignment Model>

- **Alignment Model**은 기존 NMT보다 훨씬 더 유연하고 부드러운 정렬과 번역을 하게한다
- 기존 인코딩 방법 = context vector에 모든 정보가 압축되어서 정보의 병목현상 발생

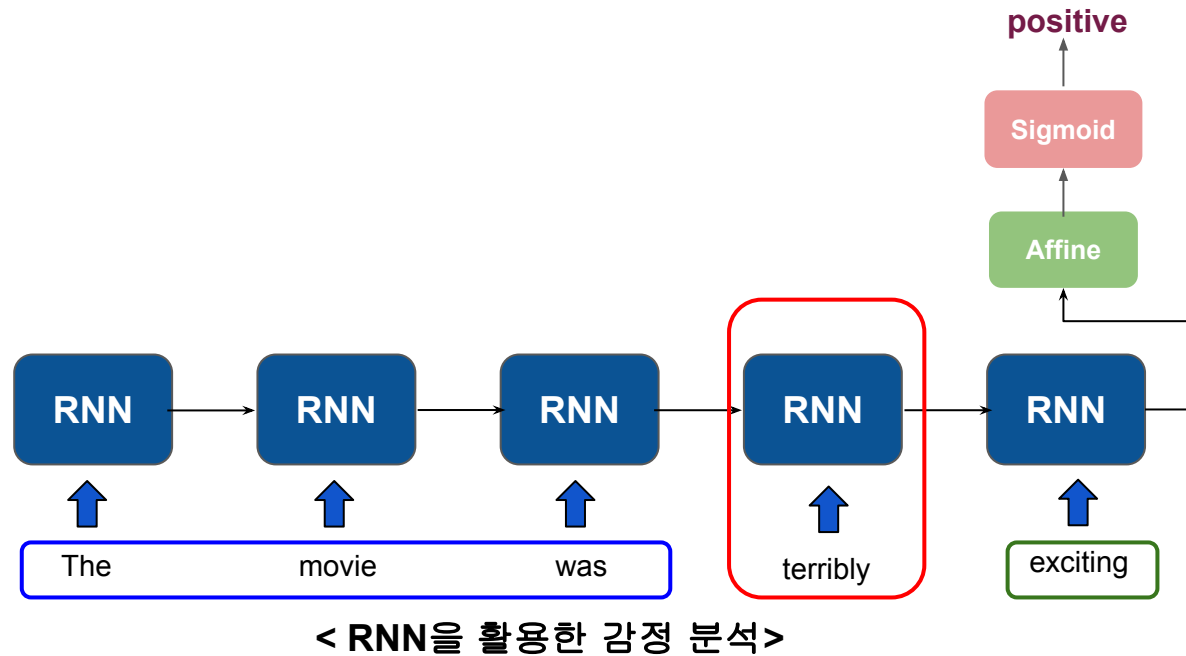


Alignment(Attention)을 사용하여
각 인코딩의 각 단계마다 집중을 해주어
정보의 흐름을 마지막에 집중시키지 않고,
부드럽게 정렬(Soft-Alignment) 한다

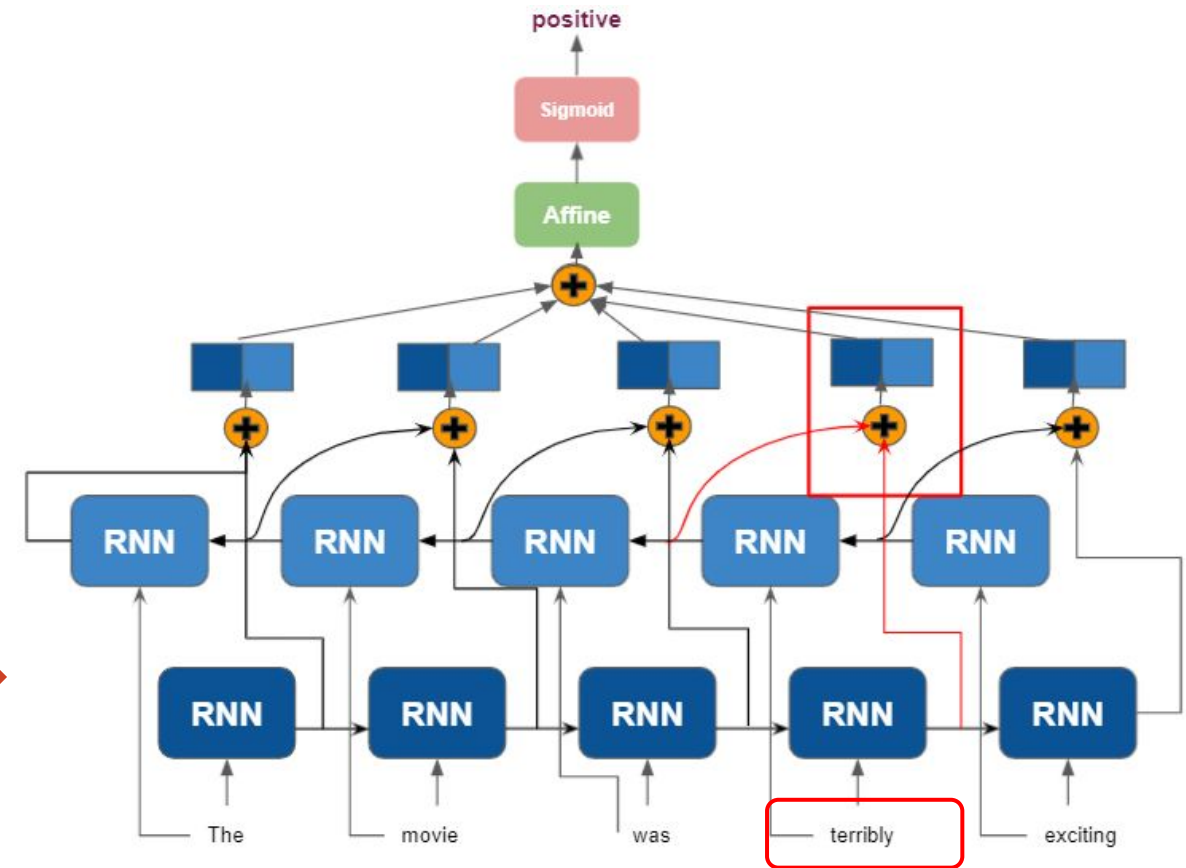
2. Alignment Model이 필요한 이유

2-2. RNN Encoder-Decoder 한계 및 해결책

(2) RNN Problem - Bidirectional RNN Solve



- RNN은 일반 Neural Networks에 비해 데이터의 이전 상태 정보를 “메모리” 형태로 저장할 수 있는 장점이 있다
- 하지만 RNN, LSTM, GRU 등 모두 단방향으로만 정보를 기억하기 때문에 “정보의 불균형”이 존재한다
- 즉, 정방향 뿐만 아니라 역방향까지 고려한다면 유의미한 결과를 도출할 수 있다



< Bidirectional RNN을 활용한 감정 분석 >

- Bidirectional RNN은 2개의 Hidden Layer를 가지고 있다
 - **Forward state** 정보를 가지는 hidden layer
 - **Backward state** 정보를 가지는 hidden layer
- 두 hidden layer에서 나온 출력을 Concatenate해서 과거 정보와 미래 정보를 기반으로 진행한다.

03 ALIGNMENT MODEL

본 논문이 제안한 방식 Alignment Model (Attention mechanism) 에 대한
자세한 과정



3. Alignment Model

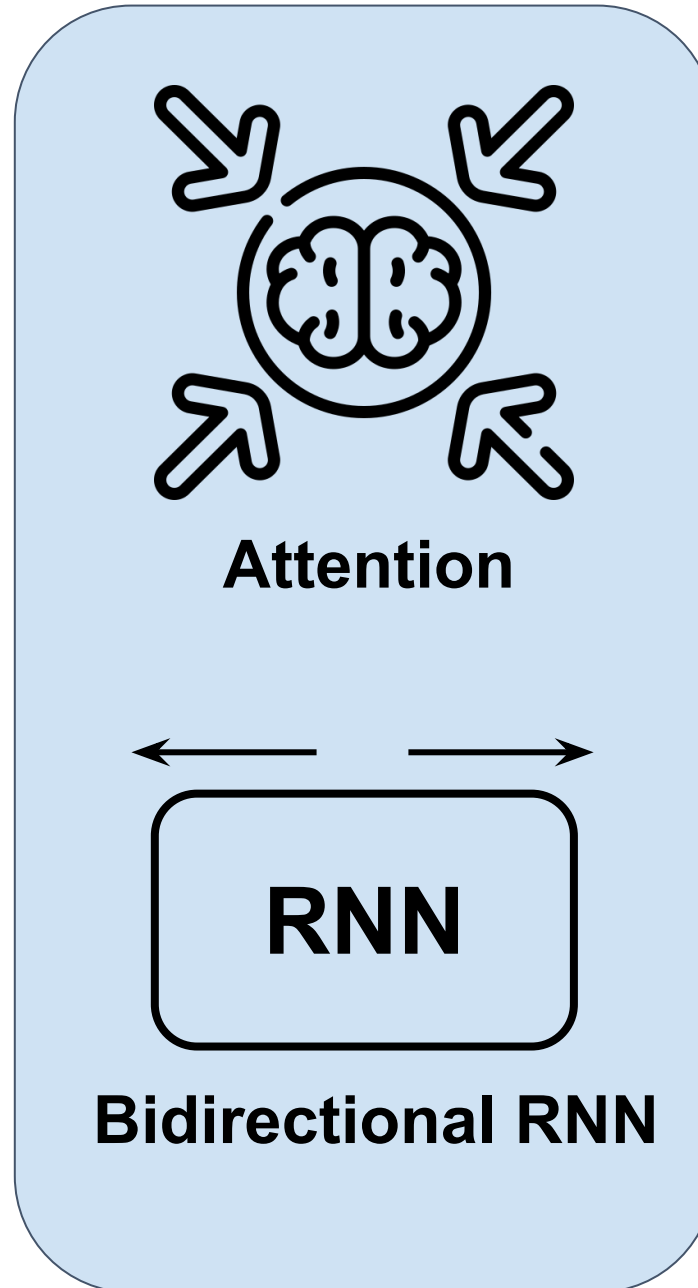
3.1 Attention

Seq2Seq

Problem

- Bottleneck Problem
- Forward RNN Problem

+



=

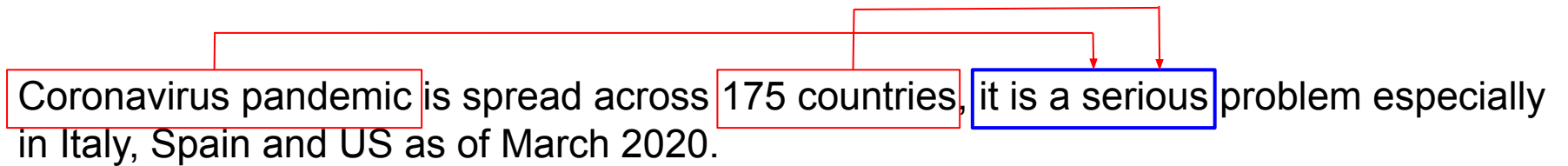


3. Alignment Model

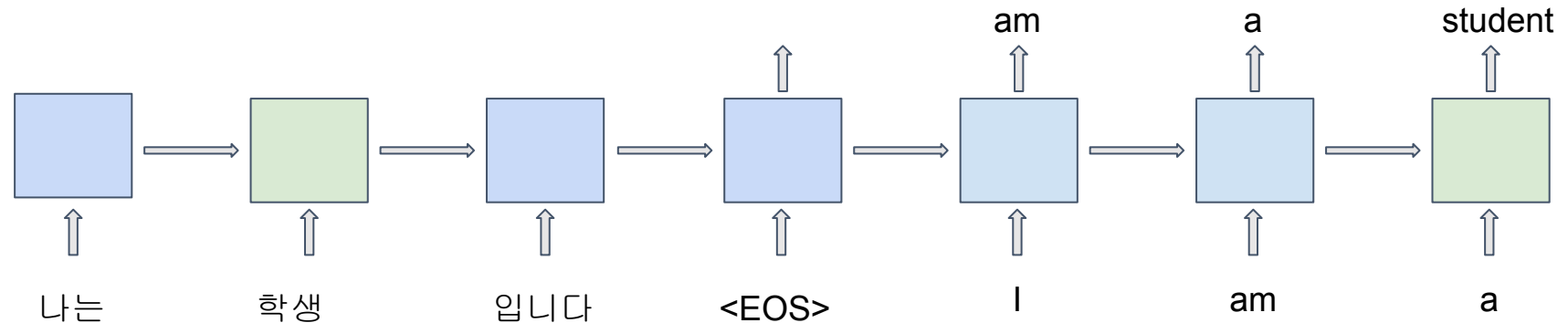
3.1 Attention

1. 이어지는 두 문장 간의 Attention

Coronavirus pandemic is spread across 175 countries, it is a serious problem especially in Italy, Spain and US as of March 2020.

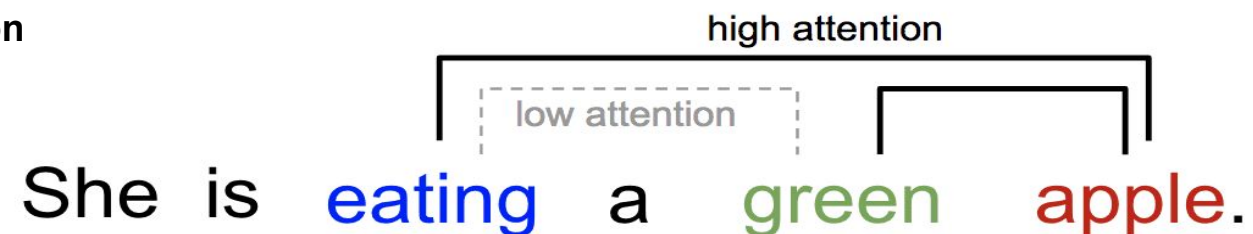


2. Pair Data 문장 간의 Attention



3. Self-Attention

She is eating a green apple.

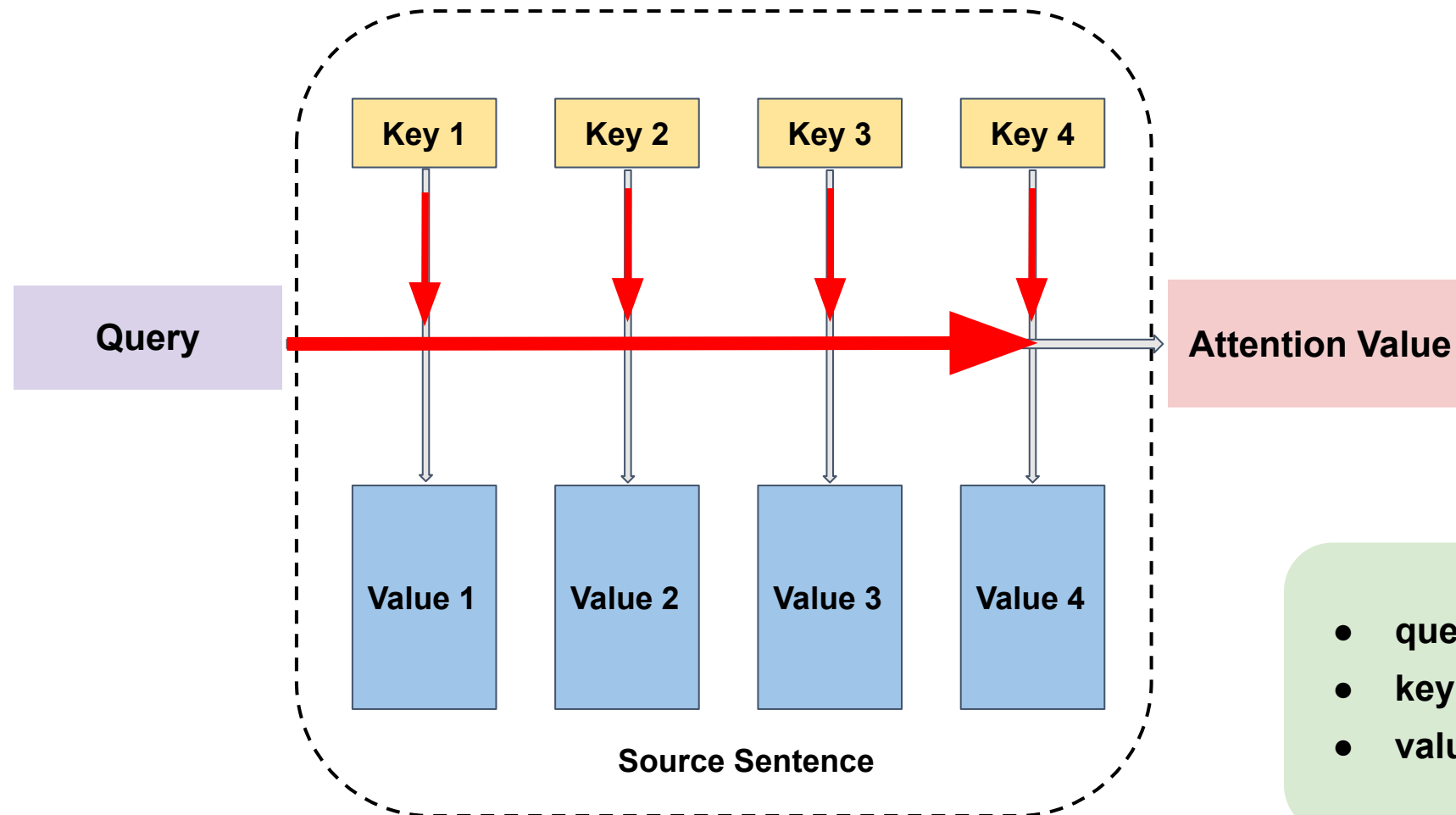


Type of
Attention

3. Alignment Model

3.1 Attention

$$\text{Attention}(\text{Query}, \text{Source}) = \sum_{i=1}^{L_s} \text{Similarity}(\text{Query}_i, \text{Key}_i) \cdot \text{Value}_i$$

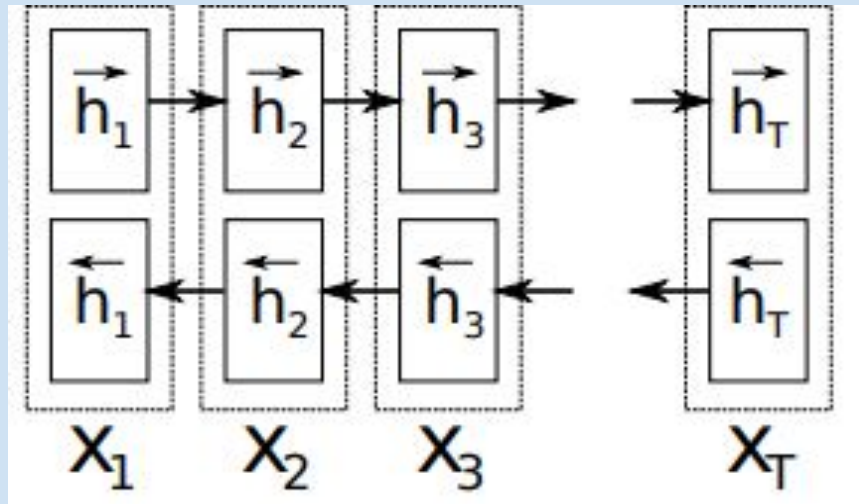


- query : 물음의 주체가 되는 값
- key : 정보를 제공하는 단어집합
- value: key에 대한 의미적 결과

3. Alignment Model

3.2 Encoder

(1) Bidirectional GRU



<Bidirectional GRU>

forward hidden state

$$(\vec{h}_1, \dots, \vec{h}_{T_x})$$

backward hidden state

$$(\overleftarrow{h}_1, \dots, \overleftarrow{h}_{T_x})$$

\oplus

$$h_i = \begin{bmatrix} \vec{h}_i \\ \overleftarrow{h}_i \end{bmatrix}$$

Hidden state가 의미하는바는
Encoder의 **input** 단어와 그 주변의 단어의 의미를 담고 있는것이다.

3. Alignment Model

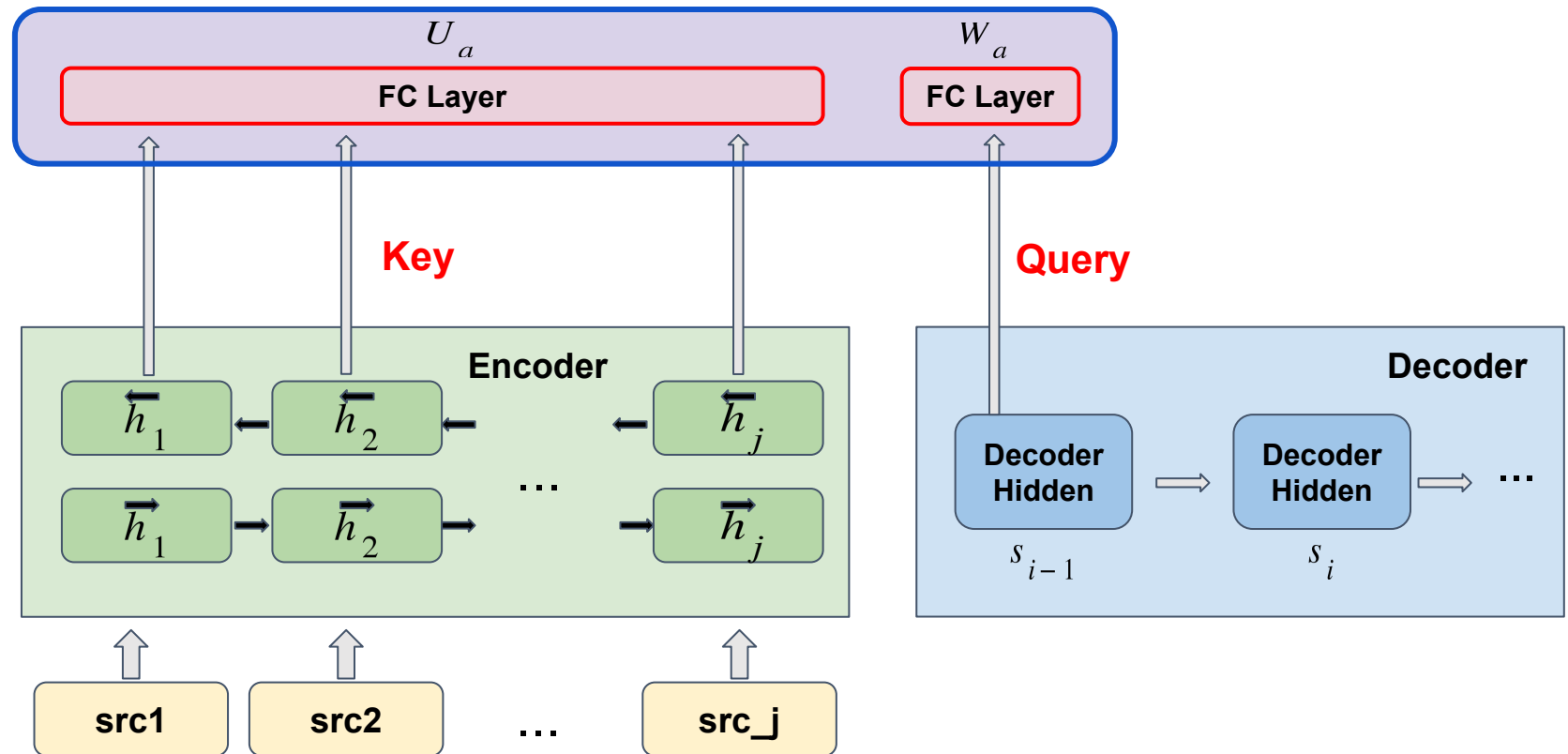
3.2 Encoder

(2) Attention Energy

$$a(s_{i-1}, h_j) = v_a^\top \tanh(W_a s_{i-1} + U_a h_j)$$

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

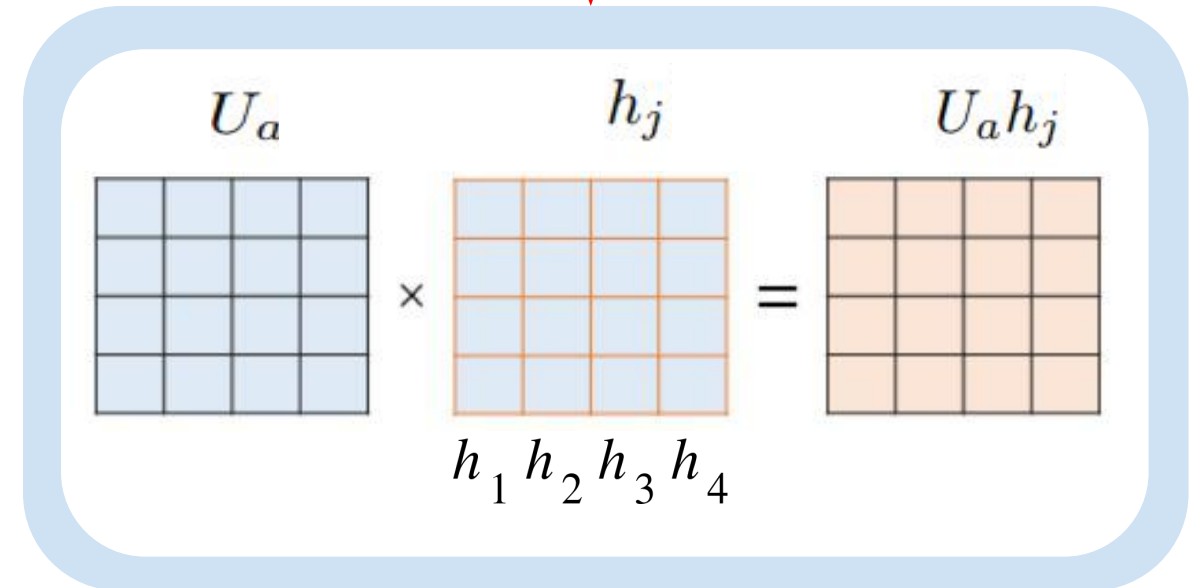
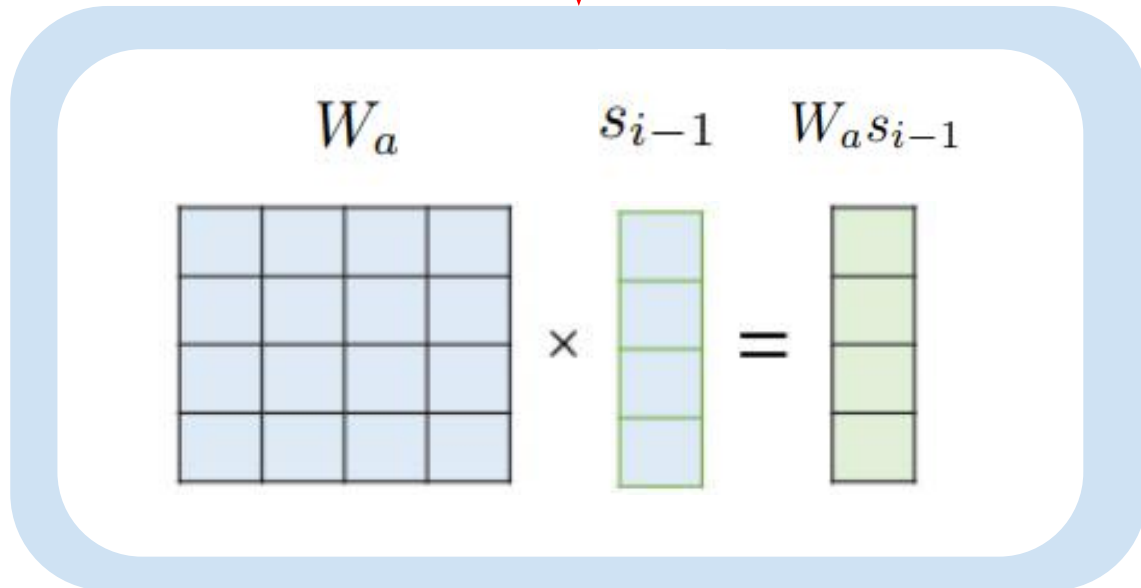
Query와 Key 사이의
연관성을 나타내주는
Attention Energy



3. Alignment Model

3.3 Bahdanau Attention

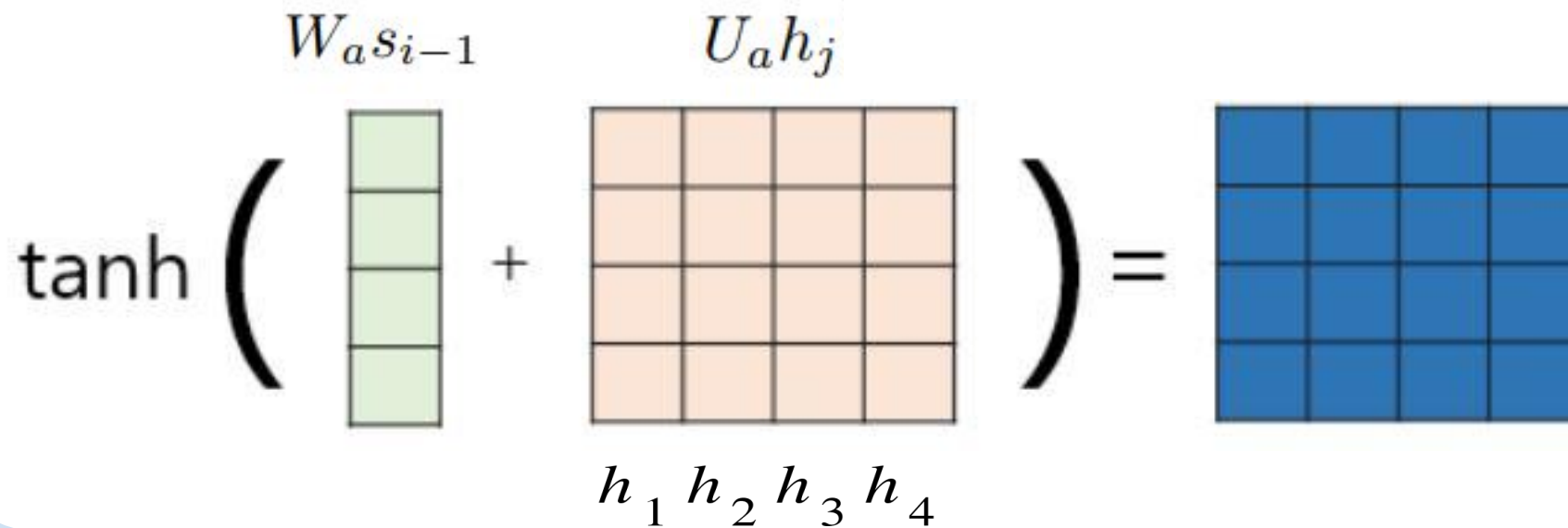
$$a(s_{i-1}, h_j) = v_a^\top \tanh(W_a s_{i-1} + U_a h_j)$$



3. Alignment Model

3.3 Bahdanau Attention

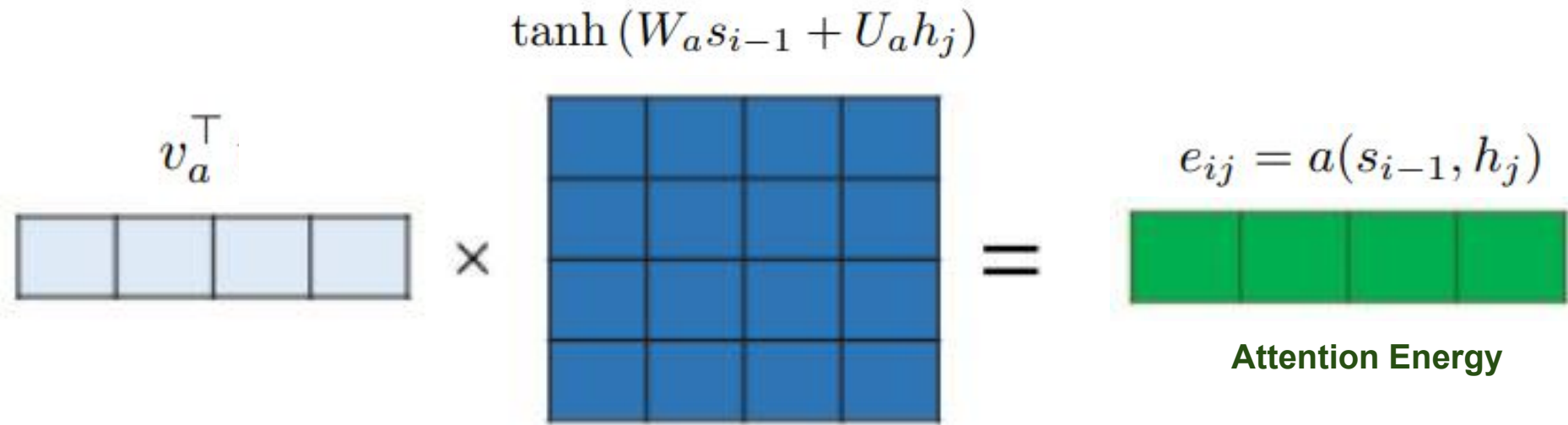
$$a(s_{i-1}, h_j) = v_a^\top \tanh(W_a s_{i-1} + U_a h_j)$$



3. Alignment Model

3.3 Bahdanau Attention

$$a(s_{i-1}, h_j) = v_a^\top \tanh(W_a s_{i-1} + U_a h_j)$$



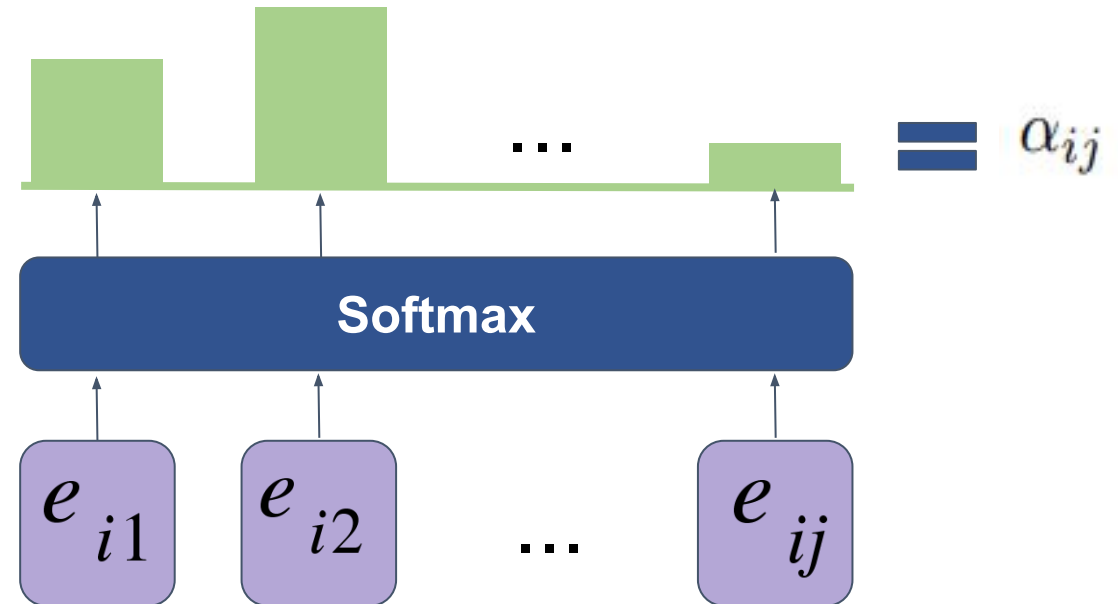
3. Alignment Model

3.4 Decoder

(1) Attention Distribution

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

<SoftMax>



Bahdanau attention을 통해 얻은 energy e_{ij} 값에 softmax 함수를 통과시켜 e_{ij} 를 확률화 시킨다

계산된 α_{ij} 는 0~1 사이의 값을 가지는데, 이는 입력시점의 hidden state에 대한 가중치, '시간의 가중치'를 의미한다

3. Alignment Model

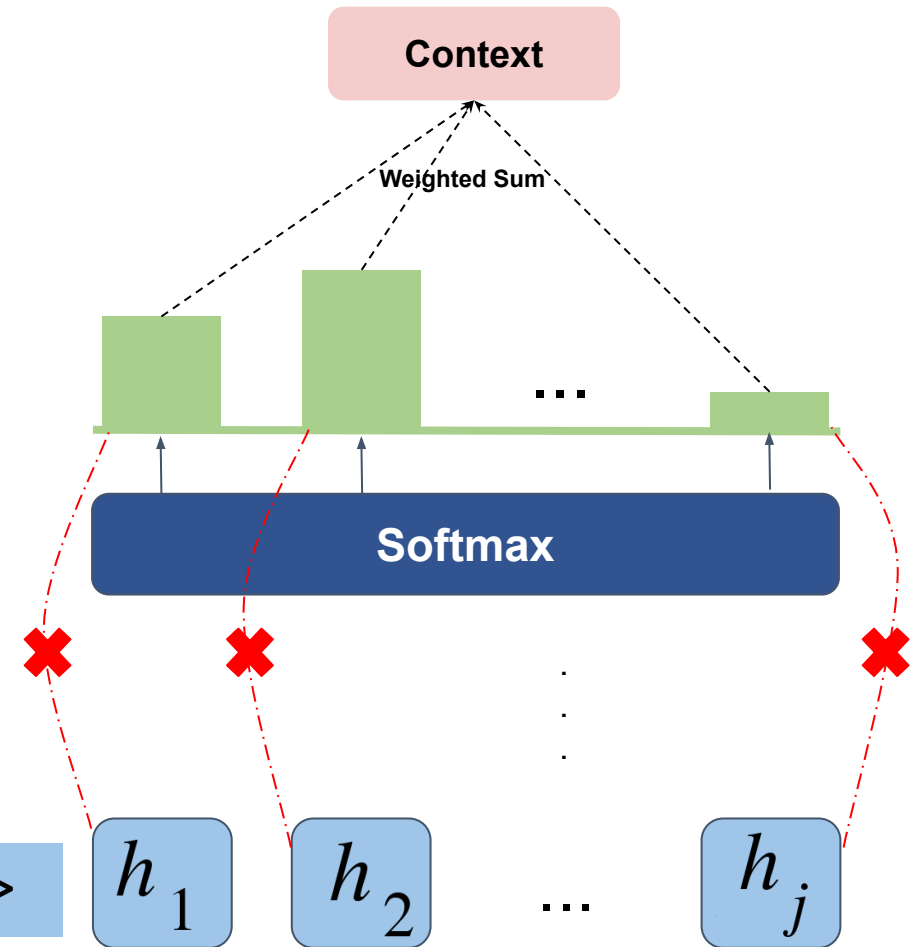
3.4 Decoder

(2) Context vector

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

<Context Vector>

<Encoder Hidden States>

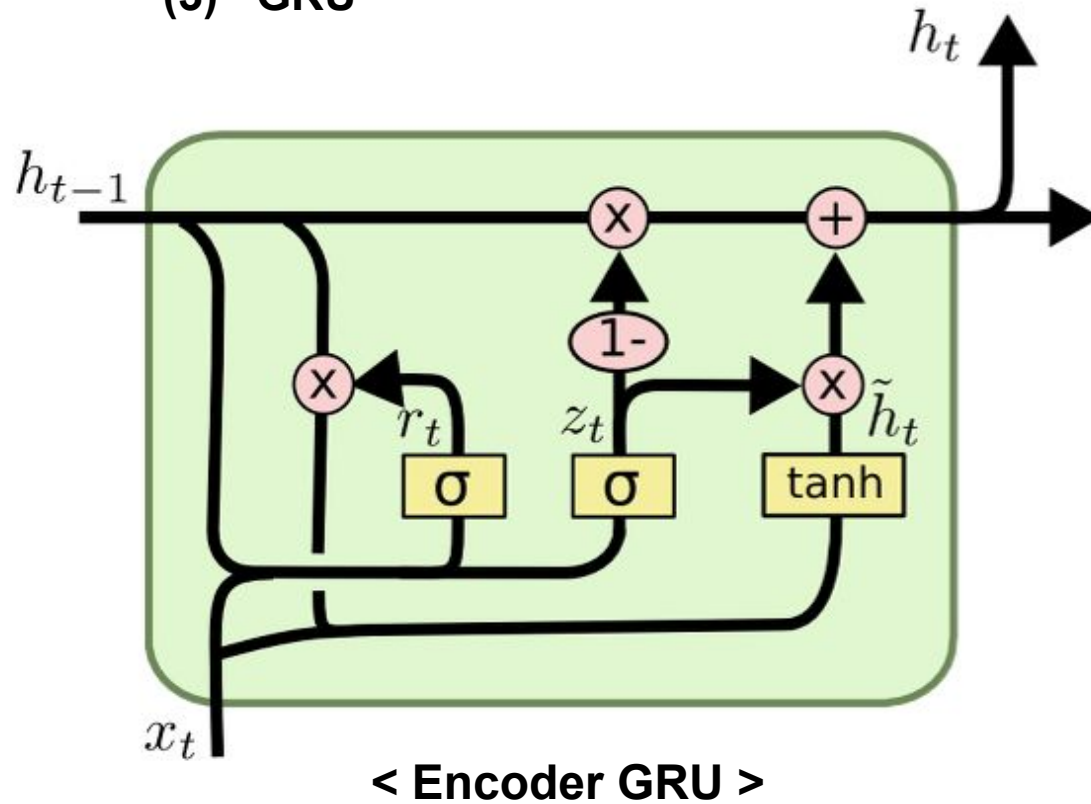


각 인코더의 어텐션 가중치와 hidden state를 가중합하여 Context vector를 구한다
Context Vector에는 decoder가 집중해야할 source word의 정보가 반영되어있다

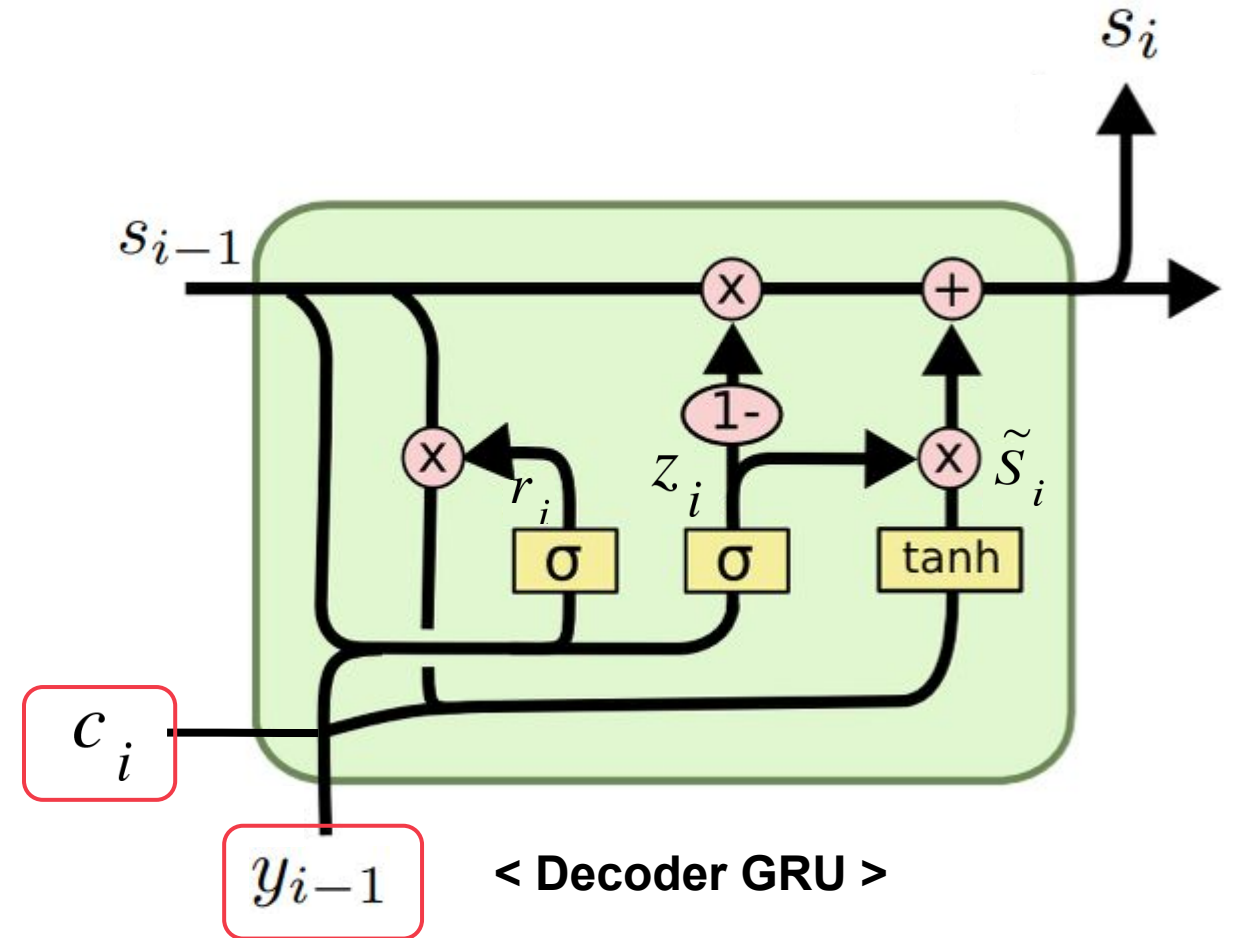
3. Alignment Model

3.4 Decoder

(3) GRU



Input Value : x_t



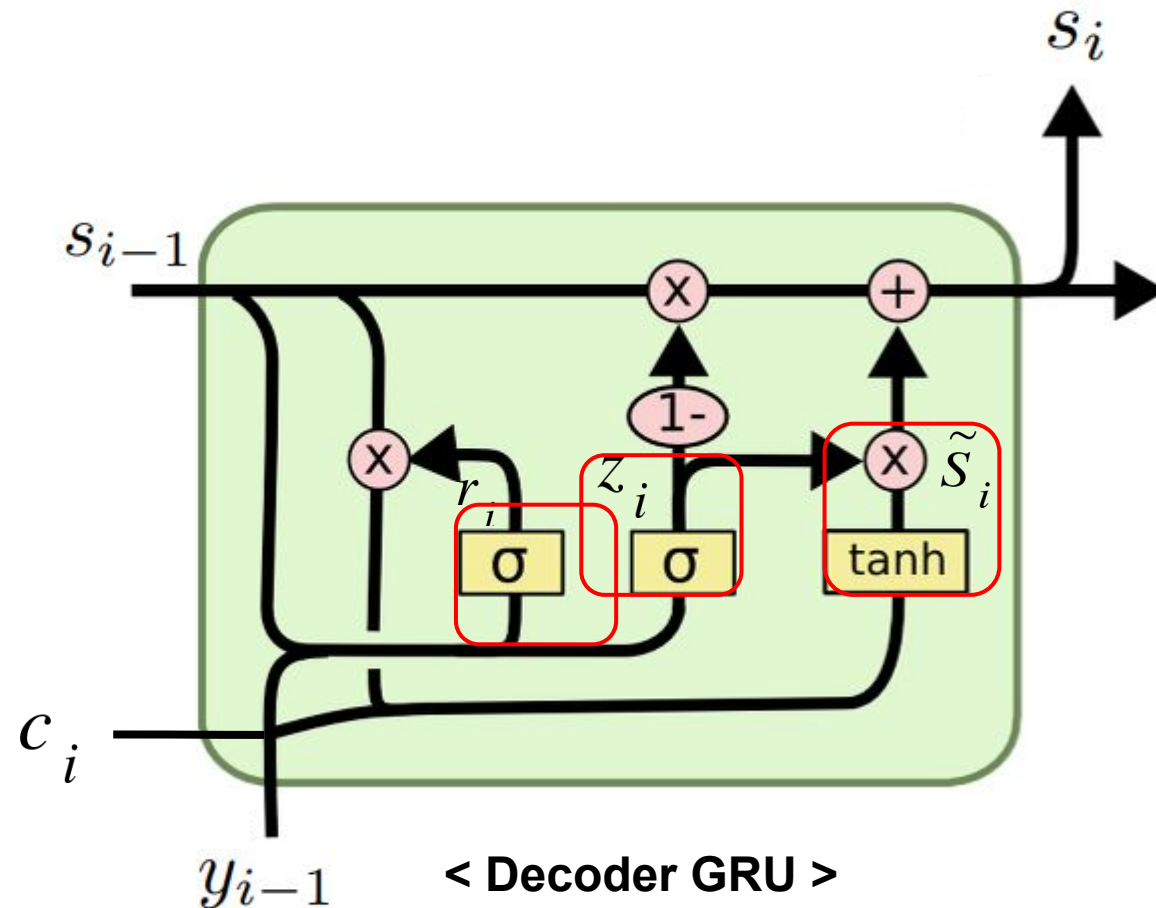
Input Value : y_{i-1}

Context Vector가 GRU Gate에 입력

3. Alignment Model

3.4 Decoder

(3) GRU



$$s_i = (1 - z_i) \circ s_{i-1} + z_i \circ \tilde{s}_i$$



$$\tilde{s}_i = \tanh(W E y_{i-1} + U[r_i \circ s_{i-1}] + C c_i)$$

$$z_i = \sigma(W_z E y_{i-1} + U_z s_{i-1} + C_z c_i)$$

$$r_i = \sigma(W_r E y_{i-1} + U_r s_{i-1} + C_r c_i)$$

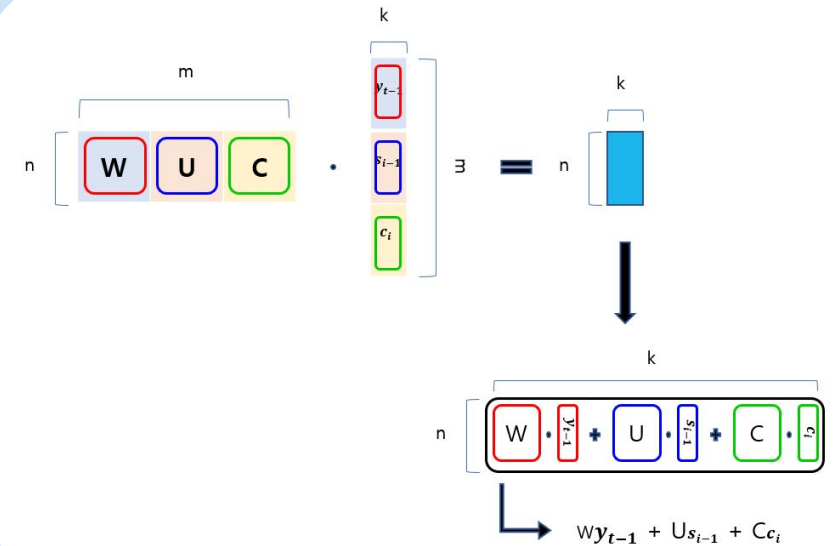
3. Alignment Model

3.4 Decoder

(3) GRU

$$W_a s_{i-j} + U_a h_j = (W_a \oplus U_a) \cdot (s_{i-1} \oplus h_j)$$

$$\begin{aligned} [(W_a \oplus U_a) \cdot (s_{i-1} \oplus h_j)]_k &= \sum_{l=1}^{d_s+d_h} (W_a \oplus U_a)_{k,l} \cdot (s_{i-1} \oplus h_j)_l = \\ &= \sum_{l=1}^{d_s} (W_a)_{k,l} \cdot (s_{i-1})_l + \sum_{l=1}^{d_h} (U_a)_{k,l} \cdot (h_j)_l = [W_a s_{i-1}]_k + [U_a h_j]_k \end{aligned}$$



Bahdanau의 공식에서 가중치가 곱해진 두 값의 합은 벡터의 **Concatenation**과 같다

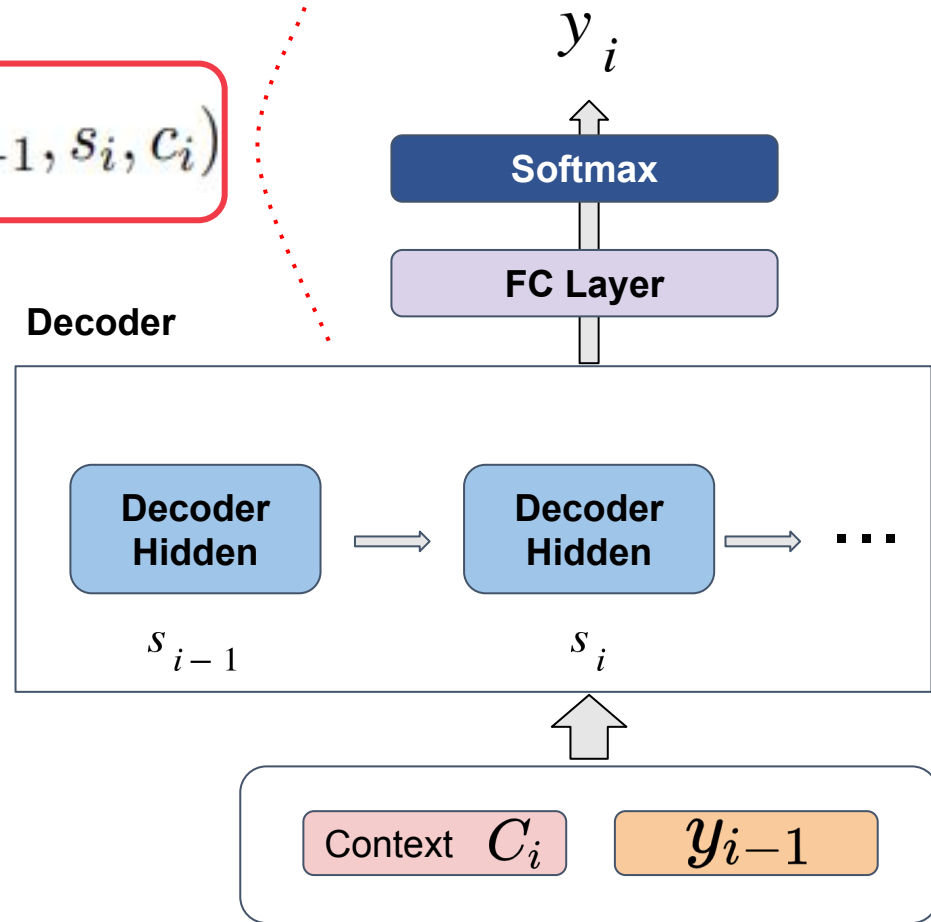
3. Alignment Model

3.4 Decoder

(4) 정리

$$p(y_i | y_1, \dots, y_{i-1}, \mathbf{X}) = g(y_{i-1}, s_i, c_i)$$

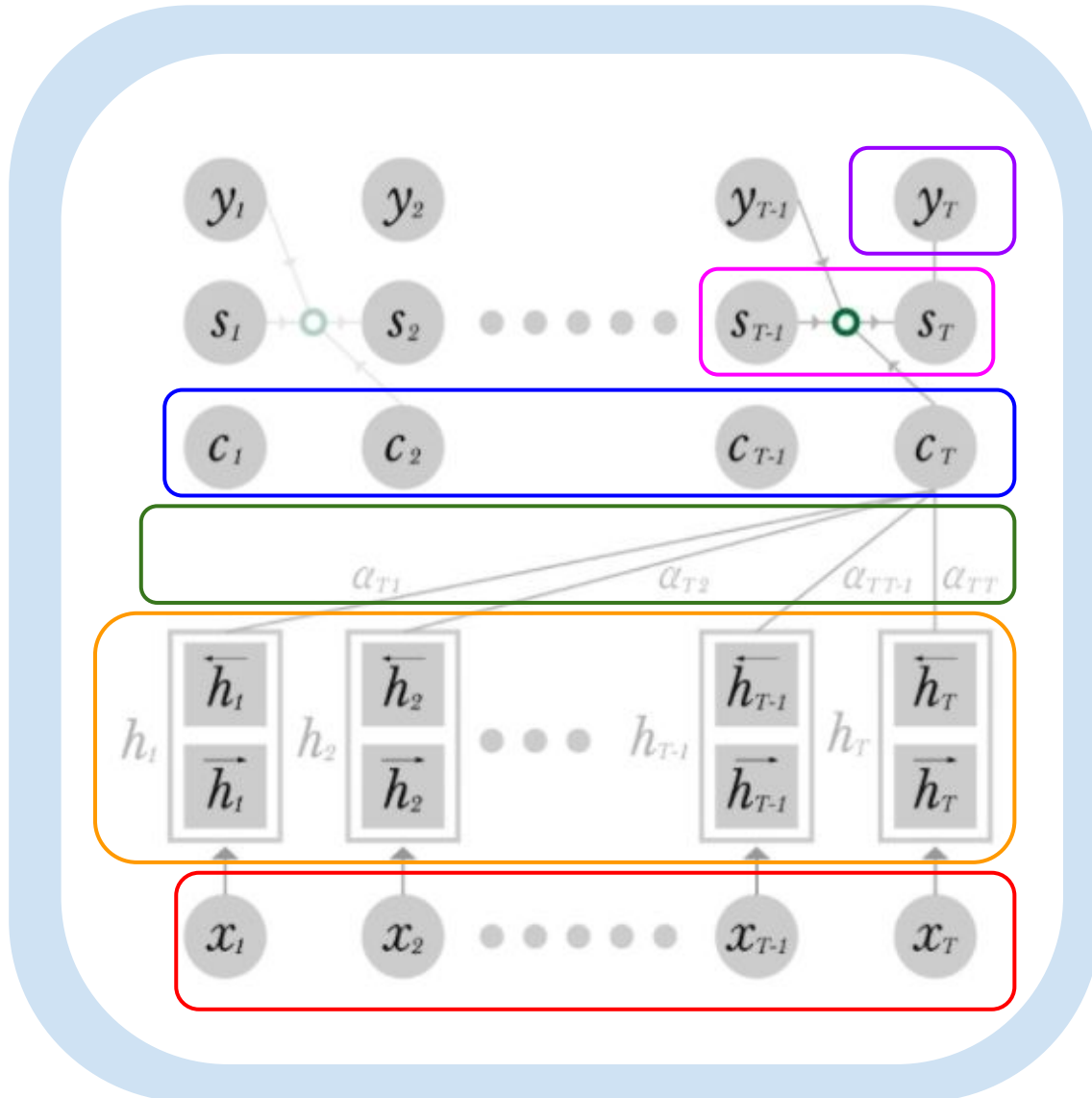
$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$



Context, input, hidden state 값을 사용하여 sentence source에 대한 attention이 반영된 출력 y 을 산출한다

3. Alignment Model

3.5 Encoder-Decoder with Attention



y_T : Output Decoder $p(y_T | y_{T-1}, \dots, y_{T-1}, X) = p(y_{T-1}, s_T, c_T)$

$$s_T = f(s_{T-1}, y_{T-1}, c_T)$$

$$c_T = \sum_{j=1}^T \alpha_{Tj} h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^T \exp(e_{ik})} \quad \left| \quad \begin{array}{l} e_{ij} = a(s_{i-1}, h_j) \\ \text{where } a: \text{alignment model (parameterized)} \end{array} \right.$$

$$h_j = [\vec{h}_j^\top; \overleftarrow{h}_j^\top]^\top \quad \text{concatenating the forward/backward hidden state}$$

x_T : Input

04 CODE IMPLEMENTATION

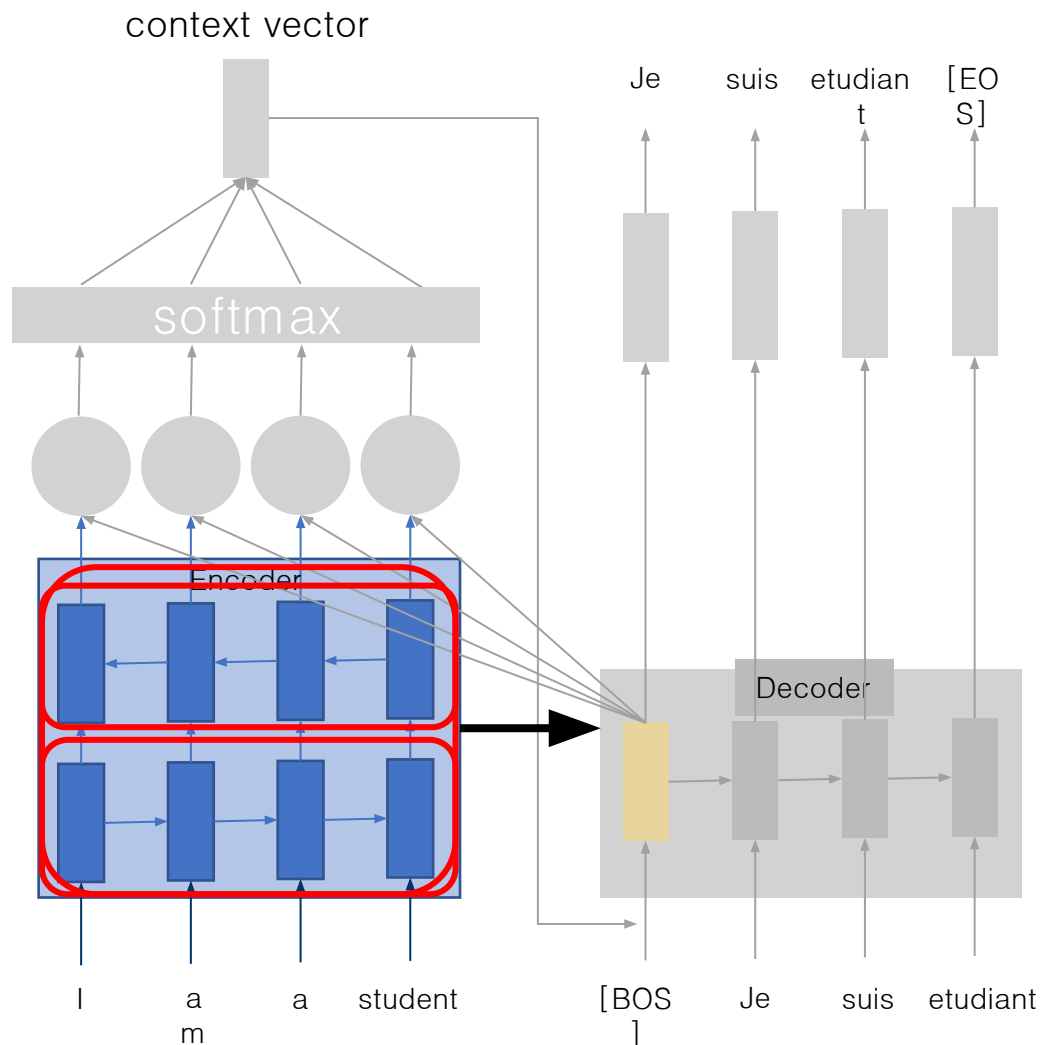
Bahdanau Attention 코드 구현

4. 코드 구현(Bahdanau Attention)

Encoder

Attention

Decoder



```
class Encoder(tf.keras.layers.Layer):
    def __init__(self, vocab_size, enc_dim=256, num_embedding=256, batch_size=32):
        super(Encoder, self).__init__()
        self.vocab_size = vocab_size
        self.batch_size = batch_size
        self.enc_dim = enc_dim
        self.num_embedding = num_embedding
        self.embedding = tf.keras.layers.Embedding(self.vocab_size, self.num_embedding)

        self.gru_fw = tf.keras.layers.GRU(enc_dim,
                                           return_sequences=True,
                                           return_state=True,
                                           recurrent_initializer='glorot_uniform')

        self.gru_bw = tf.keras.layers.GRU(enc_dim,
                                           go_backwards=True,
                                           return_sequences=True,
                                           return_state=True,
                                           recurrent_initializer='glorot_uniform')

        self.gru = tf.keras.layers.Bidirectional(self.gru_fw, backward_layer=self.gru_bw)

    def call(self, x, hidden):
        # 워드 임베딩
        # (batch, seq_length) -> (batch, seq_length, num_embedding)
        x = self.embedding(x)

        # RNN 출력
        # output.shape: (batch, seq_length, enc_dim * 2)
        # fw_hidden.shape: (batch, enc_dim)
        # bw_hidden.shape: (batch, enc_dim)
        output, fw_hidden, bw_hidden = self.gru(x, initial_state=hidden)

        hidden = tf.concat([fw_hidden, bw_hidden], axis=-1) # (bs, d_model * 2)

        return output, hidden

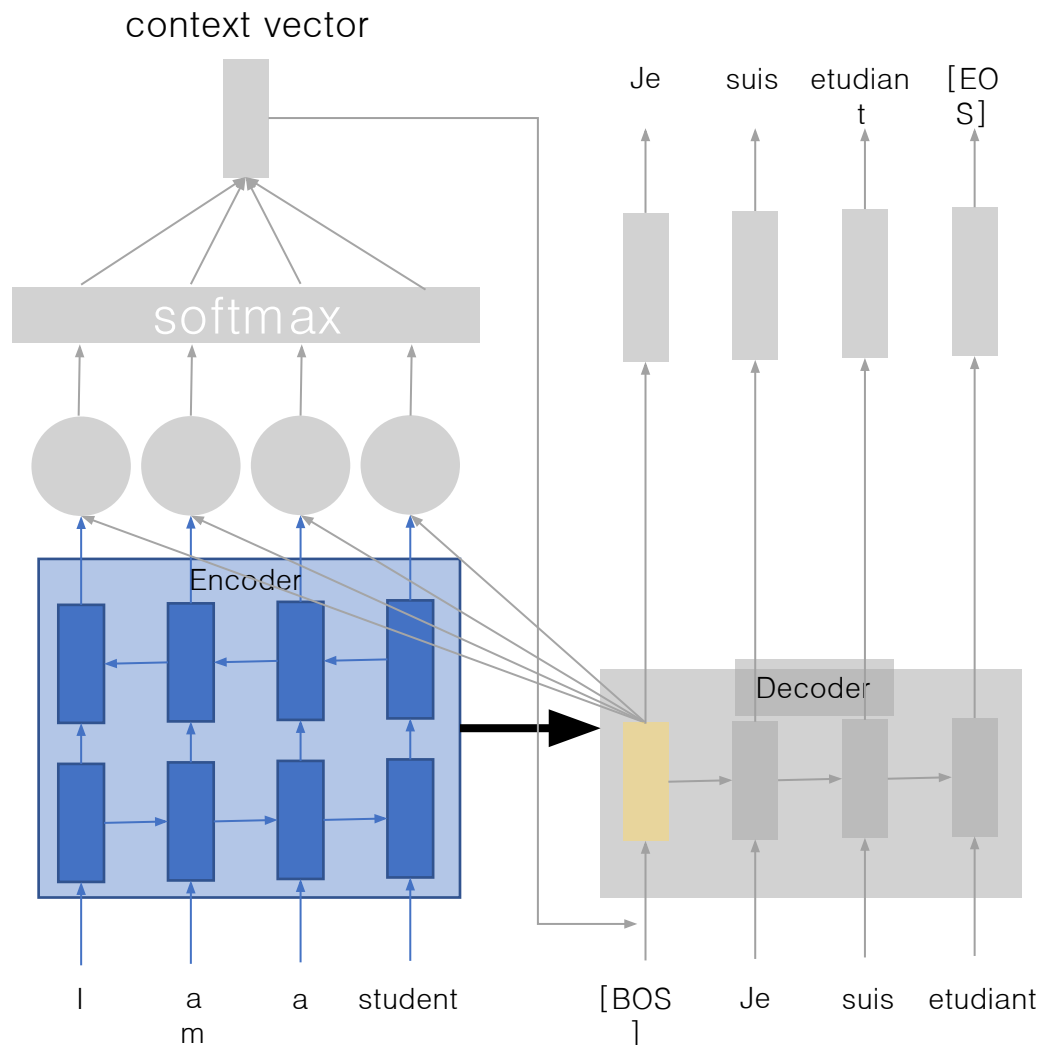
    def init_hidden(self, input_):
        return [tf.zeros((tf.shape(input_)[0], self.enc_dim)) for _ in range(2)]
```


4. 코드 구현(Bahdanau Attention)

Encoder

Attention

Decoder



```
class Encoder(tf.keras.layers.Layer):
    def __init__(self, vocab_size, enc_dim=256, num_embedding=256, batch_size=32):
        super(Encoder, self).__init__()
        self.vocab_size = vocab_size
        self.batch_size = batch_size
        self.enc_dim = enc_dim
        self.num_embedding = num_embedding
        self.embedding = tf.keras.layers.Embedding(self.vocab_size, self.num_embedding)

        self.gru_fw = tf.keras.layers.GRU(enc_dim,
                                           return_sequences=True,
                                           return_state=True,
                                           recurrent_initializer='glorot_uniform')

        self.gru_bw = tf.keras.layers.GRU(enc_dim,
                                           go_backwards=True,
                                           return_sequences=True,
                                           return_state=True,
                                           recurrent_initializer='glorot_uniform')

        self.gru = tf.keras.layers.Bidirectional(self.gru_fw, backward_layer=self.gru_bw)

    def call(self, x, hidden):
        # 워드 임베딩
        # (batch, seq_length) -> (batch, seq_length, num_embedding)
        x = self.embedding(x)

        # RNN 출력
        # output.shape: (batch, seq_length, enc_dim * 2)
        # fw_hidden.shape: (batch, enc_dim)
        # bw_hidden.shape: (batch, enc_dim)
        output, fw_hidden, bw_hidden = self.gru(x, initial_state=hidden)

        hidden = tf.concat([fw_hidden, bw_hidden], axis=-1) # (bs, d_model * 2)

        return output, hidden

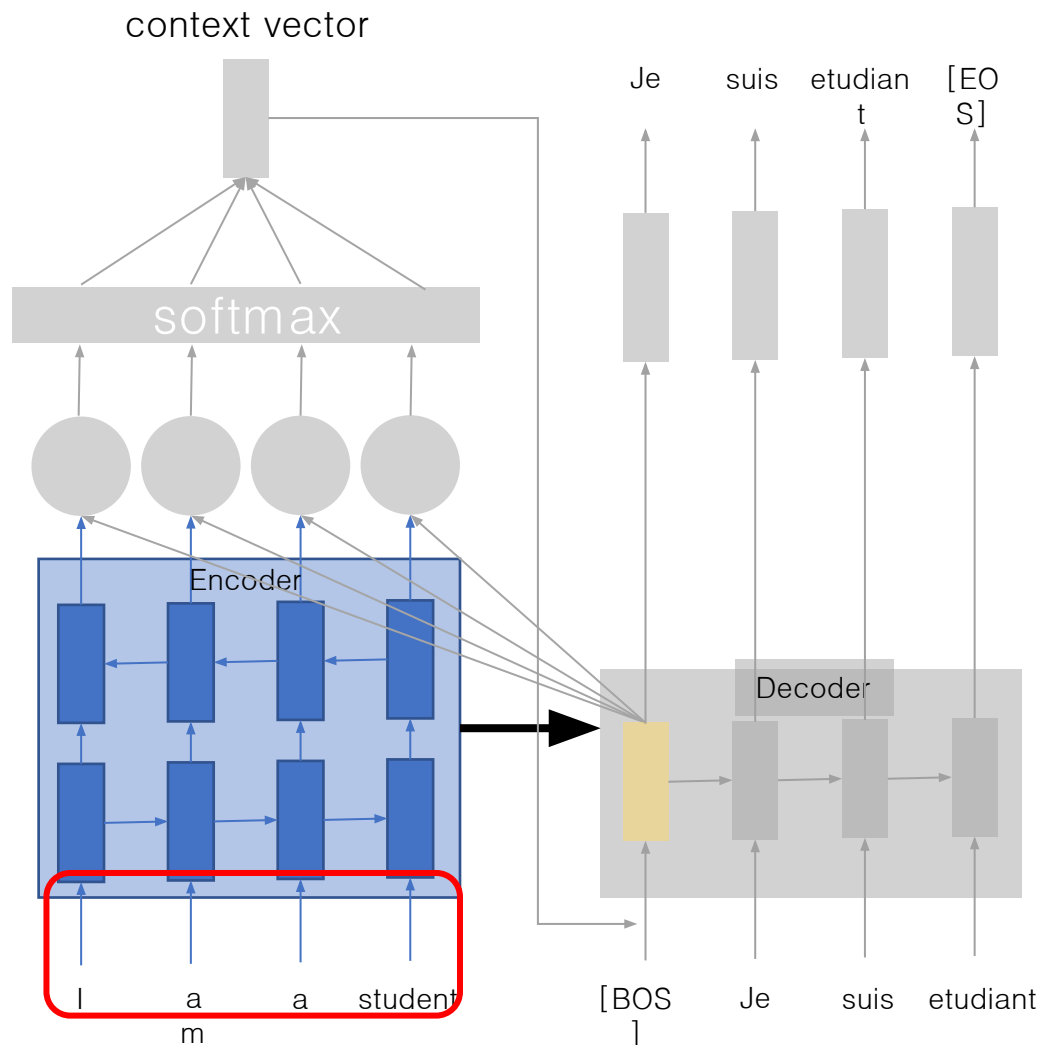
    def init_hidden(self, input_):
        return [tf.zeros((tf.shape(input_)[0], self.enc_dim)) for _ in range(2)]
```

4. 코드 구현(Bahdanau Attention)

Encoder

Attention

Decoder



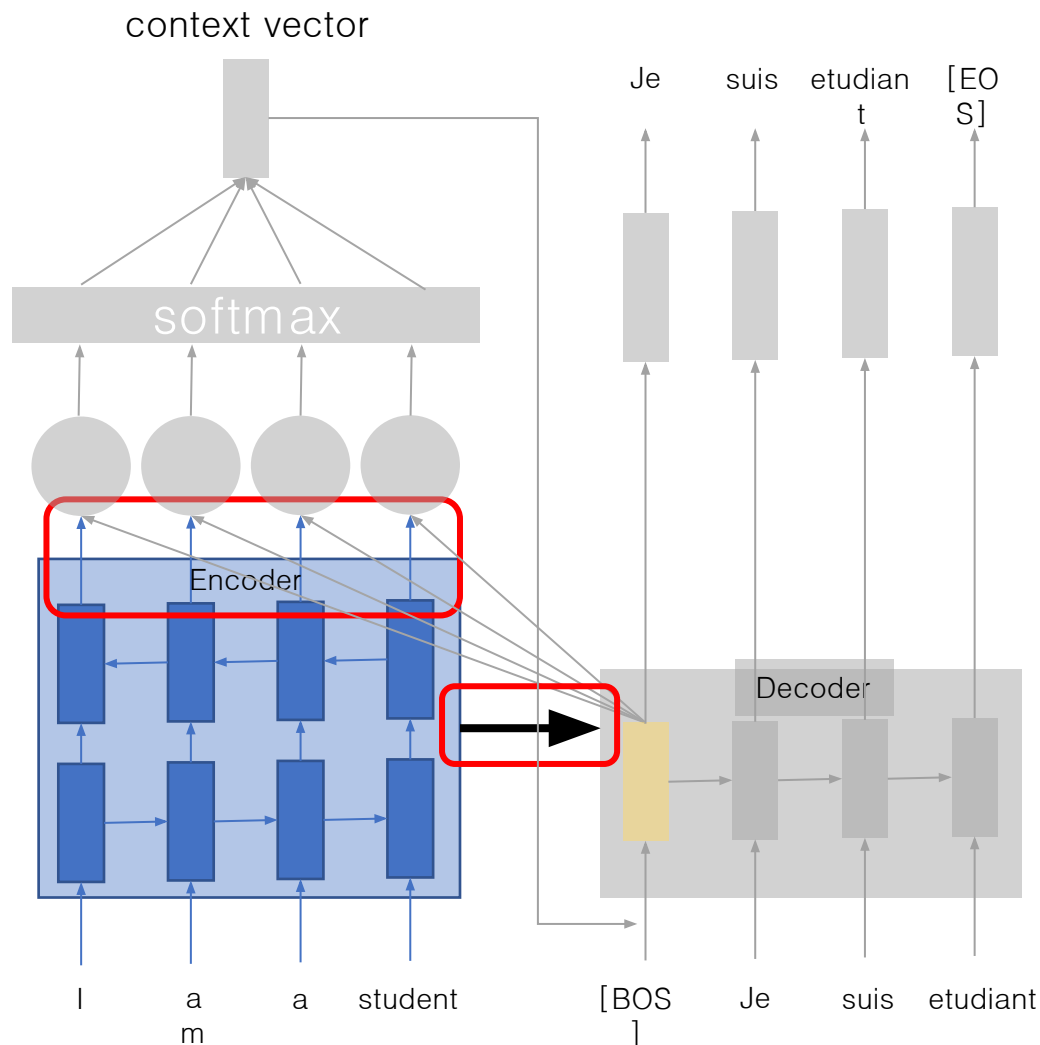
```
class Encoder(tf.keras.layers.Layer):  
    def __init__(self, vocab_size, enc_dim=256, num_embedding=256, batch_size=32):  
        super(Encoder, self).__init__()  
        self.vocab_size = vocab_size  
        self.batch_size = batch_size  
        self.enc_dim = enc_dim  
        self.num_embedding = num_embedding  
        self.embedding = tf.keras.layers.Embedding(self.vocab_size, self.num_embedding)  
  
        self.gru_fw = tf.keras.layers.GRU(enc_dim,  
                                           return_sequences=True,  
                                           return_state=True,  
                                           recurrent_initializer='glorot_uniform')  
  
        self.gru_bw = tf.keras.layers.GRU(enc_dim,  
                                           go_backwards=True,  
                                           return_sequences=True,  
                                           return_state=True,  
                                           recurrent_initializer='glorot_uniform')  
  
        self.gru = tf.keras.layers.Bidirectional(self.gru_fw, backward_layer=self.gru_bw)  
  
    def call(self, x, hidden):  
        # 워드 임베딩  
        # (batch, seq_length) -> (batch, seq_length, num_embedding)  
        x = self.embedding(x)  
  
        # RNN 출력  
        # output.shape: (batch, seq_length, enc_dim * 2)  
        # fw_hidden.shape: (batch, enc_dim)  
        # bw_hidden.shape: (batch, enc_dim)  
        output, fw_hidden, bw_hidden = self.gru(x, initial_state=hidden)  
  
        hidden = tf.concat([fw_hidden, bw_hidden], axis=-1) # (bs, d_model * 2)  
  
        return output, hidden  
  
    def init_hidden(self, input_):  
        return [tf.zeros((tf.shape(input_)[0], self.enc_dim)) for _ in range(2)]
```

4. 코드 구현(Bahdanau Attention)

Encoder

Attention

Decoder



```
class Encoder(tf.keras.layers.Layer):
    def __init__(self, vocab_size, enc_dim=256, num_embedding=256, batch_size=32):
        super(Encoder, self).__init__()
        self.vocab_size = vocab_size
        self.batch_size = batch_size
        self.enc_dim = enc_dim
        self.num_embedding = num_embedding
        self.embedding = tf.keras.layers.Embedding(self.vocab_size, self.num_embedding)

        self.gru_fw = tf.keras.layers.GRU(enc_dim,
                                           return_sequences=True,
                                           return_state=True,
                                           recurrent_initializer='glorot_uniform')

        self.gru_bw = tf.keras.layers.GRU(enc_dim,
                                           go_backwards=True,
                                           return_sequences=True,
                                           return_state=True,
                                           recurrent_initializer='glorot_uniform')

        self.gru = tf.keras.layers.Bidirectional(self.gru_fw, backward_layer=self.gru_bw)

    def call(self, x, hidden):
        # 워드 임베딩
        # (batch, seq_length) -> (batch, seq_length, num_embedding)
        x = self.embedding(x)

        # RNN 출력
        # output.shape: (batch, seq_length, enc_dim * 2)
        # fw_hidden.shape: (batch, enc_dim)
        # bw_hidden.shape: (batch, enc_dim)
        output, fw_hidden, bw_hidden = self.gru(x, initial_state=hidden)

        hidden = tf.concat([fw_hidden, bw_hidden], axis=-1) # (bs, d_model * 2)

        return output, hidden

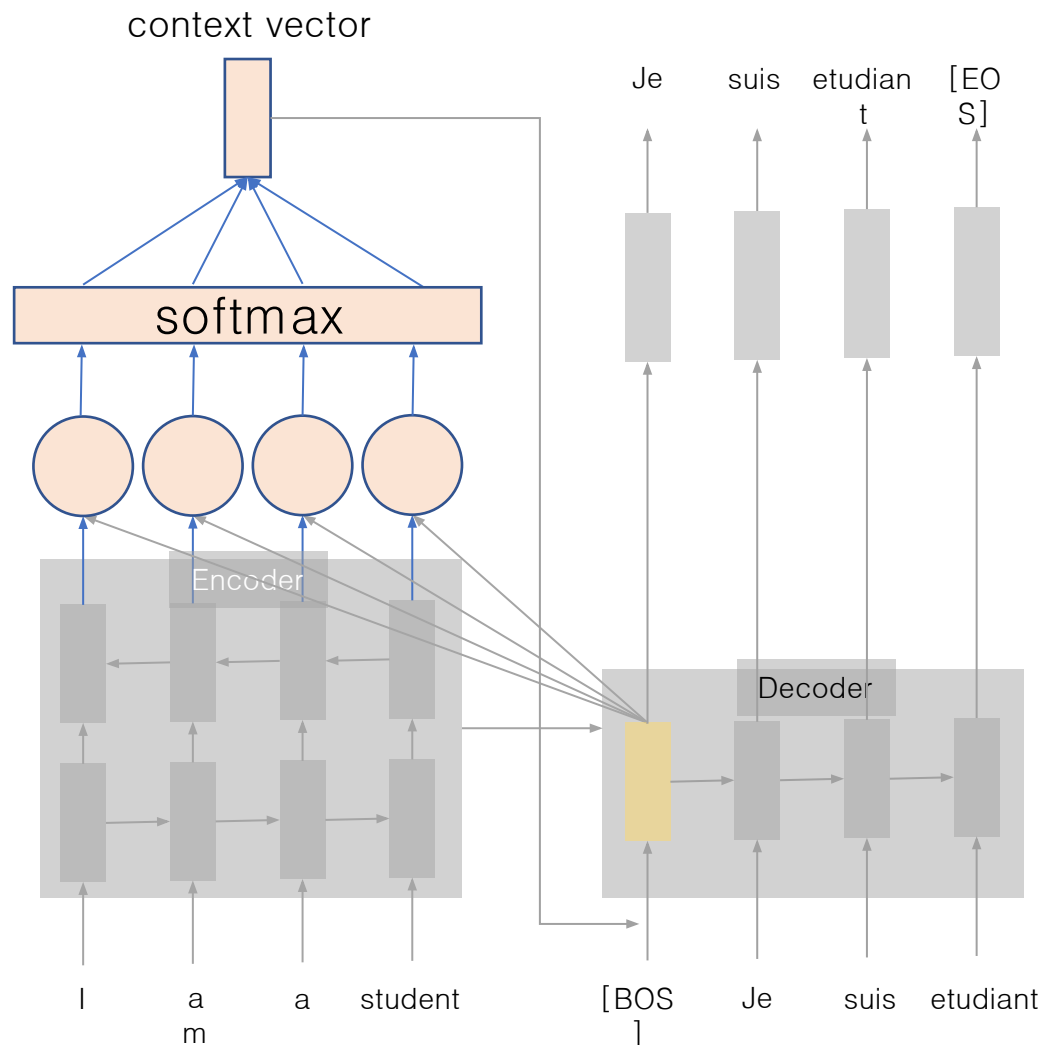
    def init_hidden(self, input_):
        return [tf.zeros((tf.shape(input_)[0], self.enc_dim)) for _ in range(2)]
```

4. 코드 구현(Bahdanau Attention)

Encoder

Attention

Decoder



```
class BahdanauAttention(tf.keras.models.Model):
    def __init__(self, units):
        super(BahdanauAttention, self).__init__()

        self.W1 = tf.keras.layers.Dense(units)
        self.W2 = tf.keras.layers.Dense(units)
        self.V = tf.keras.layers.Dense(1)

    def call(self, encoder_out, hidden):
        # output.shape: (batch, seq_length, enc_dim)
        # hidden.shape: (batch, enc_dim)

        # hidden에 시계열 축 추가
        hidden = tf.expand_dims(hidden, axis=1) #out: (16, 1, 1024)

        # Bahdanau attention score 계산
        # (batch, enc_dim) -> (batch, 1, enc_dim)
        score = self.V(tf.nn.tanh(self.W1(encoder_out) + \
                                     self.W2(hidden))) #out:

        # softmax를 통해 attention weights 계산
        attn_weights = tf.nn.softmax(score, axis=1)

        # context vector 계산
        # ((batch, 1, enc_dim) * (batch, seq_length, enc_dim)) -> (batch, seq_length, enc_dim)
        context_vector = attn_weights * encoder_out

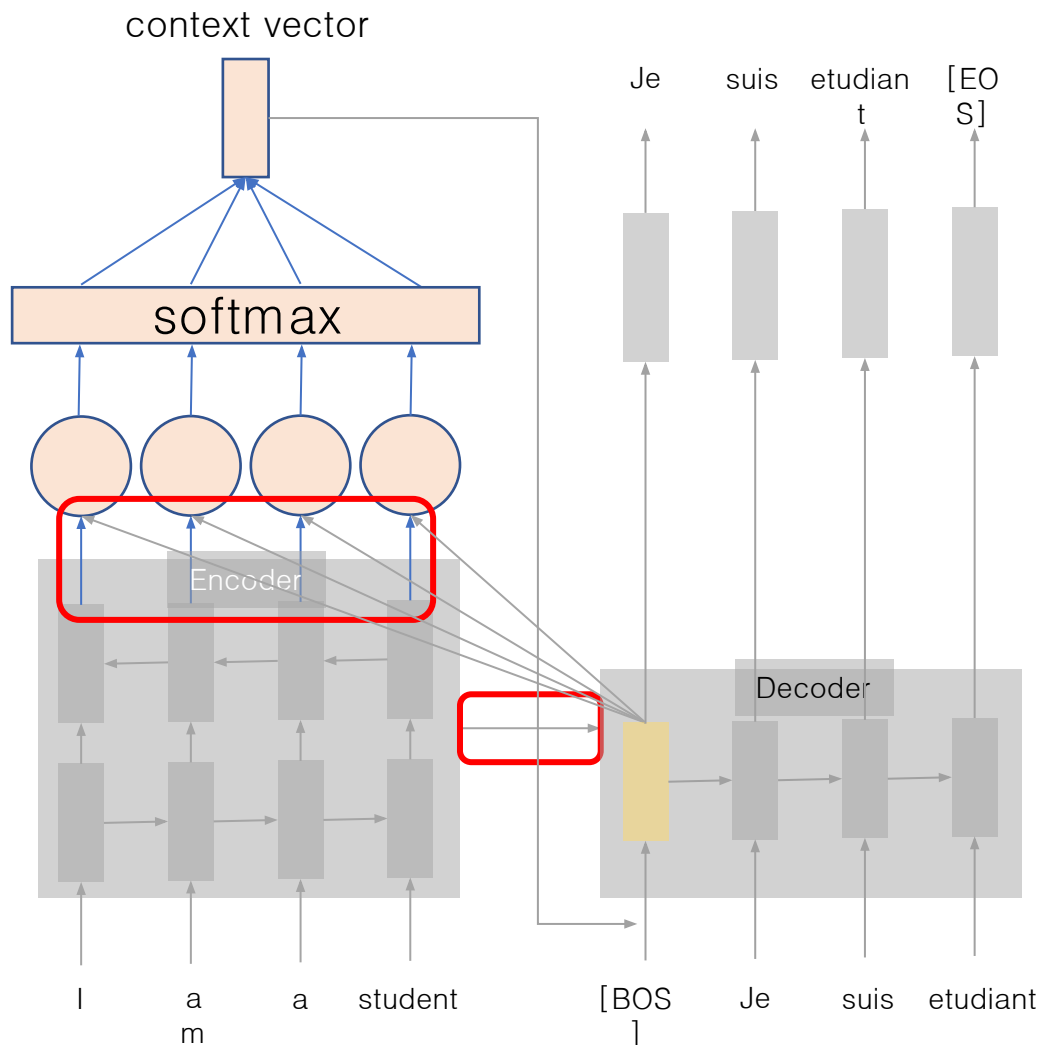
        # (batch, seq_length, enc_dim) -> (batch, enc_dim)
        context_vector = tf.reduce_sum(context_vector, axis=1)
        return context_vector, attn_weights
```


4. 코드 구현(Bahdanau Attention)

Encoder

Attention

Decoder



```
class BahdanauAttention(tf.keras.models.Model):
    def __init__(self, units):
        super(BahdanauAttention, self).__init__()

        self.W1 = tf.keras.layers.Dense(units)
        self.W2 = tf.keras.layers.Dense(units)
        self.V = tf.keras.layers.Dense(1)

    def call(self, encoder_out, hidden):
        # output.shape: (batch, seq_length, enc_dim)
        # hidden.shape: (batch, enc_dim)

        # hidden에 시계열 축 추가
        hidden = tf.expand_dims(hidden, axis=1) #out: (16, 1, 1024)

        # Bahdanau attention score 계산
        # (batch, enc_dim) -> (batch, 1, enc_dim)
        score = self.V(tf.nn.tanh(self.W1(encoder_out) + \
                                     self.W2(hidden))) #out:

        # softmax를 통해 attention weights 계산
        attn_weights = tf.nn.softmax(score, axis=1)

        # context vector 계산
        # ((batch, 1, enc_dim) * (batch, seq_length, enc_dim)) -> (batch, seq_length, enc_dim)
        context_vector = attn_weights * encoder_out

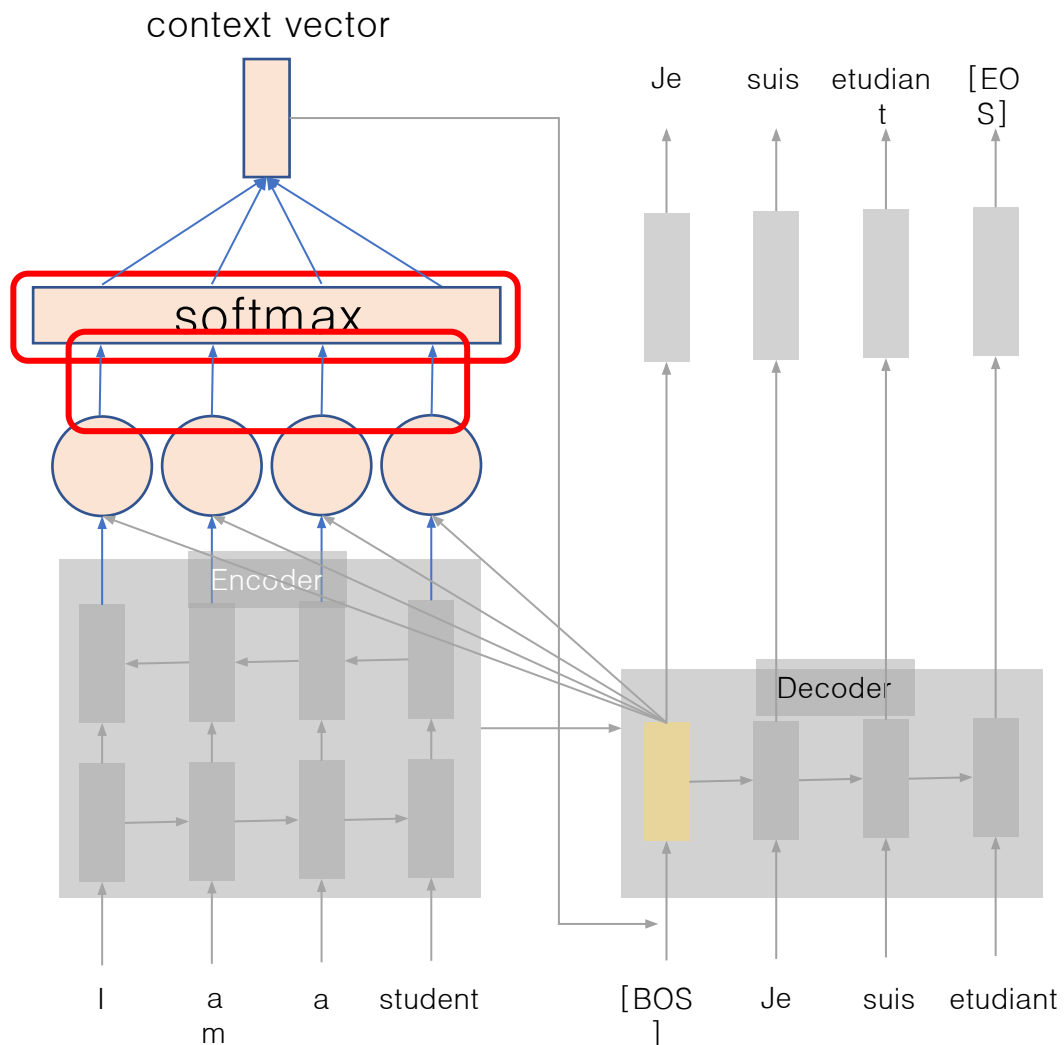
        # (batch, seq_length, enc_dim) -> (batch, enc_dim)
        context_vector = tf.reduce_sum(context_vector, axis=1)
        return context_vector, attn_weights
```

4. 코드 구현(Bahdanau Attention)

Encoder

Attention

Decoder



```
class BahdanauAttention(tf.keras.models.Model):  
    def __init__(self, units):  
        super(BahdanauAttention, self).__init__()  
  
        self.W1 = tf.keras.layers.Dense(units)  
        self.W2 = tf.keras.layers.Dense(units)  
        self.V = tf.keras.layers.Dense(1)
```

```
    def call(self, encoder_out, hidden):  
        # output.shape: (batch, seq_length, enc_dim)  
        # hidden.shape: (batch, enc_dim)
```

$$a(s_{i-1}, h_j) = v_a^T \tanh(W_a s_{i-1} + U_a h_j)$$

```
        score = self.V(tf.nn.tanh(self.W1(encoder_out) +  
                                   self.W2(hidden))) #out:
```

```
        # softmax을 통해 attention weights 계산
```

```
        attn_weights = tf.nn.softmax(score, axis=1)
```

```
        # context vector 계산
```

```
        # ((batch, 1, enc_dim) * (batch, seq_length, enc_dim)) -> (batch, seq_length, enc_dim)  
        context_vector = attn_weights * encoder_out
```

```
        # (batch, seq_length, enc_dim) -> (batch, enc_dim)
```

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

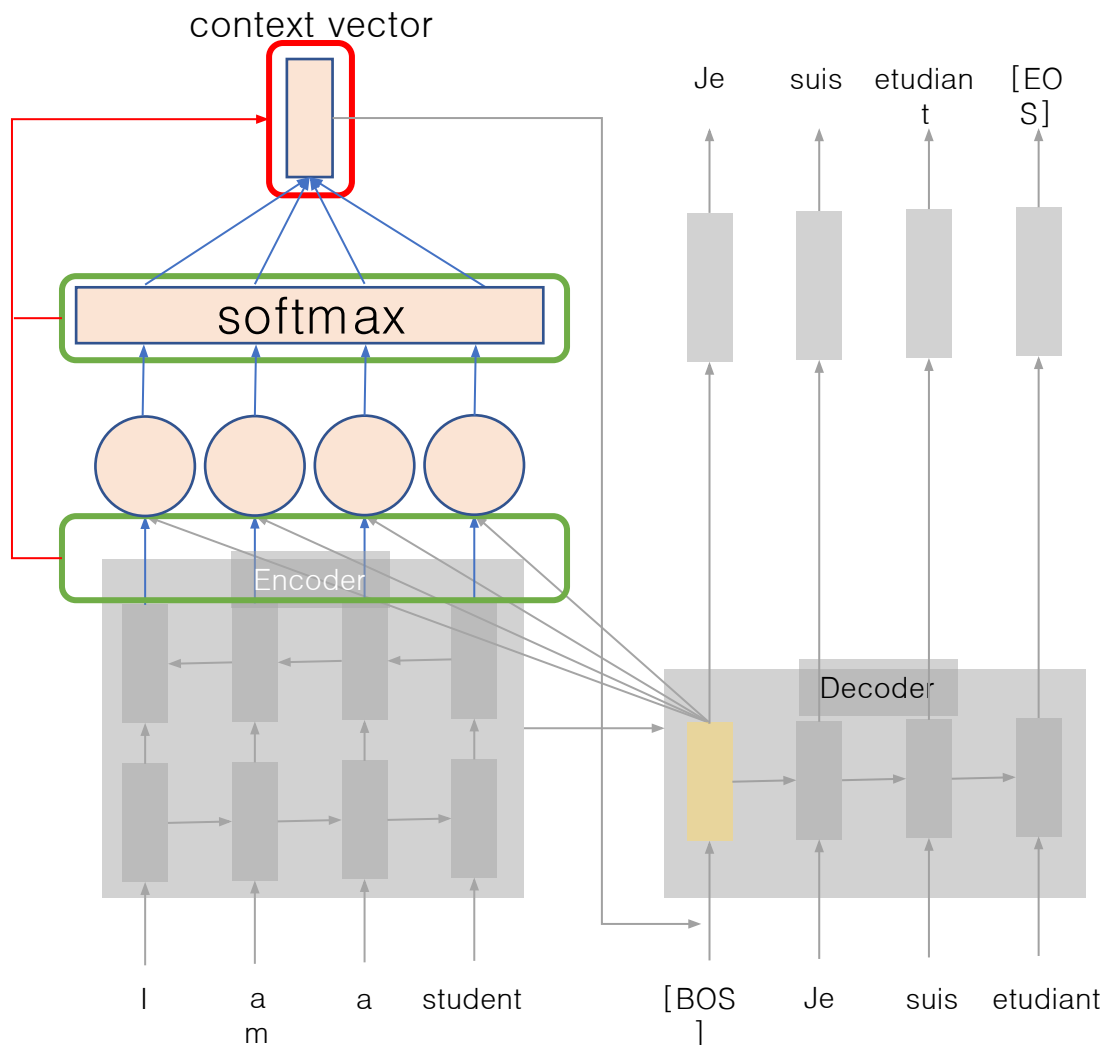
```
        return context_vector, attn_weights
```

4. 코드 구현(Bahdanau Attention)

Encoder

Attention

Decoder



```
class BahdanauAttention(tf.keras.models.Model):
    def __init__(self, units):
        super(BahdanauAttention, self).__init__()

        self.W1 = tf.keras.layers.Dense(units)
        self.W2 = tf.keras.layers.Dense(units)
        self.V = tf.keras.layers.Dense(1)

    def call(self, encoder_out, hidden):
        # output.shape: (batch, seq_length, enc_dim)
        # hidden.shape: (batch, enc_dim)

        # hidden에 시계열 축 추가
        hidden = tf.expand_dims(hidden, axis=1) #out: (16, 1, 1024)

        # Bahdanau attention score 계산
        # (batch, enc_dim) -> (batch, 1, enc_dim)
        score = self.V(tf.nn.tanh(self.W1(encoder_out) + \
                                     self.W2(hidden))) #out:

        # softmax를 통해 attention weights 계산
        attn_weights = tf.nn.softmax(score, axis=1)

        # context vector 계산
        # ((batch, 1, enc_dim) * (batch, seq_length, enc_dim)) -> (batch, seq_length, enc_dim)
        context_vector = attn_weights * encoder_out

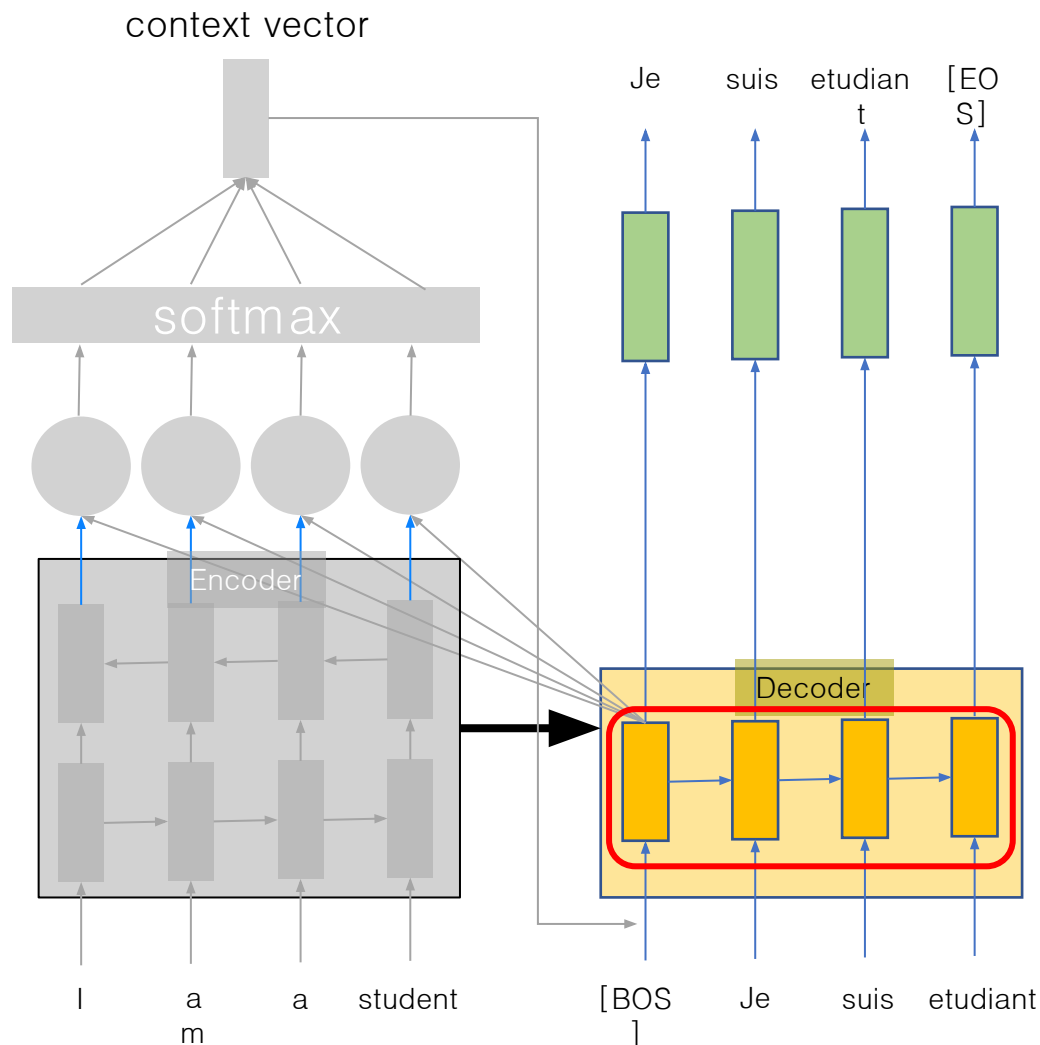
        # (batch, seq_length, enc_dim) -> (batch, enc_dim)
        context_vector = tf.reduce_sum(context_vector, axis=1)
        return context_vector, attn_weights
```

4. 코드 구현(Bahdanau Attention)

Encoder

Attention

Decoder



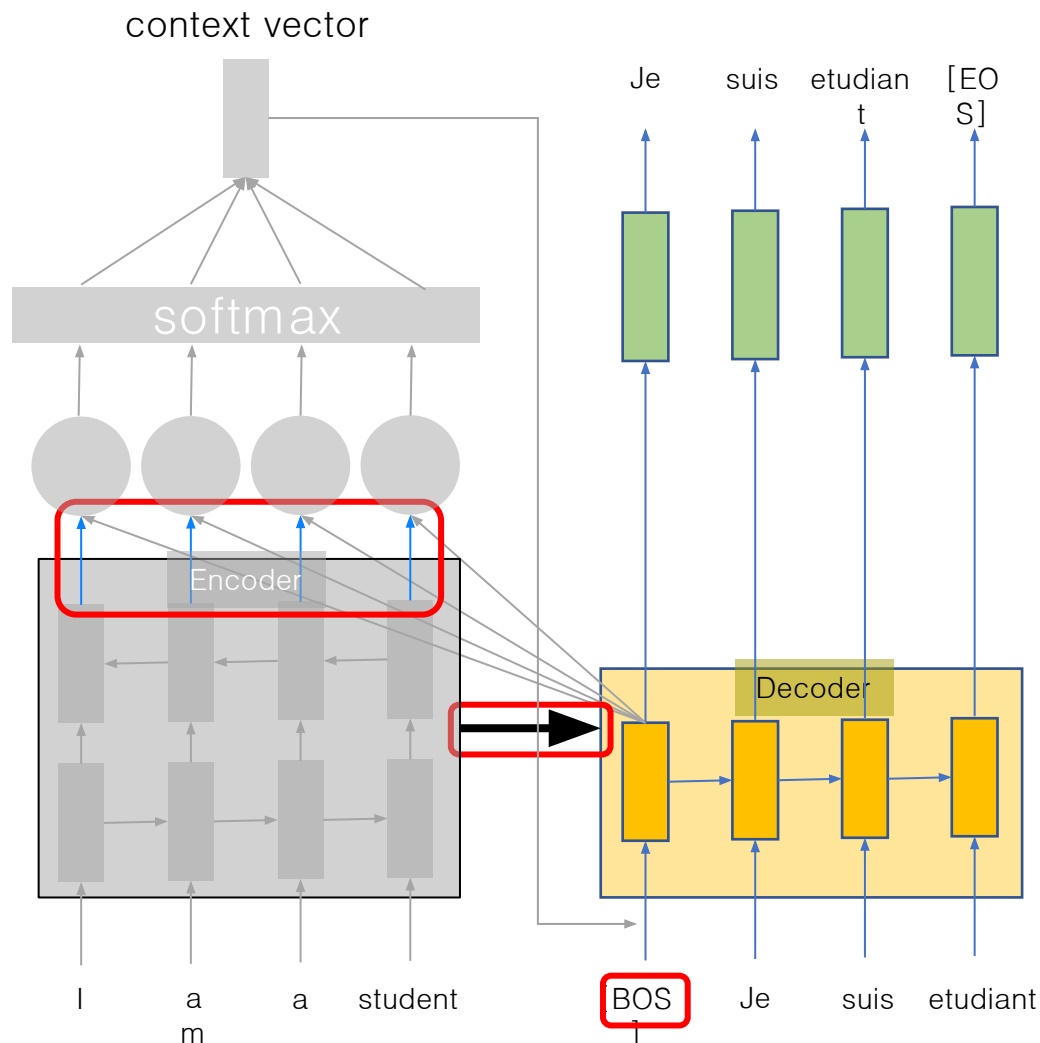
```
class Decoder(tf.keras.models.Model):  
    def __init__(self, vocab_size, embedding_dim=256, dec_dim=256, batch_size=32):  
        super(Decoder, self).__init__()  
        self.vocab_size = vocab_size  
        self.dec_dim = dec_dim  
        self.embedding_dim = embedding_dim  
        self.batch_size = batch_size  
        self.attn = BahdanauAttention(self.dec_dim)  
        self.embedding = tf.keras.layers.Embedding(self.vocab_size, self.embedding_dim)  
        self.gru = tf.keras.layers.GRU(self.dec_dim,  
                                       recurrent_initializer='glorot_uniform',  
                                       return_sequences=True,  
                                       return_state=True)  
        self.fc = tf.keras.layers.Dense(self.vocab_size)  
  
    def call(self, x, hidden, enc_out):  
        # x.shape = (None, 1)  
        # enc_out.shape = (None, seq_length, enc_dim)  
        # enc_hidden.shape = (None, enc_dim)  
  
        # decoder input의 워드 임베딩  
        # (None, 1) -> (None, 1, embedding_dim)  
        x = self.embedding(x)  
  
        # attention 가중치 계산  
        # context.shape = (None, enc_dim)  
        # attn_weights.shape = (None, seq_length, enc_dim)  
        context, attn_weights = self.attn(enc_out, hidden)  
  
        # x.shape = (None, 1, enc_dim + embedding_dim)  
        x = tf.concat((tf.expand_dims(context, 1), x), -1)  
  
        # Decoder RNN sequence 출력  
        # r_out.shape = (None, 1, dec_dim)  
        # r_out.shape = (None, dec_dim)  
        r_out, hidden = self.gru(x, initial_state=hidden)  
  
        # 시계열 축 제거  
        # (None, 1, dec_dim) -> (None, dec_dim)  
        out = tf.reshape(r_out, shape=(-1, r_out.shape[2]))  
  
        return self.fc(out), hidden, attn_weights
```


4. 코드 구현(Bahdanau Attention)

Encoder

Attention

Decoder



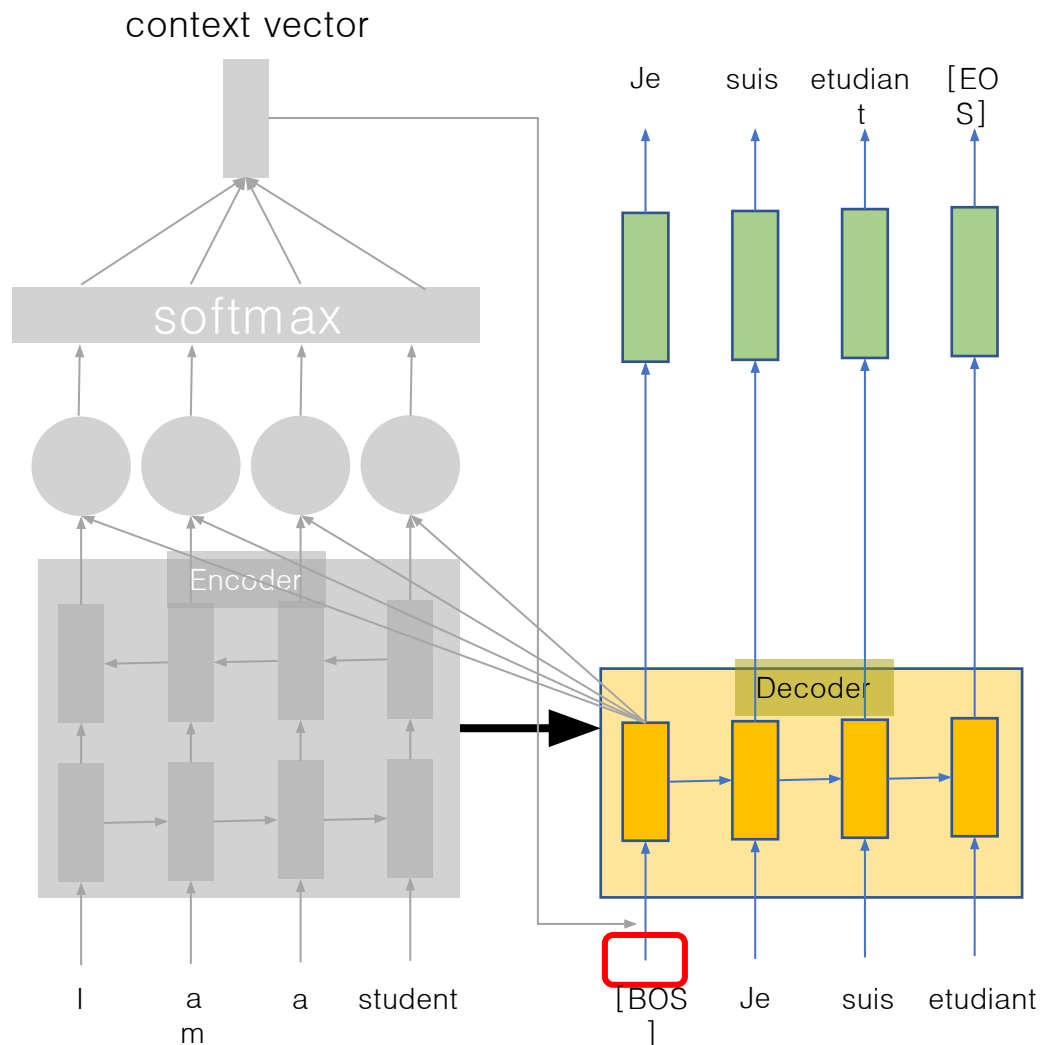
```
class Decoder(tf.keras.models.Model):  
    def __init__(self, vocab_size, embedding_dim=256, dec_dim=256, batch_size=32):  
        super(Decoder, self).__init__()  
        self.vocab_size = vocab_size  
        self.dec_dim = dec_dim  
        self.embedding_dim = embedding_dim  
        self.batch_size = batch_size  
        self.attn = BahdanauAttention(self.dec_dim)  
        self.embedding = tf.keras.layers.Embedding(self.vocab_size, self.embedding_dim)  
        self.gru = tf.keras.layers.GRU(self.dec_dim,  
                                         recurrent_initializer='glorot_uniform',  
                                         return_sequences=True,  
                                         return_state=True)  
        self.fc = tf.keras.layers.Dense(self.vocab_size)  
  
    def call(self, x, hidden, enc_out):  
        # x.shape = (None, 1)  
        # enc_out.shape = (None, seq_length, enc_dim)  
        # enc_hidden.shape = (None, enc_dim)  
  
        # decoder input의 워드 임베딩  
        # (None, 1) -> (None, 1, embedding_dim)  
        x = self.embedding(x)  
  
        # attention 가중치 계산  
        # context.shape = (None, enc_dim)  
        # attn_weights.shape = (None, seq_length, enc_dim)  
        context, attn_weights = self.attn(enc_out, hidden)  
  
        # x.shape = (None, 1, enc_dim + embedding_dim)  
        x = tf.concat((tf.expand_dims(context, 1), x), -1)  
  
        # Decoder RNN sequence 출력  
        # r_out.shape = (None, 1, dec_dim)  
        # r_out.shape = (None, dec_dim)  
        r_out, hidden = self.gru(x, initial_state=hidden)  
  
        # 시계열 축 제거  
        # (None, 1, dec_dim) -> (None, dec_dim)  
        out = tf.reshape(r_out, shape=(-1, r_out.shape[2]))  
  
        return self.fc(out), hidden, attn_weights
```

4. 코드 구현(Bahdanau Attention)

Encoder

Attention

Decoder



```
class Decoder(tf.keras.models.Model):
    def __init__(self, vocab_size, embedding_dim=256, dec_dim=256, batch_size=32):
        super(Decoder, self).__init__()
        self.vocab_size = vocab_size
        self.dec_dim = dec_dim
        self.embedding_dim = embedding_dim
        self.batch_size = batch_size
        self.attn = BahdanauAttention(self.dec_dim)
        self.embedding = tf.keras.layers.Embedding(self.vocab_size, self.embedding_dim)
        self.gru = tf.keras.layers.GRU(self.dec_dim,
                                       recurrent_initializer='glorot_uniform',
                                       return_sequences=True,
                                       return_state=True)
        self.fc = tf.keras.layers.Dense(self.vocab_size)

    def call(self, x, hidden, enc_out):
        # x.shape = (None, 1)
        # enc_out.shape = (None, seq_length, enc_dim)
        # enc_hidden.shape = (None, enc_dim)

        # decoder input의 워드 임베딩
        # (None, 1) -> (None, 1, embedding_dim)
        x = self.embedding(x)

        # attention 가중치 계산
        # context.shape = (None, enc_dim)
        # attn_weights.shape = (None, seq_length, enc_dim)
        context, attn_weights = self.attn(enc_out, hidden)

        # x.shape = (None, 1, enc_dim + embedding_dim)
        x = tf.concat((tf.expand_dims(context, 1), x), -1)

        # Decoder RNN sequence 출력
        # r_out.shape = (None, 1, dec_dim)
        # r_out.shape = (None, dec_dim)
        r_out, hidden = self.gru(x, initial_state=hidden)

        # 시계열 축 제거
        # (None, 1, dec_dim) -> (None, dec_dim)
        out = tf.reshape(r_out, shape=(-1, r_out.shape[2]))

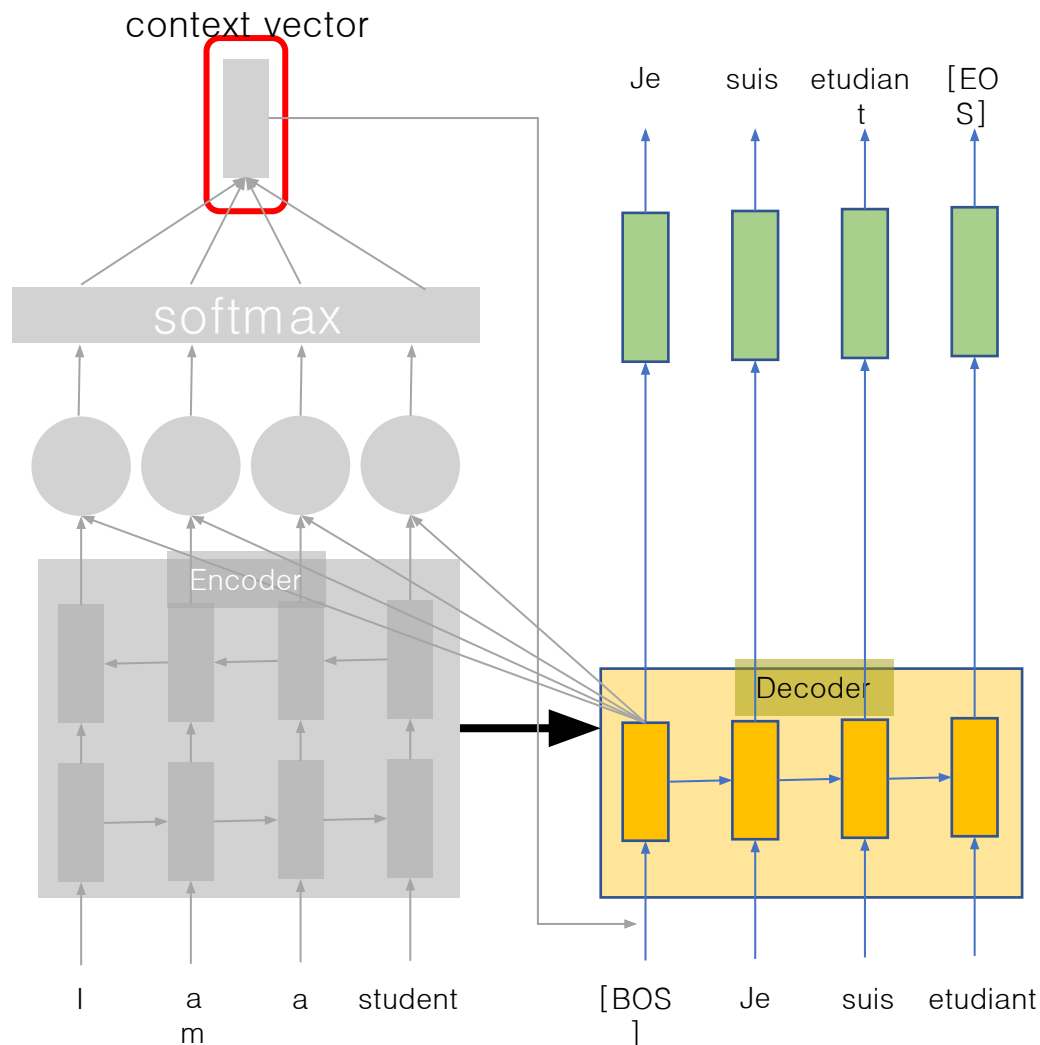
        return self.fc(out), hidden, attn_weights
```

4. 코드 구현(Bahdanau Attention)

Encoder

Attention

Decoder



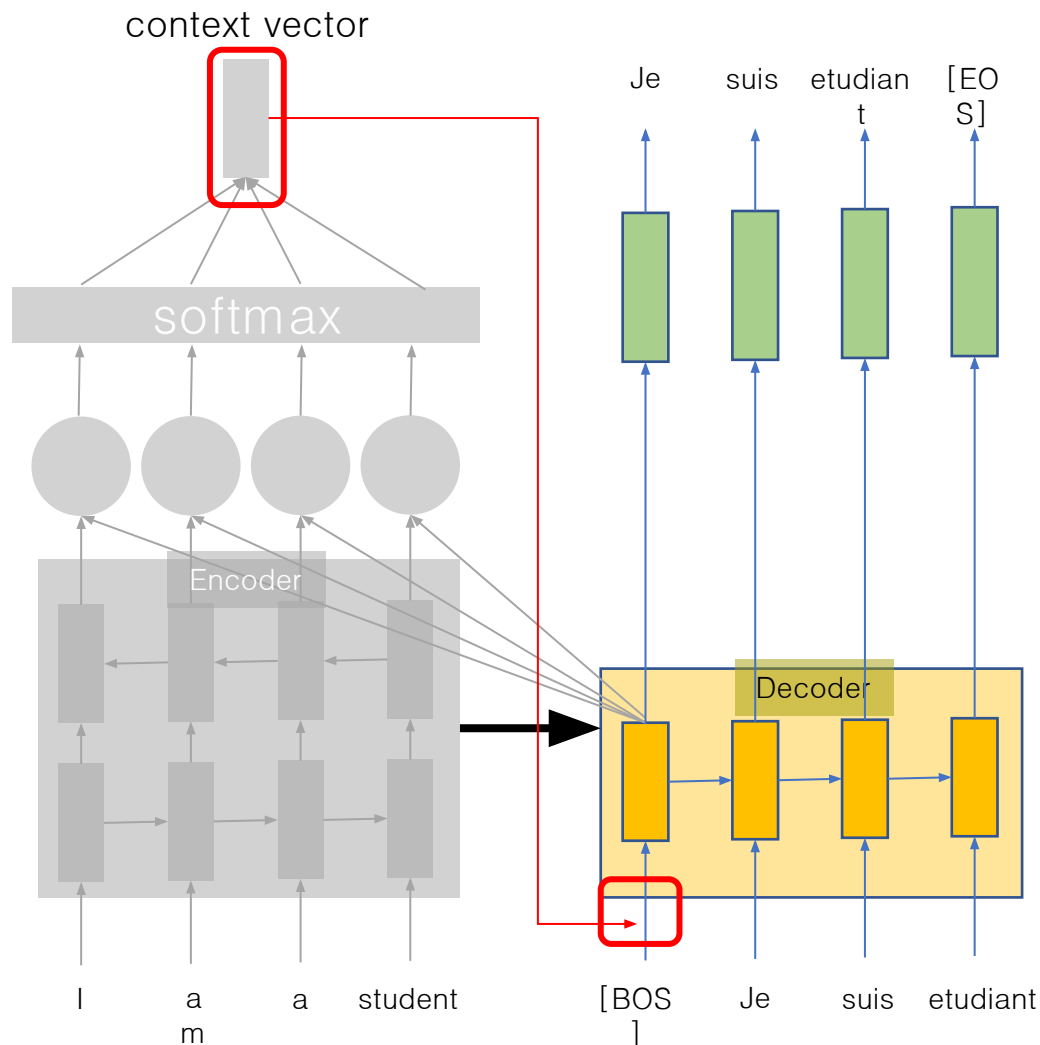
```
class Decoder(tf.keras.models.Model):  
    def __init__(self, vocab_size, embedding_dim=256, dec_dim=256, batch_size=32):  
        super(Decoder, self).__init__()  
        self.vocab_size = vocab_size  
        self.dec_dim = dec_dim  
        self.embedding_dim = embedding_dim  
        self.batch_size = batch_size  
        self.attn = BahdanauAttention(self.dec_dim)  
        self.embedding = tf.keras.layers.Embedding(self.vocab_size, self.embedding_dim)  
        self.gru = tf.keras.layers.GRU(self.dec_dim,  
                                         recurrent_initializer='glorot_uniform',  
                                         return_sequences=True,  
                                         return_state=True)  
        self.fc = tf.keras.layers.Dense(self.vocab_size)  
  
    def call(self, x, hidden, enc_out):  
        # x.shape = (None, 1)  
        # enc_out.shape = (None, seq_length, enc_dim)  
        # enc_hidden.shape = (None, enc_dim)  
  
        # decoder input의 워드 임베딩  
        # (None, 1) -> (None, 1, embedding_dim)  
        x = self.embedding(x)  
  
        # attention 가중치 계산  
        # context.shape = (None, enc_dim)  
        # attn_weights.shape = (None, seq_length, enc_dim)  
        context, attn_weights = self.attn(enc_out, hidden)  
  
        # x.shape = (None, 1, enc_dim + embedding_dim)  
        x = tf.concat((tf.expand_dims(context, 1), x), -1)  
  
        # Decoder RNN sequence 출력  
        # r_out.shape = (None, 1, dec_dim)  
        # r_out.shape = (None, dec_dim)  
        r_out, hidden = self.gru(x, initial_state=hidden)  
  
        # 시계열 축 제거  
        # (None, 1, dec_dim) -> (None, dec_dim)  
        out = tf.reshape(r_out, shape=(-1, r_out.shape[2]))  
  
        return self.fc(out), hidden, attn_weights
```


4. 코드 구현(Bahdanau Attention)

Encoder

Attention

Decoder



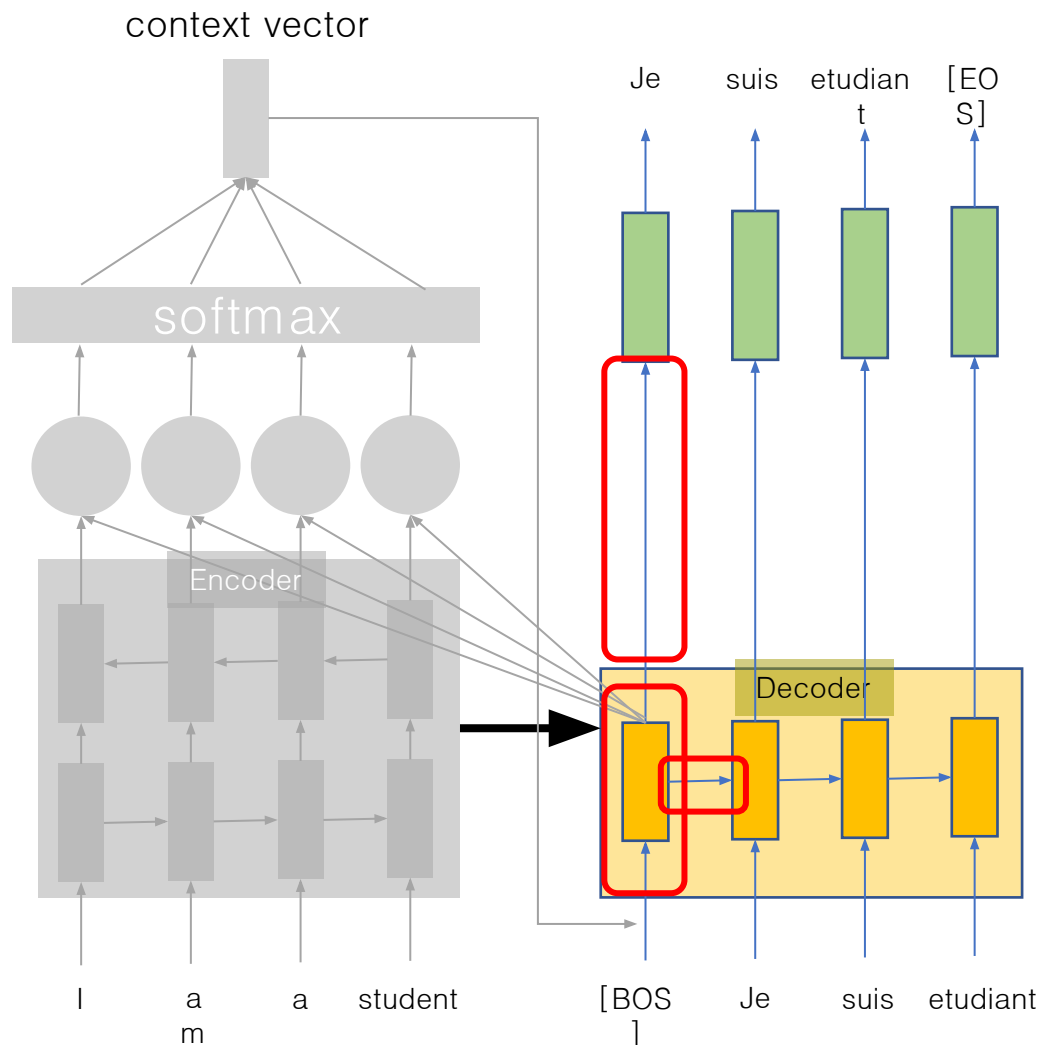
```
class Decoder(tf.keras.models.Model):  
    def __init__(self, vocab_size, embedding_dim=256, dec_dim=256, batch_size=32):  
        super(Decoder, self).__init__()  
        self.vocab_size = vocab_size  
        self.dec_dim = dec_dim  
        self.embedding_dim = embedding_dim  
        self.batch_size = batch_size  
        self.attn = BahdanauAttention(self.dec_dim)  
        self.embedding = tf.keras.layers.Embedding(self.vocab_size, self.embedding_dim)  
        self.gru = tf.keras.layers.GRU(self.dec_dim,  
                                         recurrent_initializer='glorot_uniform',  
                                         return_sequences=True,  
                                         return_state=True)  
        self.fc = tf.keras.layers.Dense(self.vocab_size)  
  
    def call(self, x, hidden, enc_out):  
        # x.shape = (None, 1)  
        # enc_out.shape = (None, seq_length, enc_dim)  
        # enc_hidden.shape = (None, enc_dim)  
  
        # decoder input의 워드 임베딩  
        # (None, 1) -> (None, 1, embedding_dim)  
        x = self.embedding(x)  
  
        # attention 가중치 계산  
        # context.shape = (None, enc_dim)  
        # attn_weights.shape = (None, seq_length, enc_dim)  
        context, attn_weights = self.attn(enc_out, hidden)  
  
        # x.shape = (None, 1, enc_dim + embedding_dim)  
        x = tf.concat((tf.expand_dims(context, 1), x), -1)  
  
        # Decoder RNN sequence 출력  
        # r_out.shape = (None, 1, dec_dim)  
        # r_out.shape = (None, dec_dim)  
        r_out, hidden = self.gru(x, initial_state=hidden)  
  
        # 시계열 축 제거  
        # (None, 1, dec_dim) -> (None, dec_dim)  
        out = tf.reshape(r_out, shape=(-1, r_out.shape[2]))  
  
        return self.fc(out), hidden, attn_weights
```

4. 코드 구현(Bahdanau Attention)

Encoder

Attention

Decoder



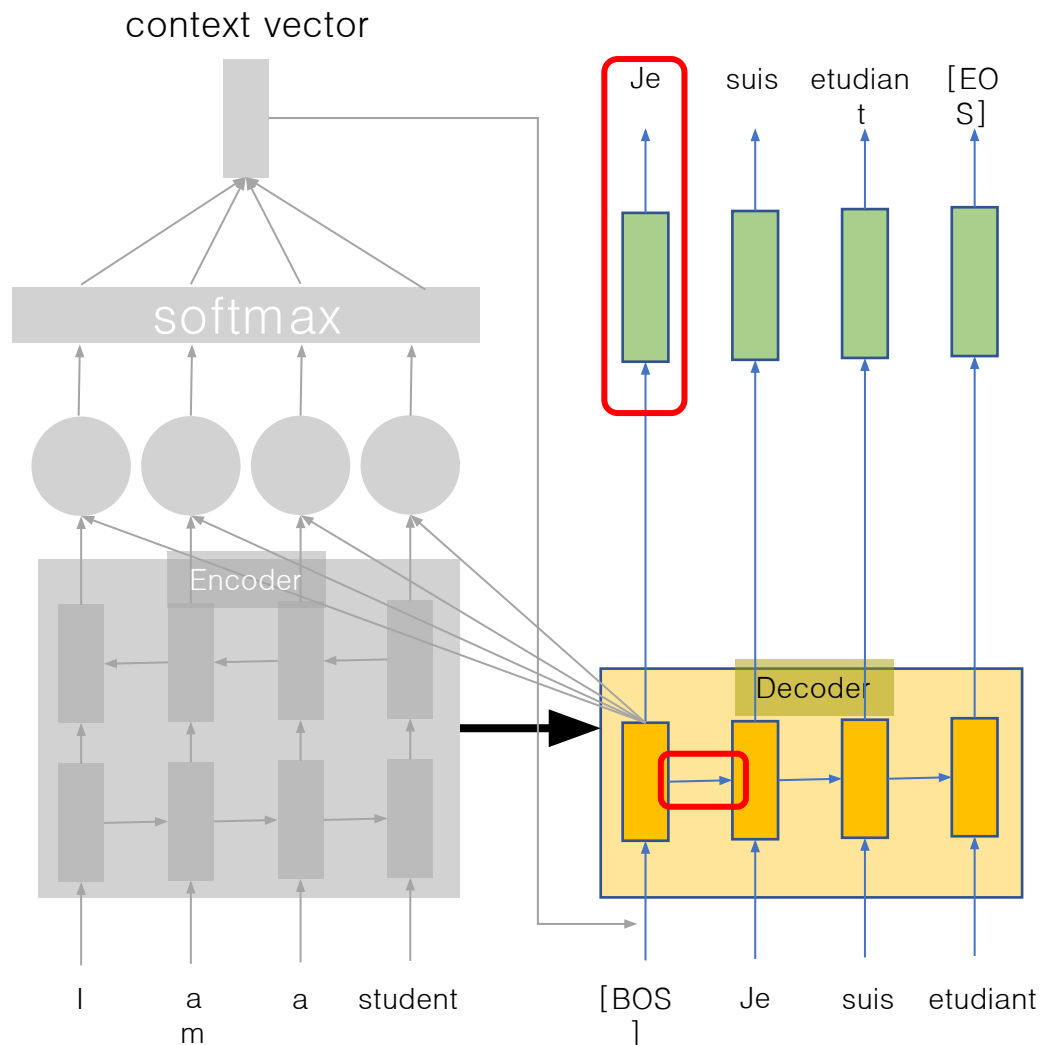
```
class Decoder(tf.keras.models.Model):  
    def __init__(self, vocab_size, embedding_dim=256, dec_dim=256, batch_size=32):  
        super(Decoder, self).__init__()  
        self.vocab_size = vocab_size  
        self.dec_dim = dec_dim  
        self.embedding_dim = embedding_dim  
        self.batch_size = batch_size  
        self.attn = BahdanauAttention(self.dec_dim)  
        self.embedding = tf.keras.layers.Embedding(self.vocab_size, self.embedding_dim)  
        self.gru = tf.keras.layers.GRU(self.dec_dim,  
                                        recurrent_initializer='glorot_uniform',  
                                        return_sequences=True,  
                                        return_state=True)  
        self.fc = tf.keras.layers.Dense(self.vocab_size)  
  
    def call(self, x, hidden, enc_out):  
        # x.shape = (None, 1)  
        # enc_out.shape = (None, seq_length, enc_dim)  
        # enc_hidden.shape = (None, enc_dim)  
  
        # decoder input의 워드 임베딩  
        # (None, 1) -> (None, 1, embedding_dim)  
        x = self.embedding(x)  
  
        # attention 가중치 계산  
        # context.shape = (None, enc_dim)  
        # attn_weights.shape = (None, seq_length, enc_dim)  
        context, attn_weights = self.attn(enc_out, hidden)  
  
        # x.shape = (None, 1, enc_dim + embedding_dim)  
        x = tf.concat((tf.expand_dims(context, 1), x), -1)  
  
        # Decoder RNN sequence 출력  
        # r_out.shape = (None, 1, dec_dim)  
        # r_out.shape = (None, dec_dim)  
        r_out, hidden = self.gru(x, initial_state=hidden)  
  
        # 시계열 축 제거  
        # (None, 1, dec_dim) -> (None, dec_dim)  
        out = tf.reshape(r_out, shape=(-1, r_out.shape[2]))  
  
        return self.fc(out), hidden, attn_weights
```

4. 코드 구현(Bahdanau Attention)

Encoder

Attention

Decoder



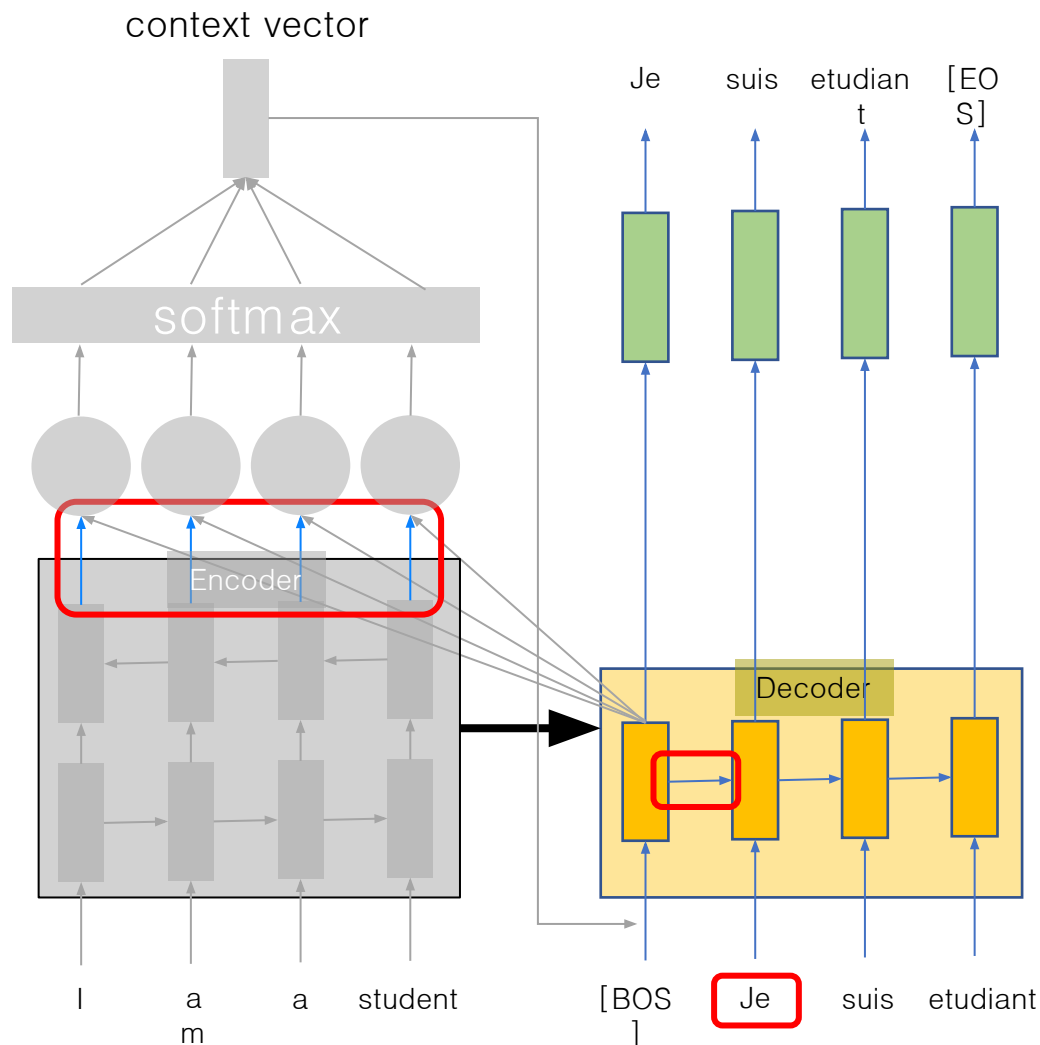
```
class Decoder(tf.keras.models.Model):  
    def __init__(self, vocab_size, embedding_dim=256, dec_dim=256, batch_size=32):  
        super(Decoder, self).__init__()  
        self.vocab_size = vocab_size  
        self.dec_dim = dec_dim  
        self.embedding_dim = embedding_dim  
        self.batch_size = batch_size  
        self.attn = BahdanauAttention(self.dec_dim)  
        self.embedding = tf.keras.layers.Embedding(self.vocab_size, self.embedding_dim)  
        self.gru = tf.keras.layers.GRU(self.dec_dim,  
                                         recurrent_initializer='glorot_uniform',  
                                         return_sequences=True,  
                                         return_state=True)  
        self.fc = tf.keras.layers.Dense(self.vocab_size)  
  
    def call(self, x, hidden, enc_out):  
        # x.shape = (None, 1)  
        # enc_out.shape = (None, seq_length, enc_dim)  
        # enc_hidden.shape = (None, enc_dim)  
  
        # decoder input의 워드 임베딩  
        # (None, 1) -> (None, 1, embedding_dim)  
        x = self.embedding(x)  
  
        # attention 가중치 계산  
        # context.shape = (None, enc_dim)  
        # attn_weights.shape = (None, seq_length, enc_dim)  
        context, attn_weights = self.attn(enc_out, hidden)  
  
        # x.shape = (None, 1, enc_dim + embedding_dim)  
        x = tf.concat((tf.expand_dims(context, 1), x), -1)  
  
        # Decoder RNN sequence 출력  
        # r_out.shape = (None, 1, dec_dim)  
        # r_out.shape = (None, dec_dim)  
        r_out, hidden = self.gru(x, initial_state=hidden)  
  
        # 시계열 축 제거  
        # (None, 1, dec_dim) -> (None, dec_dim)  
        out = tf.reshape(r_out, shape=(-1, r_out.shape[2]))  
  
        return self.fc(out), hidden, attn_weights
```


4. 코드 구현(Bahdanau Attention)

Encoder

Attention

Decoder



```
class Decoder(tf.keras.models.Model):
    def __init__(self, vocab_size, embedding_dim=256, dec_dim=256, batch_size=32):
        super(Decoder, self).__init__()
        self.vocab_size = vocab_size
        self.dec_dim = dec_dim
        self.embedding_dim = embedding_dim
        self.batch_size = batch_size
        self.attn = BahdanauAttention(self.dec_dim)
        self.embedding = tf.keras.layers.Embedding(self.vocab_size, self.embedding_dim)
        self.gru = tf.keras.layers.GRU(self.dec_dim,
                                         recurrent_initializer='glorot_uniform',
                                         return_sequences=True,
                                         return_state=True)
        self.fc = tf.keras.layers.Dense(self.vocab_size)

    def call(self, x, hidden, enc_out):
        # x.shape = (None, 1)
        # enc_out.shape = (None, seq_length, enc_dim)
        # enc_hidden.shape = (None, enc_dim)

        # decoder input의 워드 임베딩
        # (None, 1) -> (None, 1, embedding_dim)
        x = self.embedding(x)

        # attention 가중치 계산
        # context.shape = (None, enc_dim)
        # attn_weights.shape = (None, seq_length, enc_dim)
        context, attn_weights = self.attn(enc_out, hidden)

        # x.shape = (None, 1, enc_dim + embedding_dim)
        x = tf.concat((tf.expand_dims(context, 1), x), -1)

        # Decoder RNN sequence 출력
        # r_out.shape = (None, 1, dec_dim)
        # r_out.shape = (None, dec_dim)
        r_out, hidden = self.gru(x, initial_state=hidden)

        # 시계열 축 제거
        # (None, 1, dec_dim) -> (None, dec_dim)
        out = tf.reshape(r_out, shape=(-1, r_out.shape[2]))

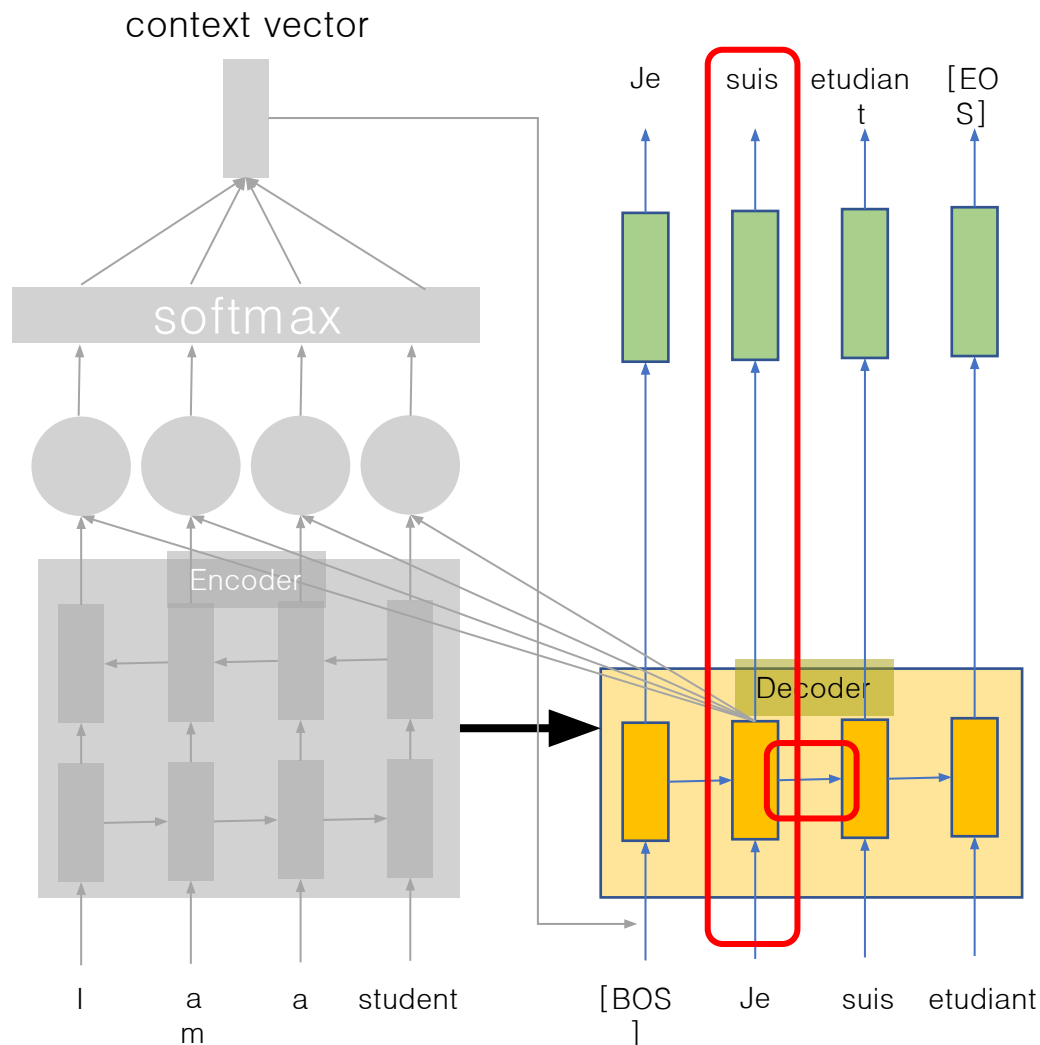
        return self.fc(out), hidden, attn_weights
```

4. 코드 구현(Bahdanau Attention)

Encoder

Attention

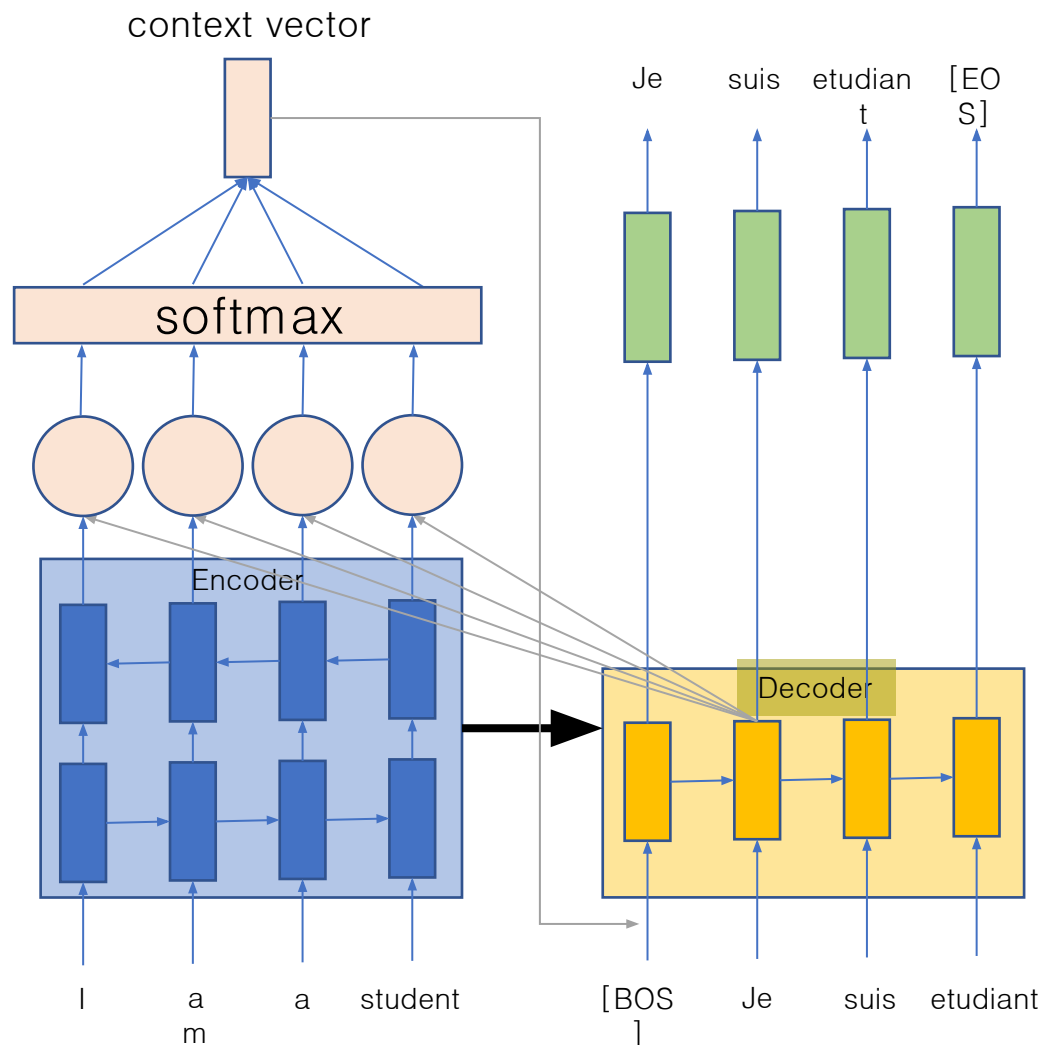
Decoder



```
class Decoder(tf.keras.models.Model):  
    def __init__(self, vocab_size, embedding_dim=256, dec_dim=256, batch_size=32):  
        super(Decoder, self).__init__()  
        self.vocab_size = vocab_size  
        self.dec_dim = dec_dim  
        self.embedding_dim = embedding_dim  
        self.batch_size = batch_size  
        self.attn = BahdanauAttention(self.dec_dim)  
        self.embedding = tf.keras.layers.Embedding(self.vocab_size, self.embedding_dim)  
        self.gru = tf.keras.layers.GRU(self.dec_dim,  
                                         recurrent_initializer='glorot_uniform',  
                                         return_sequences=True,  
                                         return_state=True)  
        self.fc = tf.keras.layers.Dense(self.vocab_size)  
  
    def call(self, x, hidden, enc_out):  
        # x.shape = (None, 1)  
        # enc_out.shape = (None, seq_length, enc_dim)  
        # enc_hidden.shape = (None, enc_dim)  
  
        # decoder input의 워드 임베딩  
        # (None, 1) -> (None, 1, embedding_dim)  
        x = self.embedding(x)  
  
        # attention 가중치 계산  
        # context.shape = (None, enc_dim)  
        # attn_weights.shape = (None, seq_length, enc_dim)  
        context, attn_weights = self.attn(enc_out, hidden)  
  
        # x.shape = (None, 1, enc_dim + embedding_dim)  
        x = tf.concat((tf.expand_dims(context, 1), x), -1)  
  
        # Decoder RNN sequence 출력  
        # r_out.shape = (None, 1, dec_dim)  
        # r_out.shape = (None, dec_dim)  
        r_out, hidden = self.gru(x, initial_state=hidden)  
  
        # 시계열 축 제거  
        # (None, 1, dec_dim) -> (None, dec_dim)  
        out = tf.reshape(r_out, shape=(-1, r_out.shape[2]))  
  
        return self.fc(out), hidden, attn_weights
```


4. 코드 구현(Bahdanau Attention)

Encoder Attention Decoder



```
class Attention_model(tf.keras.Model):
    def __init__(self, enc_vocab_size, dec_vocab_size, embedding_dim, enc_dim, dec_dim, batch_size, end_token_idx=3):
        super(seq2seq, self).__init__()
        # 문장의 끝 토큰 [EOS] index - 3
        self.end_token_idx = end_token_idx

        self.encoder = Encoder(enc_vocab_size, embedding_dim, enc_dim, batch_size)
        self.decoder = Decoder(dec_vocab_size, embedding_dim, dec_dim, batch_size)

    def call(self, x):
        # encoder, decoder input
        input_, target = x

        # encoder 초기값 설정
        enc_hidden = self.encoder.init_hidden(input_)
        # encoder의 RNN 연산 후 출력값
        enc_out, enc_hidden = self.encoder(input_, enc_hidden)

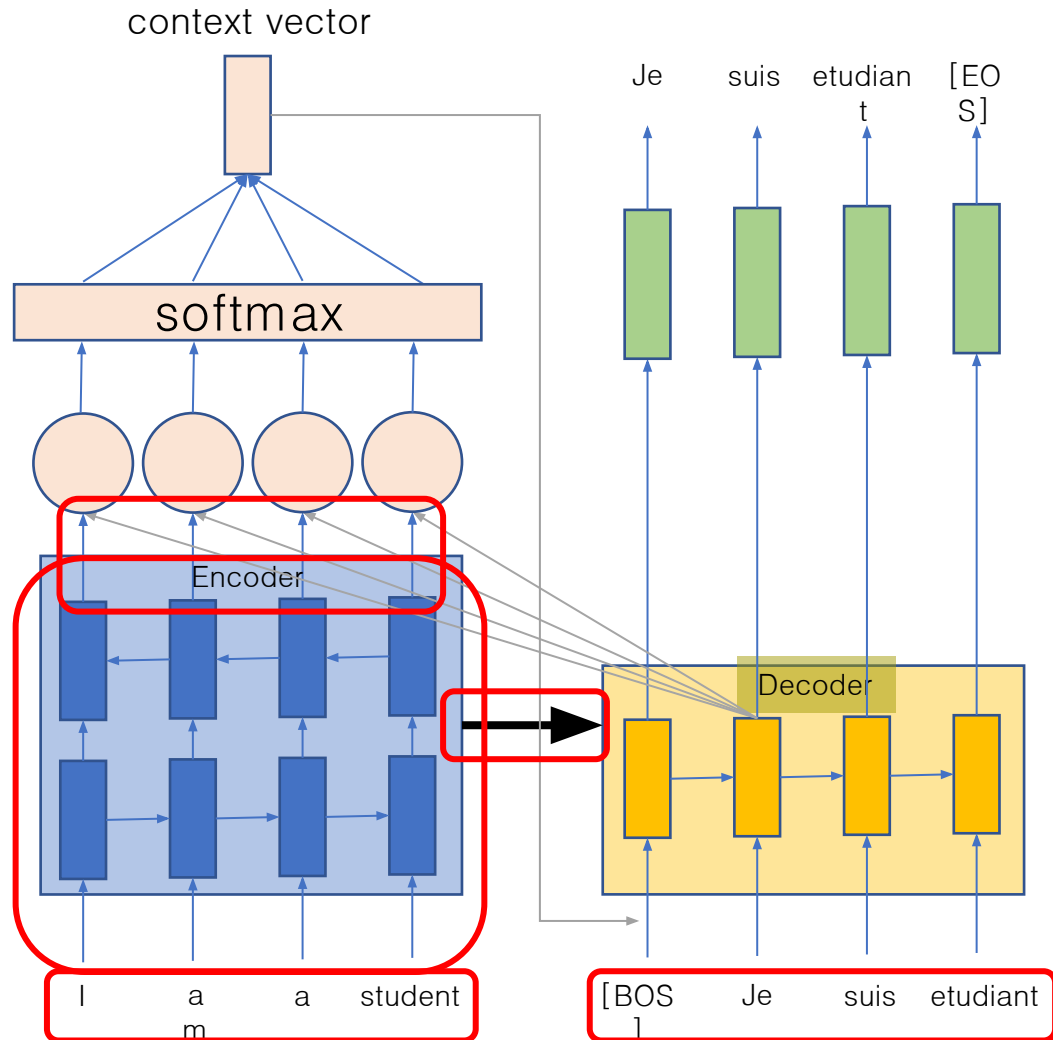
        # dec_hidden 초기값 지정
        dec_hidden = enc_hidden

        predict_tokens = list()
        for t in range(target.shape[1]):
            # decoder input에 시계열 축 추가 (None, 1, 1)
            dec_input = tf.dtypes.cast(tf.expand_dims(target[:, t], 1), tf.float32)
            # decoder RNN 연산 결과
            predictions, dec_hidden, _ = self.decoder(dec_input, dec_hidden, enc_out)
            predict_tokens.append(tf.dtypes.cast(predictions, tf.float32))

        return tf.stack(predict_tokens, axis=1)
```

4. 코드 구현(Bahdanau Attention)

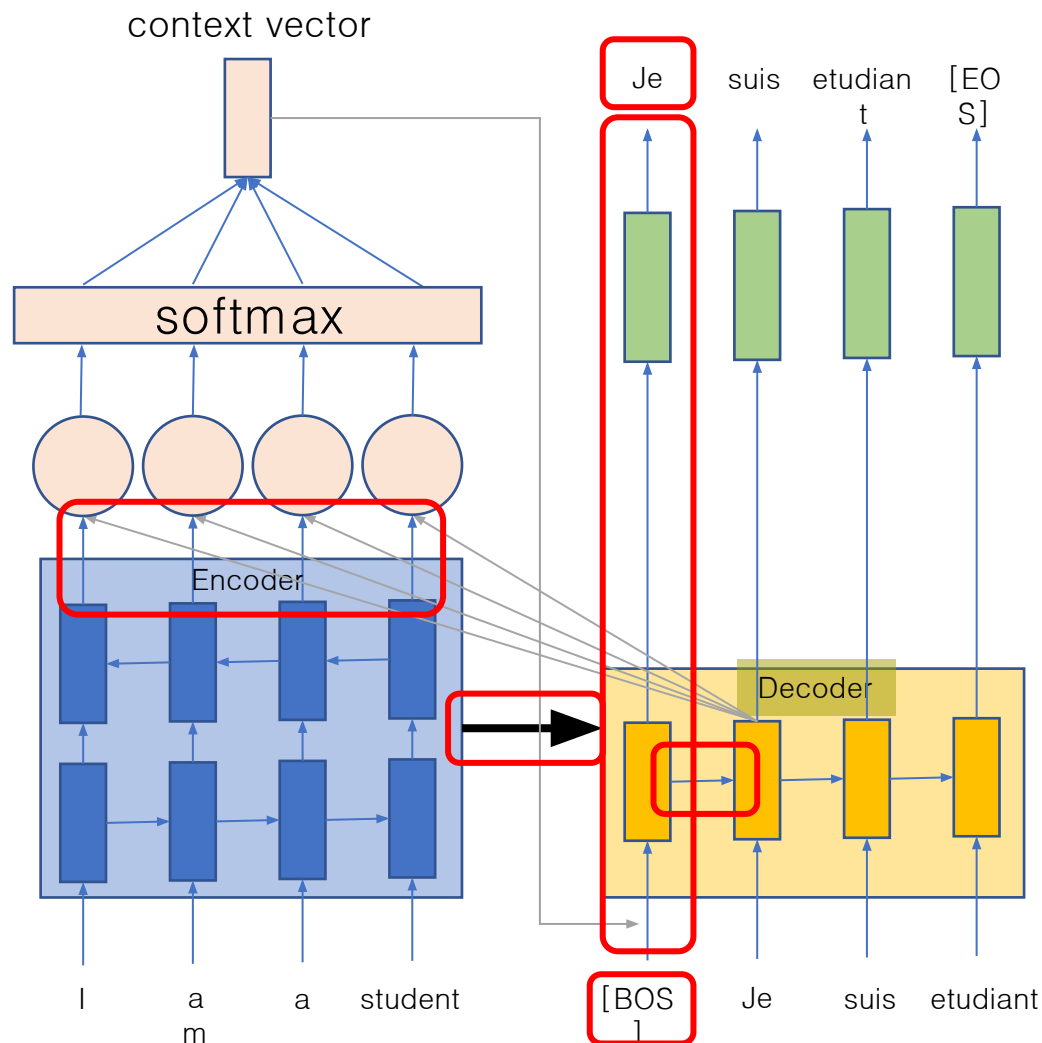
Encoder Attention Decoder



```
class Attention_model(tf.keras.Model):  
    def __init__(self, enc_vocab_size, dec_vocab_size, embedding_dim, enc_dim, dec_dim, batch_size, end_token_idx=3):  
        super(seq2seq, self).__init__()  
        # 문장의 끝 토큰 [EOS] index - 3  
        self.end_token_idx = end_token_idx  
  
        self.encoder = Encoder(enc_vocab_size, embedding_dim, enc_dim, batch_size)  
        self.decoder = Decoder(dec_vocab_size, embedding_dim, dec_dim, batch_size)  
  
    def call(self, x):  
        # encoder, decoder input  
        input_, target = x  
  
        # encoder 초기값 설정  
        enc_hidden = self.encoder.init_hidden(input_)  
        # encoder의 RNN 연산 후 출력값  
        enc_out, enc_hidden = self.encoder(input_, enc_hidden)  
  
        # dec_hidden 초기값 설정  
        dec_hidden = enc_hidden  
  
        predict_tokens = list()  
        for t in range(target.shape[1]):  
            # decoder input에 시계열 축 추가 (None, 1, 1)  
            dec_input = tf.dtypes.cast(tf.expand_dims(target[:, t], 1), tf.float32)  
            # decoder RNN 연산 결과  
            predictions, dec_hidden, _ = self.decoder(dec_input, dec_hidden, enc_out)  
            predict_tokens.append(tf.dtypes.cast(predictions, tf.float32))  
  
        return tf.stack(predict_tokens, axis=1)
```

4. 코드 구현(Bahdanau Attention)

Encoder Attention Decoder



```
class Attention_model(tf.keras.Model):
    def __init__(self, enc_vocab_size, dec_vocab_size, embedding_dim, enc_dim, dec_dim, batch_size, end_token_idx=3):
        super(seq2seq, self).__init__()
        # 문장의 끝 토큰 [EOS] index - 3
        self.end_token_idx = end_token_idx

        self.encoder = Encoder(enc_vocab_size, embedding_dim, enc_dim, batch_size)
        self.decoder = Decoder(dec_vocab_size, embedding_dim, dec_dim, batch_size)

    def call(self, x):
        # encoder, decoder input
        input_, target = x

        # encoder 초기값 설정
        enc_hidden = self.encoder.init_hidden(input_)
        # encoder의 RNN 연산 후 출력값
        enc_out, enc_hidden = self.encoder(input_, enc_hidden)

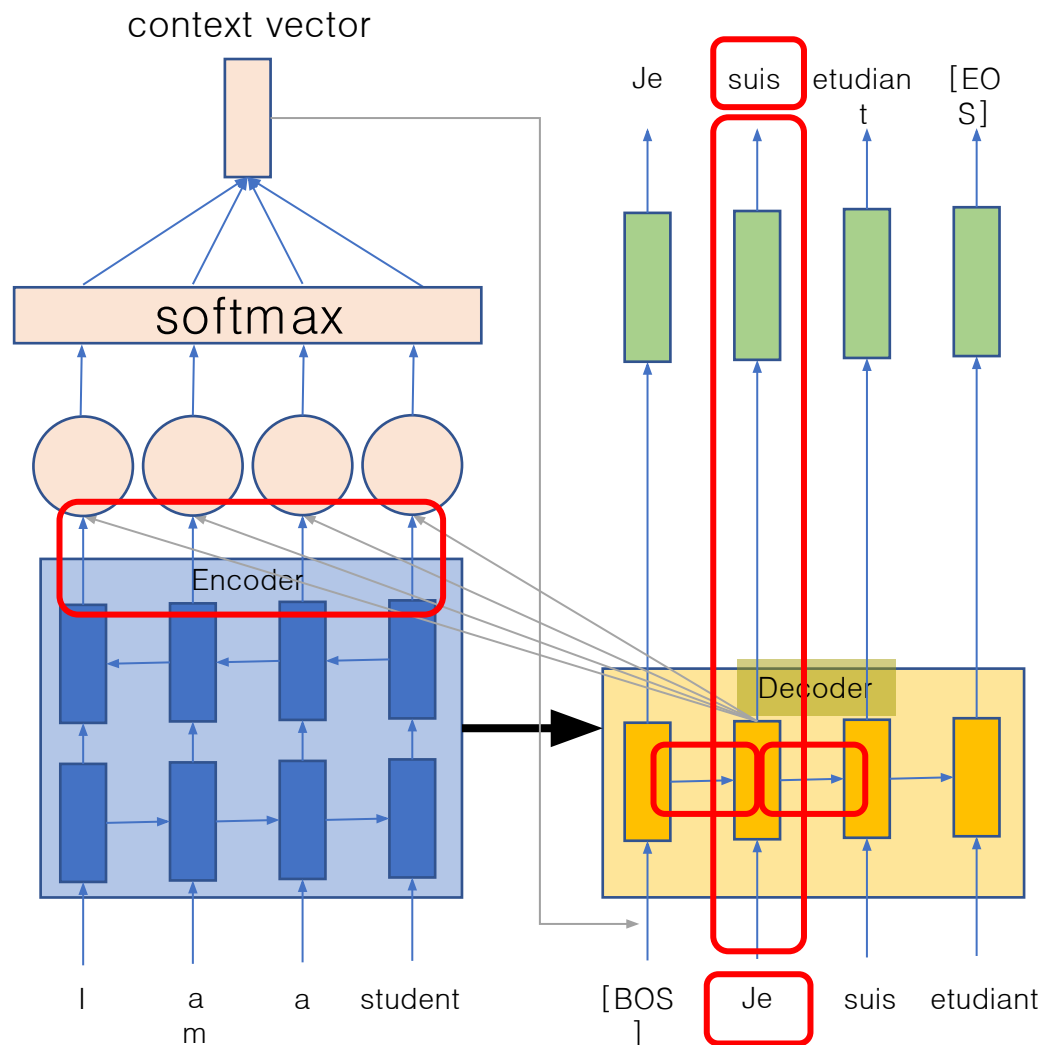
        # dec_hidden 초기값 지정
        dec_hidden = enc_hidden

        predict_tokens = list()
        for t in range(target.shape[1]):
            # decoder input에 시계열 순 추가 (None, 1, 1)
            dec_input = tf.dtypes.cast(tf.expand_dims(target[:, t], 1), tf.float32)
            # decoder RNN 연산 결과
            predictions, dec_hidden, _ = self.decoder(dec_input, dec_hidden, enc_out)
            predict_tokens.append(tf.dtypes.cast(predictions, tf.float32))

        return tf.stack(predict_tokens, axis=1)
```

4. 코드 구현(Bahdanau Attention)

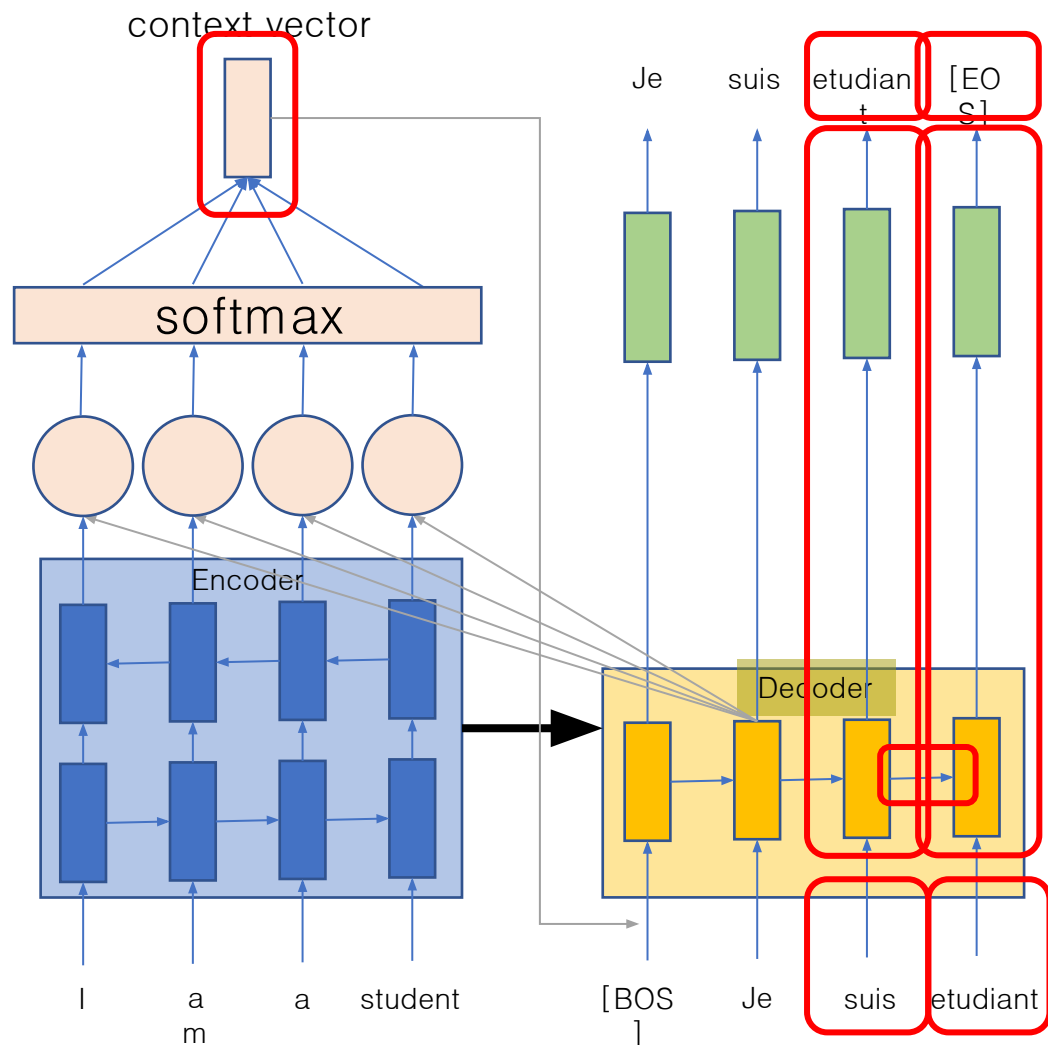
Encoder Attention Decoder



```
class Attention_model(tf.keras.Model):  
    def __init__(self, enc_vocab_size, dec_vocab_size, embedding_dim, enc_dim, dec_dim, batch_size, end_token_idx=3):  
        super(seq2seq, self).__init__()  
        # 문장의 끝 토큰 [EOS] index - 3  
        self.end_token_idx = end_token_idx  
  
        self.encoder = Encoder(enc_vocab_size, embedding_dim, enc_dim, batch_size)  
        self.decoder = Decoder(dec_vocab_size, embedding_dim, dec_dim, batch_size)  
  
    def call(self, x):  
        # encoder, decoder input  
        input_, target = x  
  
        # encoder 초기값 설정  
        enc_hidden = self.encoder.init_hidden(input_)  
        # encoder의 RNN 연산 후 출력값  
        enc_out, enc_hidden = self.encoder(input_, enc_hidden)  
  
        # dec_hidden 초기값 지정  
        dec_hidden = enc_hidden  
  
        predict_tokens = list()  
        for t in range(target.shape[1]):  
            # decoder input에 시계열 순 추가 (None, 1, 1)  
            dec_input = tf.dtypes.cast(tf.expand_dims(target[:, t], 1), tf.float32)  
            # decoder RNN 연산 결과  
            predictions, dec_hidden, _ = self.decoder(dec_input, dec_hidden, enc_out)  
            predict_tokens.append(tf.dtypes.cast(predictions, tf.float32))  
  
        return tf.stack(predict_tokens, axis=1)
```


4. 코드 구현(Bahdanau Attention)

Encoder Attention Decoder



```
class Attention_model(tf.keras.Model):
    def __init__(self, enc_vocab_size, dec_vocab_size, embedding_dim, enc_dim, dec_dim, batch_size, end_token_idx=3):
        super(seq2seq, self).__init__()
        # 문장의 끝 토큰 [EOS] index - 3
        self.end_token_idx = end_token_idx

        self.encoder = Encoder(enc_vocab_size, embedding_dim, enc_dim, batch_size)
        self.decoder = Decoder(dec_vocab_size, embedding_dim, dec_dim, batch_size)

    def call(self, x):
        # encoder, decoder input
        input_, target = x

        # encoder 초기값 설정
        enc_hidden = self.encoder.init_hidden(input_)
        # encoder의 RNN 연산 후 출력값
        enc_out, enc_hidden = self.encoder(input_, enc_hidden)

        # dec_hidden 초기값 지정
        dec_hidden = enc_hidden

        predict_tokens = list()
        for t in range(target.shape[1]):
            # decoder input에 시계열 속 추가 (None, 1, 1)
            dec_input = tf.dtypes.cast(tf.expand_dims(target[:, t], 1), tf.float32)
            # decoder RNN 연산 결과
            predictions, dec_hidden, _ = self.decoder(dec_input, dec_hidden, enc_out)
            predict_tokens.append(tf.dtypes.cast(predictions, tf.float32))

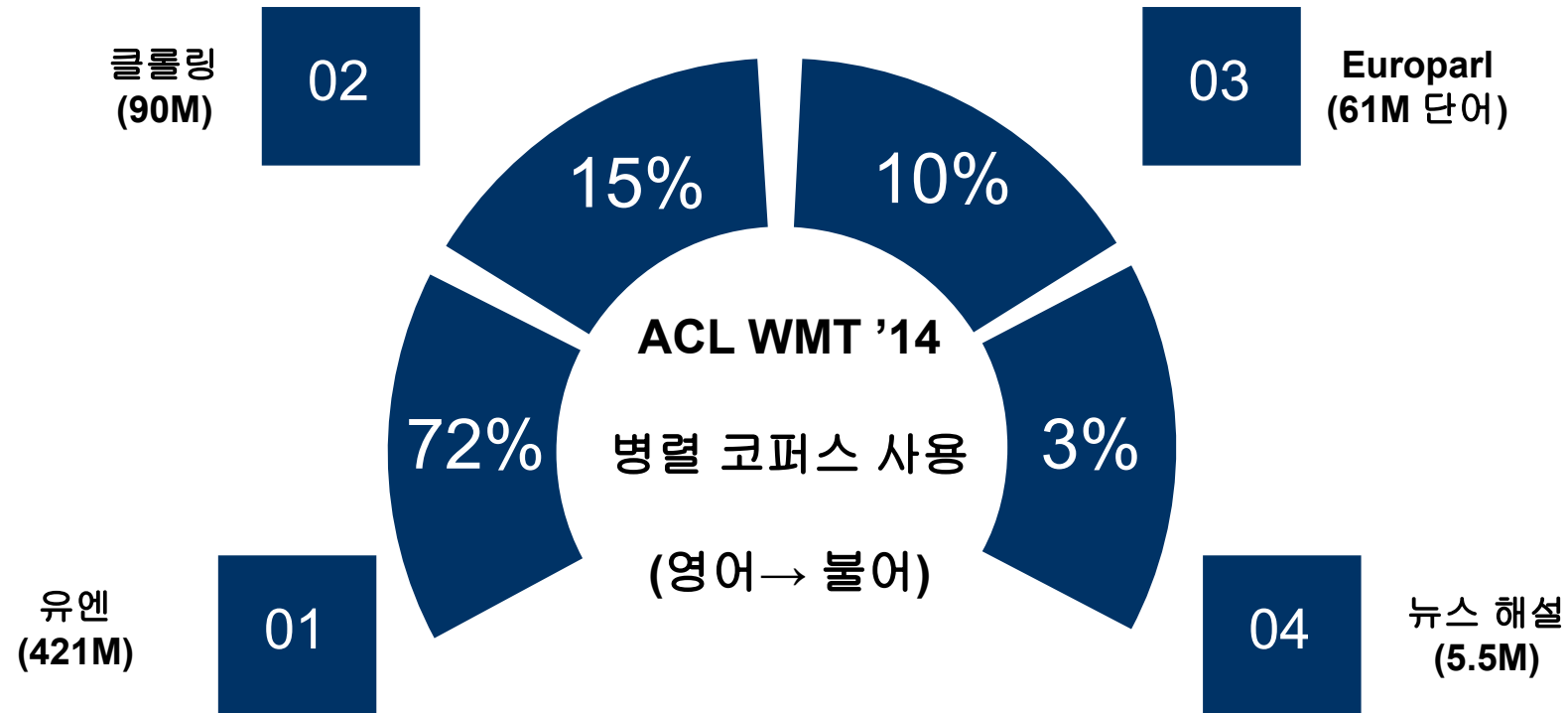
        return tf.stack(predict_tokens, axis=1)
```

05 EXPERIMENT

사용한 데이터셋과 모델 학습 및 결과

5. Experiment

5-1. Dataset



- 전체 코퍼스의 단어를 348M 개로 제한 (Data selection method)
- monolingual data는 하나도 사용하지 않음
- 토큰화 후 전체 단어 중 가장 많이 사용되는 30,000개의 단어를 사용함
- 없는 단어는 [UNK] 토큰으로 매핑하여 사용함

5. Experiment

5-2. Models

구분 \ 모델		RNN Encoder-Decoder	RNNsearch (proposed model)
최대 문장 길이	30	최대 30개의 단어로 구성된 문장으로 학습 진행 (RNNencdec-30)	최대 30개의 단어로 구성된 문장으로 학습 진행 (RNNsearch-30)
	50	최대 50개의 단어로 구성된 문장으로 학습 진행 (RNNencdec-50)	최대 50개의 단어로 구성된 문장으로 학습 진행 (RNNsearch-50)
Hidden unit	Encoder	1000개의 hidden unit 존재	Forward RNN : 1000개의 hidden unit을 보유 Backward RNN : 1000개의 hidden unit을 보유
	Decoder	1000개의 hidden unit 존재	Forward RNN : 1000개의 hidden unit을 보유

Optimizer & Train

- **SGD** 알고리즘과 **Adadelata**(Adagrad+RMSprop+Momentum) 사용
- **minibatch**는 80개의 문장으로 구성
- 총 5일 동안 학습을 진행

5. Experiment

5-3. Quantitative Results

BLEU score table

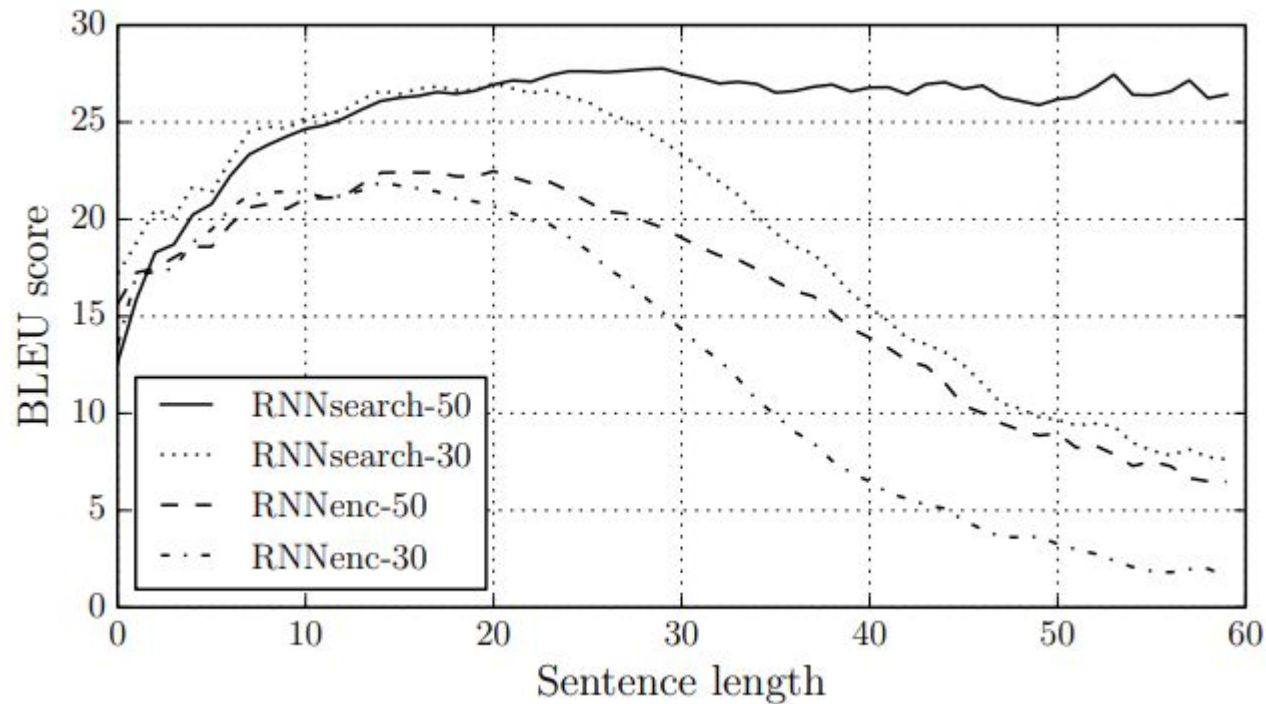
Model	All	No UNK ^o
RNNencdec-30	13.93	24.19
RNNsearch-30	21.50	31.44
RNNencdec-50	17.82	26.71
RNNsearch-50	26.75	34.16
RNNsearch-50*	28.45	36.15
Moses	33.30	35.63

- RNNsearch가 RNNencdec과 비교할 때 모든 모델에서 높은 성능을 보임(BLEU score)
- <UNK>가 없을 때 기존의 Moses(phrase-based translation system)보다 높은 성능을 보임
 - Moses 모델의 경우 기존 데이터 외에 추가적인 monolingual corpus를 사용

5. Experiment

5-3. Quantitative Results

BLEU score figure



1. 모델 제안 배경

- 고정된 길이의 **context vector**를 사용한 모델은 긴 문장의 번역 성능을 저하시킴

2. 성능

- RNNsearch모델이 RNNencdec모델 보다 모두 높은 성능을 보임
- RNNsearch-50모델은 **문장이 길이가 길어져도 성능 저하가 없음**

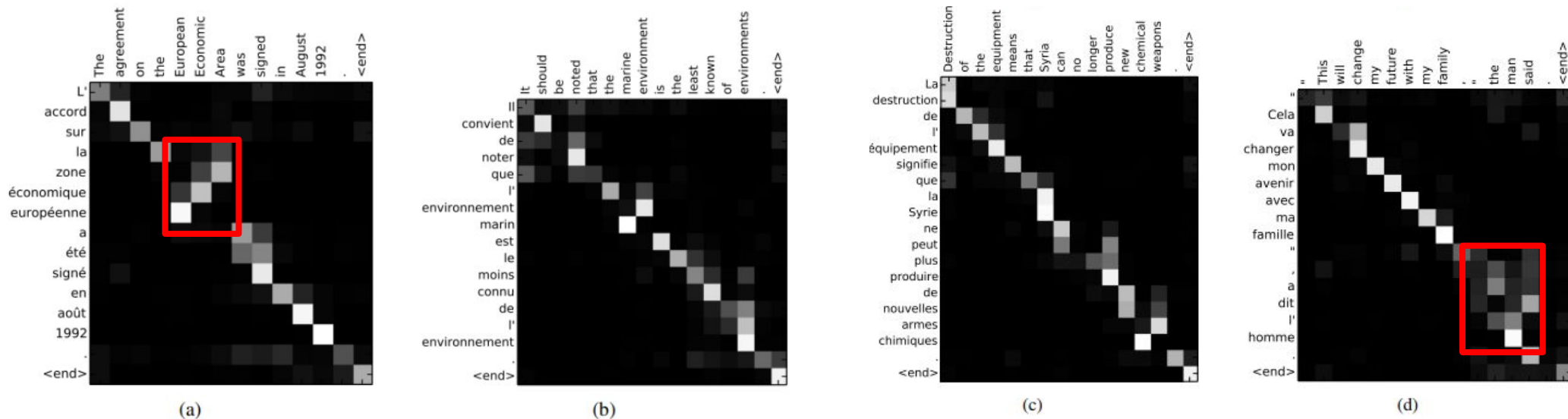
5. Experiment

5-4. Qualitative Analysis

(1) Alignment

RNNsearch는 원본 문장과 생성된 번역문 간의 **soft-alignment**를 검사하는 직관적인 방법 나타냄

annotation 가중치 (a_{ij})를 통해 원본 문장의 어떤 부분이 번역 단어를 생성하는데 더 중요한지 시각화하여 파악 가능

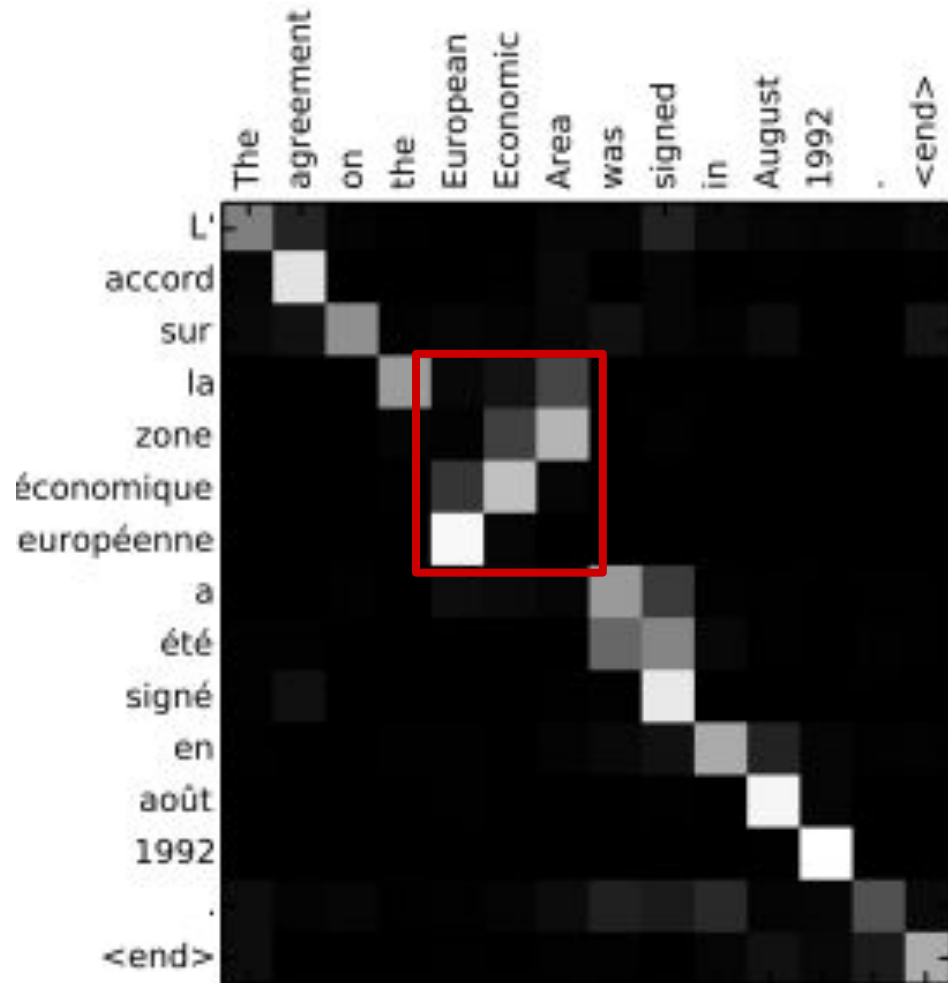


- x축은 원본 문장인 영어이고 y축은 번역된 불어
- 주로 대각선이 강한 weight를 가지는 것을 볼 수 있음
- 빨간색 box부분과 같이 가끔 단조롭지 않은 가중치를 볼 수 있음

5. Experiment

5-4. Qualitative Analysis

(1) Alignment



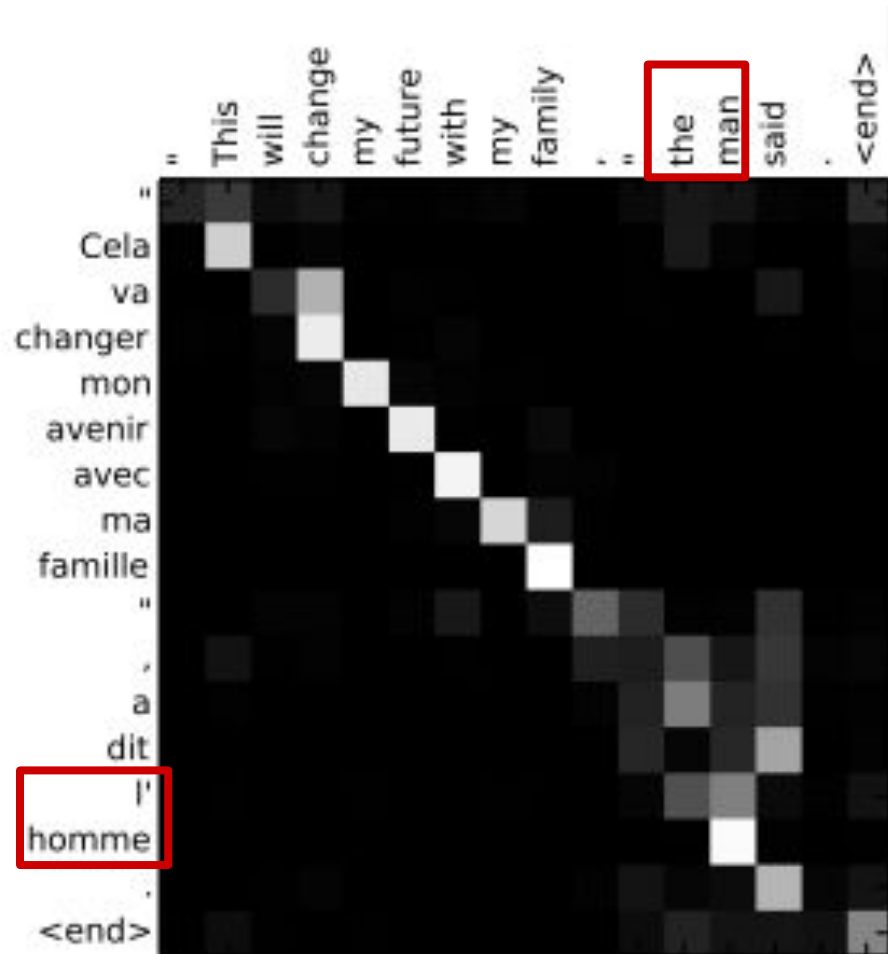
(a)

- 영어와 불어는 형용사와 명사 어순이 다름
- [European Economic **Area**] → [**zone** économique européenne]
- 어순이 다른 문장도 잘 번역한 것을 확인할 수 있음

5. Experiment

5-4. Qualitative Analysis

(1) Alignment



(d)

- 불어는 여성형, 남성형이 존재하기 때문에 [the]가 [le], [la], [les], [l']으로 해석이 될 수 있음
- hard-alignment(일대일 매핑)의 경우 단순히 [the]를 [l']로 매핑할 것임
- soft-alignment는 [the man] → [l' homme]로 올바르게 번역함
- 원본 문장과 번역된 문장이 다른 길이를 가지는 것도 해결할 수 있음

5. Experiment

5-4. Qualitative Analysis

(2) Long Sentences

원본 문장

An admitting privilege is the right of a doctor to admit a patient to a hospital or a medical centre to carry out a diagnosis or a procedure, based on his status as a health care worker at a hospital.

RNNencdec-50

Un privilège d'admission est le droit d'un médecin de reconnaître un patient à l'hôpital ou un centre médical d'un diagnostic ou de prendre un diagnostic en fonction de son état de santé.

밑줄친 부분부터 원본 문장과 의미가 다름

→ [based on his status as a health care worker at a hospital] → [based on his state of health]

RNNsearch-50

Un privilège d'admission est le droit d'un médecin d'admettre un patient à un hôpital ou un centre médical pour effectuer un diagnostic ou une procédure, selon son statut de travailleur des soins de santé à l'hôpital.

긴 문장도 원본 문장의 의미를 모두 올바르게 번역

5. Experiment

5-4. Qualitative Analysis

(2) Long Sentences

원본 문장

This kind of experience is part of Disney's efforts to "extend the lifetime of its series and build new relationships with audiences via digital platforms that are becoming ever more important," he added.

RNNencdec-50

Ce type d'expérience fait partie des initiatives du Disney pour "prolonger la durée de vie de ses nouvelles et de développer des liens avec les lecteurs numériques qui deviennent plus complexes.

- 대략 30개의 단어 이후 원본 문장의 의미와 다르게 번역함
- 달는 따옴표가 생략되는 등 기본적인 오류가 발생함

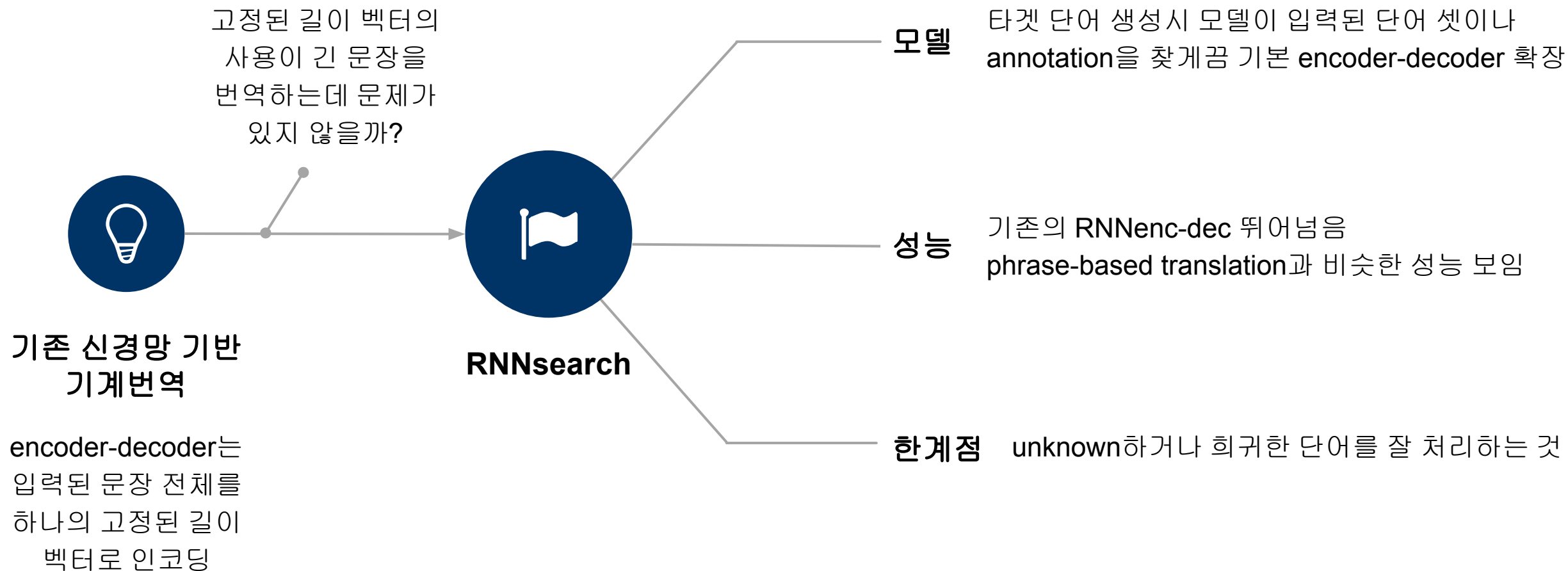
RNNsearch-50

Ce genre d'expérience fait partie des efforts de Disney pour "prolonger la durée de vie de ses séries et créer de nouvelles relations avec des publics via des plateformes numériques de plus en plus importantes", a-t-il ajouté.

- 긴 문장도 원본 문장의 의미를 모두 올바르게 번역

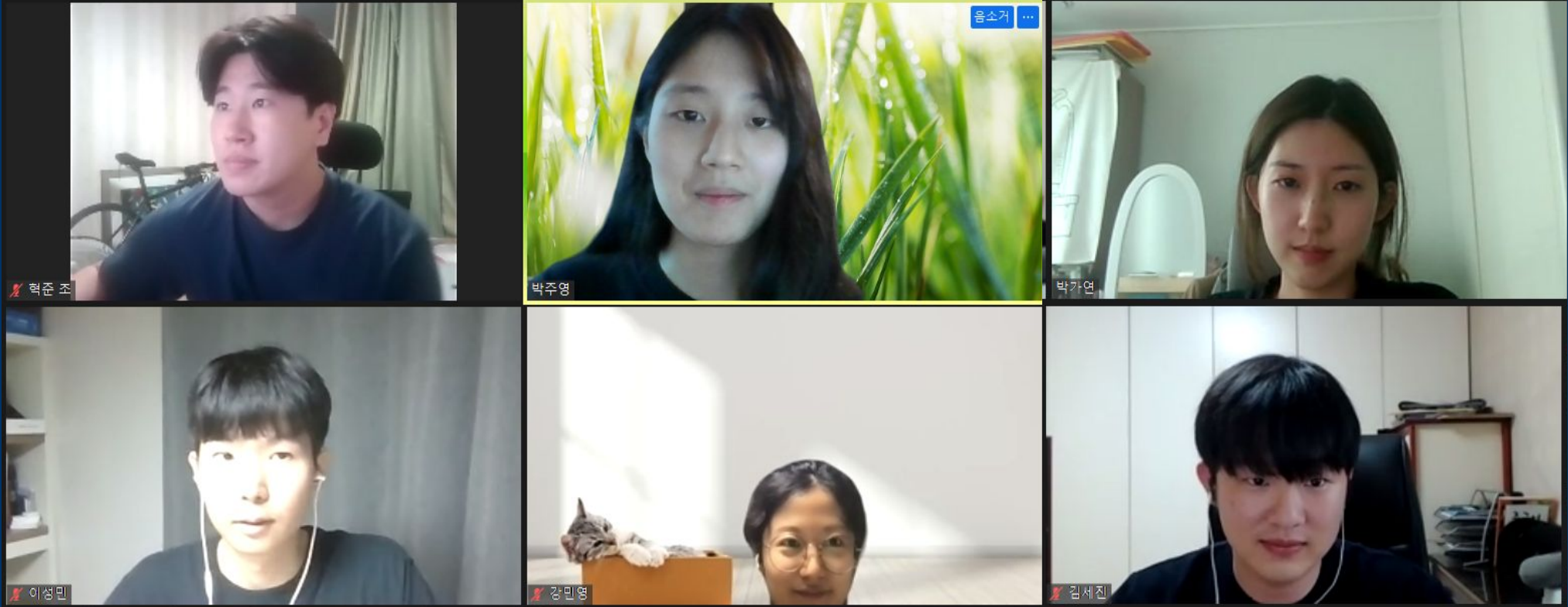
5. Experiment

5-5. Conclusion



<3조 질 수 없조>

박주영(팀장), 강민영, 김세진, 박가연, 이성민, 조혁준



<A반 3조> 질 수 없조

THANK
YOU

<A반 3조> 질 수 없조

Q&A

참고 문헌

- [기계 번역 역사 및 Seq2Seq, Attention] : <https://velog.io/@tobigs-text1415/Lecture-8-Translation-Seq2Seq-Attention>
- [SMT 및 Seq2Seq] : <https://velog.io/@tobigs-text1314/CS224n-Lecture-8-Machine-Translation-Sequence-to-sequence-and-Attention>
- [바다나우 attention 공식] : <https://cpm0722.github.io/paper-review/neural-machine-translation-by-jointly-learning-to-align-and-translate>
- [Seq2Seq 및 공식] : <https://wonjun.oopy.io/papers/seq2seq-with-attention>
- [Seq2Seq] : <https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=winddori2002&logNo=222001331899>
- [Concatenation] : <https://stats.stackexchange.com/questions/524039/why-is-bahdanas-attention-sometimes-called-concat-attention>
-