10. Attention

AILab
Hanyang Univ.

오늘 수업 내용

Attention mechanism?

Attention mechanism 코드

Attention mechanism 결과

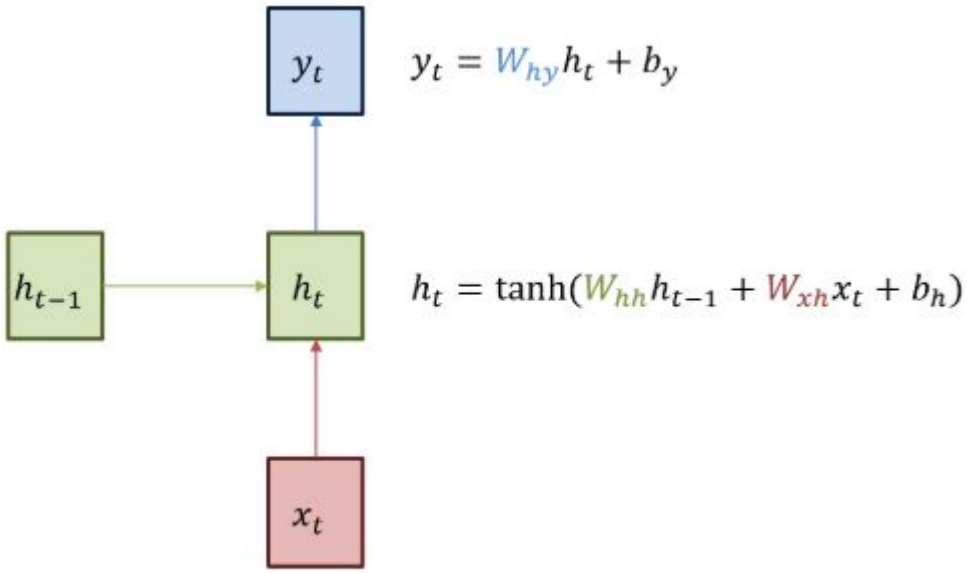
부록 - Neural Machine Translation (seq2seq)

Attention mechanism

Why?

recurrent model

→ t번째에 대한 output을 만들기 위해, t번째 input과 t-1번째 hidden state 이용



(그림1) RNN 기본 구조

Attention mechanism

Why?

recurrent model은 문장의 순차적 특성 유지

하지만, 두 정보 사이의 **거리가 멀 때 해당 정보를 이용하지 못하는** 문제 발생

(Long-term dependency problem)

recurrent model은 학습 시 t번째 hidden state를 얻기 위해 t-1번째 hidden state 필요

→ 즉, 순서대로 계산되어야 하기 때문에 **병렬 처리 불가능** & **속도가 느림**

(Parallelization)

Attention mechanism

Idea?

Attention

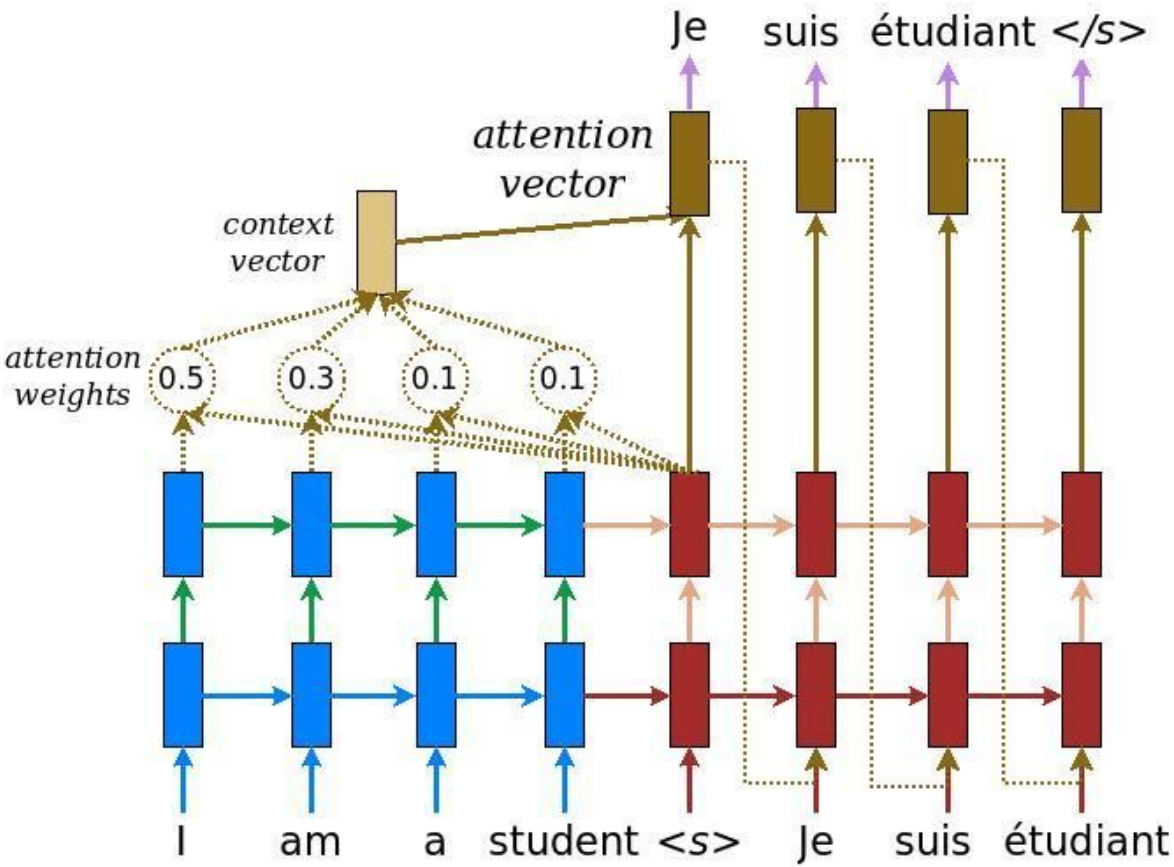
→ 모델이 중요한 부분에 **집중(Attention)**하게 만들자!

•Decoder에서 출력 단어를 예측하는 매 시점 마다 Encoder에서의 전체 입력 문장을 다시 참고

•동일한 비율로 참고하는 것이 아닌 해당 시점에서 예측해야 할 단어와 연관 있는 입력 단어 부분에 좀 더 집중해서(Attention)해서 보겠다

Attention mechanism

대표적으로 Neural Machine Translation 에 사용됨



Attention mechanism

Idea?

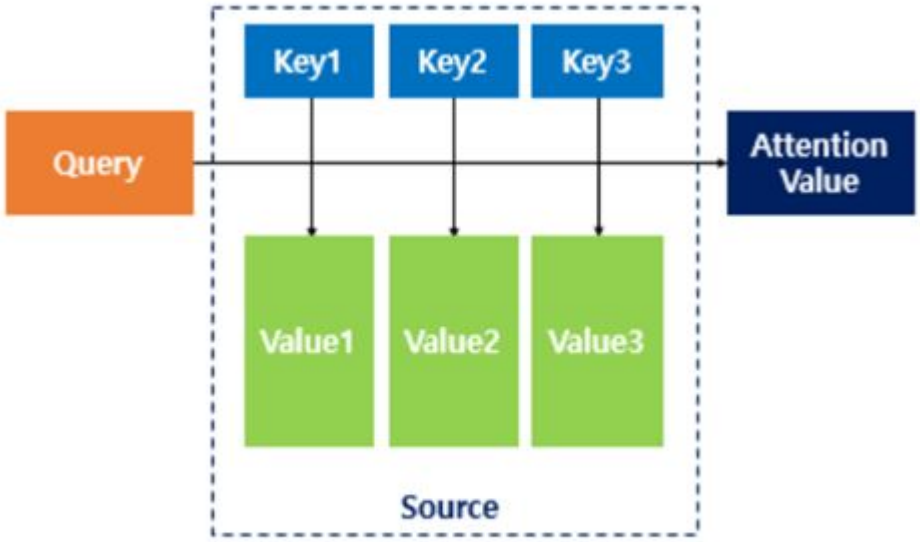
독일어 “Ich mochte ein bier”를 영어 “I’d like a beer”로 번역할 때,

모델이 네번째 단어인 ‘beer’ 예측 시 ‘bier’에 주목하게 만드는 것

Attention mechanism

Attention Function

Attention을 함수로 표현하면, **Attention(Q, K, V) = Attention Value**



(그림2) Attention 함수

Attention mechanism

Attention Function

Attention(Q, K, V) = Attention Value

→ Query에 대해 모든 Key와의 유사도를 각각 구함

→ 이 유사도를 키와 매핑 되어 있는 각각의 Value에 반영

→ 유사도가 반영된 Value를 모두 더해서 리턴 (이 값이 Attention Value)

Attention mechanism

Attention Function

Seq2seq+Attention 모델에서 Q, K, V ?

Attention(Q, K, V) = Attention Value

Q = Query : **Decoder의 t-1 셀에서의 은닉 상태**

K = Keys : 모든 **Encoder** 셀의 은닉 상태들

V = Values : 모든 **Encoder** 셀의 은닉 상태들

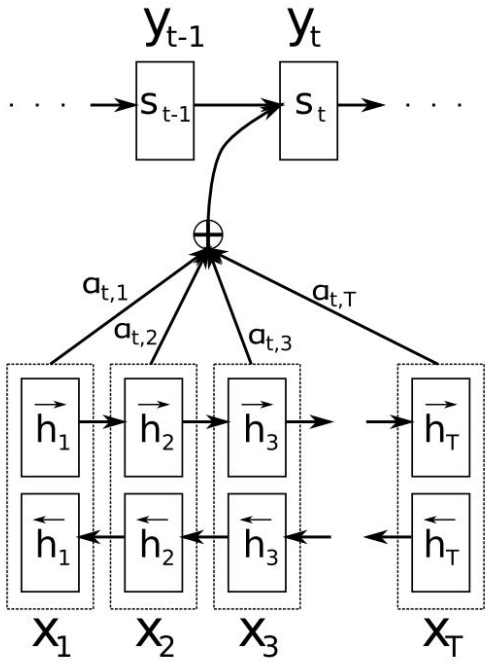
Attention mechanism

Code

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

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

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



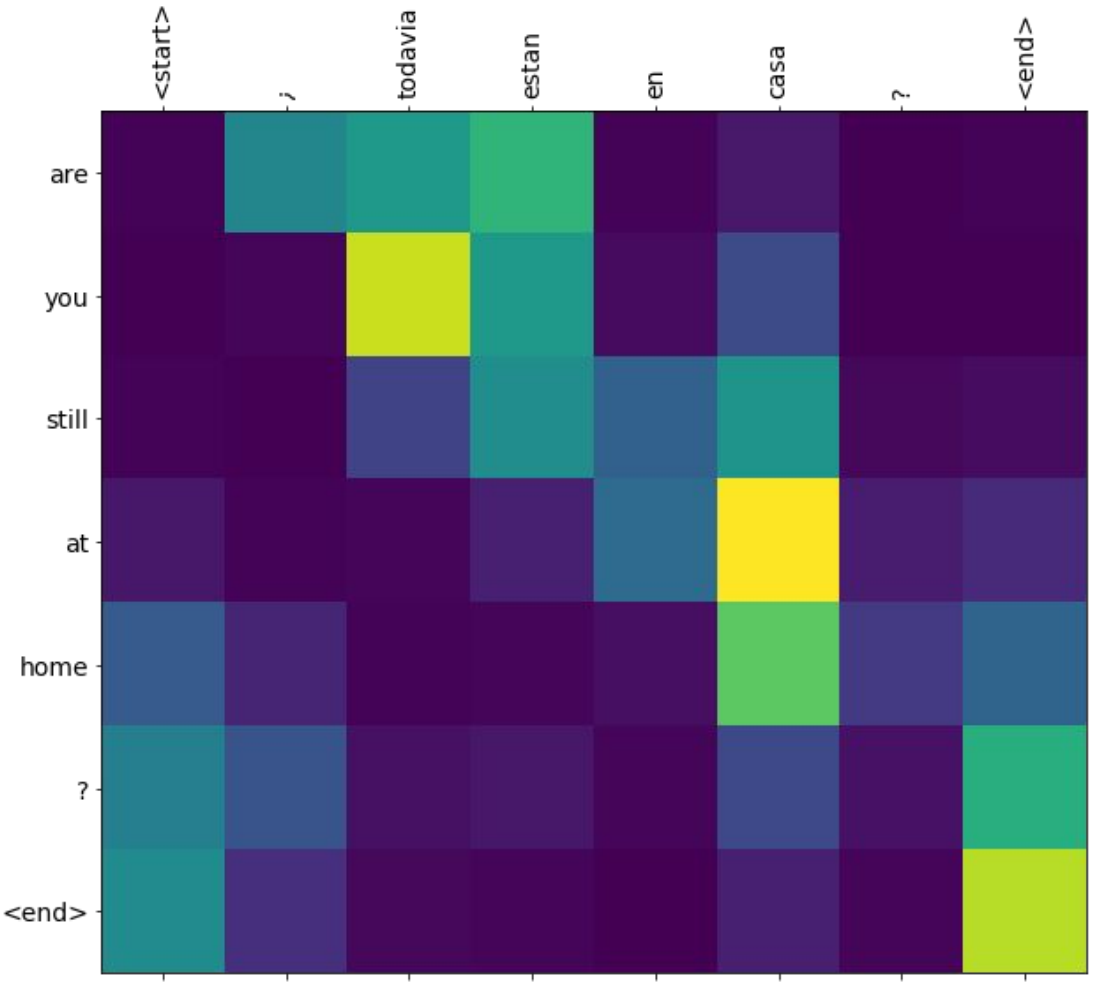

```
# Trainable parameters
w_omega = tf.Variable(tf.random_normal([hidden_size, attention_size], stddev=0.1))
b_omega = tf.Variable(tf.random_normal([attention_size], stddev=0.1))
u_omega = tf.Variable(tf.random_normal([attention_size], stddev=0.1))

with tf.name_scope('v'):
        # Applying fully connected layer with non-linear activation to each of the B*T timestamps;
        #  the shape of `v` is (B,T,D)*(D,A)=(B,T,A), where A=attention_size
        v = tf.tanh(tf.tensordot(inputs, w_omega, axes=1) + b_omega)

    # For each of the timestamps its vector of size A from `v` is reduced with `u` vector
    vu = tf.tensordot(v, u_omega, axes=1, name='vu')  # (B,T) shape
    alphas = tf.nn.softmax(vu, name='alphas')         # (B,T) shape

    # Output of (Bi-)RNN is reduced with attention vector; the result has (B,D) shape
    output = tf.reduce_sum(inputs * tf.expand_dims(alphas, -1), 1)
```

Attention mechanism 사용 결과



부록 - Neural Machine Translation (seq2seq)

```python
import tensorflow as tf
import numpy as np

# S: 디코딩 입력의 시작을 나타내는 심볼
# E: 디코딩 출력을 끝을 나타내는 심볼
# P: 현재 배치 데이터의 time step 크기보다 작은 경우 빈 시퀀스를 채우는 심볼
#    예) 현재 배치 데이터의 최대 크기가 4 인 경우
#       word -> ['w', 'o', 'r', 'd']
#       to   -> ['t', 'o', 'P', 'P']
char_arr = [c for c in 'SEPabcdefghijklmnopqrstuvwxyz단어나무놀이소녀키스사랑']
num_dic = {n: i for i, n in enumerate(char_arr)}
dic_len = len(num_dic)


# 영어를 한글로 번역하기 위한 학습 데이터
seq_data = [['word', '단어'], ['wood', '나무'],
            ['game', '놀이'], ['girl', '소녀'],
            ['kiss', '키스'], ['love', '사랑']]


def make_batch(seq_data):
    input_batch = []
    output_batch = []
    target_batch = []

    for seq in seq_data:
        # 인코더 셀의 입력값. 입력단어의 글자들을 한글자씩 떼어 배열로 만든다.
        input = [num_dic[n] for n in seq[0]]
        # 디코더 셀의 입력값. 시작을 나타내는 S 심볼을 맨 앞에 붙여준다.
        output = [num_dic[n] for n in ('S' + seq[1])]
        # 학습을 위해 비교할 디코더 셀의 출력값. 끝나는 것을 알려주기 위해 마지막에 E 를 붙인다.
        target = [num_dic[n] for n in (seq[1] + 'E')]

        input_batch.append(np.eye(dic_len)[input])
        output_batch.append(np.eye(dic_len)[output])
        # 출력값만 one-hot 인코딩이 아님 (sparse_softmax_cross_entropy_with_logits 사용)
        target_batch.append(target)

    return input_batch, output_batch, target_batch

```

부록 - Neural Machine Translation (seq2seq)

```
41
42  ## 옵션 설정
43  learning_rate = 0.01
44  n_hidden = 128
45  total_epoch = 100
46  # 입력과 출력의 형태가 one-hot 인코딩으로 같으므로 크기도 같다.
47  n_class = n_input = dic_len
48
49
50  ## 신경망 모델 구성
51  # Seq2Seq 모델은 인코더의 입력과 디코더의 입력의 형식이 같다.
52  # [batch size, time steps, input size]
53  enc_input = tf.placeholder(tf.float32, [None, None, n_input])
54  dec_input = tf.placeholder(tf.float32, [None, None, n_input])
55  # [batch size, time steps]
56  targets = tf.placeholder(tf.int64, [None, None])
57
58
59  # 인코더 셀을 구성한다.
60  with tf.variable_scope('encode'):
61      enc_cell = tf.nn.rnn_cell.BasicRNNCell(n_hidden)
62      enc_cell = tf.nn.rnn_cell.DropoutWrapper(enc_cell, output_keep_prob=0.5)
63
64      outputs, enc_states = tf.nn.dynamic_rnn(enc_cell, enc_input, dtype=tf.float32)
65
66  # 디코더 셀을 구성한다.
67  with tf.variable_scope('decode'):
68      dec_cell = tf.nn.rnn_cell.BasicRNNCell(n_hidden)
69      dec_cell = tf.nn.rnn_cell.DropoutWrapper(dec_cell, output_keep_prob=0.5)
70
71      # Seq2Seq 모델은 인코더 셀의 최종 상태값을
72      # 디코더 셀의 초기 상태값으로 넣어주는 것이 핵심.
73      outputs, dec_states = tf.nn.dynamic_rnn(dec_cell, dec_input, initial_state=enc_states, dtype=tf.float32)
74
75  model = tf.layers.dense(outputs, n_class, activation=None)
76  cost = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(logits=model, labels=targets))
77  optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)
78
```

부록 - Neural Machine Translation (seq2seq)

```python
## 신경망 모델 학습
sess = tf.Session()
sess.run(tf.global_variables_initializer())

input_batch, output_batch, target_batch = make_batch(seq_data)

for epoch in range(total_epoch):
    _, loss = sess.run([optimizer, cost], feed_dict={enc_input: input_batch, dec_input: output_batch, targets: target_batch})

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.6f}'.format(loss))

print('최적화 완료!')


## 번역 테스트
# 단어를 입력받아 번역 단어를 예측하고 디코딩하는 함수
def translate(word):
    # 이 모델은 입력값과 출력값 데이터로 [영어단어, 한글단어] 사용하지만,
    # 예측시에는 한글단어를 알지 못하므로, 디코더의 입출력값을 의미 없는 값인 P 값으로 채운다.
    # ['word', 'PPPP']
    seq_data = [word, 'P' * len(word)]

    input_batch, output_batch, target_batch = make_batch([seq_data])

    # 결과가 [batch size, time step, input] 으로 나오기 때문에,
    # 2번째 차원인 input 차원을 argmax 로 취해 가장 확률이 높은 글자를 예측 값으로 만든다.
    prediction = tf.argmax(model, 2)

    result = sess.run(prediction, feed_dict={enc_input: input_batch, dec_input: output_batch, targets: target_batch})

    # 결과 값인 숫자의 인덱스에 해당하는 글자를 가져와 글자 배열을 만든다.
    decoded = [char_arr[i] for i in result[0]]

    # 출력의 끝을 의미하는 'E' 이후의 글자들을 제거하고 문자열로 만든다.
    end = decoded.index('E')
    translated = ''.join(decoded[:end])

    return translated
```

부록 - Neural Machine Translation (seq2seq)

```
117
118  print('\n=== 번역 테스트 ===')
119
120  print('word ->', translate('word'))
121  print('wodr ->', translate('wodr'))
122  print('love ->', translate('love'))
123  print('loev ->', translate('loev'))
124  print('abcd ->', translate('abcd'))
125
```

실행 결과

```
Epoch: 0093 cost = 0.000834
Epoch: 0094 cost = 0.000586
Epoch: 0095 cost = 0.000480
Epoch: 0096 cost = 0.000424
Epoch: 0097 cost = 0.000228
Epoch: 0098 cost = 0.000295
Epoch: 0099 cost = 0.001055
Epoch: 0100 cost = 0.000260
최적화 완료!

=== 번역 테스트 ===
word -> 단어
wodr -> 나무
love -> 사랑
loev -> 사랑
abcd -> 사랑이어

Process finished with exit code 0
```