DL Seminar

Attention Mechanism

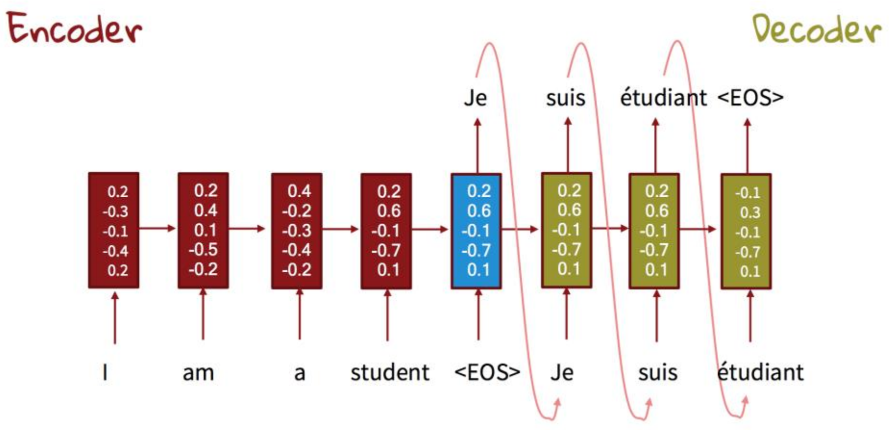
HANYANG UNIVERSITY

인공지능 Lab 김지성
인공지능 Lab 엄희송
인공지능 Lab 유재창

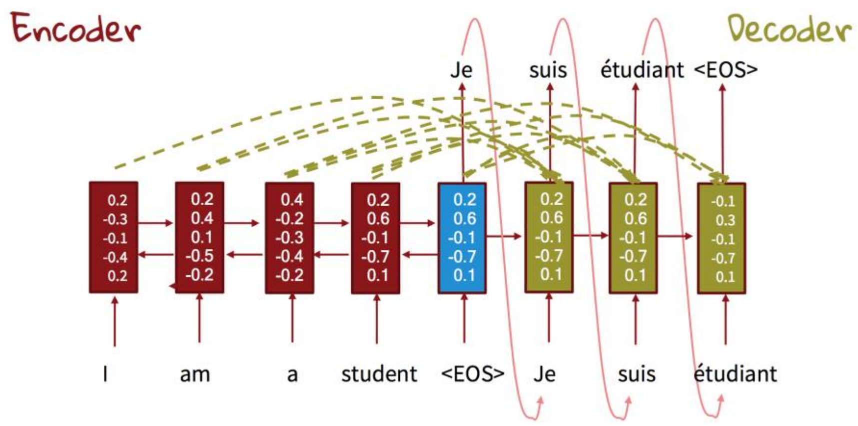
Index

Introduction

Seq2Seq



Fig?. Sec2Sec Model



Fig?.

- 문장 길이가 길고 층이 깊으면, 인코더에서는 정보 손실이 , 디코더에서는 bottle-neck문제 발생
- 이 문제를 해결하기 위해 Attention Machanism이 제안됨
- Bi-directional 네트워크와 함께 사용함

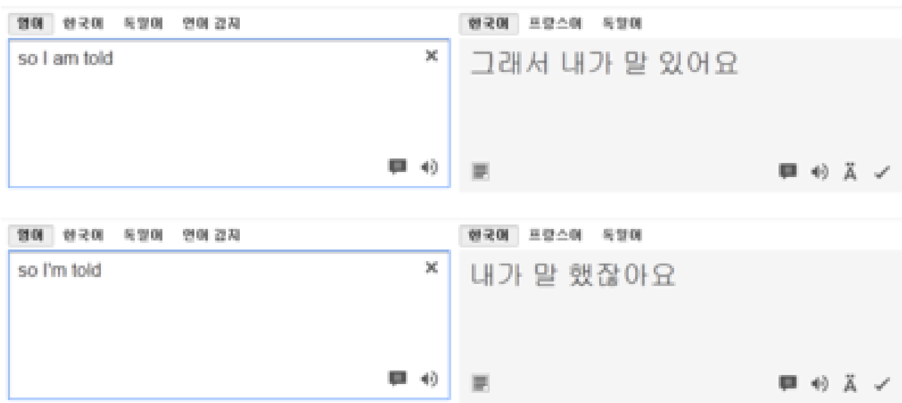
Attention

Idea

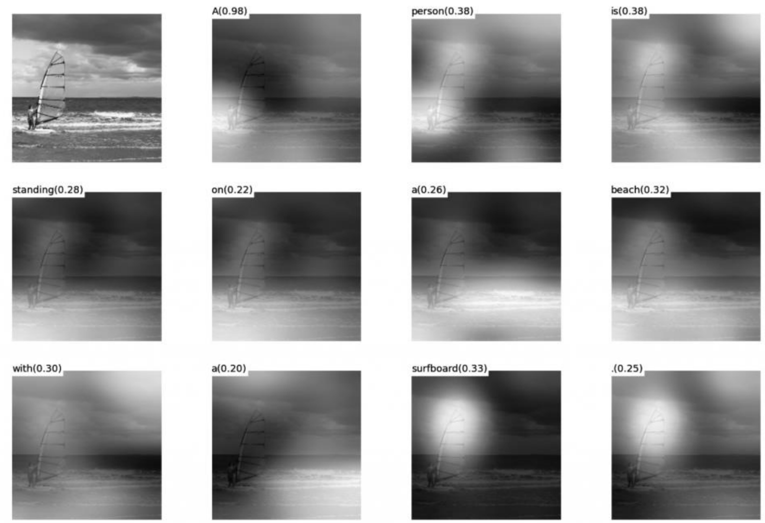
독일어 “ Ich mochte ein bier ”를 영어 “ I’d like a beer ”로 번역하는 S2S 모델에서
인코더가 ‘bier’를 받아서 벡터로 만든 결과(인코더 출력)는 디코더가 ‘beer’를 예측할 때 쓰는 벡터(디코더 입력)와 유사할 것

Attention

When Use This



Fig?. NLP



Fig?. Image captioning

Attention

Mechanism - Encoder

$$\vec{h}_i = \begin{cases} (1 - \vec{z}_i) \circ \vec{h}_{i-1} + \vec{z}_i \circ \vec{\underline{h}}_i & , \text{if } i > 0 \\ 0 & , \text{if } i = 0 \end{cases}$$

$$\begin{aligned} \vec{\underline{h}}_i &= \tanh\left(\vec{W} \bar{E} x_i + \vec{U}\left[\vec{r}_i \circ \vec{h}_{i-1}\right]\right) \\ \vec{z}_i &= \sigma\left(\vec{W}_z \bar{E} x_i + \vec{U}_z \vec{h}_{i-1}\right) \\ \vec{r}_i &= \sigma\left(\vec{W}_r \bar{E} x_i + \vec{U}_r \vec{h}_{i-1}\right). \end{aligned}$$

Fig?. ?? formula

Bi-directional RNN encoder

Forward RNN : $\vec{f} = (\overrightarrow{h_1}, \dots, \overrightarrow{h_{T_x}})$. x_1부터 x_{T_x} 순으로
Backward RNN $\overleftarrow{f} = (\overleftarrow{h_1}, \dots, \overleftarrow{h_{T_x}})$. x_{T_x}부터 x_1 순으로

두 개를 합친 $h_j = [\overrightarrow{h_j^T}; \overleftarrow{h_j^T}]^T$ 벡터를 모아 행렬로 저장.

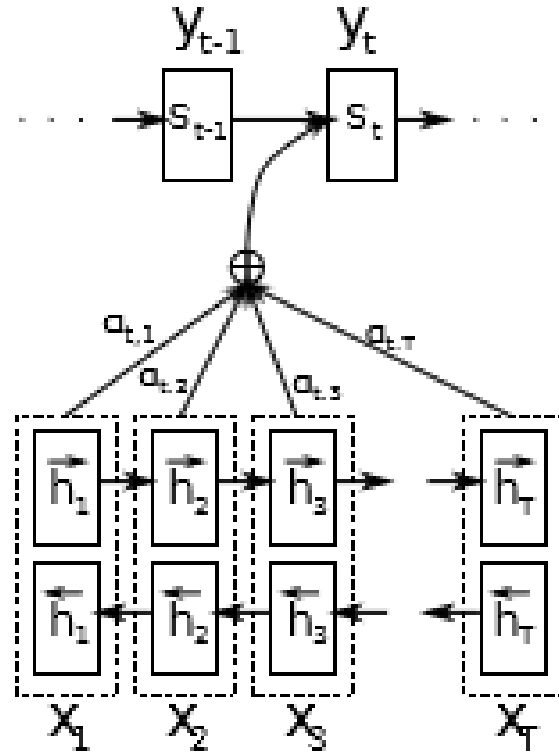


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \dots, x_T)$.

Fig?. ??

Attention

Mechanism - Decoder

$$p(y_i|y_1,\dots,y_{i-1},x) = g(y_{i-1},s_i,c_i)$$
$$s_i = f(s_{i-1},y_{i-1},c_i)$$
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij}h_j$$
$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{Tx} exp(e_{ik})}$$
$$e_{ij} = a(s_{i-1},h_j)$$

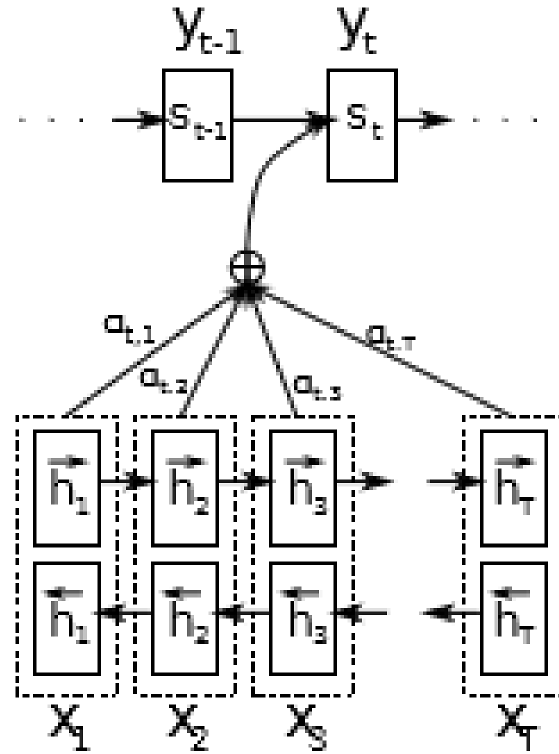


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \dots, x_T)$.

Fig?. ??

Attention

Mechanism - Decoder

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

유사도를 도출할 수 있는 모델이면 모두 사용 가능

$${s_{i-1}}^T \bar{h}_j (dot)$$
$${s_{i-1}}^T W_a \bar{h}_j (general)$$
$$a_t = softmax(W_a h_t)$$

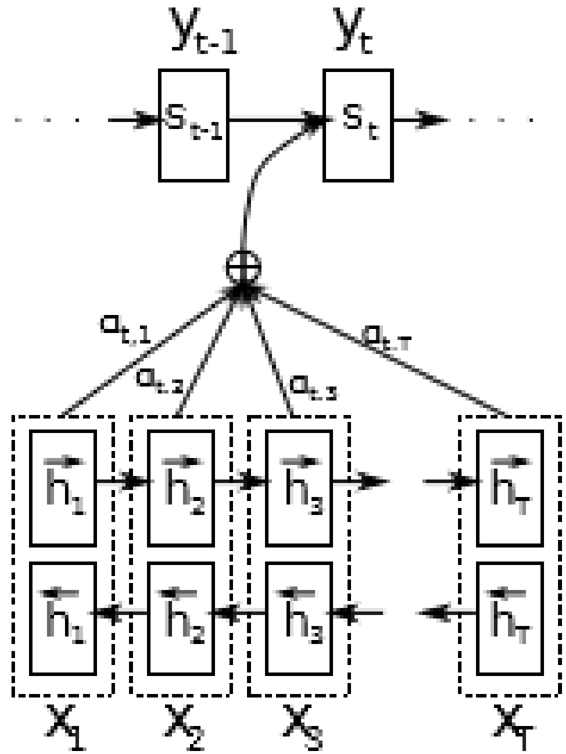


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Fig?. ??

Attention

Mechanism - Decoder

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

e_{ij} : i번째 output이 j번째 h와 얼마나 유사한지를 계산한 함수를 간단히 표현.

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

α_{ij} : weight. source sentence에서 생성된 h에 대해 각각 얼마의 비중을 두고 합할 것인지.

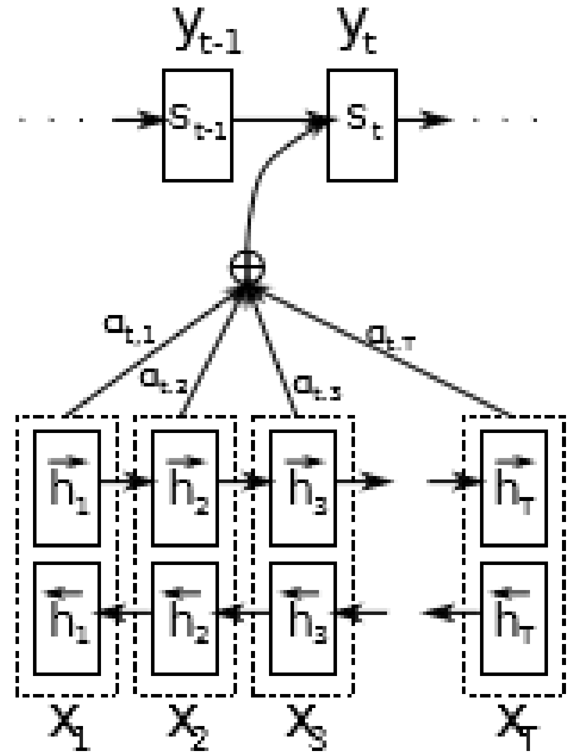


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Fig?. ??

Attention

Mechanism - Decoder

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

c_i : i번째 단어를 추측하기 위해 생성된 context vector

여기까지의 과정이 s_{i-1}에서

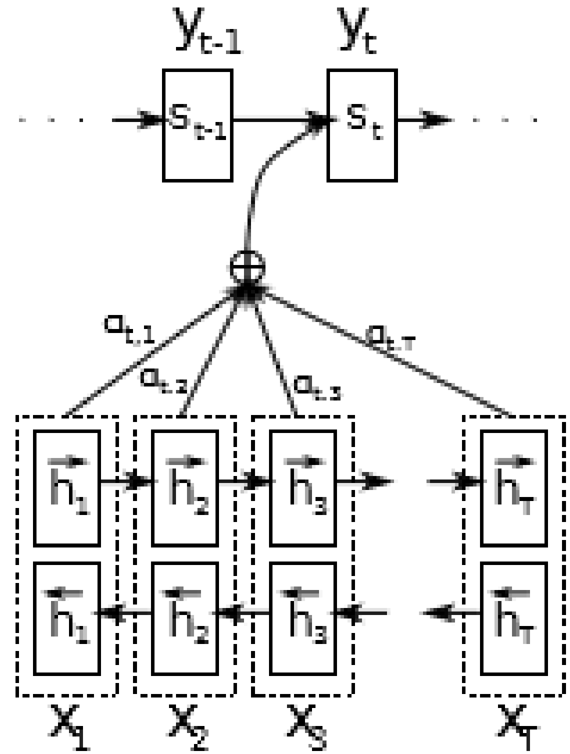


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Fig?. ??

Attention

Mechanism - Decoder

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

f : nonlinear function. Seq2seq모델에서 사용하는 LSTM 또는 GRU 등.
s_i : i번째 디코더 RNN 셀에서의 hidden state
c_i : i번째 단어를 추측하기 위해 생성된 context vector

Initial state $s_0 = tanh(s_{i-1}, y_{i-1}, c_i)$

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

$$\begin{aligned} \tilde{s}_i &= \tanh\left(W E y_{i-1} + U\left[r_i \circ s_{i-1}\right] + C c_i\right) \\ z_i &= \sigma\left(W_z E y_{i-1} + U_z s_{i-1} + C_z c_i\right) \\ r_i &= \sigma\left(W_r E y_{i-1} + U_r s_{i-1} + C_r c_i\right) \end{aligned}$$

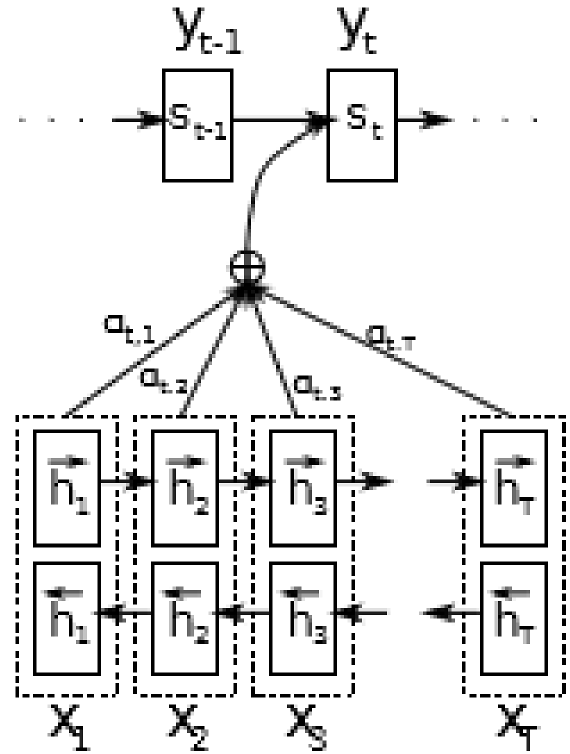


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Fig?. ??

Attention

Mechanism - Decoder

$$p(y_i|y_1, \ldots, y_{i-1}, \mathrm{x}) = g(y_{i-1}, s_i, c_i)$$

g : a nonlinear, potentially multi-layered, function that outputs the probability of y_i

$$p(y_i|s_i, y_{i-1}, c_i) \propto \exp\left(y_i^\top W_o t_i\right),$$

$$t_i = \left[\max\left\{\tilde{t}_{i,2j-1}, \tilde{t}_{i,2j}\right\}\right]_{j=1,\ldots,l}^\top$$

$$\tilde{t}_i = U_o s_{i-1} + V_o E y_{i-1} + C_o c_i.$$

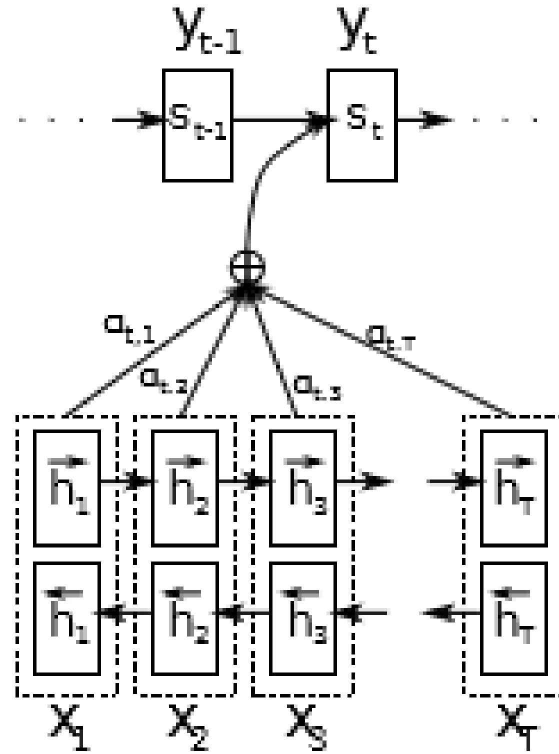


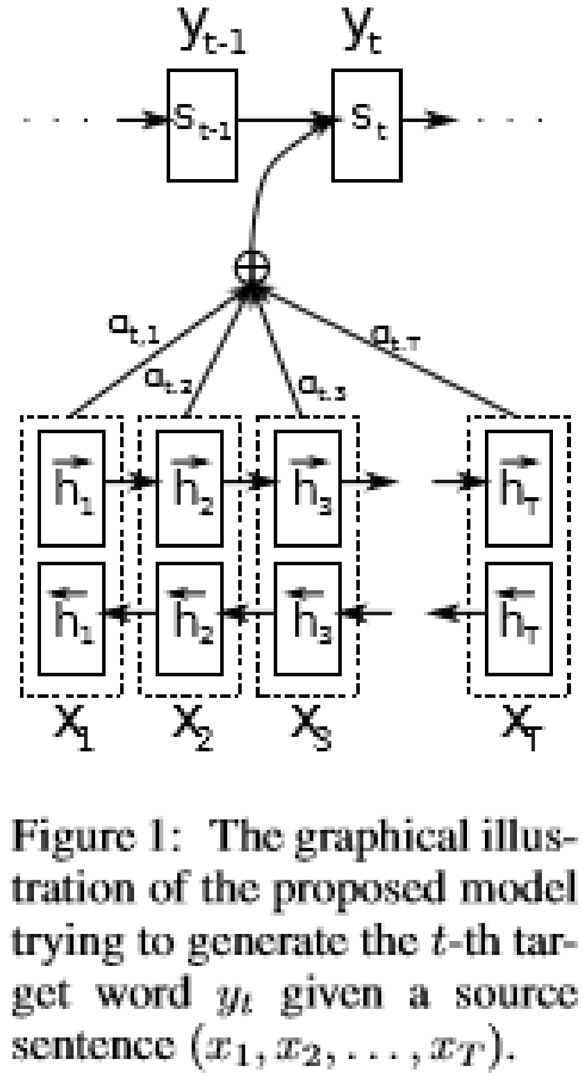
Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Fig?. ??

Attention

Mechanism - Decoder

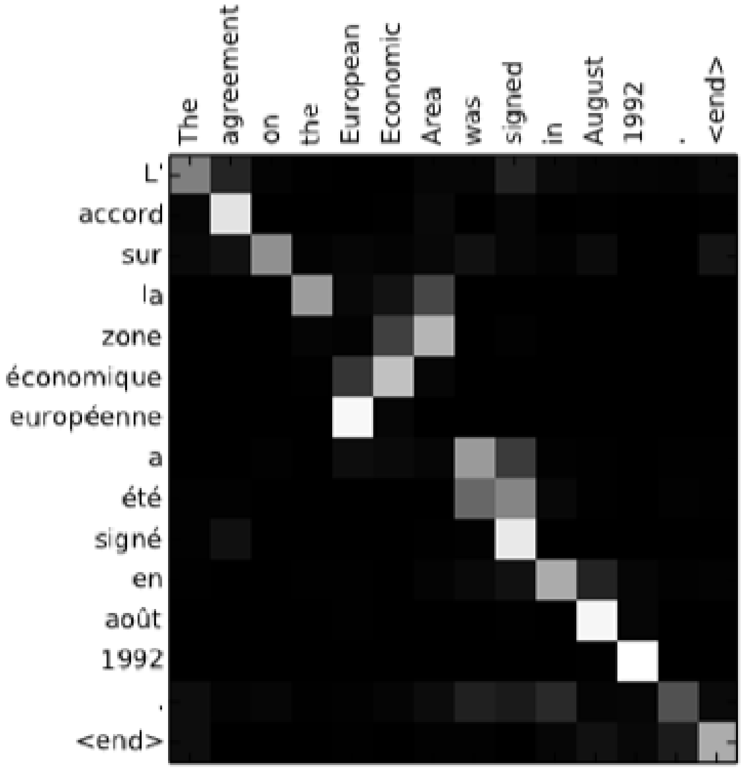
$$p(y_i|y_1, \dots, y_{i-1}, x)$$
$$= g(y_{i-1}, s_i, c_i)$$
$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$
$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{Tx} exp(e_{ik})}$$
$$e_{ij} = a(s_{i-1}, h_j)$$



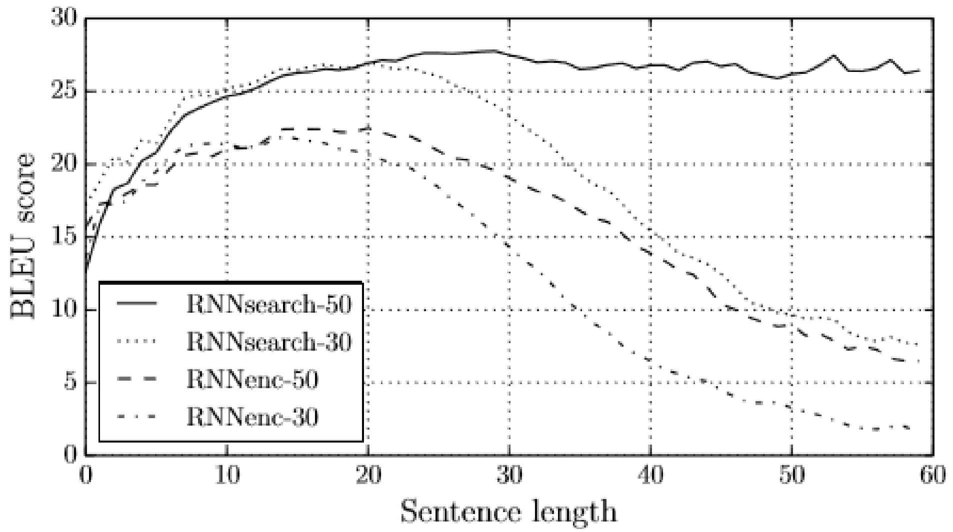
Fig?. ??

Attention

Performance



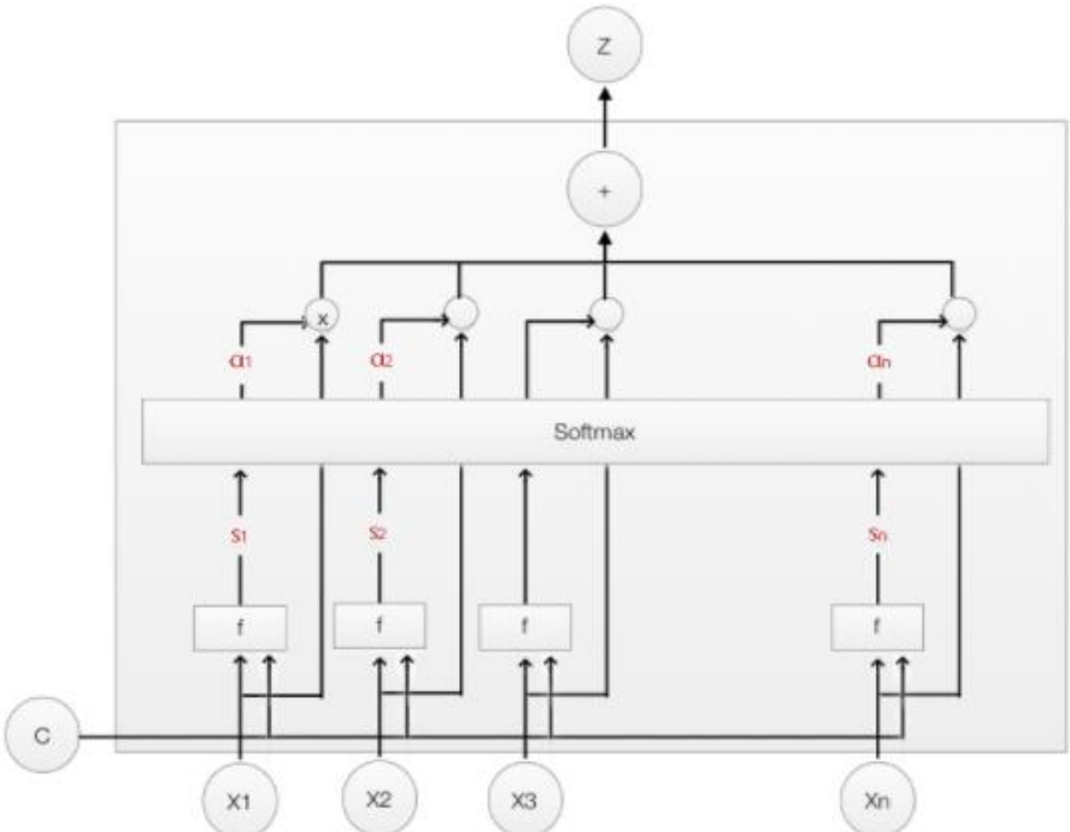
Fig?. ??



Fig?. Curve BLEU score by Sentence length

Improve

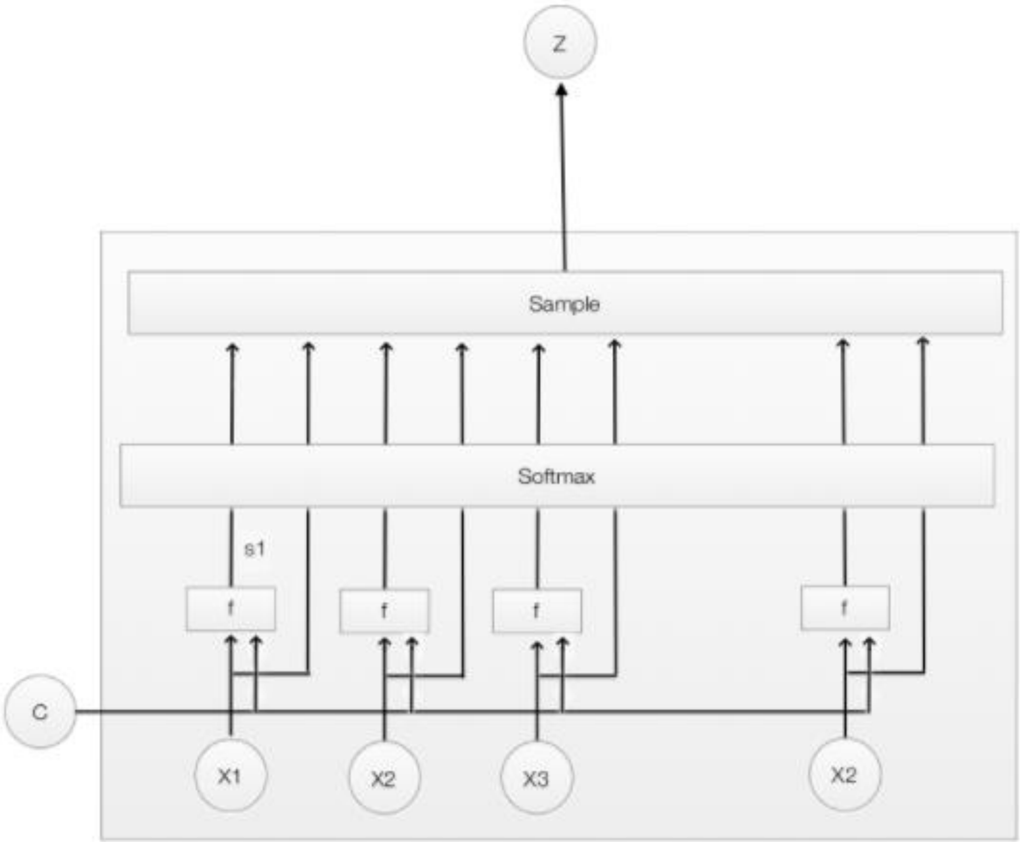
Soft & Hard Mechanism



Fig?. ??

Improve

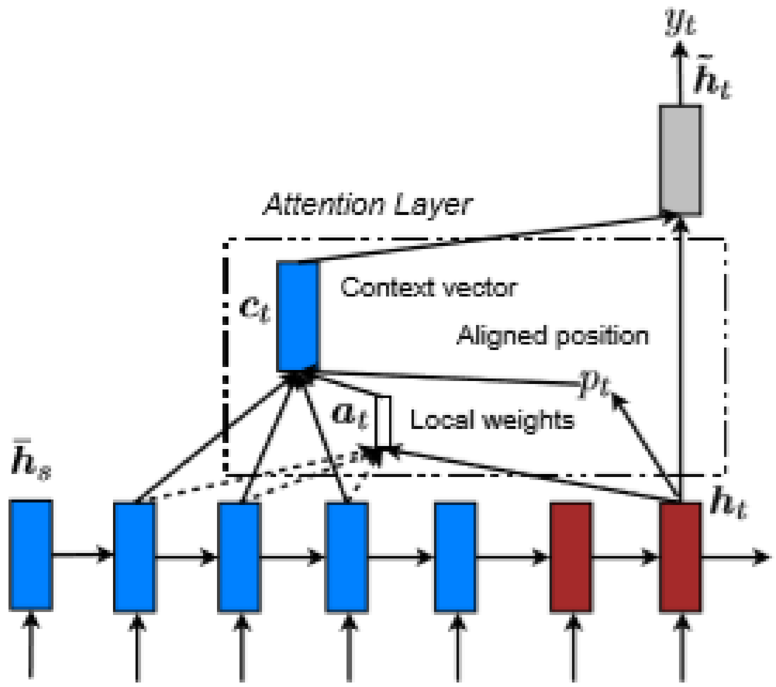
Soft & Hard Mechanism



Fig?. ??

Improve

Global & Local



Fig?. ??

Local m – monotonic local. $p_t = t$

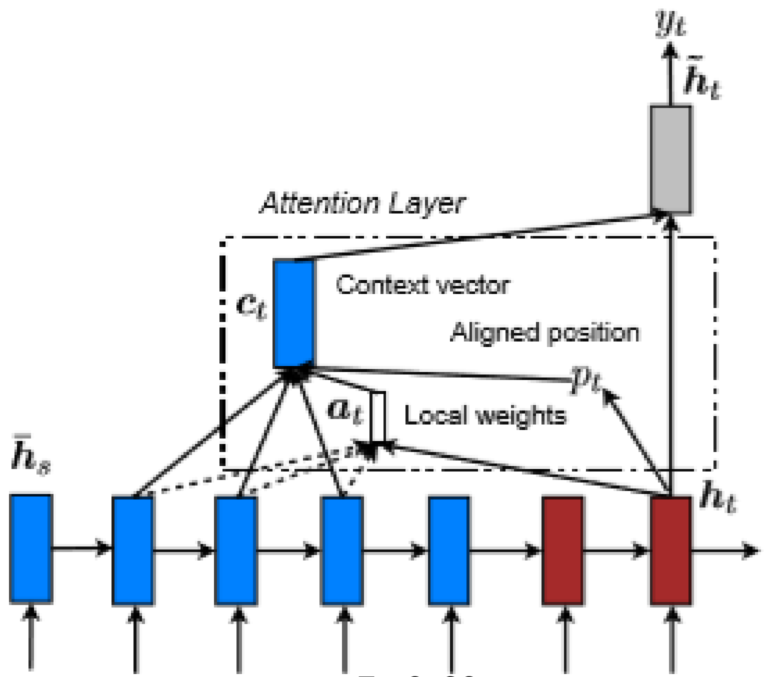
Local p – predictive local. $p_t = S \cdot sigmoid(v_p{}^T \, tanh(W_p h_t))$

align을 계산할 때 p_t를 중점으로 고정된 window size 크기로 계산 + 가우시안 분포

$$e_{ij} = a(s_{i-1}, h_j)\exp(-\frac{(s-p_t)^2}{2\sigma^2})$$

Improve

Mechanism – Local



Fig?. ??

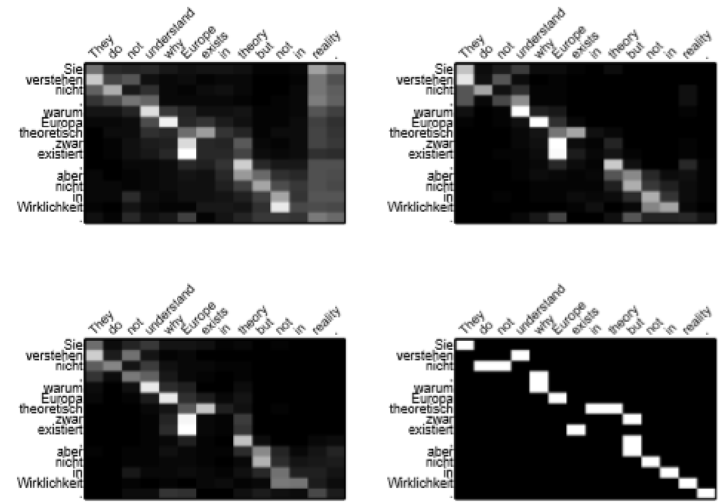


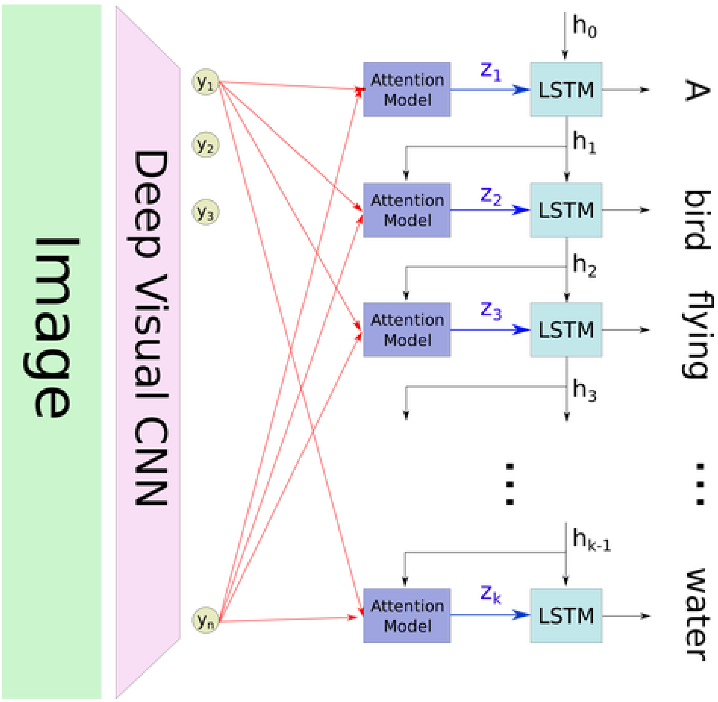
Figure 7: **Alignment visualizations** – shown are images of the attention weights learned by various models: (top left) global, (top right) local-m, and (bottom left) local-p. The *gold* alignments are displayed at the bottom right corner.

Fig?. ??

- Local attention : context 전부를 보는 (global) 계산이 오래 걸림
- Source sentence에서의 위치와 target sentence에서의 위치 를 예측할 수 있다면 계산을 줄일 수 있지 않을까?
- Local-m(monotonic) : sourc와 target의 위치가 같다고 가정, Local-p(predictive) : aligned position을 sigmoid 등으로 예측

Improve

Attention for Image Captioning

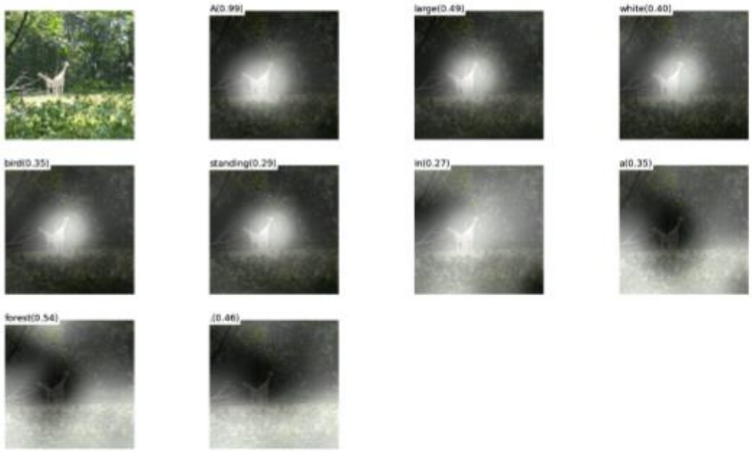


Fig?. ??

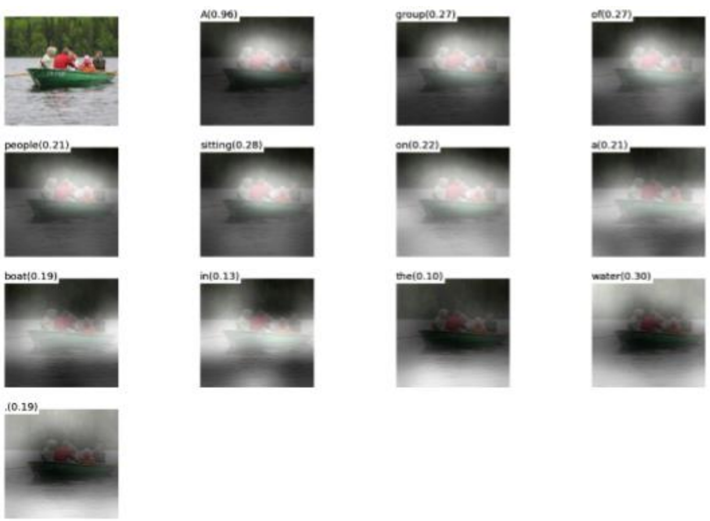
Improve

Attention for Image Captioning

(b) A woman is throwing a frisbee in a park.



(b) A large white bird standing in a forest.

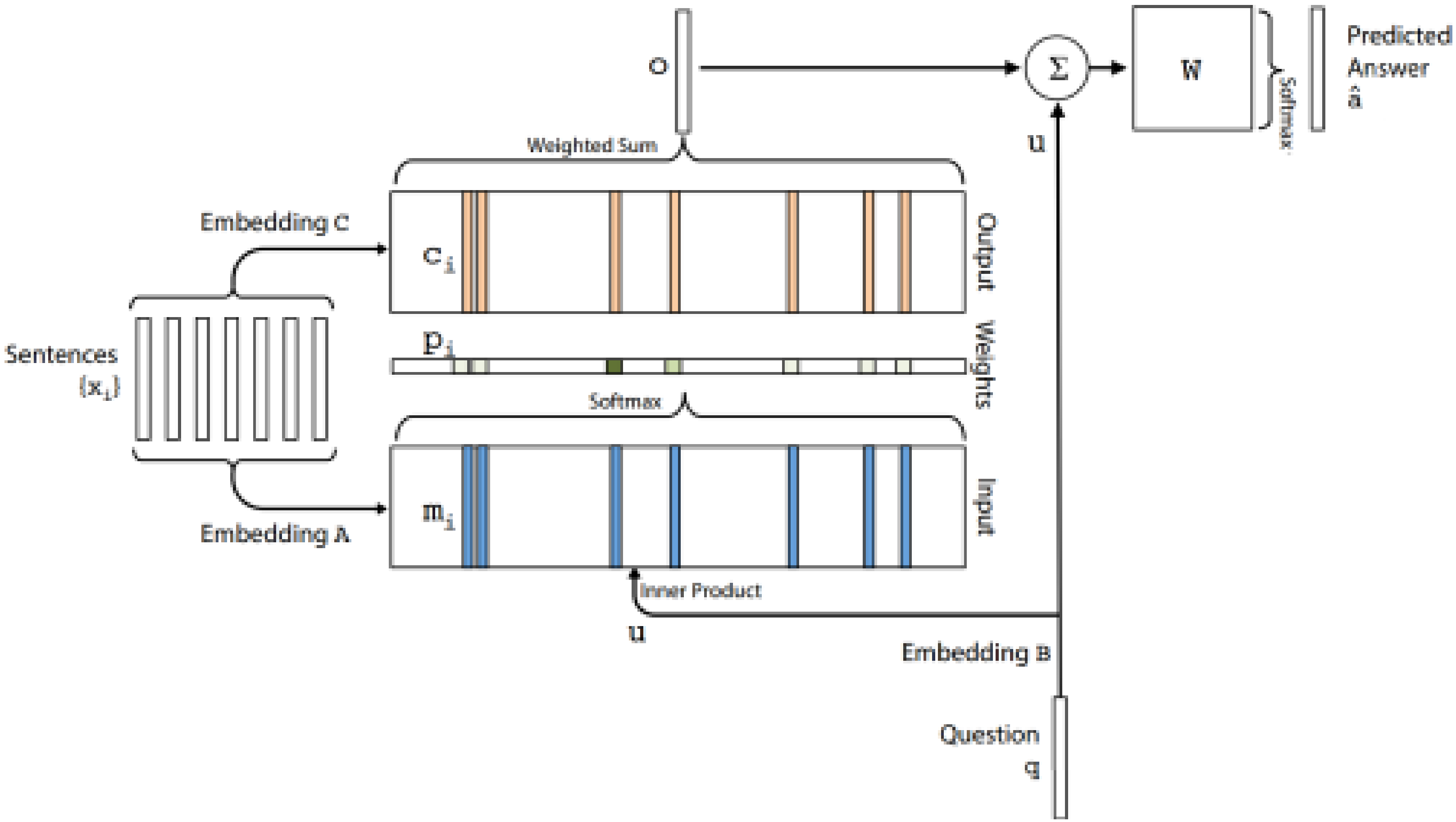


(b) A group of people sitting on a boat in the water.

(b) A woman is throwing a frisbee in a park.

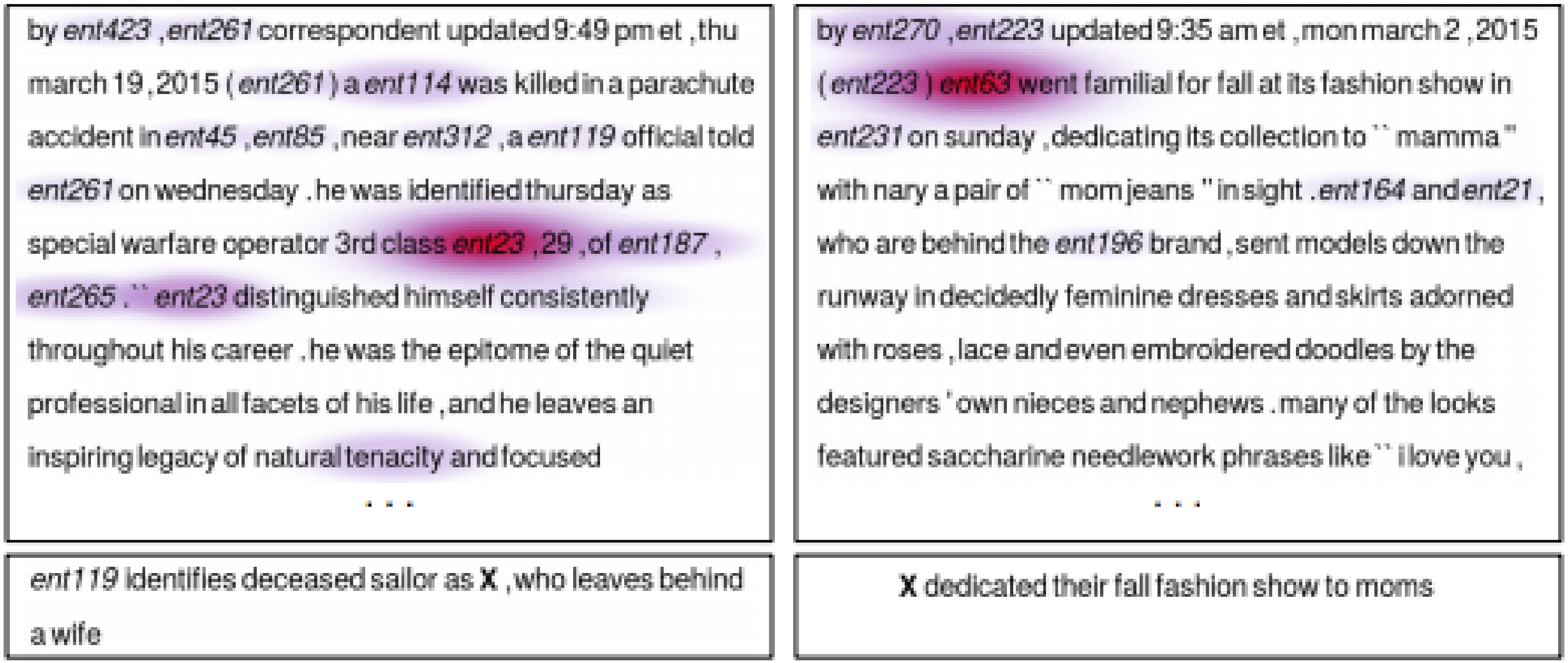
Improve

Neural Turing Machine &
Memory-based QA Models



Improve

Neural Turing Machine & Memory-based QA Models



Code

RNN model with Attention

```
# Embedding layer
with tf.name_scope('Embedding_layer'):
    embeddings_var = tf.Variable(tf.random_uniform([vocabulary_size, EMBEDDING_DIM], -1.0, 1.0), trainable=True)
    tf.summary.histogram('embeddings_var', embeddings_var)
    batch_embedded = tf.nn.embedding_lookup(embeddings_var, batch_ph)

# (Bi-)RNN layer(-s)
rnn_outputs, _ = bi_rnn(GRUCell(HIDDEN_SIZE), GRUCell(HIDDEN_SIZE),
                        inputs=batch_embedded, sequence_length=seq_len_ph, dtype=tf.float32)
tf.summary.histogram('RNN_outputs', rnn_outputs)

# Attention layer
with tf.name_scope('Attention_layer'):
    attention_output, alphas = attention(rnn_outputs, ATTENTION_SIZE, return_alphas=True)
    tf.summary.histogram('alphas', alphas)

# Dropout
drop = tf.nn.dropout(attention_output, keep_prob_ph)

# Fully connected layer
with tf.name_scope('Fully_connected_layer'):
    W = tf.Variable(tf.truncated_normal([HIDDEN_SIZE * 2, 1], stddev=0.1))  # Hidden size is multiplied by 2 for Bi-RNN
    b = tf.Variable(tf.constant(0., shape=[1]))
    y_hat = tf.nn.xw_plus_b(drop, W, b)
    y_hat = tf.squeeze(y_hat)
    tf.summary.histogram('W', W)

with tf.name_scope('Metrics'):
    # Cross-entropy loss and optimizer initialization
    loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=y_hat, labels=target_ph))
    tf.summary.scalar('loss', loss)
    optimizer = tf.train.AdamOptimizer(learning_rate=1e-3).minimize(loss)

    # Accuracy metric
    accuracy = tf.reduce_mean(tf.cast(tf.equal(tf.round(tf.sigmoid(y_hat)), target_ph), tf.float32))
    tf.summary.scalar('accuracy', accuracy)
```

Code

RNN model with Attention

```
# Embedding layer
with tf.name_scope('Embedding_layer'):
    embeddings_var = tf.Variable(tf.random_uniform([vocabulary_size, EMBEDDING_DIM], -1.0, 1.0), trainable=True)
    tf.summary.histogram('embeddings_var', embeddings_var)
    batch_embedded = tf.nn.embedding_lookup(embeddings_var, batch_ph)

# (Bi-)RNN layer(-s)
rnn_outputs, _ = bi_rnn(GRUCell(HIDDEN_SIZE), GRUCell(HIDDEN_SIZE),
                        inputs=batch_embedded, sequence_length=seq_len_ph, dtype=tf.float32)
tf.summary.histogram('RNN_outputs', rnn_outputs)

# Attention layer
with tf.name_scope('Attention_layer'):
    attention_output, alphas = attention(rnn_outputs, ATTENTION_SIZE, return_alphas=True)
    tf.summary.histogram('alphas', alphas)
```

- Batch size만큼 Look up table을 보고 Word들의 값에 따라 각각을 벡터 값으로 바꿈
- 바꾼 batch_embedded를 Rnn Layer의 input으로 넣고 Rnn Layer의 output을 Attention Layer의 input으로 넣음.

Code

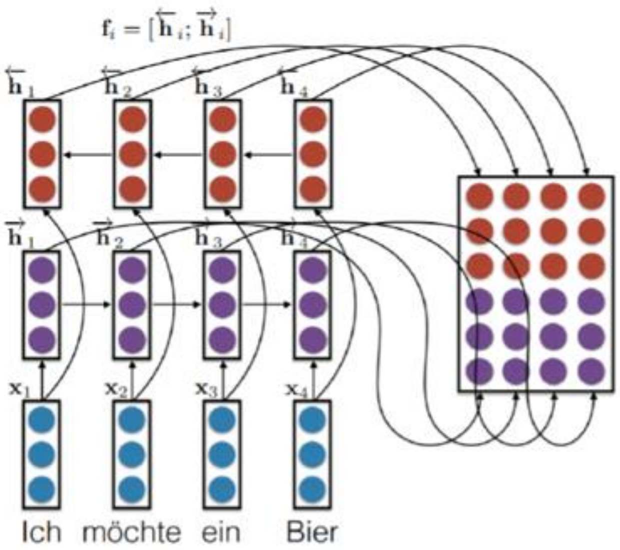
RNN model with Attention

```python
if isinstance(inputs, tuple):
    # In case of Bi-RNN, concatenate the forward and the backward RNN outputs.
    inputs = tf.concat(inputs, 2)

if time_major:
    # (T,B,D) => (B,T,D)
    inputs = tf.array_ops.transpose(inputs, [1, 0, 2])

hidden_size = inputs.shape[2].value  # D value - hidden size of the RNN layer

# 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)

if not return_alphas:
    return output
else:
    return output, alphas
```

Code

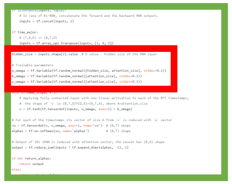
RNN model with Attention

```python
if isinstance(inputs, tuple):
    # In case of Bi-RNN, concatenate the forward and the backward RNN outputs.
    inputs = tf.concat(inputs, 2)
```



Forward 와 backward RNN output을 concat함.

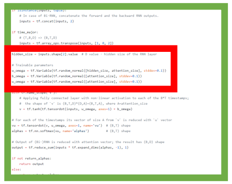
Code



RNN model with Attention

```
if isinstance(inputs, tuple):
    # In case of Bi-RNN, concatenate the forward and the backward RNN outputs.
    inputs = tf.concat(inputs, 2)

if time_major:
    # (T,B,D) => (B,T,D)
    inputs = tf.array_ops.transpose(inputs, [1, 0, 2])

hidden_size = inputs.shape[2].value  # D value - hidden size of the RNN layer

# 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)

if not return_alphas:
    return output
else:
    return output, alphas
```

Code



RNN model with Attention

```
hidden_size = inputs.shape[2].value  # D value - hidden size of the RNN layer

# 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))
```

- Hidden_size는 input의 shape가 (batch_size, Max_time, cell.output_size) 이므로 cell.output_size = cell_fw.output_size + cell_bw.output_size이다.
- W, B, U는 학습 파라미터

Code

RNN model with Attention

```python
if isinstance(inputs, tuple):
    # In case of Bi-RNN, concatenate the forward and the backward RNN outputs.
    inputs = tf.concat(inputs, 2)

if time_major:
    # (T,B,D) => (B,T,D)
    inputs = tf.array_ops.transpose(inputs, [1, 0, 2])

hidden_size = inputs.shape[2].value  # D value - hidden size of the RNN layer

# 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)

if not return_alphas:
    return output
else:
    return output, alphas
```

Code

RNN model with Attention

```python
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)
```

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

- Seq2seq모델에 attention모델을 적용한 것이 아니고 many to one의 Rnn 모델에 attention을 적용한 것이므로 Si-1가 없고, 따라서 tanh(w*hj + b)라고 생각.

Code

RNN model with Attention

```python
if isinstance(inputs, tuple):
    # In case of Bi-RNN, concatenate the forward and the backward RNN outputs.
    inputs = tf.concat(inputs, 2)

if time_major:
    # (T,B,D) => (B,T,D)
    inputs = tf.array_ops.transpose(inputs, [1, 0, 2])

hidden_size = inputs.shape[2].value  # D value - hidden size of the RNN layer

# 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)

if not return_alphas:
    return output
else:
    return output, alphas
```

Code

RNN model with Attention

```
# 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
```

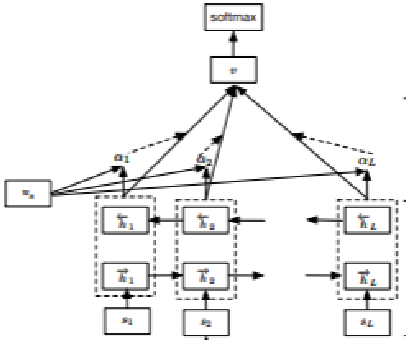
$$a(s_{i-1}, h_j) = v_a^T \tanh(W_a S_{i-1} + U_a h_j)$$

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

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

- 이전에 구한 v값과 u를 곱한후 softmax함수를 취하는 부분.

```
# 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)
```



- 위와 같은 구조로 매번 context vector를 구하지 않으므로

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$ 가 아닌 c의 값을 weighted sum하는 부분.

Code

RNN model with Attention

```python
# Embedding layer
with tf.name_scope('Embedding_layer'):
    embeddings_var = tf.Variable(tf.random_uniform([vocabulary_size, EMBEDDING_DIM], -1.0, 1.0), trainable=True)
    tf.summary.histogram('embeddings_var', embeddings_var)
    batch_embedded = tf.nn.embedding_lookup(embeddings_var, batch_ph)

# (Bi-)RNN layer(-s)
rnn_outputs, _ = bi_rnn(GRUCell(HIDDEN_SIZE), GRUCell(HIDDEN_SIZE),
                        inputs=batch_embedded, sequence_length=seq_len_ph, dtype=tf.float32)
tf.summary.histogram('RNN_outputs', rnn_outputs)

# Attention layer
with tf.name_scope('Attention_layer'):
    attention_output, alphas = attention(rnn_outputs, ATTENTION_SIZE, return_alphas=True)
    tf.summary.histogram('alphas', alphas)

# Dropout
drop = tf.nn.dropout(attention_output, keep_prob_ph)

# Fully connected layer
with tf.name_scope('Fully_connected_layer'):
    W = tf.Variable(tf.truncated_normal([HIDDEN_SIZE * 2, 1], stddev=0.1))  # Hidden size is multiplied by 2 for Bi-RNN
    b = tf.Variable(tf.constant(0., shape=[1]))
    y_hat = tf.nn.xw_plus_b(drop, W, b)
    y_hat = tf.squeeze(y_hat)
    tf.summary.histogram('W', W)

with tf.name_scope('Metrics'):
    # Cross-entropy loss and optimizer initialization
    loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=y_hat, labels=target_ph))
    tf.summary.scalar('loss', loss)
    optimizer = tf.train.AdamOptimizer(learning_rate=1e-3).minimize(loss)

    # Accuracy metric
    accuracy = tf.reduce_mean(tf.cast(tf.equal(tf.round(tf.sigmoid(y_hat)), target_ph), tf.float32))
    tf.summary.scalar('accuracy', accuracy)
```

Code

RNN model with Attention

```
# Fully connected layer
with tf.name_scope('Fully_connected_layer'):
    W = tf.Variable(tf.truncated_normal([HIDDEN_SIZE * 2, 1], stddev=0.1))  # Hidden size is multiplied by 2 for Bi-RNN
    b = tf.Variable(tf.constant(0., shape=[1]))
    y_hat = tf.nn.xw_plus_b(drop, W, b)
    y_hat = tf.squeeze(y_hat)
    tf.summary.histogram('W', W)
```

- 예측한 결과 값이 y_hat 이됨.

```
with tf.name_scope('Metrics'):
    # Cross-entropy loss and optimizer initialization
    loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=y_hat, labels=target_ph))
    tf.summary.scalar('loss', loss)
    optimizer = tf.train.AdamOptimizer(learning_rate=1e-3).minimize(loss)

    # Accuracy metric
    accuracy = tf.reduce_mean(tf.cast(tf.equal(tf.round(tf.sigmoid(y_hat)), target_ph), tf.float32))
    tf.summary.scalar('accuracy', accuracy)
```

- y_hat과 target_ph를 비교하여 loss값을 계산하고 accuracy를 계산.

QNA

감사합니다.