DL Seminar

Attention Mechanism

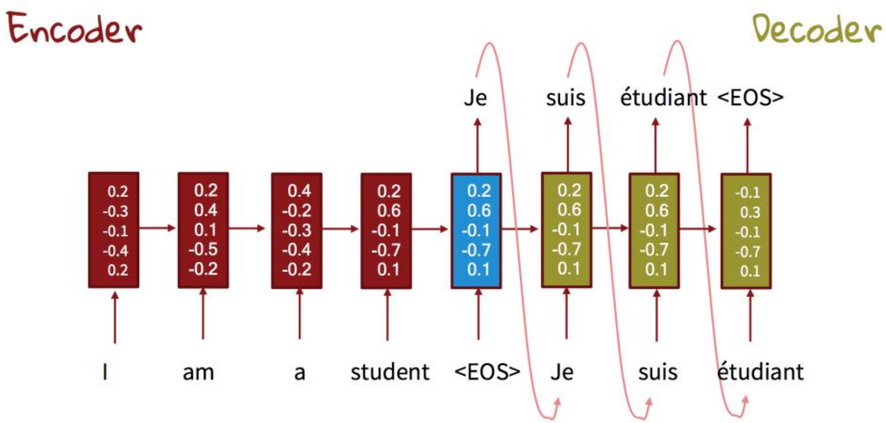
한양대학교
HANYANG UNIVERSITY

인공지능 Lab 김지성
인공지능 Lab 엄희송
인공지능 Lab 유재창

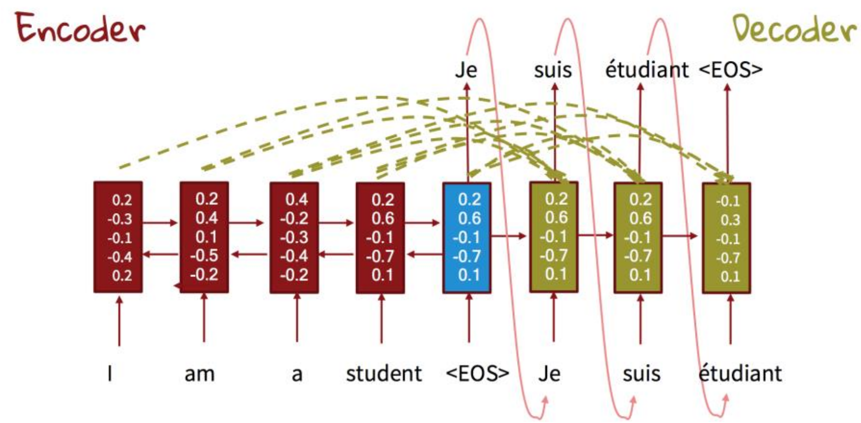
Index

Introduction

Seq2Seq



Sec2Sec Model



Se2Sec with BiRNN enc

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

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

NLP

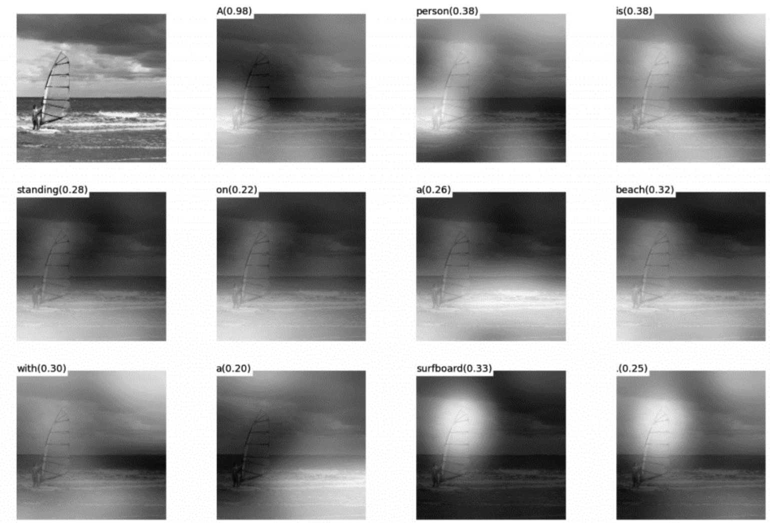


Image captioning

Attention

Mechanism - Encoder

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

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

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.

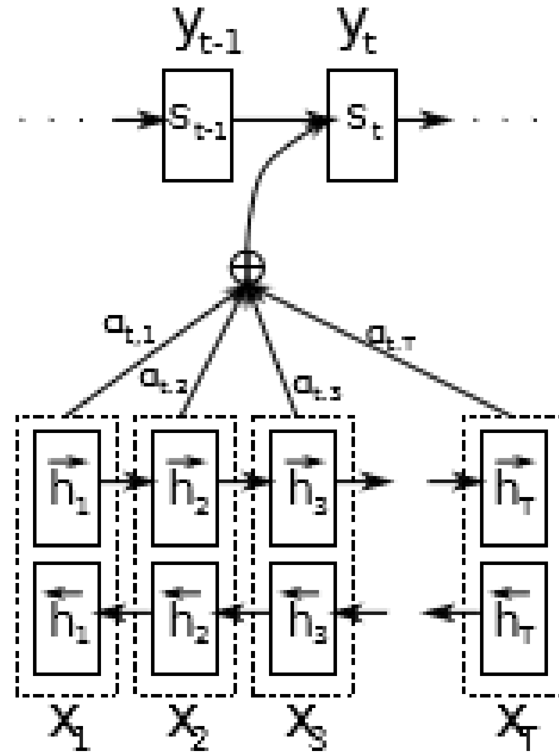


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \dots, x_T)$.

Enc-dec with attention

Attention

Mechanism - Decoder

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

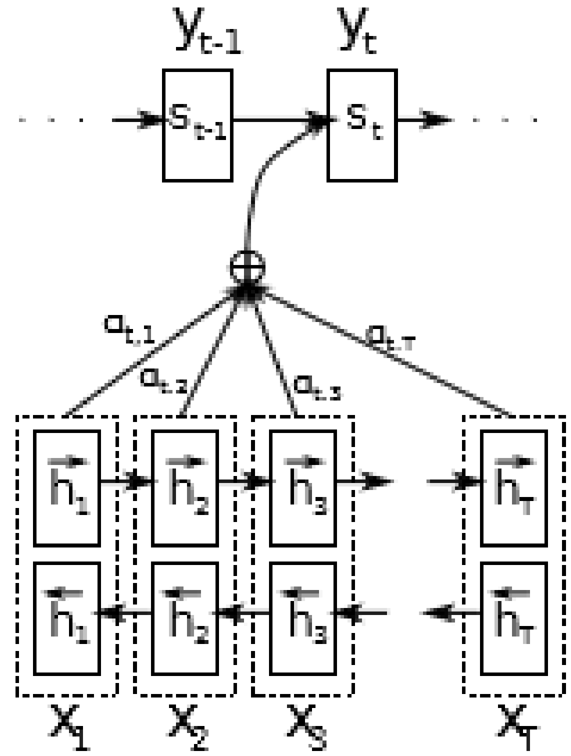


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Enc-dec with attention

Attention

Mechanism - Decoder

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

유사도를 도출할 수 있는 모델이면 모두 사용 가능

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

Initial state $s_0 = tanh(W_s \overleftarrow{h_1})$

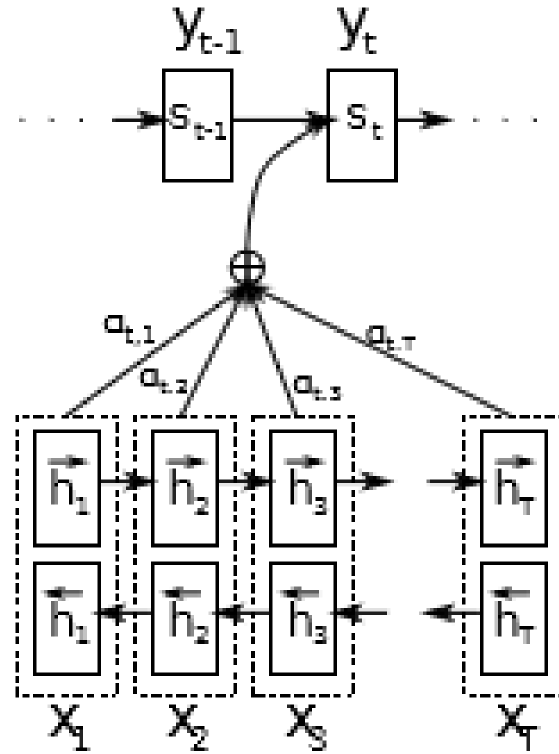


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Enc-dec with attention

Attention

Mechanism - Decoder

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

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

α_{ij} : softmax를 통해 확률값으로 보정해줌

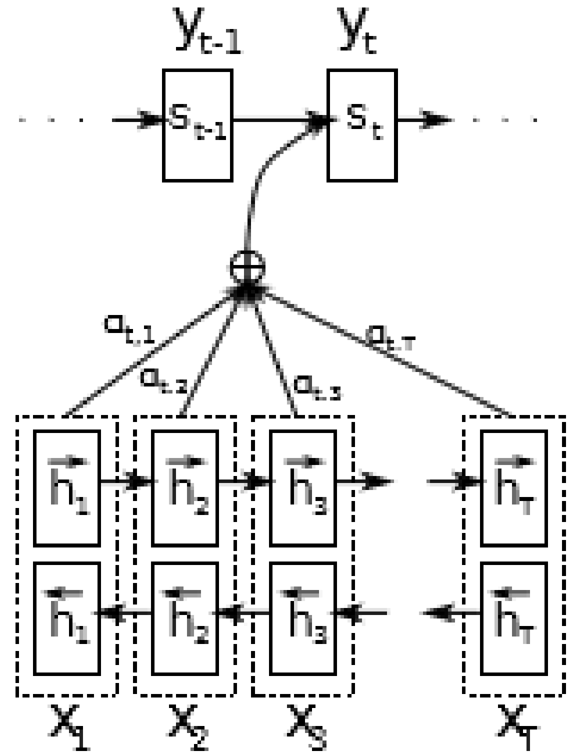


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Enc-dec with attention

Attention

Mechanism - Decoder

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

c_i : i번째 단어를 추측하기 위해 생성된 context vector

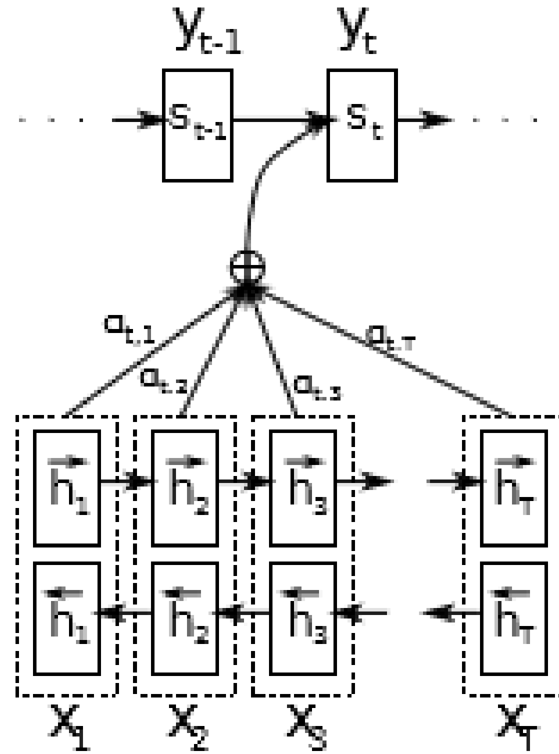


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Enc-dec with attention

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

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

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

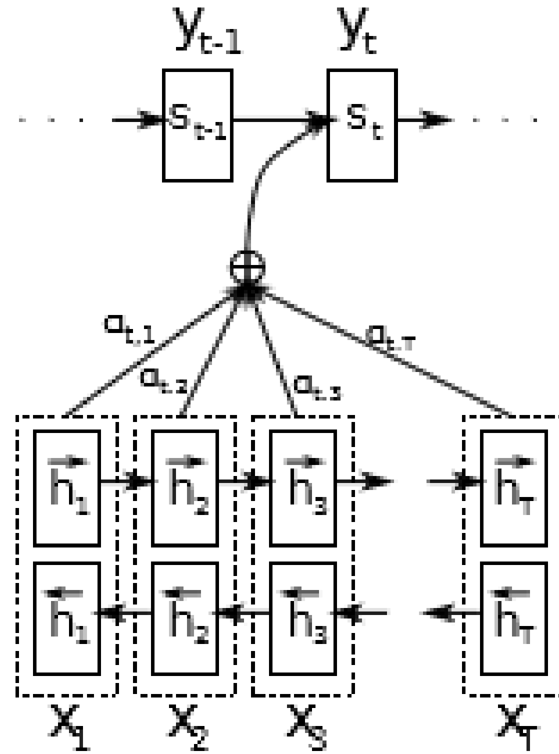


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Enc-dec with attention

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

How to Construct Deep Recurrent Neural Networks (Pascanu et al., 2014)

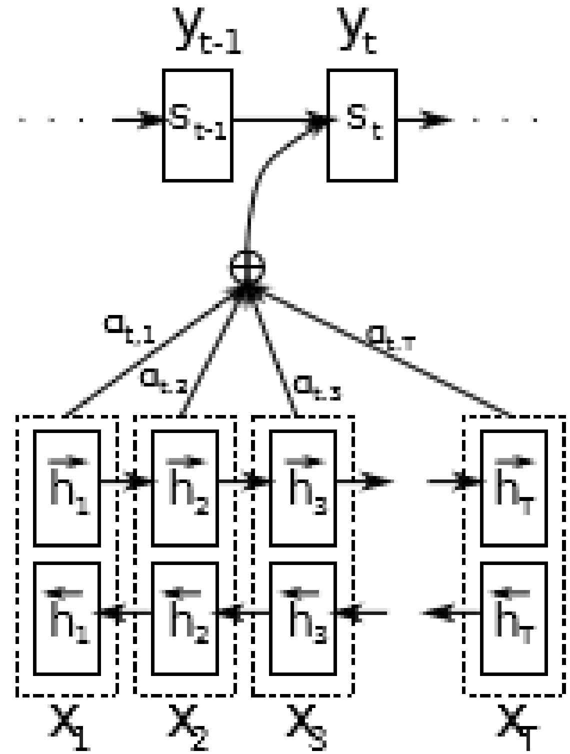


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

Enc-dec with attention

Attention

Mechanism - Decoder

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

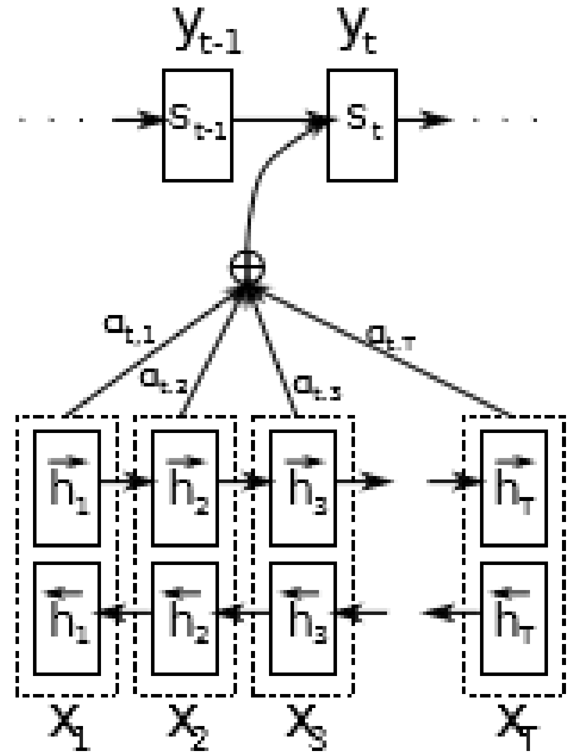
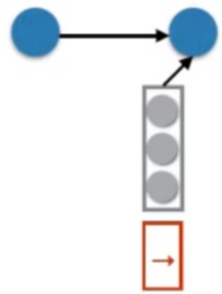


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \dots, x_T)$.

Enc-dec with attention

Attention

Calculate Flow

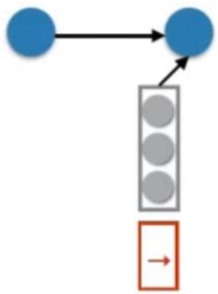


Initial state $s_0 = tanh(W_s\overleftarrow{h_1})$

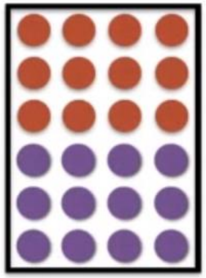
$$p(y_i|y_1,\dots,y_{i-1},x) = g(y_{i-1},s_i,c_i)$$
$$s_i = f(s_{i-1},y_{i-1},c_i)$$
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij}h_j$$
$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{Tx} exp(e_{ik})}$$
$$e_{ij} = a(s_{i-1},h_j)$$

Attention

Calculate Flow



Initial state $s_0 = tanh(W_s \overleftarrow{h_1})$

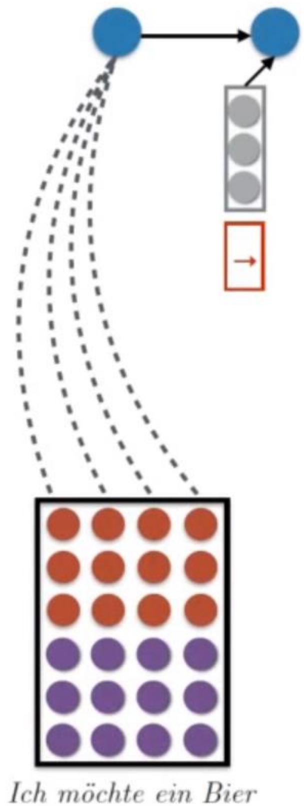


Ich möchte ein Bier

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

Attention

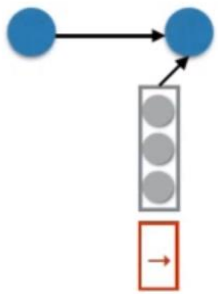
Calculate Flow



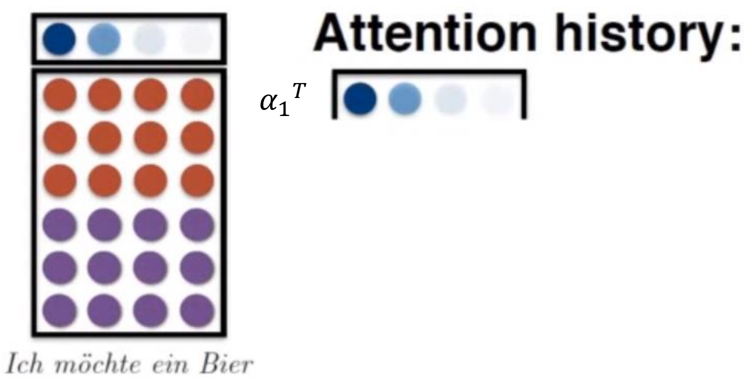
Initial state $s_0 = tanh(W_s \overleftarrow{h_1})$

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

Calculate Flow



Initial state $s_0 = tanh(W_s\overleftarrow{h_1})$



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

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

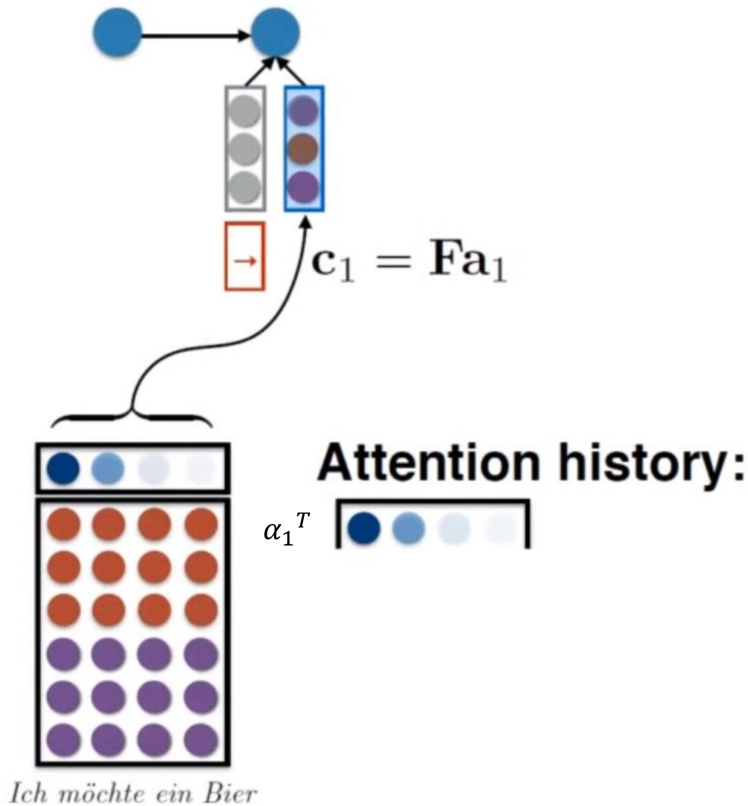
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

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

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

Attention

Calculate Flow

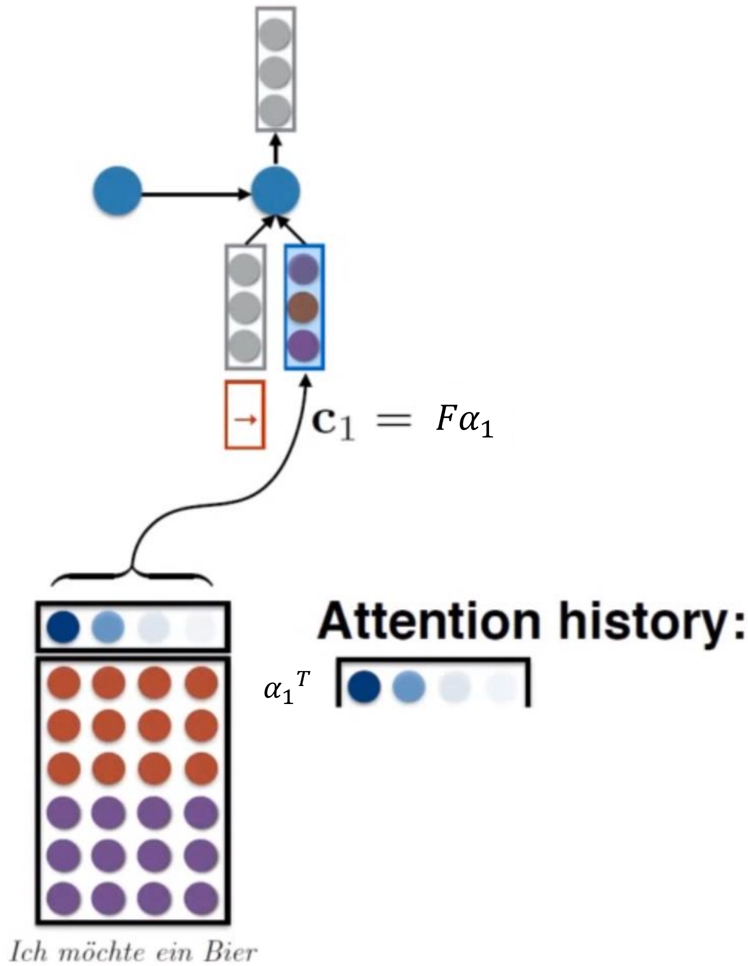


Initial state $s_0 = tanh(W_s \overleftarrow{h_1})$

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

Attention

Calculate Flow



Initial state $s_0 = tanh(W_s \overleftarrow{h_1})$

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

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

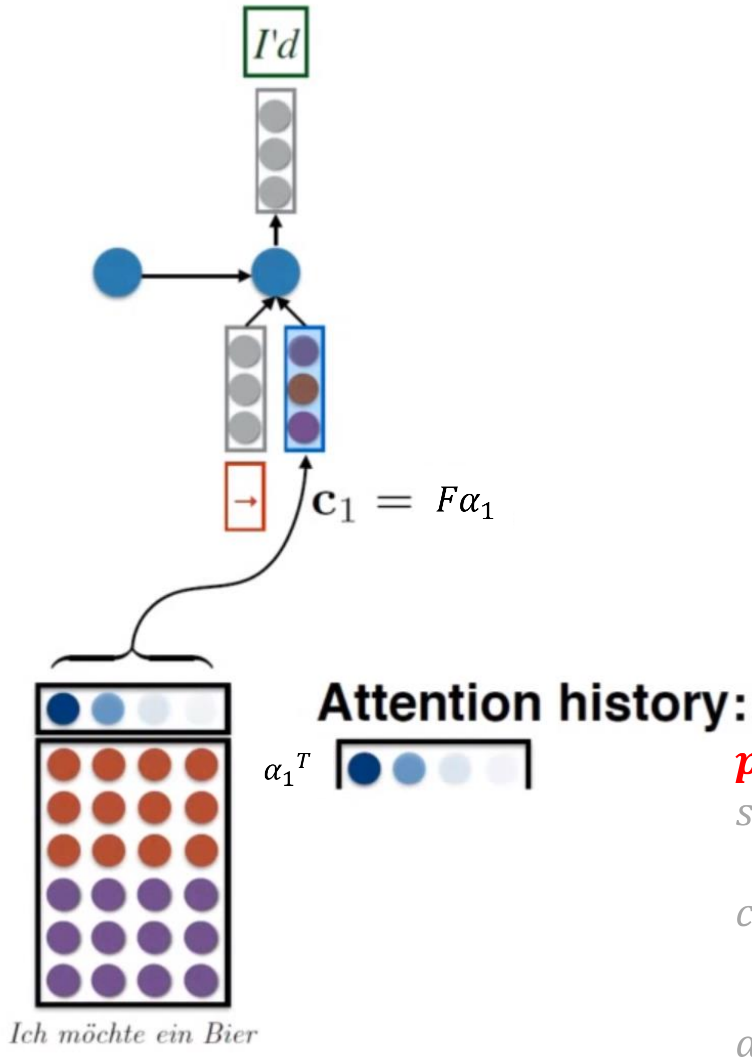
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

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

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

Attention

Calculate Flow



We got s_1 and y_1

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

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

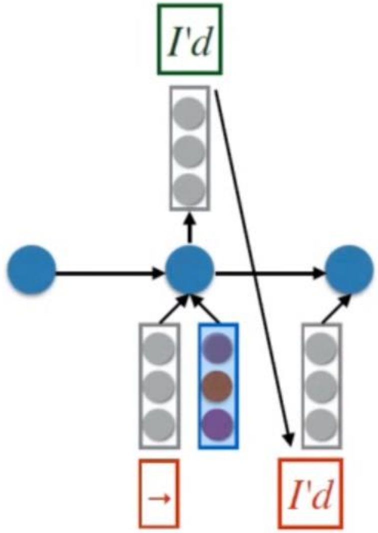
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij}h_j$$

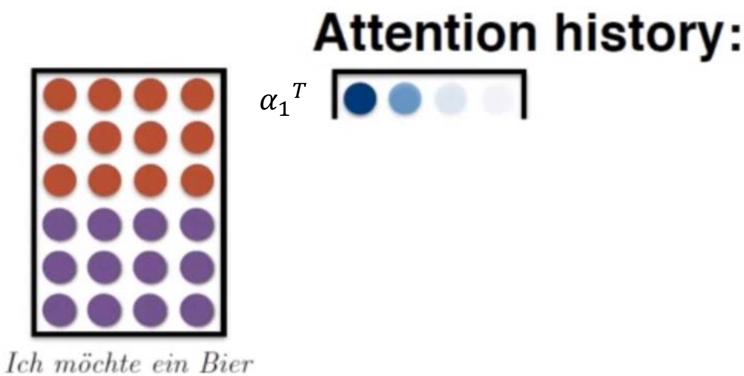
$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{Tx} exp(e_{ik})}$$

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

Attention

Calculate Flow





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

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

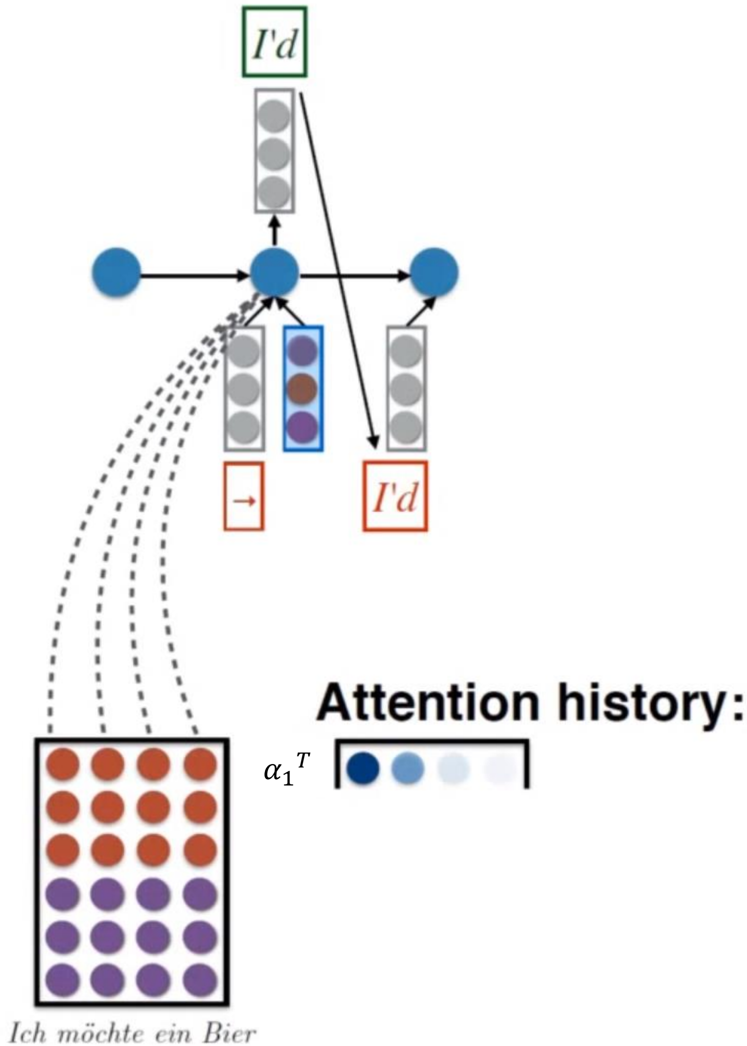
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

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

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

Attention

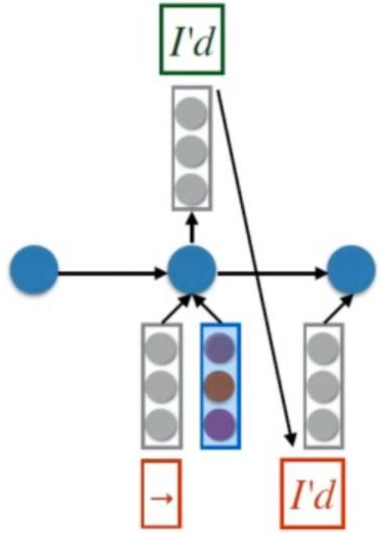
Calculate Flow

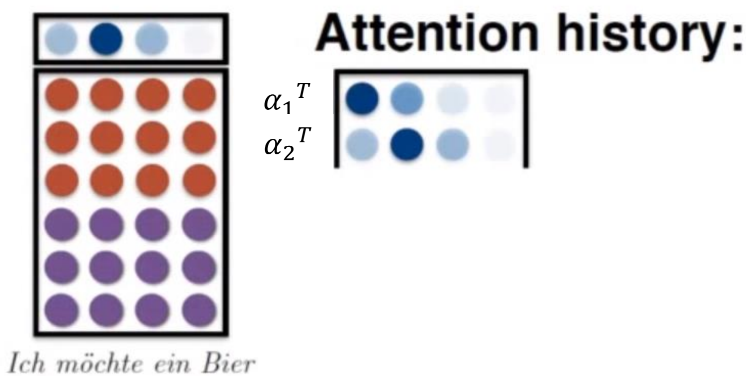


$$p(y_i|y_1, \dots, y_{i-1}, x) = g(y_{i-1}, s_i, c_i)$$
$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$
$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{Tx} exp(e_{ik})}$$
$$e_{ij} = a(s_{i-1}, h_j)$$

Attention

Calculate Flow





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

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

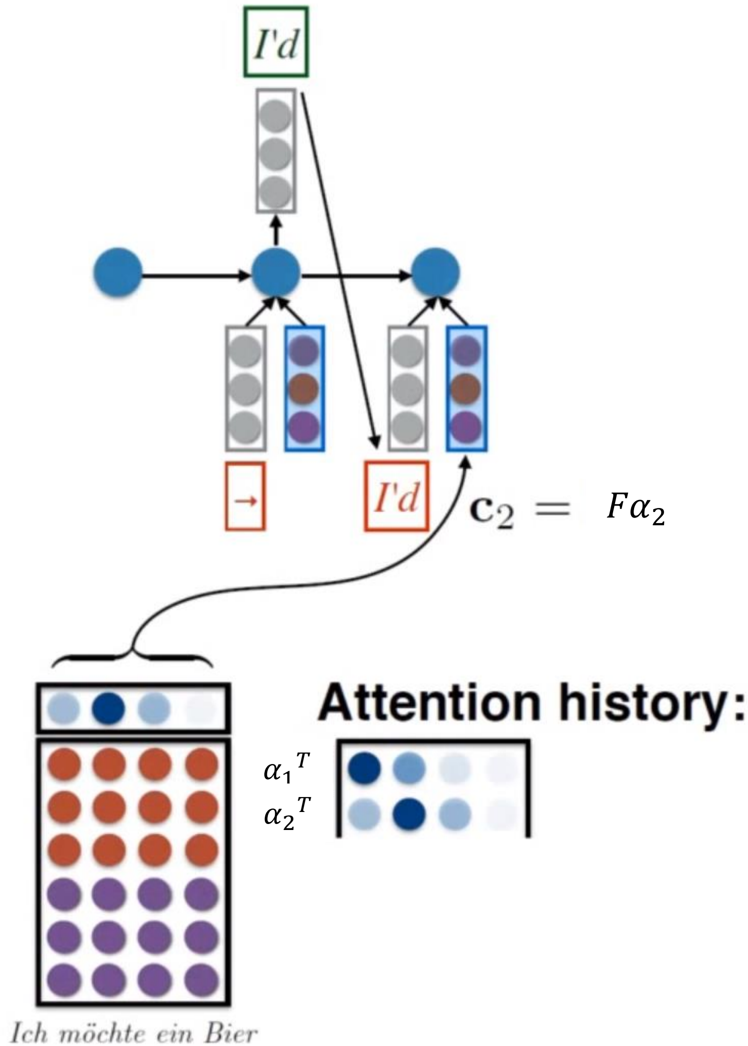
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

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

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

Attention

Calculate Flow



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

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

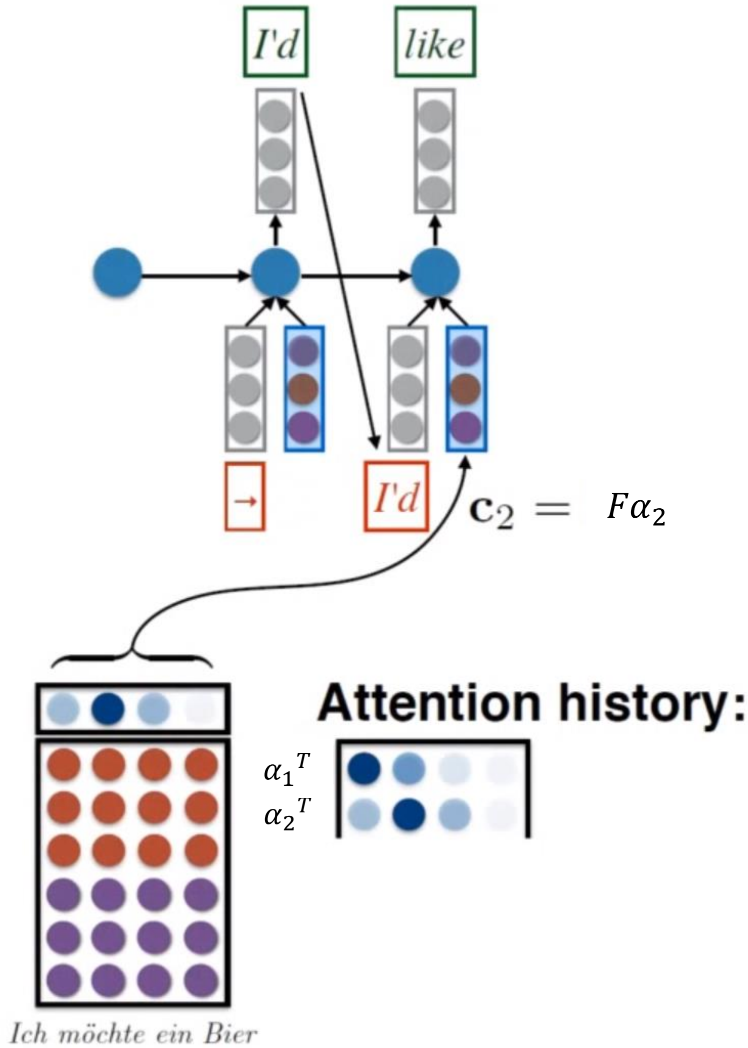
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

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

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

Attention

Calculate Flow



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

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

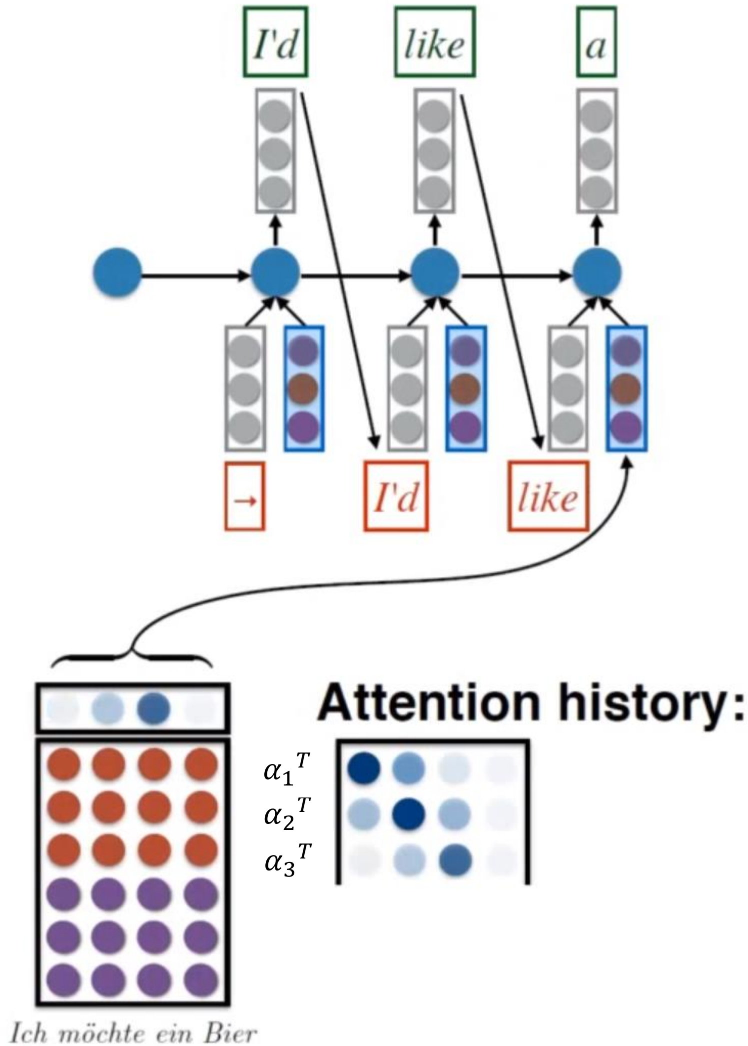
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

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

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

Attention

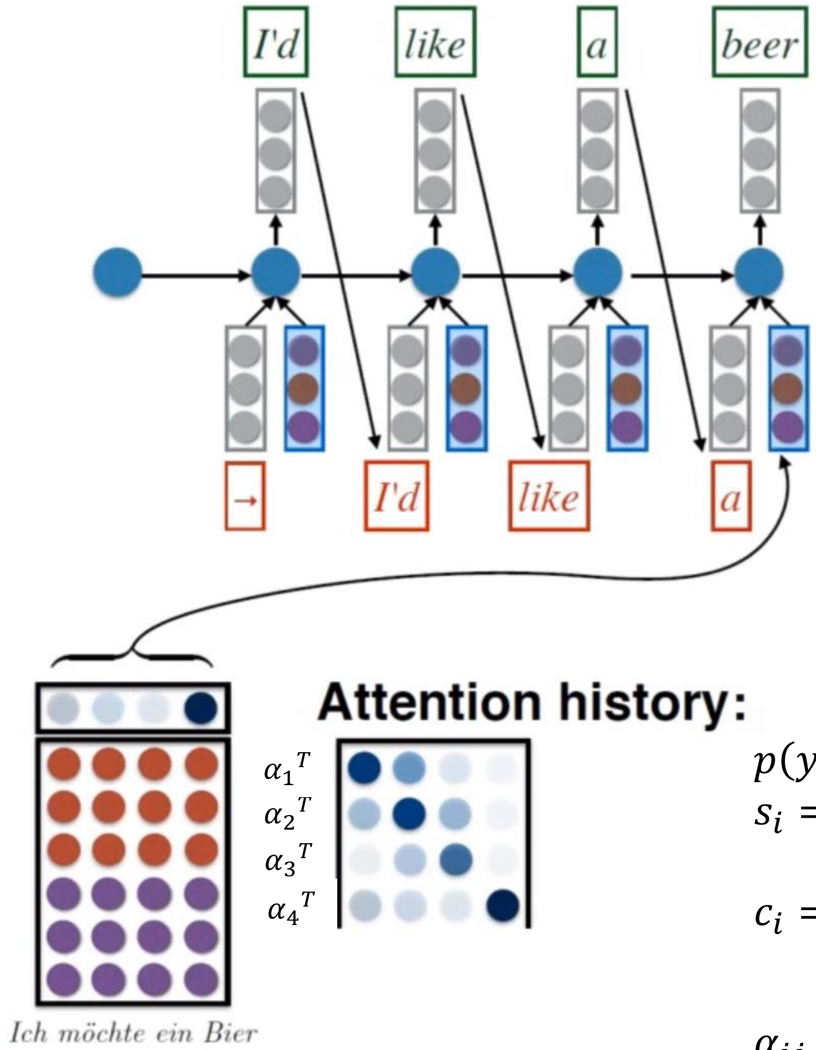
Calculate Flow



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

Attention

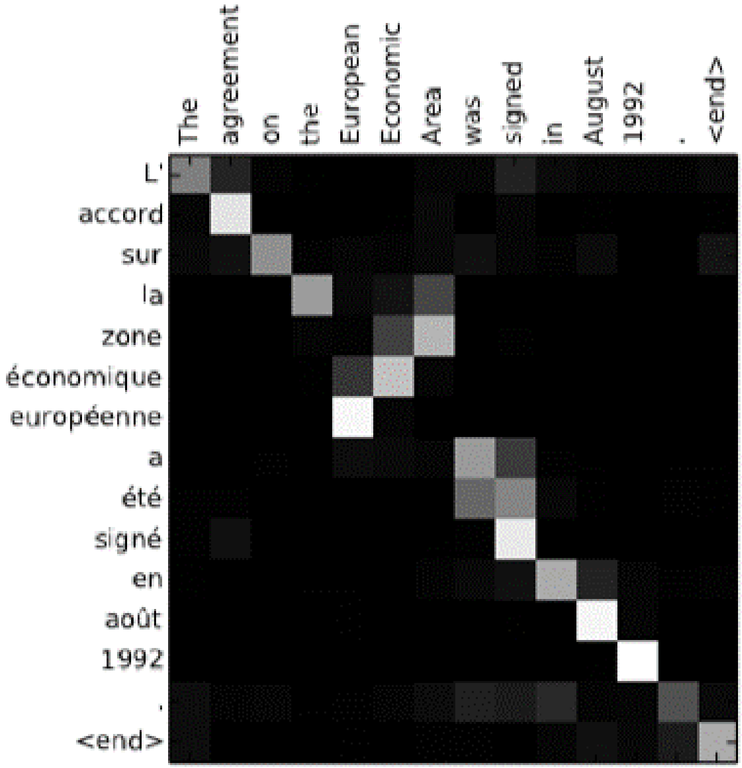
Calculate Flow



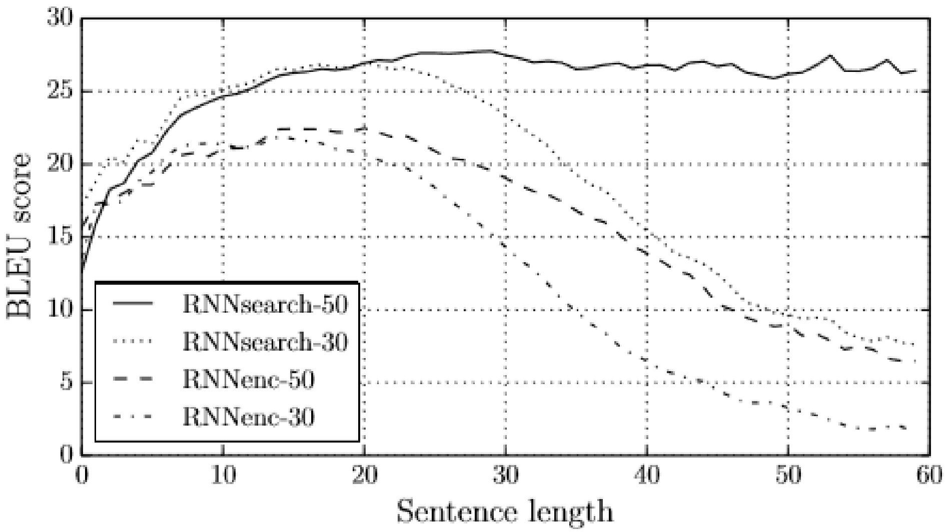
$$p(y_i|y_1, \ldots, y_{i-1}, x) = g(y_{i-1}, s_i, c_i)$$
$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$
$$\alpha_{ij} = \frac{exp(e_{ij})}{\sum_{k=1}^{Tx} exp(e_{ik})}$$
$$e_{ij} = a(s_{i-1}, h_j)$$

Attention

Performance



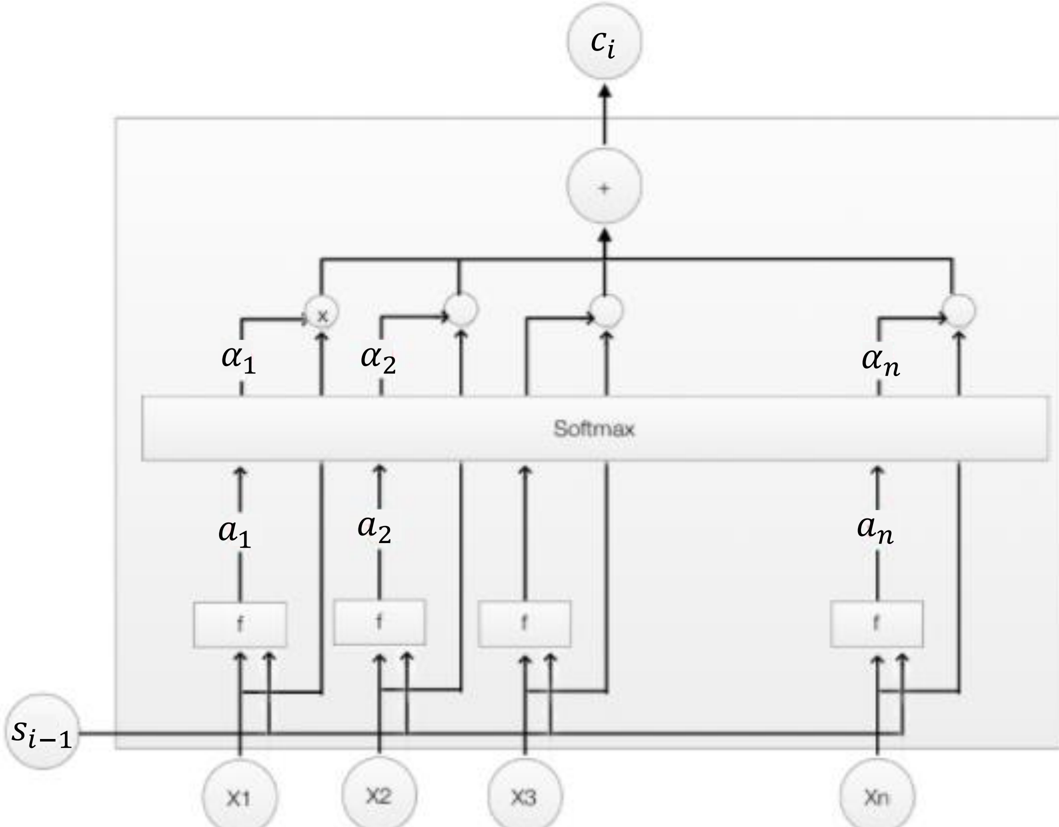
Attention visualization



Curve BLEU score by Sentence length

Improve

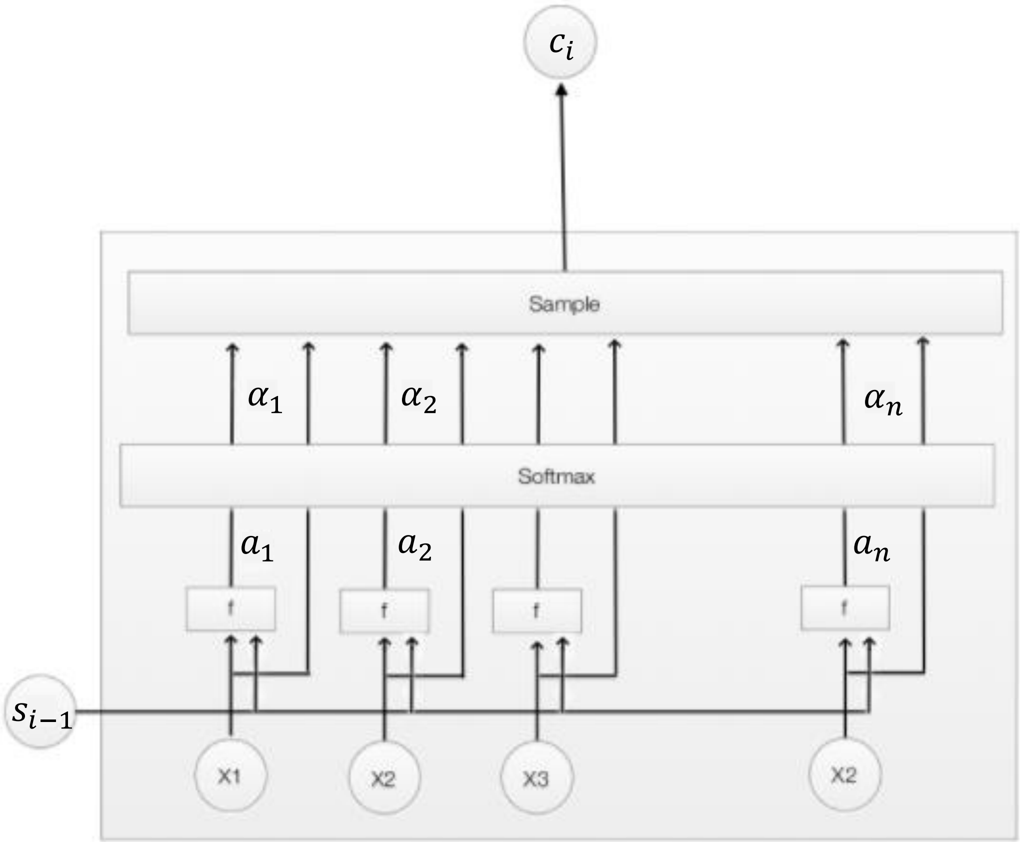
Soft & Hard Mechanism



Soft attention

Improve

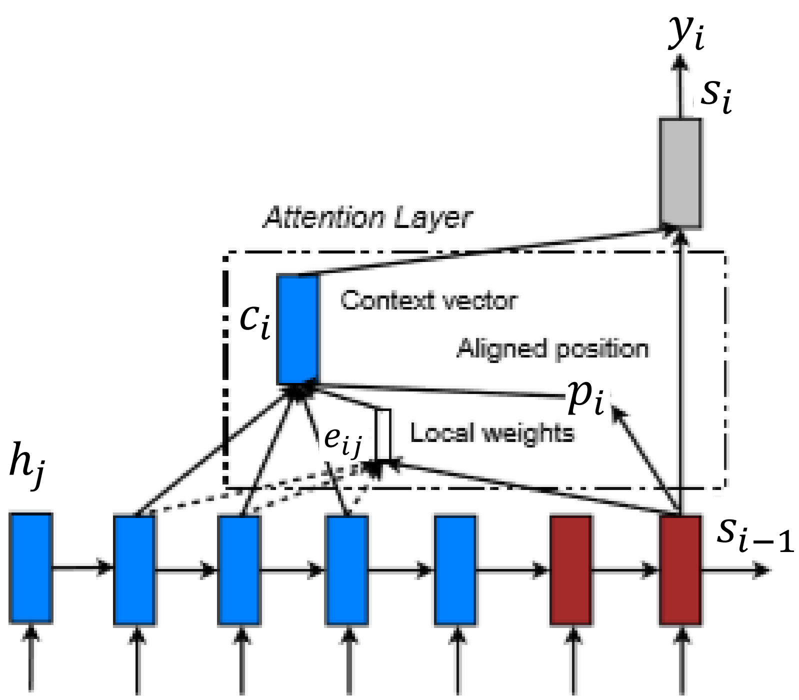
Soft & Hard Mechanism



Hard attention

Improve

Global & Local



Local attention model

Local m – monotonic local. $p_i = i$

Local p – predictive local. $p_i = S \cdot sigmoid(v_p{}^T\, tanh(W_p s_{i-1}))$

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

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

Improve

Mechanism – Local

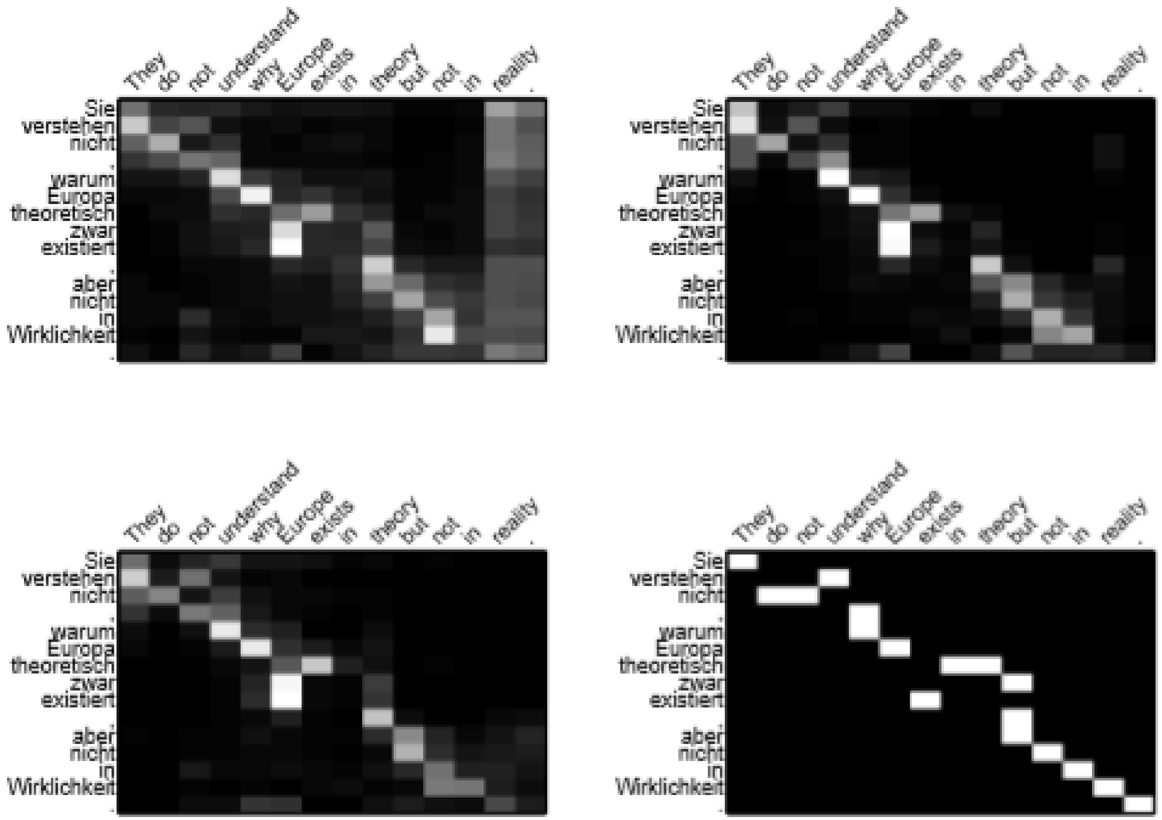
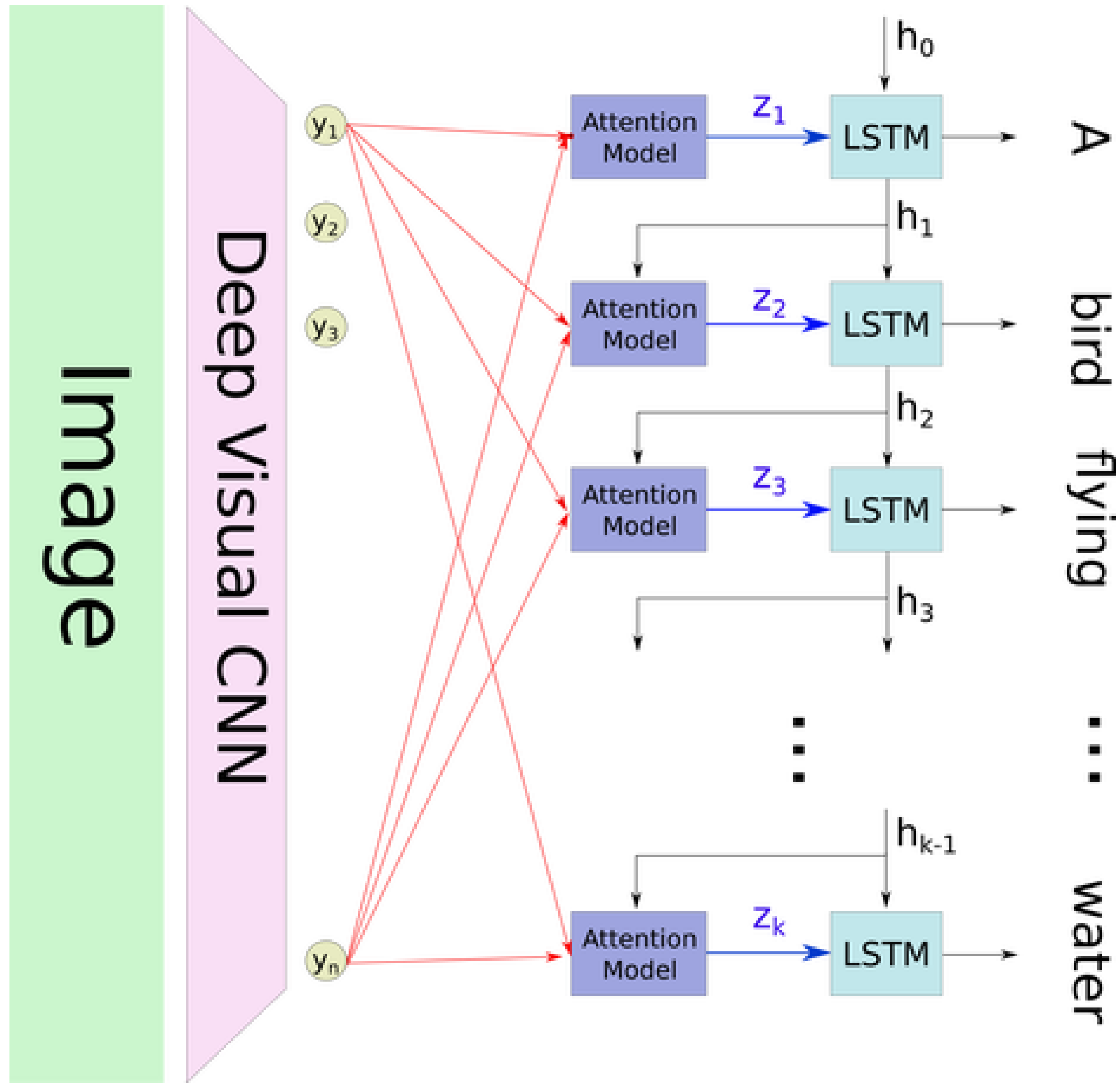


Figure 7: **Alignment visualizations** – shown are images of the attention weights learned by various models: (top left) global, (top right) local-m, and (bottom left) local-p. The *gold* alignments are displayed at the bottom right corner.

Application

Attention for Image Captioning



Application

Attention for Image Captioning

(a) A man and a woman playing frisbee in a field.

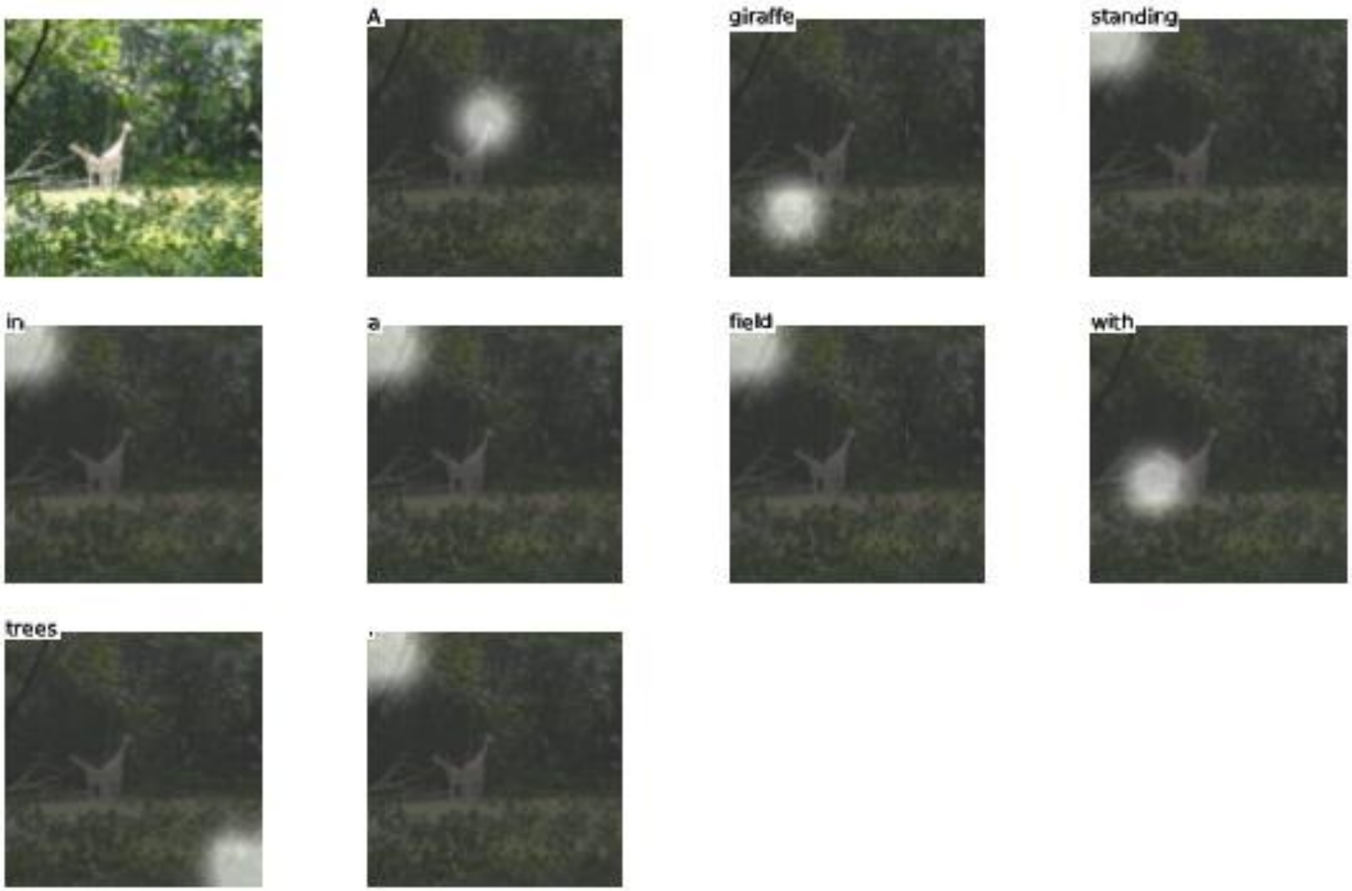
Application

Attention for Image Captioning

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

Application

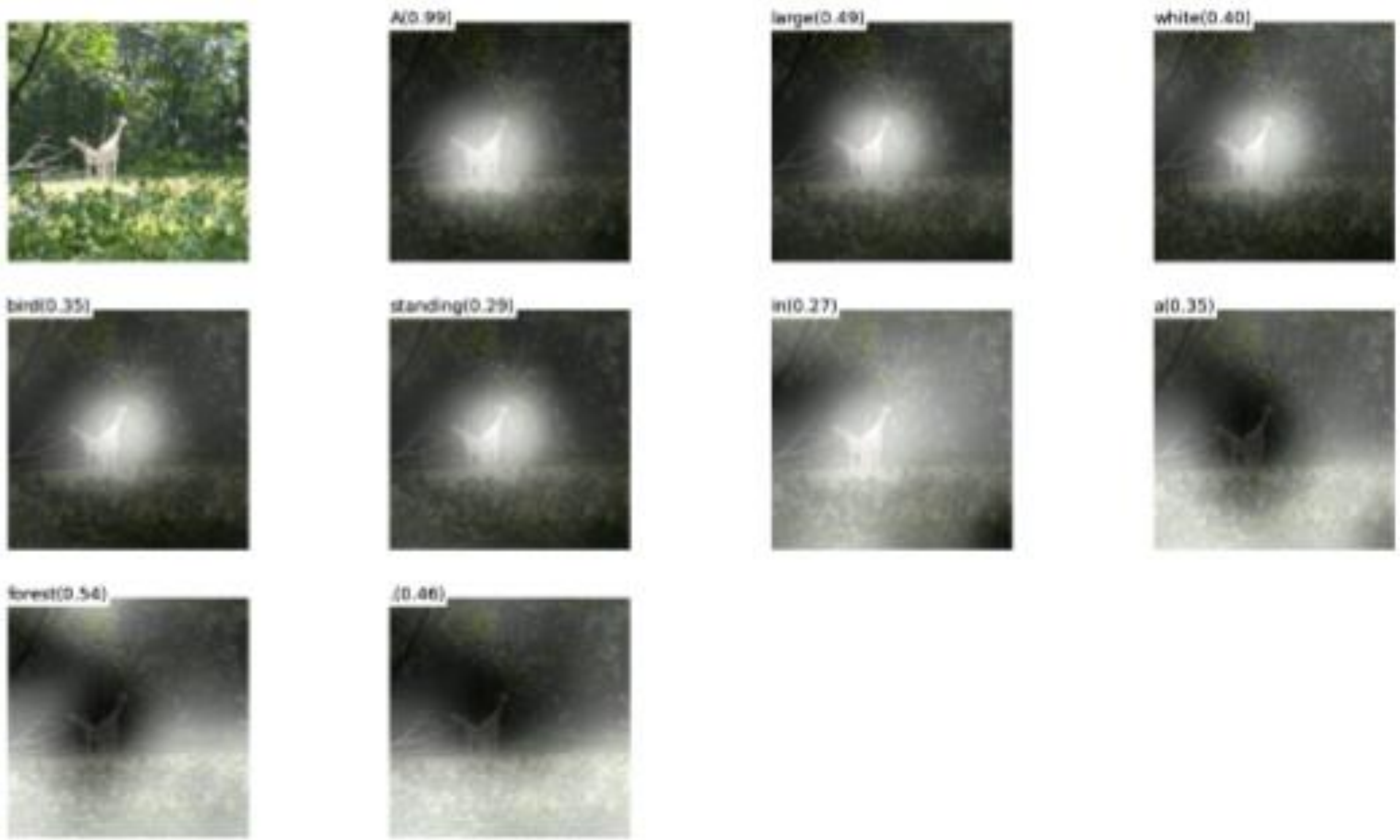
Attention for Image Captioning



(a) A giraffe standing in the field with trees.

Application

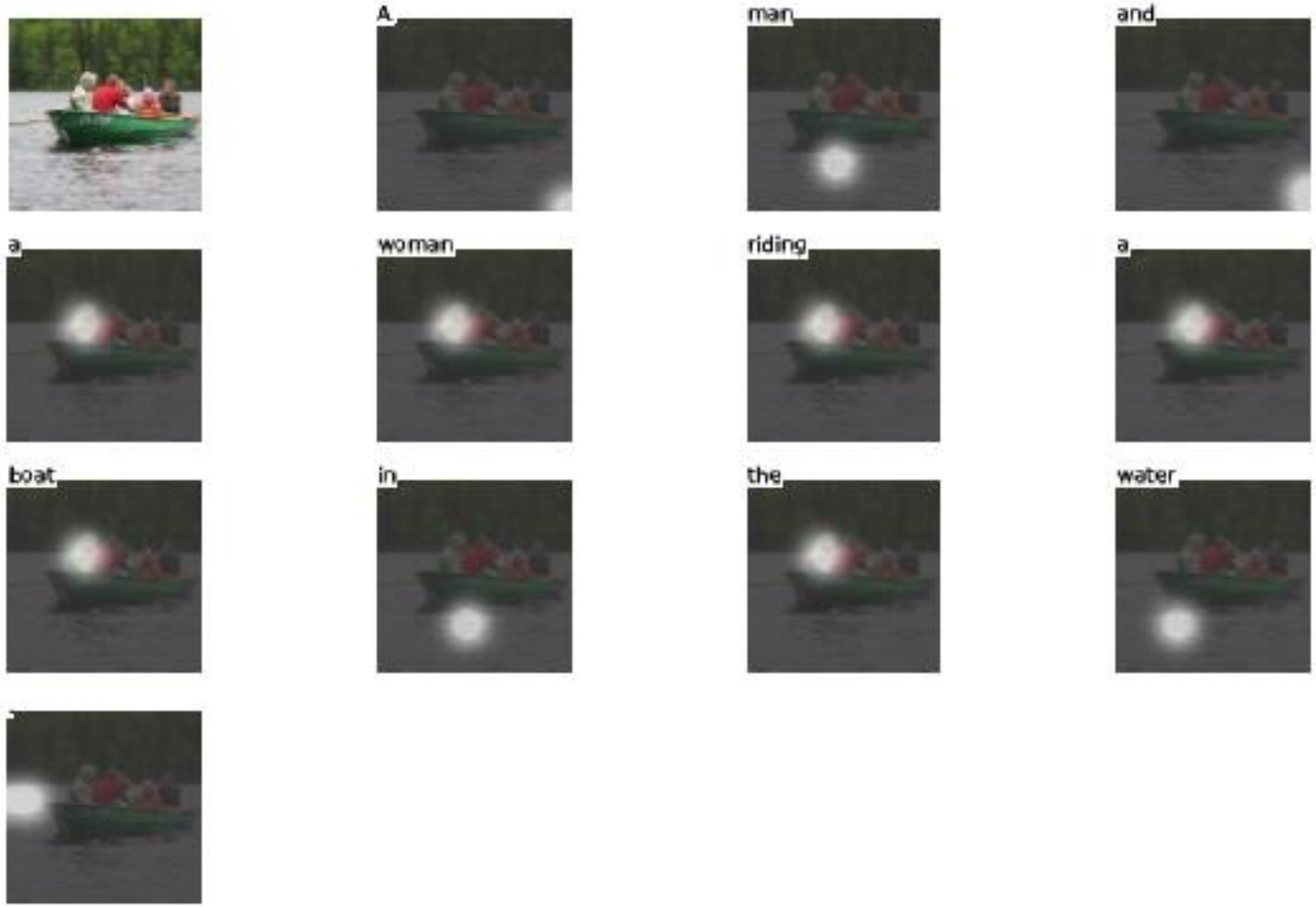
Attention for Image Captioning



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

Application

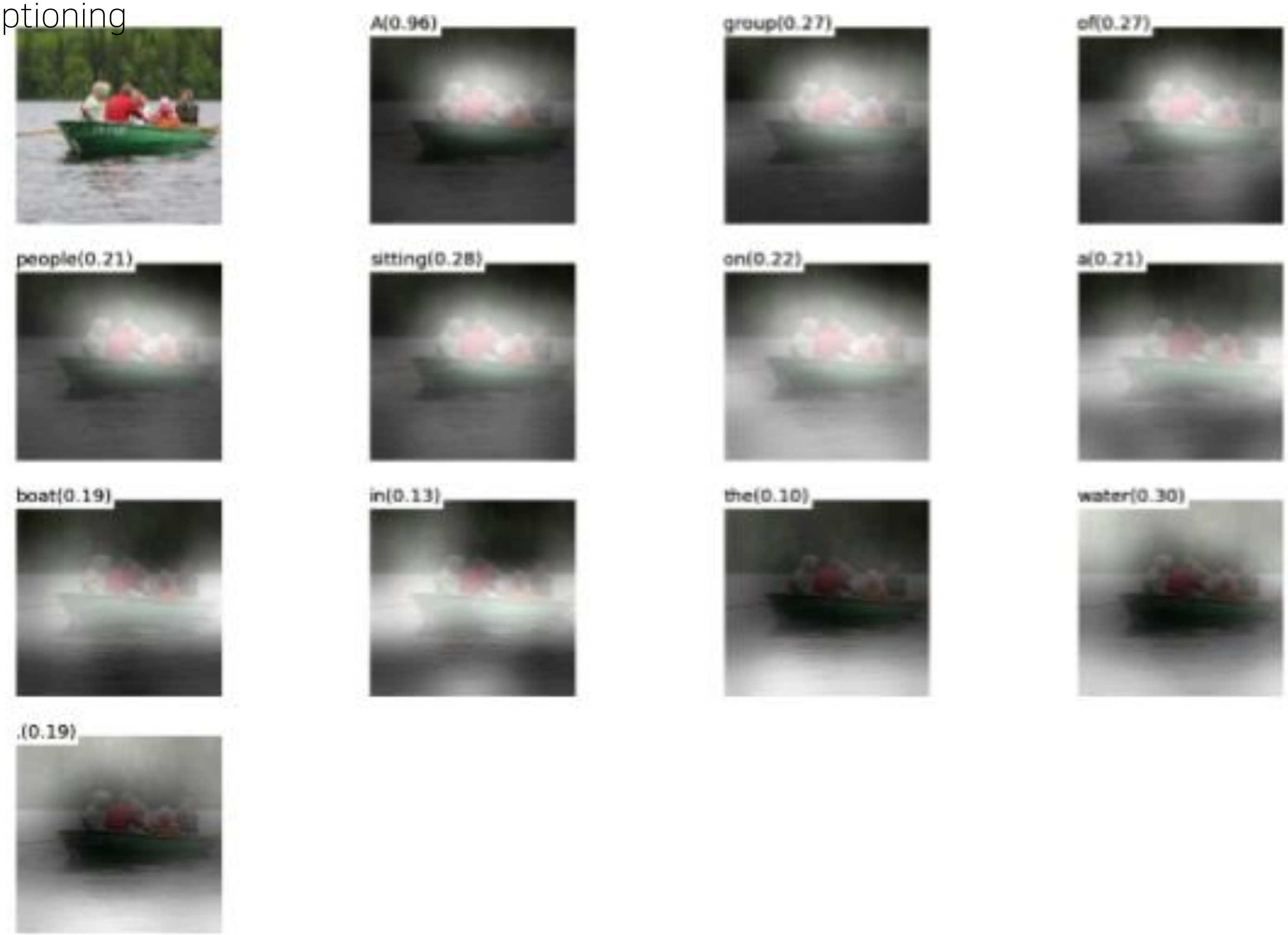
Attention for Image Ca[illegible]



(a) A man and a woman riding a boat in the water.

Application

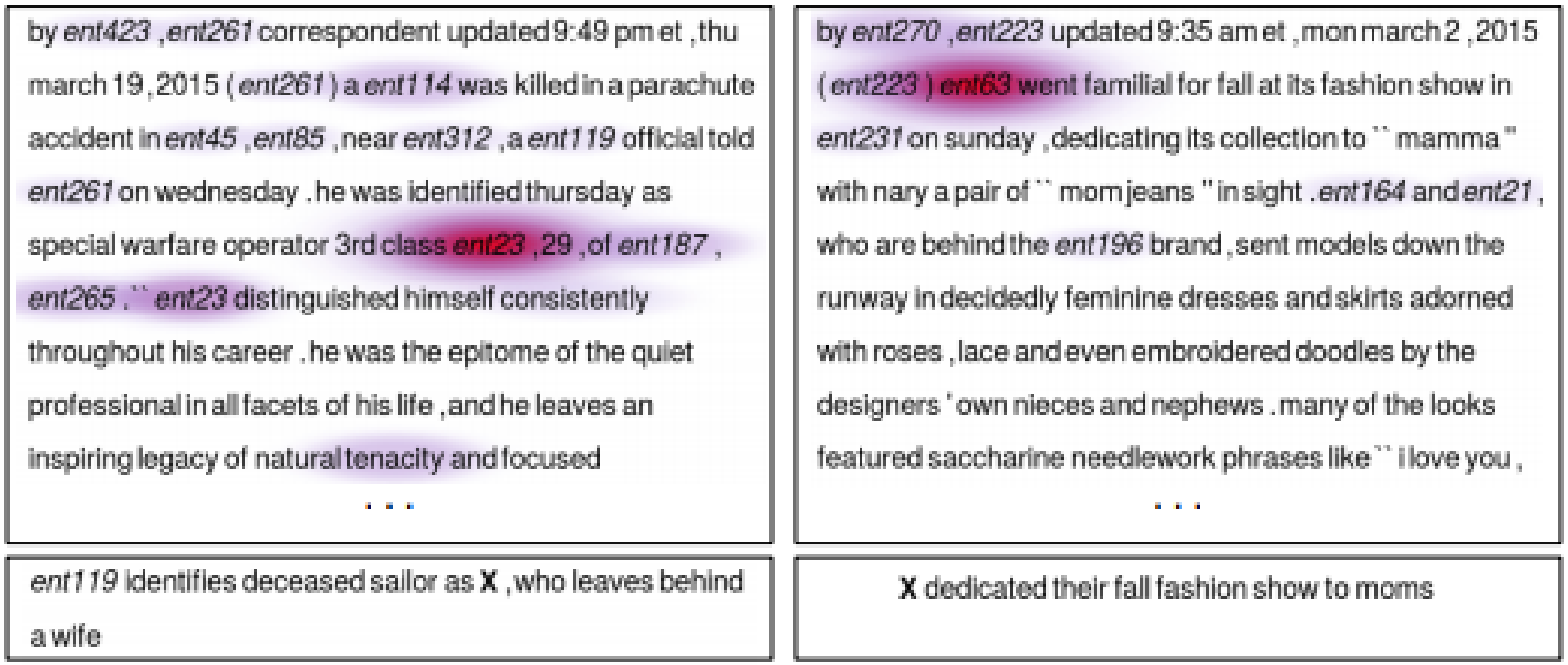
Attention for Image Captioning



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

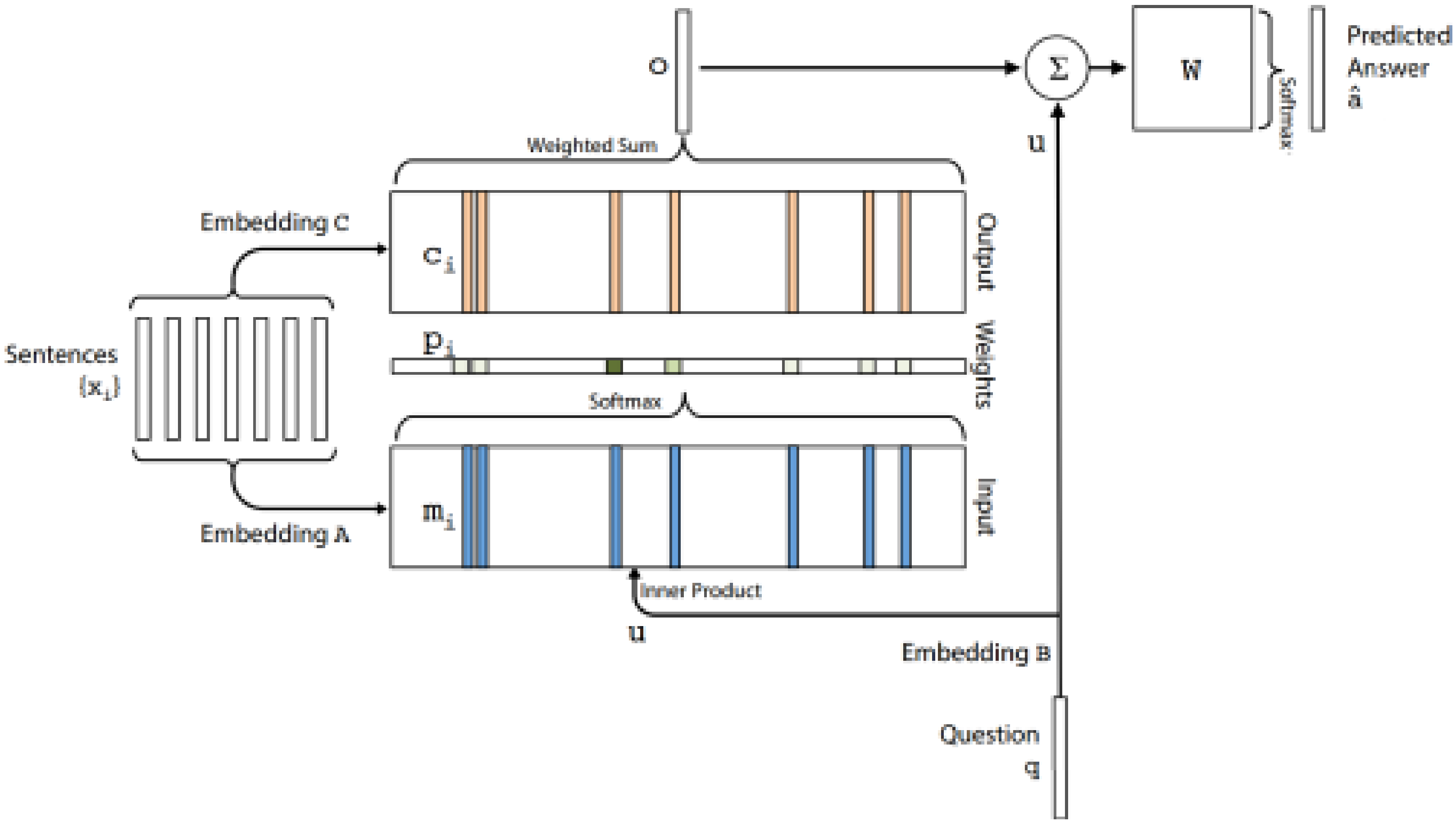
Application

Neural Turing Machine & Memory-based QA Models



Application

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

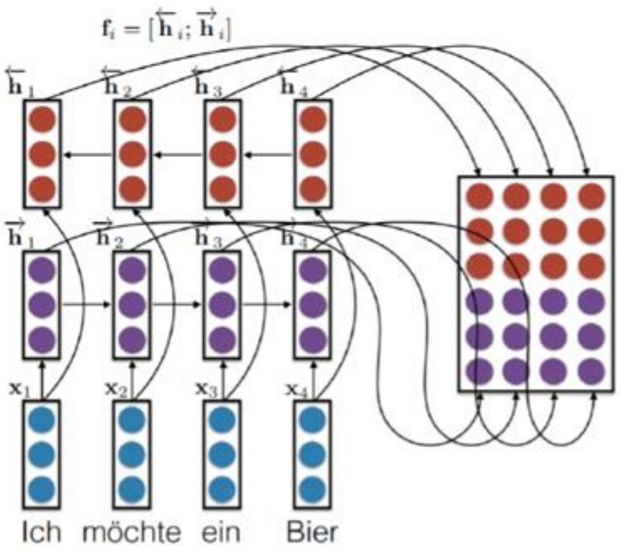
RNN model with Attention

```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

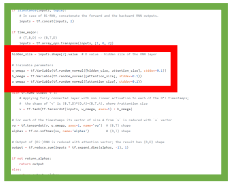
```
if isinstance(inputs, tuple):
    # In case of Bi-RNN, concatenate the forward and the backward RNN outputs.
    inputs = tf.concat(inputs, 2)
```



Forward 와 backward RNN output을 concat함.

Code

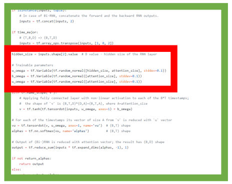
RNN model with Attention



```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



```
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
```

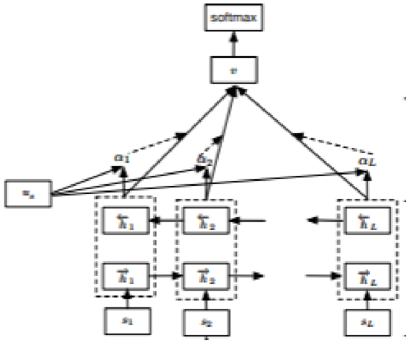
$$a(s_{i-1}, h_j) = v_a^T \tanh(W_a s_{i-1} + U_a h_j)$$

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

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

- 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

```
# 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.

Q&A

감사합니다.