딥러닝_언어분석 Day5

Outline

딥러닝 언어분석 Day5

1. Word Embedding
2. RNN
3. LSTM
4. Named Entity Recognition with Bidirectional LSTM-CNNs

Word Embedding

One-hot encoding

"영화 진짜 너무 감동적" → 스플릿 → '영화', '진짜', '너무', '감동적'

"황정민 진짜 최고인듯" → '황정민', '진짜', '최고인듯'

"내 인생 영화 오브 영화" → '내', '인생', '영화', '오브', '영화'

사전 = {'영화': 0, '진짜':1, '너무':2, '감동적':3, '황정민':4, '최고인듯':5, '내':6, '인생':7, '오브':8}

영화 → [1,0,0,0,0,0,0,0,0]

진짜 → [0,1,0,0,0,0,0,0,0]

⋮

인생 → [0,0,0,0,0,0,0,1,0]

One-hot encoding

Problem? → 단어 간 상관관계를 구할 수 없고, 차원이 너무 커짐!

dimension= $|V|$; 큰 데이터셋에서는 최대 천 만 개 이상

차원이 커짐에 따라 계산 및 공간 비용이 너무 비싸진다.

one-hot encoding 으로는 문맥적, 의미론적 정보를 담을 수 없다.

Distributed Representation of words

영화 ⟶ $[0.123, 0.643, \cdots, 0.212]$

진짜 ⟶ $[0.533, 0.186, \cdots, 0.935]$

⋮

인생 ⟶ $[0.864, 0.111, \cdots, 0.486]$

단어를 특정 차원의 실수 값을 가지는 분산 표현으로 잘 나타낼 수 있으면,

단어 간의 유사도와 단어의 문맥적 의미를 파악할 수 있지 않을까?

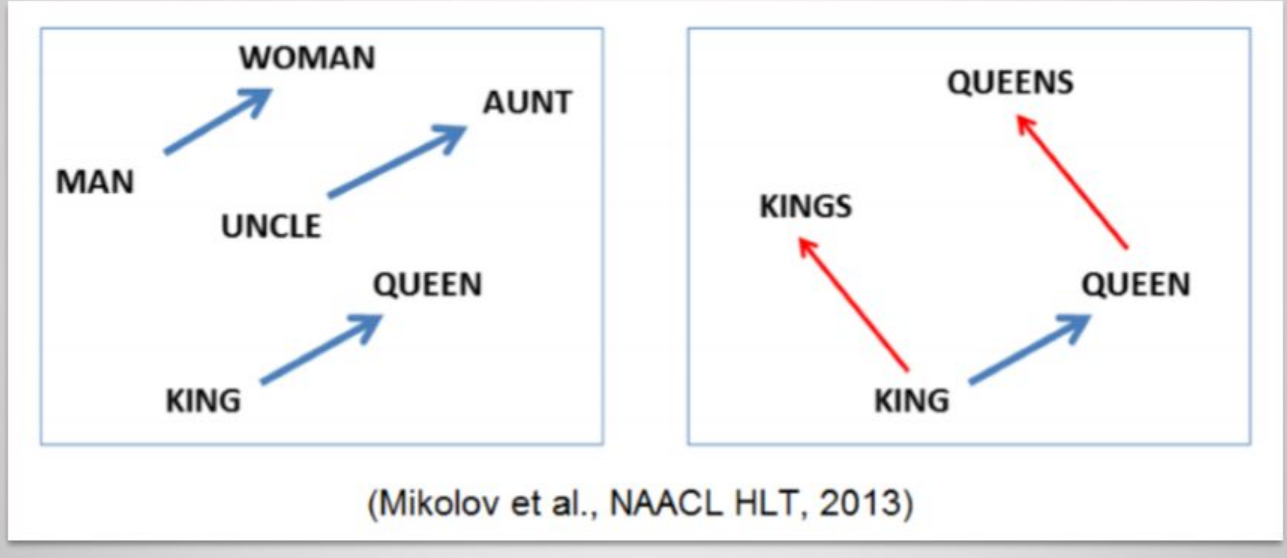
Distributed Representation of words

'비슷한 분포를 가진 단어들은 비슷한 의미를 가진다' 는 언어학의 distributional hypothesis 에 입각

비슷한 분포를 가진다는 것은 기본적으로 단어들이 같은 문맥에서 등장한다는 것을 의미

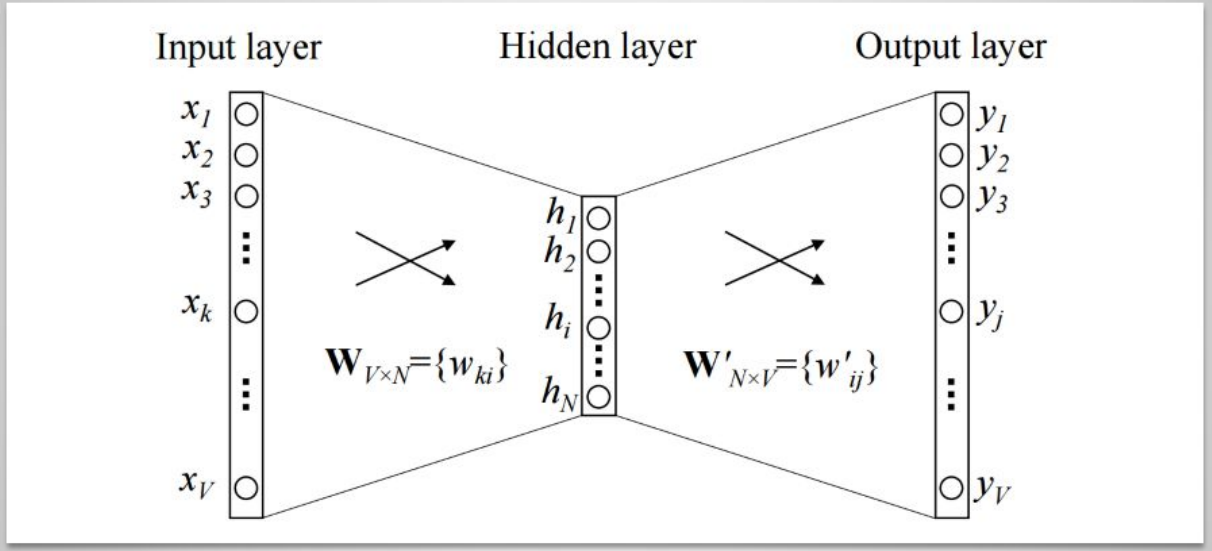
예를 들어, '사과', '포도', '딸기'라는 단어가 같은 맥락에서 등장하는 일이 빈번하게 일어난다면, 이 단어들이 유사한 의미를 가진 것으로 유추할 수 있다는 것

Distributed Representation of words



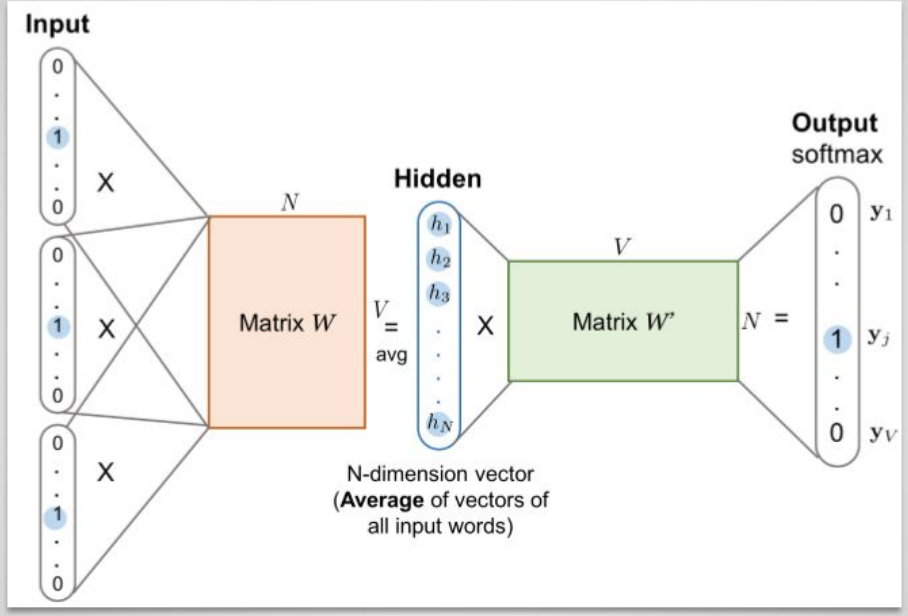
워드 임베딩에 의한 단어 간 연관성

Word2Vec methods



Word2Vec: Continuous Bag of Word/ Skip Gram Model

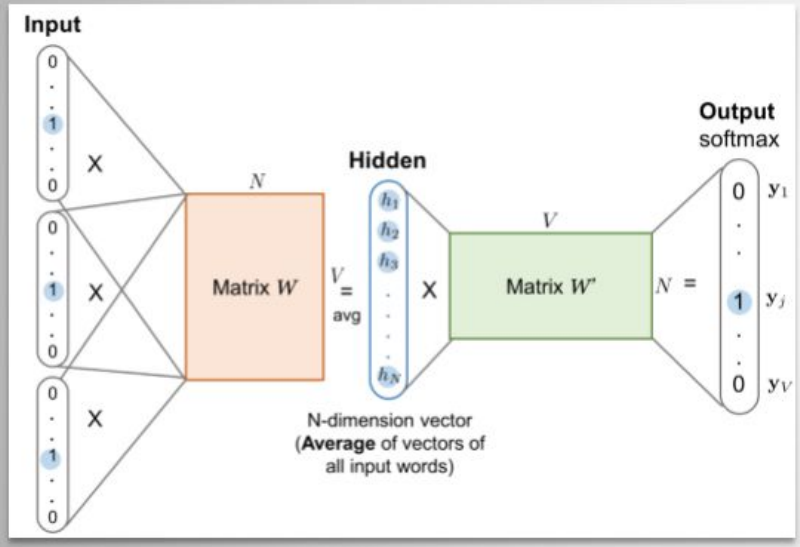
Word2Vec methods : CBOW



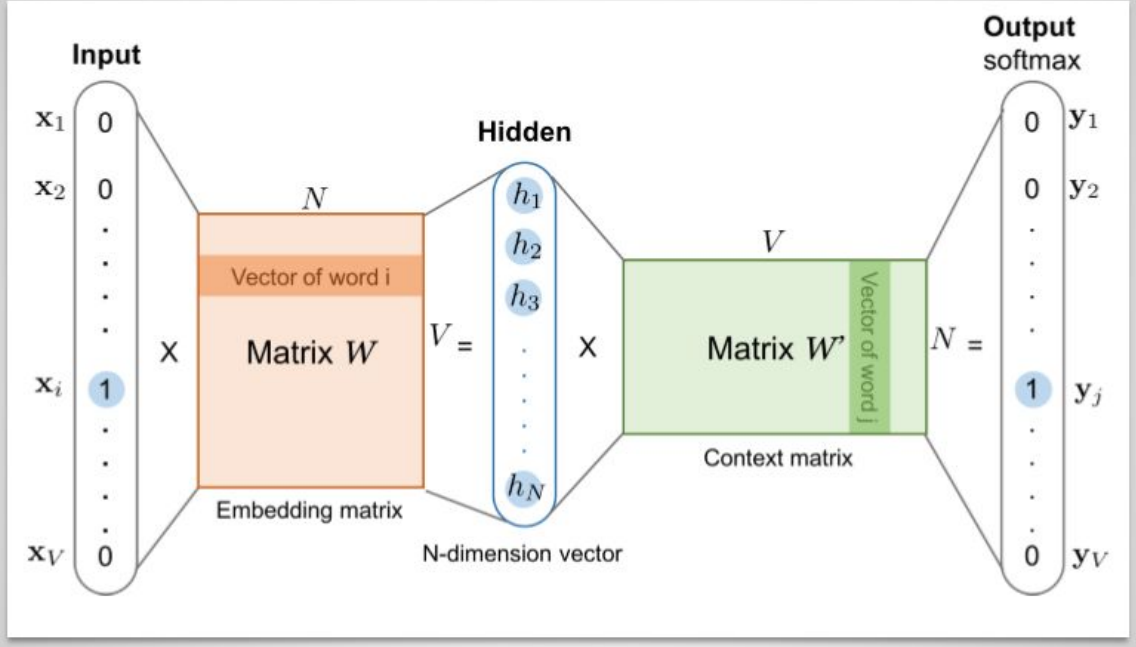
Continuous Bag of Word

주어진 단어 앞 뒤로 m개 씩 총 $2m$개의 단어를 Input 으로 이용하여 target 단어를 맞춤

- **Continuous Bag of Word**



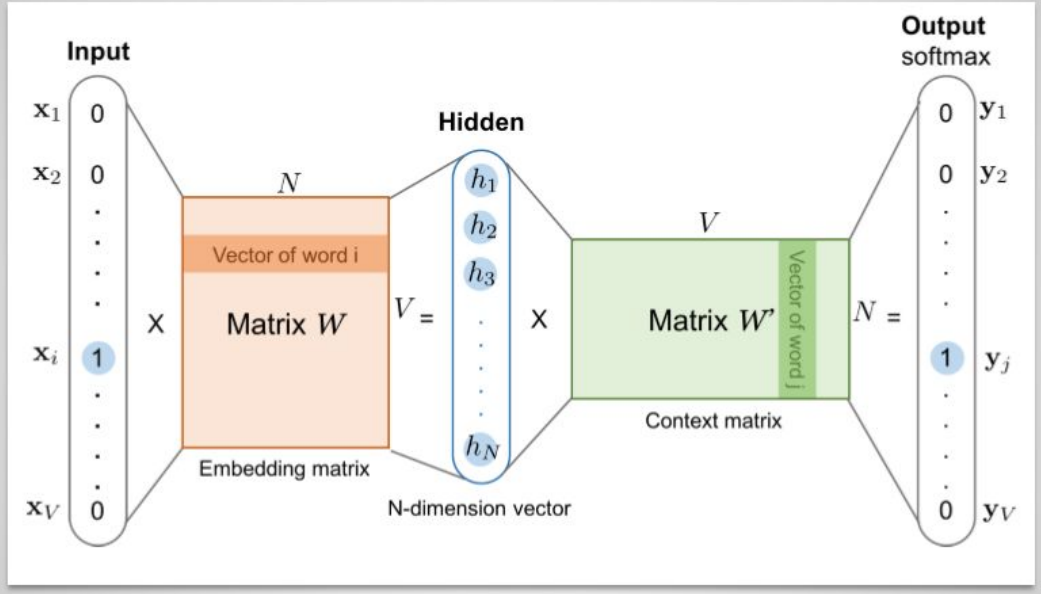
1. Generate one hot word vectors $(x_{c-m}, \dots, x_{c-1}, \dots, x_{c+1}, \dots, x_{c+m})$ for the input context of size $2m$
2. Get our embedded word vectors for the context $(v_{c-m} = Wx_{c-m}, \dots, v_{c+m} = Wx_{c+m})$
3. Average these vectors to get $\hat{v} = \frac{v_{c-m}+v_{c-m+1}+\cdots v_{c+m-1}+v_{c+m}}{2m}$
4. Generate a score vector $z = W'\hat{v}$
5. Turn the scores into probabilities $\hat{y} = softmax(z)$
6. Train embedded vectors using $H(\hat{y}, y) = -y_i \log(\hat{y}_i) = -\log(\hat{y}_i)$



Skip-Gram Model

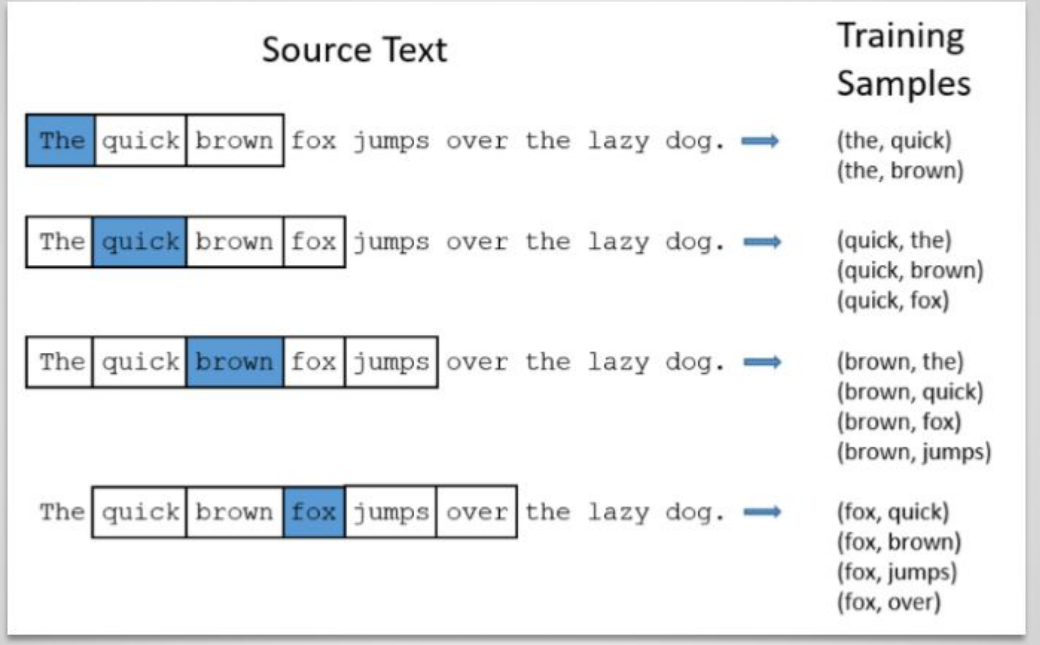
주어진 단어를 Input word 로 이용하여 주변 m개 단어를 예측

Word2Vec methods : CBOW



Skip-Gram Model

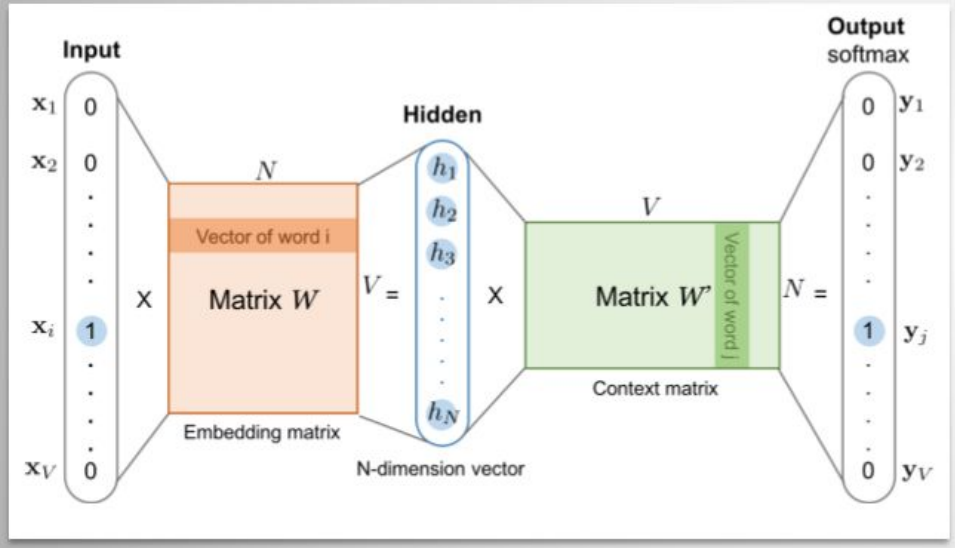
주어진 단어를 Input word 로 이용하여 주변 m개 단어를 예측



Skip-Gram Model

주어진 단어를 Input word 로 이용하여 주변 m개 단어를 예측

- **Skip-Gram**



$$p(w_o|w_I) = \frac{exp(v'_{w_o} \cdot v_{w_o})}{\sum_{i=1}^{V} exp(v'_{w_i} \cdot v_{w_o})}$$

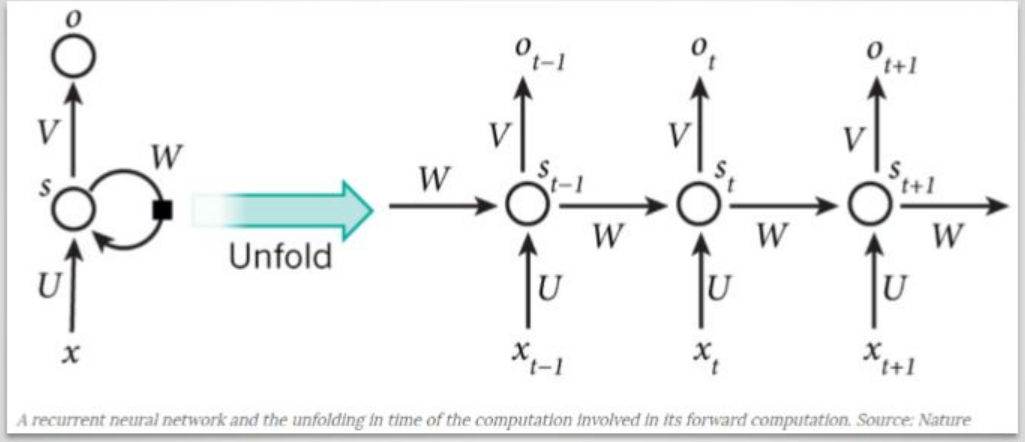
$$\zeta_\theta = -\log \frac{\exp(v'_{w_o} \cdot v_{w_o})}{\sum_{i=1}^{V} \exp(v'_{w_i} \cdot v_{w_o})}$$

$$= -v'_{w_o} \cdot v_{w_o} + \log \sum_{i=1}^{V} \exp(v'_{w_i} \cdot v_{w_o})$$

Calculating $\sum_{i=1}^{V} \exp(v'_{w_i} \cdot v_{w_o})$ is too expensive!

RNN

RNN (Recurrent Neural Network)



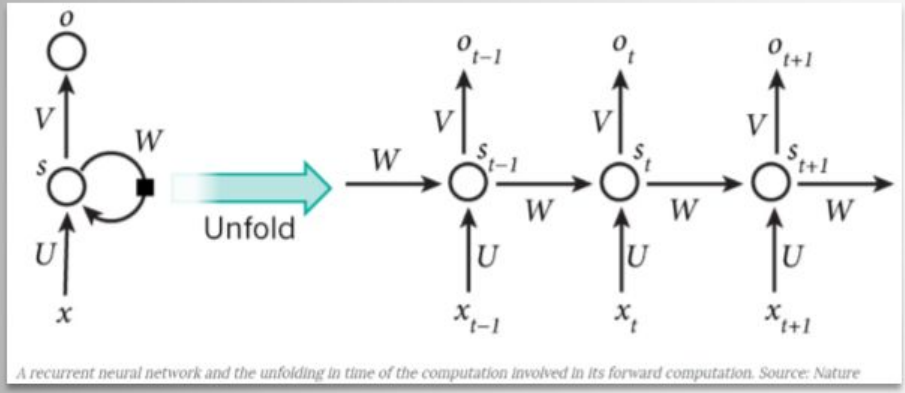
A recurrent neural network and the unfolding in time of the computation involved in its forward computation. Source: Nature

$$s_t = \tanh(U_s \cdot x_t + W_s \cdot s_{t-1} + b_s) \quad x_t \in \mathbf{R}^n,\ s_t \in \mathbf{R}^m,\ U_s \in \mathbf{R}^{n \times m},\ W_s \in \mathbf{R}^{m \times m}$$

$$s_t = \tanh(W \cdot [s_{t-1}, x_t] + b_s) \qquad W \in \mathbf{R}^{(m+n) \times m}, b_s \in \mathbf{R}^m$$

$$\hat{y}_t = o_t = \text{softmax}(V \cdot s_t + b_o)$$

- **RNN (Recurrent Neural Network)**



A recurrent neural network and the unfolding in time of the computation involved in its forward computation. Source: Nature

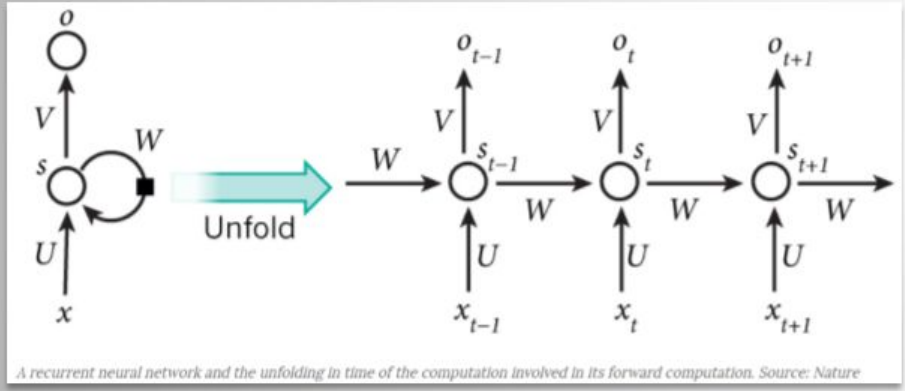
$$s_t = \tanh(U_s \cdot x_t + W_s \cdot s_{t-1} + b_s)$$

$$\hat{y}_t = o_t = \text{softmax}(V \cdot s_t + b_o)$$

$$J^{(t)}(\theta) = -\sum_{j=1}^{j=|V|} y_{t,j} \times \log(\hat{y}_{t,j})$$

$$J = -\frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta) = --\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{j=|V|} y_{t,j} \times \log(\hat{y}_{t,j})$$

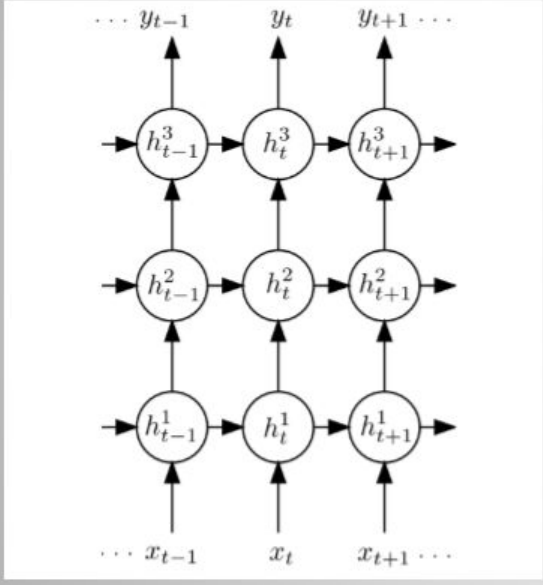
- **RNN (Recurrent Neural Network)**



A recurrent neural network and the unfolding in time of the computation involved in its forward computation. Source: Nature

- Hidden state s_t 는 previous step에서 발생한 정보 또한 다룰 수 있으며 output state인 o_t 는 오직 s_t를 이용하여 계산이 이루어진다.
- 기존 전통 딥러닝과는 다르게, RNN은 각 time step이 같은 parameter를 공유한다.
- 위 그림에서는 output state를 각 time step마다 계산하여 도출하지만 task에 따라서는 모든 output state가 꼭 필요로 되지 않는다. 예를 들어 감성분석의 경우, 우리는 각 단어 이후의 감성이 아닌 모든 단어를 본 뒤의 마지막 감성 즉, 마지막 state에 대해서만 관심이 있다.

- **Deep RNN**



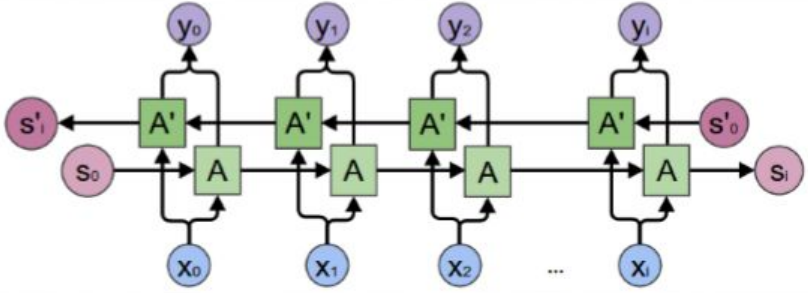
RNN에서도 hidden layer 층을 더 깊게 쌓음으로써 complexity가 더 높은 데이터에 대해서도 확률분포를 근사할 수 있다.

$$h_t^{(1)} = \tanh(U^{(1)} \cdot x_t + W^{(1)} \cdot h_{t-1}^{(1)} + b^{(1)})$$

$$h_t^{(i)} = \tanh\left(U^{(i)} \cdot h_t^{(i-1)} + W^{(i)} \cdot h_{t-1}^{(i)} + b^{(i)}\right), i \geq 2$$

$$\hat{y}_t = \text{softmax}(V \cdot h_t^{(n)} + b_o)$$

- **Bidirectional RNN**



Unidirectional RNN의 경우 time step이 길어질수록 앞쪽 time step에 대한 정보가 희석되기 때문에 양 방향으로 RNN을 진행함으로써 RNN을 향상시킨다.

$$\vec{h}_t = \tanh(\vec{U} \cdot x_t + \vec{W} \cdot \vec{h}_{t-1} + \vec{b})$$

$$\overleftarrow{h}_t = \tanh(\overleftarrow{U} \cdot x_t + \overleftarrow{W} \cdot \overleftarrow{h}_{t+1} + \overleftarrow{b})$$

$$\hat{y}_t = \text{softmax}(V \cdot [\vec{h}_t, \overleftarrow{h}_t] + b_o)$$

RNN (Recurrent Neural Network)

- 이론적으로는 RNN으로 parameter를 잘 조정하면 긴 문장에 대해서도 표현할 수 있는 능력을 가지고 있지만, 실제로 그렇게 동작하도록 학습하는 것이 매우 어렵거나 불가능하다.

→ Gradient 소실(vanishing) 또는 폭발(explosion) 문제 발생 때문

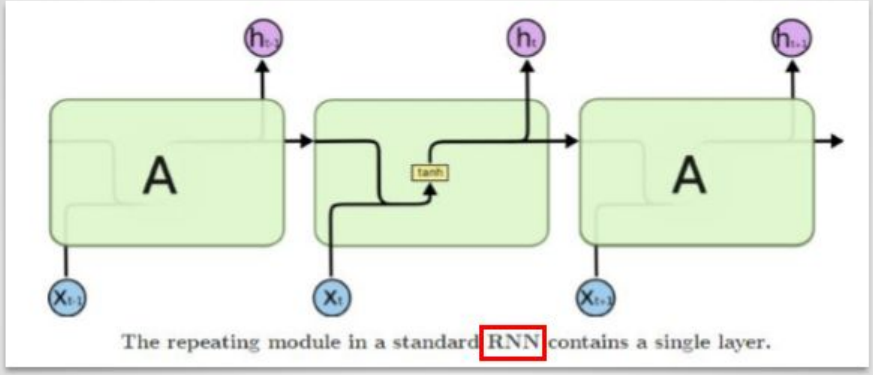
$$\frac{\partial \mathcal{L}}{\partial W_h} = \sum_t \sum_{k=1}^{t+1} \frac{\partial L(t+1)}{\partial z_{t+1}} \cdot \frac{\partial z_{t+1}}{\partial h_{t+1}} \cdot \prod_{i=k}^{t} \left(\frac{\partial h_{i+1}}{\partial h_i}\right) \cdot \frac{\partial h_k}{\partial W_h}$$

$$\frac{\partial h_{t+1}}{\partial h_t} = (1 - h_{t+1}^2) \cdot W_h$$

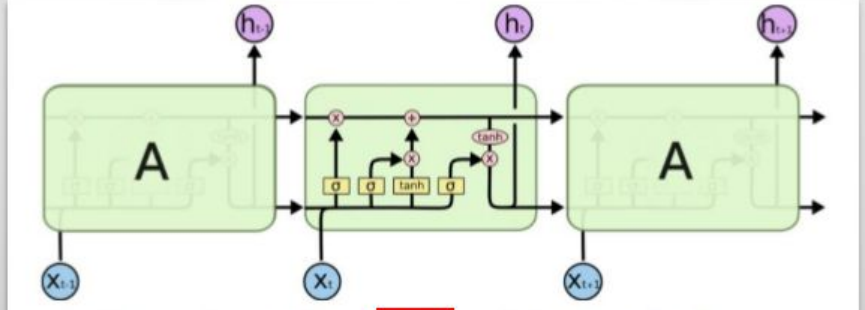
$$\frac{\partial h_{t+1}}{\partial h_{t-k+1}} = \prod_{j=1}^{k} \{(1 - h_{t-j+2}^2)\} \cdot W_h^k$$

LSTM

LSTM (Long Short-Term Memory)

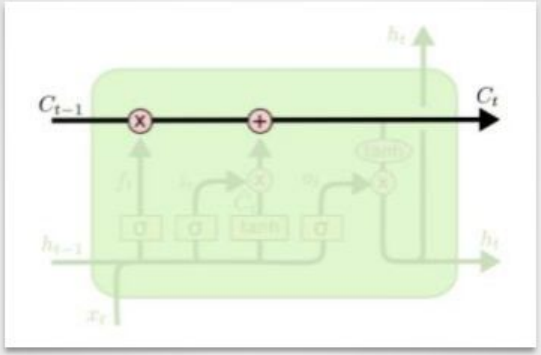


The repeating module in a standard RNN contains a single layer.



The repeating module in an LSTM contains four interacting layers.

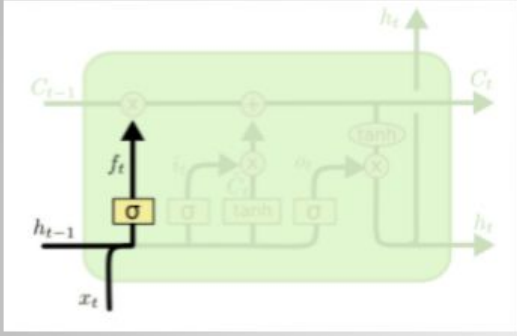
LSTM (Long Short-Term Memory)



- LSTM의 핵심은 위 그림에 나와있는 cell state 에 있다!
- 위 cell state를 보면 비선형 구조가 없이 단순한 선형 연산으로 이루어져 있는 것을 볼 수 있다.
- Gate 라는 구조를 통해 정보의 선별적 학습이 가능하게 된다.

LSTM (Long Short-Term Memory)

1. forget gate

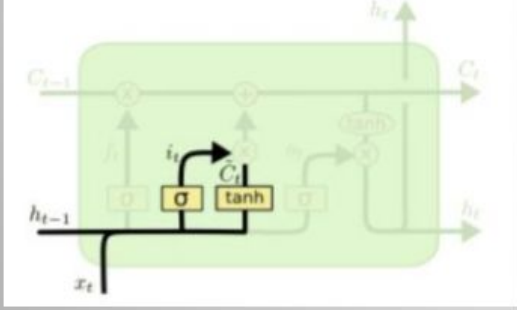


- previous hidden state 에서 어떤 정보를 잊을 것인지 결정

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

- (0, 1) 사이의 값으로 이루어진 vector

2. input gate



- new candidate hidden state에서 어떤 정보를 취할 것인지 결정

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

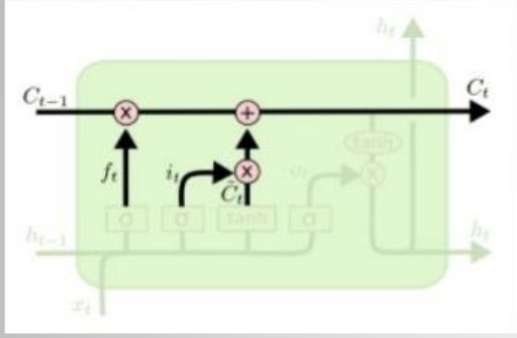
- (0, 1) 사이의 값으로 이루어진 vector

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

- new candidate state

LSTM (Long Short-Term Memory)

3. New hidden state

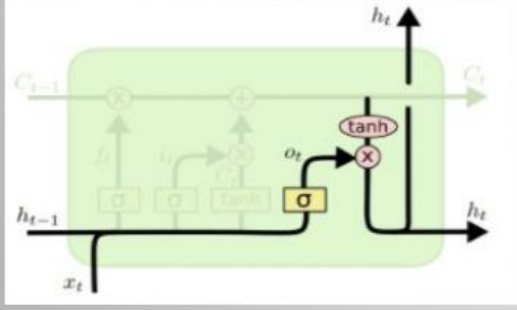


- forget gate 및 input gate를 통한 new hidden state 계산

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- $*$ 는 point-wise multiplication

4. output gate



- output gate 를 통한 output state (hidden state 2) 결정

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

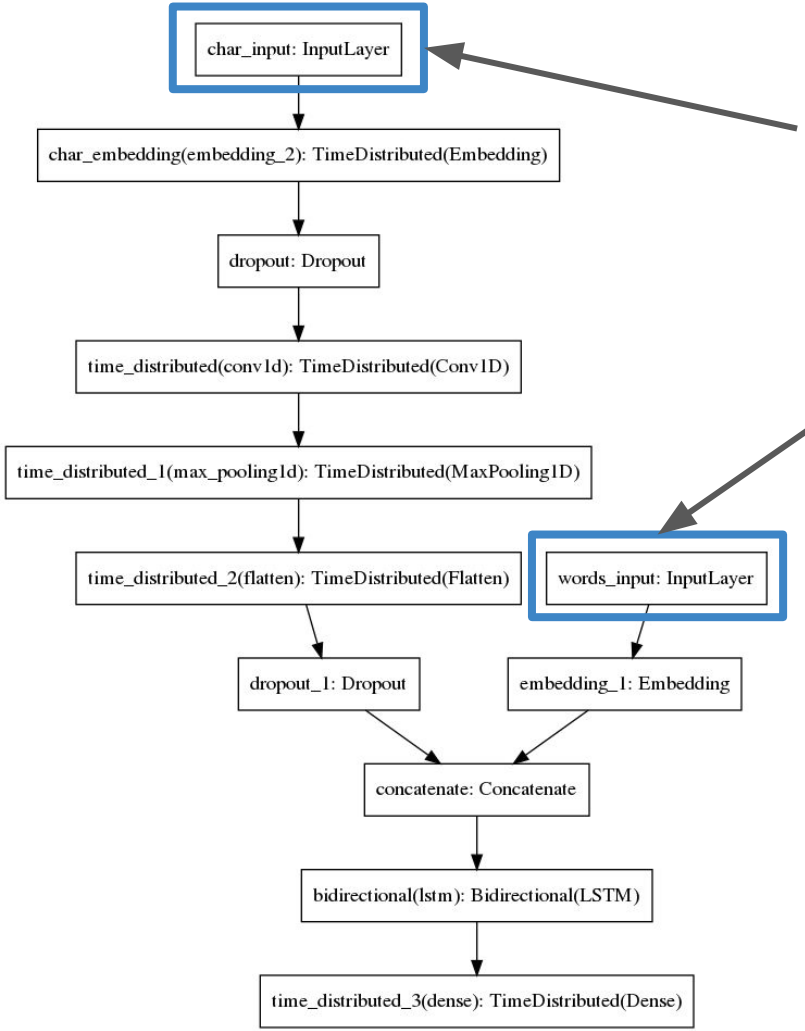
- (0, 1) 사이의 값으로 이루어진 vector

$$h_t = o_t * tanh(C_t)$$

- output state at time step t

Name Entity Recognition


char_input: InputLayer
char_embedding(embedding_2): TimeDistributed(Embedding)
dropout: Dropout
time_distributed(conv1d): TimeDistributed(Conv1D)
time_distributed_1(max_pooling1d): TimeDistributed(MaxPooling1D)
time_distributed_2(flatten): TimeDistributed(Flatten)
words_input: InputLayer
dropout_1: Dropout
embedding_1: Embedding
concatenate: Concatenate
bidirectional(lstm): Bidirectional(LSTM)
time_distributed_3(dense): TimeDistributed(Dense)



문장을 음절 단위로 나눠
integer coding

문장을 워드 단위로
토크나이징 후 integer coding

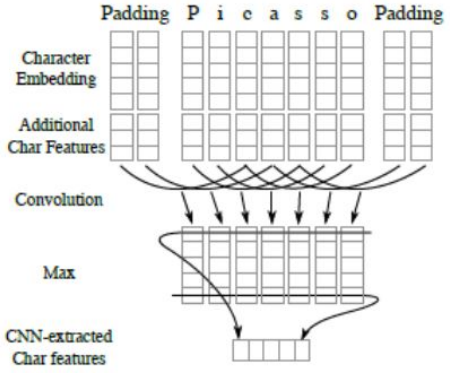
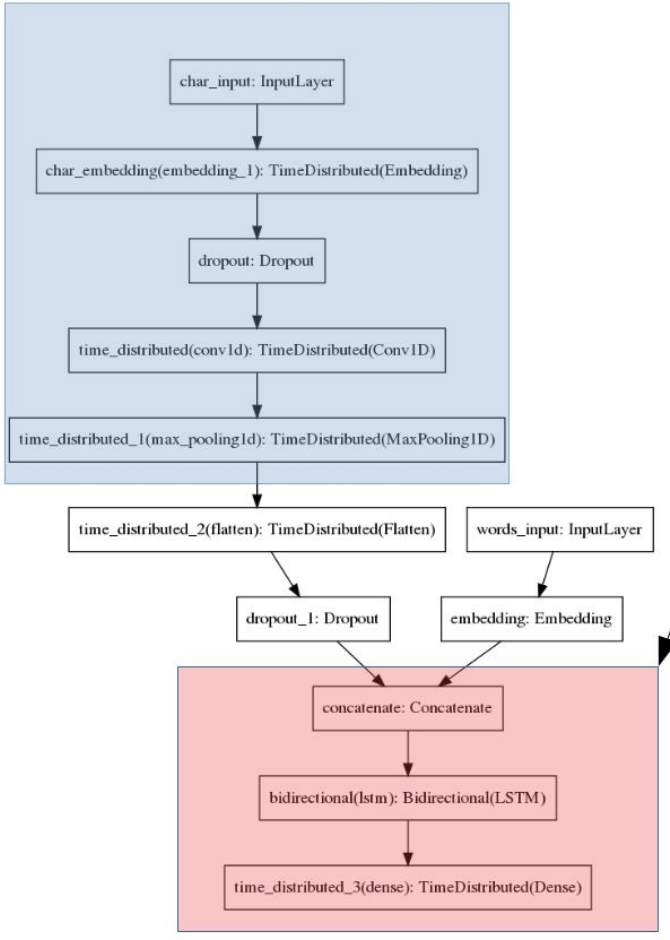


Figure 2: The convolutional neural network extracts character features from each word. The character embedding and (optionally) the character type feature vector are computed through lookup tables. Then, they are concatenated and passed into the CNN.



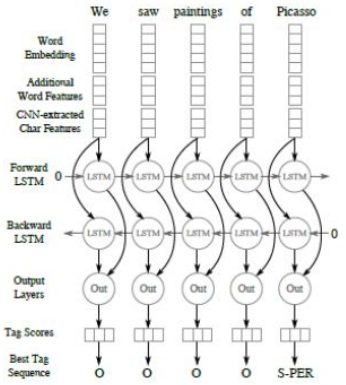


Figure 1: The (unrolled) BLSTM for tagging named entities. Multiple tables look up word-level feature vectors. The CNN (Figure 2) extracts a fixed length feature vector from character-level features. For each word, these vectors are concatenated and fed to the BLSTM network and then to the output layers (Figure 3).

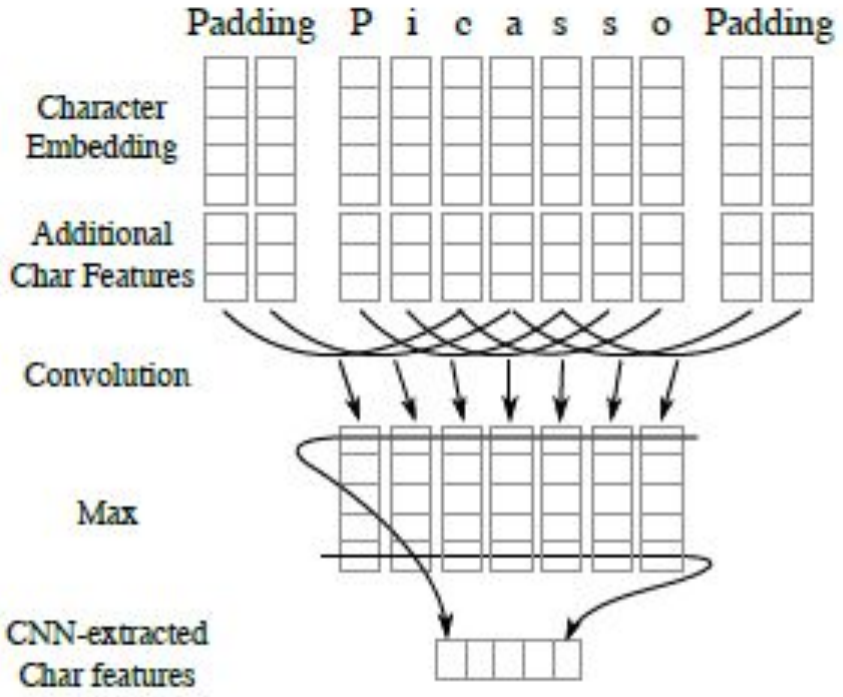
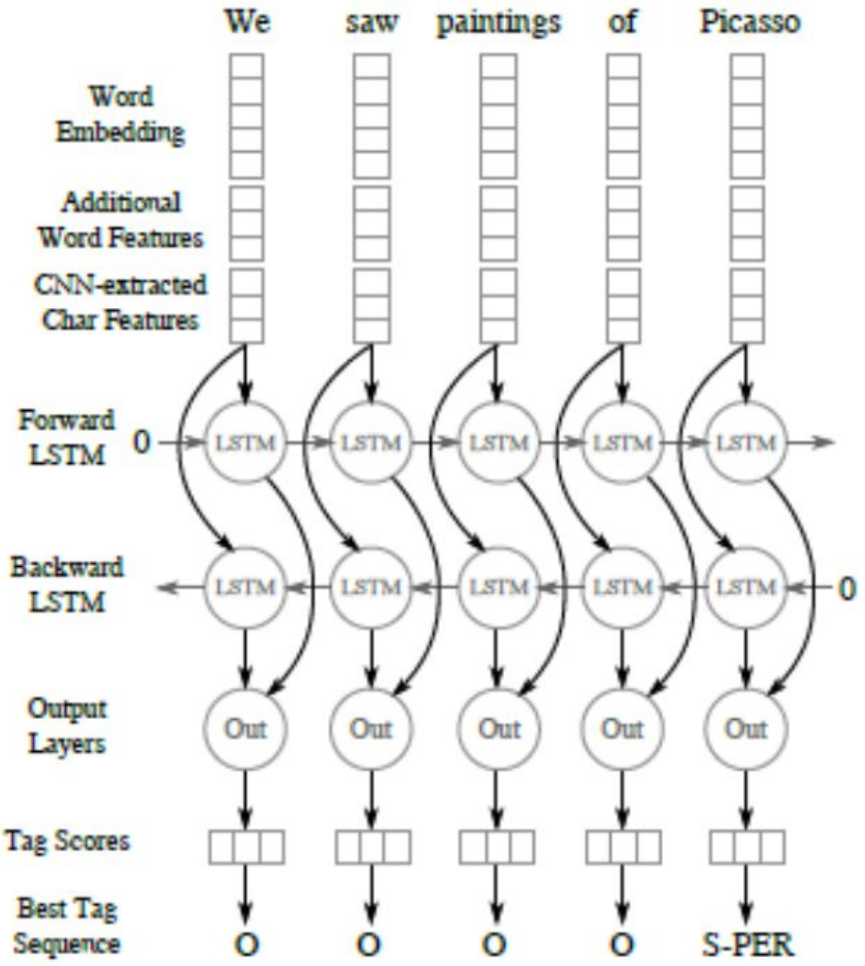


Figure 2: The convolutional neural network extracts character features from each word. The character embedding and (optionally) the character type feature vector are computed through lookup tables. Then, they are concatenated and passed into the CNN.


We
saw
paintings
of
Picasso
Word Embedding
Additional Word Features
CNN-extracted Char Features
Forward LSTM
0
LSTM
Backward LSTM
0
Output Layers
Out
Tag Scores
Best Tag Sequence
O
O
O
O
S-PER

Free Google Slides Templates