

Lecture 5. Language Models and Recurrent Neural Networks

Conventional Feature Representation

- Notations

$S_i.W$: Stack의 첫 단어

$b_i.W$: buffer의 첫 단어

$S_i.t$: Stack의 첫 단어의 POS Tag

$lc(S_i).W$: Stack의 첫 단어의 left-child 단어

$rc(S_i).W$: Stack의 첫 단어의 right-child 단어

$lc(S_i).t$: Stack의 첫 단어의 left-child 단어의 POS Tag

- State(c)

STACK(s)	BUFFER(b)
ROOT hit the	ball

POS Tags		Parsing Tree	Example			
John	NNP		$S_i.W$	$b_i.t$	$lc(S_i).W$	$rc(S_i).t$
hit	VBD	hit	the	MN	John	NULL
the	DT	nssubj				
ball	NN	John				

Indicator Features

1	$S_i.W = \text{the}$	$S_i.t = \text{DT}$	(True)
0	$S_2.W = \text{hit}$	$S_2.t = \text{VBD}$	$b_i.t = \text{NN}$ (False)
0	$lc(S_2).W = \text{John}$	$lc(S_2).W = \text{NULL}$	(False)
1	$lc(S_2).W = \text{John}$	$lc(S_2).t = \text{NNP}$	(True)
:			

$10^6 \sim 10^9$

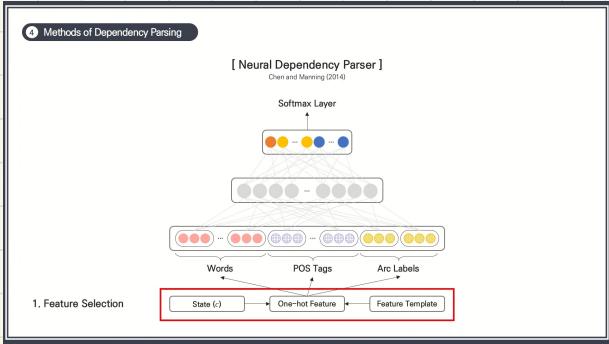
→ Binary & Sparse representation

일반적으로 1~3개 RSSI 결합된 indicator features

Parsing 소요시간 중 95% 이상을 feature 연산이 차지 (계산비용 높음)

단어 or POS Tag 의미 반영 못함

Neural Dependency Parser



Feature Template

- STACK, BUFFER의 top 3 단어 (6개)

$S_1, S_2, S_3, b_1, b_2, b_3$

- STACK top 1,2 단어의 1st and 2nd left and right child 단어 (8개)

$lc_1(S_1), rc_1(S_1), lc_2(S_1), rc_2(S_1), lc_1(S_2), rc_1(S_2), lc_2(S_2), rc_2(S_2)$

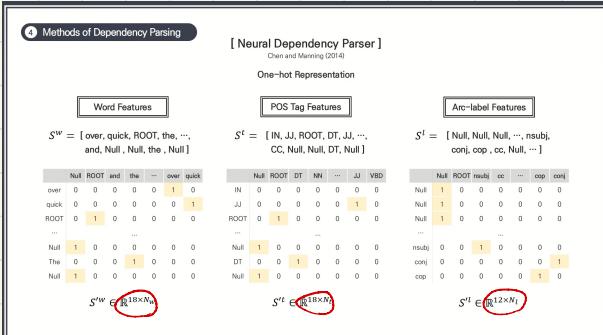
- STACK top 1,2 단어의 (left of left) and (right of right) child 단어 (4개)

$lc_1(lc_1(S_1)), rc_1(rc_1(S_1)), lc_1(lc_2(S_2)), rc_1(rc_2(S_2))$

- 선택된 word feature의 해당하는 POS Tag (18개)

- STACK과 BUFFER의 6개 단어를 제외하고 선택된 word에 따른 arc-label (12개)

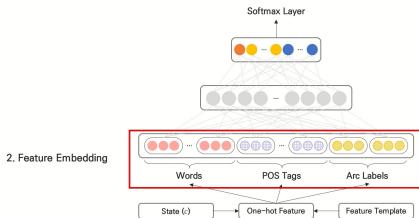
One-hot Representation



④ Methods of Dependency Parsing

[Neural Dependency Parser]

Chen and Manning (2014)

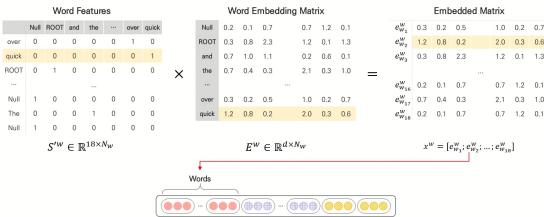


④ Methods of Dependency Parsing

[Neural Dependency Parser]

Chen and Manning (2014)

Feature Embedding

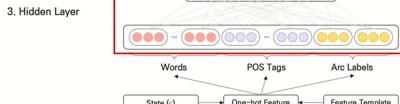


④ Methods of Dependency Parsing

[Neural Dependency Parser]

Chen and Manning (2014)

Hidden Layer



Hidden Layer

- Embedding vectorset weight matrix를 끌어와서 bias vector를 더하는 일반적인 feed forward network
- 하지만 ReLU, Sigmoid, Tanh와 같은 일반적인 activation function을 사용하지 X
- 대신 word, POS Tag, arc-label 간 상호작용을 반영할 수 있는 cube function을 사용함
- 엄밀한 수학적 증명을 하지는 않지만 이 방법은 non-linearity 대비 우수한 성능을 기록함

$$h = (W_i [x^w; x^t; x^l] + b)^3 = (w_1 x_1 + w_2 x_2 + \dots + w_{48} x_{48} + b)^3 = \sum_{i,j,k} (w_i w_j w_k) x_i x_j x_k + \sum_{i,j} b (w_i w_j) x_i x_j + \dots$$

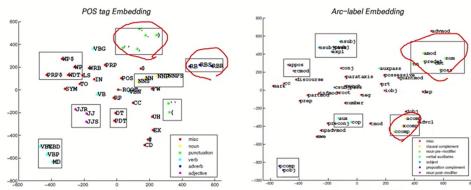
↑ word, POS tag, arc-label의 차례

4. Softmax Layer

[Neural Dependency Parser]

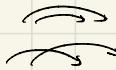
Self-Dependency I

- Random Initialization된 POS tag와 Arc-label vector가 학습이 진행되면서 의미적 유사성을 네포
 - t-SNE를 통해 2차원 공간 상에 표현하면 아래의 같이 유사한 요소들이 가까이 위치함



Projectivity & Non-Projectivity

- Projectivity(특사성): dependency arc가 서로 겹치지 않는 성질
non-projectivity(비특사성): " 겹치는 성질
 - 대부분 dependency parser는 projectivity를 가정함



Non-Greedy Algorithms

- Neural dependency Parser 는 Greedy 를 따른다
 - Beam search, Conditional Random Field, Global Normalization 은 전통적인 Transition-based dependency parser 보다
→ 더 높은 성능

Language Model

- 단어의 시퀀스(문장)에 대해, 얼마나 자연스러운 문장인지 확률 이용해 예측하는 모델
- Language Modeling: 주어진 단어의 시퀀스에 대해 다음에 나타날 단어가 무엇인지 예측하는 작업
- 기계번역, 음성인식, 자동완성 등에 이용

ex) the students opened their ____.

$\begin{array}{l} \text{books} \rightarrow P(\text{books}|\text{their}, \dots, \text{the}) \\ \text{laptops} \rightarrow P(\text{laptops}|\text{their}, \dots, \text{the}) \\ \text{exams} \rightarrow \dots \\ \text{minds} \rightarrow \dots \end{array}$

$$P(w_t | w_{t-1}, \dots, w_1)$$

$w_1, w_2, \dots, w_{t-n+1}, \dots, w_{t-1}, w_t, \dots, w_{T-1}, w_T$

$$\begin{aligned} P(w_1, \dots, w_T) &= P(w_1) \times P(w_2|w_1) \times \dots \times P(w_T|w_{T-1}, \dots, w_1) \\ &= \prod_{t=1}^T P(w_t|w_{t-1}, \dots, w_1) \end{aligned}$$

n-gram Language Models

- Neural Network 이전에 사용되었던 Language Model
- 예측에 사용할 앞 단어들의 개수를 정하여 모델링하는 방법
- n-gram: n개의 연속된 단어의 뭉치

$w_1, w_2, \dots, w_{t-n+1}, \dots, w_{t-1}, \text{window of } n \text{ previous words}$

$$\begin{aligned} P(w_1, \dots, w_T) &= P(w_1) \times P(w_2|w_1) \times \dots \times P(w_T|w_{T-1}, \dots, w_1) \\ &= \prod_{t=1}^T P(w_t|w_{t-1}, \dots, w_1) \end{aligned}$$

$$\approx \prod_{t=1}^T P(w_t|w_{t-1}, \dots, w_{t-n+1})$$

$$\begin{aligned} P(w_t | w_{t-1}, \dots, w_1) &\approx P(w_t | w_{t-1}, \dots, w_{t-n+1}) \\ &= \frac{P(w_t, w_{t-1}, \dots, w_{t-n+1})}{P(w_{t-1}, \dots, w_{t-n+1})} \quad \begin{array}{l} \text{→ prob of n-gram} \\ \text{→ prob of (n-1)-gram} \end{array} \\ &\approx \frac{\text{count}(w_t, w_{t-1}, \dots, w_{t-n+1})}{\text{count}(w_{t-1}, \dots, w_{t-n+1})} \end{aligned}$$

- 문제점 ① Sparsity Problems: n이 커질수록 안좋아지며, 일반적으로 $n < 5$ 로 설정

$$P(w | \text{students opened their}) = \frac{\text{count}(\text{students opened their } w)}{\text{count}(\text{students opened their})} \quad \begin{array}{l} \text{→ } w \in V \text{의 count를 더함} \\ \text{작은 단어의 count를 더함 (smoothing)} \end{array}$$

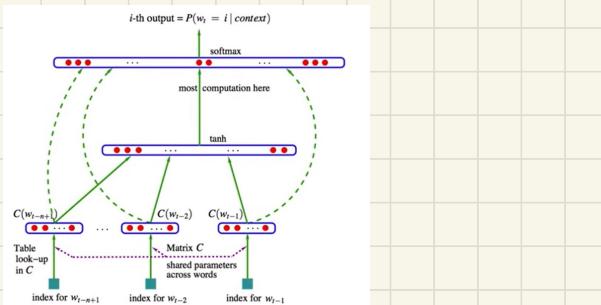
("opened their" 3 n-grams 단어를 낮춤 (back-off))

- ② Storage problems: n이 커지면 corpus가 증가하면 모델의 크기 증가함

$$P(w | \text{students opened their}) = \frac{\text{count}(\text{students opened their } w)}{\text{count}(\text{students opened their})} \quad \begin{array}{l} \text{→ Corpus 내 모든 n-gram에 대한} \\ \text{count를 저장해두어야 함} \end{array}$$

Window-based Neural Network Language Model (NNLM)

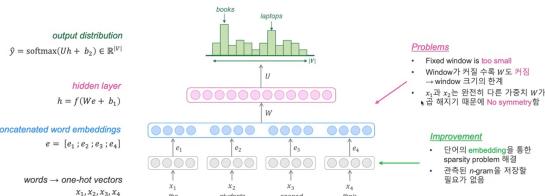
- "Curse of dimensionality"를 해결하기 위해 제안된 신경망 기반 Language Model
- Language Model 이면서 동시에 단어의 "distributed representation"을 학습



Language Model

✓ Window-based Neural Network Language Model(NNLM)

- "curse of dimensionality"를 해결하기 위해 제안된 신경망 기반 Language Model (Bengio et al. 2003)
- Language Model이면서 동시에 단어의 "distributed representation"을 학습



ANSWER

20

Recurrent Neural Network (RNN)

Recurrent Neural Network

✓ RNN Language Model

$$\hat{y}_t = \text{softmax}(Uh_t + b_2) \in \mathbb{R}^{|V|}$$

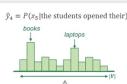
$$h_t = \sigma(W_h h_{t-1} + W_e e_t + b_1)$$

h_0 is the initial hidden state

$$e_t = E x_t$$

$$\text{words} \rightarrow \text{one-hot vectors}$$

$$x_1, x_2, x_3, x_4$$



- Advantages**
- 임베딩(embed)의 차이에 대한 처리의 편리성 (이론적으로) 깊이가 3 timestep I에 대해 처리 가능함
 - 입력에 따른 모델의 크기 증가 (단일 모델)
 - 각 timestep I에 동일한 가중치를 적용하여도 symmetry 할

ANSWER

23

✓ Training a RNN Language Model

- ① x_1, \dots, x_T 의 단어들로 이루어진 시퀀스의 Corpus를 준비한다.
- ② x_1, \dots, x_T 을 차례대로 RNN-LM에 주입하고, 각 step t 에 대한 \hat{y}_t 를 계산한다.
 - 주어진 단어에서부터 시작하여 그 다음 모든 단어들에 대한 확률을 예측

- ③ Step t 에 대한 손실함수 Cross-Entropy를 계산한다. (y_t is one-hot for x_{t+1})

$$L_t = CE(y_t, \hat{y}_t) = - \sum_{w \in \mathcal{V}} y_{t,w} \times \log(\hat{y}_{t,w}) = -\log(\hat{y}_{t,x_{t+1}})$$

- ④ 전체 step T 에 대해 계산한 손실함수 L_t 의 평균을 계산한다.

$$L = \frac{1}{T} \sum_{t=1}^T L_t = -\frac{1}{T} \sum_{t=1}^T \sum_{w=1}^{|\mathcal{V}|} y_{t,w} \times \log(\hat{y}_{t,w}) = -\frac{1}{T} \sum_{t=1}^T -\log(\hat{y}_{t,x_{t+1}})$$

✓ Training a RNN Language Model

