Stack LSTM 기반 한국어 의존 파싱을 위한 음절과 형태소의 결합 단어 표상 방법

나승훈○, 신종훈, 김강일
전북대학교, 한국전자통신연구원, 건국대학교
nash@jbnu.ac.kr, jhshin82@etri.re.kr, kikim01@konkuk.ac.kr

Improving Stack LSTMs by Combining Syllables and Morphemes for Korean Dependency Parsing

Seung-Hoon Na○, Jong-Hoon Shin, Kangil Kim
Chonbuk National University, ETRI, Konkuk University

요 약

Stack LSTM기반 의존 파싱은 전이 기반 파싱에서 스택과 버퍼의 내용을 Stack LSTM으로 인코딩하여 이들을 조합하여 파서 상태 벡터(parser state representation)를 유도해 낸후 다음 전이 액션을 결정하는 방식이다. Stack LSTM기반 의존 파싱에서는 버퍼 초기화를 위해 단어 표상 (word representation) 방식이 중요한데, 한국어와 같이 형태적으로 복잡한 언어 (morphologically rich language)의 경우에는 무수히 많은 단어가 파생될 수 있어 이들 언어에 대해 단어 임베딩 벡터를 직접적으로 얻는 방식에는 한계가 있다. 본 논문에서는 Stack LSTM 을 한국어 의존 파싱에 적용하기 위해 음절-태그과 형태소의 표상들을 결합 (hybrid)하여 단어 표상을 얻어내는 합성 방법을 제안한다. Sejong 테스트셋에서 실험 결과, 제안 단어 표상 방법은 음절-태그 및 형태소를 이용한 방법을 더욱 개선시켜 UAS 93.65% (Rigid평가셋에서는 90.44%)의 우수한 성능을 보여주었다.

주제어: Stack LSTM, Word embedding, morpheme embedding, Compositional representation, 의존 파싱

1. 서론

딥 러닝은 기본 자질 (raw features)로부터 여러 단계에서 걸쳐 복잡한 자질을 합성해 내는 표상 학습 (representation learning) 방법이다 [1]. 딥 러닝이 지향하는 표상 학습은 전통적인 기계 학습 방법과 구분되는 측면으로, 기존의 기계 학습에서는 별도의 자질 추출 (feature extraction) 단계를 통해 정교한 자질들을 직접 정의하지만, 표상 학습에서는 기본 자질로부터 복잡하고 정교한 자질까지 자동으로 학습하는 것을 목적으로 한다. 딥 러닝은 딥 아키텍처상의 모델을 효과적으로 학습할 수 있는 레이어 별 (layer-wise) 학습 방법 [2]이 제안된 이래, 컴퓨터비전, 음성인식 등에서 성능을 크게 높이기 시작하면서 다양한 영역으로 확대되고 있다.

자연언어처리에서도 딥 러닝을 이용한 연구들이 계속 늘어나고 있으며, 초기의 딥 러닝 기반 언어처리는 순차열 태깅을 위한 [3-4]의 ConvNet (Convolutional Neural Network)기반 방법, 문장의 구조 분석 등에 적용된 [5]의 재귀 신경망 (Recursive Neural Network)이 대표적이고, 최근에는 RNN (Recurrent Neural Network) [6-7]을 이용한 연구가 크게 급성장하고 있는 추세이다. 대표적인 RNN 응용 사례는 뉴럴 기계 번역 (Neural Machine Translation) [8-9]으로, 이는 뉴럴 기계 번역은 단일 딥 아키텍터상에서 End-to-end 번역을 수행하는 딥 러닝 방법이다. 최근 기존의 통계 기반 번역 (Statistical Machine Translation)의 성능을 뛰어넘는 결과들이 계속 발표되면서 주목을 받고 있다.

의존 파싱의 경우에도 딥 러닝에 기반한 연구 결과가 다수 발표 되고 있다. 초기에는 FNN (Feedforward neural network)에 기반을 두었으나 [10-11], 최근 들어 LSTM (Long short term memory)과 같은 RNN 계열이 확장 적용되고 있다 [12-15]. 이러한 RNN기반 딥 러닝 의존 방법 중, [12]는 Stack LSTM기반 의존 파싱을 제안하였다. 이 방법은 스택과 버퍼내의 모든 정보를 Stack LSTM을 통해 인코딩하고 이들로부터 다음 파싱 전이 액션을 결정하는 방식이다. Stack LSTM기반 의존 파싱은 영어권에서 약 93%대의 성능을, 다른 언어에 대해서도 best published결과를 뛰어넘거나 비슷한 성능을 보이고 있다 [12-13].

Stack LSTM기반의 의존 파싱에서는 버퍼의 초기화를 위한 입력 단어 표상 (word representation) 방식이 중요하다. 그러나, 한국어와 같이 형태적으로 복잡한 언어 (morphologically rich languages)의 경우에는 무수히 많은 단어가 파생될 수 있어 이들 언어에 대해 단어 임베딩 벡터를 직접적으로 얻는 방식에는 한계가 있다 .

본 논문에서는 Stack LSTM기반의 한국어 의존 파싱을 위해 *음절-태그*와 *형태소* 정보로부터 얻은 표상들을 결합하여 최종적으로 단어 표상을 얻어내는 방법을 제안한다. 음절-태그는 음절과 음절 태그로 구성된 단위를, 형태소는 형태소의 표층정보와 형태소 태그로 구성된 단위를 가리킨다. 실험 결과, 제안 결합 기반 단어 표상 방식은 Sejong테스트 집합에서 기존의 음절-태그와 형태소

를 이용한 방법을 각각 개선시켜, [11]의 테스트셋에서는 UAS 90.44%, LAS 87.93%, [16-17]의 테스트셋에서는 UAS 93.65%, LAS 91.57%의 우수한 성능을 보여주였다.

2. 관련 연구

의존 파싱 연구는 그래프 기반 방법 [18]과 전이 기반 방법 [19], 다단계 청킹 (cascaded chunking) [19], 결합 방식 [17,21-22]으로 크게 나뉘어 연구되고 있다. 한국어의 경우에도 [23-24]의 그래프 기반 방법, [11,14-16]의 전이 기반 방법, [25-26]의 다단계 순차 태깅 방법, [17]의 결합 모델, [27]의 번역 모델 기반 방법 등으로 연구되고 있다. 딥 러닝 기반 한국어 의존 파싱에서는 주로 전이 기반 방법에서 연구가 주축을 이루고 있는데, [11]에서는 FNN을 이용, [14-15]에서는 RNN 계열에 기반을 둔 방법을 이용한 방법을 제안하였다.

본 논문과 가장 연관도가 높은 기존 연구는 [13]으로 이는 [12]를 확장하여 단어의 벡터 표현을 위해 문자단위의 임베딩 벡터를 이용하여 단어 표상을 얻는 방식을 제안하였고, 한국어를 포함하여 일부 형태적으로 복잡한 언어에 대해 추가적인 성능 개선을 가져왔다. 한편, [14]은 RNN을 이용한 전이 기반 한국어 의존 파싱을 제안하였는데, [12]과 달리 스택과 버퍼별로 LSTM을 별도로 적용하지 않고 스택과 버퍼내의 자질 정보를 통합하여 하나의 입력 층을 구성하여 이를 RNN에 적용한 것이다. [15]에서는 [13]의 Stack LSTM을 이용하여 한국어 의존 파싱에 적용하였으며, 문자 단위 임베딩 대신 음절-태그와 형태소를 각각 개별적으로 이용한 방법을 제안하였다.

본 연구는 [15]에 더 나아가 기존의 음절-태그와 형태소에 기반한 표상들을 결합하는 방법을 제안한다. 추가로, [15]에서는 형태소의 경우 형태소 임베딩 벡터를 랜덤하게 초기화시킨 결과만을 제시하였으나, 본 연구에서는 추가로 대용량의 코퍼스로부터 사전학습된 임베딩 벡터로 초기화시켜 기존 방법의 성능을 더욱 개선시켰다.

3. Stack LSTM 기반 의존 파싱

3.1 전이 액션 결정 모델

그림 1은 [12]의 Stack LSTM을 이용한 전이 기반 의존 파싱 과정의 구조도를 보여주고 있다. 전이 액션(transition action)을 결정하기 위해 스택 (S)과 버퍼 (B), 액션 히스토리 (A) 정보들을 각각 저장하기 위해 별개의 Stack LSTM를 활용하고 있다. 스택과 버퍼를 초기화 할 때, 스택 S에 <s>의 노드를 추가하여 TOP노드가 <s>를 가리키도록 하며, 버퍼 B에는 입력 단어들의 역순으로 Stack LSTM의 push를 이용하여 버퍼의 stack LSTM의 TOP노드가 입력의 첫 단어를 가리키도록 한다. 버퍼의 마지막에는 문장의 끝을 의미하는 </s>가 추가된다.

그림 1의 제시된 구조는 세 개의 Stack LSTM의 내용을 입력으로 취하여 각각의 전이 액션의 확률을 출력하는 FNN이다. S, B, A의 상태 벡터들이 연결(concatenation)된 입력층, ReLU 활성화 함수를 사용하는 은닉층, 마지막에 액션 노드들에 대응되는 출력층으로 구성된다. 전이 액션이 결정되면 버퍼와 스택의 내용이 갱신이 되고 이렇게 갱신된 버퍼와 스택 내용은 다시 Stack LSTM을 이용하여 인코딩되어 다음 액션을 결정하기 위한 FNN 모델의 입력이 된다.

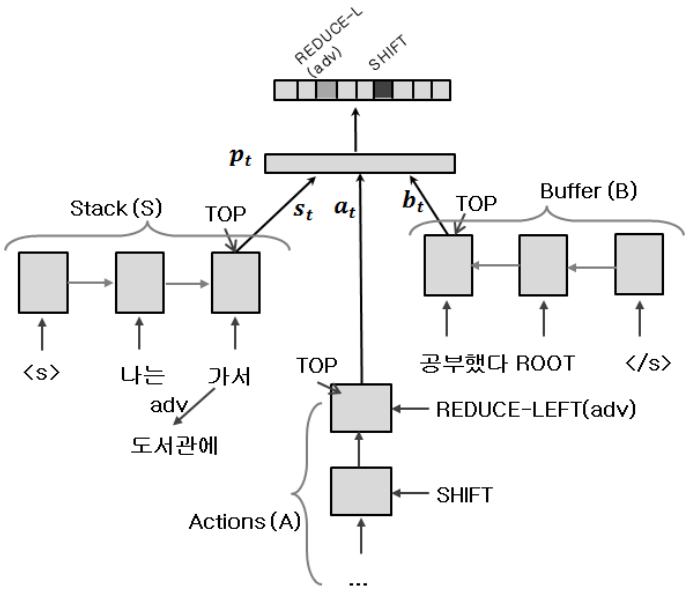

그림 1 Stack LSTM을 이용한 전이 기반 의존 파싱에서 다음 전이 액션 분류를 위한 FNN [12]

그림 2의 FNN을 형식적으로 기술하기 위해, S, B, A의 TOP노드의 상태 벡터를 각각 $\boldsymbol{s}_t$, $\boldsymbol{b}_t$, $\boldsymbol{a}_t$ 이라고 하자. 입력층은 이들 상태 벡터를 연결한 벡터인 $[\boldsymbol{s}_t;\boldsymbol{b}_t;\boldsymbol{a}_t]$으로 표현된다. 다음으로, ReLU기반 은닉층에서는 이들 연결 벡터 $[\boldsymbol{s}_t;\boldsymbol{b}_t;\boldsymbol{a}_t]$에 다음 식 (1)을 적용하여 *파서 상태 표상* (parser state representation) $\boldsymbol{p}_t$을 얻는다.

$$\boldsymbol{p}_t = \max(\boldsymbol{W}[\boldsymbol{s}_t;\boldsymbol{b}_t;\boldsymbol{a}_t] + \boldsymbol{d}, \boldsymbol{0}) \quad (1)$$

식 (1)에서 $\boldsymbol{W}$와 $\boldsymbol{d}$은 ReLU은닉층을 위한 파라미터이다. 출력층은 $\boldsymbol{p}_t$에 다음 식 (2)에 적용하여 각 전이 액션에 대한 확률값을 얻는다.

$$\boldsymbol{q}_t = softmax(\boldsymbol{V}\boldsymbol{p}_t + \boldsymbol{g}) \quad (2)$$

식 (2)에서 $\boldsymbol{V}$와 $\boldsymbol{g}$은 출력층을 위한 파라미터이고, $\boldsymbol{q}_t$는 요소들의 합이 1인 확률벡터이다. 위의 전이 확률 중 최대가 되는 액션을 다음 전이로 취하여 S, B, A를 갱신하고 위의 과정을 반복한다.

3.2 스택 및 버퍼의 내용 갱신

표준 Stack LSTM기반 의존 파싱에서 가능한 전이 액션은 Arc-standard 방식의 REDUCE-LEFT, REDUCE-RIGHT, SHIFT으로 구성된다. 그림 2는 각 전이 액션 별로 스택 S과 버퍼 B의 정보가 갱신되는 과정을 요약해서 보여주고 있다. 스택과 버퍼 상의 각 요소는 (vector,symbol)

$Stack_t$	$Buffer_t$	Action	$Stack_{t+1}$	$Buffer_{t+1}$
$(\mathbf{u},u),(\mathbf{v},v),S$	B	REDUCE-RIGHT(r)	$(g_r(\mathbf{u},\mathbf{v}),u),S$	B
$(\mathbf{u},u),(\mathbf{v},v),S$	B	REDUCE-LEFT(r)	$(g_r(\mathbf{v},\mathbf{u}),v),S$	B
S	$(\mathbf{u},u),B$	SHIFT	$(\mathbf{u},u),S$	B

그림 2 전이 액션 별 스택 및 버퍼 정보의 갱신 과정 요약 [12]

의 형태인 $(\boldsymbol{u},u)$로 표현될 수 있으며,여기서 u는 단어나 부분트리의 root노드를 $\boldsymbol{u}$는 u에 대응되는 벡터 표현을 의미한다.

먼저, SHIFT의 경우에는 단순히 버퍼내의 Top 단어가 스택에 추가되는 것이므로 버퍼의 TOP노드의 상태 벡터를 스택에 그대로 push하면 된다.

다음, REDUCE-LEFT, REDUCE-RIGHT는 스택의 두개의 top노드들 간에 의존관계를 형성시키는 액션으로, 스택의 두 개의 top노드를 pop을 한 후에, 이들 노드의 트리들 간에 arc를 생성하여 병합된 트리를 만들고, 병합된 트리에 대한 표상 및 root노드를 스택 S에서 push하는 과정이다. 그림 2에서 $g_r(\boldsymbol{u},\boldsymbol{v})$은 부분트리들의 표상으로부터 병합트리의 표상을 유도하는 함수인데, r은 해당 액션의 의존관계를 $\boldsymbol{u}$는 head가 되는 부분트리의 표상을 $\boldsymbol{v}$를 dependent가 되는 부분트리의 표상을 가리킨다. $g_r(\boldsymbol{u},\boldsymbol{v})$을 위해서 [12]에서는 재귀적 신경망을 사용하였으며, 의존 관계 r에 대한 임베딩 벡터를 $\boldsymbol{r}$이라고 할 때 $g_r(\boldsymbol{u},\boldsymbol{v})$는 아래 식 (3)으로 정의된다.

$$g_r(\boldsymbol{u},\boldsymbol{v}) = \tanh(\boldsymbol{U}[\boldsymbol{u};\boldsymbol{v};\boldsymbol{r}]+\boldsymbol{e}) \quad (3)$$

이때 $\boldsymbol{U}$와 $\boldsymbol{e}$는 재귀적 신경망을 위한 추가 파라미터이다.

Stack LSTM에 기반한 의존 파싱에 필요한 기존 파라미터는 1) S, B, A를 위한 stack LSTM의 파라미터, 2) 단어, 액션, 의존 관계 임베딩 벡터 (lookup tables), 3) ReLU 은닉층의 파라미터 $\boldsymbol{W}$와 $\boldsymbol{d}$, 4) 출력층을 위한 파라미터 $\boldsymbol{V}$와 $\boldsymbol{g}$, 5)재귀적 신경망을 위한 파라미터 $\boldsymbol{U}$와 $\boldsymbol{e}$로 구성된다.

4. 음절과 형태소 결합을 이용한 단어 표상

[13]에서는 형태적으로 복잡한 언어를 위한 단어 표상으로 단어를 구성하는 문자열을 입력으로 받아, 문자 임베딩 벡터로부터 LSTM을 적용하여 단어의 고정 길이 벡터를 얻어내는 합성 방법을 제안하였다. 이를 한국어에 적용하기 위해서는 문자 처리 단위를 정의해야 한다.

4.1 음절-태그와 형태소 기반 단어 표상

[15]에서는 한국어 단어 표상을 위해 문자 처리 단위로 음절-태그와 형태소 단위의 사용을 제안하였다.

음절-태그 단위란 음절 정보과 음절의 품사 태그로 구성된 정보를, *형태소* 단위는 형태소 표층 정보과 형태소의 품사 태그로 구성된 정보이다. 예를 들어 "도서관에서" 단어에 대해 각각 음절-태그열과 형태소 열을 추출하면 다음과 같다.

- 음절-태그 기반: ["도/B-NNG", "서/I-NNG", "관/I-NNG", "에/B-JKB", "서/I-JKB"]
- 형태소 기반: ["도서관/NNG", "에서/JKB"]

기존 [15]의 문자만을 사용하는 것과는 달리, 음절-태그 의 경우에는 음절외에도 음절 품사 정보도 부가적으로 포함되어 있기 때문에, 음절 자체만을 활용할 때의 애매성을 줄일 수 있다. 또한, 형태소의 경우에는 음절보다 더 명확한 의미를 지녀 애매성을 더욱 감소시킬 수 있다.

4.2 음절-태그 및 형태소 결합 기반 단어 표상

본 논문에서는 음절-태그와 형태소-태그 기반 단어 표상을 조합하는 방법을 제안한다. 그림 4는 제안 단어 표상을 얻는 과정을 보여주고 있다.

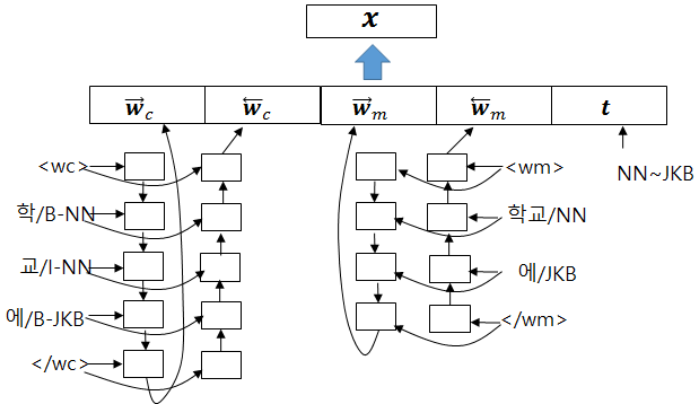

그림 4 음절-태그와 형태소 임베딩 벡터들의 결합을 통해 단어 표상을 얻는 과정

그림 4에서 보다시피 입력층은 음절-태그열과 형태소열 각각에 대해서 bidirectional LSTM을 적용하여 마지막 시점의 상태 벡터들과 단어 레벨의 태그 벡터를 연결하여 얻는다, 최종적으로, 단어 벡터는 이를 ReLU 층에 연결하여 다음 식 (4)를 적용하여 얻는다.

$$\boldsymbol{x} = max(\boldsymbol{C}[\overrightarrow{\boldsymbol{w}_c};\overleftarrow{\boldsymbol{w}_c};\overrightarrow{\boldsymbol{w}_m};\overleftarrow{\boldsymbol{w}_m};\boldsymbol{t}] + \boldsymbol{b}, \boldsymbol{0}) \quad (4)$$

식 (4)에서 $\boldsymbol{C}$와 $\boldsymbol{b}$는 단어 벡터를 합성하는 ReLU층에 필요한 추가 파라미터이다.

4.3 형태소 임베딩 학습

형태소 임베딩 (morpheme embedding)을 위해 본 논문에서는 GloVe [28] 사용하였다. 형태소 임베딩을 위한 학습 코퍼스로 한국어 Sejong학습코퍼스와에 블로그에서 수집된 문장들을 결합하여 약 1300만 문장을 구축하였다. 전처리로 Sejong 학습코퍼스로부터 학습된 CRF기반 형태소 분석기 [29-30] 를 적용하여,총 5억개의 형태소로 이루어진 코퍼스를 얻을 수 있었다. 이에 GloVe를 적용하여 총 약 70만개의 형태소에 대한 임베딩 벡터 (차원수=50)를 학습하였다.

5. 실험

실험 집합으로 기존 방법과 비교를 위해서 두 유형의 세종 코퍼스를 활용하였고, [16-17]에서 사용된 약 7만 3천 문장과 [11,15]의 약 6만 문장을 이용하였다. 편의를 위해 각각 *Sejong*, *Sejong Rigid* 셋이라 부르도록 한다. 본 실험에서 비교하고자 하는 방법들은 다음과 같다.

- **syll**: 음절 기반 단어 임베딩을 활용한 Stack LSTM 기반 의존 파싱 [13]
- **sylltag**: 음절-태그 기반 단어 표상 [15]
- **morpheme**: 형태소 기반 단어 표상 ([15]의 형태소-태그 방법에 대응). 형태소 lookup table은 4.3절의 GloVe 사전학습 결과로 초기화함.
- **sylltag+morpheme**: 제안 음절-태그 및 형태소 결합을 통한 단어 표상.

형태소 분석 결과로 4.3절에서 사용한 것과 동일한 Sejong 품사 부착 데이터에서 학습된 음절 기반 형태소 분석기 [29-30] 를 활용 하였다.

표 1은 Sejong Rigid셋에서 각 방법을 비교한 결과를 보여주고 있다. 동일셋을 사용한 benchmark 시스템으로 [11,15]의 성능도 함께 제시하였다. Stack LSTM을 사용한 기존 연구 [15]에서는 morpheme 단위의 성능이 sylltag에 비해 낮았으나, 본 연구에서는 sylltag보다 높은 성능을 보여주었다. 이러한 차이는 GloVe로 사전학습한 형태소 임베딩 벡터를 초기화로 인한 것으로, 여기서 약 0.5%의 정도의 성능 증가가 있었다. 제안 sylltag+morpheme방법은 추가로 0.1%의 성능 증가를 달성해 동일 셋에서 최고 성능인 [11]의 UAS를 뛰어넘었다. 특히 [11]의 경우에는 별도의 자질인 MI를 사용하여 90.37%의 UAS 성능을 보여주었으나, 본 연구는 별도의 자질을 사용하지 않고, 형태소와 음절, 태그 등 기본 자질만으로 90.44%의 UAS를 얻을 수 있었다.

표 1. Sejong Rigid데이터셋에 성능 비교 (자동 형태소 분석 결과를 활용한 것으로 [11,15]에 대응).

	UAS	LAS
이창기 [11] (no MI)	90.05%	87.87%
이창기 [11] (with MI)	90.37%	**88.17%**
morpheme ([15])	89.85%	87.33%
syll	90.10%	87.69%
sylltag	90.11%	87.70%
morpheme	90.33%	87.86%
sylltag+morpheme	**90.44%**	87.93%

표 2는 약 7만 문장의 Sejong데이셋에서 실험 결과이다. 정답형태소 분석 결과를 사용한 결과로, 비슷한 셋에서 실험한 [16,17]의 연구를 benchmark시스템으로 제시하였다. [16,17]은 딥 러닝 방식이 아닌 전통적인 기계 학습 방법을 활용하여 풍부한 자질셋을 바탕으로 높은 성능을 얻은 시스템들이다. 또한 트랜지션 방식에서 [11,16,17]은 backward를 사용하고 있으나 본 논문에서는 forward 방식에 기반을 두고 있다. Stack-LSTM 기반 의존 파싱 방법은 기초 자질만으로도 sylltag방법 적용시 benchmark시스템들의 성능을 뛰어넘을 수 있었으며, 제안 sylltag+morpheme 방법 적용 시 UAS 93.65%로 기존 최고 성능 [17]에 비해 약 0.5%의 성능 증가를 가져왔다.

표 2. Sejong데이터셋에 성능 비교 (정답 형태소 분석 결과를 활용한 것으로, [16,17]에 대응)

	UAS	LAS
임준호 [16] forward	92.58%	90.58%
임준호 [16] backward	92.85%	90.82%
나승훈 [17]	93.15%	-
sylltag	93.29%	91.15%
morpheme	93.30%	91.19%
sylltag+morpheme	**93.65%**	**91.57%**

6. 결론

본 논문에서는 Stack LSTM기반 한국어 의존 파싱을 위해 음절-태그, 형태소를 결합하는 단어 표상 방법을 제안하였다. 실험 결과, 제안 방법은 두 유형의 Sejong데이터셋에서 최고의 UAS성능을 달성하였다. 별도의 자질 없이, 형태소나 품사 정보 등의 기본 자질만으로 딥 러

닝의 표상 학습을 통해 우수한 파싱 성능을 이끌어낼 수 있음을 보여주고 있다.

감사의 글

본 연구는 미래창조과학부 및 정보통신기술진흥센터의 정보통신·방송 연구개발사업의 일환으로 수행하였음. [R7119-16-1001, 지식증강형 실시간 동시통역 원천기술 개발]

참고문헌

[1] Bengio, Y. “Learning Deep Architectures for AI,” Foundations and Trends in Machine Learning, 2(1), 2009

[2] Hinton, G. E., Osindero, S., & Teh, Y. “A fast learning algorithm for deep belief nets. Neural Computation, vol. 18, 2006

[3] Collobert, R., & Weston, J. “A Unified Architecture for Natural Language Processing : Deep Neural Networks with Multitask Learning,” ICML 2008

[4] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. “Natural Language Processing (Almost) from Scratch,” Journal of Machine Learning Research, 2012

[5] Socher, R., Chiung, C., Lin, -Yu, Ng, A. Y., & Manning, C. D. “Parsing Natural Scenes and Natural Language with Recursive Neural Networks,” ICML 2011

[6] Hochreiter, S., & Urgen Schmidhuber, “Long Short-Term Memory,” Neural Computation, 9(8), 1997

[7] Graves, A. “Generating sequences with recurrent neural networks,” arXiv Preprint arXiv:1308.0850., 2013

[8] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation,” EMNLP 2014

[9] Sutskever, I., Vinyals, O., & Le, Q. V. “Sequence to Sequence Learning with Neural Networks,” NIPS 2014

[10] Chen, D., & Manning, C. D., “A Fast and Accurate Dependency Parser using Neural Networks,” In EMNLP ’14, 2014

[11] 이창기, 김준석, 김정희., “딥 러닝을 이용한 한국어 의존 구문 분석,” HCLT 2014

[12] C. Dyer, M. Ballesteros, W. Ling, A. Matthews, N. A. Smith, "Transition-Based Dependency Parsing with Stack Long Short-Term Memory," ACL 2015

[13] M. Ballesteros, C. Dyer, N. A. Smith, “Improved Transition-Based Parsing by Modeling Characters instead of Words with LSTMs,” EMNLP 2015

[14] 이건일, 이종혁, “순환 신경망을 이용한 전이 기반 한국어 의존 구문 분석,” 정보과학회 컴퓨팅의 실제 논문지 제21권 제8호, 2015

[15] 나승훈, 김강일, 김영길, “Stack LSTM을 이용한 전이 기반 한국어 의존 파싱,” KCC 2016

[16] 임준호, 윤여찬, 배용진, 임수종, 김현기, 이규철, “지배소 후위 제약을 적용한 트랜지션 시스템 기반 한국어 의존 파싱 모델,” HCLT 2014

[17] 나승훈, 정상근, “그래프 기반 및 전이 기반 방법의 결합 모델을 이용한 한국어 의존 파서,” KCC 2014

[18] McDonald, R., & Pereira, F. “Online Learning of Approximate Dependency Parsing Algorithms,” EACL 2006

[19] Nivre. “An Efficient Algorithm for Projective Dependency Parsing,” IWPT 2003

[20] T. Kudo and Y. Matsumoto, “Japanese Dependency Analysis using Cascaded Chunking,” CONLL 2002

[21] Zhang Y. & Clark S. “A Tale of Two Parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search,” EMNLP 2008

[22] Mcdonald, R., & Nivre, J. “Analyzing and Integrating Dependency Parsers,” Computational Linguistics, 2011

[23] 임수종, 김영태, 나동열, “자질 가중치의 기계학습에 기반한 한국어 의존파싱,” 정보과학회논문지, 소프트웨어 및 응용 제 38권 제 4호, 2011

[24] 나승훈, "3차 의존 파싱에 기반한 한국어 구문 분석," HCLT 2014

[25] 오진영, 차정원, “키 어절을 이용한 새로운 한국어 구문 분석,” 정보과학회논문지, 소프트웨어 및 응용 제40권 제10호, 2013

[26] 최맹식, 정석원, 김학수, “CRFs를 이용한 의존 구조 분석 및 의존 관계명 부착,” 정보과학회논문지 : 소프트웨어 및 응용 제 41 권 제 4 호, 2014

[27] 나승훈, “String-to-Dependency SMT 모델을 이용한 구 기반 의존 파싱,” KCC 2015

[28] J. Pennington, R. Socher, C. D. Manning, “Glove: Global Vectors for Word Representation,” EMNLP2014

[29] 나승훈, 양성일, 김창현, 권오욱, 김영길, “CRF에 기반한 한국어 형태소 분할 및 품사 태깅,” HCLT 2012

[30] 이창기, “Structural SVM을 이용한 한국어 띄어쓰기 및 품사 태깅 결합 모델,” KCC 2013