동적 오라클을 이용한 한국어 의존 구문분석*

이경호○, 이공주
충남대학교 전자전파정보통신공학과, 전파정보통신공학과
gyholee@gmail.com, kjoolee@cnu.ac.kr

Korean Dependency Parsing using Dynamic Oracle

Gyoung Ho Lee○, Kong Joo Lee
Chungnam National University

요 약

구문분석은 자연언어처리의 오랜 관심 분야로 다양한 접근방법과 알고리즘이 시도되어 계속 발전하고 있다. 하지만 기존의 접근방법은, 학습단계에서는 정답으로부터 추출된 이전 정보를 사용하고 평가 단계에서는 예측으로 이루어진 정보를 활용한다는 근본적인 차이가 있다. 이러한 차이를 극복하기 위한 다양한 시도가 있었고 그 중 동적 오라클 기법이 합리적인 시간 증가와 성능향상을 보였다. 본 연구에서는 이러한 동적 오라클 기법을 한국어 구문분석에 적용하였다. 동적 오라클 기법을 한국어에 적용할 때 고려해야하는 부분에 대해 탐구하고 실험을 통해 동적 오라클 기법을 한국어 구문분석에 적용하여 결과를 살펴보았다.

주제어: 구문분석, 동적 오라클, 인공신경망, 의존문법

1. 서론

구문분석은 자연언어 처리 분야의 오랜 관심 분야로 여러 접근 방법과 알고리즘이 도입되어 계속 발전해나가고 있다. 최근에는 포인터 네트워크를 활용한 구문분석 모델이나 순환신경망(Recurrent Neural Network)을 이용한 구문분석기 등, 인공신경망을 활용한 다양한 모델이 개발되고 있고 좋은 성능을 보이고 있다[1][2][3]. 한국어는 어순 배열의 자유도가 높고 문장 성분 생략이 빈번한 특성이 있다. 그렇기 때문에 이런 특성에 적합하다고 알려진 의존문법 구문분석이 한국어 구문분석의 주된 연구 대상이 되어 왔다[3, 4].

기존의 의존문법 구문분석 알고리즘은 주로 의존 트리로부터 구문분석 진행 상태(state)와 이 상태에서 어떤 행동(action)을 해야 하는지를 추출하고 상태로부터 행동을 결정하는 모델을 학습한다.

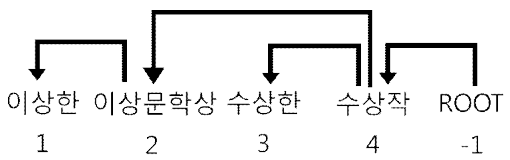

그림 1. 구문트리 예

그림 1과 같은 구문트리의 구문분석은 표 1과 같이 진행된다. 본 연구에서는 한국어 구문분석의 특성인 지배소 후위 원칙과 투사성 원칙을 활용하여 backward 방식의 Arc-Eager 알고리즘[5]으로 구문분석을 진행하였다[1].

표 1. 구문분석 예

Step	Stack	Buffer	Action
0	-1	4, 3, 2, 1	Right-Arc
1	-1, 4	3, 2, 1	Right-Arc
2	-1, 4, 3	2, 1	Reduce
3	-1, 4	2, 1	Right-Arc
4	-1, 4, 2	1	Right-Arc
5	-1, 4, 2, 1		

구문분석은 Stack과 Buffer를 초기화 하는 것으로 시작된다. Stack에는 이미 자신의 지배소를 찾은 단어들이 들어있고 Buffer에는 지배소를 찾을 단어들이 위치한다. Stack의 최상단 단어(표 1의 Stack 열에서 가장 오른쪽)는 Buffer의 최하단 단어(표 1의 Buffer 열의 가장 왼쪽)의 지배소 후보이다. 이 두 단어와 Stack, Buffer의 상태를 이용해 다음에 취할 행동을 결정한다(Action 열). 학습과정에서는 정답트리를 이용하여 두 단어의 관계가 지배소-피지배소 관계인지 알 수 있기 때문에 이 둘을 연결할지(Right-Arc), 현재의 지배소 후보를 Stack에서 버리고 다음 단어로 넘어갈지(Reduce)를 결정할 수 있다. 이러한 행동을 정적 오라클(Static oracle)이라 한다.

학습단계에서는 오라클을 통한 전이를 수행하면서 상태와 행동을 수집하고, 수집된 상태와 행동을 이용해 상태가 주어지면 올바른 행동을 예측하도록 기계학습 모델을 학습시킨다. 실제 구문분석을 수행할 때는 오라클을 알 수 없으므로 학습된 모델에서 예측한 행동에 따라 구문분석을 진행한다.

* 이 논문은 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. NRF-2015R1C1A2A01051685)

하지만, 학습과정에서 도달하는 특정 상태 c와 실제 적용 단계의 상태c'은 근본적으로 차이가 있다. 학습과정의 상태 c는 정답으로부터 추출된 올바른 상태와 행동을 통해 도달한 결과이다. 하지만 c'의 경우 모델로부터 예측된 행동을 따라온 상태이기 때문에 올바른 경로를 통해 도달한 상태라는 것을 보장할 수 없다.

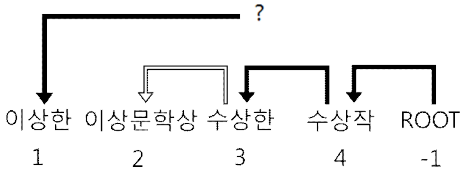

그림 2 모델을 이용한 구문분석 예

그림 2는 모델을 이용한 구문분석을 가정한 예이다. 표 1과 같이 정답 트리를 알고 있다면 '이상문학상'과 '수상한'의 관계가 피지배소-지배소관계가 될 수 없는 것을 알고 $Reduce$를 수행했을 것이다. 하지만 모델의 예측으로 구문분석이 진행된다면, 그림 2와 같이 잘못된 관계를 예측 할 수 있다. 이러한 경우 그 다음 단어인 '이상한'과 '이상문학상'의 관계에도 변화가 생긴다. 만약 학습 단계에서 이런 문장을 보았다면, '수상작'을 지배소로 가지고 있는 '이상문학상'과 '이상한'의 관계가 지배소-피지배소 관계인 것을 학습하였을 것이다. 하지만 예측과정에서는, '이상문학상'의 지배소를 잘못 예측한 결과로, '수상한'을 지배소로 가지는 '이상문학상'과 '이상한'의 관계를 추론해야 하지만 학습단계에서 이러한 관계를 학습하지 않았기 때문에 문제가 될 수 있다.

이러한 문제는 학습단계에서 정답트리로부터 추출된 정적 오라클 경로뿐만 아니라 다양한 경로를 탐색해보고 그것에 대해 학습하도록 함으로써 완화할 수 있다. 그림 2와 같은 경우, 비록 3-2번 단어의 관계는 틀렸지만, 이 상태에서 2-1번 단어의 관계를 학습 할 수 있다면 3-2 단어에서 발생한 에러가 다음단계로 전파되는 것을 최소화 할 수 있다. 하지만 기존의 정적 오라클 방법으로는 이러한 작업을 수행하기 어렵기 때문에 다른 접근법이 필요하다.

본 연구에서는 이러한 문제점의 해결 방법으로 개발된 동적 오라클(Dynamic Oracle)을 한국어 구문분석에 적용해보고자 한다. 동적 오라클을 이용하면 특정 상태에서 적용할 행동을 정답트리를 통해 미리 결정해 놓지 않고 해당 상태에서 적용 가능 하면서 적용하였을 때 남아 있는 정답 트리의 훼손을 최소화하는 행동을 구할 수 있다. 이러한 행동을 통해 지금까지 잘못된 경로를 왔다고 하더라도 다음에 선택하는 행동은 에러의 전파를 막고 현재 상태에서 가능한 최선의 선택을 할 수 있도록 해준다.

이러한 시도 중 하나로 강화학습을 이용한 구문분석연구가 진행된 바 있다[6, 7]. 이들 연구에서는 상태에 따른 행동을 결정하는 정책(policy)을 설정한다. 이 정책에 따라서 행동의 확률을 구하고 이 확률에 따라 트리를 완성해 나간다. 트리가 완성되면, 트리의 완성도에 따라 보상(reward)을 받게 된다. 강화학습은 이 보상이 커지는 방향으로 정책을 발전시켜나간다. 이때 확률적으로 새로운 경로가 개척될 수 있는데 이러한 경로의 보상값을 통해 이전의 경로 외에 새로운 경로가 더 좋은지 나쁜지 탐색하면서 가장 높은 보상을 받을 수 있는 경로를 찾아간다. 하지만 강화학습의 경우, 여러 상태와 행동에 대한 다양한 탐색을 수행해야 하므로 학습 속도가 느린 단점이 있다.

동적 오라클은 학습단계에서의 오라클 설정에만 영향을 미치며 실제 적용에는 기존과 같이 탐욕적 알고리즘(Greedy parsing)으로 적용가능하다. 그렇기 때문에 기존의 구문분석 알고리즘과 학습 모델에 대한 적은 변화로도 적용가능하다. 또한 구문분석에서 발생 가능한 상태와 오류들에 대한 탐색을 수행하기 때문에 강화학습을 통한 학습보다 더 빠른 학습을 수행 할 수 있다.

본 논문의 구성은 다음과 같다. 2장에서는 동적 오라클과 관련된 구문분석 관련 연구에 대해 소개하고 3장에서는 이를 한국어 구문분석에 적용하는 방법과 예측 모델에 대해 설명한다. 그리고 4장에서는 한국어 구문분석에 적용한 실험 결과와 이전의 구문분석 연구들과 비교를 하고 마지막 5장에서 이 연구의 결론을 맺는다.

2 관련 연구

본 연구에서는 동적 오라클을 이용한 구문분석을 한국어 구문분석에 적용하였다. [5]에서 동적 오라클 방법에 대해 처음 제안하였다. 이 논문에서는 동적 오라클의 필요성과 동적 오라클의 정의 방법에 대해 제안하였다. 이를 학습 할 수 있는 Online 학습 절차에 대해 제안하고 이를 Arc-eager 구문분석 방법에 적용하였다. CoNLL 2007데이터에 이를 적용한 실험 결과, 영어의 경우 정적 오라클을 사용 할 때와 동적 오라클을 사용할 때 UAS(Unlabeled Attachment Scores) 점수가 86.24에서 88.81 증가함으로써 동적 오라클이 성능향상에 도움이 되는 것을 증명하였다. [8]의 연구에서는 앞선 [5]연구를 발전시켜 Arc-Eager, Arc-hybrid, Easy-First등의 구문분석 알고리즘에 동적 오라클을 적용하였다. 같은 데이터에 대해 ARC-Eager: 88.69, arc-hybrid: 88.69, easy-first: 89.41의 성능향상을 보였다.

또한 이 논문에서는 arc-standard 알고리즘이 아크 분해(Arc Decomposition)되지 않음을 증명하여 다른 알고리즘에 비해 동적 오라클 적용의 효율이 낮음을 증명하였다. [9] 논문에서는 딥러닝 기법을 이용한 Stack LSTM[10] 알고리즘에 동적 오라클을 적용하였다. 기존의 정적 오라클을 통해 학습한 Stack LSTM보다 동적 오라클을 적용하였을 때 최대 93.56의 성능을 나타냄으로 써 최근의 딥러닝을 이용한 파서와 동적 오라클의 사용이 성능향상을 이끌어 낼 수 있음을 나타내었다.

본 연구에서는 앞서 연구들이 영어나 다른 언어에 적용한 바와 같이 동적 오라클 방법을 한국어 구문분석에 적용해보고자 한다. [9]의 논문과 같이 arc-eager를 사용하며 이 연구에서 적용한 Stack LSTM을 단순화하여 한국어 구문분석을 수행하였다.

3. 본 론

3.1 동적 오라클

본 연구에서는 구문분석의 학습단계와 적용단계에서 발생하는 차이를 줄이기 위한 방법으로 동적 오라클을 적용하였다. 구문분석의 현재 상태(c)에서 전이를 통해 도달 가능한 완성된 구문트리(G) 중, 정답 트리와의 손실($loss$)이 가장 적은 트리를 식 1과 같이 나타낸다[5].

$$\min_{G:c\rightarrow G} Loss(G, G_{gold}) \quad \cdots (1)$$

이때 손실은 정답 트리(G_{gold})의 arc들과 임의의 트리 G의 arc의 차집합 개수를 의미한다.

[5]의 연구에서 상태 c에서 전이 행동 t를 취했을 때 발생하는 비용함수를 다음과 같이 정의한다.

$$COST(t;\ c,\ G_{gold}) = [\min_{G:t(c)\rightarrow G} Loss(G, G_{gold})] - [\min_{G:c\rightarrow G} Loss(G, G_{gold})] \quad \ldots (2)$$

즉, c에서 t를 취할 때의 비용은 현재 상태에서 도달 가능한 가장 좋은 트리(정답 트리와의 차이가 크지 않은 트리)와 상태 t를 취한 새로운 상태에서 선택한 가장 좋은 트리와의 차이를 의미한다. [5] 연구에서 이 c에서 t를 수행할 때 비용(cost)이 0이 되는 t가 존재함을 증명하였으며 이러한 t들을 동적 오라클로 정의하였다. 구문분석의 어느 단계에서든, 만약 정답 트리를 만드는 길에서 벗어났더라도 동적 오라클을 따라간다면 현재 상태에서 도달 가능한 가장 적절한 트리를 생성할 수 있게 된다. [5]와 [8] 연구에서는 c와 정답트리를 이용하여 동적 오라클을 구하는 규칙에 대해 정의하였다. 이들 연구에서 Arc-eager의 4 종류 행동에 대한 규칙을 정의했지만 한국어 구문분석 알고리즘에서는 2 종류(Right－Arc$_{label}$, Reduce) 행동에 대한 규칙만 필요하다.

현재 상태 c를 $S=\sigma|s$, $B=b|\beta$와 같이 표시할 수 있다. 이때 S는 Stack, s는 Stack 최상단, σ는 Sack의 나머지 단어들을 의미하고 B는 Buffer, b는 Buffer의 최하단, β는 Buffer 나머지 부분을 의미한다. 전이 행동 Reduce는 s를 S에서 제거하는 것이다. c에서 이 행동을 취하면 B의 단어 중 s를 지배소로 하는 단어들은 지배소를 찾을 수 없게 된다. 그렇기 때문에 Reduce의 비용은 정답 트리에서 (s,k) 아크의 개수(k: B의 단어 중 s를 지배소로 가지는 단어)이다. Right－Arc$_{label}$는, b의 지배소를 s로 결정하는 것이다. 이렇게 되면 b는 더 이상 σ에서 자신의 지배소를 찾을 수 없다. 그렇기 때문에 정답트리에서 b의 지배소 k가 σ에 있다면 cost 값은 1이 된다. 만약, 앞서 단계에서 잘못된 예측으로 k가 이미 S에서 제거 되었다면, b의 지배소로 어떤 단어로 선택해도 손실은 달라지지 않는다. 그렇기 때문에 이러한 경우 비용은 0이다. 또한 b의 k가 s인 경우 정답트리에 있는 행동이기 때문에 비용에 영향을 주지 않아 비용은 0이다. 본 연구에서는 지배소 후위 원칙과 투사성 원칙에 따르기 때문에 b의 지배소를 β에서 찾는 규칙은 적용하지 않았다.

3.2 모델과 자질

상태로부터 행동을 결정하는 분류기로 [9]와 [10]에서 사용한 Stack LSTM을 단순화한 인공신경망 모델을 사용하였다. 본 연구에서 사용한 모델은 그림 3과 같다.

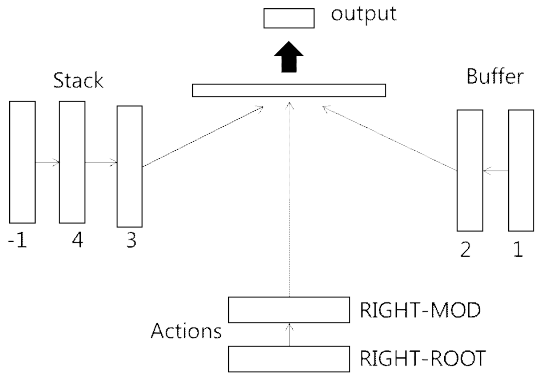

그림 3 분류기 모델

이 모델은 3개의 순환신경망(RNN) 레이어와 1개의 전방향 인공신경망(FNN) 레이어로 구성된다. 3개의 RNN 레이어는 Stack과 Buffer, 그리고 지금까지 예측했던 전이 행동들을 표현하는 벡터를 각각 생성한다. RNN을 통해 각각의 벡터로 표현된 Stack 정보와 Buffer정보, 그리고 행동들의 정보는 하나의 벡터로 연결되어 FNN의 입력으로 사용된다. Stack과 Buffer의 RNN 입력은 각각의 어절을 표현하는 벡터들이다.

하나의 어절은 1) 어절의 첫 번째 형태소와 태그 2)어절의 두 번째 형태소와 태그 3) 어절의 끝에서 두 번째 형태소와 태그 4) 어절의 마지막 형태소와 태그를 이용하여 표현된다. 각 형태소와 태그는 50차원의 벡터로 표현되고 어절은 이들 벡터를 연결하여 400차원의 벡터로 표현되다. 전이 행동은 50차원의 벡터로 표현한다. 이들은 Stack, Buffer의 입력 순서와 모델이 예측한 순서대로 RNN 레이어에 입력된다. 각각의 어절과 전이 행동을 입력받는 RNN은 400차원의 출력을 가지며 2개의 layer를 가진 GRU를 사용하였다. 이들 RNN의 출력은 하나의 벡터로 연결되어 1200 차원의 히든 레이어를 가진 FNN에 입력된다. 한국어 구문분석의 경우, Arc-Eager 알고리즘을 사용하면서 backward로 구문분석을 수행할 경우 Right-Arc와 Reduce만을 이용하여 구문분석을 수행할 수 있는 장점이 있다. 그렇기 때문에 이 모델의 출력은 $Reduce$와, Right－Arc$_{label}$들을 출력으로 가진다.

3.3 학습 알고리즘

본 연구에서 사용한 학습 알고리즘은 표 2와 같다. 이 알고리즘은 학습데이터에 있는 문장 W과 그 문장의 정답트리 T를 이용하여 학습을 진행한다. 먼저 W를 이용하여 구문분석의 상태를 초기화 한다(c). c에는 현재 진행 중인 Stack과 Buffer 그리고 지금까지 채결된 arc정보를 담고 있다. c의 상태에 따라 구문분석의 진행 여부를

판단한다. 구문분석은 먼저 해당 상태에서 선택 가능한 행동들(lt)의 수집한다(line 5). 그리고 현재 상태와 lt를 이용하여 3.1절에서 설명한 동적 오라클과 현재 모델의 각 전이 행동에 대한 예측 확률을 수집한다(line 6, 7). 알고리즘은 학습단계에서 구문분석이 동적 오라클의 경로를 따라가다가 이따금씩 새로운 경로를 시도해보도록 설계되었다. 다음번 전이를 위한 행동 nt는 line 9와 같이 정의된다. 구문분석은 80%의 확률로 동적 오라클의 경로를 따라간다. 나머지 경우, lt중 하나를 선택하여 진행한다. 학습이 진행됨에 따라 모델의 예측 정확도가 높아지면서 모델로부터 가장 높은 점수를 받는 행동이 수렴된다. 이런 경우 새로운 경로를 개척할 수 있는 기회가 줄어든다. 이를 방지하기 위해 일정한 확률로 lt 중 임의의 하나의 행동을 선택해 구문분석을 진행하도록 하였다.

구문분석을 수행하는 동안 상태와 그 상태에 대한 동적 오라클을 수집한다(line 10). 수집된 상태와 오라클을 이용하여 일정 주기마다 모델을 학습시켜 준다(line 12). 상황에 따라 동적 오라클은 1개 이상의 오라클을 가질 수 있다. 이를 학습하기 위해 multilabel soft-margin loss[12]를 이용하였다.

표 2. 학습 알고리즘

1: $FeatureList = \{\varnothing\}$
   $\mathrm{model} = ModelInitial()$
2: $for\ (W,\ T) \in TrainingData:$
3: $\quad c \leftarrow Initial(W)$
4: $\quad while$ not Terminal(c):
5: $\quad\quad lt \leftarrow LegalTransition(c)$
6: $\quad\quad dynamic \leftarrow Dynamic(c,\ lt)$
7: $\quad\quad actionProb \leftarrow \mathrm{model}(c)$
8: $\quad\quad \alpha = random(),\ \beta = random()$
9: $\quad\quad nt = \begin{cases} choice(dynamic) & \text{if } \alpha < 0.8 \\ \max(lt,\ actionProb) & \text{if } \alpha \geq 0.8 \text{ and } \beta < 0.8 \\ choice(lt) & else \end{cases}$
10: $\quad\quad FeatureList \leftarrow \{\mathrm{c,dynamic}\} \cup \mathrm{FeatureList}$
11: $\quad\quad Next(c,\ nt)$
12: $\quad\quad for\ each\ every\ N\ step\ do:$
$\quad\quad\quad Train(\mathrm{model},\ FeatureList)$

4. 실험

동적 오라클을 한국어 구문분석에 적용하였을 때의 효과를 알아보기 위한 실험을 수행하였다. 실험에는 21세기 세종[13]의 구구조 구문분석 트리를 의존 구조 구문분석 트리로 변환한 결과를 사용하였다. 그리고 용언들 사이의 관계를 [14]의 참조하여 재조정하였다. 변환과정 후 2개 이상의 노드를 가진 트리를 수집하여 총 57,912개의 트리를 얻었고 이를 9:1로 나누어 평가와 학습과 평가에 각각 52,120, 5,792개의 트리를 사용하였다.

표 3은 본 연구의 베이스라인과 이전 연구, 그리고 동적 오라클을 사용한 모델의 구문분석 성능 표이다.

표 3. 성능 비교

Algorithms	UAS	LAS
Baseline	90.65	88.97
Dynamic Oracle	90.97	89.30
Pointer Network[2]	91.65	89.34
Stack LSTM[3]	90.83	88.49

표 3은 평가 데이터의 전체 [피지배소, 전이 행동, 지배소] 목록 중, 피지배소의 지배소를 맞게 찾은 비율(unlabeled attachment score, UAS)와 전이 행동과 지배소까지 맞게 찾은 경우(labeled attachment score, LAS)비율을 백분율로 나타낸 결과이다. 표 3의 Baseline은 동적 오라클을 적용한 모델과 같은 모델에서 정적 오라클을 사용한 결과이다. Pointer Network와 Stack LSTM은 각각 구문분석에 딥러닝 알고리즘을 적용한 모델로 후자는 전이기반 알고리즘을 기반으로 한다. 평가 결과, 동적 오라클을 사용하였을 때, 정적 오라클을 사용한 결과보다 좀 더 높은 결과를 나타내었다.

표 4는 문장의 어절 수가 2~10개, 10~15개, 15~20개인 문장들을 수집하고 수집된 문장들에 대한 정적 오라클과 동적 오라클의 UAS 차이를 비교한 표이다.

표 4. 어절 길이별 UAS 점수

어절 길이	문장 수	정적오라클	동적 오라클	차이
2~10	2,725	95.10	95.21	0.11
10~15	1,363	92.17	92.34	0.17
15~20	759	89.71	90.11	0.40

표 4에서 문장의 길이가 길어질수록 정적오라클과 동적오라클의 성능 차이가 벌어지는 것을 볼 수 있다. 앞에서 발생한 오류의 영향을 줄여주는 동적오라클의 효과가 긴 문장일수록 발휘되기 유리하기 때문에, 문장의 길이가 길어질수록 두 방법의 성능차이가 벌어지는 것으로 생각된다.

5. 결론

본 연구에서는 구문분석의 학습단계와 적용단계의 차이로부터 발생할 수 있는 문제점을 줄여 줄 수 있는 동적 오라클에 대해 탐구하고 이를 한국어 구문분석에 적용하기 위한 방법을 연구하였다. 연구 결과 동적 오라클을 사용했을 때 정적 오라클 사용보다 더 나은 성능을 나타냄을 알 수 있다. 이러한 결과는 기존의 정적 오라클이 탐색하지 못한 영역을 동적 오라클을 통해 탐색 할 수 있기 때문에 적용단계에서 발생하는 에러 상황에 대한 강건한 대처가 가능했기 때문으로 생각된다. 향후연구로 동적 오라클을 이용하면서 더 다양한 경로를 효율적으로 탐색 할 수 있도록 강화학습이나 모방학습 등을

한국어 구문분석에 적용해볼 계획이다.

참고 문헌

[1] 이창기, 김준석, and 김정희. "딥 러닝을 이용한 한국어 의존 구문분석." 제 26 회 한글 및 한국어 정보처리 학술대회 (2014): 87-91.

[2] 박천음, and 이창기. "멀티 태스크 학습 기반 포인터 네트워크를 이용한 한국어 의존 구문분석." 한국정보과학회 학술발표논문집 (2016): 440-442.

[3] 나승훈, et al. "순환 컨트롤러를 이용한 Stack LSTM 기반 한국어 의존 파싱." 한국정보과학회 학술발표논문집 (2016): 446-448.

[4] 이건일, and 이종혁. "인공 신경망을 이용한 형태소 기반 한국 H 의존 구문분석.", 동계학술발표회논문집 (2014)

[5] Goldberg, Yoav, and Joakim Nivre. "A Dynamic Oracle for Arc-Eager Dependency Parsing." COLING. 2012.

[6] Zhang, Lidan, and Kwok Ping Chan. "Dependency parsing with energy-based reinforcement learning." Proceedings of the 11th International Conference on Parsing Technologies. Association for Computational Linguistics, 2009.

[7] Lê, Minh, and Antske Fokkens. "Tackling Error Propagation through Reinforcement Learning: A Case of Greedy Dependency Parsing." arXiv preprint arXiv:1702.06794 (2017).

[8] Goldberg, Yoav, and Joakim Nivre. "Training deterministic parsers with non-deterministic oracles." Transactions of the association for Computational Linguistics 1 (2013): 403-414.

[9] Dyer, Chris, et al. "Transition-based dependency parsing with Stack long short-term memory." arXiv preprint arXiv:1505.08075 (2015).

[10] Dyer, Chris, et al. "Transition-based dependency parsing with Stack long short-term memory." arXiv preprint arXiv:1505.08075 (2015).

[11] Chen, Danqi, and Christopher Manning. "A fast and accurate dependency parser using neural networks." Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.

[12] Zanoci, C., & Andress, J. Exploring CNN-RNN Architectures for Multilabel Classification of the Amazon.

[13] 국립국어원. 21세기세종계획. 2012.

[14] 의존 구문분석 말뭉치 구축을 위한 의존 관계 태그 세트 및 의존 관계 설정 방법, 정보통신단체표준, TTA, 2015