Sequence-to-sequence 모델을 이용한 한국어 구구조 구문 분석

황현선○, 이창기
강원대학교
{hhs4322, leeck}@kangwon.ac.kr

Korean phrase structure parsing using sequence-to-sequence learning

Hyunsun Hwang○, Changki Lee
Kangwon National University

요 약

Sequence-to-sequence 모델은 입력열을 길이가 다른 출력열로 변환하는 모델로, 단일 신경망 구조만을 사용하는 End-to-end 방식의 모델이다. 본 논문에서는 Sequence-to-sequence 모델을 한국어 구구조 구문 분석에 적용한다. 이를 위해 구구조 구문 트리를 괄호와 구문 태그 및 어절로 이루어진 출력열의 형태로 만들고 어절들을 단일 기호 'XX'로 치환하여 출력 단어 사전의 수를 줄였다. 그리고 최근 기계번역의 성능을 높이기 위해 연구된 Attention mechanism과 Input-feeding을 적용하였다. 실험 결과, 세종말뭉치의 구구조 구문 분석 데이터에 대해 기존의 연구보다 높은 F1 89.03%의 성능을 보였다.

주제어: Deep Learning, 구구조 구문분석, Sequence-to-sequence learning, Input-feeding

1. 서론

자연어처리의 과정 중 하나인 구문 분석은 문장의 구조를 분석하는 방법으로 구구조 구문 분석과 의존 구문 분석이 사용된다. 그러나 구구조 구문 분석은 한국어 특성상 난이도가 높고 복잡도가 $O(n^3)$이상으로 높아 의존 구문 분석이 주로 연구되고 있다[1,2,3].

최근 기계학습 알고리즘 중 하나인 딥 러닝(Deep Learning)을 자연어처리에 적용하는 연구가 많이 진행되고 있다[3,4]. 그 중 Sequence-to-sequence 모델은 입력열을 길이가 다른 출력열로 변환하는 모델로, End-to-end 방식의 신경망 구조를 사용하는 Neural Machine Translation(NMT) 모델이 대표적다[5,6]. 최근 연구되는 NMT 모델은 Attention mechanism과 Input-feeding 등의 연구로 기존의 기계번역 모델들 보다 높은 성능을 보이고 있다[7,8,9].

본 논문에서는 다양한 자연어처리 분야에 적용되고 있는 Sequence-to-sequence 모델[10,11]을 한국어 구구조 구문 분석에 적용하여 규칙이나 자질 튜닝 없이 기존의 연구보다 높은 성능을 보인다.

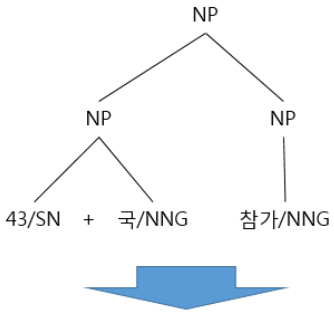
2. 관련 연구

한국어 구구조 구문 분석은 주로 확률을 이용한 방법들이 연구되었고[12], 최근에는 영어 구구조 구문 분석기를 이용하여 한국어 구구조 구문 분석을 시도한 연구도 있었다[13].

최근 영어 구구조 구문 분석에 Sequence-to-sequence 모델을 사용하여 기존의 영어 구구조 구문 분석보다 높은 성능을 보인 연구가 진행 되었다[10].

본 논문에서는 Sequence-to-sequence 모델을 한국어 구구조 구문 분석에 적용하고 Attention mechanism과 Input-feeding등의 기술을 적용하여 기존 연구보다 높은 성능을 낼 수 있음을 보인다.

3. Sequence-to-sequence 모델을 이용한 한국어 구구조 구문 분석



(NP (NP 43/SN + 국/NNG) (NP 참가/NNG))

그림1. 한국어 구구조 구문 분석의 예시

구구조 구문 분석은 그림 1의 상단과 같이 문장을 트리 형태로 분석을 하게 되며, 실제 데이터는 그림 1의 하단과 같이 구문 분석 태그가 포함된 괄호를 이용하여 하나의 열로서 표현하게 된다. 본 논문에서는 Sequence-to-sequence 모델에 맞추어 다음과 같이 구구조 구문 분석 시스템을 설계 하였다.

3.1 입력 및 출력 설계

입력 예시 1	43/SN 국/NNG <sp> 참가/NNG
입력 예시 2	4 3 <SN> 국 <NNG> <sp> 참 가 <NNG>

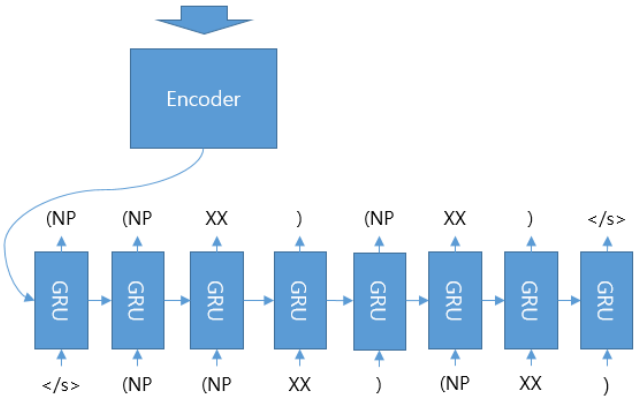


그림 2. Sequence-to-sequence 모델을 이용한 구문 분석

그림 2는 구구조 구문 분석을 위한 Sequence-to-sequence 모델의 입력과 출력을 나타낸다. 세종 구문분석 말뭉치의 구의 최소 단위는 하나의 어절이기 때문에 구문 분석 결과의 어절을 'XX'로 바꾸어 입력 어절과 1:1 매칭이 되도록 하였다. 입력은 정답 형태소 분석 결과를 사용하였고 어절의 구분을 위해서 '<sp>'(띄어쓰기)를 추가하였다(그림 2의 입력 예시 1). 추가적으로 입력을 형태소 단위로 설계할 시 발생하는 Out-of-vocabulary(OOV) 문제를 극복하기 위해 형태소를 형태소의 음절과 해당 형태소의 품사태그 및 띄어쓰기 정보를 입력으로 사용하는 모델도 설계 하였다(그림 2의 입력 예시 2).

3.2 Attention mechanism과 Input-feeding을 적용한 Sequence-to-sequence 모델

Sequence-to-sequence 모델은 신경망을 이용하여 P(y|x)를 직접 최적화하는 모델로(x는 입력 문장 열, y는 출력 태그 열), 본 논문의 인코더에 사용한 Gated Recurrent Unit(GRU)[6]의 식은 다음과 같다.

$$z = \sigma(W_z E_{src}(x_t) + U_z \vec{h}_{t-1} + b_z)$$

$$r = \sigma(W_r E_{src}(x_t) + U_r \vec{h}_{t-1} + b_r)$$

$$m = f(W_m E_{src}(x_t) + U_m(\vec{h}_{t-1} \odot r) + b_m)$$

$$\vec{h}_t = (1 - z) \odot \vec{h}_{t-1} + z \odot m$$

$E_{src}(x_t)$는 t번째의 입력 단어 x_t의 입력언어의 Word embedding이며 h_t는 t번째의 히든레이어를 의미한다. W와 U는 가중치 매트릭스이며 b는 바이어스이고, σ는 sigmoid 함수 이다. ⊙는 element-wise product이며 f는 비선형 변환 함수로 본 논문에서는 Tanh를 사용하였다. 인코더에서는 Bidirectional GRU를 사용하여 다음과 같이 입력 문장을 인코딩 한다.

$$\vec{h}_t = GRU_{Forward}(E_{src}(x_t), \vec{h}_{t-1})$$

$$\overleftarrow{h}_t = GRU_{Backward}(E_{src}(x_t), \overleftarrow{h}_{t+1})$$

$$h_t = [\vec{h}_t; \overleftarrow{h}_t]$$

이때 Forward와 Backward의 GRU의 경우 가중치들을 공유하지 않으며, 각각의 GRU 결과인 Hidden State Vector들은 concatenate하여 h_t를 생성한다.

디코더에서는 인코딩 된 정보를 이용하여 출력 태그 열을 생성해 내며 본 논문에서는 Attntion mechanism[7]과 Input-feeding[9]을 추가하여 다음과 같이 설계하였다.

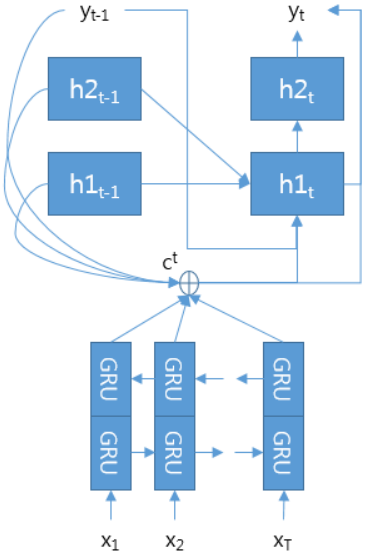


그림 3. Attention mechanism과 Input-feeding을 적용한 Sequence-to-sequence 모델

기존 Sequence-to-sequence 모델은 입력 언어의 문장을 길이에 상관없이 항상 고정된 차원으로 인코딩 하게 되는데, 이로 인해 입력 언어 문장이 길어질 경우 성능이 떨어지게 된다는 문제가 있다. Attention mechanism은 그림 3과 같이 현재의 디코딩 시간 t에 맞는 새로운 Local Context Vector c^t를 만들어 이와 같은 단점을 극복하고, 특정 디코딩 시점의 Attention Weight를 눈으로 확인하여 결과 분석을 할 수 있다는 장점이 있다. Input-feeding은 그림 3에서의 이전 시간의 두 번째 히든레이어 인 $h2_{t-1}$를 디코딩 시의 추가 입력 정보로 넣어 이전 시간에 결정된 Attention Weight 정보를 현재시간에도 활용하며 신경망을 수직, 수평으로 복잡하게 만들어 준다.

Attention mechanism과 Input-feeding을 적용한 디코더의 상세한 식은 다음과 같다.

$$e_i^t = f_{ATT}(E_{tgt}(y_{t-1}), h1_{t-1}, h2_{t-1}, h_i)$$

$$a_i^t = \frac{\exp(e_i^t)}{\sum_{j=1}^{T}\exp(e_j^t)}$$

$$c^t = \sum_{i=1}^{T} a_i^t h_i$$

$$z = \sigma(W_z E_{tgt}(y_{t-1}) + U1_z h1_{t-1} + U2_z h2_{t-1} + W_{zc}c^t + b_z)$$

$$r = \sigma(W_r E_{tgt}(y_{t-1}) + U1_r h1_{t-1} + U2_r h2_{t-1} + W_{rc}c^t + b_r)$$

$$m = f(W_m E_{tgt}(y_{t-1}) + U2_m h2_{t-1} + U1_m(h1_{t-1} \odot r) + W_{mc}c^t + b_m)$$

$$h1_t = (1-z)\odot h1_{t-1} + z \odot m$$

$$h2_t = f_2(W_{h2}h1_t + b_{h2})$$

$$y_t = argmax(softmax(W_{yh}h1_t + W_{yh2}h2_t + W_{yy}E_{tgt}(y_{t-1}) + W_{yc}c^t + b_y))$$

$E_{tgt}(y_{t-1})$은 이전 시간의 디코딩 결과로 생성된 단어인 y_{t-1}의 출력언어 Word Embedding이다. f_{ATT}는 Attenion Weight(a)를 결정하기 위한 Feed-Forward Neural Network(FFNN)이며 현재의 입력 정보($E_{tgt}(y_{t-1})$), 디코더의 이전 히든레이어 상태($h1_{t-1}$,$h2_{t-1}$) 및 인코더의 hidden sate vector(h_i)를 입력으로 받는다. 생성된 Attenion Weight(a)와 인코더의 hidden state vector들을 이용하여 t 시간의 새로운 Context Vector c^t를 생성한다. 디코더의 히든레이어는 두 층을 사용하며 첫 번째 층은 Attetion mechanism(위의 식에서 추가된 c^t부분)과 Input-feeding(위의 식에서 추가된 $h2_{t-1}$부분)을 이용하여 재설계된 GRU로 비선형 변환 함수로 tanh를 사용하였다. 두 번째 층은 비선형 변환 함수로 ReLU를 사용하는 FFNN 이다. Output에서는 softmax 함수를 사용하여 각 태그의 생성 확률을 만들어 내며 이중에서 값이 가장 큰 태그를 해당 디코딩 시간의 결과로 생성해 내게 된다.

3.3 구구조 구문분석을 위한 Beam search

Sequence-to-sequence 모델은 이전 시간의 디코딩 결과가 현재 시간 디코더의 입력으로 들어가게 된다. 이에 따라 디코딩 도중 잘못된 결과가 나올시 그 이후의 디코더에 지속적으로 잘못된 영향을 미치게 된다는 문제가 있으며, 이러한 문제를 해결하기 위해 Beam search 알고리즘을 사용하였다.

본 논문에서 설계한 구구조 구문 분석의 경우, 입력 어절을 출력 결과에서 'XX' 태그로 바꾸어 1:1 매칭 되도록 하였으나, 이 경우 생성된 출력 태그 열의 'XX' 태그의 수가 입력 어절의 수와 다른 경우가 발생할 수 있다. 이에 따라 Beam search 알고리즘에서 마지막 결과를 내기 전에 입력 어절의 수와 생성된 'XX' 태그의 수가 같은 후보들을 우선적으로 선택 하도록 Beam search 알고리즘을 수정 하였다.

4. 실험 및 결과

본 논문에서 제안한 모델의 성능을 평가하기 위해 [13]과 동일한 세종말뭉치의 구구조 구문 분석 데이터 43,828문장을 사용하였고, 그 중 90%는 학습데이터로, 10%는 평가데이터로 사용하였다. 그림 2의 입력열의 형태에 따른 성능 차이를 확인하기 위해 Attention mechanism이 포함된 RNN-search[7]의 모델을 사용하여 우선적으로 비교 실험을 하였다. 학습 프로그램은 Theano[14]를 이용하여 자체적으로 구현하였으며, Source Word embedding과 Target Word Embedding은 200차원을 사용하였고, 히든레이어의 크기는 500과 1000을 실험 하였다.

표 1. 한국어 구구조 구문 분석 성능 평가 결과

모델		F1
스탠포드 구문분석기[13]		74.65
버클리 구문분석기[13]		78.74
입력 예시 1	RNN-search[7] (Beam size 10)	87.34
		87.65*
입력 예시 2	RNN-search[7] (Beam size 10)	87.69
		88.00*
	RNN-search + Input-feeding (Beam size 10)	88.23
		88.68*
	RNN-search + Input-feeding + Dropout (Beam size 10)	88.78
		89.03*

표 1은 한국어 구구조 구문 분석 성능 평가 결과 이다. 실험 결과 Sequence-to-sequence 모델의 성능이 [13]의 영어 구문분석기를 한국어에 적용한 결과보다 좋음을 알 수 있다. 입력 문장의 경우 형태소로 입력하는 것 보다 형태소의 음절과 품사 태그 정보로 나누어 넣는 것이 좋음을 확인할 수 있으며, Input-feeding을 추가시 성능이 크게 향상되는 것을 볼 수 있다. 또한 Overfitting을 방지 할 수 있는 Dropout[15]을 히든레이어에 0.2의 확률로 적용하였을 때 성능이 향상되는 것을 볼 수 있다. 표 1의 (*)표시는 수정된 Beam search 알고리즘을 사용한 결과로 전체적으로 수정 전보다 성능이 향상된 것을 볼 수 있다.

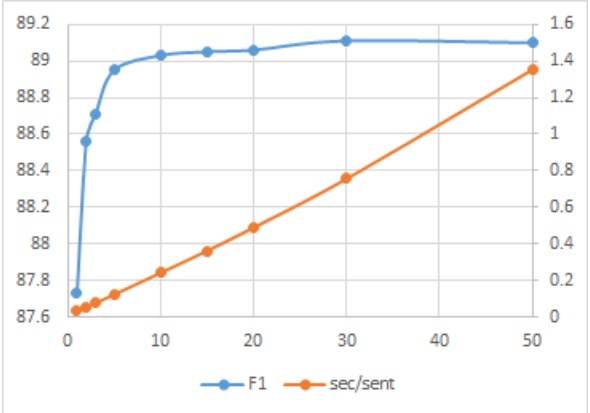


그림 4. Beam size별 성능 및 속도 그래프

그림 4는 Beam size 별 성능과 처리 속도를 나타낸다. 처리 속도를 측정하기 위해 Sequence-to-sequence 모델을 cuda라이브러리를 사용하여 C++로 구현하였으며 Intel(R) core(TM) i5-4690(3.50GHz), DDR3RAM, GeForce GTX 980Ti의 리눅스 환경에서 측정하였다. Beam size가 커지면 그에 따라 성능도 오르나 10정도에서 부터는 성능 향상이 미미한 것을 볼 수 있다. 그리고 Beam size가 커짐에 따라 한 문장을 처리 하는데 걸리는 시간이 늘어나 Beam size 10정도가 최적임을 확인 할 수 있다. 본 논문에서 제안한 모델은 Beam size가 10일 때 평가데이터에 대해 초당 평균 4.12문장을 처리하였다. 기존의 구구조 구문 분석의 시간 복잡도는 $O(n^3)$이상 이었으나, 본 논문에서 제안한 모델은 $O(n^2)$의 복잡도를 가지어 긴 문장의 구문 분석에 좀 더 빠른 속도를 보일 것으로 예상된다.

그림 5는 Attention Weight의 예시이다. 하나의 어절로 매칭되는 'XX' 태그를 생성할 때에는 해당 어절에 Attention Weight가 집중 되는 것을 볼 수 있으며, 구를 구분하는 닫는 괄호를 생성 할 때에는 띄어쓰기 정보인 <sp>에 Attention Weight가 집중 되는 것을 볼 수 있다. 그리고 출력 태그 열의 뒷부분에서 괄호를 닫으며 구문 분석을 마무리 하는 과정에서는 문장의 앞쪽 정보도 함께 보려는 것을 볼 수 있다.

표 2는 RNN-search[7] (Beam size 10) 모델(표 2의 모델 1)과 RNN-search + Input-feeding + Dropout (Beam size 10) 모델(표 2의 모델 2)의 구문 분석 결과를 비교한 표이다. 입력 문장은 "선생님의 이야기 끝나자 마치는 종이 울렸다."를 입력 하였고 두 가지 모델 모두 형태소의 음절과 품사 태그 정보를 입력으로 넣었다. 구문 분석 결과 모델 1은 구분 분석 태그 3개가 틀린 것을 볼 수 있으며 모델 2의 경우 전부 다 맞은 것을 볼 수 있다. 두 가지 모델 모두 괄호를 열고 닫는 것을 오류 없이 잘 생성 해 내었으며, 어절을 나타내는 'XX' 태그도 잘 매칭되게 생성 해내어 Sequence-to-sequence 모델로 한국어 구구조 구문 분석을 잘 해냄을 알 수 있다.

5. 결론

본 논문에서는 기계 번역등에 쓰이던 Sequence-to-sequence 모델을 한국어 구구조 구문 분석에 맞게 설계하고, Attention mechanism과 Input-feeding을 적용한 모델을 제안하였다. 실험 결과, 기존의 모델들보다 좋은 성능을 보였다. 향후 연구로는 Sequence-to-sequence 모델을 다양한 자연어처리 분야에 적용하는 방법을 연구할 계획이다.

감사의 글

이 논문은 2016년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임. (No.R0101-16-0062, (엑소브레인-1세부) 휴먼 지식증강 서비스를 위한 지능진화형 WiseQA 플랫폼 기술 개발)

참고문헌

[1] 이공주, 김재훈, 김길창. "제한된 형태의 구구조 문법에 기반한 한국어 구문 분석." 정보과학회논문지(B), 제25권, 제4호, pp.722-732, 1998.

[2] 남웅, 윤애선, 권혁철. "무제한 한국어 의존 구문분석기 'PNU-KLParser 2.0'의 개발." 정보과학회논문지: 컴퓨팅의 실제 및 레터, 제20권, 제6호, pp354-358, 2014.

[3] 이창기, 김준석, 김정희. "딥 러닝을 이용한 한국어 의존 구문 분석." 제26회 한글 및 한국어 정보처리 학술대회 논문집, 2014.

[4] Collobert, Ronan, et al. "Natural language processing (almost) from scratch." Journal of Machine Learning Research, 12, 2011.

[5] Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le. "Sequence to sequence learning with neural networks." Advances in neural information processing systems, 2014.

[6] Cho, Kyunghyun, et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." arXiv preprint arXiv:1406.1078, 2014.

[7] Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473, 2014.

[8] 이창기, 김준석, 이형규, 이재송. "문자 단위의 Neural Machine Translation." 제27회 한글 및 한국어 정보처리학술대회 논문집, 2015.

[9] Luong, Minh-Thang, Hieu Pham, and Christopher D. Manning. "Effective approaches to attention-based neural machine translation." arXiv preprint arXiv:1508.04025, 2015.

[10] Vinyals, Oriol, et al. "Grammar as a foreign language." Advances in Neural Information Processing Systems, 2015.

[11] 이건일, 이의현, 이종혁. "Sequence-to-sequence 모델을 이용한 한국어 형태소 분석 및 품사 태깅." 한국정보과학회 학술발표논문집, 2016.

[12] 이공주, 김재훈, 김길창. "한국어 구구조 문법을 기반으로 하는 확률적 구문 분석." 한국정보과학회 1996 년도 가을 학술발표논문집, 제23권, 제2호(A), pp557-560, 1996.

[13] 최동현, 박정열, 임경태, 함영균, 최기선. "구문 분석기 성능 향상을 위한 세종 트리뱅크 변환 방법." 한국정보과학회 2012 한국컴퓨터종합학술대회 논문집, 제39권, 제1호(B), pp342-344, 2012.

[14] Bastien, Frédéric, et al. "Theano: new features and speed improvements." arXiv preprint arXiv:1211.5590, 2012.

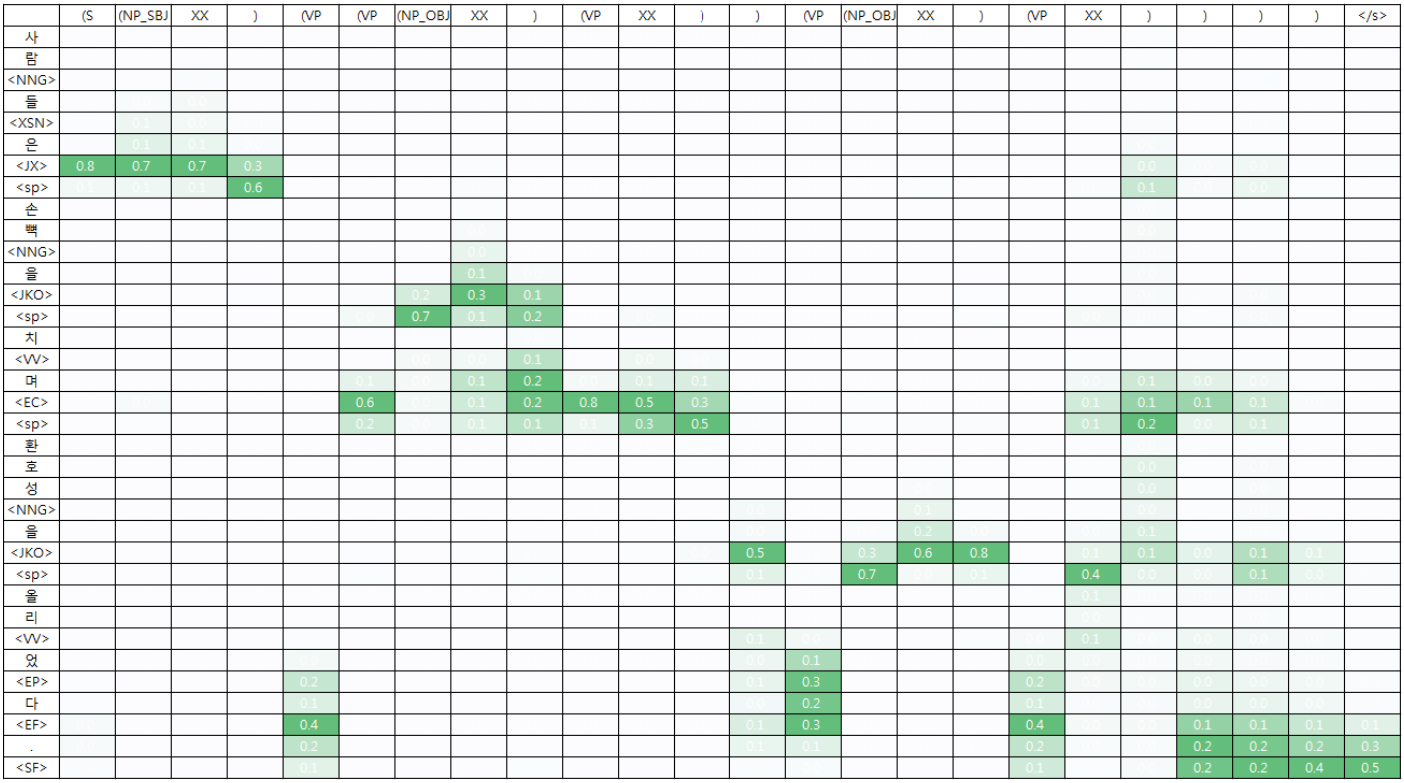


그림 5. 한국어 구구조 구문 분석 결과의 Attention Weight의 예시

표 2. 모델에 따른 한국어 구구조 구문 분석 결과의 비교

입력	선 생 <NNG> 님 <XSN> 의 <JKG> <sp> 이 야 기 <NNG> <sp> 끝 나 <VV> 자 <EC> <sp> 마 치 <VV> 는 <ETM> <sp> 종 <NNG> 이 <JKS> <sp> 울 리 <VV> 었 <EP> 다 <EF> . <SF>
정답	(S (S (NP_SBJ (NP_MOD XX) (NP_SBJ XX)) (VP XX)) (S (NP_SBJ (VP_MOD XX) (NP_SBJ XX)) (VP XX)))
모델 1	(S **(VP (NP_OBJ** (NP_MOD XX) **(NP_OBJ** XX)) (VP XX)) (S (NP_SBJ (VP_MOD XX) (NP_SBJ XX)) (VP XX)))
모델 2	(S **(S (NP_SBJ** (NP_MOD XX) **(NP_SBJ** XX)) (VP XX)) (S (NP_SBJ (VP_MOD XX) (NP_SBJ XX)) (VP XX)))