2022년 IHER 제6차 전문가 초청 강연회
2022-11-24(목) 15:00-17:00 / 단국대학교 죽전캠퍼스 사회과학관 105호

Universal Dependencies와 BERT/RoBERTa 모델을 통한 고전 중국어 정보처리

야스오카 고이치(安岡孝一, YASUOKA Koichi)*

1 개요

이 논문에서는 Universal Dependencies(이하 UD)[1]라고 불리는 다언어 의존관계 말뭉치[2]를 사용한 자연어 처리 중 특히, 품사 태그 부여(품사 태깅)와 의존관계 분석에 관해 자세하게 소개하겠습니다. 사실 이런 분석기법의 뒤에는 복잡한 수식이 존재하지만, 여기서는 가능한 한 수식을 사용하지 않고 설명하도록 하겠습니다. 한편 현대의 자연어 처리는 그 대부분이 BERT[3]·RoBERTa[4]·DeBERTa[5] 등의 사전학습모델을 활용하기 때문에 이 사전학습모델에 대해서도 개요를 소개하겠습니다. 그리고 각 방법을 독자가 실제로 시도해 볼 수 있도록 Google Colaboratory[6]를 사용한 프로그래밍 사례도 같이 설명하겠습니다. 프로그래밍 사례는 이해하기 쉽도록 또 그대로 따라 하기 쉽도록 길어도 25줄 정도로 끝나도록 신경 썼습니다. 파이썬(python) 등의 기초지식이 없어도 일단 시도해 볼 수 있을 것입니다.

이 논문은 제1장이 언어횡단적인(cross-lingual) '개요'이며 제2장부터는 각 언어별로 방법을 설명합니다. 어느 장부터 읽기 시작해도 괜찮지만, 의문이 생긴다면 반드시 제1장으로 돌아가는 것을 추천합니다. 또한 각각의 분석기법에 대해서는 가능한 선에서 원논문과 URL을 실었습니다. 그리고 이 논문에서 다 다루지 못한 언어에 대해서는 deplacy[7]의 서포트 페이지[8]에 Google Colaboratory 사용 사례를 제시했습니다(표1). 응용을 검토할 때 도움이 되었으면 좋겠습니다.

1.1 Universal Dependencies와 CoNLL-U

UD는 서사언어의 품사·형태소 속성·의존구조를 언어에 관계없이 분석하는 기법입니다. 구구조를 고려하지 않고 의존관계를 분석함으로써 언어횡단성을 높이고 모든 문법구조를 단어 간 관계(링크)로 설명하는 것이 특징입니다.

---

* 교토대학 인문과학연구소 부속 동아시아인문정보학연구센터 교수

1) Marie-Catherine de Marneffe, Christopher D. Manning, Joakim Nivre, Daniel Zeman: Universal Dependencies, Computational Linguistics, Vol.47, No.2 (June 2021), pp.255-308.
2) https://universaldependencies.org 에서 카렐대학의 LINDAT/CLARIN을 중심으로 한 100 이상의 그룹이 전세계 언어를 대상으로 개발 중.
3) Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova: BERT: Pre-training of Deep Bidirectional Transformers for Language, Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (June 2019), Human Language Technologies, Vol.1, pp.4171-4186.
4) https://arxiv.org/abs/1907.11692
5) Pencheng He, Xiaodong Liu, Jianfeng Gao, Weizhu Chen: DeBERTa: Decoding-enhanced Bert With Disentangled Attention, 9th International Conference on Learning Representations (May 2021), Poster 03-2562.
6) https://colab.research.google.com
7) 安岡孝一: Universal Dependenciesにもとづく多言語係り受け可視化ツール deplacy,人文科学とコンピュータシンポジウム「じんもんこん 2020」論文集 (2020年 12月), pp.95-100.
8) https://koichiyasuoka.github.io/deplacy/#templates-for-google-colaboratory

의존구조 분석 자체는 Tesnière의 구조적통어론[9]에서 출발하여 Мельчук의 방향 그래프 기술[10]에 의해 어느 정도 완성이 된 기법입니다. 가장 큰 특징은 이른바 동사중심주의로 언어횡단적인 분석이 가능하다는 점이며, Мельчук 의존문법이 컴퓨터용으로 다듬어진 UD에서도 언어에 상관없이 쓸 수 있다는 장점을 전면에 내세우고 있습니다. UD에서의 문법구조 기술은 구 구조를 고려하지 않고 모든 것을 단어 사이의 링크로 표현합니다. 이에 따라 언어횡단적인 문법구조 분석이 가능합니다.

표1: deplacy에 접속 가능한 각 언어의 의존관계 분석 프로그램(2022년 10월 25일 기준)

アフリカーンス語	Camphr-Udify, UDPipe 2, Trankit, spaCy-jPTDP, Stanza, COMBO-pytorch, spacy-udpipe
アラビア語	Trankit, UDPipe 2, spacy-udpipe, Stanza, spaCy-jPTDP
ベラルーシ語	Camphr-Udify, Stanza, Trankit, UDPipe 2, spacy-udpipe
ブルガリア語	Trankit, Camphr-Udify, UDPipe 2, CLASSLA, Stanza, spaCy-jPTDP, spacy-udpipe
カタルーニャ語	Camphr-Udify, Stanza, spacy-udpipe, COMBO-pytorch, Trankit, UDPipe 2, spaCy, spaCy-jPTDP
コプト語	esupar, spaCy-Coptic, UDPipe 2, Stanza
チェコ語	Camphr-Udify, spaCy-jPTDP, Trankit, UDPipe 2, spacy-udpipe, Stanza, COMBO-pytorch など
ウェールズ語	UDPipe 2, Stanza, Camphr-Udify
デンマーク語	Camphr-Udify, UDPipe 2, Stanza, Trankit, spaCy, COMBO-pytorch, spacy-udpipe, spaCy-jPTDP など
ドイツ語	Camphr-Udify, Stanza, COMBO-pytorch, UDPipe 2, spaCy-jPTDP, NLP-Cube, spacy-udpipe など
ギリシア語	Stanza, UDPipe 2, Trankit, spacy-udpipe, NLP-Cube, spaCy-jPTDP, Camphr-Udify など
英語	Camphr-Udify, Stanza, COMBO-pytorch, UDPipe 2, NLP-Cube, Trankit, spacy-udpipe など
スペイン語	Stanza, UDPipe 2, NLP-Cube, spaCy, spacy-udpipe, Trankit, Camphr-Udify, spaCy-jPTDP
エストニア語	Stanza, spacy-udpipe, EstMalt, COMBO-pytorch, spaCy-jPTDP, UDPipe 2, Trankit など
バスク語	spaCy-ixaKat, COMBO-pytorch, Trankit, Stanza, spacy-udpipe, Camphr-Udify, spaCy-jPTDP など
ペルシア語	Stanza, spaCy-jPTDP, spacy-udpipe, Trankit, Camphr-Udify, UDPipe 2 など
スオミ語	Stanza, UDPipe 2, spacy-udpipe, spacy-fi, Turku-neural-parser-pipeline, COMBO-pytorch など
フェロー語	Stanza
フランス語	spaCy, Stanza, UDPipe 2, Trankit, spacy-udpipe, Camphr-Udify, NLP-Cube, spaCy-jPTDP など
ゲール語 (アイルランド)	Stanza, UDPipe 2, COMBO-pytorch, Trankit, spacy-udpipe, spaCy-jPTDP, Camphr-Udify
ゲール語 (スコットランド)	Stanza, UDPipe 2, spacy-udpipe, Trankit
ガリシア語	Camphr-Udify, Stanza, spaCy-jPTDP, spacy-udpipe, COMBO-pytorch, Trankit, UDPipe 2
古典ギリシア語	Camphr-Udify, UDPipe 2, spaCy-jPTDP, Stanza, Trankit, spacy-udpipe
ヘブライ語	Trankit, HebPipe, Stanza, spaCy-jPTDP, UDPipe 2, COMBO-pytorch, spacy-udpipe
ヒンディー語	UDPipe 2, Trankit, Stanza, spaCy-jPTDP, spacy-udpipe, Camphr-Udify
クロアチア語	UDPipe 2, COMBO-pytorch, Trankit, Camphr-Udify, Stanza, CLASSLA, spaCy, spaCy-jPTDP など
ハンガリー語	Trankit, UDPipe 2, COMBO-pytorch, Camphr-Udify, HuSpacy, spacy-udpipe, Stanza など
アルメニア語	Stanza, Camphr-Udify, Trankit, UDPipe 2, spacy-udpipe
インドネシア語	Stanza, Trankit, spacy-udpipe, Camphr-Udify, UDPipe 2, COMBO-pytorch, spaCy-jPTDP など
アイスランド語	COMBO-pytorch, Stanza, UDPipe 2, is_ud_is_pud
イタリア語	NLP-Cube, COMBO-pytorch, Trankit, Camphr-Udify, Stanza, spaCy, UDPipe 2 など
日本語	spaCy-SynCha, spaCy-ChaPAS, UniDic-COMBO, spaCy, GiNZA, UDPipe 2, Stanza, UniDic2UD など
カザフ語	Camphr-Udify, Trankit, NLP-Cube, kazdet
韓国語	Camphr-Udify, Stanza, UDPipe 2, Trankit, spaCy-jPTDP, spacy-udpipe, Stanza
ラテン語	Trankit, Stanza, UDPipe 2, spacy-udpipe, Camphr-Udify, spaCy-jPTDP
リトアニア語	spaCy, Trankit, Stanza, Camphr-Udify, UDPipe 2, spacy-udpipe
ラトビア語	Camphr-Udify, Trankit, UDPipe 2, Stanza, spacy-udpipe, spaCy-jPTDP
古典中国語	SuPar-Kanbun, UD-Kanbun, GuwenCOMBO, UD-Chinese, Trankit, esupar, Stanza, UDPipe 2 など
マケドニア語	Camphr-Udify, spaCy
マルタ語	Stanza, UDPipe 2, spacy-udpipe, COMBO-pytorch, Camphr-Udify
ノルウェー語 (ブークモール)	Stanza, COMBO-pytorch, Camphr-Udify, UDPipe 2, Trankit, spaCy-jPTDP, spaCy
ノルウェー語 (ニーノルスク)	Stanza, spacy-udpipe, spaCy-jPTDP, UDPipe 2, Trankit
オランダ語	spaCy-Alpino, Camphr-Udify, Stanza, UDPipe 2, Trankit, spacy-udpipe, spaCy, COMBO-pytorch など
ポーランド語	spaCy, Camphr-Udify, Stanza, UDPipe 2, spacy-udpipe, Trankit, COMBO-pytorch, spaCy-jPTDP など
ポルトガル語	spaCy, Camphr-Udify, Stanza, COMBO-pytorch, spaCy-jPTDP, UDPipe 2, spacy-udpipe など
ルーマニア語	UDPipe 2, COMBO-pytorch, Trankit, NLP-Cube, Camphr-Udify, spaCy, Stanza, spaCy-jPTDP など
ロシア語	Stanza, Trankit, Camphr-Udify, UDPipe 2, NLP-Cube, spaCy, COMBO-pytorch, spacy-udpipe など
スロバキア語	Camphr-Udify, UDPipe 2, Trankit, spacy-udpipe, NLP-Cube, Stanza, spaCy-jPTDP など
スロベニア語	CLASSLA, UDPipe 2, COMBO-pytorch, spacy-udpipe, Camphr-Udify, Trankit, spaCy-jPTDP など
セルビア語 (キリル)	esupar, Camphr-Udify
セルビア語 (ラテン)	CLASSLA, UDPipe 2, Trankit, Camphr-Udify, esupar, Stanza, spacy-udpipe, spaCy-jPTDP
スウェーデン語	Camphr-Udify, COMBO-pytorch, Trankit, Stanza, UDPipe 2, spacy-udpipe, spaCy-jPTDP
タミル語	Camphr-Udify, Trankit, UDPipe 2, spacy-udpipe, Stanza
タイ語	esupar, spaCy-Thai
タガログ語	Camphr-Udify
トルコ語	Stanza, spaCy-jPTDP, Camphr-Udify, UDPipe 2, spacy-udpipe, COMBO-pytorch, Trankit
ウクライナ語	Stanza, NLP-Cube, Camphr-Udify, UDPipe 2, Trankit, spacy-udpipe, spaCy-jPTDP など
ベトナム語	Trankit, Stanza, spaCy-jPTDP, COMBO-pytorch, UDPipe 2, spacy-udpipe など
ウォロフ語	UDPipe 2, Stanza
中国語 (簡化字)	Trankit, Stanza, UDPipe 2, NLP-Cube, esupar, spacy-udpipe, UD-Chinese, spaCy など
中国語 (繁體字)	Trankit, Stanza, UDPipe 2, esupar, NLP-Cube, spacy-udpipe, UD-Chinese, spaCy など

9) Lucien Tesni`ere: El´ements de Syntaxe Structurale, Paris: C. Klincksieck (1959).

10)]Igor A. Mel'ˇcuk: Dependency Syntax: Theory and Practice, New York: State University of New York Press (1988).

표2: CoNLL-U의 각 필드

1. ID: 単語ごとに付与されたインデックスで、文ごとに 1 から始まる整数。縮約語に対しては、単語の範囲を示すのも可。
2. FORM: 語、または、句読記号。
3. LEMMA: 基底形、語幹。
4. UPOS: UD で規定された言語普遍な品詞タグ (表 3)。
5. XPOS: 言語固有の品詞タグ。
6. FEATS: UD で規定された言語普遍な形態素属性のリスト。言語固有の拡張も可。
7. HEAD: 当該の単語の係り受け元 ID。係り受け元が無い場合は 0 とする。
8. DEPREL: UD で規定された言語普遍な係り受けタグ (表 4)。HEAD が 0 の場合は root とする。言語固有の拡張も可。
9. DEPS: 複数の係り受け元を持つ場合、全ての HEAD:DEPREL ペア。
10. MISC: その他のアノテーション。

표3: UD 품사 태그(UPOS)

Open class words	Closed class words	Other
ADJ 形容詞	ADP 側置詞	PUNCT 句読点
ADV 副詞	AUX 助動詞	SYM 記号
INTJ 感嘆詞	CCONJ 並列接続詞	X その他
NOUN 名詞	DET 限定詞	
PROPN 固有名詞	NUM 数詞	
VERB 動詞	PART 接辞	
	PRON 代名詞	
	SCONJ 従属接続詞	

표4: UD 의존관계 태그(DEPREL)

	Nominals	Clauses	Modifier Words	Function Words
Core arguments	nsubj 主語 obj 目的語 iobj 間接目的語	csubj 節主語 ccomp 節目的語 xcomp 節補語		
Non-core dependents	obl 斜格補語 vocative 呼称語 expl 形式語 dislocated 外置語	advcl 連用修飾節	advmod 連用修飾語 discourse 談話要素	aux 動詞補助成分 cop 繋辞 mark 標識
Nominal dependents	nmod 体言による連体修飾語 appos 同格 nummod 数量による修飾語	acl 連体修飾節	amod 用言による連体修飾語	det 決定語 clf 類別語 case 格表示
Coordination	**MWE**	**Loose**	**Special**	**Other**
conj 接続 cc 接続語	fixed 固着 flat 並列 compound 複合	list 細目 parataxis 隣接表現	orphan 親なし goeswith 泣き別れ reparandum 言い損じ	punct 句読点 root 親 dep 未定義

```
# text = 人莫知其子之惡
1    人        人        NOUN    n,名詞,人,人          _    3    nsubj    _    SpaceAfter=No
2    莫        莫        ADV     v,副詞,否定,禁止      _    3    advmod   _    SpaceAfter=No
3    知        知        VERB    v,動詞,行為,動作      _    0    root     _    SpaceAfter=No
4    其        其        PRON    n,代名詞,人称,起格    _    5    det      _    SpaceAfter=No
5    子        子        NOUN    n,名詞,人,関係        _    7    nmod     _    SpaceAfter=No
6    之        之        SCONJ   p,助詞,接続,属格      _    5    case     _    SpaceAfter=No
7    惡        惡        NOUN    n,名詞,描写,態度      _    3    obj      _    SpaceAfter=No

# text = 人は其の子の悪しきを知らない
1    人        人        NOUN    名詞-普通名詞-一般     _    8    nsubj    _    SpaceAfter=No
2    は        は        ADP     助詞-係助詞            _    1    case     _    SpaceAfter=No
3    其の      其の      DET     連体詞                 _    4    det      _    SpaceAfter=No
4    子        子        NOUN    名詞-普通名詞-一般     _    6    nsubj    _    SpaceAfter=No
5    の        の        ADP     助詞-格助詞            _    4    case     _    SpaceAfter=No
6    悪しき    悪しい    ADJ     形容詞-一般            _    8    ccomp    _    SpaceAfter=No
7    を        を        ADP     助詞-格助詞            _    6    case     _    SpaceAfter=No
8    知ら      知る      VERB    動詞-一般              _    0    root     _    SpaceAfter=No
9    ない      ない      AUX     助動詞                 _    8    aux      _    SpaceAfter=No

# text = A man doesn't know the evil of his son
1    A         a         DET     DT                     _    2    det      _    _
2    man       man       NOUN    NN                     _    5    nsubj    _    _
3-4  doesn't   _         _       _                      _    _    _        _    _
3    does      do        AUX     VBZ                    _    5    aux      _    SpaceAfter=No
4    n't       not       PART    RB                     _    5    advmod   _    _
5    know      know      VERB    VB                     _    0    root     _    _
6    the       the       DET     DT                     _    7    det      _    _
7    evil      evil      NOUN    NN                     _    5    obj      _    _
8    of        of        ADP     IN                     _    10   case     _    _
9    his       he        PRON    PRP$                   _    10   det      _    _
10   son       son       NOUN    NN                     _    7    nmod     _    SpaceAfter=No
```

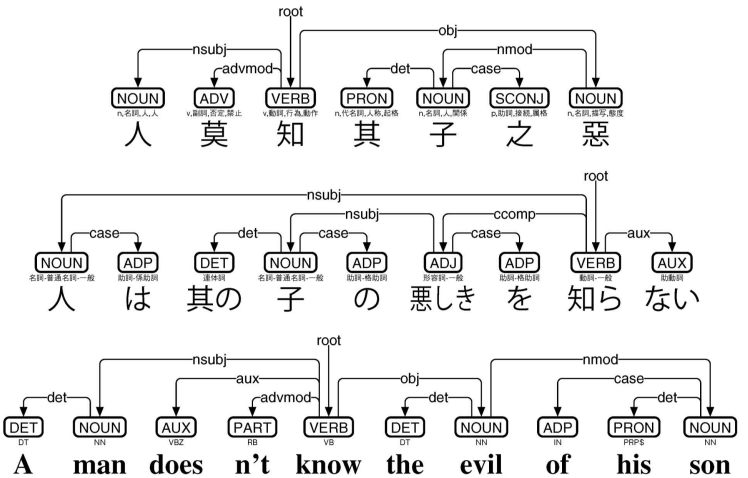

그림1: 한일영 CoNLL-U와 deplacy를 통한 가시화

UD 의존관계 말뭉치의 교환용 포맷으로는 CoNLL-U라고 불리는 탭으로 구분되는 텍스트(문자코드는 UTF-8[11])가 규정되어 있습니다. CoNLL-U의 각 행은 각 단어에 대응하여 표2처럼 10개의 탭으로 구분되는 필드로 구성됩니다. ID·FORM·LEMMA는 단어 그 자체에 관한 필드입니다. UPOS·XPOS·FEATS는 단어의 품사와 형태소 속성에 관한 필드입니다. HEAD·DEPREL·DEPS는 단어의 의존관계와 관련된 필드입니다.

11) ISO/IEC 10646-1:1993/Amd.2:1996 Information Technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane, Amendment 2: UCS Transformation Format 8 (UTF-8), International Organization for Standardization, Gen`eve (October 15, 1996).

UD에서의 의존관계는, 단어 사이의 방향 그래프를 HEAD와 DEPREL로 나타냅니다. HEAD는 그 단어로 향하는 링크의 원 ID를 나타내고, DEPREL은 이 링크가 보여주는 의존관계 태그입니다. 다만 HEAD가 0이라면, HEAD로 들어오는 링크는 존재하지 않습니다. 링크의 개수는 단어의 개수와 같으며, 각 링크가 향하는 곳은 전부 다릅니다. 즉, 각 단어에서 나오는 링크는 복수일 가능성이 있지만, 각 단어로 들어가는 링크는 하나뿐입니다. 또한 링크는 양방향이 될 수 없습니다.

UD의 의존관계 링크는 Мельчук 의존문법의 후예로, 이른바 동사중심주의입니다. 동사를 중심으로 주어나 목적어로 링크됩니다. 수식관계에서 보면 피수식어에서 수식어로 링크됩니다. 다만 전치사와 후치사를 체언의 수식어로 간주한다는 점[12]이 Мельчук와는 다릅니다. 그리고 계사문에서는 동사중심주의를 채택하지 않고 보어를 중심으로 주어나 계사(코퓰러)로 링크됩니다.

UD의 언어횡단성을 보여주는 실제 사례로 고전중국어(한문) 문장 '人莫知其子之惡'와 일본어 문장 '人は其の子の悪しきを知らない'와 영어 문장 'A man doesn't know the evil of his son'의 CoNLL-U를 그림1에서 비교해 봅시다. 이 CoNLL-U는 각 언어 UD에 관계없이 수작업으로 쓴 것으로 FEATS·DEPS는 사용하지 않았습니다. 가시화에는 deplacy를 사용했습니다. 한문, 일본어, 영어는 어순은 다르지만 nsubj(주어)와 nmod(명사관형어)와 obj(목적어)와 ccomp(보문)의 링크가 각각에 대응한다고 할 수 있습니다.

그리고 계사문에 대한 UD의 예시로 러시아어 문장 'Это ручка'와 프랑스어 문장 'C'est un stylo'와 일본어 문장 'これはペンです'의 CoNLL-U를 그림2에서 비교해 봅시다. 러시아어, 프랑스어, 일본어 전부 보어에서 주어로 nsubj가 연결되어 각각에 대응하고 있다고 할 수 있습니다. 또한 프랑스어와 일본어에서는 계사가 cop로 연결되어 있습니다.

1.2 의존문법이론과 Universal Dependencies

UD의 의존구조는 Мельчук의 방향 그래프 기술을 이론적 지주로 삼으면서도 실천적인 면에서는 Reed-Kellogg의 문법구조[13]를 적용합니다. 이 부분에 대해서 간단히 소개하겠습니다.

Reed-Kellogg의 문법구조표는 '주어-술어-목적어'라는 구조를 기본으로 영문을 시각화합니다. 주문(main sentence)은 굵은 선으로, 이 이외의 절(clause)은 얇은 선으로 표시한 다음, 수식어를 사선으로 나타냈습니다. 예를 들어 'Those who labour with their minds rule others'라는 영어 문장에서는 'Those rule others'라는 주문의 'Those'를 'who labour'이라는 절이 수식하고, 이 'labour'을 'with minds'가 수식하고, 'minds'를 'their'가 수식하기 때문에 그림3처럼 됩니다. Reed-Kellogg의 문법구조표는 영어에 특화되어 설계되었기 때문에 반드시 다른 언어에 적용되는 것은 아닙니다.

12) Joakim Nivre: Towards a Universal Grammar for Natural Language Processing, CICLing 2015: 16th International Conference on Intelligent Text Processing and Computational Linguistics (April 2015), pp.3-16.

13) Alonzo Reed and Brainerd Kellogg: Higher Lessons in English: A Work on English Grammar and Composition, New York: Clark & Maynard (1877).

```
# text = Это ручка
1    Это      это      PRON  _                    _  2  nsubj  _  _
2    ручка    ручка    NOUN  _                    _  0  root   _  SpaceAfter=No

# text = C'est un stylo
1    C'       ce       PRON  _                    _  4  nsubj  _  SpaceAfter=No
2    est      être     AUX   _                    _  4  cop    _  _
3    un       un       DET   _                    _  4  det    _  _
4    stylo    stylo    NOUN  _                    _  0  root   _  SpaceAfter=No

# text = これはペンです
1    これ     此れ     PRON  代名詞               _  3  nsubj  _  SpaceAfter=No
2    は       は       ADP   助詞-係助詞          _  1  case   _  SpaceAfter=No
3    ペン     ペン     NOUN  名詞-普通名詞-一般   _  0  root   _  SpaceAfter=No
4    です     です     AUX   助動詞               _  3  cop    _  SpaceAfter=No
```

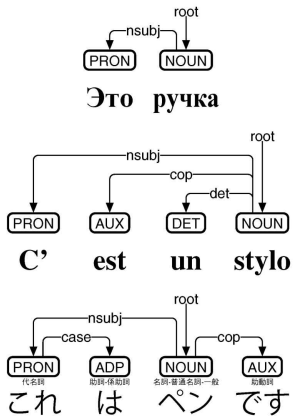

그림2: 계사문의 CoNLL-U와 deplacy에 따른 가시화

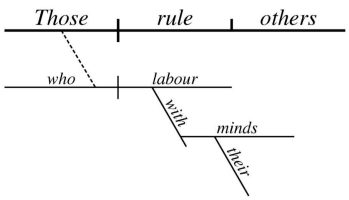

그림3: Reed-Kellogg의 문법구조표

Tesnière는 슬라브어파의 연구를 통해 후에 의존문법이라고 불리게 되는 언어횡단적인 기술 기법을 생각해 냅니다. 문장 안의 단어는 서로 의존관계에 있다는 것이 의존문법의 기본적인 생각입니다. 단어와 단어 사이의 의존관계에서는 상위항(régissant)이 하위항(subordonné)을 지배하고 하위항이 상위항에 의존합니다. 예를 들어 영어 문장 'Those who labour with their minds rule others'에서는 'rule'이 'Those'와 'others'를 지배하고, 'Those'와 'others'가 'rule'에 의존합니다. 다만 전용(轉用)이 이루어진 경우는 이들 단어 사이에 의존관계는 없으며 병치하는 형태로 표현합니다. 그림4의 왼쪽을 보면 명사 'minds'가 'with'로 인해 E전용(연용수식어로 전용)되고, 동사 'labour'가 'who'로 인해 A전용(연체수식어로 전용)되었습니다.

Tesnière의 의존문법은 문장 안에서 단어의 순서에 관계없이 단어의 의존관계를 나타내기

때문에 고전중국어에도 응용이 가능합니다. 그러니까 '勞心者治人'에서 '心'가 '勞'에 의존하고, '勞'가 '者'로 인해 O전용(체언으로 전용)되어 '治'에 의존합니다. 이를 그림4의 오른쪽처럼 표시해 보면 'Those who labour with their minds rule others'와 언어를 뛰어넘어 비교도 가능합니다.

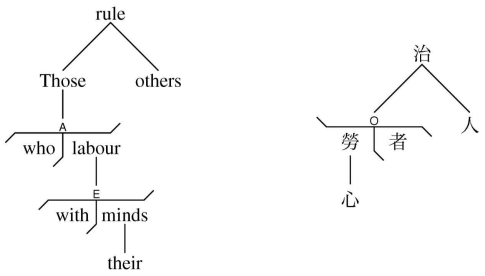

그림4: Tesnière의 의존문법표

Tesnière의 의존문법은 그 후에 Robinson[14]과 Hudson[15] 등이 Chomsky의 구 구조 문법[16]과의 융합을 시도했습니다. 한편 Мельчук는 단어의 의존관계의 방향 그래프로 의존문법을 형식화하면서 Chomsky의 구 구조 문법과 결별했습니다.

Мельчук의 의존문법은 문장 안의 단어 X와 Y에 대해 의존관계 'X→Y'로 문장을 분석합니다. 'X→Y'는 X는 Y를 지배한다, 또는 Y는 X에 의존한다고 읽습니다. 각 화살표에는 의존관계에 관한 적절한 태그가 부여됩니다. 각 단어가 의존하는 단어는 하나이며 의존관계가 상호적인 경우는 없습니다. 의존관계는 어디까지나 단어와 단어 사이의 관계로, 설사 절(clause)이나 구(phrase)와 관련된 것이라도 딱 잘라 단어와 단어 사이의 관계로 기술합니다. 그 결과로 Мельчук의 의존문법은 언어횡단적인 문법구조 기술이 되었습니다. 영어 문장 'Those who labour with their minds rule others'와 일본어 문장 '心を労する者は人を治める'와 고전중국어(한문) '勞心者治人'을 Мельчук의 의존문법으로 문법구조를 표시한 것이 그림5입니다. 태그 가운데 subjectival과 direct-objectival은 과거에는 predicative와 1st completive가 사용되었으며, Мельчук 자신도 태그가 몇 종류 필요한지는 확실히 밝히지 않았습니다[17].

14) Jane J. Robinson: Dependency Structures and Transformational Rules, Language, Vol.46, No.2 (June 1970), pp.259-285.
15) Richard Hudson: Word Grammar, New York: Basil Blackwell (1984).
16) Noam Chomsky: Aspects of the Theory of Syntax, Cambridge: MIT Press (1965).
17) Alain Polgu` cuk: John Benjaminsere, Igor A. Mel'˘ Dependency in Linguistic Description, Amsterdam: (2009).

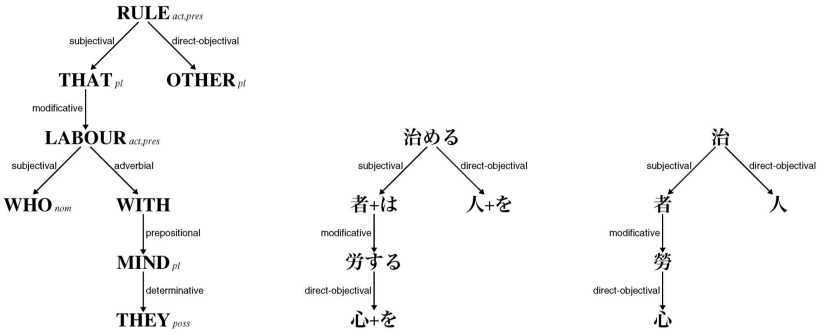

그림5: Мельчук의 의존문법을 이용한 문법구조 분석

UD의 의존구조는 Мельчук의 의존문법의 태그를 표4의 37종류로 한정하는 야심적인 시도로 볼 수 있습니다. 그림5의 subjectival·direct-objectival·modificative·determinative가 그림6에서 nsubj·obj·acl·det가 되었다고 생각할 수도 있습니다. 그런데 UD의 전신인 Stanford Typed Dependencies[18]가 초기에는 Bresnan 어휘기능문법[19]에서 시작되었고 그 후에 의존문법으로 이행되었기 때문에 몇 가지 특이한 개념이 섞여 들어갔습니다. 그중 하나가 절(clause)입니다.

Мельчук는 절이라는 개념을 의존문법에서 배제했지만, UD는 이것을 재도입했습니다. 표4의 nsubj와 csubj는 전부 주어를 나타내는 UD 의존관계 태그이며, 링크가 향하는 곳이 절이라면 csubj를, 그렇지 않으면 nsubj를 사용합니다. obj와 ccomp에 대해서도 마찬가지, advmod와 advcl에 대해서도 마찬가지, amod와 acl에 대해서도 마찬가지입니다. case와 mark에 대해서는 링크가 나오는 곳이 절이라면 mark를, 그렇지 않으면 case를 사용합니다.[20]

영어에서 절이라는 개념은 어느 정도 공감대가 형성되었다고 생각됩니다. 하지만 다른 언어에서 절이라는 개념은 반드시 명확한 것은 아닙니다. UD는 언어횡단적이지만 꽤 영어에 치우쳐 있다는 점은 기억해 두면 좋습니다.

18) Marie-Catherine de Marneffe and Christopher D. Manning: The Stanford Typed Dependencies Representation, Coling 2008: Proceedings of the Workshop on Cross-Framework and Cross-Domain Parser Evaluation (August 2008), pp.1-8.
19) Joan Bresnan: Lexical-Functional Syntax, Malden: Blackwell (2001).
20) Marie-Catherine de Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, Christopher D. Manning: Universal Stanford Dependencies: A Cross-Linguistic Typology, Proceedings of the 9th International Conference on Language Resources and Evaluation (May 2014), pp.4585-4592.

```
# text = Those who labour with their minds rule others
1   Those   that    PRON  DT                        _   7   nsubj  _   _
2   who     who     PRON  WP                        _   3   nsubj  _   _
3   labour  labour  VERB  VBP                       _   1   acl    _   _
4   with    with    ADP   IN                        _   6   case   _   _
5   their   they    PRON  PRP$                      _   6   det    _   _
6   minds   mind    NOUN  NNS$                      _   3   obl    _   _
7   rule    rule    VERB  VBP                       _   0   root   _   _
8   others  other   NOUN  NNS                       _   7   obj    _   SpaceAfter=No

# text = 心を労する者は人を治める
1   心      心      NOUN  名詞-普通名詞-サ変可能    _   3   obj    _   SpaceAfter=No
2   を      を      ADP   助詞-格助詞               _   1   case   _   SpaceAfter=No
3   労する  労する  VERB  動詞-一般                _   4   acl    _   SpaceAfter=No
4   者      者      NOUN  名詞-普通名詞-一般        _   8   nsubj  _   SpaceAfter=No
5   は      は      ADP   助詞-係助詞               _   4   case   _   SpaceAfter=No
6   人      人      NOUN  名詞-普通名詞-一般        _   8   obj    _   SpaceAfter=No
7   を      を      ADP   助詞-格助詞               _   6   case   _   SpaceAfter=No
8   治める  治める  VERB  動詞-一般                _   0   root   _   SpaceAfter=No

# text = 勞心者治人
1   勞      勞      VERB  v,動詞,描写,境遇          _   3   acl    _   SpaceAfter=No
2   心      心      NOUN  n,名詞,不可譲,身体        _   1   obj    _   SpaceAfter=No
3   者      者      PART  p,助詞,提示,*             _   4   nsubj  _   SpaceAfter=No
4   治      治      VERB  v,動詞,行為,動作          _   0   root   _   SpaceAfter=No
5   人      人      NOUN  n,名詞,人,人              _   4   obj    _   SpaceAfter=No
```

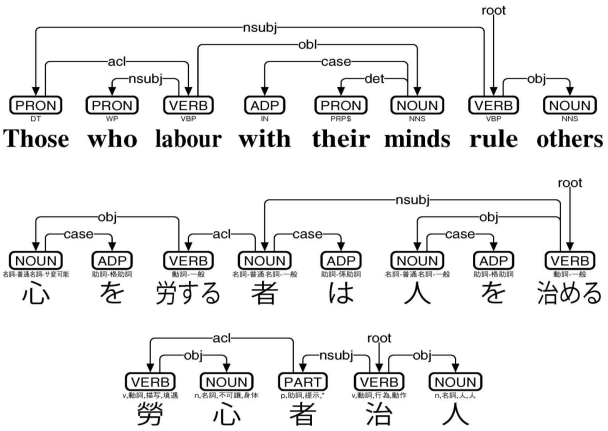

그림6: UD와 CoNLL-U의 의존구조 분석

1.3 BERT/RoBERTa/DeBERTa 모델과 빈칸 채우기 게임

Google AI Language가 발표한 BERT는 대량의 문장을 기계가 통째로 암기하도록 할 때 어떤 방법이 유효할지 단적으로 보여주는 사전학습모델입니다. 단순히 문장을 다 외우게 하는 것이 아니라 글 안의 문장과 문장의 연결 또는 문장 안의 토큰(단어 또는 단어보다 짧은 문자열)과 토큰의 연결을 효과적으로 학습할 수 있도록 여러 장치가 마련되어 있습니다.

토큰과 토큰의 연결을 학습시키기 위해 사용되는 것이 바로 '빈칸 채우기 게임'입니다. 'This [MASK] a pen.'의 [MASK]에 대해 'This is a pen.'이라는 답을 얻을 수 있는 빈칸 채우기 게임(그림7)을 생각해 봅시다. bert-base-cased[21]에서 각 토큰의 입출력은 문자코드(UTF-8)로 이루어지는데 내부적으로는 768차원의 벡터(수치의 열)로 표현됩니다. 입력측에서는 각 입력 토큰에 대응하는 벡터를 그대로 사용하고, 출력측에서는 각 출력 토큰에 가까운[22] 벡터가 되도록 학습시킵니다. 'This [MASK] a pen.'의 열에서 보면, 'This', 'a', 'pen', '.'은 입력 벡터 거의 그대로 통과되고, 'is'에 가까운 출력 벡터를 얻을 수 있도록 내부

21) https://huggingface.co/bert-base-cased

22) 여기서 말하는 '가깝다'란 코사인 유사도(두 벡터의 내적을 벡터의 유클리드 노름으로 나눈 값)가 1에 가깝다는 의미이다.

의 기억소자를 조작합니다. 이런 빈칸 채우기 게임을 대량의 문장에서 실행하여 토큰과 토큰의 연결을 학습[23]할 수 있다는 것이 BERT의 특징 중 하나[24]입니다.

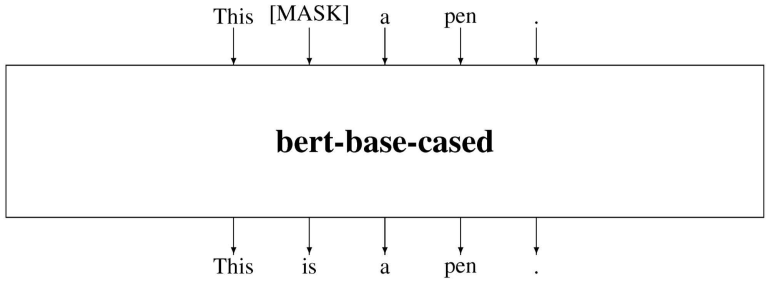

그림7: bert-base-cased의 빈칸 채우기 게임

Google Colaboratory상의 Transformers[25]를 사용하여 bert-base-cased가 영문을 어느 정도 학습할 수 있는지를 시험해 봅시다(그림8). 'This [MASK] a pen.'에서 빈칸을 채워보면 'was'가 48.2퍼센트, 'is'가 40.2퍼센트로, 출력되는 벡터가 'was'와 'is'의 중간으로 'was'에 조금 더 가깝다는 사실을 알 수 있습니다.

```
!pip install transformers
from transformers import pipeline
fmp=pipeline("fill-mask","bert-base-cased")
prd=fmp("This [MASK] a pen.")
print("\n".join("{:8} {:.3f}".format(t["token_str"],t["score"]) for t in prd))
```

```
was      0.482
is       0.402
from     0.008
had      0.007
has      0.007
```

그림8: 'This [MASK] a pen.'의 빈칸 채우기 게임 프로그램과 결과

BERT를 다언어로 확장할 때 문제가 되는 것은 모델의 입출력에 문장이라는 단위를 가정한다는 점입니다. 언어에 따라서는(예를 들면 고전중국어) 문장이라는 단위가 꼭 명백한 것은 아닙니다. 그리고 SNS상의 '트윗' 등 문장의 경계가 확실하지 않은 경우도 많이 있습니다. 이에 비해 RoBERTa나 DeBERTa는 문장이라는 단위를 가정하지 않고 토큰의 나열에

23) bert-base-cased 내부에서는 빈칸 채우기용 [MASK]에 더해서 문장의 첫머리를 나타내는 [CLS], 문장의 분절위치를 나타내는 [SEP], 알 수 없는 단어를 나타내는 [UNK], 빈 토큰을 나타내는 [PAD] 등도 사용됩니다.

24) BERT의 또 다른 노력은 문장과 문장의 연결을 학습하는 인접문 게임인데 본서에서는 다루지 않습니다.

25) Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R´emi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, Alexander M. Rush: Transformers: State-of-the-Art Natural Language Processing, Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (October 2020): System Demonstrations, pp.38-45.

빈칸 채우기 게임에만 특화된 학습을 실행합니다. 이에 따라 문장이라는 단위에 얽매이지 않고 대량의 문장을 통째로 암기할 수 있습니다.

roberta-classical-chinese-base-char[26]는 고전중국어의 한자를 토큰으로 간주하여 한자 단위의 빈칸 채우기 게임(그림9)을 학습한 모델입니다. 각 한자(문자의 종류 약 26,000개)의 입출력은 UTF-8이지만, 내부적으로는 768차원의 벡터로 표현합니다. Google Colaboratory상의 Transformers로 만든 '勞[MASK]者治人'의 빈칸을 채우는 프로그램과 그 결과가 그림10입니다. '心'이 19.9퍼센트로 '人'의 32.7퍼센트보다 낮다는 사실을 알 수 있습니다.

BERT·RoBERTa·DeBERTa 등의 사전학습모델은 출력부를 바꾸는 것으로 다양한 용도로 응용이 가능합니다. 출력 벡터를 품사 정보라고 간주하면 품사 태깅에 사용할 수 있습니다. 의존관계 태그라고 보면 의존관계 분석에도 사용할 수 있습니다. UD에서는 계열 레이블링이나 의존관계 분석에 사전학습모델을 응용할 수 있습니다.

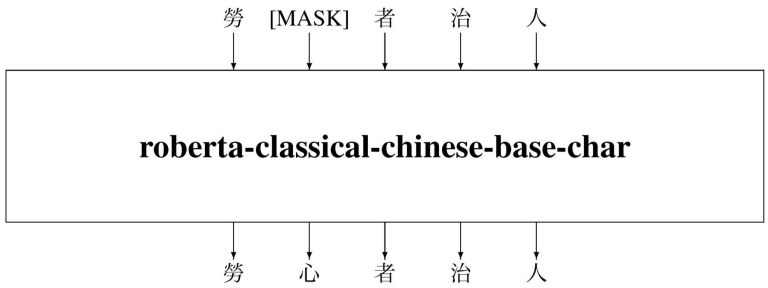

그림9: roberta-classical-chinese-base-char의 빈칸 채우기 게임

```
!pip install transformers
from transformers import pipeline
fmp=pipeline("fill-mask","KoichiYasuoka/roberta-classical-chinese-base-char")
prd=fmp("勞[MASK]者治人")
print("\n".join("{:8} {:.3f}".format(t["token_str"],t["score"]) for t in prd))
```

```
人      0.327
心      0.199
民      0.129
神      0.054
神      0.047
```

그림10: '勞[MASK]者治人'의 빈칸 채우기 게임 프로그램과 결과

1.4 Universal Dependencies의 의존관계 분석

의존관계 분석은 UD를 이용한 언어처리의 핵심이며, 많은 분석 알고리즘이 난립하고 있습니다. 이런 의존관계 분석 알고리즘을 상태전이형 알고리즘과 인접행렬형 알고리즘으로 나

26) https://huggingface.co/KoichiYasuoka/roberta-classical-chinese-base-char

눠서 해설하겠습니다.

arc-swap[27]으로 대표되는 상태전이형 알고리즘은 단어열의 앞쪽에서 뒤쪽으로 향해 '담장(stack-buffer boundary)'을 이동시키는 이미지로 처리를 합니다. '담장'이 하는 전이는 다음의 6종류로 규정합니다.

- Shift '담장'을 오른쪽으로 한 단어만큼 이동한다.
- Reduce '담장'의 바로 왼쪽 단어를 제거하고 분석 결과로 간다.
- Left-Arc '담장'의 바로 오른쪽 단어에서 바로 왼쪽 단어로 링크를 연결한다.
- Right-Arc '담장'의 바로 왼쪽 단어에서 바로 오른쪽 단어로 링크를 연결한다.
- Unshift '담장'을 왼쪽으로 한 단어만큼 이동한다.
- Swap '담장'의 바로 오른쪽 단어와 바로 왼쪽 단어를 바꾼다.

단어가 모두 Reduce되어 '담장'만 남게 되는 시점에서 상태전이형 알고리즘은 종료됩니다. 분석 사례를 그림11에서 확인할 수 있습니다. 분석 결과에서 링크가 들어오지 않는 단어가 일반적으로 root가 됩니다. 또한 6종류 가운데 Unshift·Swap을 사용하지 않는 경우는 링크에 교차가 있는 UD는 사용할 수 없습니다. 그리고 현실에서는 Left-Arc의 직후에는 Reduce를, Right-Arc의 직후에는 Shift와 Reduce를, 각각 같이 전이시키는 방법이 주류입니다.

Biaffine[28]로 대표되는 인접행렬형 알고리즘은 UD 의존구조를 방향 그래프로 보고 그 인접행렬을 구합니다. 예를 들어, 그림11의 오른쪽 아래 'Who did you wait for?'의 방향 그래프라면,

$$\begin{bmatrix} - & - & - & - & \text{case} & - \\ - & - & - & - & - & - \\ - & - & - & - & - & - \\ \text{obl} & \text{aux} & \text{nsubj} & \text{root} & - & \text{punct} \\ - & - & - & - & - & - \\ - & - & - & - & - & - \end{bmatrix}$$

라는 6×6의 인접행렬을 찾습니다(그림12). 구체적으로는 인접행렬의 각 열별로 요소를 하나씩 고르고 대각성분상의 root에서 하나를 선택하는 것을 확률적으로 실행합니다. 그런데 루프가 발생하는 경우가 있기 때문에 적절하게 Chu-Liu-Edmonds 등[29]을 사용하여 이를 해소합니다.

27) Joakim Nivre: Non-Projective Dependency Parsing in Expected Linear Time, Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP (August 2009), pp.351-359.

28) Timothy Dozat, Christopher D. Manning: Deep Biaffine Attention for Neural Dependency Parsing, 5th International Conference on Learning Representations (April 2017), C25.

29) H. N. Gabow, Z. Galil, T. Spencer and R. E. Tarjan: Efficient Algorithms for Finding Minimum Spanning Trees in Undirected and Directed Graphs, Combinatorica, Vol.6, No.2 (June 1986), pp.109-122.

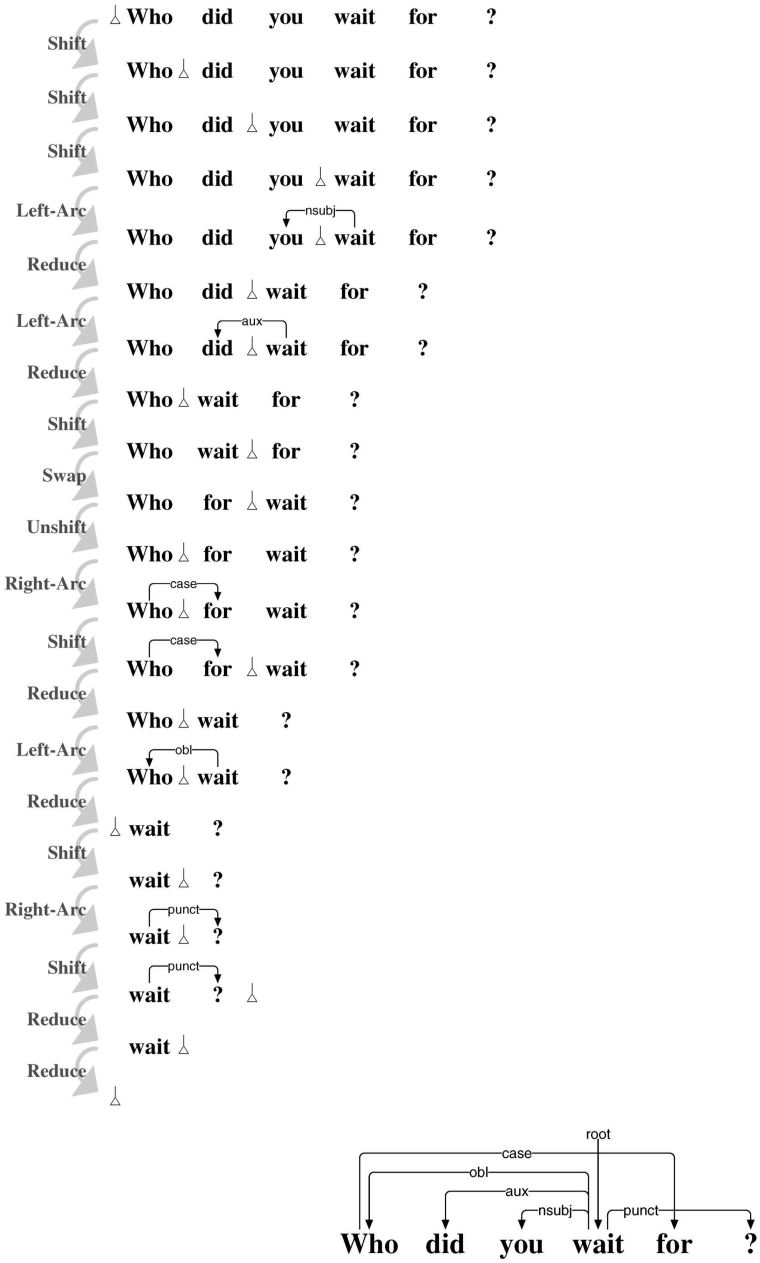

그림11: 상태전이형 알고리즘을 통한 'Who did you wait for?'의 의존관계 분석

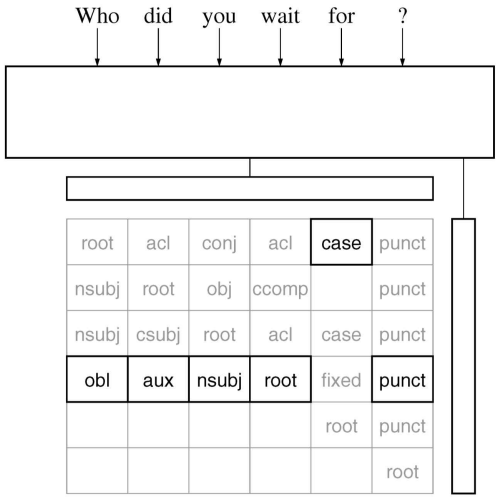

그림12: 인접행렬형 알고리즘을 통한 'Who did you wait for?'의 의존관계 분석

상태전이형 알고리즘은 BERT·RoBERTa·DeBERTa 등의 사전학습모델을 사용하여 정밀도 향상을 기대할 수 있지만[30], 이것은 실제로 적용이 쉽지 않습니다. 한편 인접행렬형 알고리즘은 사전학습모델과 궁합이 좋아서 실제 적용도 간단한 편입니다. Google Colaboratory상의 Transformers를 사용하여 roberta-base-english-ud-goeswith[31]로 'Who did you wait for?'의 인접행렬 로짓[32]을 구하는 프로그램과 그 결과가 그림13입니다. 그림13의 예시에서는 각 열마다 로짓 최대의 요소를 선택하기만 하면 'Who did you wait for?'의 의존구조 그래프를 정확하게 도출할 수 있지만, 현실의 처리에서는 Chu-Liu-Edmonds 등을 사용하게 됩니다.

```
!pip install transformers
import torch,numpy
from transformers import AutoTokenizer,AutoModelForTokenClassification
brt="KoichiYasuoka/roberta-base-english-ud-goeswith"
txt="Who did you wait for?"
tkz=AutoTokenizer.from_pretrained(brt)
mdl=AutoModelForTokenClassification.from_pretrained(brt)
v,l=tkz(txt,return_offsets_mapping=True),mdl.config.id2label
w,u=v["input_ids"],[txt[s:e] for s,e in v["offset_mapping"] if s<e]
x=[w[:i]+[tkz.mask_token_id]+w[i+1:]+[j] for i,j in enumerate(w[1:-1],1)]
with torch.no_grad():
  m=mdl(input_ids=torch.tensor(x)).logits.numpy()[:,1:-2,:]
r=[1 if i==0 else -1 if l[i].endswith("|root") else 0 for i in range(len(l))]
m+=numpy.where(numpy.add.outer(numpy.identity(m.shape[0]),r)==0,0,numpy.nan)
d,p=numpy.nanmax(m,axis=2),numpy.nanargmax(m,axis=2)
print(" ".join(x[:9].rjust(9) for x in u))
for i,j in enumerate(u):
  print("\n"+" ".join("{:9.3f}".format(x) for x in d[i])," ",j)
  print(" ".join(l[x].split("|")[-1][:9].rjust(9) for x in p[i]))
```

30)]Alireza Mohammadshahi, James Henderson: Graph-to-Graph Transformer for Transition-based Dependency Parsing, Findings of the Association for Computational Linguistics: EMNLP 2020 (November 2020), pp.32783289.

31) https://huggingface.co/KoichiYasuoka/roberta-base-english-ud-goeswith

32) 라벨의 생기 확률을 p로 놓을 때 로짓(로그 오드)은 log 1− pp가 됩니다.

Who	did	you	wait	for	?	
2.036	2.522	3.163	3.705	12.105	8.971	Who
root	punct	punct	acl:relcl	case	punct	
3.170	0.727	3.536	2.851	8.797	7.458	did
obj	root	punct	xcomp	case	punct	
3.715	1.873	0.808	4.100	9.255	7.630	you
obj	punct	root	parataxis	case	punct	
11.452	14.337	14.130	15.159	8.094	15.684	wait
obl	aux	nsubj	root	compound:	punct	
5.232	1.943	2.694	2.293	2.099	7.492	for
obj	aux	punct	parataxis	root	punct	
2.858	2.092	2.884	1.936	9.437	1.919	?
goeswith	goeswith	goeswith	goeswith	case	root	

그림13: ‘Who did you wait for?’의 인접행렬 로짓의 도출

2 고전중국어(한문)

2.1 고전중국어 UD의 품사 태깅과 문장 구분

고전중국어(한문)는 단어과 단어 사이에 구분이 없고 문장과 문장 사이에도 구분이 없습니다. 이것이 백문(白文)이라고 불리는 고전중국어의 서사 형태이며, 모르는 사람이 본다면 그저 한자가 연속적으로 나열되어 있을 뿐입니다.

고전중국어 UD의 품사 태깅은 단적으로 말하자면 백문의 각 한자에 UPOS를 부여하는 작업입니다. 그런데 고전중국어에서 단어(漢語)는 1자 이상도 있기 때문에 그 경우는 복수의 한자를 합쳐서 UPOS를 부여합니다. Transformers의 계열 레이블링을 사용한다면 1자의 한어에는 UPOS를 레이블로 부여하고 2자 이상의 한어에는 첫 글자에 ‘B-’를 붙인 UPOS를, 그 이후의 글자에 ‘I-’를 붙인 UPOS를 각각 부여하는 방법을 생각할 수 있습니다. 이렇게 원리적으로는 복수의 한자를 한어로 합치면서 UPOS를 부여하는 것이 가능합니다.

```
!pip install deplacy transformers
class SeqL(object):
  def __init__(self,bert):
    from transformers import pipeline
    self.tagger=pipeline(task="ner",model=bert)
  def __call__(self,text):
    w=[(t["start"],t["end"],t["entity"]) for t in self.tagger(text)]
    u="# text = "+text.replace("\n"," ")+"\n"
    for i,(s,e,p) in enumerate(w,1):
      m,q="_" if i<len(w) and e<w[i][0] else "SpaceAfter=No",p.split("|")
      f="_" if p.find("=")<0 else "|".join(t for t in q if t.find("=")>0)
      u+="\t".join([str(i),text[s:e],"_",q[0],"_",f,"_","_","_",m])+"\n"
    return u+"\n"
nlp=SeqL("KoichiYasuoka/roberta-classical-chinese-base-upos")
doc=nlp("子曰學而時習之不亦説乎有朋自遠方來不亦樂乎人不知而不慍不亦君子乎")
import deplacy
deplacy.serve(doc,port=None)
```

그림14: roberta-classical-chinese-base-upos의 UPOS 부여

Google Colaboratory상의 Transformers를 사용하여 roberta-classical-chinese-base-upos[33]로 '子曰學而時習之不亦説乎有朋自遠方來不亦樂乎人不知而不慍不亦君子乎'에 UPOS를 부여하는 프로그램과 그 결과가 그림14입니다. '君子'는 2자의 한어이며, '君'에 B-NOUN가, '子'에 I-NOUN가 부여됩니다. '君子' 이외에는 한자 1자가 한어 1자로 판정되어 각각에 UPOS가 부여됩니다.

Transformers의 계열 레이블링은 다른 용도로도 적용이 가능합니다. roberta-classical-chinesebase-sentence-segmentation[34]은 고전중국어의 문장 구분에 Transformers의 계열 레이블링을 사용합니다. 구체적으로는 문두의 한자에 대한 레이블을 B, 문말의 한자에 대한 레이블을 E로 하고, B·M·E3·E2·E를 사용하여 각 문장을 나타냅니다. 1자로만 구성된 문장은 레이블을 S로 합니다. 2자로 구성된 문장의 레이블은 B·E로 합니다. 3자로 구성된 문장의 레이블은 B·E2·E로 합니다. 4자로 구성된 문장의 레이블은 B·E3·E2·E로 합니다.

```
!pip install deplacy transformers
class SeqL(object):
  def __init__(self,bert):
    from transformers import pipeline
    self.tagger=pipeline(task="ner",model=bert)
  def __call__(self,text):
    w=[(t["start"],t["end"],t["entity"]) for t in self.tagger(text)]
    u="# text = "+text.replace("\n"," ")+"\n"
    for i,(s,e,p) in enumerate(w,1):
      m,q="_" if i<len(w) and e<w[i][0] else "SpaceAfter=No",p.split("|")
      f="_" if p.find("=")<0 else "|".join(t for t in q if t.find("=")>0)
      u+="\t".join([str(i),text[s:e],"_",q[0],"_",f,"_","_","_",m])+"\n"
    return u+"\n"
nlp=SeqL("KoichiYasuoka/roberta-classical-chinese-base-sentence-segmentation")
doc=nlp("子曰學而時習之不亦説乎有朋自遠方來不亦樂乎人不知而不慍不亦君子乎")
import deplacy
deplacy.serve(doc,port=None)
```

33) https://huggingface.co/KoichiYasuoka/roberta-classical-chinese-base-upos

34) 安岡孝一: Transformersを用いた古典中国語 (漢文)文切りモデルの製作,人文科学とコンピュータシンポジウム「じんもんこん 2021」論文集 (2021年 12月), pp.104-109.

그림15: roberta-classical-chinesebase-sentence-segmentation의 고전중국어의 문장 구분

```
!pip install deplacy transformers
class SeqB(object):
  def __init__(self,bert,trans=[]):
    from transformers import AutoTokenizer,AutoModelForTokenClassification
    import numpy
    self.tokenizer=AutoTokenizer.from_pretrained(bert)
    self.tagger=AutoModelForTokenClassification.from_pretrained(bert)
    x=self.tagger.config.label2id
    self.transition=numpy.full((len(x),len(x)),0 if trans==[] else numpy.nan)
    for f,t in trans:
      self.transition[x[f],x[t]]=0
  def __call__(self,text):
    import torch,numpy
    v=self.tokenizer(text,return_offsets_mapping=True)
    with torch.no_grad():
      m=self.tagger(torch.tensor([v["input_ids"]])).logits[0].numpy()
    for i in range(m.shape[0]-1,0,-1):
      m[i-1]+=numpy.nanmax(m[i]+self.transition,axis=1)
    j,k=v["offset_mapping"],[numpy.nanargmax(m[0])]
    for i in range(1,m.shape[0]):
      k.append(numpy.nanargmax(m[i]+self.transition[k[-1]]))
    w=[(s,e,self.tagger.config.id2label[q]) for (s,e),q in zip(j,k) if s<e]
    u="# text = "+text.replace("\n"," ")+"\n"
    for i,(s,e,p) in enumerate(w,1):
      m="_" if i<len(w) and e<w[i][0] else "SpaceAfter=No"
      u+="\t".join([str(i),text[s:e],"_",p]+["_"]*5+[m])+"\n"
    return u+"\n"
t=[("S","S"),("S","B"),("B","M"),("B","E3"),("B","E2"),("B","E"),
  ("M","M"),("M","E3"),("E3","E2"),("E2","E"),("E","S"),("E","B")]
nlp=SeqB("KoichiYasuoka/roberta-classical-chinese-base-sentence-segmentation",t)
doc=nlp("子曰學而時習之不亦説乎有朋自遠方來不亦樂乎人不知而不慍不亦君子乎")
import deplacy
deplacy.serve(doc,port=None)
```

그림16: 고전중국어의 문장 구분에 Bellman-Ford를 응용

Google Colaboratory상의 Transformers를 사용하여 roberta-classical-chinese-base-s

entence-segmentation으로 '子曰學而時習之不亦説乎有朋自遠方來不亦樂乎人不知而不慍不亦君子乎'를 문장으로 구분하는 프로그램과 그 결과가 그림15입니다. '子曰', '學而時習之', '不亦説乎', '有朋自遠方來', '不亦樂乎', '人不知而不慍', '不亦君子乎'의 7개의 문장으로 나눠진 것을 알 수 있습니다. 그런데 계열 레이블링으로 문장을 나눌 때는 Bellman-Ford[35]와 Conditional Random Fields[36]를 응용하는 것이 더 높은 정밀도를 달성할 수 있을 것이라 생각됩니다(그림16).

2.2 고전중국어 UD의 의존관계 분석

SuPar-Kanbun[37]은 인접행렬형 의존관계 분석 알고리즘을 고전중국어에 실제로 적용합니다. UPOS 품사 태깅과 문장 구분에도 동시에 적용하여 roberta-classical-chinese-base-char를 바탕으로 한 '올인원 설계'라고 할 수 있습니다. Google Colaboratory상의 SuPar-Kanbun를 사용하여 '子曰學而時習之不亦説乎有朋自遠方來不亦樂乎人不知而不慍不亦君子乎'를 품사 태깅, 문장 구분, 의존관계 분석을 하는 프로그램과 그 결과가 그림17입니다. 각 문장의 분석 결과를 순서대로 살펴봅시다.

'子曰'는 동사 '曰'의 주어가 '子'가 되는 문장이고 주어를 나타내는 nsubj로 연결되어 있습니다. '學而時習之'는 병렬접속사 '而'로 인해 '學'과 '習'이 병치되어 '學'과 '習'이 conj로, '習'과 '而'가 cc로 연결되어 있습니다. '之'는 '習'의 목적어로 obj로 연결되어 있습니다. '時'는 '習'의 사격보어로 본래는 obl로 연결되어야 하지만, 고전중국어 UD에서는 obl을 확장한 obl:tmod(시간에 관한 사격보어)를 도입했습니다. '不亦説乎'는 '不'과 '亦' 모두 '説'을 연용수식하여 전부 advmod로 연결되어 있습니다. '乎'는 문장 전체를 수식하는 구 끝의 종조사(sentence particle)인데, 이를 나타낼 수 있는 UD 의존관계 태그가 표4에 없기 때문에 discourse:sp를 도입[38]했습니다. '有朋自遠方來'는 '有朋'과 '自遠方來'의 2개의 구로 구성되어 parataxis로 연결되어 있습니다. '朋'는 '有'의 목적어로 obj로 연결되어 있습니다. '遠方'는 '來'의 사격보어로 본래는 obl로 연결되어야 하지만, 고전중국어 UD에서는 obl을 확장한 obl:lmod(장소에 관한 사격보어)를 도입했습니다. '遠方'는 동사 '遠'(고전중국어 UD에는 형용사라는 분류가 없고 동사로 취급[39]합니다)가 명사 '方'를 연체수식하여 amod로 연결되어 있습니다. 또한 전치사 '自'도 '方'를 수식한다고 간주하여 격 표시를 나타내는 case로 연결되어 있습니다. '不亦樂乎'는 '不亦説乎'와 동일한 의존구조입니다. '人不知而不慍'는 '而'로 인해 '知'와 '慍'가 병치되어 있고 '知'와 '慍'가 conj로, '慍'와 '而'가 cc로 연결되어 있습니다. '人'는 '知'와 '慍' 양쪽의 주어로 생각되지만, 가까운 쪽인 '知'에 nsubj로 연결되어 있습니다. 그리고 '知'와 '慍'는 각각 부정의 '不'가 연용수식하여 전부 advmod로 연결

35) Richard Bellman: On a Routing Problem, Quarterly of Applied Mathematics, Vol.16, No.1 (April 1958), pp.87-90.

36) John Lafferty, Andrew McCallum, Fernando Pereira: Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data, Proceedings of the 18th International Conference on Machine Learning (June 2001), pp.282-289.

37) 安岡孝一,ウィッテルンクリスティアン ,守岡知彦,池田巧,山崎直樹,二階堂善弘,鈴木慎吾,師茂樹,藤田一乗:古典中国語 (漢文) Universal Dependenciesとその応用,情報処理学会論文誌, Vol.63, No.2 (2022年 2月), pp.355-363.

38) Herman Leung, Rafa¨el Poiret, Tak-sum Wong, Xinying Chen, Kim Gerdes and John Lee: Developing Universal Dependencies for Mandarin Chinese, 12th Workshop on Asian Language Resources (December 2016), pp.20-29.

39) 安岡孝一,ウィッテルンクリスティアン ,守岡知彦,池田巧,山崎直樹,二階堂善弘,鈴木慎吾,師茂樹:古典中国語 (漢文)の形態素解析とその応用,情報処理学会論文誌, Vol.59, No.2 (2018年 2月), pp.323-331.

되어 있습니다. '不亦君子乎'는 '不亦説乎'나 '不亦樂乎'와 비슷한 의존구조이지만 '君子'가 명사이며 계사문입니다(다만, 주어도 계사도 없습니다).

```
!pip install suparkanbun
import suparkanbun
nlp=suparkanbun.load(Danku=True)
doc=nlp("子曰學而時習之不亦説乎有朋自遠方來不亦樂乎人不知而不慍不亦君子乎")
import deplacy
deplacy.serve(doc,port=None)
```

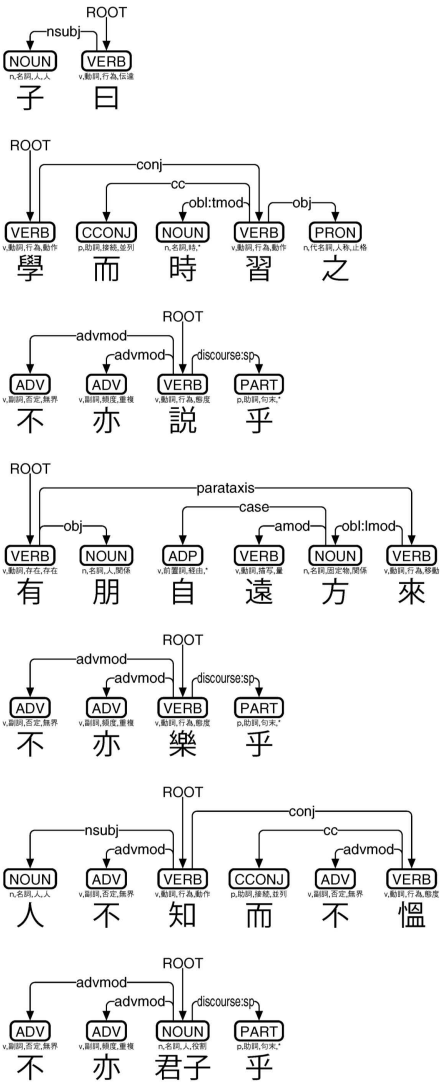

그림17: SuPar-Kanbun의 고전중국어의 의존관계 분석

이와 같은 형태로 SuPar-Kanbun는 한자의 열인 백문에 품사 태깅을 하고 한어의 열의 의존관계를 분석하여 한문의 의존구조를 추출합니다. 또한 고전중국어 UD의 의존관계 태그에는 표4와 그림17에 더해 수동문의 주어를 나타내는 nsubj:pass, 중첩을 나타내는 compound:redup, 동사 병렬을 나타내는 flat:vv, 고전중국어 이외를 나타내는 flat:foreign도 사용됩니다.[40]

2.3 고전중국어 미니 DeBERTa 모델의 제작

Google Colaboratory상의 esupar[41]에서 고전중국어 미니 DeBERTa 모델을 제작해 봅시다. 그림18부터 그림22가 그 순서입니다.

먼저 훈련을 위한 백문을 준비합니다(그림18). train.txt의 백문 재료로, 여기서는 UD_Classical_Chinese-Kyoto[42]를 사용하였지만, 원하는 바에 따라 다른 백문을 증량해도 괜찮습니다. 그리고 UD_Classical_Chinese-Kyoto의 각 CoNLL-U 파일은 품사 태깅·의존관계 분석의 학습에도 사용합니다.

```
!pip install esupar
import os
url="https://github.com/UniversalDependencies/UD_Classical_Chinese-Kyoto"
d=os.path.basename(url)
!test -d {d} || git clone --depth=1 {url}
!for F in train dev test ; do cp {d}/*-$$F.conllu $$F.conllu ; done
!sed -n "s/^# text = //p" *.conllu > train.txt
```

그림18: 고전중국어 미니 모델 제작을 위한 준비

```
from transformers import BertTokenizerFast
from tokenizers.pre_tokenizers import Sequence,Whitespace,Split
from tokenizers import Regex
s=["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"]
with open("train.txt","r",encoding="utf-8") as r:
  v=set(c for c in r.read() if not c.isspace())
with open("vocab.txt","w",encoding="utf-8") as w:
  print("\n".join(s+sorted(v)),file=w)
tkz=BertTokenizerFast(vocab_file="vocab.txt",never_split=s,
  do_lower_case=False,strip_accents=False,tokenize_chinese_chars=True)
b=tkz.backend_tokenizer
b.pre_tokenizer=Sequence([Whitespace(),Split(Regex("."),"isolated")])
b.decoder.prefix=b.model.continuing_subword_prefix=""
tkz.save_pretrained("my-dir/tokenizer-lzh")
```

그림19: 고전중국어용 단문자 토크나이저의 작성

40) https://universaldependencies.org/lzh/#syntax
41) https://github.com/KoichiYasuoka/esupar
42) Koichi Yasuoka: Universal Dependencies Treebank of the Four Books in Classical Chinese, DADH2019: 10th International Conference of Digital Archives and Digital Humanities (December 2019), pp.20-28.

```
from transformers import (AutoTokenizer,DebertaV2Config,DebertaV2ForMaskedLM,
  DataCollatorForLanguageModeling,TrainingArguments,Trainer)
tkz=AutoTokenizer.from_pretrained("my-dir/tokenizer-lzh",model_max_length=128)
tkz.backend_tokenizer.model.max_input_chars_per_word=tkz.model_max_length
t=tkz.convert_tokens_to_ids(["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"])
cfg=DebertaV2Config(hidden_size=256,num_hidden_layers=12,num_attention_heads=4,
  intermediate_size=768,max_position_embeddings=tkz.model_max_length,
  vocab_size=len(tkz),tokenizer_class=type(tkz).__name__,
  bos_token_id=t[0],pad_token_id=t[1],eos_token_id=t[2])
arg=TrainingArguments(num_train_epochs=3,per_device_train_batch_size=64,
  output_dir="/tmp",overwrite_output_dir=True,save_total_limit=2)
class ReadLineDS(object):
  def __init__(self,file,tokenizer):
    self.tokenizer=tokenizer
    with open(file,"r",encoding="utf-8") as r:
      self.lines=[s.strip() for s in r if s.strip()!=""]
  __len__=lambda self:len(self.lines)
  __getitem__=lambda self,i:self.tokenizer(self.lines[i],truncation=True,
    add_special_tokens=True,max_length=self.tokenizer.model_max_length-2)
trn=Trainer(args=arg,data_collator=DataCollatorForLanguageModeling(tkz),
  model=DebertaV2ForMaskedLM(cfg),train_dataset=ReadLineDS("train.txt",tkz))
trn.train()
trn.save_model("my-dir/deberta-lzh")
tkz.save_pretrained("my-dir/deberta-lzh")
```

그림20: 고전중국어 미니 DeBERTa 모델의 작성

```
import torch
from transformers import AutoTokenizer,AutoModelForMaskedLM
from esupar.tradify import tradify
tkz=AutoTokenizer.from_pretrained("my-dir/deberta-lzh")
mdl=AutoModelForMaskedLM.from_pretrained("my-dir/deberta-lzh")
c=[(k,v) for k,v in tradify.items() if tkz.add_tokens([k,v])==1]
e=mdl.resize_token_embeddings(len(tkz))
with torch.no_grad():
  for k,v in c:
    t=sorted(tkz.convert_tokens_to_ids([k,v]))
    e.weight[t[1],:]=e.weight[t[0],:]
mdl.set_input_embeddings(e)
mdl.save_pretrained("my-dir/deberta-lzh-ext")
tkz.save_pretrained("my-dir/deberta-lzh-ext")
```

그림21: 고전중국어 미니 DeBERTa 모델의 확장(이체자 추가)

```
!python -m esupar.train my-dir/deberta-lzh-ext my-dir/deberta-lzh-upos .
```

그림22: 고전중국어용 품자 태깅·의존관계 분석 미니 모델의 제작

이어서 입력문자열을 단문자로 분석하기 위한 토크나이저를 my-dir/tokenizer-lzh에 작성합니다. 그림19에서는 BertTokenizerFast라는 Transformers의 토크나이저를 사용하여 단문자 토크나이저를 작성합니다.

다음으로 train.txt의 백문으로 DeBERTa 모델을 my-dir/deberta-lzh에 작성합니다. 그림20에서는 입출력폭 128토큰, 입출력 벡터 256차원, 깊이 12층, 어텐션 헤드 4개, 중간 벡터 768차원이라는 약간 작은 모델로 만들었으며, CPU라면 1시간 정도, GPU라면 10분 정도에 작성할 수 있습니다. 그런데 이 DeBERTa 모델은 번체자로 만들어졌으며 일본의 상용한자와 중국의 간체자는 포함되지 않습니다. 그래서 이런 이체자에 대해 벡터를 복사하여

새로운 모델을 my-dir/deberta-lzh-ext에 보존합니다(그림21).

마지막으로 my-dir/deberta-lzh-ext와 UD_Classical_Chinese-Kyoto로 품사 태깅·의존관계 분석 모델 my-dir/deberta-lzh-upos을 제작합니다(그림22). 다만 인접행렬형 알고리즘의 학습은 시간이 오래 걸려 GPU라도 3시간 정도 걸립니다.

my-dir/deberta-lzh-upos가 제대로 제작되었는지 esupar로 테스트를 해봅니다(그림23). 번체자뿐만 아니라 일본의 상용한자나 중국의 간체자로 쓰인 한문도 품사 태깅·의존관계 분석이 가능하기 때문에 꼭 한번 시험해 보시기 바랍니다.

참고로 그림24는 my-dir/deberta-lzh-ext와 UD_Classical_Chinese-Kyoto로 미니 문장 구분 모델 my-dir/deberta-lzh-seg를 작성(파인튜닝)하는 프로그램입니다. 다만 미니 모델로는 문장 구분의 정밀도가 높아지지 않기 때문에 더 큰 모델을 만드는 편이 좋습니다.

```
!pip install esupar
import esupar
nlp=esupar.load("my-dir/deberta-lzh-upos")
doc=nlp("劳心者治人")
import deplacy
deplacy.serve(doc,port=None)
```

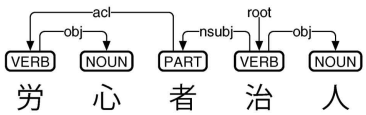

그림23: 고전중국어용 품자 태깅·의존관계 분석 미니 모델의 테스트

```
from transformers import (AutoTokenizer,AutoModelForTokenClassification,
  AutoConfig,DataCollatorForTokenClassification,TrainingArguments,Trainer)
src,lid="my-dir/deberta-lzh-ext",{"B":0,"E":1,"E2":2,"E3":3,"M":4,"S":5}
tkz,cfg=AutoTokenizer.from_pretrained(src),AutoConfig.from_pretrained(src,
  num_labels=len(lid),label2id=lid,id2label={i:l for l,i in lid.items()})
wd=cfg.max_position_embeddings-3
dat=[{"input_ids":[tkz.cls_token_id],"labels":[5]}]
with open("train.conllu","r",encoding="utf-8") as r:
  for t in [s[9:].strip() for s in r if s.startswith("# text = ")]:
    i=tkz(t,add_special_tokens=False)["input_ids"]
    j=[0 if len(i)>1 else 5]+[min(k,4) for k in range(len(i)-1,0,-1)]
    if len(dat[-1]["input_ids"])+len(i)>wd:
      dat.append({"input_ids":[tkz.cls_token_id]+i[0:wd],"labels":[5]+j[0:wd]})
    elif len(i)>0:
      dat[-1]["input_ids"].extend(i)
      dat[-1]["labels"].extend(j)
trn=Trainer(train_dataset=dat,args=TrainingArguments(num_train_epochs=3,
  per_device_train_batch_size=64,output_dir="/tmp",overwrite_output_dir=True,
  save_total_limit=2),data_collator=DataCollatorForTokenClassification(tkz),
  model=AutoModelForTokenClassification.from_pretrained(src,config=cfg))
trn.train()
trn.save_model("my-dir/deberta-lzh-seg")
tkz.save_pretrained("my-dir/deberta-lzh-seg")
```

그림24: 고전중국어 문장 구분 미니 모델의 작성

3 일본어

서사 언어로서의 일본어는 단어와 단어 사이에 구분이 없습니다. 애초에 단어의 길이에 제한이 없기 때문에 다양한 단어의 길이가 사용되고[43] 있습니다(그림25). 일본어 UD에서는 이들 가운데 일본 국립국어연구소(이하 국어연) 단단위와 장단위가 지원됩니다. 한편, 일본어의 사전학습모델의 토큰은 국어연 단단위(또는 그 세분화)나 문자 단위가 많지만, 그림25 이외의 토큰을 채용하는 모델도 있어서 상당히 혼란스럽습니다.

国語研短単位 || 全 || 学年 || に || わたっ|| て || 小 || 学校 || の || 国語 || の ||
教科 || 書 || に || 大量 || の || 挿し絵 || が || 用い || られ || て || いる ||

国語研長単位 || 全学年 || にわたって || 小学校 || の || 国語 || の ||
教科書 || に || 大量 || の || 挿し絵 || が || 用い || られ || ている ||

文節 || 全学年にわたって || 小学校の || 国語の ||
教科書に || 大量の || 挿し絵が || 用いられている ||

그림25: 국어연 단단위·국어연 장단위·문절

3.1 국어연 단단위를 이용한 품사 태깅·의존관계 분석

SuPar-UniDic[44]은 아오조라문고(+Wikipedia) BERT 모델[45]를 베이스로 인접행렬형 의존관계 분석 알고리즘을 국어연 단단위용으로 적용합니다. 형태소 분석(단어 구분·XPOS 품사 태깅)부에는 MeCab[46]과 UniDic[47]을 이용합니다. Google Colaboratory상의 SuPar-UniDic을 사용하여 '全学年にわたって小学校の国語の教科書に大量の挿し絵が用いられている'의 품사 태깅·의존관계 분석을 하는 프로그램과 그 결과가 그림26입니다. 분석 결과를 살펴보겠습니다.

국어연 단단위 UD에서는 '全学年', '小学校', '教科書'가 복수의 단어로 나눠져 compound로 연결되는 것이 특징입니다. '小学校の国語の教科書'는 명사가 명사를 수식하여 nmod로 연결되며 동시에 'の'가 case로 이어져 있습니다. '大量の挿し絵'도 마찬가지입니다. '全学年'은 'わたって'의 니격(ニ格)이며 obl로 연결된 동시에 'に'가 case로 연결되어 있습니다. '挿し絵'는 '用いられて'의 가격(ガ格)이고 nsubj로 연결되는 동시에 'が'가 case로 이어져 있습니다. '全学年にわたって'는 '挿し絵が用いられて'의 연용수식절이며 advcl로 연결되어 있습니다. 'いる'는 UniDic 품사(XPOS)가 '동사-비자립 가능'인데, 여기서는 비자립으로 간주하여 UPOS를 AUX로 하는 동시에 aux로 연결되어 있습니다.

---

43) Mai Omura, Aya Wakasa, Masayuki Asahara: Word Delimitation Issues in UD Japanese, Proceedings of the 5th Workshop on Universal Dependencies (December 2021), pp.142-150.

44) 安岡孝一:世界の Universal Dependenciesと係り受け解析ツール群,第 3回 Universal Dependencies公開研究会 (2021年 6月).

45) https://github.com/akirakubo/bert-japanese-aozora

46) Taku Kudo, Kaoru Yamamoto, Yuji Matsumoto: Applying Conditional Random Fields to Japanese Morphological Analysis, Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing (July 2004), pp.230-237.

47) 伝康晴,小木曽智信,小椋秀樹,山田篤,峯松信明,内元清貴,小磯花絵:ユーパス日本語学のための言語資源:形態素解析用電子化辞書の開発とその応用,日本語科学,第 22号 (2007年 10月), pp.101-123.

```
!pip install suparunidic
import suparunidic
nlp=suparunidic.load("gendai")
doc=nlp("全学年にわたって小学校の国語の教科書に大量の挿し絵が用いられている")
import deplacy
deplacy.serve(doc,port=None)
```

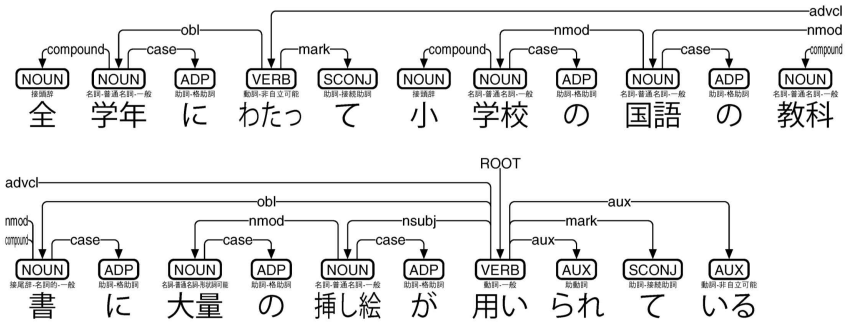

그림26: SuPar-UniDic을 이용한 일본어 의존관계 분석(국어연 단단위)

국어연 단단위 UD에서는 XPOS와 UPOS가 미묘하게 괴리되어 있습니다. XPOS는 UniDic 품사를 사용하지만, UPOS의 품사 태깅 규칙과는 조금 다르기 때문입니다. 극단적인 예로 '難儀な難儀は難儀する'(그림27)를 살펴봅시다. '難儀'의 UniDic 품사는 '명사-보통명사-사변(サ변) 형상사 가능'이지만, '難儀な'는 형상사(형용동사)로 취급해야 하며 UPOS는 ADJ가 타당합니다. '難儀する'는 '사변 동사(경동사)'로 취급해야 하며 UPOS는 VERB가 타당합니다. 다만 '難儀な'와 '難儀する'도 국어연 단단위로는 두 개의 단어로 나눠지기 때문에 '難儀'의 UPOS에 각각 ADJ와 VERB를 부여합니다.

```
!pip install suparunidic
import suparunidic
nlp=suparunidic.load("gendai")
doc=nlp("難儀な難儀は難儀する")
import deplacy
deplacy.serve(doc,port=None)
```

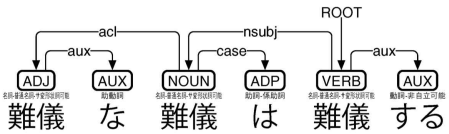

그림27: 국어연 단단위 UD의 XPOS(UniDic 품사)와 UPOS의 관계

3.2 국어연 단단위 미니 DeBERTa 모델의 제작

Google Colaboratory상의 esupar로 국어연 단단위 미니 DeBERTa 모델을 제작해 봅시다. 그림28부터 31이 그 순서입니다.

먼저 훈련을 위한 문장을 준비합니다(그림28). train.txt의 문장 재료로, 여기서는 UD_Japanese-GSD[48]와 Wikitext-JA[49]를 사용했지만, 원하는 바에 따라 다른 일본어 문장을 증량

48) 松田寛,若狭絢,山下華代,大村舞,浅原正幸: UD Japanese GSDの再整備と固有表現情報付与,言語処理学会第 26回年次大会発表論文集 (2020年 3月), pp.133-136.

해도 괜찮습니다. 그리고 UD_Japanese-GSD의 각 CoNLL-U 파일은 품사 태깅·의존관계 분석 학습에도 사용합니다.

```
!pip install esupar fugashi unidic-lite
import os,urllib.request
url="https://github.com/UniversalDependencies/UD_Japanese-GSD"
d=os.path.basename(url)
!test -d {d} || git clone --depth=1 {url}
!for F in train dev test ; do cp {d}/*-$$F.conllu $$F.conllu ; done
!sed -n "s/^# text = //p" *.conllu > train.txt
url="http://www.lsta.media.kyoto-u.ac.jp/resource/data/wikitext-ja/"
with open("train.txt","a",encoding="utf-8") as w:
  for t in ["Featured_Contents.txt","Good_Contents.txt"]:
    with urllib.request.urlopen(url+t) as r:
      print(r.read().decode("utf-8").replace("。","。\n"),file=w)
```

그림28: 국어연 단단위 미니 모델 제작을 위한 준비

다음으로 입력문을 국어연 단단위로 분해하기 위한 토크나이저를 my-dir/tokenizer-suw에 작성합니다. 그림29에서는 fugashi[50]와 unidic-lite[51]로 train.txt를 국어연 단단위로 구분해서 그것을 BertJapaneseTokenizer라는 Transformers의 토크나이저로 작업한 후 국어연 단단위 토크나이저를 작성합니다. 어휘 수(vocab_size)는 8,000으로, 토크나이저에서 사용할 수 있는 글자 종류의 상한(limit_alphabet)은 3,000으로 좁혔지만, 조금 더 큰 편이 좋을 것 같습니다.

```
!fugashi -Owakati < train.txt > token.txt
from transformers import BertJapaneseTokenizer
from tokenizers import BertWordPieceTokenizer
s=["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"]
wpt=BertWordPieceTokenizer(lowercase=False,strip_accents=False,
  handle_chinese_chars=False)
wpt.train(files=["token.txt"],vocab_size=8000,limit_alphabet=3000,
  special_tokens=s)
wpt.save_model(".")
tkz=BertJapaneseTokenizer(vocab_file="vocab.txt",word_tokenizer_type="mecab",
  mecab_kwargs={"mecab_dic":"unidic_lite"},do_lower_case=False,never_split=s)
tkz.save_pretrained("my-dir/tokenizer-suw")
```

그림29: 국어연 단단위 토크나이저의 작성

49) http://www.lsta.media.kyoto-u.ac.jp/resource/data/wikitext-ja/
50) https://github.com/polm/fugashi
51) https://github.com/polm/unidic-lite

```
from transformers import (AutoTokenizer,DebertaV2Config,DebertaV2ForMaskedLM,
  DataCollatorForLanguageModeling,TrainingArguments,Trainer)
tkz=AutoTokenizer.from_pretrained("my-dir/tokenizer-suw",model_max_length=128)
t=tkz.convert_tokens_to_ids(["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"])
cfg=DebertaV2Config(hidden_size=256,num_hidden_layers=12,num_attention_heads=4,
  intermediate_size=768,max_position_embeddings=tkz.model_max_length,
  vocab_size=len(tkz),tokenizer_class=type(tkz).__name__,
  bos_token_id=t[0],pad_token_id=t[1],eos_token_id=t[2])
arg=TrainingArguments(num_train_epochs=3,per_device_train_batch_size=64,
  output_dir="/tmp",overwrite_output_dir=True,save_total_limit=2)
class ReadLineDS(object):
  def __init__(self,file,tokenizer):
    self.tokenizer=tokenizer
    with open(file,"r",encoding="utf-8") as r:
      self.lines=[s.strip() for s in r if s.strip()!=""]
  __len__=lambda self:len(self.lines)
  __getitem__=lambda self,i:self.tokenizer(self.lines[i],truncation=True,
    add_special_tokens=True,max_length=self.tokenizer.model_max_length-2)
trn=Trainer(args=arg,data_collator=DataCollatorForLanguageModeling(tkz),
  model=DebertaV2ForMaskedLM(cfg),train_dataset=ReadLineDS("train.txt",tkz))
trn.train()
trn.save_model("my-dir/deberta-suw")
tkz.save_pretrained("my-dir/deberta-suw")
```

그림30: 국어연 단단위 미니 DeBERTa 모델의 작성

```
!python -m esupar.train my-dir/deberta-suw my-dir/deberta-suw-upos .
```

그림31: 국어연 단단위용 품사 태깅·의존관계 분석 미니 모델의 제작

```
!pip install esupar fugashi unidic-lite pytokenizations
import esupar
nlp=esupar.load("my-dir/deberta-suw-upos")
doc=nlp("全学年にわたって小学校の国語の教科書に大量の挿し絵が用いられている")
import deplacy
deplacy.serve(doc,port=None)
```

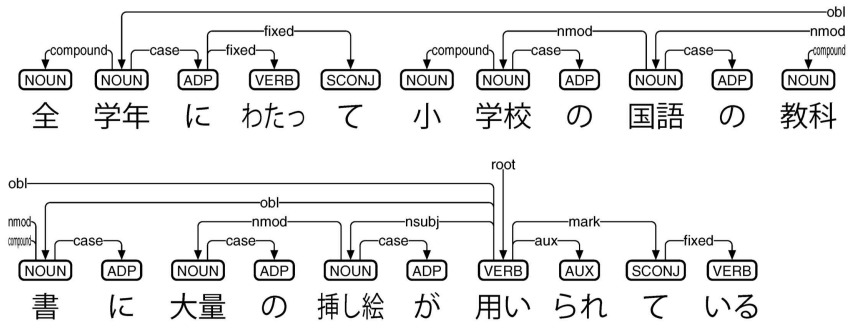

그림32: 국어연 단단위용 품사 태깅·의존관계 분석 미니 모델의 테스트

다음으로 train.txt의 문장으로 DeBERTa 모델을 my-dir/deberta-suw에 만들어봅시다. 그림30에서는 입출력 폭 128토큰, 입출력 벡터 256차원, 깊이 12층, 어텐션 헤드 4개, 중간 벡터 768차원이라는 비교적 작은 모델로 만들었지만, GPU라도 3시간 정도 걸립니다.

마지막으로 my-dir/deberta-suw와 UD_Japanese-GSD로 품사 태깅·의존관계 분석 모델 my-dir/deberta-suw-upos를 제작합니다(그림31). 다만 인접행렬형 알고리즘의 학습은 시간이 오래 걸려 GPU로도 2시간 정도 걸립니다.

my-dir/deberta-suw-upos가 제대로 제작이 되었는지 esupar로 테스트를 합니다. esupar에서 BertJapaneseTokenizer를 사용하는 경우에는 fugashi와 unidic-lite와 함께 pytokenizations[52]도 필요합니다. 그림32의 예시에서는 'にわたって', 'ている'의 링크에 fixed가 나타나 그림26과는 다른 결과가 나왔습니다.

3.3 국어연 장단위를 이용한 품사 태깅과 의존관계 분석

국어연 장단위 토크나이저 Japanese-LUW-Tokenizer[53]와 아오조라문고 RoBERTa 모델 roberta-base-japanese-luw-upos[54]를 이용하여 국어연 장단위를 이용한 품사 태깅을 생각해 봅시다. Google Colaboratory상의 Transformers를 사용하여 '全学年にわたって小学校の国語の教科書に大量の挿し絵が用いられている'에 UPOS를 부여하는 프로그램과 그 결과가 그림33입니다. 각 토큰에 계열 레이블링으로 UPOS를 부여했지만, Japanese-LUW-Tokenizer가 완벽하지 않아 조금 짧게 '全', '学年', 'にわたって', '小学校', 'の', '国語', 'の', '教科書', 'に', '大量', 'の', '挿し', '絵', 'が', '用い', 'られて', 'いる'로 분할되기 때문에 '全'과 '挿し'에 B-NOUN, '学年'과 '絵'에 I-NOUN가 부여됩니다. 한편 '大量'에는 ADJ가 부여되지만, 이것은 NOUN이 더 적절하다고 생각됩니다.

이어서 국어연 장단위의 의존관계 분석[55]도 살펴보겠습니다. Google Colaboratory상의 esupar를 사용하여 '全学年にわたって小学校の国語の教科書に大量の挿し絵が用いられている'의 의존관계 분석을 하는 프로그램과 그 결과가 그림34입니다. '小学校の国語の教科書'는 명사가 명사를 수식하여 nmod로 연결되어 있으며 동시에 'の'가 case로 이어져 있습니다. '大量の挿し絵'도 동일해야 하지만, '大量'가 NOUN이 아니라 ADJ가 되어버립니다. '挿し絵'는 가격이며, nsubj로 연결되는 동시에 'が'가 case로 이어져 있습니다. '教科書'는 니격이며, obl로 연결되는 동시에 'に'가 case로 이어져 있습니다. '全学年'도 니격으로 판단되어 obl로 연결되어 있으며 동시에 'にわたって'가 case로 이어져 있습니다.

3.4 국어연 장단위 미니 DeBERTa 모델의 제작

Google Colaboratory상의 esupar로 국어연 장단위 미니 DeBERTa 모델을 제작해 봅시다. 그림35는 그림28~30에서 작성한 my-dir/deberta-suw를 국어연 장단위에 사용하는 형태로 품사 태깅·의존관계 분석 모델 my-dir/deberta-luw-upos를 제작한 것입니다. 국어연 단단위의 BertJapaneseTokenizer(내부적으로는 fugashi와 unidic-lite)를 사용하기 때문에 국어연 장단위의 UD_Japanese-GSDLUW[56]에 비해 토큰이 짧아지지만, esupar가 UPOS 부여에 'B-', 'I-'를 구사하여 제대로 잘 작동할 것입니다(그림36).

---

52) https://github.com/explosion/tokenizations
53) https://github.com/KoichiYasuoka/Japanese-LUW-Tokenizer
54) https://huggingface.co/KoichiYasuoka/roberta-base-japanese-luw-upos
55) 安岡孝一: Transformersと国語研長単位による日本語係り受け解析モデルの製作,情報処理学会研究報告, Vol.2022-CH-128『人文科学とコンピュータ』, No.7 (2022年 2月 19日), pp.1-8.
56) 大村舞,若狭絢,浅原正幸:国語研長単位に基づく UD Japanese,言語処理学会第 28回年次大会発表論文集 (2022年 3月), pp.1618-1623.

```
!pip install deplacy transformers
class SeqL(object):
  def __init__(self,bert):
    from transformers import pipeline
    self.tagger=pipeline(task="ner",model=bert)
  def __call__(self,text):
    w=[(t["start"],t["end"],t["entity"]) for t in self.tagger(text)]
    u="# text = "+text.replace("\n"," ")+"\n"
    for i,(s,e,p) in enumerate(w,1):
      m,q="_" if i<len(w) and e<w[i][0] else "SpaceAfter=No",p.split("|")
      f="_" if p.find("=")<0 else "|".join(t for t in q if t.find("=")>0)
      u+="\t".join([str(i),text[s:e],"_",q[0],"_",f,"_","_","_",m])+"\n"
    return u+"\n"
nlp=SeqL("KoichiYasuoka/roberta-base-japanese-luw-upos")
doc=nlp("全学年にわたって小学校の国語の教科書に大量の挿し絵が用いられている")
import deplacy
deplacy.serve(doc,port=None)
```

그림33: roberta-base-japanese-luw-upos의 UPOS 부여

```
!pip install esupar
import esupar
nlp=esupar.load("KoichiYasuoka/roberta-base-japanese-luw-upos")
doc=nlp("全学年にわたって小学校の国語の教科書に大量の挿し絵が用いられている")
import deplacy
deplacy.serve(doc,port=None)
```

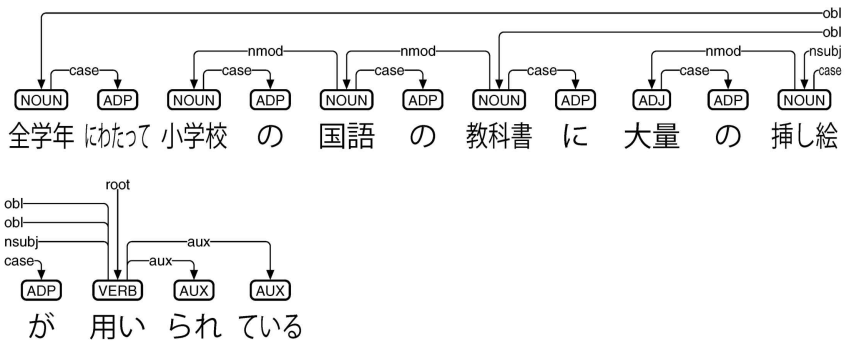

그림34: roberta-base-japanese-luw-upos의 일본어 의존관계 분석(국어연 장단위)

```
!pip install esupar fugashi unidic-lite pytokenizations
url="https://github.com/UniversalDependencies/UD_Japanese-GSDLUW"
!python -m esupar.train my-dir/deberta-suw my-dir/deberta-luw-upos {url}
```

그림35: 국어연 장단위용 품사 태깅·의존관계 분석 미니 모델의 제작

```
!pip install esupar fugashi unidic-lite pytokenizations
import esupar
nlp=esupar.load("my-dir/deberta-luw-upos")
doc=nlp("全学年にわたって小学校の国語の教科書に大量の挿し絵が用いられている")
import deplacy
deplacy.serve(doc,port=None)
```

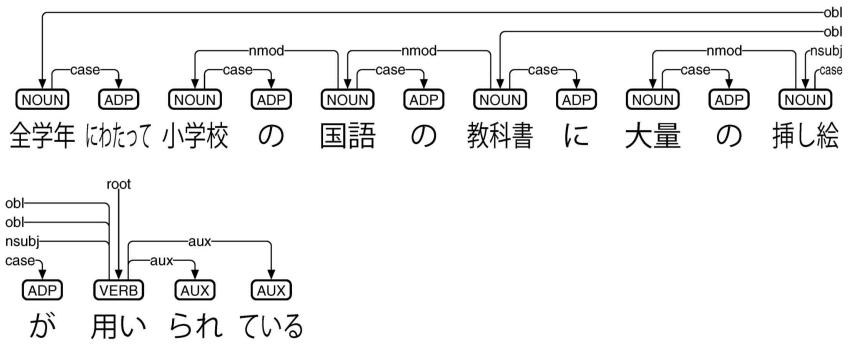

그림36: 국어연 장단위용 품사 태깅·의존관계 분석 미니 모델의 테스트

3.5 문절을 이용한 의존관계 분석

GiNZA[57]는 국어연 단단위를 이용한 상태전이형 의존관계 분석 알고리즘을 사용하고, 문절을 분석 단위로 한 의존관계 분석이 가능합니다. 다만 문절을 단위로 하는 의존관계는 관습적으로 문두에서 문말로 링크하기 때문에[58], 화살표 방향이 일본어 UD와 반대가 되어버립니다.

Google Colaboratory상의 GiNZA를 사용하여 '全学年にわたって小学校の国語の教科書に大量の挿し絵が用いられている'를 문절 의존관계 분석을 하는 프로그램과 그 결과(가시화는 Graphviz[59]를 사용)가 그림37입니다. 문절을 단위로 하는 의존관계 분석은 UD과 완전히 다르기 때문에 여기서는 참고 수준에서 끝내겠습니다.

```
!pip install ja_ginza ginza deplacy
import ja_ginza,ginza
nlp=ja_ginza.load()
doc=nlp("全学年にわたって小学校の国語の教科書に大量の挿し絵が用いられている")
d=ginza.bunsetu_spans(doc)
from deplacy.deprelja import deprelja
g="digraph{"+";".join(['x{}[label="{}"]'.format(b.start,b.text) for b in d]+
  ['x{}->x{}[label="{}",fontsize=9]'.format(ginza.bunsetu_span(t).start,
  b.start,deprelja[t.dep_]) for b in d for t in b.lefts])+"}"
import graphviz
graphviz.Source(g)
```

57) 松田寛: GiNZA -Universal Dependenciesによる実用的日本語解析,自然言語処理, Vol.27, No.3 (2020年 9月), pp.695-701.
58) 吉田将:二文節間の係り受けを基礎とした日本語文の構文分析,電子通信学会論文誌, Vol.55-D, No.4 (1972年 4月), pp.238-244.
59) https://graphviz.readthedocs.io

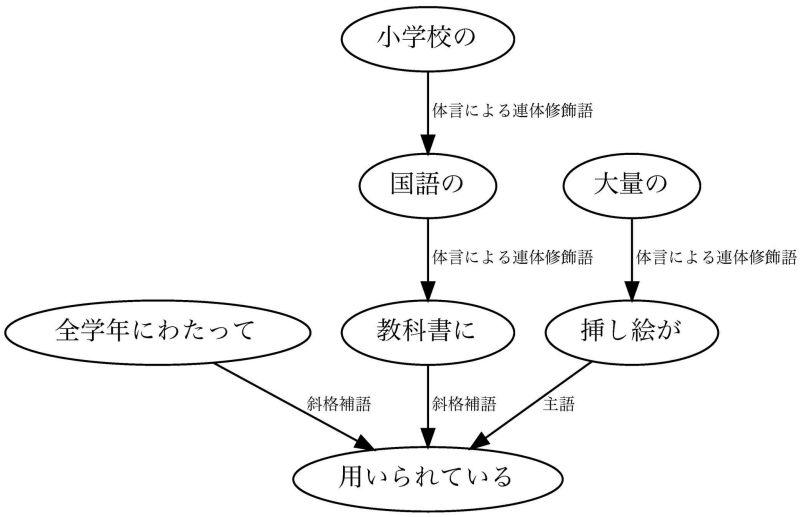

그림37: 문절을 이용한 일본어 의존관계 분석