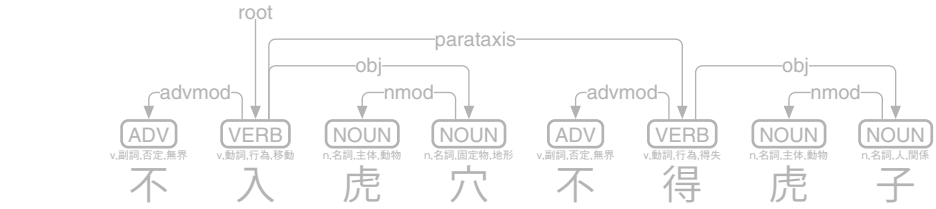
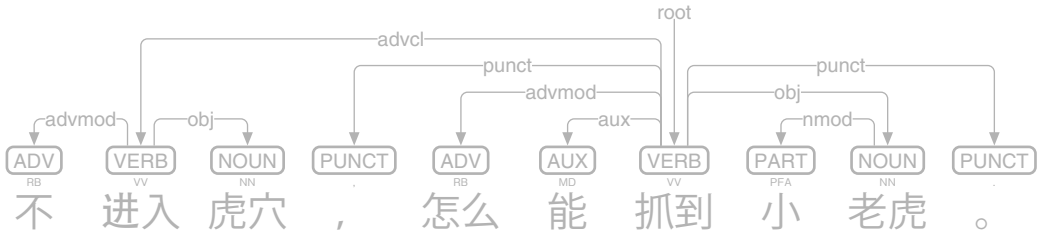
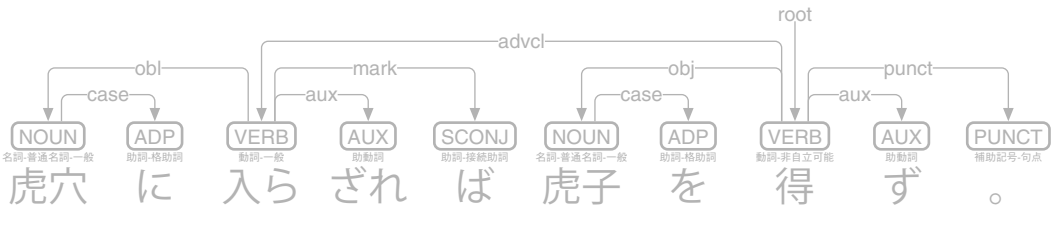
Universal Dependencies と BERT/RoBERTa モデルによる

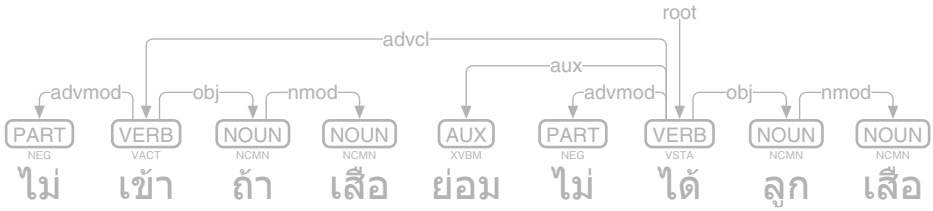
多言語情報処理

安岡孝一









2022 年 5 月版

Universal Dependencies と BERT/RoBERTa モデルによる多言語情報処理 (2022 年 5 月版)

安岡孝一 (京都大学人文科学研究所附属東アジア人文情報学研究センター)

1 概説編 **4**
1.1 Universal Dependencies と CoNLL-U 4
1.2 依存文法理論と Universal Dependencies 8
1.3 BERT/RoBERTa/DeBERTa モデルと穴埋めゲーム 13
1.4 Universal Dependencies における係り受け解析 15

2 古典中国語 (漢文) 編 **18**
2.1 古典中国語 UD における品詞付与と文切り 18
2.2 古典中国語 UD における係り受け解析 20
2.3 古典中国語ミニ RoBERTa モデルの製作 22

3 日本語編 **26**
3.1 国語研短単位にもとづく品詞付与と係り受け解析 26
3.2 国語研短単位ミニ RoBERTa モデルの製作 28
3.3 国語研長単位にもとづく品詞付与と係り受け解析 30
3.4 国語研長単位ミニ RoBERTa モデルの製作 32
3.5 文節にもとづく係り受け解析 32

4 英語編 **34**
4.1 英語 UD における品詞付与と係り受け解析 34
4.2 英語ミニ RoBERTa モデルの製作 35

5 フランス語編 **38**
5.1 フランス語 UD における品詞付与と係り受け解析 38
5.2 フランス語ミニ RoBERTa モデルの製作 39

6 ドイツ語編 **42**
6.1 ドイツ語 UD における品詞付与と係り受け解析 42
6.2 ドイツ語ミニ RoBERTa モデルの製作 43

7 オランダ語編 **46**
7.1 オランダ語 UD における品詞付与と係り受け解析 46
7.2 オランダ語ミニ RoBERTa モデルの製作 46

8 タイ語編 **49**
8.1 タイ語 UD における品詞付与 49
8.2 タイ語 UD における係り受け解析 50
8.3 タイ語ミニ DeBERTa モデルの製作 50

9 ロシア語編 **53**
9.1 ロシア語 UD における品詞付与と係り受け解析 53
9.2 ロシア語ミニ RoBERTa モデルの製作 53

10 ウクライナ語編 **56**
10.1 ウクライナ語 UD における品詞付与と係り受け解析 56
10.2 ウクライナ語ミニ RoBERTa モデルの製作 56

11 ベラルーシ語編 **59**
11.1 ベラルーシ語 UD における品詞付与と係り受け解析 59
11.2 ベラルーシ語ミニ RoBERTa モデルの製作 60

12 セルビア語編 **62**
12.1 セルビア語 UD における品詞付与と係り受け解析 62
12.2 セルビア語ミニ RoBERTa モデルの製作 63

13 クロアチア語編 **67**
13.1 クロアチア語 UD における品詞付与と係り受け解析 67
13.2 クロアチア語ミニ RoBERTa モデルの製作 67

14 コプト語編 **70**
14.1 コプト語 UD における品詞付与 70
14.2 コプト語 UD における係り受け解析 71
14.3 コプト語ミニ DeBERTa モデルの製作 71

1 概説編

本書では、Universal Dependencies (UD)[1]と呼ばれる多言語係り受けコーパス[2]を用いて、自然言語処理のうち、特に、品詞付与と係り受け解析に関して解説します。これらの解析手法の背後には、現実にはヤヤコシイ数式が横たわっているのですが、本書では、できる限り数式を使わずに、これらの解析手法を解説します。一方で、現代の自然言語処理は、その多くを BERT[3]・RoBERTa[4]・DeBERTa[5]などの事前学習モデルに拠っていることから、これら事前学習モデルについても概要を紹介します。また、各手法を読者が実際に試せるよう、Google Colaboratory[6]でのプログラミング例も、同時に示しています。プログラミング例は、理解しやすいよう、また「写経」しやすいよう、長くても 25 行程度に収めることを心がけました。python 等の基本知識が無くとも、とりあえずは動かしてみることが出来るでしょう。

本書の構成は、第 1 章が言語横断的な「概説編」となっており、第 2 章以降で各言語ごとの手法を解説しています。どの章から読み始めても大丈夫ですが、疑問が起こったら、必ず第 1 章に戻ってみることをオススメします。なお、それぞれの解析手法に関しては、できるだけ原論文や URL を示すようにしました。また、本書で書き切れなかった諸言語についても、deplacy[7]のサポートページ[8]に、Google Colaboratory での使用例を示しました (表 1)。応用を検討する際の一助となれば幸いです。

1.1 Universal Dependencies と CoNLL-U

UD は、書写言語における品詞・形態素属性・依存構造 (係り受け関係) を、言語に関わらず記述する手法です。句構造を考慮せずに係り受け関係を記述することで、言語横断性を高めていて、全ての文法構造を単語間のリンクで記述するのが特徴です。

依存構造解析それ自体は、Tesnière の構造的統語論[9]に源を発し、Мельчук の有向グラフ記述[10]によって、一応の完成を見た手法です。その最大の特長は、いわゆる動詞中心主義によって言語横断的な記述が可能だという点にあり、Мельчук 依存文法をコンピュータ向けに洗練した UD においても、言語に関わらない記述、という特長が前面に押し出されています。UD における文法構造記述は、句構造を考慮せず、全てを単語間のリンクとして表現します。これにより、言語横断的な文法構造記述を可能としているのです。

[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` において、カレル大学の LINDAT/CLARIN を中心とした 100 以上のグループが、世界中の言語に対して開発中。

[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`

[9]Lucien Tesnière: Éléments de Syntaxe Structurale, Paris: C. Klincksieck (1959).

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

表 1: deplacy に接続可能な各言語の係り受け解析ツール (2022 年 5 月 27 日現在)

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, NLP-Cube, spaCy, UDPipe 2, spacy-udpipe, Trankit, Camphr-Udify など
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-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, 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, spaCyPL, Camphr-Udify, Stanza, UDPipe 2, spacy-udpipe, Trankit, COMBO-pytorch 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

表 2: CoNLL-U の各フィールド

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。

表 3: UD 品詞タグ (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	

表 4: UD 係り受けタグ (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
```

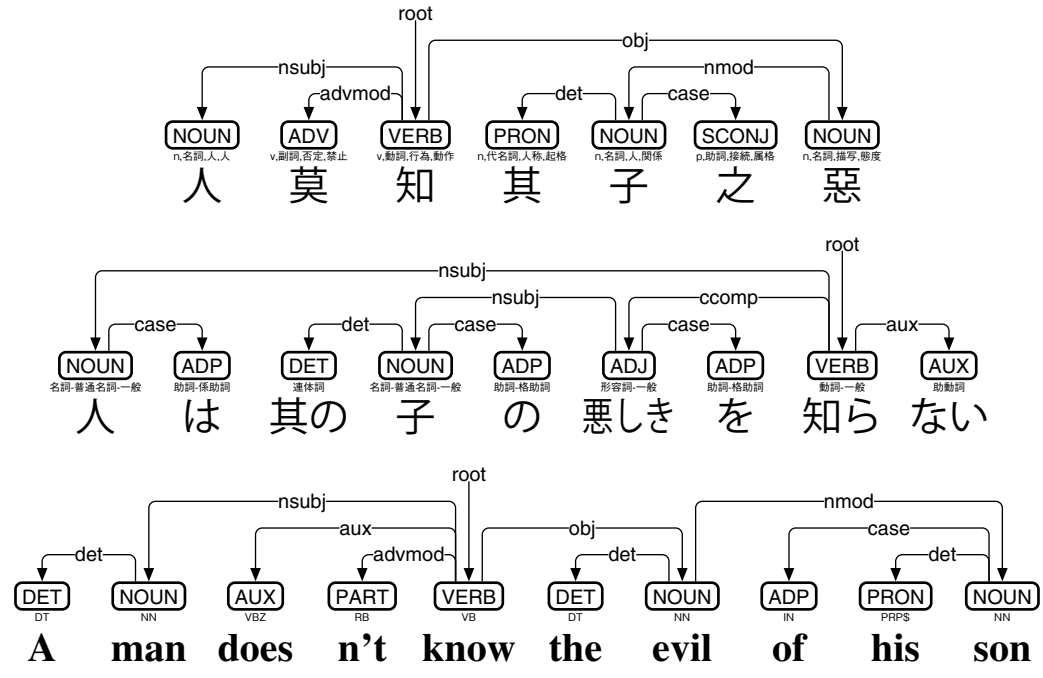


図 1: 漢日英 CoNLL-U と deplacy による可視化

UD係り受けコーパスの交換用フォーマットとして、CoNLL-Uと呼ばれるタブ区切りテキスト(文字コードはUTF-8[11])が規定されています。CoNLL-Uの各行は各単語に対応していて、表2に示す10個のタブ区切りフィールドで構成されます。ID・FORM・LEMMAは、単語そのものに関するフィールドです。UPOS・XPOS・FEATSは、単語の品詞と形態素属性に関するフィールドです。HEAD・DEPREL・DEPSは、単語の係り受けに関するフィールドです。

UDにおける係り受け関係は、単語間の有向グラフをHEADとDEPRELで記述します。HEADは、その単語に入る有向枝のリンク元IDを示していて、DEPRELは、その有向枝における係り受けタグです。ただし、HEADが0の場合、その枝に入るリンク元は存在しません。リンクの本数は単語の個数に等しく、各リンクのリンク先は、全て互いに異なっています。すなわち、各単語から出るリンクは複数の可能性がありますが、各単語に入るリンクは1つだけなのです。なお、リンクはループしません。

UDの係り受けリンクは、Мельчук依存文法の後裔にあたり、いわゆる動詞中心主義です。動詞をリンク元として、主語や目的語へとリンクします。修飾関係においては、被修飾語から修飾語へとリンクします。ただし、側置詞(前置詞や後置詞)を体言の修飾語だとみなす点[12]が、Мельчукとは異なっています。また、コピュラ文においては動詞中心主義を採らず、補語をリンク元として、主語や繋辞へとリンクします。

UDの言語横断性を示す実際例として、古典中国語(漢文)「人莫知其子之惡」と日本語文「人は其の子の悪しきを知らない」と英文「A man doesn't know the evil of his son」のCoNLL-Uを、図1で比較してみましょう。これらのCoNLL-Uは、各言語UDを横目に手作業で書いたもので、FEATS・DEPSは使用していません。可視化にはdeplacyを用いています。漢日英で語順は異なっているものの、nsubj (主語) と nmod(体言による連体修飾語)、obj (目的語) と ccomp(節目的語) のリンクが、それぞれに対応していると言えます。

また、コピュラ文に対するUDの例として、露文「Это ручка」と仏文「C'est un stylo」と日本語文「これはペンです」のCoNLL-Uを、図2で比較してみましょう。露仏日のいずれにおいても、補語から主語へnsubjが繋がれていて、それぞれに対応していると言えます。なお、仏日においては、繋辞がcopで繋がれています。

1.2 依存文法理論とUniversal Dependencies

UDの係り受け構造は、Мельчукの有向グラフ記述を理論的支柱としながらも、実践面ではReed-Kelloggの文法構造[13]を取り入れています。このあたりについて、簡単に紹介しておきましょう。

Reed-Kellogg文法構造図は、主語 | 述語 | 目的語という構造を基本に、英文を視覚化します。主文(main sentence)を太線で、それ以外の節(clause)を細線で視覚化した上に、修飾語を斜線に載せて示します。たとえば英文「Those who labour with their minds

[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, Genève (October 15, 1996).

[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
```

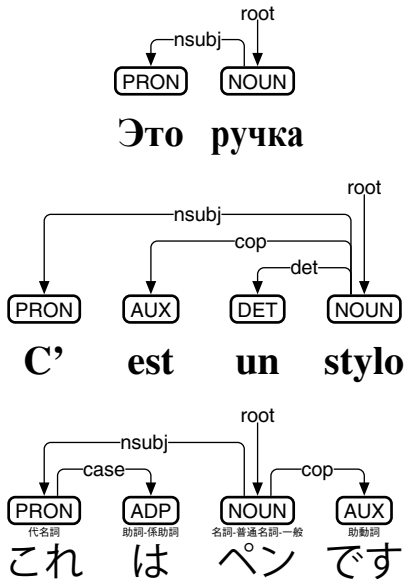


図 2: コピュラ文の CoNLL-U と deplacy による可視化

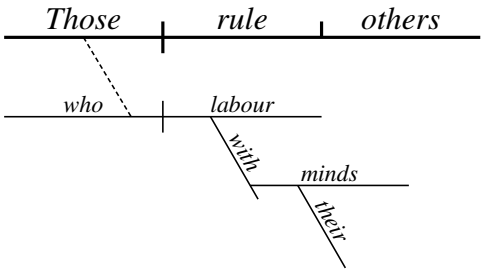


図 3: Reed-Kellogg 文法構造図

rule others」に対しては、「Those rule others」という主文の「Those」を、「who labour」という節が修飾していて、その「labour」を「with minds」が修飾し、「minds」を「their」が修飾していることから、図 3 のようになります。Reed-Kellogg 文法構造図は、英語に特化して設計されているので、必ずしも他の言語には適用できません。

Tesnière は、スラブ諸語の研究を通して、のちに依存文法と呼ばれる言語横断的な記述手法を発案しました。文に現れる語は互いに依存関係にある、というのが、依存文法の基本的な考え方です。語と語の間の依存関係においては、上位項 (régissant) が下位項 (subordonné) を支配し、下位項が上位項に依存します。たとえば英文「Those who labour with their minds rule others」であれば、rule が Those と others を支配し、Those と others が rule に依存します。ただし、転用がおこなわれている場合は、それらの語の間には依存関係は無く、並置する形で表現します。たとえば図 4 左では、名詞 minds が with によって E 転用 (連用修飾語に転用) されており、あるいは動詞 labour が who によって A 転用 (連体修飾語に転用) されています。

Tesnière の依存文法は、文における語の順序に関わらず、語の依存関係を記述するので、たとえば古典中国語にも応用可能です。すなわち「勞心者治人」であれば、「心」が「勞」に依存し、「勞」が「者」によって O 転用 (体言に転用) され「治」に依存します。これを図 4 右のように図示すれば、「Those who labour with their minds rule others」との間で、言語をまたいだ比較も可能となるのです。

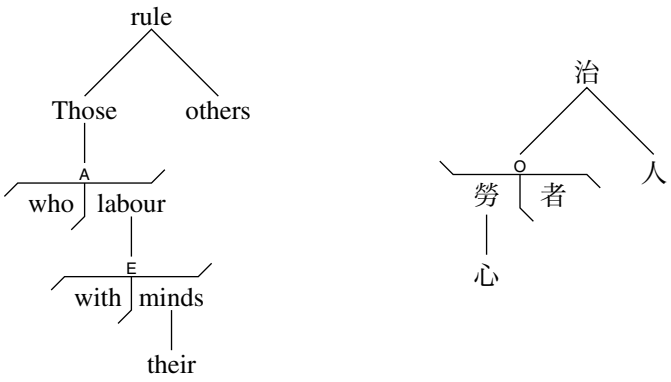


図 4: Tesnière 依存文法図

Tesnière の依存文法は、その後、Robinson[14]や Hudson[15]らにより、Chomsky 句構造文法[16]との融合が試みられました。一方Мельчук は、語の依存関係の有向グラフ記述によって依存文法の形式化をおこない、Chomsky 句構造文法と決別しました。

Мельчук の依存文法は、文中の語 X と Y に対し、依存関係 X→Y によって文を記述します。X→Y は、X は Y を支配する、あるいは、Y は X に依存する、と読みます。各矢印には、依存関係に関する適切なタグが付与されます。各語が依存する語は高々1 つであり、依存関係はループしません。依存関係は、あくまで語と語の間の記述で、それがたとえ節 (clause) や句 (phrase) に係るものであっても、割り切って語と語の関係として記述します。この「割り切り」の結果として、Мельчук の依存文法は、言語横断的な文法構造記述と

[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).

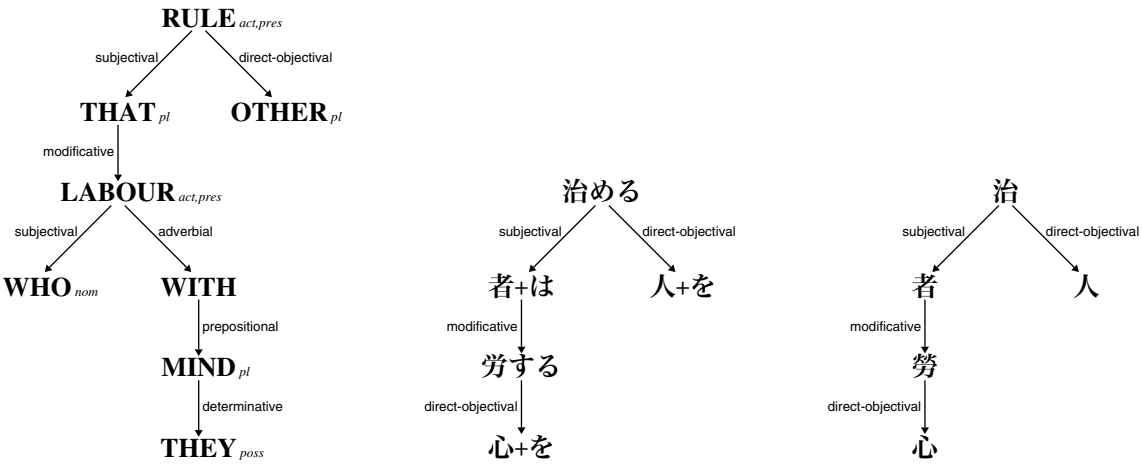


図 5: Мельчук の依存文法による文法構造記述

なっています。英文「Those who labour with their minds rule others」と、日本語文「心を労する者は人を治める」と、古典中国語 (漢文)「勞心者治人」に対して、Мельчук の依存文法による文法構造記述を、図 5 に示します。ただし、タグのうち subjectival と direct-objectival は、過去には predicative と 1st completive が用いられており、Мельчук 自身も、タグが何種類必要なのかは明らかにしていません[17]。

UD の係り受け構造は、Мельчук の依存文法におけるタグを、表 4 の 37 種類に限定する野心的な試み、として捉えることができます。図 5 での subjectival・direct-objectival・modificative・determinative が、図 6 では nsubj・obj・acl・det になった、と考えることもできます。ただ、UD の前身にあたる Stanford Typed Dependencies[18]が、初期には Bresnan 語彙機能文法[19]からスタートしており、その後に依存文法へとシフトしたことから、いくつか不思議な概念が紛れ込んでしまっています。その一つが、節 (clause) です。

節という概念を、Мельчук は依存文法から削除したのですが、これを UD は再導入してしまいました。表 4 の nsubj と csubj は、いずれも主語を表す UD 係り受けタグであり、リンク先が節なら csubj を、さもなくば nsubj を使います。obj と ccomp についても同様、advmod と advcl についても同様、amod と acl についても同様です。case と mark については、リンク元が節なら mark を、さもなくば case を使います[20]。

英語における節という概念は、一定のコンセンサスが得られていると考えられます。しかし、他の言語において、節という概念は、必ずしも自明ではありません。UD は言語横断的ではあるものの、かなり英語寄りである、という点は意識しておくべきでしょう。

[17]Alain Polguère, Igor A. Mel'čuk: Dependency in Linguistic Description, Amsterdam: John Benjamins (2009).

[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
```

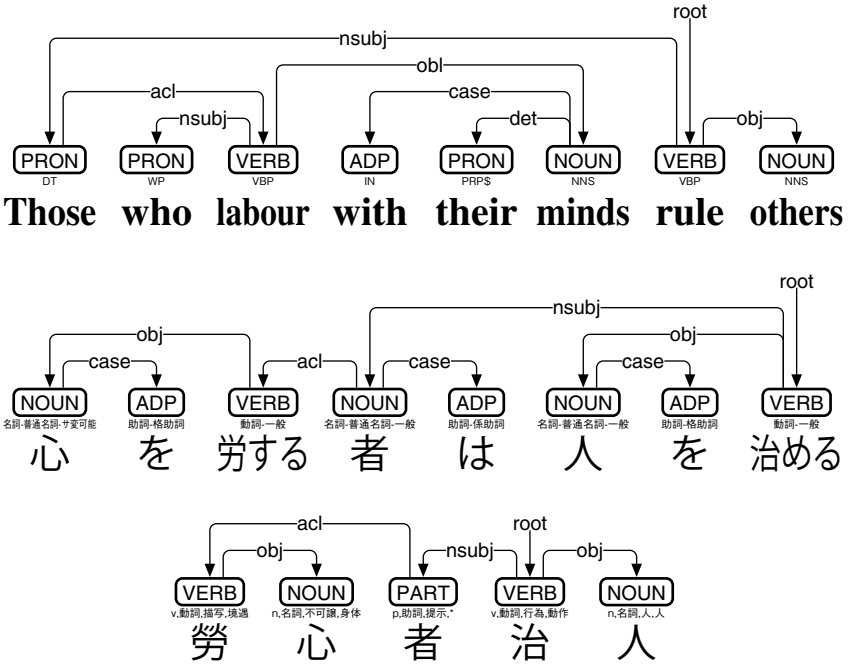


図 6: UD と CoNLL-U による係り受け構造記述

1.3 BERT/RoBERTa/DeBERTa モデルと穴埋めゲーム

Google AI Language が発表した BERT は、大量の文章を機械に丸暗記させる際にどのような手法が有効なのかを、端的に示した事前学習モデルです。単純に文章を丸暗記するのではなく、文章中に現れる文と文の繋がり、あるいは、文中に現れるトークン (単語もしくは単語より短い文字列) とトークンの繋がりを、効果的に学習できるような工夫が詰め込まれています。

トークンとトークンの繋がりを学習するために用いられるのが、穴埋めゲームです。「This [MASK] a pen.」の [MASK] に対し、「This is a pen.」という答を得るような穴埋めゲーム (図 7) を考えてみましょう。bert-base-cased[21]においては、各トークンの入出力は文字コード (UTF-8) でおこなわれますが、内部的には 768 次元のベクトル (数値の列) で表現されます。入力側においては、各入力トークンに対応するベクトルをそのまま用いており、出力側においては、各出力トークンに近い[22]ベクトルになるように学習をおこないます。「This [MASK] a pen.」の例で言えば、「This」「a」「pen」「.」は入力ベクトルをほぼ素通ししつつ、「is」に近い出力ベクトルが得られるよう、内部の記憶素子をいじるわけです。このような穴埋めゲームを大量の文に対しておこなうことで、トークンとトークンの繋がりを学習[23]できるというのが、BERT の工夫の一つ[24]です。

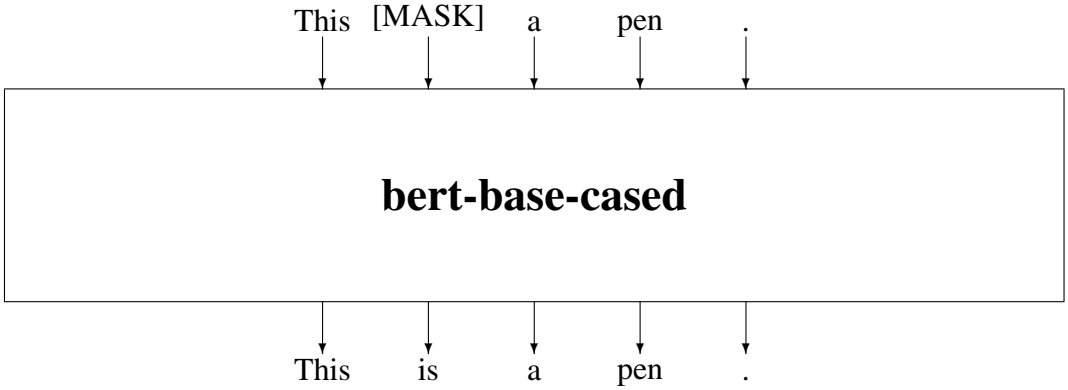


図 7: bert-base-cased における穴埋めゲーム

Google Colaboratory 上の Transformers[25]を使って、bert-base-cased が英文をどのぐらい学習できているのか、試してみましょう (図 8)。「This [MASK] a pen.」を穴埋めさせてみると、「was」が 48.2%、「is」が 40.2%となっており、出力されているベクトルが「was」と「is」の中間で、やや「was」に近いことがわかります。

[21] `https://huggingface.co/bert-base-cased`

[22] ここでいう「近い」は、コサイン類似度 (2 つのベクトルの内積を、ベクトルのユークリッド・ノルムで割った値) が 1 に近い、という意味です。

[23] bert-base-cased の内部では、穴埋め用の [MASK] に加えて、文頭を表す [CLS]、文の切れ目を表す [SEP]、未知語を表す [UNK]、空きトークンを表す [PAD] なども用いられます。

[24] BERT のもう一つの工夫は、文と文の繋がりを学習する隣接文ゲームですが、本書では扱いません。

[25] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi 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.

```
!pip install transformers
from transformers import AutoTokenizer,AutoModelForMaskedLM,FillMaskPipeline
brt="bert-base-cased"
tkz=AutoTokenizer.from_pretrained(brt)
mdl=AutoModelForMaskedLM.from_pretrained(brt)
fmp=FillMaskPipeline(tokenizer=tkz,model=mdl)
prd=fmp("This [MASK] a pen.")
print("\n".join("{:8} {:.03f}".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

図 8: 「This [MASK] a pen.」に対する穴埋めゲームプログラムと結果

BERT を多言語拡張する際に問題となるのは、モデルの入出力に文という単位を仮定している点です。言語によっては (たとえば古典中国語においては)、文という単位は必ずしも明確ではありません。あるいは SNS での「つぶやき」など、文境界がハッキリしない例は多くあります。これに対し RoBERTa や DeBERTa は、文という単位を仮定せず、トークンの並びを用いて、穴埋めゲームのみに特化した学習をおこないます。これにより、文という単位に捉われることなく、大量の文章を丸暗記できるのです。

roberta-classical-chinese-base-char[26]は、古典中国語における漢字をトークンとみなし、漢字単位での穴埋めゲーム (図 9) を学習したモデルです。各漢字 (約 26000 字種) の入出力は UTF-8 ですが、内部的には 768 次元のベクトルで表現します。Google Colaboratory 上の Transformers で「勞 [MASK] 者治人」を穴埋めするプログラムと、その結果を図 10 に示します。「心」が 19.9%となっていて、「人」の 32.7%に負けているのがわかります。

BERT・RoBERTa・DeBERTa などの事前学習モデルは、出力部を付け替えることで、様々な用途に応用可能です。出力ベクトルを品詞情報だとみなせば、品詞付与に使えま

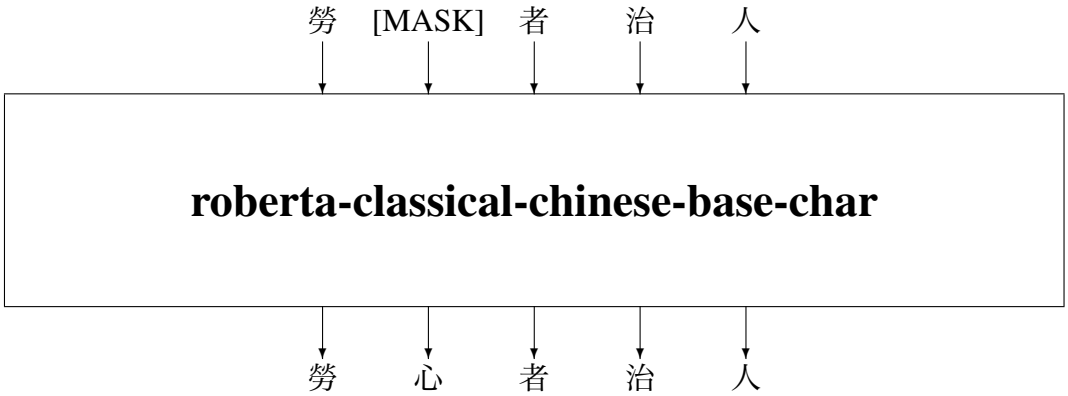


図 9: roberta-classical-chinese-base-char における穴埋めゲーム

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

```
!pip install transformers
from transformers import AutoTokenizer,AutoModelForMaskedLM,FillMaskPipeline
brt="KoichiYasuoka/roberta-classical-chinese-base-char"
tkz=AutoTokenizer.from_pretrained(brt)
mdl=AutoModelForMaskedLM.from_pretrained(brt)
fmp=FillMaskPipeline(tokenizer=tkz,model=mdl)
prd=fmp("勞[MASK]者治人")
print("\n".join("{:8} {:.03f}".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

図 10:「勞 [MASK] 者治人」に対する穴埋めゲームプログラムと結果

す。係り受けタグだとみなせば、係り受け解析に使えます。UD においては、系列ラベリングや係り受け解析に、事前学習モデルを応用することができます。

1.4 Universal Dependencies における係り受け解析

係り受け解析は、UD による言語処理の「キモ」であり、多くの解析アルゴリズムが乱立しています。これらの係り受け解析アルゴリズムを、状態遷移型アルゴリズムと隣接行列型アルゴリズムに分けて、概説します。

arc-swap[27]に代表される状態遷移型アルゴリズムは、単語列の先頭から末尾に向かって「垣根」(stack-buffer boundary) を移動していく、というイメージで処理をおこないます。「垣根」がおこなう遷移は、以下の 6 種類に定式化されます。

- **Shift** 「垣根」を右に 1 単語分、移動する。
- **Reduce** 「垣根」のすぐ左の単語を除去して、解析結果へ移す。
- **Left-Arc** 「垣根」のすぐ右の単語から、すぐ左の単語へリンクを繋ぐ。
- **Right-Arc** 「垣根」のすぐ左の単語から、すぐ右の単語へリンクを繋ぐ。
- **Unshift** 「垣根」を左に 1 単語分、移動する。
- **Swap** 「垣根」のすぐ右の単語と、すぐ左の単語を入れ替える。

単語が全て **Reduce** されて、「垣根」がポツンと取り残された時点で、状態遷移型アルゴリズムは終了です。解析例を図 11 に示します。解析結果において、入るリンクがない単語に対しては、通常は root を割り当てます。なお、6 種類のうち **Unshift・Swap** を実装しない場合は、リンクに交差がある UD を扱えません。また、現実の実装においては、**Left-Arc** の直後には **Reduce** を、**Right-Arc** の直後には **Shift** と **Reduce** を、それぞれまとめて遷移させる手法が主流です。

[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.

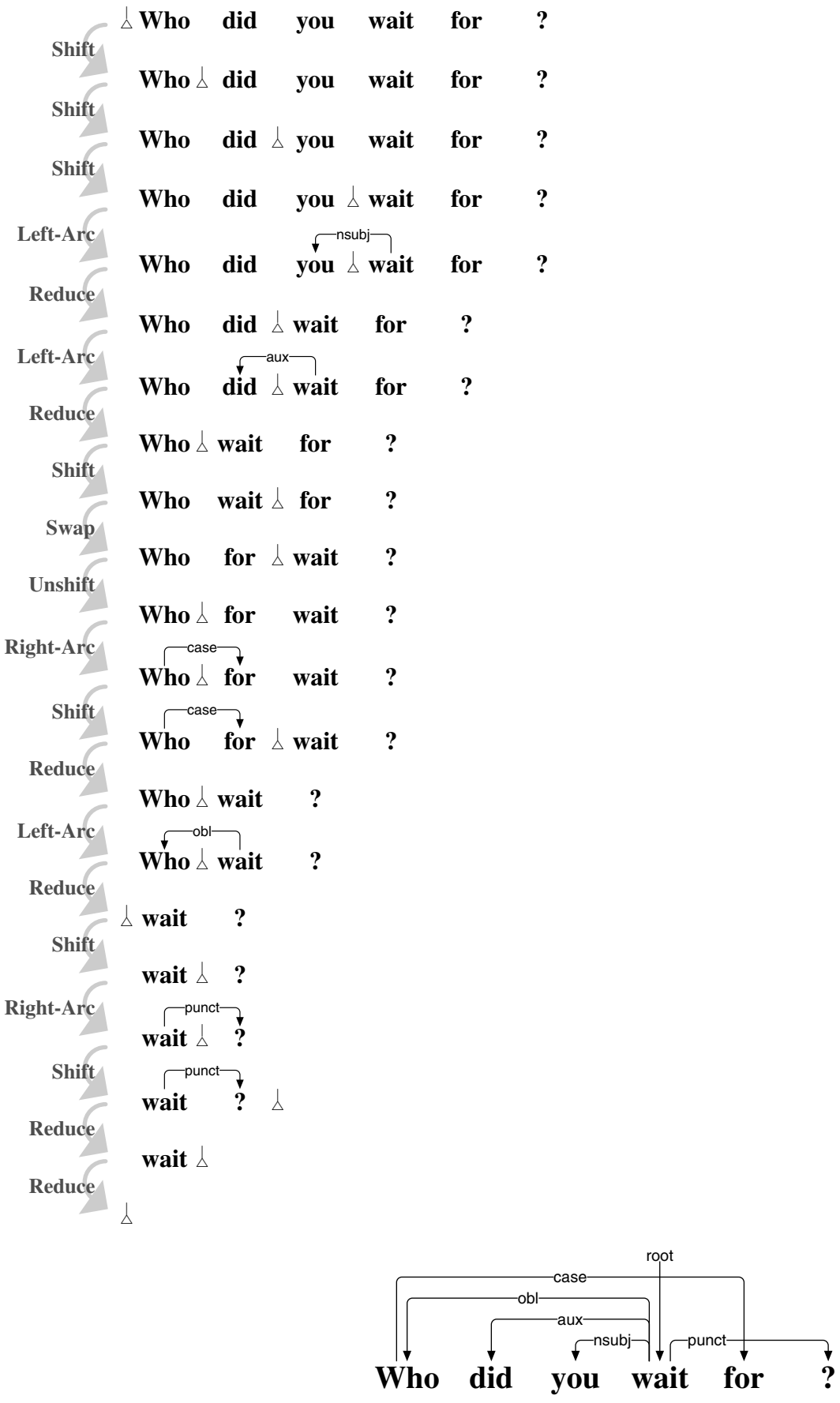


図 11: 状態遷移型アルゴリズムによる「Who did you wait for?」の係り受け解析

Biaffine[28]に代表される隣接行列型アルゴリズムは、UD 依存構造を有向グラフとみなし、その隣接行列を求めます。たとえば、図 11 右下「Who did you wait for?」の有向グラフであれば

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

という 6×6 の隣接行列を求めます (図 12)。具体的には、隣接行列の各列ごとに要素を 1 つ選びつつ、対角成分上の root から 1 つを選ぶ、というのを確率的におこないます。ただ、ループが発生してしまう場合があるので、適宜 Chu-Liu-Edmonds 等[29]を使ってループを解消します。

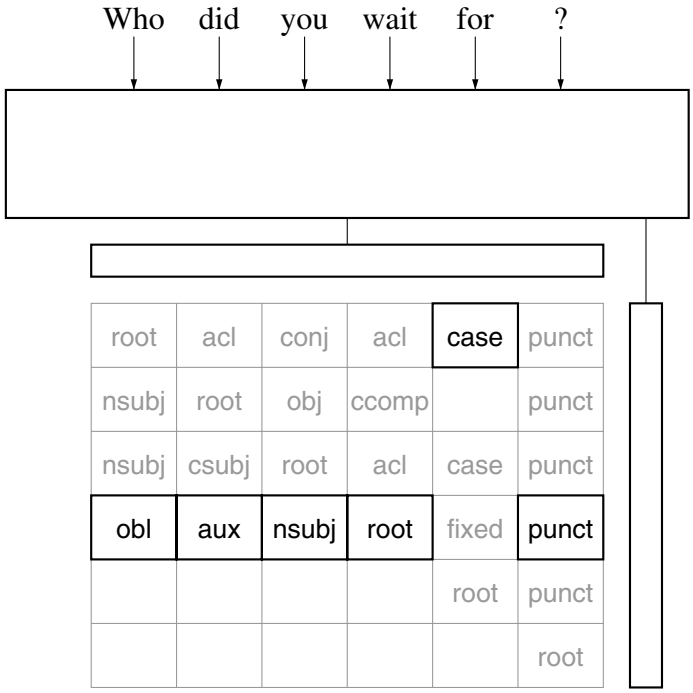


図 12: 隣接行列型アルゴリズムによる「Who did you wait for?」の係り受け解析

状態遷移型アルゴリズムは、BERT・RoBERTa・DeBERTa などの事前学習モデルを用いることで精度向上が期待できる[30]のですが、これは実装がかなり難しいのです。一方、隣接行列型アルゴリズムは、事前学習モデルとの相性が良いと考えられます。ただし、隣接行列型アルゴリズムに事前学習モデルを用いると、計算機資源 (メモリや GPU) を大量に必要とすることから、事前学習モデルを使用しない実装もありえます。

[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.

[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.3278-3289.

2　古典中国語(漢文)編

2.1　古典中国語UDにおける品詞付与と文切り

古典中国語(漢文)は、単語と単語の間に区切りがなく、文と文の間にも区切りがありません。これが、白文と呼ばれる古典中国語の書写形態であり、傍目には、漢字が連続的に並んでいるだけです。

古典中国語UDにおける品詞付与は、端的に言えば、白文の各漢字にUPOSを付与する作業です。ただし、古典中国語における単語(漢語)は2文字以上のものもあるので、その場合は、複数の漢字をまとめつつUPOSを付与します。Transformersの系列ラベリングを用いるなら、1文字の漢語に対してはUPOSをラベルとして付与し、2文字以上の漢語に対しては、1文字目に「B-」を付けたUPOSを、2文字目以降に「I-」を付けたUPOSを、それぞれ付与するやり方が考えられます。これにより原理的には、複数の漢字を漢語にまとめつつ、UPOSを付与することができます。

```
!pip install deplacy transformers
class SeqL(object):
  def __init__(self,bert,aggr="none"):
    from transformers import pipeline
    self.pipeline=pipeline(task="ner",model=bert,aggregation_strategy=aggr)
    self.entity="entity" if aggr=="none" else "entity_group"
  def __call__(self,text):
    w=[(t["start"],t["end"],t[self.entity]) for t in self.pipeline(text)]
    u="# text = "+text.replace("\n"," ")+"\n"
    for i,(s,e,p) in enumerate(w,1):
      m="_" if i<len(w) and w[i][0]<e else "SpaceAfter=No"
      u+="\t".join([str(i),text[s:e],"_",p]+["_"]*5+[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)
```

図 13: roberta-classical-chinese-base-upos によるUPOS付与

Google Colaboratory 上の Transformers を用いて、roberta-classical-chinese-base-upos[31] で「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」に UPOS を付与するプログラムと、その結果を図 13 に示します。「君子」が 2 文字の漢語であり、「君」に B-NOUN が、「子」に I-NOUN が付与されています。「君子」以外は漢字 1 字が漢語 1 語と判定されていて、それぞれに UPOS が付与されています。

Transformers の系列ラベリングは、他の用途にも適用できます。roberta-classical-chinese-base-sentence-segmentation[32]は、古典中国語の文切りに Transformers の系列ラベリングを用いています。具体的には、文頭の漢字に対するラベルを B、文末の漢字に対するラベルを E とし、B・M・E3・E2・E を用いて各文を表します。ただし、1 文字のみで構成される文は、ラベルを S とします。2 文字で構成される文のラベルは、B・E とします。3 文字で構成される文のラベルは、B・E2・E とします。4 文字で構成される文のラベルは、B・E3・E2・E とします。

```
!pip install deplacy transformers
class SeqL(object):
  def __init__(self,bert,aggr="none"):
    from transformers import pipeline
    self.pipeline=pipeline(task="ner",model=bert,aggregation_strategy=aggr)
    self.entity="entity" if aggr=="none" else "entity_group"
  def __call__(self,text):
    w=[(t["start"],t["end"],t[self.entity]) for t in self.pipeline(text)]
    u="# text = "+text.replace("\n"," ")+"\n"
    for i,(s,e,p) in enumerate(w,1):
      m="_" if i<len(w) and w[i][0]<e else "SpaceAfter=No"
      u+="\t".join([str(i),text[s:e],"_",p]+["_"]*5+[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)
```

図 14: roberta-classical-chinese-base-sentence-segmentation による古典中国語の文切り

[31]https://huggingface.co/KoichiYasuoka/roberta-classical-chinese-base-upos

[32]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.

Google Colaboratory 上の Transformers を用いて、roberta-classical-chinese-base-sentence-segmentation で「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」を文切りするプログラムと、その結果を図 14 に示します。「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」の 7 つの文に切られているのがわかります。

2.2 古典中国語 UD における係り受け解析

SuPar-Kanbun[33]は、隣接行列型の係り受け解析アルゴリズムを、古典中国語向けに実装しています。UPOS 品詞付与と文切りも同時に実装しており、roberta-classical-chinese-base-char[26]を元にした「オールインワン設計」です。Google Colaboratory 上の SuPar-Kanbun を用いて、「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」を品詞付与・文切り・係り受け解析するプログラムと、その結果を図 15 に示します。各文の解析結果を、順に見ていくことにしましょう。

「子曰」は、動詞「曰」の主語が「子」という文であり、主語を表す nsubj で繋がれています。「學而時習之」は、並列接続詞「而」によって「學」と「習」が並置されていて、「學」と「習」が conj で、「習」と「而」が cc で繋がれています。「之」は「習」の目的語で、obj で繋がれています。「時」は「習」の斜格補語で、本来 obl で繋ぐところですが、古典中国語 UD では、obl を拡張した obl:tmod (時に関する斜格補語) を導入しています。「不亦説乎」は、「不」と「亦」がいずれも「説」を連用修飾していて、いずれも advmod で繋がれています。「乎」は文全体を修飾する句末の終助詞 (sentence particle) ですが、これを表すような UD 係り受けタグが表 4 に無いため、discourse:sp を導入[34]しています。「有朋自遠方來」は、「有朋」と「自遠方來」の 2 句から構成されていて、これらが parataxis で繋がれています。「朋」は「有」の目的語で、obj で繋がれています。「遠方」は「來」の斜格補語で、本来 obl で繋ぐところですが、古典中国語 UD では、obl を拡張した obl:lmod (場所に関する斜格補語) を導入しています。ただし「遠方」の内部では、動詞「遠」(古典中国語 UD には形容詞という分類は無く、動詞として扱います[35]) が名詞「方」を連体修飾していて、amod で繋がれています。なお、前置詞「自」も「方」を修飾しているとみなし、格表示を表す case で繋がれています。「不亦樂乎」は「不亦説乎」と同様の依存構造です。「人不知而不慍」は、「而」によって「知」と「慍」が並置されていて、「知」と「慍」が conj で、「慍」と「而」が cc で繋がれています。「人」は「知」と「慍」の両方の主語だと考えられるのですが、近い方の「知」に nsubj で繋がれています。また、「知」と「慍」には、それぞれ否定の「不」が連用修飾していて、いずれも advmod で繋がれています。「不亦君子乎」は、「不亦説乎」や「不亦樂乎」と類似した依存構造ですが、「君子」が名詞でありコピュラ文です (ただし主語も繋辞もありません)。

[33]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.

[34]Herman Leung, Rafaël 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.

[35]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.

```
!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)
```

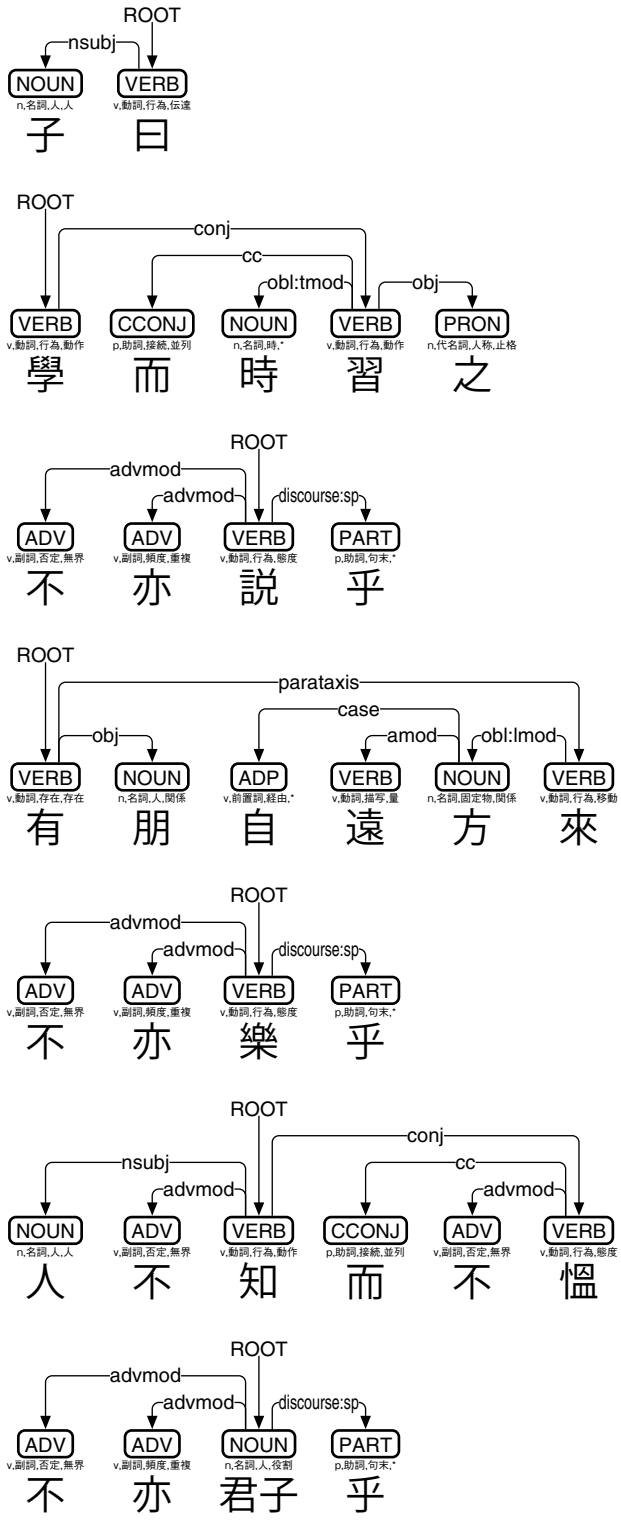


図 15: SuPar-Kanbun による古典中国語の係り受け解析

このような形でSuPar-Kanbunは、漢字の列である白文に品詞付与をおこない、さらに漢語の列に係り受け解析をおこなうことで、漢文の依存構造を抽出します。なお、古典中国語UDの係り受けタグには、表4と図15に加え、受動文の主語を表すnsubj:pass、重畳を表すcompound:redup、動詞並列を表すflat:vv、古典中国語以外を表すflat:foreignも用いられます[36]。

2.3 古典中国語ミニRoBERTaモデルの製作

Google Colaboratory上のesupar[37]で、古典中国語ミニRoBERTaモデルを製作してみましょう。手順を図16〜20に示します。

最初に、訓練のための白文を準備しましょう(図16)。train.txtの白文の材料として、ここではUD_Classical_Chinese-Kyoto[38]を元にしていますが、好みに応じ、他の白文を増量するのもいいでしょう。なお、UD_Classical_Chinese-Kyotoの各CoNLL-Uファイルは、品詞付与・係り受け解析の学習にも用います。

続けて、入力文字列を単文字に分解するためのトークナイザを、my-dir/tokenizer-lzhに作成しましょう。図17では、BertTokenizerFastというTransformersのトークナイザを

```
!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
```

図16: 古典中国語ミニモデル製作のための準備

```
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")
```

図17: 古典中国語向け単文字トークナイザの作成

[36]https://universaldependencies.org/lzh/#syntax

[37]https://github.com/KoichiYasuoka/esupar

[38]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,RobertaConfig,RobertaForMaskedLM,
  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=RobertaConfig(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=RobertaForMaskedLM(cfg),train_dataset=ReadLineDS("train.txt",tkz))
trn.train()
trn.save_model("my-dir/roberta-lzh")
tkz.save_pretrained("my-dir/roberta-lzh")
```

図 18: 古典中国語ミニ RoBERTa モデルの作成

```
import torch
from transformers import AutoTokenizer,AutoModelForMaskedLM
from esupar.tradify import tradify
tkz=AutoTokenizer.from_pretrained("my-dir/roberta-lzh")
mdl=AutoModelForMaskedLM.from_pretrained("my-dir/roberta-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/roberta-lzh-ext")
tkz.save_pretrained("my-dir/roberta-lzh-ext")
```

図 19: 古典中国語ミニ RoBERTa モデルの拡張 (異体字追加)

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

図 20: 古典中国語向け品詞付与・係り受け解析ミニモデルの製作

使って、単文字トークナイザを作成しています。

次に train.txt の白文から、RoBERTa モデルを my-dir/roberta-lzh に作成しましょう。図 18 では、入出力幅 128 トークン・入出力ベクトル 256 次元・深さ 12 層・アテンションヘッド 4 個・中間ベクトル 768 次元という、やや小さめのモデルにしており、CPU なら 1 時間程度、GPU なら 10 分程度で作成できます。ただし、この RoBERTa モデルは繁體字で作られており、日本の常用漢字や中国の簡化字は含まれていません。そこで、これらの異体字に対しベクトルのコピーをおこない、新たなモデルを my-dir/roberta-lzh-ext に保存しましょう (図 19)。

最後に、my-dir/roberta-lzh-ext と UD_Classical_Chinese-Kyoto から、品詞付与・係り受け解析モデル my-dir/roberta-lzh-upos を製作しましょう (図 20)。ただ、隣接行列型アルゴリズムの学習は収束が遅く、GPU でも 3 時間程度かかってしまいます。

うまく my-dir/roberta-lzh-upos が製作できたかどうか、esupar でテストしてみましょう (図 21)。繁體字のみならず、日本の常用漢字や中国の簡化字で書かれた漢文も、品詞付与・係り受け解析できるはずなので、ぜひ試してみて下さい。

ちなみに図 22 は、my-dir/roberta-lzh-ext と UD_Classical_Chinese-Kyoto から、ミニ文切りモデル my-dir/roberta-lzh-seg を作成 (ファインチューニング) するプログラムです。ただし、ミニモデルでは文切りの精度が上がらないので、もっと大きなモデルを作成した方がいいでしょう。

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

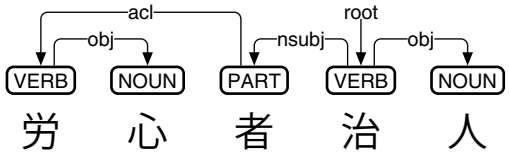


図 21: 古典中国語向け品詞付与・係り受け解析ミニモデルのテスト

```
from transformers import (AutoTokenizer,AutoModelForTokenClassification,
  AutoConfig,DataCollatorForTokenClassification,TrainingArguments,Trainer)
brt="my-dir/roberta-lzh-ext"
tkz=AutoTokenizer.from_pretrained(brt)
lid={"B":0,"E":1,"E2":2,"E3":3,"M":4,"S":5}
cfg=AutoConfig.from_pretrained(brt,num_labels=len(lid),label2id=lid,
  id2label={i:l for l,i in lid.items()})
arg=TrainingArguments(num_train_epochs=3,per_device_train_batch_size=64,
  output_dir="/tmp",overwrite_output_dir=True,save_total_limit=2)
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(args=arg,data_collator=DataCollatorForTokenClassification(tkz),
  model=AutoModelForTokenClassification.from_pretrained(brt,config=cfg),
  train_dataset=dat)
trn.train()
trn.save_model("my-dir/roberta-lzh-seg")
tkz.save_pretrained("my-dir/roberta-lzh-seg")
```

図 22: 古典中国語文切りミニモデルの作成

3　日本語編

書写言語としての日本語は、単語と単語の間に区切りがありません。そもそも、単語の長さに決まりがないので、様々な単語長が使われている[39]のです (図 23)。日本語 UD では、これらの単語長のうち、国語研短単位と国語研長単位がサポートされています。その一方で、日本語 BERT/RoBERTa モデルのトークン長は、国語研短単位 (あるいはその細分化) や文字単位のものが多いのですが、図 23 以外のトークン長を採用しているモデルもあり、かなり混沌としています。

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 ||

図 23: 国語研短単位・国語研長単位・文節

3.1　国語研短単位にもとづく品詞付与と係り受け解析

SuPar-UniDic[40]は、青空文庫 (+Wikipedia) BERT モデル[41]をベースに、隣接行列型の係り受け解析アルゴリズムを、国語研短単位向けに実装しています。形態素解析 (単語切り・XPOS 品詞付与) 部には、MeCab[42]と UniDic[43]を流用しています。Google Colaboratory 上の SuPar-UniDic を用いて、「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」を品詞付与・係り受け解析するプログラムと、その結果を図 24 に示します。解析結果を見ていくことにしましょう。

国語研短単位 UD では、「全学年」「小学校」「教科書」が複数の単語に分かれていて、compound で繋がれているのが特徴的です。「小学校の国語の教科書」は、名詞が名詞を修飾していて nmod で繋がれていると同時に、「の」が case で繋がれています。「大量の挿し絵」も同様です。「全学年」は「わたって」のニ格であり、obl で繋がれていると同時に、「に」が case で繋がれています。「挿し絵」は「用いられて」のガ格であり、nsubj で繋がれていると同時に、「が」が case で繋がれています。「全学年にわたって」は「挿

[39]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.
[40]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).
[41]https://github.com/akirakubo/bert-japanese-aozora
[42]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.
[43]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: コ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)
```

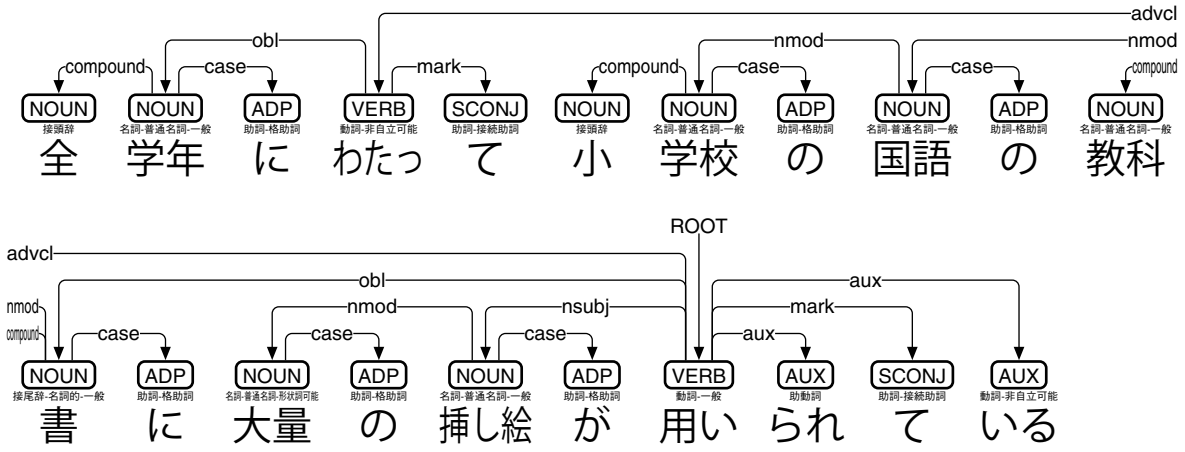


図 24: SuPar-UniDic による日本語係り受け解析 (国語研短単位)

し絵が用いられて」の連用修飾節であり、advcl で繋がれています。「いる」は、UniDic 品詞 (XPOS) が「動詞-非自立可能」なのですが、ここでは非自立だとみなして UPOS を AUX にすると同時に、aux で繋がれています。

国語研短単位 UD では、XPOS と UPOS が微妙に乖離しています。XPOS は UniDic 品詞を流用しているのですが、UPOS の品詞付与ルールとはズレているためです。極端な例として「難儀な難儀は難儀する」(図 25) を見てみましょう。「難儀」の UniDic 品詞は「名詞-普通名詞-サ変形状詞可能」ですが、「難儀な」は形状詞 (形容動詞) として扱うべきであり、UPOS は ADJ が妥当です。「難儀する」はサ変動詞として扱うべきで、UPOS は VERB が妥当です。ただし「難儀な」も「難儀する」も、国語研短単位では 2 つの単語に分かれてしまうので、「難儀」の UPOS に、それぞれ ADJ と VERB を付与しています。

```
!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)
```

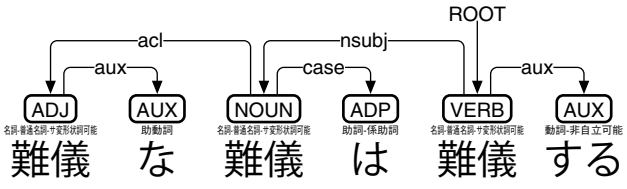


図 25: 国語研短単位 UD における XPOS (UniDic 品詞) と UPOS の関係

3.2 国語研短単位ミニ RoBERTa モデルの製作

Google Colaboratory 上の esupar[37]で、国語研短単位ミニ RoBERTa モデルを製作してみましょう。手順を図 26〜29 に示します。

最初に、訓練のための文章を準備しましょう (図 26)。train.txt の文章の材料として、ここでは UD_Japanese-GSD[44]と Wikitext-JA[45]を元にしていますが、好みに応じ、他の日本語文を増量するのもいいでしょう。なお、UD_Japanese-GSD の各 CoNLL-U ファイルは、品詞付与・係り受け解析の学習にも用います。

```
!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)
```

図 26: 国語研短単位ミニモデル製作のための準備

続けて、入力文を国語研短単位に分解するためのトークナイザを、my-dir/tokenizer-suw に作成しましょう。図 27 では、fugashi[46]と unidic-lite[47]で train.txt を国語研短単位に区切り、それを BertJapaneseTokenizer という Transformers のトークナイザに組み上げて、国語研短単位トークナイザを作成しています。語彙数 (vocab_size) を 8000 に、異なり字数 (limit_alphabet) を 3000 に絞っていますが、もう少し大きい方がいいかもしれません。

```
!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")
```

図 27: 国語研短単位トークナイザの作成

[44]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.

[45]http://www.lsta.media.kyoto-u.ac.jp/resource/data/wikitext-ja/

[46]https://github.com/polm/fugashi

[47]https://github.com/polm/unidic-lite

```
from transformers import (AutoTokenizer,RobertaConfig,RobertaForMaskedLM,
  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=RobertaConfig(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=RobertaForMaskedLM(cfg),train_dataset=ReadLineDS("train.txt",tkz))
trn.train()
trn.save_model("my-dir/roberta-suw")
tkz.save_pretrained("my-dir/roberta-suw")
```

図 28: 国語研短単位ミニ RoBERTa モデルの作成

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

図 29: 国語研短単位向け品詞付与・係り受け解析ミニモデルの製作

```
!pip install esupar fugashi unidic-lite pytokenizations
import esupar
nlp=esupar.load("my-dir/roberta-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)
```

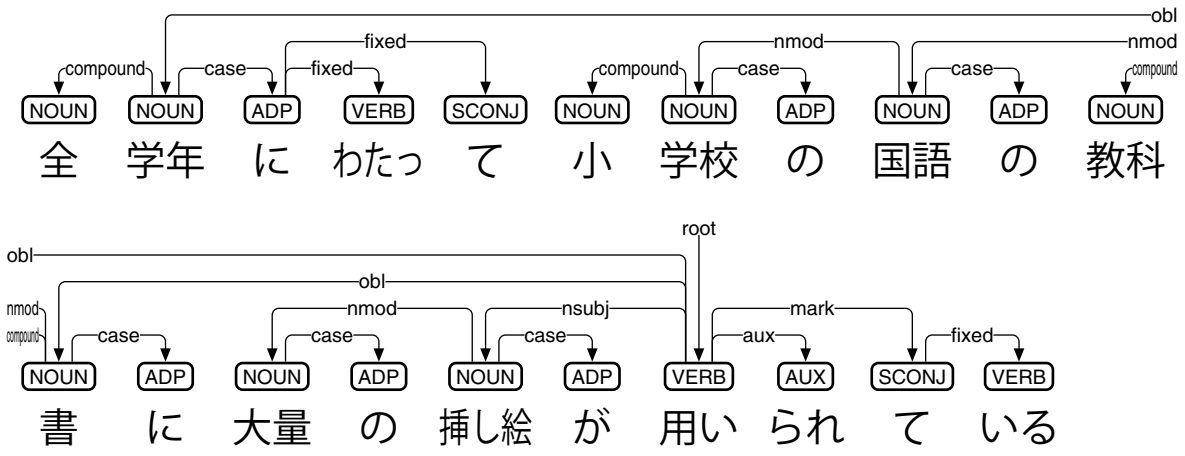


図 30: 国語研短単位向け品詞付与・係り受け解析ミニモデルのテスト

次に train.txt の文章から、RoBERTa モデルを my-dir/roberta-suw に作成しましょう。図 28 では、入出力幅 128 トークン・入出力ベクトル 256 次元・深さ 12 層・アテンションヘッド 4 個・中間ベクトル 768 次元という、やや小さめのモデルにしていますが、GPU でも 3 時間程度かかってしまいます。

最後に、my-dir/roberta-suw と UD_Japanese-GSD から、品詞付与・係り受け解析モデル my-dir/roberta-suw-upos を製作しましょう (図 29)。ただ、隣接行列型アルゴリズムの学習は収束が遅く、GPU でも 2 時間程度かかってしまいます。

うまく my-dir/roberta-suw-upos が製作できたかどうか、esupar でテストしてみましょう。なお、esupar で BertJapaneseTokenizer を用いる場合、fugashi と unidic-lite に加えて、pytokenizations[48] も必要です。図 30 の例では、「にわたって」「ている」のリンクに fixed が現れていて、図 24 とは異なる結果になっているようです。

3.3 国語研長単位にもとづく品詞付与と係り受け解析

国語研長単位トークナイザ Japanese-LUW-Tokenizer[49] と、青空文庫 RoBERTa モデル roberta-base-japanese-luw-upos[50] を元に、国語研長単位にもとづく品詞付与を考えてみま

```
!pip install deplacy transformers
class SeqL(object):
  def __init__(self,bert,aggr="none"):
    from transformers import pipeline
    self.pipeline=pipeline(task="ner",model=bert,aggregation_strategy=aggr)
    self.entity="entity" if aggr=="none" else "entity_group"
  def __call__(self,text):
    w=[(t["start"],t["end"],t[self.entity]) for t in self.pipeline(text)]
    u="# text = "+text.replace("\n"," ")+"\n"
    for i,(s,e,p) in enumerate(w,1):
      m="_" if i<len(w) and w[i][0]<e else "SpaceAfter=No"
      u+="\t".join([str(i),text[s:e],"_",p]+["_"]*5+[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)
```

図 31: roberta-base-japanese-luw-upos による UPOS 付与

[48] https://github.com/explosion/tokenizations
[49] https://github.com/KoichiYasuoka/Japanese-LUW-Tokenizer
[50] https://huggingface.co/KoichiYasuoka/roberta-base-japanese-luw-upos

```
!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)
```

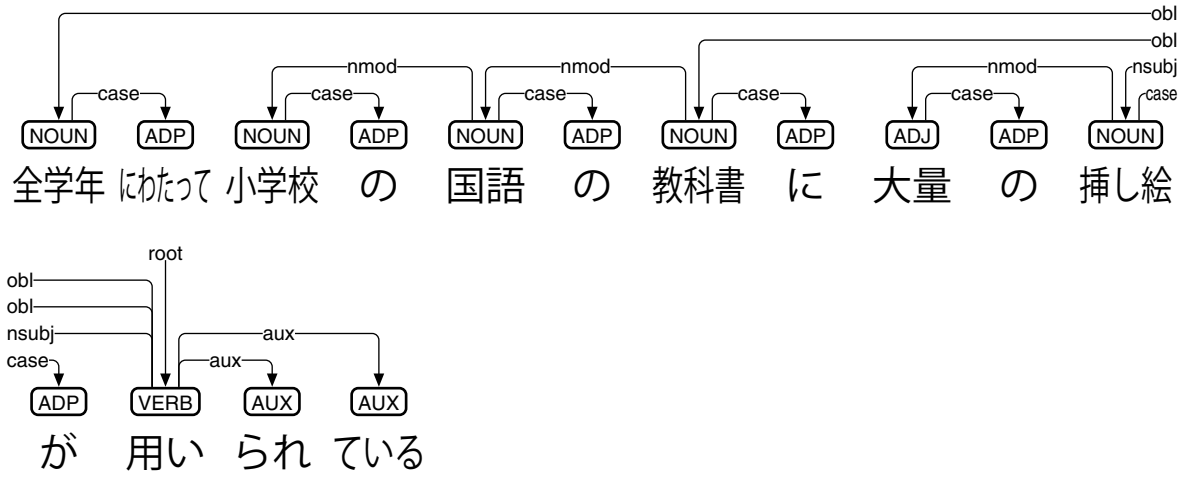


図 32: roberta-base-japanese-luw-upos による日本語係り受け解析 (国語研長単位)

しょう。Google Colaboratory 上の Transformers を用いて、「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」に UPOS を付与するプログラムと、その結果を図 31 に示します。各トークンに系列ラベリングで UPOS を付与しているのですが、Japanese-LUW-Tokenizer が完璧ではなく、やや短めに「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」とトークナイズしてしまうため、「全」と「挿し」に B-NOUN、「学年」と「絵」に I-NOUN が付与されています。一方、「大量」に ADJ が付与されていますが、これは NOUN の方が適切だと考えられます。

続いて、国語研長単位での係り受け解析[51]も見てみましょう。Google Colaboratory 上の esupar を用いて、「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」を係り受け解析するプログラムと、その結果を図 32 に示します。「小学校の国語の教科書」は、名詞が名詞を修飾していて nmod で繋がれていると同時に、「の」が case で繋がれています。「大量の挿し絵」も同様のはずですが、「大量」が NOUN ではなく ADJ になってしまっています。「挿し絵」はガ格であり、nsubj で繋がれていると同時に、「が」が case で繋がれています。「教科書」はニ格であり、obl で繋がれていると同時に、「に」が case で繋がれています。「全学年」もニ格と判断されていて、obl で繋がれていると同時に、「にわたって」が case で繋がれています。

[51]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.

3.4 国語研長単位ミニ RoBERTa モデルの製作

Google Colaboratory 上の esupar で、国語研長単位ミニ RoBERTa モデルを製作してみましょう。図 33 では、図 26〜28 で作成した my-dir/roberta-suw を、国語研長単位に使い回す形で、品詞付与・係り受け解析モデル my-dir/roberta-luw-upos を製作しています。国語研短単位の BertJapaneseTokenizer (内部的には fugashi と unidic-lite) を使うことになるので、国語研長単位の UD_Japanese-GSDLUW[52]に較べてトークンが短くなるのですが、esupar が UPOS 付与に「B-」「I-」を駆使することで、うまく動作するはずです (図 34)。

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

図 33: 国語研長単位向け品詞付与・係り受け解析ミニモデルの製作

```
!pip install esupar fugashi unidic-lite pytokenizations
import esupar
nlp=esupar.load("my-dir/roberta-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)
```

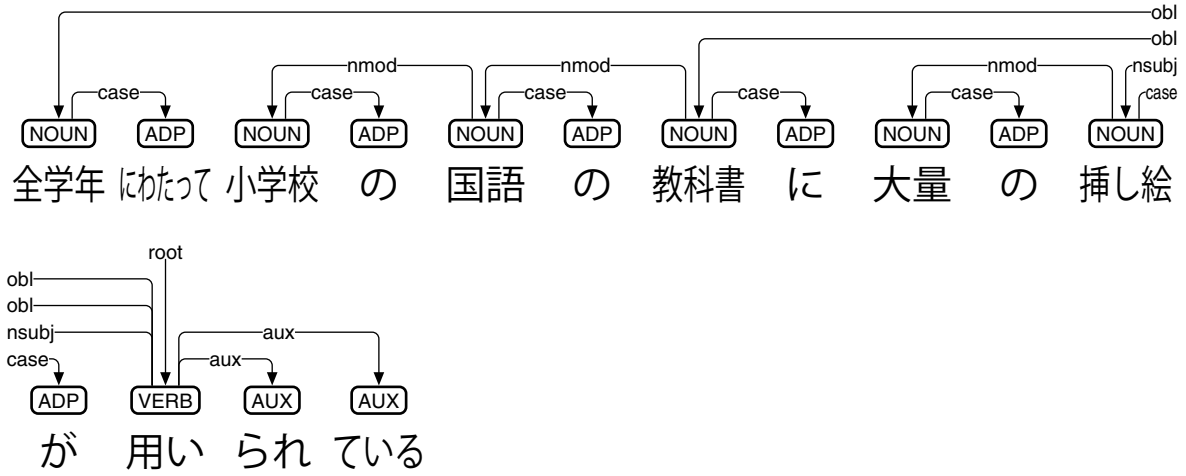


図 34: 国語研長単位向け品詞付与・係り受け解析ミニモデルのテスト

3.5 文節にもとづく係り受け解析

GiNZA[53]は、国語研短単位にもとづく状態遷移型係り受け解析アルゴリズムを実装しており、その上で、文節にもとづく係り受け解析が可能です。ただし、文節にもとづく

[52]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.

[53]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.

係り受けは、慣習的に文頭から文末へとリンクする[54]ため、矢印の方向が日本語 UD と逆になってしまいます。

Google Colaboratory 上の GiNZA を用いて、「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」を文節係り受け解析するプログラムと、その結果 (可視化は Graphviz[55]を使用) を図 35 に示します。文節にもとづく係り受け解析は、UD とは全く異なっていることから、ここでは参考に留めておきます。

```
!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)
```

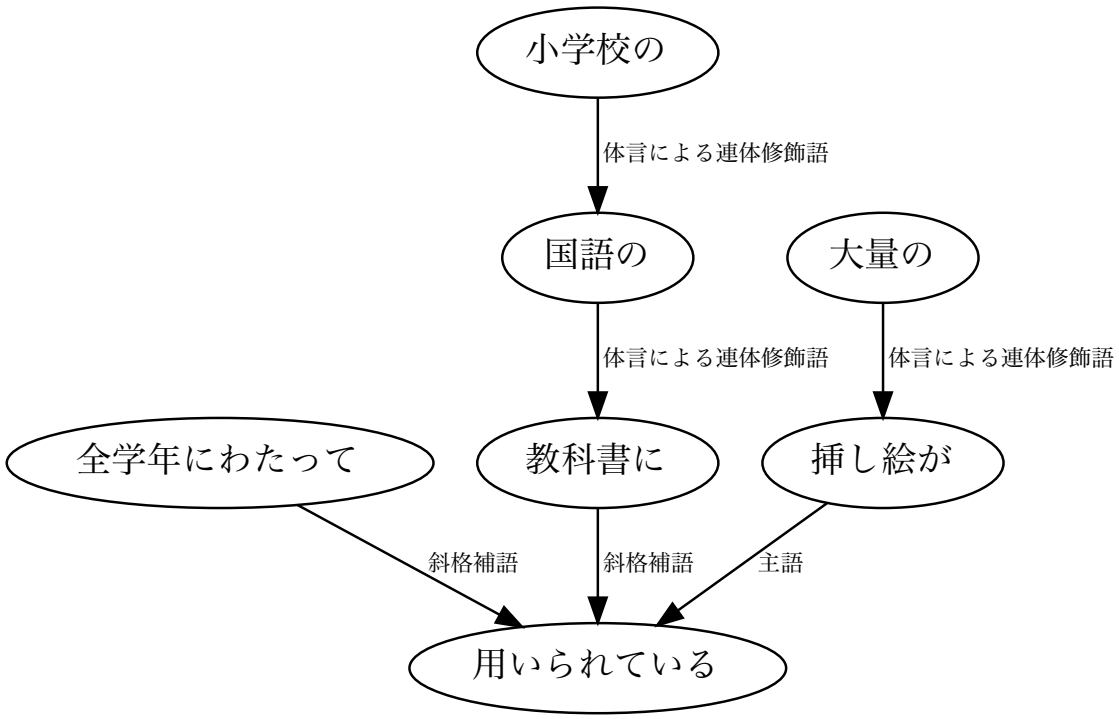


図 35: 文節にもとづく日本語係り受け解析

[54]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.

[55]https://graphviz.readthedocs.io

4 英語編

4.1 英語 UD における品詞付与と係り受け解析

Trankit[56]は、XLM-RoBERTa[57]をベースに、隣接行列型係り受け解析アルゴリズムを、様々な言語向けに実装しています。各言語向けに品詞付与も実装していて、複数の言語を同時に扱うことも可能です。Google Colaboratory 上の Trankit を用いて、英文「It didn't take long to repair my car」を品詞付与・係り受け解析するプログラムと、その結果を図 36 に示します。解析結果を見ていくことにしましょう。

```
!pip install trankit transformers deplacy
import trankit
nlp=trankit.Pipeline("english")
doc=nlp("It didn't take long to repair my car")
import deplacy
deplacy.serve(doc,port=None)
```

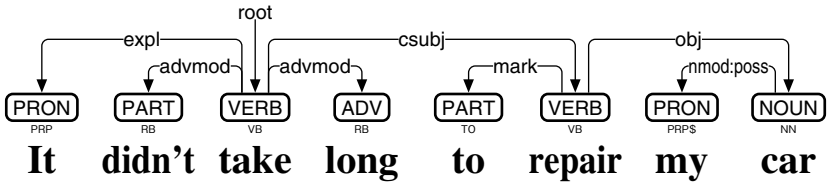


図 36: Trankit による英語の係り受け解析

動詞「take」の主語は「It」ですが、「It」は形式主語で、実際の主語は「to repair my car」という節 (clause) です。これを反映して、「It」が expl で、「repair」が csubj で繋がれています。「didn't」と「long」は「take」を連用修飾していて、いずれも advmod で繋がれています。「to repair my car」においては、「to」が節全体に前置されていて、mark で繋がれています。「car」は「repair」の目的語で、obj で繋がれています。「my」は「car」を修飾しており、本来は det が適切だと考えられるのですが、英語 UD では nmod:poss (所有に関する連体修飾) を導入しています。なお、Trankit は「didn't」を 1 語だとみなしていますが、英語 UD では「did」「n't」の 2 語に分けるべきであり、単語切りが不十分です。

UDPipe 2 WebAPI[58]は、UDPipe 2 の係り受け解析モデル[59]を、WebAPI の形で公開しています。内部的には、隣接行列型係り受け解析アルゴリズムを使っているようです。Google Colaboratory から UDPipe 2 WebAPI の英語モデルにアクセスするプログラムと、

[56]Minh Van Nguyen, Viet Dac Lai, Amir Pouran Ben Veyseh and Thien Huu Nguyen: Trankit: A Light-Weight Transformer-based Toolkit for Multilingual Natural Language Processing, EACL 2021: 16th Conference of the European Chapter of the Association for Computational Linguistics (April 2021), System Demonstrations, pp.80-90.

[57]Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, Veselin Stoyanov: Unsupervised Cross-lingual Representation Learning at Scale, Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (July 2020), pp.8440-8451.

[58]`https://lindat.mff.cuni.cz/services/udpipe/`

[59]Milan Straka: UDPipe 2.0 Prototype at CoNLL 2018 UD Shared Task, Proceedings of the CoNLL 2018 Shared Task (October 2018), pp.197-207.

```
!pip install deplacy
class UDPipe2WebAPI(object):
  def __init__(self,lang):
    self.url="https://lindat.mff.cuni.cz/services/udpipe/api/process"
    self.params={"model":lang,"tokenizer":"","tagger":"","parser":""}
  def __call__(self,text):
    import urllib.parse,urllib.request,json
    self.params["data"]=urllib.parse.quote(text)
    u=self.url+"?"+"&".join(k+"="+v for k,v in self.params.items())
    with urllib.request.urlopen(u) as r:
      return json.loads(r.read())["result"]
nlp=UDPipe2WebAPI("en")
doc=nlp("It didn't take long to repair my car")
import deplacy
deplacy.serve(doc,port=None)
```

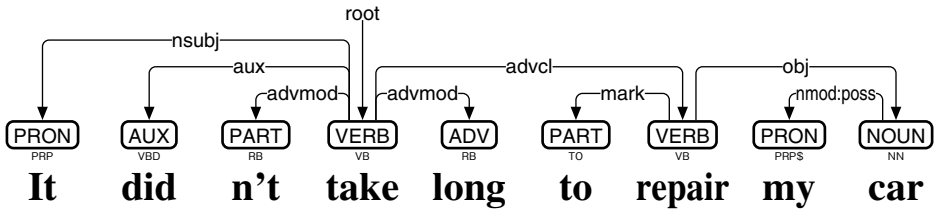


図 37: UDPipe 2 WebAPI による英語の係り受け解析

その結果を図 37 に示します。「It didn't take long to repair my car」の解析において、「did」「n't」が 2 語に分かれていて、それぞれ aux と advmod で繋がれています。ただし「It」が形式主語であることを解析しきれておらず、nsubj で繋がれています。この結果、「repair」は advcl で繋がれています。

なお、英語 UD の係り受けタグには、表 4 と nmod:poss に加え、斜格補語による連用修飾を表す obl:npmod、時に関する連体修飾を表す nmod:tmod、時に関する斜格補語を表す obl:tmod、受動態の主語を表す nsubj:pass、受動態の節主語を表す csubj:pass、関係代名詞による連体修飾節を表す acl:relcl、限定詞の前に置かれる形容詞を表す det:predet、接続における前置副詞を表す cc:preconj、句動詞を表す compound:prt も用いられます[60]。

4.2 英語ミニ RoBERTa モデルの製作

Google Colaboratory 上の esupar[37]で、英語ミニ RoBERTa モデルを製作してみましょう。手順を図 38〜41 に示します。

最初に、訓練のための文章を準備しましょう (図 38)。train.txt の文章の材料として、ここでは UD_English-EWT[61] と WikiText-2[62] を元にしていますが、好みに応じ、他の英文を増量するのもいいでしょう。なお、UD_English-EWT の各 CoNLL-U ファイルは、品詞付与・係り受け解析の学習にも用います。

続けて、英語向けトークナイザを、my-dir/tokenizer-en に作成しましょう。図 39 では、

[60] https://universaldependencies.org/en/#syntax
[61] https://github.com/UniversalDependencies/UD_English-EWT
[62] https://arxiv.org/abs/1609.07843

```
!pip install esupar pytokenizations
import os
url="https://github.com/UniversalDependencies/UD_English-EWT"
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="https://s3.amazonaws.com/research.metamind.io/wikitext/wikitext-2-v1.zip"
!test -d wikitext-2 || ( curl -LO {url} ; unzip {os.path.basename(url)} )
!sed "s/<unk>/[UNK]/g" wikitext-2/*.tokens >> train.txt
```

図 38: 英語ミニモデル製作のための準備

```
from transformers import BertTokenizerFast
from tokenizers import BertWordPieceTokenizer
s=["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"]
wpt=BertWordPieceTokenizer()
wpt.train(files=["train.txt"],vocab_size=8000,special_tokens=s)
wpt.save_model(".")
tkz=BertTokenizerFast(vocab_file="vocab.txt",never_split=s)
tkz.save_pretrained("my-dir/tokenizer-en")
```

図 39: 英語向けトークナイザの作成

```
from transformers import (AutoTokenizer,RobertaConfig,RobertaForMaskedLM,
  DataCollatorForLanguageModeling,TrainingArguments,Trainer)
tkz=AutoTokenizer.from_pretrained("my-dir/tokenizer-en",model_max_length=128)
t=tkz.convert_tokens_to_ids(["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"])
cfg=RobertaConfig(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=RobertaForMaskedLM(cfg),train_dataset=ReadLineDS("train.txt",tkz))
trn.train()
trn.save_model("my-dir/roberta-en")
tkz.save_pretrained("my-dir/roberta-en")
```

図 40: 英語ミニ RoBERTa モデルの作成

Transformers の BertTokenizerFast を使って、英語向けトークナイザを作成しています。語彙数 (vocab_size) を 8000 に絞っていますが、もう少し大きくした方がいいかもしれません。

次に train.txt の文章から、RoBERTa モデルを my-dir/roberta-en に作成しましょう。図 40 では、入出力幅 128 トークン・入出力ベクトル 256 次元・深さ 12 層・アテンションヘッド 4 個・中間ベクトル 768 次元という、やや小さめのモデルにしており、CPU なら 2 時間程度、GPU なら 20 分程度で作成できます。

最後に、my-dir/roberta-en と UD_English-EWT から、品詞付与・係り受け解析モデル my-dir/roberta-en-upos を製作しましょう (図 41)。うまく my-dir/roberta-en-upos が製作できたら、esupar でテストしてみましょう (図 42)。

```
!python -m esupar.train my-dir/roberta-en my-dir/roberta-en-upos .
```

図 41: 英語向け品詞付与・係り受け解析ミニモデルの製作

```
!pip install esupar
import esupar
nlp=esupar.load("my-dir/roberta-en-upos")
doc=nlp("It didn't take long to repair my car")
import deplacy
deplacy.serve(doc,port=None)
```

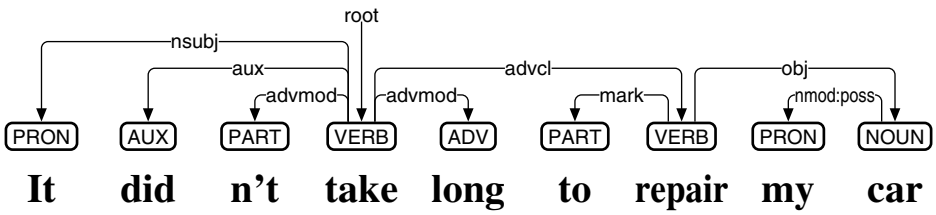


図 42: 英語向け品詞付与・係り受け解析ミニモデルのテスト

5　フランス語編

5.1　フランス語 UD における品詞付与と係り受け解析

spacy-transformers[63]は、事前学習モデルによる状態遷移型係り受け解析アルゴリズムを実装しており、フランス語向けには、CamemBERT[64]を元にした fr_dep_news_trf を公開[65]しています。Google Colaboratory 上の spacy-transformers を用いて、フランス語の例文「Il ne peut pas prêter attention aux articles contractés」を品詞付与・係り受け解析するプログラムと、その結果を図 43 に示します。解析結果を見ていくことにしましょう。

```
!pip install spacy-transformers deplacy
!python -m spacy download fr_dep_news_trf
import spacy
nlp=spacy.load("fr_dep_news_trf")
doc=nlp("Il ne peut pas prêter attention aux articles contractés")
import deplacy
deplacy.serve(doc,port=None)
```

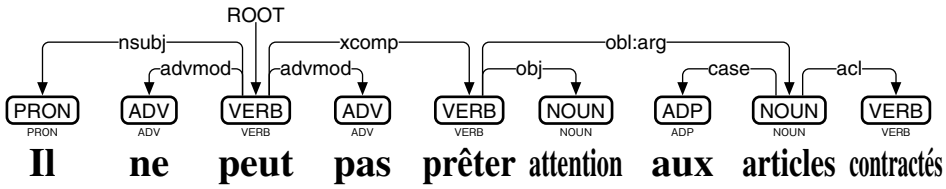


図 43: spacy-transformers によるフランス語の係り受け解析

動詞「peut」の主語「Il」は、同時に「prêter」の主語でもあることから、「Il」が nsubj で、「prêter」が xcomp で繋がれています。「ne」「pas」は「peut」を否定しているとみなして、いずれも advmod で繋がれています。「attention」は「prêter」の目的語であり、obj で繋がれています。斜格補語「articles」については、説明が必要です。フランス語 UD では、obl を obl:agent・obl:arg・obl:mod の 3 つに分離しており、それぞれ動作主 (agent)・項 (argument)・修飾子 (modificateur) を表しています。「articles」は obl:arg で繋がれていることから、「prêter」の項 (ただし動作主ではない) だとみなされています。「articles」に対し「aux」は前置詞、「contractés」は連体修飾節だとみなされており、それぞれ case・acl で繋がれています。なお、spacy-transformers は「aux」を 1 語だとみなしていますが、フランス語 UD では「à」「les」の 2 語に分離した上で扱うことになっており、縮合語の解析が不十分です。

Trankit[56]は、XLM-RoBERTa をベースに、隣接行列型係り受け解析アルゴリズムを、様々な言語向けに実装しています。Google Colaboratory 上の Trankit を用いて「Il ne peut pas prêter attention aux articles contractés」を品詞付与・係り受け解析するプログラムと、その結果を図 44 に示します。「aux」が内部的には 2 語として扱われており、「à」が「articles」の前置詞、「les」が限定詞 (冠詞) として、それぞれ case・det で繋がれています。

[63]https://github.com/explosion/spacy-transformers

[64]Louis Martin, Benjamin Muller, Pedro Javier Ortiz Suárez, Yoann Dupont, Laurent Romary, Éric Villemonte de la Clergerie, Djamé Seddah, Benoît Sagot: CamemBERT: a Tasty French Language Model, Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (July 2020), pp.7203-7219.

[65]https://huggingface.co/spacy/fr_dep_news_trf

```
!pip install trankit transformers deplacy
import trankit
nlp=trankit.Pipeline("french")
doc=nlp("Il ne peut pas prêter attention aux articles contractés")
import deplacy
deplacy.serve(doc,port=None)
```

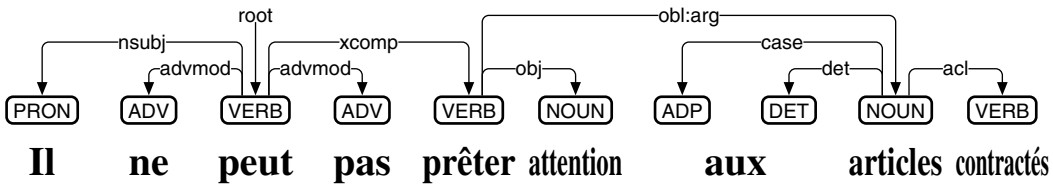


図 44: Trankit によるフランス語の係り受け解析

なお、フランス語 UD の係り受けタグには、表 4 と obl:agent・obl:arg・obl:mod に加え、数多くの拡張が用いられている[66]のですが、拡張どうしの間で混乱が起こっており、事態の収拾が望まれます。

5.2 フランス語ミニ RoBERTa モデルの製作

Google Colaboratory 上の esupar[37]で、フランス語ミニ RoBERTa モデルを製作してみましょう。手順を図 45〜48 に示します。

最初に、訓練のための文章を準備しましょう (図 45)。train.txt の文章の材料として、ここでは UD_French-Sequoia[67] と UD_French-GSD[68] を元にしていますが、好みに応じ、フランス語の他の文章を増量するのもいいでしょう。なお、各 CoNLL-U ファイルは、品詞付与・係り受け解析の学習にも用います。

続けて、フランス語向けトークナイザを、my-dir/tokenizer-fr に作成しましょう。図 46 では、Transformers の BertTokenizerFast を使って、フランス語向けトークナイザを作成しています。

次に train.txt の文章から、RoBERTa モデルを my-dir/roberta-fr に作成しましょう。図 47 では、入出力幅 128 トークン・入出力ベクトル 256 次元・深さ 12 層・アテンションヘッ

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

図 45: フランス語ミニモデル製作のための準備

[66] https://universaldependencies.org/fr/dep
[67] https://github.com/UniversalDependencies/UD_French-Sequoia
[68] https://github.com/UniversalDependencies/UD_French-GSD

```
from transformers import BertTokenizerFast
from tokenizers import BertWordPieceTokenizer
s=["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"]
wpt=BertWordPieceTokenizer(strip_accents=False)
wpt.train(files=["train.txt"],vocab_size=30000,special_tokens=s)
wpt.save_model(".")
tkz=BertTokenizerFast(vocab_file="vocab.txt",strip_accents=False,never_split=s)
tkz.save_pretrained("my-dir/tokenizer-fr")
```

図 46: フランス語向けトークナイザの作成

```
from transformers import (AutoTokenizer,RobertaConfig,RobertaForMaskedLM,
  DataCollatorForLanguageModeling,TrainingArguments,Trainer)
tkz=AutoTokenizer.from_pretrained("my-dir/tokenizer-fr",model_max_length=128)
t=tkz.convert_tokens_to_ids(["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"])
cfg=RobertaConfig(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=RobertaForMaskedLM(cfg),train_dataset=ReadLineDS("train.txt",tkz))
trn.train()
trn.save_model("my-dir/roberta-fr")
tkz.save_pretrained("my-dir/roberta-fr")
```

図 47: フランス語ミニ RoBERTa モデルの作成

```
!python -m esupar.train my-dir/roberta-fr my-dir/roberta-fr-upos .
```

図 48: フランス語向け品詞付与・係り受け解析ミニモデルの製作

ド 4 個・中間ベクトル 768 次元という、やや小さめのモデルにしており、CPU なら 1 時間程度、GPU なら 10 分程度で作成できます。

最後に、my-dir/roberta-fr と UD_French-Sequoia・UD_French-GSD から、品詞付与・係り受け解析モデル my-dir/roberta-fr-upos を製作しましょう (図 48)。ただ、隣接行列型アルゴリズムの学習は収束が遅く、GPU でも 3 時間程度かかってしまいます。

うまく my-dir/roberta-fr-upos が製作できたら、esupar でテストしてみましょう (図 49)。esupar は縮合語を UPOS 付与においてサポート (図 50) しており、係り受け解析では「aux」を 2 語として処理します。

```
!pip install esupar
import esupar
nlp=esupar.load("my-dir/roberta-fr-upos")
doc=nlp("Il ne peut pas prêter attention aux articles contractés")
import deplacy
deplacy.serve(doc,port=None)
```

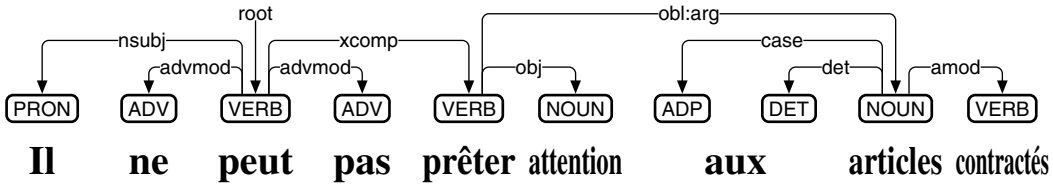


図 49: フランス語向け品詞付与・係り受け解析ミニモデルのテスト

```
!pip install deplacy transformers
class SeqL(object):
  def __init__(self,bert,aggr="none"):
    from transformers import pipeline
    self.pipeline=pipeline(task="ner",model=bert,aggregation_strategy=aggr)
    self.entity="entity" if aggr=="none" else "entity_group"
  def __call__(self,text):
    w=[(t["start"],t["end"],t[self.entity]) for t in self.pipeline(text)]
    u="# text = "+text.replace("\n"," ")+"\n"
    for i,(s,e,p) in enumerate(w,1):
      m="_" if i<len(w) and w[i][0]<e else "SpaceAfter=No"
      u+="\t".join([str(i),text[s:e],"_",p]+["_"]*5+[m])+"\n"
    return u+"\n"
nlp=SeqL("my-dir/roberta-fr-upos")
doc=nlp("Il ne peut pas prêter attention aux articles contractés")
import deplacy
deplacy.serve(doc,port=None)
```

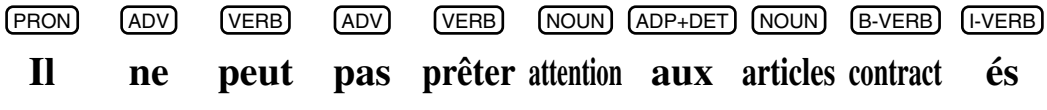


図 50: フランス語向け品詞付与・係り受け解析ミニモデルでの UPOS 付与

6　ドイツ語編

6.1　ドイツ語 UD における品詞付与と係り受け解析

Trankit[56]は、XLM-RoBERTa をベースに、隣接行列型係り受け解析アルゴリズムを、様々な言語向けに実装しています。Google Colaboratory 上の Trankit を用いて、ドイツ語の例文「Im Krankenzimmer sah er krank aus aber wollte gut aussehen」を品詞付与・係り受け解析するプログラムと、その結果を図 51 に示します。

```
!pip install trankit transformers deplacy
import trankit
nlp=trankit.Pipeline("german")
doc=nlp("Im Krankenzimmer sah er krank aus aber wollte gut aussehen")
import deplacy
deplacy.serve(doc,port=None)
```

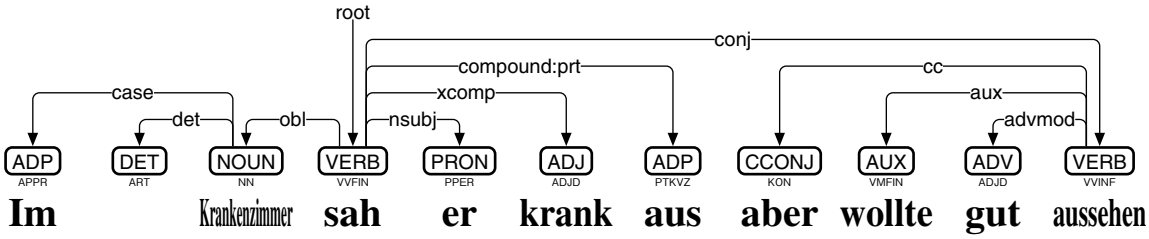


図 51: Trankit によるドイツ語の係り受け解析

ドイツ語 UD は、単語の扱いが非常に独特です。図 51 の「Im Krankenzimmer」では、縮合語「Im」を前置詞「In」と冠詞「dem」の 2 語に分離して扱う一方、複合語である「Krankenzimmer」は 1 語として扱っています。この結果、「Krankenzimmer」から前置詞へ case が、冠詞へ det が繋がれています。「aussehen」など分離動詞は、分離していない場合は 1 語として扱い、分離している場合は 2 語として扱うと同時に compound:prt で繋ぐ、というルールが採用されています。結果として「sah」から「aus」へ compound:prt が繋がれています。また「sah」からは、主語「er」が nsubj で、補語「krank」が xcomp で、斜格補語「Krankenzimmer」が obl で、それぞれ繋がれています。「sah aus」と「aussehen」は並置されていて、conj で繋がれています。「aussehen」からは、接続詞「aber」が cc で、助動詞「wollte」が aux で繋がれています。補語「gut」は advmod で繋がれていますが、これは xcomp の方が適切だと考えられます。

なお、ドイツ語 UD の係り受けタグには、表 4 と compound:prt に加え、受動態を表す nsubj:pass・csubj:pass・aux:pass、斜格補語の動作主を表す obl:agent、項 (動作主以外) の斜格補語を表す obl:arg、所有を表す det:pos・nmod:poss、再帰動詞における再帰代名詞を表す expl:pv も用いられます[69]。

[69]https://universaldependencies.org/de/#relations-overview

6.2 ドイツ語ミニ RoBERTa モデルの製作

Google Colaboratory 上の esupar[37]で、ドイツ語ミニ RoBERTa モデルを製作してみましょう。手順を図 52〜55 に示します。

最初に、訓練のための文章を準備しましょう (図 52)。train.txt の文章の材料として、ここでは UD_German-HDT[70]を元にしていますが、好みに応じ、ドイツ語の他の文章を増量するのもいいでしょう。なお、UD_German-HDT の CoNLL-U ファイルは、品詞付与・係り受け解析の学習にも用います。

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

図 52: ドイツ語ミニモデル製作のための準備

続けて、ドイツ語向けトークナイザを、my-dir/tokenizer-de に作成しましょう。図 53 では、Transformers の BertTokenizerFast を使って、ドイツ語向けトークナイザを作成しています。

```
from transformers import BertTokenizerFast
from tokenizers import BertWordPieceTokenizer
s=["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"]
wpt=BertWordPieceTokenizer(strip_accents=False)
wpt.train(files=["train.txt"],vocab_size=30000,special_tokens=s)
wpt.save_model(".")
tkz=BertTokenizerFast(vocab_file="vocab.txt",strip_accents=False,never_split=s)
tkz.save_pretrained("my-dir/tokenizer-de")
```

図 53: ドイツ語向けトークナイザの作成

次に train.txt の文章から、RoBERTa モデルを my-dir/roberta-de に作成しましょう。図 54 では、入出力幅 128 トークン・入出力ベクトル 256 次元・深さ 12 層・アテンションヘッド 4 個・中間ベクトル 768 次元という、やや小さめのモデルにしていますが、GPU でも 1 時間程度かかってしまいます。

最後に、my-dir/roberta-de と UD_German-HDT から、品詞付与・係り受け解析モデル my-dir/roberta-de-upos を製作しましょう (図 55)。うまく my-dir/roberta-de-upos が製作できたら、esupar でテストしてみましょう (図 56)。esupar は縮合語を UPOS 付与においてサポート (図 57) しており、係り受け解析では「Im」を 2 語として処理します。

[70] https://github.com/UniversalDependencies/UD_German-HDT

```
from transformers import (AutoTokenizer,RobertaConfig,RobertaForMaskedLM,
  DataCollatorForLanguageModeling,TrainingArguments,Trainer)
tkz=AutoTokenizer.from_pretrained("my-dir/tokenizer-de",model_max_length=128)
t=tkz.convert_tokens_to_ids(["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"])
cfg=RobertaConfig(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=RobertaForMaskedLM(cfg),train_dataset=ReadLineDS("train.txt",tkz))
trn.train()
trn.save_model("my-dir/roberta-de")
tkz.save_pretrained("my-dir/roberta-de")
```

図 54: ドイツ語ミニ RoBERTa モデルの作成

```
!python -m esupar.train my-dir/roberta-de my-dir/roberta-de-upos .
```

図 55: ドイツ語向け品詞付与・係り受け解析ミニモデルの製作

```
!pip install esupar
import esupar
nlp=esupar.load("my-dir/roberta-de-upos")
doc=nlp("Im Krankenzimmer sah er krank aus aber wollte gut aussehen")
import deplacy
deplacy.serve(doc,port=None)
```

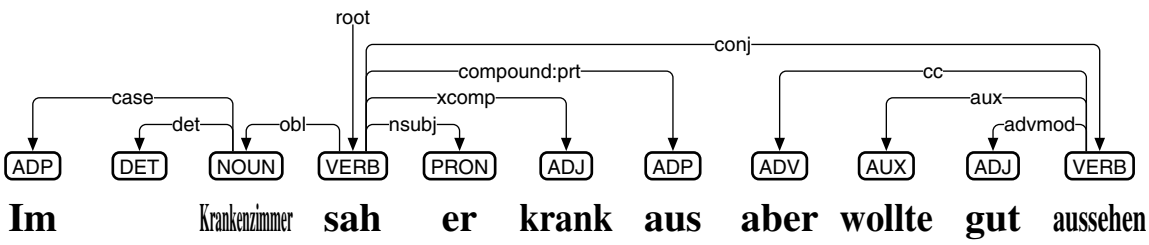


図 56: ドイツ語向け品詞付与・係り受け解析ミニモデルのテスト

```
!pip install deplacy transformers
class SeqL(object):
  def __init__(self,bert,aggr="none"):
    from transformers import pipeline
    self.pipeline=pipeline(task="ner",model=bert,aggregation_strategy=aggr)
    self.entity="entity" if aggr=="none" else "entity_group"
  def __call__(self,text):
    w=[(t["start"],t["end"],t[self.entity]) for t in self.pipeline(text)]
    u="# text = "+text.replace("\n"," ")+"\n"
    for i,(s,e,p) in enumerate(w,1):
      m="_" if i<len(w) and w[i][0]<e else "SpaceAfter=No"
      u+="\t".join([str(i),text[s:e],"_",p]+["_"]*5+[m])+"\n"
    return u+"\n"
nlp=SeqL("my-dir/roberta-de-upos")
doc=nlp("Im Krankenzimmer sah er krank aus aber wollte gut aussehen")
import deplacy
deplacy.serve(doc,port=None)
```

図 57: ドイツ語向け品詞付与・係り受け解析ミニモデルでの UPOS 付与

7　オランダ語編

7.1　オランダ語 UD における品詞付与と係り受け解析

Trankit[56]は、XLM-RoBERTa をベースに、隣接行列型係り受け解析アルゴリズムを、様々な言語向けに実装しています。Google Colaboratory 上の Trankit を用いて、オランダ語の例文「In de ziekenkamer zag hij ziek uit maar wilde goed uitzien」を品詞付与・係り受け解析するプログラムと、その結果を図 58 に示します。解析結果を見ていくことにしましょう。

```
!pip install trankit transformers deplacy
import trankit
nlp=trankit.Pipeline("dutch")
doc=nlp("In de ziekenkamer zag hij ziek uit maar wilde goed uitzien")
import deplacy
deplacy.serve(doc,port=None)
```

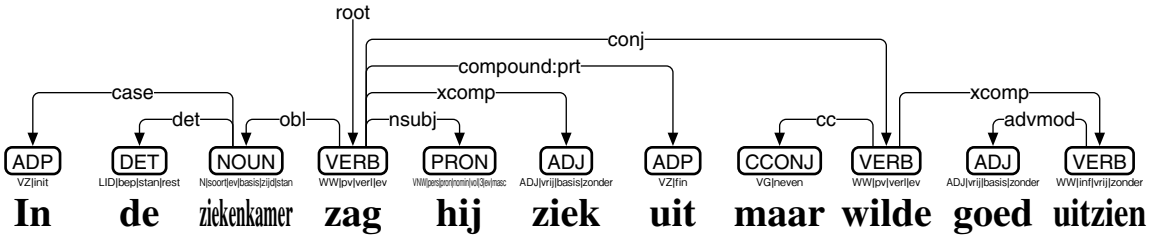


図 58: Trankit によるオランダ語の係り受け解析

「uitzien」など分離動詞は、分離していない場合は 1 語として扱い、分離している場合は 2 語として扱うと同時に compound:prt で繋ぐ、というルールが採用されています。結果として「zag」から「uit」へ compound:prt が繋がれています。また「zag」からは、主語「hij」が nsubj で、補語「ziek」が xcomp で、斜格補語「ziekenkamer」が obl で、それぞれ繋がれています。「ziekenkamer」からは、前置詞「In」が case で、冠詞「de」が det で繋がれています。「zag uit」と「wilde uitzien」は並置されていて、conj で繋がれています。「wilde」には接続詞「maar」が cc で、「uitzien」が xcomp で繋がれています。「uitzien」と「goed」は advmod で繋がれていますが、これは xcomp の方が適切だと考えられます。

なお、オランダ語 UD の係り受けタグには、表 4 と compound:prt に加え、受動態を表す nsubj:pass と aux:pass、受動態の動作主を表す obl:agent、所有を表す nmod:pos、関係代名詞による連体修飾節を表す acl:relcl、再帰動詞における再帰代名詞を表す expl:pv も用いられます[71]。

7.2　オランダ語ミニ RoBERTa モデルの製作

Google Colaboratory 上の esupar[37]で、オランダ語ミニ RoBERTa モデルを製作してみましょう。手順を図 59〜62 に示します。

[71] https://universaldependencies.org/nl/#syntax

最初に、訓練のための文章を準備しましょう (図 59)。train.txt の文章の材料として、ここでは UD_Dutch-Alpino[72] と UD_Dutch-LassySmall[73] を元にしていますが、好みに応じ、オランダ語の他の文章を増量するのもいいでしょう。なお、各 CoNLL-U ファイルは、品詞付与・係り受け解析の学習にも用います。

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

図 59: オランダ語ミニモデル製作のための準備

続けて、オランダ語向けトークナイザを、my-dir/tokenizer-nl に作成しましょう。図 60 では、Transformers の BertTokenizerFast を使って、オランダ語向けトークナイザを作成しています。語彙数 (vocab_size) を 8000 に絞っていますが、もう少し大きくした方がいいかもしれません。

```
from transformers import BertTokenizerFast
from tokenizers import BertWordPieceTokenizer
s=["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"]
wpt=BertWordPieceTokenizer(strip_accents=False)
wpt.train(files=["train.txt"],vocab_size=8000,special_tokens=s)
wpt.save_model(".")
tkz=BertTokenizerFast(vocab_file="vocab.txt",strip_accents=False,never_split=s)
tkz.save_pretrained("my-dir/tokenizer-nl")
```

図 60: オランダ語向けトークナイザの作成

次に train.txt の文章から、RoBERTa モデルを my-dir/roberta-nl に作成しましょう。図 61 では、入出力幅 128 トークン・入出力ベクトル 256 次元・深さ 12 層・アテンションヘッド 4 個・中間ベクトル 768 次元という、やや小さめのモデルにしており、CPU なら 1 時間程度、GPU なら 10 分程度で作成できます。

最後に、my-dir/roberta-nl と UD_Dutch-Alpino・UD_Dutch-LassySmall から、品詞付与・係り受け解析モデル my-dir/roberta-nl-upos を製作しましょう (図 62)。うまく my-dir/roberta-nl-upos が製作できたら、esupar でテストしてみましょう (図 63)。

[72] https://github.com/UniversalDependencies/UD_Dutch-Alpino
[73] https://github.com/UniversalDependencies/UD_Dutch-LassySmall

```
from transformers import (AutoTokenizer,RobertaConfig,RobertaForMaskedLM,
  DataCollatorForLanguageModeling,TrainingArguments,Trainer)
tkz=AutoTokenizer.from_pretrained("my-dir/tokenizer-nl",model_max_length=128)
t=tkz.convert_tokens_to_ids(["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"])
cfg=RobertaConfig(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=RobertaForMaskedLM(cfg),train_dataset=ReadLineDS("train.txt",tkz))
trn.train()
trn.save_model("my-dir/roberta-nl")
tkz.save_pretrained("my-dir/roberta-nl")
```

図 61: オランダ語ミニ RoBERTa モデルの作成

```
!python -m esupar.train my-dir/roberta-nl my-dir/roberta-nl-upos .
```

図 62: オランダ語向け品詞付与・係り受け解析ミニモデルの製作

```
!pip install esupar
import esupar
nlp=esupar.load("my-dir/roberta-nl-upos")
doc=nlp("In de ziekenkamer zag hij ziek uit maar wilde goed uitzien")
import deplacy
deplacy.serve(doc,port=None)
```

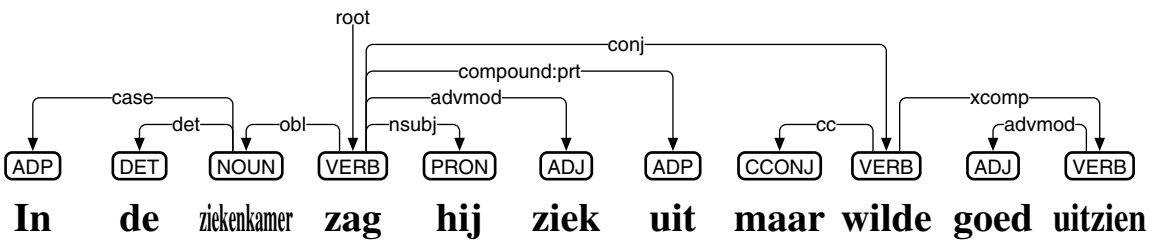


図 63: オランダ語向け品詞付与・係り受け解析ミニモデルのテスト

8　タイ語編

8.1　タイ語 UD における品詞付与

書写言語としてのタイ語は、単語と単語の間に区切りがありません。数字や氏名の前後には、空白を入れる場合があるのですが、単語の区切りというわけではないのです。事前学習モデルでタイ語を扱う際には、各トークンを単語と合致させるのが理想です。しかし、それは現実には困難なため、各トークンは単語より短めにトークナイズした上で、系列ラベリング (UPOS 付与) によって、単語をまとめ上げる手法が考えられます。すなわち、トークンが単語と合致した場合には UPOS をラベルとして付与し、そうでない場合には、最初のトークンに「B-」を付けた UPOS を、それ以降のトークンに「I-」を付けた UPOS を、それぞれ付与するのです。

```
!pip install deplacy transformers
class SeqL(object):
  def __init__(self,bert,aggr="none"):
    from transformers import pipeline
    self.pipeline=pipeline(task="ner",model=bert,aggregation_strategy=aggr)
    self.entity="entity" if aggr=="none" else "entity_group"
  def __call__(self,text):
    w=[(t["start"],t["end"],t[self.entity]) for t in self.pipeline(text)]
    u="# text = "+text.replace("\n"," ")+"\n"
    for i,(s,e,p) in enumerate(w,1):
      m="_" if i<len(w) and w[i][0]<e else "SpaceAfter=No"
      u+="\t".join([str(i),text[s:e],"_",p]+["_"]*5+[m])+"\n"
    return u+"\n"
nlp=SeqL("KoichiYasuoka/roberta-base-thai-spm-upos")
doc=nlp("หลายหัวดีกว่าหัวเดียว")
import deplacy
deplacy.serve(doc,port=None)
```

図 64: roberta-base-thai-spm-upos による UPOS 付与

Google Colaboratory 上の Transformers を用いて、roberta-base-thai-spm-upos[74]でタイ語の例文「หลายหัวดีกว่าหัวเดียว」に UPOS を付与するプログラムと、その結果を図 64 に示します。「หลาย」「หัว」「ดี」「กว่า」の各単語では、各トークンがそのまま単語と合致しており、それぞれ UPOS 品詞 DET・NOUN・VERB・ADP が付与されています。「เดียว」は、2 つのトークンに泣き別れてしまっており、それぞれに B-NUM と I-NUM が付与されています。なお「ดี」は形容詞とみなせますが、タイ語においては形容詞と動詞に文法的な区別がないことから、ADJ ではなく VERB が付与されています。

[74]https://huggingface.co/KoichiYasuoka/roberta-base-thai-spm-upos

8.2 タイ語UDにおける係り受け解析

esupar[37]は、隣接行列型の係り受け解析アルゴリズムを、事前学習モデルで実装しています。Google Colaboratory 上の esupar を用いて、roberta-base-thai-spm-upos で例文「หลายหัวดีกว่าหัวเดียว」を係り受け解析するプログラムと、その結果を図 65 に示します。解析結果を見ていくことにしましょう。

```
!pip install esupar
import esupar
nlp=esupar.load("KoichiYasuoka/roberta-base-thai-spm-upos")
doc=nlp("หลายหัวดีกว่าหัวเดียว")
import deplacy
deplacy.serve(doc,port=None)
```

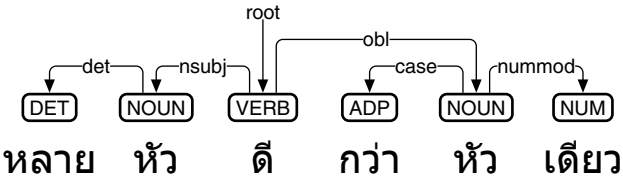


図 65: roberta-base-thai-spm-upos によるタイ語の係り受け解析

動詞 (形容詞)「ดี」には、名詞「หัว」が 2 つ繋がっており、前者が主語 (nsubj)、後者が斜格補語 (obl) です。2 つの「หัว」は、それぞれ数量詞「หลาย」「เดียว」で修飾されており、前者は名詞に前置されていて det で、後者は名詞に後置されていて nummod で繋がれています。後者の「หัว」には、前置詞「กว่า」が case で繋がれています。

8.3 タイ語ミニDeBERTaモデルの製作

Google Colaboratory 上の esupar で、タイ語ミニ DeBERTa モデルを製作してみましょう。手順を図 66〜69 に示します。

最初に、訓練のための文章を準備しましょう (図 66)。train.txt の文章の材料として、ここでは VISTEC-TP-TH-2021[75]と UD_Thai-Corpora[76]を元にしていますが、好みに応じ、タイ語の他の文章を増量するのもいいでしょう。train.txt と同時に、各単語を空白で区切った token.txt も準備しており、トークナイザの作成に用います。UD_Thai-Corpora の各 CoNLL-U ファイルは、train.upos にまとめ上げて、品詞付与の学習にも用います。また、これらの CoNLL-U ファイルのうち、依存構造記述が UD のフォーマットを満たしているものは、train.conllu・dev.conllu・test.conllu に分け、係り受け解析の学習にも用います。

続けて、token.txt の各単語から、タイ語向けトークナイザを、my-dir/tokenizer-th に作成しましょう。図 67 では、Unigram トークナイザ (SentencePiece[77]の改造版) を token.txt

[75]Peerat Limkonchotiwat, Wannaphong Phatthiyaphaibun, Raheem Sarwar, Ekapol Chuangsuwanich, Sarana Nutanong: Handling Cross- and Out-of-Domain Samples in Thai Word Segmentaion, Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021 (August 2021), pp.1003-1016.

[76]https://github.com/KoichiYasuoka/spaCy-Thai/tree/master/UD_Thai-Corpora

[77]Taku Kudo, John Richardson: SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing, Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (November 2018): System Demonstrations, pp.66-71.

```
!pip install esupar pytokenizations
import os
url="https://github.com/mrpeerat/OSKut"
d=os.path.join(os.path.basename(url),"VISTEC-TP-TH-2021")
!test -d {d} || git clone --depth=1 {url}
!cat {d}/*/*raw.txt > train.txt
!sed "s/<[^>]*>//g;s/[|_]/ /g" {d}/*/*processed.txt > token.txt
url="https://github.com/KoichiYasuoka/spaCy-Thai"
d=os.path.join(os.path.basename(url),"UD_Thai-Corpora")
!test -d {d} || git clone --depth=1 {url}
s='''{print>"train.upos";if($0~/^# text = /)print substr($0,10)>>"train.txt";
  if(NF==10&&$1~/^[1-9][0-9]*$/)printf($1>1?" %s":"%s",$2)>>"token.txt";
  if(NF>0)u=u$0"\\n";else{print>>"token.txt";f=FILENAME;if(u~/\\t0\\troot\\t/)
  print u>(f~/-dev/?"dev":f~/-test/?"test":"train")".conllu";u=""}}'''
!nawk -F'\t' '{s}' {d}/*-ud-*.conllu
```

図 66: タイ語ミニモデル製作のための準備

```
from transformers import DebertaV2TokenizerFast
from tokenizers import (Tokenizer,models,pre_tokenizers,normalizers,processors,
  decoders,trainers)
s=["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"]
spt=Tokenizer(models.Unigram())
spt.pre_tokenizer=pre_tokenizers.Sequence([pre_tokenizers.Whitespace(),
  pre_tokenizers.Punctuation()])
spt.normalizer=normalizers.Sequence([normalizers.Nmt(),normalizers.NFKC()])
spt.post_processor=processors.TemplateProcessing(single="[CLS] $A [SEP]",
  pair="[CLS] $A [SEP] $B:1 [SEP]:1",special_tokens=[("[CLS]",0),("[SEP]",2)])
spt.decoder=decoders.WordPiece(prefix="",cleanup=True)
spt.train(trainer=trainers.UnigramTrainer(vocab_size=3000,max_piece_length=4,
  special_tokens=s,unk_token="[UNK]",n_sub_iterations=2),files=["token.txt"])
spt.save("tokenizer.json")
tkz=DebertaV2TokenizerFast(tokenizer_file="tokenizer.json",split_by_punct=True,
  do_lower_case=False,keep_accents=True,vocab_file="/dev/null")
tkz.save_pretrained("my-dir/tokenizer-th")
```

図 67: タイ語向けトークナイザの作成

に適用し、それを DebertaV2TokenizerFast という Transformers のトークナイザに組み上げて、タイ語向けトークナイザを作成しています。語彙数を 3000 に絞った上で、各トークンの文字数を最大 4 文字 (母音や声調記号を 1 文字に数える) としています。

次に train.txt の文章から、DeBERTa モデルを my-dir/deberta-th に作成しましょう。図 68 では、入出力幅 128 トークン・入出力ベクトル 256 次元・深さ 12 層・アテンションヘッド 4 個・中間ベクトル 768 次元という、やや小さめのモデルにしていますが、GPU でも 1 時間程度かかってしまいます。

最後に、my-dir/deberta-th と UD_Thai-Corpora から、品詞付与・係り受け解析モデル my-dir/deberta-th-upos を製作しましょう (図 69)。うまく my-dir/deberta-th-upos ができたら、esupar でテストしてみましょう (図 70)。

```
from transformers import (AutoTokenizer,DebertaV2Config,DebertaV2ForMaskedLM,
  DataCollatorForLanguageModeling,TrainingArguments,Trainer)
tkz=AutoTokenizer.from_pretrained("my-dir/tokenizer-th",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-th")
tkz.save_pretrained("my-dir/deberta-th")
```

図 68: タイ語ミニ DeBERTa モデルの作成

```
s,d="my-dir/deberta-th","my-dir/deberta-th-upos"
!python -m esupar.train {s} {d} 32 /tmp train.upos
!python -m esupar.train {d} {d} 32 /// train.conllu dev.conllu test.conllu
```

図 69: タイ語向け品詞付与・係り受け解析ミニモデルの製作

```
!pip install esupar
import esupar
nlp=esupar.load("my-dir/deberta-th-upos")
doc=nlp("หลายหัวดีกว่าหัวเดียว")
import deplacy
deplacy.serve(doc,port=None)
```

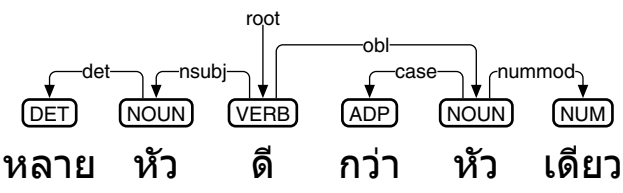


図 70: タイ語向け品詞付与・係り受け解析ミニモデルのテスト

9　ロシア語編

9.1　ロシア語 UD における品詞付与と係り受け解析

Trankit[56]は、XLM-RoBERTa をベースに、隣接行列型係り受け解析アルゴリズムを、様々な言語向けに実装しています。Google Colaboratory 上の Trankit を用いて、ロシア語の例文「Москва слезам не верила а верила любви」を品詞付与・係り受け解析するプログラムと、その結果を図 71 に示します。解析結果を見ていくことにしましょう。

```
!pip install trankit transformers deplacy
import trankit
nlp=trankit.Pipeline("russian")
doc=nlp("Москва слезам не верила а верила любви")
import deplacy
deplacy.serve(doc,port=None)
```

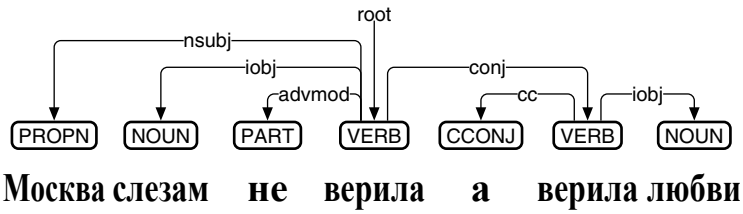


図 71: Trankit によるロシア語の係り受け解析

2 つの動詞「верила」が並置されていて、conj で繋がれています。前者には、否定の「не」が advmod で繋がれています。後者には、接続詞「а」が cc で繋がれています。「Москва」は、両方の「верила」の主語ですが、前者に nsubj で繋がれています。前者の目的語は「слезам」、後者の目的語は「любви」で、いずれも obj ではなく iobj で繋がれています。Trankit は、与格 (датив) の目的語を、間接目的語 (iobj) だとみなしてしまうようです。

なお、ロシア語 UD の係り受けタグには、表 4 に加え、受動態を表す nsubj:pass・csubj:pass・aux:pass、受動態の動作主を表す obl:agent、関係代名詞による連体修飾節を表す acl:relcl、数詞と名詞の格変化の関係を示す nummod:gov・nummod:entity、氏名を表す flat:name、ロシア語以外を表す flat:foreign も用いられます[78]。

9.2　ロシア語ミニ RoBERTa モデルの製作

Google Colaboratory 上の esupar[37]で、ロシア語ミニ RoBERTa モデルを製作してみましょう。手順を図 72〜75 に示します。

最初に、訓練のための文章を準備しましょう (図 72)。train.txt の文章の材料として、ここでは UD_Russian-SynTagRus[79]と UD_Russian-Taiga[80]を元にしていますが、好みに応

[78]https://universaldependencies.org/ru/#relations-overview

[79]Kira Droganova, Olga Lyashevskaya, Daniel Zeman: Data Conversion and Consistency of Monolingual Corpora: Russian UD Treebanks, Proceedings of the 17th International Workshop on Treebanks and Linguistic Theories (December 2018), pp.52-65.

[80]T. O. Sharvina, O. Shapovalova: To the Methodology of Corpus Construction for Machine Learning: “Taiga” Syntax Tree Corpus and Parser, Proceedings of «CORPUS LINGUISTICS–2017» (June 2017), pp.78-84.

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

図 72: ロシア語ミニモデル製作のための準備

```
from transformers import BertTokenizerFast
from tokenizers import BertWordPieceTokenizer
s=["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"]
wpt=BertWordPieceTokenizer(strip_accents=False)
wpt.train(files=["train.txt"],vocab_size=30000,special_tokens=s)
wpt.save_model(".")
tkz=BertTokenizerFast(vocab_file="vocab.txt",strip_accents=False,never_split=s)
tkz.save_pretrained("my-dir/tokenizer-ru")
```

図 73: ロシア語向けトークナイザの作成

```
from transformers import (AutoTokenizer,RobertaConfig,RobertaForMaskedLM,
  DataCollatorForLanguageModeling,TrainingArguments,Trainer)
tkz=AutoTokenizer.from_pretrained("my-dir/tokenizer-ru",model_max_length=128)
t=tkz.convert_tokens_to_ids(["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"])
cfg=RobertaConfig(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=RobertaForMaskedLM(cfg),train_dataset=ReadLineDS("train.txt",tkz))
trn.train()
trn.save_model("my-dir/roberta-ru")
tkz.save_pretrained("my-dir/roberta-ru")
```

図 74: ロシア語ミニ RoBERTa モデルの作成

じ、ロシア語の他の文章を増量するのもいいでしょう。なお、各 CoNLL-U ファイルは、品詞付与・係り受け解析の学習にも用います。

続けて、ロシア語向けトークナイザを、my-dir/tokenizer-ru に作成しましょう。図 73 では、Transformers の BertTokenizerFast を使って、ロシア語向けトークナイザを作成しています。

次に train.txt の文章から、RoBERTa モデルを my-dir/roberta-ru に作成しましょう。図 74 では、入出力幅 128 トークン・入出力ベクトル 256 次元・深さ 12 層・アテンションヘッド 4 個・中間ベクトル 768 次元という、やや小さめのモデルにしています。

最後に、my-dir/roberta-ru と、UD_Russian-SynTagRus・UD_Russian-Taiga の CoNLL-U ファイルから、品詞付与・係り受け解析モデル my-dir/roberta-ru-upos を製作しましょう (図 75)。うまく my-dir/roberta-ru-upos が製作できたら、esupar でテストしてみましょう (図 76)。

```
!python -m esupar.train my-dir/roberta-ru my-dir/roberta-ru-upos .
```

図 75: ロシア語向け品詞付与・係り受け解析ミニモデルの製作

```
!pip install esupar
import esupar
nlp=esupar.load("my-dir/roberta-ru-upos")
doc=nlp("Москва слезам не верила а верила любви")
import deplacy
deplacy.serve(doc,port=None)
```

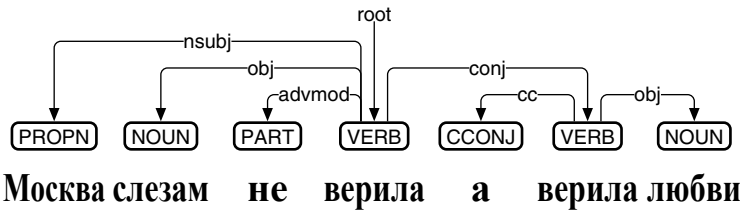


図 76: ロシア語向け品詞付与・係り受け解析ミニモデルのテスト

10　ウクライナ語編

10.1　ウクライナ語 UD における品詞付与と係り受け解析

Trankit[56]は、XLM-RoBERTa をベースに、隣接行列型係り受け解析アルゴリズムを、様々な言語向けに実装しています。Google Colaboratory 上の Trankit を用いて、ウクライナ語の例文「Не скупись на втіху їй і ласку любий брате」を品詞付与・係り受け解析するプログラムと、その結果を図 77 に示します。解析結果を見ていくことにしましょう。

```
!pip install trankit transformers deplacy
import trankit
nlp=trankit.Pipeline("ukrainian")
doc=nlp("Не скупись на втіху їй і ласку любий брате")
import deplacy
deplacy.serve(doc,port=None)
```

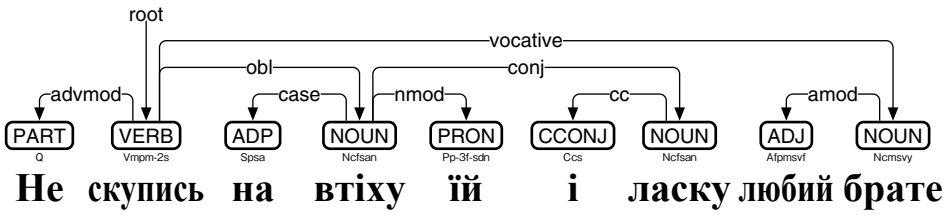


図 77: Trankit によるウクライナ語の係り受け解析

不完了相動詞「скупись」は、否定の「Не」が advmod で繋がれていて、禁止の命令を表しています。名詞 (呼格)「брате」が vocative で繋がれていて、さらに「любий」が amod で繋がれています。「втіху」と「ласку」は、いずれも「скупись」の斜格補語で、たがいに conj で繋がれていると同時に、前者が「скупись」に obl で繋がれています。前置詞「на」と代名詞 (与格)「їй」は、前者にそれぞれ case と nmod で繋がれており、接続詞「і」は後者に cc で繋がれています。

なお、ウクライナ語 UD の係り受けタグには、表 4 に加え、数多くの拡張が用いられている[81]のですが、拡張どうしの間で混乱が起こっており、事態の収拾が望まれます。

10.2　ウクライナ語ミニ RoBERTa モデルの製作

Google Colaboratory 上の esupar[37]で、ウクライナ語ミニ RoBERTa モデルを製作してみましょう。手順を図 78〜81 に示します。

最初に、訓練のための文章を準備しましょう (図 78)。train.txt の文章の材料として、ここでは UD_Ukrainian-IU[82]と Ukr-Synth[83]を元にしていますが、好みに応じ、ウクライナ語の他の文章を増量するのもいいでしょう。なお、UD_Ukrainian-IU・Ukr-Synth の各 CoNLL-U ファイルは、train.upos にまとめ上げて、品詞付与の学習に用います。また、UD_Ukrainian-IU の各 CoNLL-U ファイルは、係り受け解析の学習にも用います。

[81]https://universaldependencies.org/treebanks/uk_iu/#relations-overview
[82]https://github.com/UniversalDependencies/UD_Ukrainian-IU
[83]https://huggingface.co/datasets/ukr-models/Ukr-Synth

```
!pip install esupar pytokenizations
import os
url="https://github.com/UniversalDependencies/UD_Ukrainian-IU"
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
url="https://huggingface.co/datasets/ukr-models/Ukr-Synth"
d=os.path.basename(url)
!test -d {d} || git-lfs clone --depth=1 {url}
!zcat {d}/data/*.conllu.gz | tr -d "\015" | cat - *.conllu > train.upos
!sed -n "s/# text = //p" train.upos > train.txt
```

図 78: ウクライナ語ミニモデル製作のための準備

```
from transformers import BertTokenizerFast
from tokenizers import BertWordPieceTokenizer
s=["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"]
wpt=BertWordPieceTokenizer(strip_accents=False)
wpt.train(files=["train.txt"],vocab_size=30000,special_tokens=s)
wpt.save_model(".")
tkz=BertTokenizerFast(vocab_file="vocab.txt",strip_accents=False,never_split=s)
tkz.save_pretrained("my-dir/tokenizer-uk")
```

図 79: ウクライナ語向けトークナイザの作成

続けて、ウクライナ語向けトークナイザを、my-dir/tokenizer-uk に作成しましょう。図 79 では、Transformers の BertTokenizerFast を使って、ウクライナ語向けトークナイザを作成しています。

次に train.txt の文章から、RoBERTa モデルを my-dir/roberta-uk に作成しましょう。図 80 では、入出力幅 128 トークン・入出力ベクトル 256 次元・深さ 12 層・アテンションヘッド 4 個・中間ベクトル 768 次元という、やや小さめのモデルにしています。

最後に、my-dir/roberta-uk と UD_Ukrainian-IU・Ukr-Synth から、品詞付与・係り受け解析モデル my-dir/roberta-uk-upos を製作しましょう (図 81)。うまく my-dir/roberta-uk-upos が製作できたら、esupar でテストしてみましょう (図 82)。

```
from transformers import (AutoTokenizer,RobertaConfig,RobertaForMaskedLM,
  DataCollatorForLanguageModeling,TrainingArguments,Trainer)
tkz=AutoTokenizer.from_pretrained("my-dir/tokenizer-uk",model_max_length=128)
t=tkz.convert_tokens_to_ids(["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"])
cfg=RobertaConfig(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=RobertaForMaskedLM(cfg),train_dataset=ReadLineDS("train.txt",tkz))
trn.train()
trn.save_model("my-dir/roberta-uk")
tkz.save_pretrained("my-dir/roberta-uk")
```

図 80: ウクライナ語ミニ RoBERTa モデルの作成

```
s,d="my-dir/roberta-uk","my-dir/roberta-uk-upos"
!python -m esupar.train {s} {d} 32 /tmp train.upos
!python -m esupar.train {d} {d} 32 /// train.conllu dev.conllu test.conllu
```

図 81: ウクライナ語向け品詞付与・係り受け解析ミニモデルの製作

```
!pip install esupar
import esupar
nlp=esupar.load("my-dir/roberta-uk-upos")
doc=nlp("Не скупись на втіху їй і ласку любий брате")
import deplacy
deplacy.serve(doc,port=None)
```

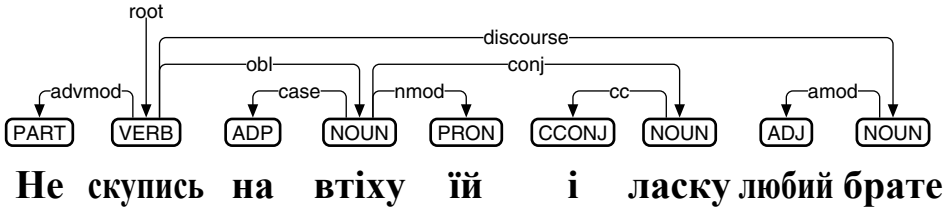


図 82: ウクライナ語向け品詞付与・係り受け解析ミニモデルのテスト

11　ベラルーシ語編

11.1　ベラルーシ語 UD における品詞付与と係り受け解析

Camphr-Udify[84]は、bert-base-multilingual-cased[85]をベースに、隣接行列型係り受け解析アルゴリズムを、75 の言語に対して[86]統合的に実装しています。Google Colaboratory 上の Camphr-Udify を用いて、ベラルーシ語の例文「Насамрэч інтэрнэт стварае міфы тых у чыіх руках інтэрнэт」を品詞付与・係り受け解析するプログラムと、その結果を図 83 に示します。解析結果を見ていくことにしましょう。

```
url="https://github.com/PKSHATechnology-Research/camphr_models"
f="en_udify-0.7.tar.gz"
!test -f {f} || curl -LO {url+"/releases/download/0.7.0/"+f}
!pip install camphr {f} deplacy
import pkg_resources,imp
imp.reload(pkg_resources)
import spacy
nlp=spacy.load("en_udify")
doc=nlp("Насамрэч інтэрнэт стварае міфы тых у чыіх руках інтэрнэт")
import deplacy
deplacy.serve(doc,port=None)
```

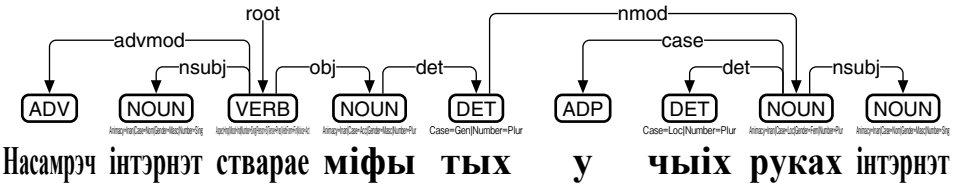


図 83: Camphr-Udify によるベラルーシ語の係り受け解析

2 つの「інтэрнэт」は、前者が「стварае」の主語、後者が「у чыіх руках інтэрнэт」というコピュラ節の主語であり、いずれも nsubj で繋がれています。コピュラ節「у чыіх руках інтэрнэт」の内部では、前置詞「у」が case で、関係代名詞「чыіх」が det で繋がれており、コピュラ節全体が「тых」を修飾しています。ベラルーシ語 UD のルール上、ここは acl:relcl で繋がれるべきなのですが、解析ミスで nmod になってしまっています。「тых」は「міфы」を修飾していて、det で繋がれています。「міфы」は「стварае」の目的語で、obj で繋がれています。「Насамрэч」は advmod で繋がれています。

なお、ベラルーシ語の係り受けタグには、表 4 に加え、受動態を表す nsubj:pass と aux:pass、受動態の動作主を表す obl:agent、関係代名詞による連体修飾節を表す acl:relcl、数詞と名詞の格変化の関係を表す nummod:gov と nummod:entity、コピュラ節に対する副詞的修飾語を表す advmod:discourse、氏名を表す flat:name、ベラルーシ語以外を表す flat:foreign も用いられます[87]。

[84]https://camphr.readthedocs.io/en/stable/notes/udify.html

[85]https://huggingface.co/bert-base-multilingual-cased

[86]Dan Kondratyuk and Milan Straka: 75 Languages, 1 Model: Parsing Universal Dependencies Universally, Proceedings of the 2019 Conference on Empirical Methods in Natural Language (November 2019), pp.2779-2795.

[87]https://universaldependencies.org/be/#relation-subtypes

11.2 ベラルーシ語ミニRoBERTaモデルの製作

Google Colaboratory 上の esupar[37]で、ベラルーシ語ミニ RoBERTa モデルを製作してみましょう。手順を図 84〜87 に示します。

最初に、訓練のための文章を準備しましょう (図 84)。train.txt の文章の材料として、ここでは UD_Belarusian-HSE[88]を元にしていますが、好みに応じ、ベラルーシ語の他の文章を増量するのもいいでしょう。なお、各 CoNLL-U ファイルは、品詞付与・係り受け解析の学習にも用います。

```
!pip install esupar
import os
url="https://github.com/UniversalDependencies/UD_Belarusian-HSE"
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
```

図 84: ベラルーシ語ミニモデル製作のための準備

続けて、ベラルーシ語向けトークナイザを、my-dir/tokenizer-be に作成しましょう。図 85 では、Transformers の BertTokenizerFast を使って、ベラルーシ語向けトークナイザを作成しています。語彙数 (vocab_size) を 8000 に絞っていますが、もう少し大きくした方がいいかもしれません。

```
from transformers import BertTokenizerFast
from tokenizers import BertWordPieceTokenizer
s=["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"]
wpt=BertWordPieceTokenizer(strip_accents=False)
wpt.train(files=["train.txt"],vocab_size=8000,special_tokens=s)
wpt.save_model(".")
tkz=BertTokenizerFast(vocab_file="vocab.txt",strip_accents=False,never_split=s)
tkz.save_pretrained("my-dir/tokenizer-be")
```

図 85: ベラルーシ語向けトークナイザの作成

次に train.txt の文章から、RoBERTa モデルを my-dir/roberta-be に作成しましょう。図 86 では、入出力幅 128 トークン・入出力ベクトル 256 次元・深さ 12 層・アテンションヘッド 4 個・中間ベクトル 768 次元という、やや小さめのモデルにしています。

最後に、my-dir/roberta-be と UD_Belarusian-HSE から、品詞付与・係り受け解析モデル my-dir/roberta-be-upos を製作しましょう (図 87)。うまく my-dir/roberta-be-upos が製作できたら、esupar でテストしてみましょう (図 88)。

[88] https://github.com/UniversalDependencies/UD_Belarusian-HSE

```
from transformers import (AutoTokenizer,RobertaConfig,RobertaForMaskedLM,
  DataCollatorForLanguageModeling,TrainingArguments,Trainer)
tkz=AutoTokenizer.from_pretrained("my-dir/tokenizer-be",model_max_length=128)
t=tkz.convert_tokens_to_ids(["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"])
cfg=RobertaConfig(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=RobertaForMaskedLM(cfg),train_dataset=ReadLineDS("train.txt",tkz))
trn.train()
trn.save_model("my-dir/roberta-be")
tkz.save_pretrained("my-dir/roberta-be")
```

図 86: ベラルーシ語ミニ RoBERTa モデルの作成

```
!python -m esupar.train my-dir/roberta-be my-dir/roberta-be-upos .
```

図 87: ベラルーシ語向け品詞付与・係り受け解析ミニモデルの製作

```
!pip install esupar
import esupar
nlp=esupar.load("my-dir/roberta-be-upos")
doc=nlp("Насамрэч інтэрнэт стварае міфы тых у чыіх руках інтэрнэт")
import deplacy
deplacy.serve(doc,port=None)
```

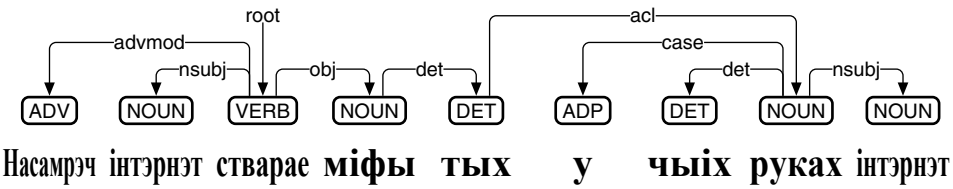


図 88: ベラルーシ語向け品詞付与・係り受け解析ミニモデルのテスト

12　セルビア語編

12.1　セルビア語 UD における品詞付与と係り受け解析

Camphr-Udify[84]は、bert-base-multilingual-cased をベースに、隣接行列型係り受け解析アルゴリズムを、75 の言語に対して統合的に実装しています。Google Colaboratory 上の Camphr-Udify を用いて、セルビア語の例文「Moja би мати знала гибати гибаницу да има сира и масла」を品詞付与・係り受け解析するプログラムと、その結果を図 89 に示します。解析結果を見ていくことにしましょう。

```
url="https://github.com/PKSHATechnology-Research/camphr_models"
f="en_udify-0.7.tar.gz"
!test -f {f} || curl -LO {url+"/releases/download/0.7.0/"+f}
!pip install camphr {f} deplacy
import pkg_resources,imp
imp.reload(pkg_resources)
from spacy.lang.sr import Serbian
nlp=Serbian()
import spacy
nlp.pipeline.extend(spacy.load("en_udify").pipeline)
doc=nlp("Moja би мати знала гибати гибаницу да има сира и масла")
import deplacy
deplacy.serve(doc,port=None)
```

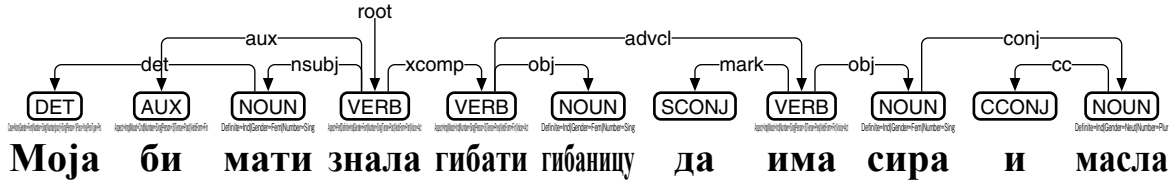


図 89: Camphr-Udify によるセルビア語 (キリル文字) の係り受け解析

図 89 の例文は、主節「Moja би мати знала гибати гибаницу」と条件節「да има сира и масла」から構成されており、advcl で繋がれています。動詞「знала」の主語「мати」は、同時に「гибати」の主語でもあることから、「мати」が nsubj で、「гибати」が xcomp で繋がれています。助動詞「би」は aux で繋がれています。所有代名詞「Moja」は「мати」を修飾しており、det のリンクが交差しています。「гибаницу」は「гибати」の目的語であり、obj で繋がれています。条件節では、「да」が mark で繋がれています。「има」の目的語は「сира」と「масла」の 2 つですが、「сира」の方が obj で繋がれていて、その先に conj で「масла」が繋がれており、そのまた先に cc で「и」が繋がれています。

セルビア語は、公式にはキリル文字で書かれるのですが、ラテン文字による表記も多く用いられており、UD_Serbian-SET[89]などのセルビア語コーパスも、多くはラテン文字で書かれています。Camphr-Udify は、ラテン文字で書かれたセルビア語も、キリル文字と同様に解析可能です。Google Colaboratory 上の Camphr-Udify を用いて、セルビア語の例文「Moja bi mati znala gibati gibanicu da ima sira i masla」を品詞付与・係り受け解析するプログラムと、その結果を図 90 に示します。

[89]https://github.com/UniversalDependencies/UD_Serbian-SET

```
url="https://github.com/PKSHATechnology-Research/camphr_models"
f="en_udify-0.7.tar.gz"
!test -f {f} || curl -LO {url+"/releases/download/0.7.0/"+f}
!pip install camphr {f} deplacy
import pkg_resources,imp
imp.reload(pkg_resources)
from spacy.lang.sr import Serbian
nlp=Serbian()
import spacy
nlp.pipeline.extend(spacy.load("en_udify").pipeline)
doc=nlp("Moja bi mati znala gibati gibanicu da ima sira i masla")
import deplacy
deplacy.serve(doc,port=None)
```

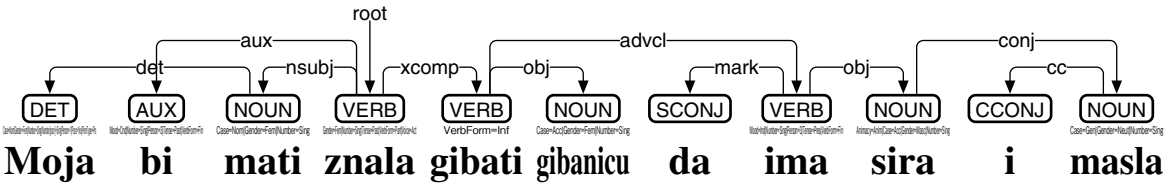


図 90: Camphr-Udify によるセルビア語 (ラテン文字) の係り受け解析

なお、セルビア語 UD の係り受けタグには、表 4 に加え、数詞と名詞 (および代名詞) の格変化の関係を表す nummod:gov と det:numgov も用いられます[90]。

12.2 セルビア語ミニ RoBERTa モデルの製作

Google Colaboratory 上の esupar[37]で、セルビア語ミニ RoBERTa モデルを製作してみましょう。手順を図 91〜95 に示します。srtools[91]の助けを借りて、キリル文字とラテン文字の両方が使えるモデルを目指します。

最初に、訓練のための文章を準備しましょう (図 91)。train.txt の文章の材料として、こ

```
!pip install esupar srtools
import os
url="https://www.clarin.si/repository/xmlui/bitstream/handle/11356/1063"
!test -f srWaC1.1.06.xml.gz || curl -LO {url}/srWaC1.1.06.xml.gz
s='{if($0~/^</){s="";if($1=="</s>")print""}else{printf("%s%s",s,$1);s=" "}}'
!zcat *.xml.gz | awk '{s}' | fgrep -v "&" > train.txt
url="https://github.com/UniversalDependencies/UD_Serbian-SET"
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
```

図 91: セルビア語ミニモデル製作のための準備

[90] https://universaldependencies.org/treebanks/sr_set/#relations-overview
[91] https://gitlab.com/andrejr/srtools

```
from transformers import BertTokenizerFast
from tokenizers import BertWordPieceTokenizer
from srtools import latin_to_cyrillic
s=["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"]
wpt=BertWordPieceTokenizer(strip_accents=False)
with open("train.txt","r",encoding="utf-8") as r:
  t=latin_to_cyrillic(r.read().strip()).split("\n")
wpt.train_from_iterator(iterator=t,vocab_size=8000,special_tokens=s)
wpt.save_model(".")
tkz=BertTokenizerFast(vocab_file="vocab.txt",strip_accents=False,never_split=s)
tkz.save_pretrained("my-dir/tokenizer-sr")
```

図 92: セルビア語向けトークナイザの作成

こでは srWaC[92]の一部と、UD_Serbian-SET を元にしていますが、好みに応じ、セルビア語の他の文章を増量するのもいいでしょう。なお、UD_Serbian-SET の各 CoNLL-U ファイルは、品詞付与・係り受け解析の学習にも用います。

続けて、train.txt の各単語をキリル文字に変換しつつ、セルビア語向けトークナイザを、my-dir/tokenizer-sr に作成しましょう。図 92 では、Transformers の BertTokenizerFast を使って、セルビア語向けトークナイザを作成しています。語彙数 (vocab_size) を 8000 に絞っていますが、もう少し大きくした方がいいかもしれません。

次に、train.txt の文章をキリル文字に変換しつつ、RoBERTa モデルを my-dir/roberta-sr に作成しましょう。図 93 では、入出力幅 128 トークン・入出力ベクトル 256 次元・深さ 12 層・アテンションヘッド 4 個・中間ベクトル 768 次元という、やや小さめのモデルにしています。さらに、キリル文字のトークンに対応するラテン文字のトークンを追加し、新たなモデルを my-dir/roberta-sr-ext に保存しましょう (図 94)。

最後に、my-dir/roberta-sr-ext と UD_Serbian-SET から、品詞付与・係り受け解析モデル my-dir/roberta-sr-upos を製作しましょう (図 95)。うまく my-dir/roberta-sr-upos が製作できたら、esupar でテストしてみましょう。キリル文字とラテン文字の両方が使えるはずです。ただ、図 96 の例では「гибати гибаницу」を固有名詞だと解釈してしまっており、図 89 とは異なる結果になっているようです。

[92] http://nlp.ffzg.hr/resources/corpora/srWaC

```
from transformers import (AutoTokenizer,RobertaConfig,RobertaForMaskedLM,
  DataCollatorForLanguageModeling,TrainingArguments,Trainer)
tkz=AutoTokenizer.from_pretrained("my-dir/tokenizer-sr",model_max_length=128)
t=tkz.convert_tokens_to_ids(["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"])
cfg=RobertaConfig(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 ReadLineCyrillic(object):
  def __init__(self,file,tokenizer):
    from srtools import latin_to_cyrillic
    self.tokenizer=tokenizer
    with open(file,"r",encoding="utf-8") as r:
      self.lines=[latin_to_cyrillic(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=RobertaForMaskedLM(cfg),train_dataset=ReadLineCyrillic("train.txt",tkz))
trn.train()
trn.save_model("my-dir/roberta-sr")
tkz.save_pretrained("my-dir/roberta-sr")
```

図 93: セルビア語ミニ RoBERTa モデルの作成

```
import torch,srtools
from transformers import BertTokenizerFast,AutoModelForMaskedLM
tkz=BertTokenizerFast.from_pretrained("my-dir/roberta-sr")
mdl=AutoModelForMaskedLM.from_pretrained("my-dir/roberta-sr")
n,s,m=len(tkz),tkz.all_special_tokens,tkz.model_max_length
c=[t if t in s else srtools.cyrillic_to_latin(t)
  for t in tkz.convert_ids_to_tokens(range(n))]
t=[False]*n+[True]*tkz.add_tokens(c)
e=mdl.resize_token_embeddings(len(tkz))
with torch.no_grad():
  for i,j in enumerate(tkz.convert_tokens_to_ids(c)):
    if t[j]:
      e.weight[j,:]=e.weight[i,:]
      t[j]=False
mdl.set_input_embeddings(e)
mdl.save_pretrained("my-dir/roberta-sr-ext")
with open("vocab.txt","w",encoding="utf-8") as w:
  print("\n".join(tkz.convert_ids_to_tokens(range(len(tkz)))),file=w)
tkz=BertTokenizerFast(vocab_file="vocab.txt",strip_accents=False,never_split=s,
  model_max_length=m)
tkz.save_pretrained("my-dir/roberta-sr-ext")
```

図 94: セルビア語ミニ RoBERTa モデルの拡張 (ラテン文字追加)

```
!python -m esupar.train my-dir/roberta-sr-ext my-dir/roberta-sr-upos .
```

図 95: セルビア語向け品詞付与・係り受け解析ミニモデルの製作

```
!pip install esupar
import esupar
nlp=esupar.load("my-dir/roberta-sr-upos")
doc=nlp("Moja би мати знала гибати гибаницу да има сира и масла")
import deplacy
deplacy.serve(doc,port=None)
```

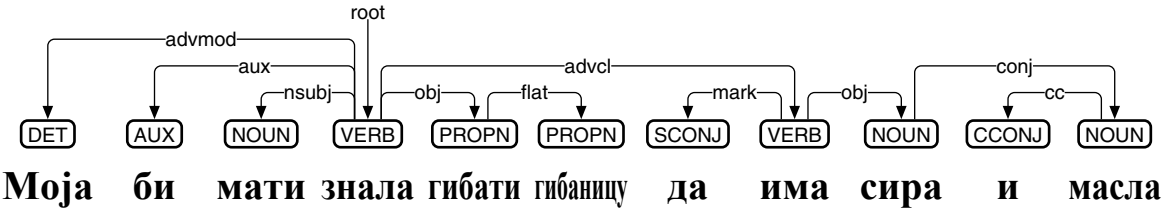


図 96: セルビア語向け品詞付与・係り受け解析ミニモデルのテスト

13　クロアチア語編

13.1　クロアチア語 UD における品詞付与と係り受け解析

Trankit[56]は、XLM-RoBERTa をベースに、隣接行列型係り受け解析アルゴリズムを、様々な言語向けに実装しています。Google Colaboratory 上の Trankit を用いて、クロアチア語の例文「Moja bi mati znala gibati gibanicu da ima sira i masla」を品詞付与・係り受け解析するプログラムと、その結果を図 97 に示します。解析結果を見ていくことにしましょう。

```
!pip install trankit transformers deplacy
import trankit
nlp=trankit.Pipeline("croatian")
doc=nlp("Moja bi mati znala gibati gibanicu da ima sira i masla")
import deplacy
deplacy.serve(doc,port=None)
```

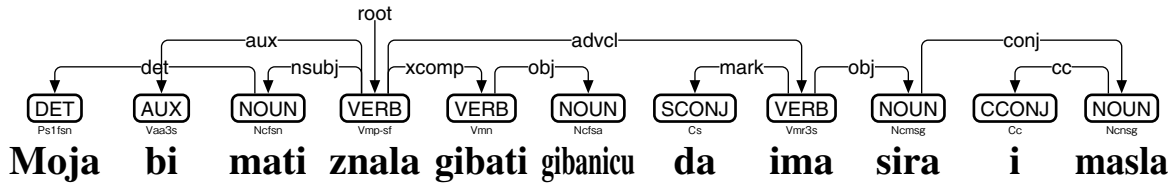


図 97: Trankit によるクロアチア語の係り受け解析

図 97 の例文は、主節「Moja bi mati znala gibati gibanicu」と条件節「da ima sira i masla」から構成されており、advcl で繋がれています。動詞「znala」の主語「mati」は、同時に「gibati」の主語でもあることから、「mati」が nsubj で、「gibati」が xcomp で繋がれています。助動詞「bi」は aux で繋がれています。所有代名詞「Moja」は「mati」を修飾しており、det のリンクが交差しています。「gibanicu」は「gibati」の目的語であり、obj で繋がれています。条件節では、「da」が mark で繋がれています。「ima」の目的語は「sira」と「masla」の 2 つですが、「sira」の方が obj で繋がれていて、その先に conj で「masla」が繋がれており、そのまた先に cc で「i」が繋がれています。

なお、クロアチア語 UD の係り受けタグには、表 4 に加え、強調の連用修飾語を表す advmod:emph、数詞と代名詞の格変化の関係を表す det:numgov、クロアチア語以外を表す flat:foreign も用いられます[93]。

13.2　クロアチア語ミニ RoBERTa モデルの製作

Google Colaboratory 上の esupar[37]で、クロアチア語ミニ RoBERTa モデルを製作してみましょう。手順を図 98〜101 に示します。

最初に、訓練のための文章を準備しましょう (図 98)。train.txt の文章の材料として、ここでは UD_Croatian-SET[94]と hr500k[95]を元にしていますが、好みに応じ、クロアチア語の他

[93]https://universaldependencies.org/treebanks/hr_set/#relations-overview
[94]https://github.com/UniversalDependencies/UD_Croatian-SET
[95]https://huggingface.co/datasets/classla/hr500k

```
!pip install esupar
import os
url="https://github.com/UniversalDependencies/UD_Croatian-SET"
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
url="https://huggingface.co/datasets/classla/hr500k"
d=os.path.basename(url)
!test -d {d} || ( git clone --depth=1 {url} ; cd {d} ; unzip data_ner.zip )
!cat {d}/data_ner/*.conllu *.conllu > train.upos
!sed -n "s/^# text = //p" train.upos > train.txt
```

図 98: クロアチア語ミニモデル製作のための準備

```
from transformers import BertTokenizerFast
from tokenizers import BertWordPieceTokenizer
s=["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"]
wpt=BertWordPieceTokenizer(lowercase=False,strip_accents=False)
wpt.train(files=["train.txt"],vocab_size=8000,special_tokens=s)
wpt.save_model(".")
tkz=BertTokenizerFast(vocab_file="vocab.txt",do_lowercase=False,
  strip_accents=False,never_split=s)
tkz.save_pretrained("my-dir/tokenizer-hr")
```

図 99: クロアチア語向けトークナイザの作成

の文章を増量するのもいいでしょう。なお、UD_Croatian-SET・hr500k の各 CoNLL-U ファイルは、train.upos にまとめ上げて、品詞付与の学習に用います。また、UD_Croatian-SET の各 CoNLL-U ファイルは、係り受け解析の学習にも用います。

続けて、クロアチア語向けトークナイザを、my-dir/tokenizer-hr に作成しましょう。図 99 では、Transformers の BertTokenizerFast を使って、クロアチア語向けトークナイザを作成しています。語彙数 (vocab_size) を 8000 に絞っていますが、もう少し大きくした方がいいかもしれません。

次に train.txt の文章から、RoBERTa モデルを my-dir/roberta-hr に作成しましょう。図 100 では、入出力幅 128 トークン・入出力ベクトル 256 次元・深さ 12 層・アテンションヘッド 4 個・中間ベクトル 768 次元という、やや小さめのモデルにしています。

最後に、my-dir/roberta-hr と UD_Croatian-SET・hr500k から、品詞付与・係り受け解析モデル my-dir/roberta-hr-upos を製作しましょう (図 101)。うまく my-dir/roberta-hr-upos が製作できたら、esupar でテストしてみましょう。ただ、図 102 の例では「gibati gibanicu」を固有名詞だと解釈してしまっており、図 97 とは異なる結果になっているようです。

```
from transformers import (AutoTokenizer,RobertaConfig,RobertaForMaskedLM,
  DataCollatorForLanguageModeling,TrainingArguments,Trainer)
tkz=AutoTokenizer.from_pretrained("my-dir/tokenizer-hr",model_max_length=128)
t=tkz.convert_tokens_to_ids(["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"])
cfg=RobertaConfig(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=RobertaForMaskedLM(cfg),train_dataset=ReadLineDS("train.txt",tkz))
trn.train()
trn.save_model("my-dir/roberta-hr")
tkz.save_pretrained("my-dir/roberta-hr")
```

図 100: クロアチア語ミニ RoBERTa モデルの作成

```
s,d="my-dir/roberta-hr","my-dir/roberta-hr-upos"
!python -m esupar.train {s} {d} 32 /tmp train.upos
!python -m esupar.train {d} {d} 32 /// train.conllu dev.conllu test.conllu
```

図 101: クロアチア語向け品詞付与・係り受け解析ミニモデルの製作

```
!pip install esupar
import esupar
nlp=esupar.load("my-dir/roberta-hr-upos")
doc=nlp("Moja bi mati znala gibati gibanicu da ima sira i masla")
import deplacy
deplacy.serve(doc,port=None)
```

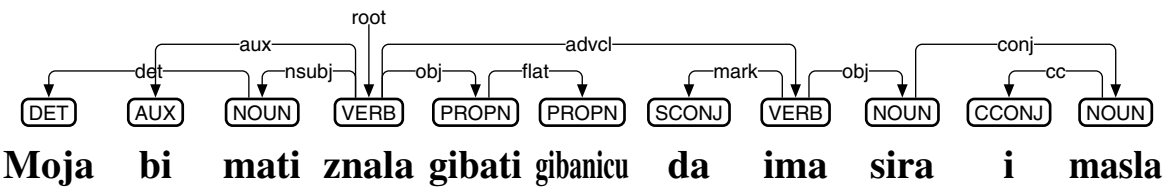


図 102: クロアチア語向け品詞付与・係り受け解析ミニモデルのテスト

14　コプト語編

14.1　コプト語 UD における品詞付与

書写言語としてのコプト語は、単語と単語の間に区切りがありません。空白や改行が入っている場合が多いのですが、単語の区切りとは限らないのです。事前学習モデルでコプト語を扱う際には、各トークンを単語と合致させるのが理想です。しかし、それは現実には困難なため、各トークンは単語より短めにトークナイズした上で、系列ラベリング (UPOS 付与) によって、単語をまとめ上げる手法が考えられます。すなわち、トークンが単語と合致した場合には UPOS をラベルとして付与し、そうでない場合には、最初のトークンに「B-」を付けた UPOS を、それ以降のトークンに「I-」を付けた UPOS を、それぞれ付与するのです。

```
!pip install deplacy transformers
class SeqL(object):
  def __init__(self,bert,aggr="none"):
    from transformers import pipeline
    self.pipeline=pipeline(task="ner",model=bert,aggregation_strategy=aggr)
    self.entity="entity" if aggr=="none" else "entity_group"
  def __call__(self,text):
    w=[(t["start"],t["end"],t[self.entity]) for t in self.pipeline(text)]
    u="# text = "+text.replace("\n"," ")+"\n"
    for i,(s,e,p) in enumerate(w,1):
      m="_" if i<len(w) and w[i][0]<e else "SpaceAfter=No"
      u+="\t".join([str(i),text[s:e],"_",p]+["_"]*5+[m])+"\n"
    return u+"\n"
nlp=SeqL("KoichiYasuoka/roberta-base-coptic-upos")
doc=nlp("ⲙⲟⲟⲩⲉⲁⲱⲥⲱⲛⲣⲉⲙⲡⲟⲩⲟⲉⲓⲛ")
import deplacy
deplacy.serve(doc,port=None)
```

図 103: roberta-base-coptic-upos による UPOS 付与

Google Colaboratory 上の Transformers を用いて、roberta-base-coptic-upos[96]でコプト語の例文「ⲙⲟⲟⲩⲉⲁⲱⲥⲱⲛⲣⲉⲙⲡⲟⲩⲟⲉⲓⲛ」に UPOS を付与するプログラムと、その結果を図 103 に示します。「ⲁⲱⲥ」「ⲱⲛⲣⲉ」「ⲙ」「ⲡ」「ⲟⲩⲟⲉⲓⲛ」の各単語では、各トークンがそのまま単語と合致しており、それぞれ UPOS 品詞 SCONJ・NOUN・ADP・DET・NOUN が付与されています。「ⲙⲟⲟⲩⲉ」は、3 つのトークンに泣き別れてしまっており、それぞれに B-VERB・I-VERB・I-VERB が付与されています。

[96] https://huggingface.co/KoichiYasuoka/roberta-base-coptic-upos

14.2 コプト語 UD における係り受け解析

esupar[37]は、隣接行列型の係り受け解析アルゴリズムを、RoBERTa などの事前学習モデルで実装しています。Google Colaboratory 上の esupar を用いて、roberta-base-coptic-upos で例文「ⲙⲟⲟⲩⲉⲁⲱⲥⲱⲛⲣⲉⲙⲡⲟⲩⲟⲉⲓⲛ」を係り受け解析するプログラムと、その結果を図 104 に示します。解析結果を見ていくことにしましょう。

```
!pip install esupar
import esupar
nlp=esupar.load("KoichiYasuoka/roberta-base-coptic-upos")
doc=nlp("ⲙⲟⲟⲩⲉⲁⲱⲥⲱⲛⲣⲉⲙⲡⲟⲩⲟⲉⲓⲛ")
import deplacy
deplacy.serve(doc,port=None)
```

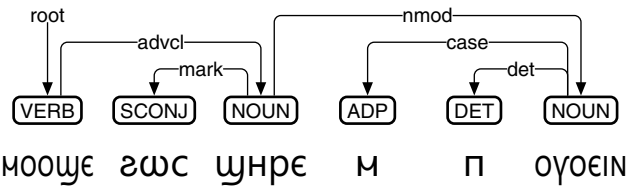


図 104: roberta-base-coptic-upos によるコプト語の係り受け解析

図 104 の例文は、主節「ⲙⲟⲟⲩⲉ」と従属節「ⲁⲱⲥⲱⲛⲣⲉⲙⲡⲟⲩⲟⲉⲓⲛ」から構成されており、advcl で繋がれています。主節は動詞「ⲙⲟⲟⲩⲉ」のみです。従属節はコピュラ文 (ただし主語も繋辞もない) で、「ⲁⲱⲥ」が mark で繋がれています。「ⲙⲡⲟⲩⲟⲉⲓⲛ」は「ⲱⲛⲣⲉ」を修飾していて、nmod で繋がれています。「ⲙⲡⲟⲩⲟⲉⲓⲛ」は、前置詞「ⲙ」と冠詞「ⲡ」と名詞「ⲟⲩⲟⲉⲓⲛ」から成っていて、それぞれ case と det で繋がれています。

なお、コプト語 UD の係り受けタグには、表 4 に加え、斜格補語による連用修飾を表す obl:npmod、関係転換詞による連体修飾節を表す acl:relcl も用いられます[97]。

14.3 コプト語ミニ DeBERTa モデルの製作

Google Colaboratory 上の esupar で、コプト語ミニ DeBERTa モデルを製作してみましょう。手順を図 105〜108 に示します。

最初に、訓練のための文章を準備しましょう (図 105)。train.txt の文章の材料として、ここでは Coptic Scriptorium Corpora[98]と UD_Coptic-Scriptorium[99]を元にしていますが、好みに応じ、コプト語の他の文章を増量するのもいいでしょう。train.txt と同時に、各単語を空白で区切った token.txt も準備しており、トークナイザの作成に用います。Coptic Scriptorium Corpora の各 CoNLL-U ファイルは、train.upos にまとめ上げて、品詞付与の学習に用います。また、UD_Coptic-Scriptorium の各 CoNLL-U ファイルは、係り受け解析の学習に用います。

続けて、token.txt の各単語から、コプト語向けトークナイザを、my-dir/tokenizer-cop に作成しましょう。図 106 では、Unigram トークナイザ (SentencePiece[77]の改造版) を

[97]https://universaldependencies.org/cop/dep
[98]http://data.copticscriptorium.org/index/corpus
[99]https://github.com/UniversalDependencies/UD_Coptic-Scriptorium

```
!pip install esupar pytokenizations
import os
url="https://github.com/UniversalDependencies/UD_Coptic-Scriptorium"
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
url="https://github.com/CopticScriptorium/corpora"
d=os.path.basename(url)
!test -d {d} || git clone --depth=1 {url}
s='''{if(NF==10&&$1~/^[1-9][0-9]*$/)printf($1>1?" %s":"%s",$2);if(NF==0){print;
  if(u!~/\\n[^#1-9]/)print u>"train.upos";u=""}else u=u$0"\\n"}'''
!nawk -F'\t' '{s}' {d}/*/*/*.conllu | tee token.txt > train.txt
!sed -n "s/^# text = //p" train.upos *.conllu >> train.txt
```

図 105: コプト語ミニモデル製作のための準備

```
from transformers import DebertaV2TokenizerFast
from tokenizers import (Tokenizer,models,pre_tokenizers,normalizers,processors,
  decoders,trainers)
s=["[CLS]","[PAD]","[SEP]","[UNK]","[MASK]"]
spt=Tokenizer(models.Unigram())
spt.pre_tokenizer=pre_tokenizers.Sequence([pre_tokenizers.Whitespace(),
  pre_tokenizers.Punctuation()])
spt.normalizer=normalizers.Sequence([normalizers.Nmt(),normalizers.NFD(),
  normalizers.StripAccents()])
spt.post_processor=processors.TemplateProcessing(single="[CLS] $A [SEP]",
  pair="[CLS] $A [SEP] $B:1 [SEP]:1",special_tokens=[("[CLS]",0),("[SEP]",2)])
spt.decoder=decoders.WordPiece(prefix="",cleanup=True)
spt.train(trainer=trainers.UnigramTrainer(vocab_size=3000,max_piece_length=8,
  special_tokens=s,unk_token="[UNK]",n_sub_iterations=2),files=["token.txt"])
spt.save("tokenizer.json")
tkz=DebertaV2TokenizerFast(tokenizer_file="tokenizer.json",split_by_punct=True,
  do_lower_case=False,keep_accents=False,vocab_file="/dev/null")
tkz.save_pretrained("my-dir/tokenizer-cop")
```

図 106: コプト語向けトークナイザの作成

token.txt に適用し、それを DebertaV2TokenizerFast という Transformers のトークナイザに組み上げて、コプト語向けトークナイザを作成しています。語彙数を 3000 に絞った上で、各トークンの文字数を最大 8 文字 (アクセント記号は除去) としています。

次に train.txt の文章から、DeBERTa モデルを my-dir/deberta-cop に作成しましょう。図 107 では、入出力幅 128 トークン・入出力ベクトル 256 次元・深さ 12 層・アテンションヘッド 4 個・中間ベクトル 768 次元という、やや小さめのモデルにしていますが、GPU でも 15 分程度かかってしまいます。

最後に、my-dir/deberta-cop と各 ConLL-U ファイルから、品詞付与・係り受け解析モデル my-dir/deberta-cop-upos を製作しましょう (図 108)。うまく my-dir/deberta-cop-upos ができたら、esupar でテストしてみましょう。ただ、図 109 の例では、「ⲱⲛⲣϵ」を「ⲙⲟⲟⲩϵ」の斜格補語 (obl) だと解釈してしまっており、図 104 とは異なる結果になっているようです。

```
from transformers import (AutoTokenizer,DebertaV2Config,DebertaV2ForMaskedLM,
  DataCollatorForLanguageModeling,TrainingArguments,Trainer)
tkz=AutoTokenizer.from_pretrained("my-dir/tokenizer-cop",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-cop")
tkz.save_pretrained("my-dir/deberta-cop")
```

図 107: コプト語ミニ DeBERTa モデルの作成

```
s,d="my-dir/deberta-cop","my-dir/deberta-cop-upos"
!python -m esupar.train {s} {d} 32 /tmp train.upos
!python -m esupar.train {d} {d} 32 /// train.conllu dev.conllu test.conllu
```

図 108: コプト語向け品詞付与・係り受け解析ミニモデルの製作

```
!pip install esupar
import esupar
nlp=esupar.load("my-dir/deberta-cop-upos")
doc=nlp("ⲙⲟⲟⲩⲉⲁⲱⲥⲩⲛⲣⲉⲙⲡⲟⲩⲟⲉⲓⲛ")
import deplacy
deplacy.serve(doc,port=None)
```

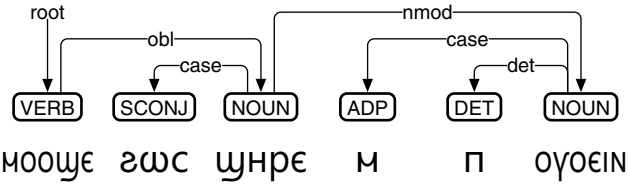


図 109: コプト語向け品詞付与・係り受け解析ミニモデルのテスト

Universal Dependencies と BERT/RoBERTa モデルによる多言語情報処理

安岡孝一

発行日　2022 年 2 月 28 日 (初版)・2022 年 5 月 27 日 (改訂版)

発行所　京都大学人文科学研究所・未踏科学研究ユニット・
データサイエンスで切り拓く総合地域研究ユニット