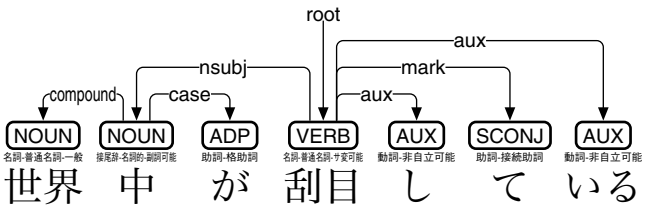
青空文庫DeBERTaモデルによる国語研長単位係り受け解析

安岡孝一*

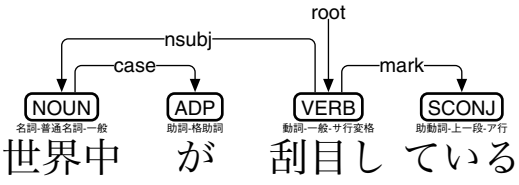
1 はじめに

2021年11月リリースのUniversal Dependencies (UD) 2.9において、国語研長単位コーパス[1]が追加された。UD 2.8.1以前の日本語係り受けコーパスは、いずれも国語研短単位を採用していた[2]が、UD 2.9以降は、国語研短単位UDと国語研長単位UDの2本建てとなったわけである。ただし、UD 2.9の国語研長単位UDは品詞(UPOS)に難があり、2022年5月リリースのUD 2.10で変更された(図1)。

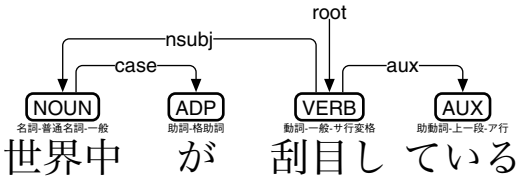
筆者が研究代表者を務める学際大規模情報基盤共同利用・共同研究拠点公募型共同研究『単語間に区切りのない書写言語における係り受け解析エンジンの開発』(共同研究者: 山崎直樹・二階堂善弘・師茂樹・鈴木慎吾・Christian Wittern・池田巧・守岡知彦・白須裕之・藤田一乗)では、多言語係り受け解析エンジンesupar [3]を開発中である。esuparはTransformers [4]上の言語モデルを用いて、系列ラベリングによる品詞付与と、隣接行列型



(a) 国語研短単位UD



(b) 国語研長単位UD 2.9



(c) 国語研長単位UD 2.10

図1: 国語研短単位UDと国語研長単位UD

*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]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.

[2]浅原正幸, 金山博, 宮尾祐介, 田中貴秋, 大村舞, 村脇有吾, 松本裕治: Universal Dependencies 日本語コーパス, 自然言語処理, Vol.26, No.1 (2019年3月), pp.3-36.

[3]`https://pypi.org/project/esupar`

[4]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.

アルゴリズム (Biaffine)[5]による係り受け解析をおこなう。

国語研長単位 UD に対して、われわれはこれまで、青空文庫 RoBERTa モデルによる係り受け解析[6]をおこなってきた。ただし、UD 2.10 での国語研長単位 UD 変更に伴い、これらの解析モデルも作り直す必要が生じた。時おりしも、Transformers が v4.19 において DeBERTa (V2)[7]をサポートしたところから、青空文庫 RoBERTa モデルに加え、青空文庫 DeBERTa モデルにも挑戦してみることにした。以下では、挑戦の概要について述べる。

2 Universal Dependencies の概要

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ある。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おり、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ある。

依存構造解析それ自体は、Tesnière の構造的統語論[8]に源を発し、Мельчук の有向グラフ記述[9]によって、一応の完成を見た手法である。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る。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する。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。

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[10]) が規定されている。CoNLL-U の各行は各単語に対応しており、表 1 に示す 10 個のタブ区切りフィールドで構成される。ID・FORM・LEMMA は、単語そのものに関するフィールドである。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ある。HEAD・DEPREL・DEPS は、単語の係り受けに関するフィールドである。

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 で記述する。HEAD は、その単語に入る有向枝のリンク元 ID を示しており、DEPREL は、その有向枝における係り受けタグである。ただし、HEAD が 0 の場合、その枝に入るリンク元は存在しない。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る。すなわち、各単語から出るリンクは複数の可能性があるが、各単語に入るリンクは 1 つだけである。なお、リンクはループしない。

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ある。動詞をリンク元として、主語や目的語へとリンクする。修飾関係においては、被修飾語から修飾語へとリンクする。ただし、側置詞 (前置詞や後置詞) を体言の修飾語だ

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

[6]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.

[7]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.

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

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

[10]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).

表 1: 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 2)。
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 3)。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 2: 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 3: 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

とみなす点[11]が、Мельчук とは異なっている。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する。

なお、UD は単語長を規定しておらず、各言語ごとに、自由に単語長を決めることができる。日本語に対しては、国語研短単位・国語研長単位のいずれを用いても、矛盾なく CoNLL-U を記述可能である。

3　トークナイザの設計と青空文庫 DeBERTa モデルの製作

青空文庫 RoBERTa モデルの製作[6]において、われわれが痛感したのは、トークナイザ設計の重要性だった。単語間に区切りのない書写言語においては、トークナイザをかなり自由に設計できるのだが、下手な設計をおこなうと解析精度が下がってしまう。

Transformers v4.19 は DeBERTa (V2) モデルに対し、SentencePiece[12]を基に Unigram トークナイザをサポートしている。Unigram トークナイザのパラメータとしては、語彙数 V (`vocab_size`) と、各トークンにおける最大の文字列長 M (`max_piece_length`) がある。これらのパラメータに関し、付録プログラム 1 と esupar 1.3.4 を用いて、UD 2.10 の国語研長単位 UD_Japanese-GSDLUW について予備実験をおこなった。

UD_Japanese-GSDLUW の ja_gsdluw-ud-train.conllu のみを用いて、日本語 DeBERTa モデルを作成し、評価・テストした結果を、表 4 に示す。評価指標は、CoNLL 2018 [13]の

表 4: プログラム 1 による DeBERTa (V2) トークナイザの比較 (UPOS / LAS / MLAS)

ja_gsdluw-ud-dev.conllu による評価 (evaluation)

	V=4000	**V=8000**	**V=16000**	**V=32000**
M=1	71.46 / 57.99 / 35.23	73.40 / 60.59 / 38.56	73.51 / 61.78 / 39.01	71.29 / 56.86 / 34.30
M=2	76.64 / 63.20 / 41.88	77.73 / 64.10 / 43.21	77.80 / 64.83 / 43.73	78.66 / 66.17 / 45.00
M=4	75.37 / 60.53 / 38.57	80.30 / 69.16 / 48.07	80.52 / 69.43 / 48.82	78.60 / 65.19 / 44.62
M=8	79.72 / 68.96 / 46.51	80.67 / 69.27 / 48.86	80.45 / 68.85 / 47.70	79.74 / 67.48 / 46.59
M=16	78.51 / 66.62 / 44.59	78.75 / 65.44 / 44.67	80.86 / 69.59 / 49.23	80.94 / 70.72 / 49.53

ja_gsdluw-ud-test.conllu によるテスト (predict)

	V=4000	**V=8000**	**V=16000**	**V=32000**
M=1	68.93 / 54.91 / 32.20	71.22 / 58.26 / 35.29	70.72 / 58.59 / 35.75	68.67 / 53.94 / 31.92
M=2	74.03 / 59.49 / 37.59	75.34 / 60.50 / 39.13	75.22 / 60.57 / 38.97	76.07 / 62.40 / 40.99
M=4	73.29 / 57.97 / 35.67	78.82 / 66.46 / 45.12	78.50 / 66.55 / 44.87	76.06 / 61.89 / 40.14
M=8	77.43 / 67.14 / 44.34	78.69 / 66.20 / 44.86	78.17 / 65.34 / 44.50	77.93 / 64.80 / 43.68
M=16	76.51 / 65.19 / 42.64	76.74 / 62.58 / 41.40	79.10 / 67.01 / 45.92	79.03 / 66.94 / 45.73

[11]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.

[12]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.

[13]Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre, and Slav Petrov: CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies, Proceedings of the CoNLL 2018 Shared Task (October 2018), pp.1-21.

表 5: 本稿で使用する DeBERTa / RoBERTa / BERT モデルとその諸元

① https://huggingface.co/KoichiYasuoka/deberta-base-japanese-aozora
青空文庫 3 億字 (元データ 2.37 億字＋異体字増量分 0.64 億字) を最大長 8 文字で Unigram トークナイズ、語彙 32000 トークン、入出力幅 512 トークン、入出力ベクトル 768 次元、深さ 12 層、アテンションヘッド 12 個、中間ベクトル 3072 次元、NVIDIA A100-SXM4-40GB での作成時間 19 時間 55 分。

② https://huggingface.co/KoichiYasuoka/deberta-large-japanese-aozora
青空文庫 3 億字 (元データ 2.37 億字＋異体字増量分 0.64 億字) を最大長 8 文字で Unigram トークナイズ、語彙 32000 トークン、入出力幅 512 トークン、入出力ベクトル 1024 次元、深さ 24 層、アテンションヘッド 16 個、中間ベクトル 4096 次元、NVIDIA A100-SXM4-40GB での作成時間 54 時間 28 分。

③ https://huggingface.co/KoichiYasuoka/deberta-base-japanese-wikipedia
青空文庫 3 億字 (元データ 2.37 億字＋異体字増量分 0.64 億字) と日本語 Wikipedia 13 億字を最大長 8 文字で Unigram トークナイズ、語彙 32000 トークン、入出力幅 512 トークン、入出力ベクトル 768 次元、深さ 12 層、アテンションヘッド 12 個、中間ベクトル 3072 次元、NVIDIA A100-SXM4-40GB での作成時間 87 時間 44 分。

④ https://huggingface.co/KoichiYasuoka/deberta-large-japanese-wikipedia
青空文庫 3 億字 (元データ 2.37 億字＋異体字増量分 0.64 億字) と日本語 Wikipedia 13 億字を最大長 8 文字で Unigram トークナイズ、語彙 32000 トークン、入出力幅 512 トークン、入出力ベクトル 1024 次元、深さ 24 層、アテンションヘッド 16 個、中間ベクトル 4096 次元、NVIDIA A100-SXM4-40GB での作成時間 254 時間 51 分。

Ⓐ https://huggingface.co/KoichiYasuoka/roberta-base-japanese-aozora-char
青空文庫 3 億字 (元データ 2.37 億字＋異体字増量分 0.64 億字) を単文字トークナイズ、語彙 9415 トークン、入出力幅 512 トークン、入出力ベクトル 768 次元、深さ 12 層、アテンションヘッド 12 個、中間ベクトル 3072 次元、NVIDIA A100-SXM4-40GB での作成時間 19 時間 11 分。

Ⓑ https://huggingface.co/KoichiYasuoka/roberta-large-japanese-aozora-char
青空文庫 3 億字 (元データ 2.37 億字＋異体字増量分 0.64 億字) を単文字トークナイズ、語彙 9415 トークン、入出力幅 512 トークン、入出力ベクトル 1024 次元、深さ 24 層、アテンションヘッド 16 個、中間ベクトル 4096 次元、NVIDIA A100-SXM4-40GB での作成時間 56 時間 57 分。

Ⓒ https://huggingface.co/cl-tohoku/bert-base-japanese-char-v2
日本語 Wikipedia 13 億字を単文字トークナイズ、語彙 6144 トークン、入出力幅 512 トークン、入出力ベクトル 768 次元、深さ 12 層、アテンションヘッド 12 個、中間ベクトル 3072 次元、v3-8 TPU での作成時間 5 日 (らしい)。

Ⓓ https://huggingface.co/cl-tohoku/bert-large-japanese-char
日本語 Wikipedia 13 億字を単文字トークナイズ、語彙 6144 トークン、入出力幅 512 トークン、入出力ベクトル 1024 次元、深さ 24 層、アテンションヘッド 16 個、中間ベクトル 4096 次元、v3-8 TPU での作成時間 14 日 (らしい)。

UPOS / LAS / MLAS [14]を用いた。表 4 を見る限り、語彙数 V は 16000 か 32000、最大長 M は 4 か 8 が良さそうである。

この結果に基づき、表 5 の青空文庫 DeBERTa モデル①〜④を製作した。モデル①の製作に用いたシェルスクリプトを、付録プログラム 2 に示す。また、比較のため、青空文庫 RoBERTa モデルⒶⒷと、東北大学の BERT モデルⒸⒹも使用した。

4　係り受け解析モデルによる評価

表 5 の 8 つのモデルに対し、esupar 1.3.4 と UD_Japanese-GSDLUW でファインチューニングをおこない、係り受け解析モデルを作成し、評価 (ja_gsdluw-ud-dev.conllu による evaluation)・テスト (ja_gsdluw-ud-test.conllu による predict) をおこなった。また、大学入学共通テストの令和 4 年度本試験『国語』(2022 年 1 月 15 日実施) 第 1 問の問題文を、手作業で UD 化 (図 2) し、これを用いて評価をおこなった。いずれも評価指標は、UPOS / LAS / MLAS を用いた。結果を表 6 に示すとともに、モデル①のファインチューニングと評価に用いたシェルスクリプトを、付録プログラム 3 に示す。

表 6: esupar を用いた係り受け解析モデルの評価 (UPOS / LAS / MLAS)

	評価 (evaluation)	テスト (predict)	第 1 問【文章 I】	第 1 問【文章 II】
①	96.19 / 91.68 / 82.09	96.05 / 90.33 / 81.00	92.09 / 81.18 / 60.96	96.32 / 87.09 / 71.03
②	96.42 / 91.89 / 82.97	96.30 / 90.86 / 82.16	93.08 / 82.70 / 64.02	96.69 / 87.34 / 72.04
③	95.38 / 90.19 / 80.43	94.99 / 89.16 / 79.39	90.40 / 77.36 / 59.29	95.26 / 84.77 / 68.49
④	97.17 / 92.38 / 84.58	97.07 / 91.29 / 83.58	93.31 / 82.95 / 63.43	96.18 / 85.96 / 70.57
Ⓐ	95.19 / 89.86 / 79.24	94.84 / 88.63 / 77.34	90.95 / 76.29 / 55.79	95.36 / 86.36 / 67.59
Ⓑ	95.22 / 90.21 / 78.83	95.12 / 88.81 / 76.90	91.13 / 77.68 / 59.09	94.43 / 82.86 / 64.25
Ⓒ	97.37 / 93.18 / 85.54	96.90 / 91.71 / 83.74	91.75 / 80.72 / 61.71	**97.18 / 88.34 / 72.54**
Ⓓ	**97.86 / 93.55 / 86.63**	**97.60 / 92.39 / 84.73**	**93.80 / 83.42 / 66.34**	96.40 / 86.02 / 71.82

さらに、Transformers の Question Answering による係り受け解析モデルを試作してみた。esupar を用いた係り受け解析モデルは、解析精度を高めるべく様々な工夫を凝らしているが、どうしてもファインチューニングのブラックボックス化が進んでしまうからだ。付録プログラム 4 に、モデル①のファインチューニングと評価に用いたシェルスクリプトを示す。プログラム 4 は、たとえば図 1(c) の UD の各リンクに対して

```
{ context="世界中が刮目している", question="世界中", answer="刮目し" }
{ context="世界中が刮目している", question="が", answer="世界中" }
{ context="世界中が刮目している", question="刮目し", answer="刮目し" }
{ context="世界中が刮目している", question="ている", answer="刮目し" }
```

のような形で Question Answering をおこなう。ただし、同一語が文中に複数あらわれる場合に備えて

[14]通常は LAS (Labeled Attachment Score) / MLAS (Morphology-aware Labeled Attachment Score) / BLEX (Bi-LEXical dependency score) の 3 つの評価指標を用いるが、esupar は LEMMA を使用していないため BLEX を外し、代わりに UPOS の F 値を用いた。

```
{ context="[MASK]が刮目している", question="世界中", answer="刮目し" }
{ context="世界中[MASK]刮目している", question="が", answer="世界中" }
{ context="世界中が[MASK]ている", question="刮目し", answer="[MASK]" }
{ context="世界中が刮目し[MASK]", question="ている", answer="刮目し" }
```

のような形での Question Answering も同時におこなう。係り受けタグ (DEPREL) は、品詞付与と同様に Transformers の Token Classification を用いて、「B-」「I-」を駆使[6]しつつ付与する。UPOS / LAS / MLAS での評価結果を、表 7 に示す。なお、評価時の隣接行列解法には、Chu-Liu-Edmonds [15]のカレル大学版[16]を流用している。

表 7: Question Answering による係り受け解析モデルの評価 (UPOS / LAS / MLAS)

	評価 (evaluation)	テスト (predict)	第 1 問【文章 I】	第 1 問【文章 II】
①	95.39 / 88.40 / 78.17	95.03 / 87.78 / 77.21	91.37 / 78.96 / 59.55	**95.76** / 84.57 / 69.42
②	95.13 / 88.99 / 78.47	95.14 / 87.68 / 77.26	90.80 / 77.17 / 56.56	94.93 / 82.23 / 67.34
③	95.05 / 87.74 / 77.75	95.17 / 87.01 / 76.92	89.00 / 76.37 / 56.81	94.29 / 83.43 / 67.12
④	95.75 / 88.87 / 79.89	95.52 / 88.21 / 79.26	90.92 / 76.97 / 56.44	94.49 / 82.89 / 66.33
Ⓐ	95.26 / 89.07 / 78.40	94.63 / 87.24 / 76.31	**91.68 / 80.40 / 60.94**	94.63 / 84.75 / 69.04
Ⓑ	92.38 / 82.11 / 66.82	91.89 / 79.53 / 64.27	87.14 / 67.40 / 45.08	91.50 / 72.95 / 53.82
Ⓒ	**96.79 / 91.69 / 83.59**	**96.46 / 89.58 / 80.98**	91.30 / 78.74 / 58.34	94.63 / 83.61 / 67.35
Ⓓ	96.72 / 91.35 / 83.11	96.23 / 89.45 / 80.60	90.37 / 76.16 / 57.84	94.91 / **85.45** / **69.62**

表 6・7 の結果による限り、Ⓓをファインチューニングした係り受け解析モデルが、全体に解析精度が高く、これにⒸと④が続いている。いずれも、日本語 Wikipedia 13 億字を学習したモデルであり、青空文庫 3 億字よりも良い、ということであろう。

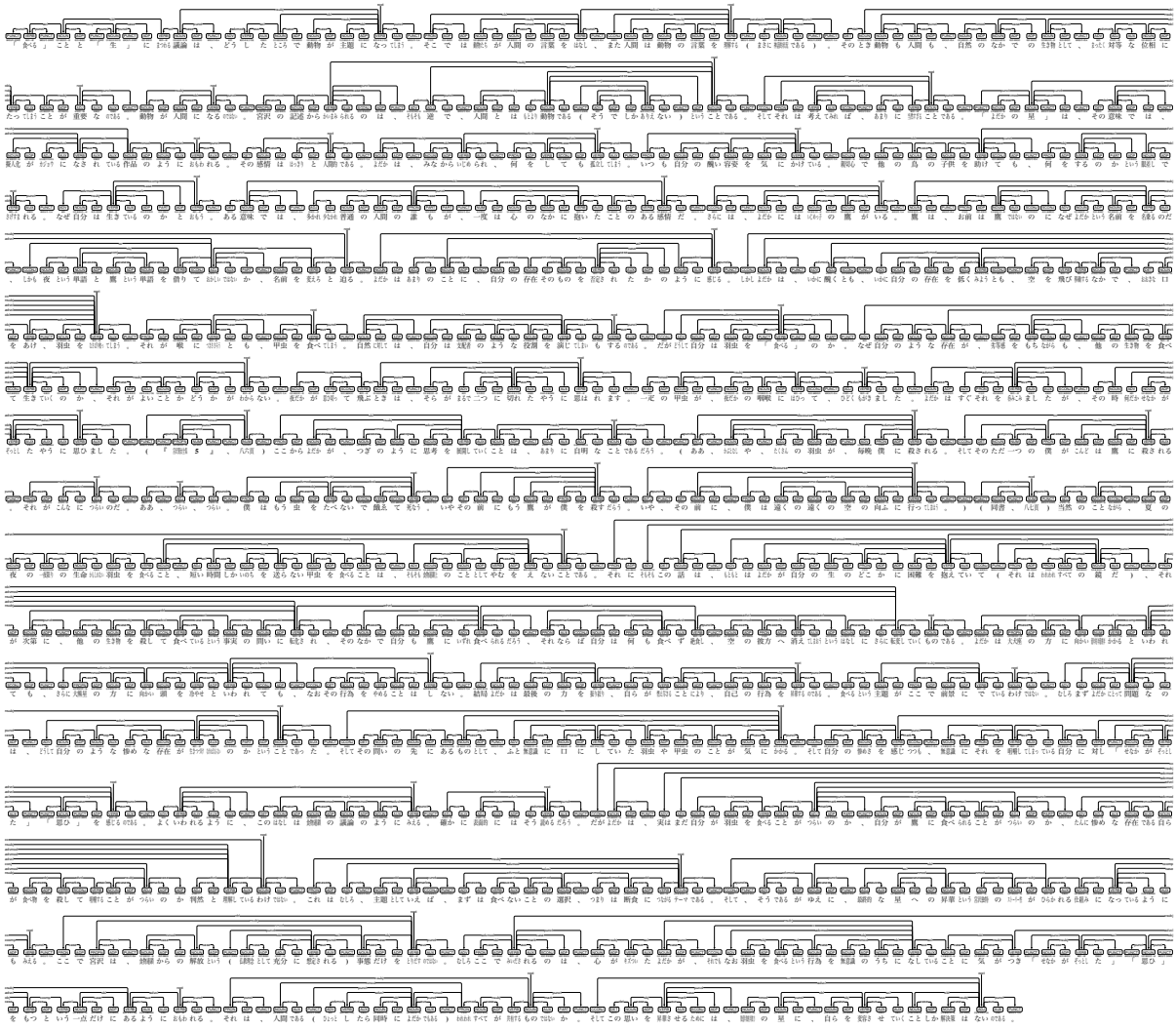
5 おわりに

青空文庫 DeBERTa モデルを製作し、国語研長単位係り受け解析モデルをファインチューニングする形で評価した。結果として、青空文庫 3 億字 (元データ 2.37 億字＋異体字増量分 0.64 億字) だけで製作したモデルより、日本語 Wikipedia 13 億字によるモデルの方が、係り受け解析の精度が高くなった。この傾向は、係り受け解析に esupar を使う場合も、Transformers の Question Answering を使う場合も同様であり、DeBERTa・RoBERTa・BERT などの事前学習モデルは、学習に用いた文章量に応じて「かしこくなる」ということだろう。

[15]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.

[16]`https://pypi.org/project/ufal.chu-liu-edmonds`

第 1 問【文章 I】



第 1 問【文章 II】

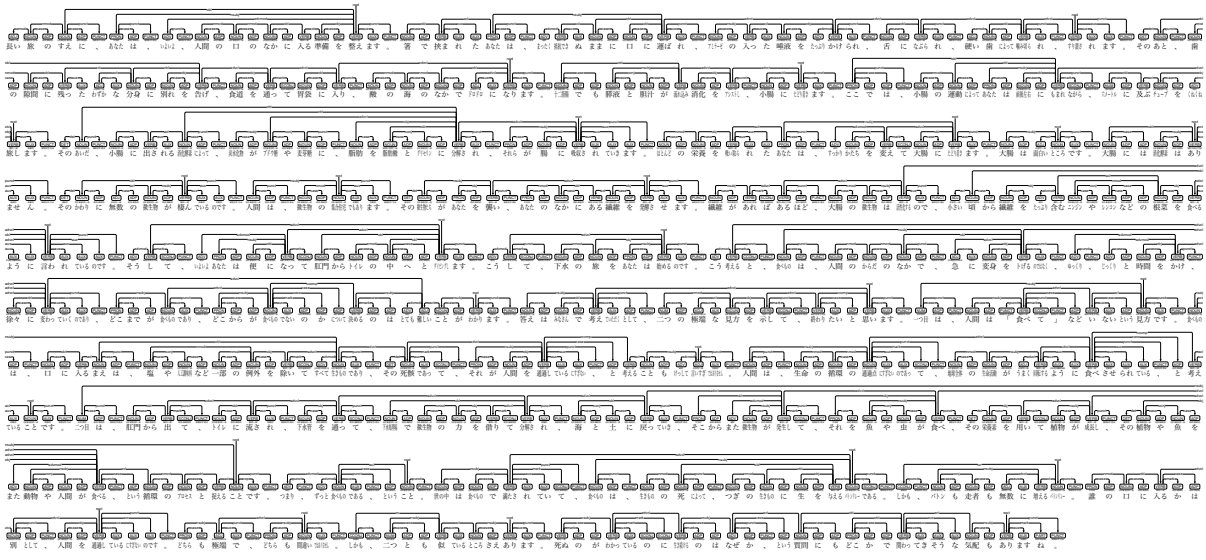


図 2: 共通テスト『国語』(2022 年 1 月 15 日実施) 第 1 問を基にした国語研長単位 UD

付録　プログラムリスト

本稿で使用したシェルスクリプトのうち、主なものを以下に示す。なお、各プログラムは、データ活用社会創生プラットフォーム mdx 上で 1GPU (NVIDIA A100-SXM4-40GB) を想定しているが、他の環境でも動作するよう書いたつもりである。

プログラム 1: DeBERTa (V2) トークナイザの比較

```
#! /bin/sh
URL=https://github.com/UniversalDependencies/UD_Japanese-GSDLUW
D=‘basename $URL‘
test -d $D || git clone --depth=1 $URL
for F in train dev test
do cat $D/*-$F*.conllu | tee $F.conllu | sed -n ’s/^# text = //p’ > $F.txt
done
S=’{if(NF==10&&$1~/^[1-9][0-9]*$/)printf($1>1?" %s":"%s",$2);if(NF==0)print}’
nawk -F’\t’ "$S" train.conllu > token.txt
URL=http://universaldependencies.org/conll18/conll18_ud_eval.py
C=‘basename $URL‘
test -f $C || curl -LO $URL
for M in 1 2 4 8 16
do for V in 4000 8000 16000 32000
   do test -d deberta$M-$V || python3 -c m,v=$M,$V’
from transformers import (DataCollatorForLanguageModeling,TrainingArguments,
  DebertaV2TokenizerFast,DebertaV2Config,DebertaV2ForMaskedLM,Trainer)
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=v,max_piece_length=m,
  special_tokens=s,unk_token="[UNK]",n_sub_iterations=2),files=["token.txt"])
spt.save("tokenizer.json")
tkz=DebertaV2TokenizerFast(tokenizer_file="tokenizer.json",
  do_lower_case=False,keep_accents=True,bos_token="[CLS]",cls_token="[CLS]",
  pad_token="[PAD]",sep_token="[SEP]",unk_token="[UNK]",mask_token="[MASK]",
  vocab_file="/dev/null",model_max_length=512,split_by_punct=True)
t=tkz.convert_tokens_to_ids(s)
cfg=DebertaV2Config(hidden_size=768,num_hidden_layers=12,num_attention_heads=12,
  intermediate_size=3072,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=8,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("deberta{}-{}".format(m,v))
tkz.save_pretrained("deberta{}-{}".format(m,v))'
      test -d esupar$M-$V || python3 -m esupar.train deberta$M-$V esupar$M-$V .
      test -f result$M-$V/result && continue
      mkdir -p result$M-$V
      for F in dev test
      do cat $F.txt | python3 -c '
import esupar
nlp=esupar.load("'esupar$M-$V'")
with open("'result$M-$V/$F.conllu'","w",encoding="utf-8") as w:
  while True:
    try:
      doc=nlp(input().strip())
    except:
      quit()
    print(doc,file=w)'
      done
      ( echo '***' esupar$M-$V dev
        python3 $C -v dev.conllu result$M-$V/dev.conllu
        echo '***' esupar$M-$V test
        python3 $C -v test.conllu result$M-$V/test.conllu
      ) | tee result$M-$V/result
   done
done
```

プログラム 2: 青空文庫から deberta-base-japanese-aozora を製作

```
#! /bin/sh
URL=https://github.com/aozorabunko/aozorabunko
D=`basename $URL`
test -d $D || git clone --depth=1 $URL
( cd $D && find gaiji -name '*.png' ) | sort | LANG=C awk '
BEGIN{
  printf("/<!DOCTYPE[^<]*$/d\n");
  printf("/^[^>]*dtd.>/d\n");
  printf("/<div[^>]*bibliographical_information.>/,$d\n");
}
{
  h=substr($1,length($1)-8,2)
  l=substr($1,length($1)-5,2)
  if($1~/1-[0-9][0-9]\/1-[0-9][0-9]-[0-9][0-9]\.png$/)
    printf("s?<img [^>]*%s[^>]*>?%c%c?g\n",$1,h+160,l+160);
  else if($1~/2-[0-9][0-9]\/2-[0-9][0-9]-[0-9][0-9]\.png$/){
    if(h!=94||l-87<0)
      printf("s?<img [^>]*%s[^>]*>?%c%c%c?g\n",$1,143,h+160,l+160);
  }
}
END{
  printf("s?<img [^>]*>?[UNK]?g\n");
  printf("s?<r[tp]>[^<]*</r[tp]>??g\n");
  printf("s/<[^>]*>//g\n");
  c=161;
  printf("s/%c%c[%c%c%c%c%c%c%c%c]*/&%c/g\n",c,163,c,203,c,205,c,215,c,217,13);
}' | iconv -f EUC-JISX0213 -t UTF-8 > aozora.sed
```

```
egrep -l '(encoding|charset)="?Shift_JIS' $D/cards/*/files/*_*.html |
( while read F
  do tr '\015' '\012' < $F | iconv -f CP932 -t UTF-8 | sed -f aozora.sed
  done
) | tr '\015' '\012' | egrep -v '^[[:blank:]]*$' > aozora.txt
python3 -c '
aug=lambda x:(x.replace("俠","侠").replace("倶","倶").replace("洗","冼")
  .replace("剥","剝").replace("即","卽").replace("呑","吞").replace("呉","吳")
  .replace("填","塡").replace("巣","巢").replace("徴","徵").replace("徳","德")
  .replace("掲","揭").replace("撃","擊").replace("教","敎").replace("晩","晚")
  .replace("横","橫").replace("歩","步").replace("歴","歷").replace("毎","每")
  .replace("冷","泠").replace("渉","涉").replace("涙","淚").replace("清","淸")
  .replace("渇","渴").replace("温","溫").replace("状","狀").replace("産","產")
  .replace("痩","瘦").replace("禰","祢").replace("箪","簞").replace("緑","綠")
  .replace("緒","緖").replace("縁","緣").replace("繋","繫").replace("莱","萊")
  .replace("薫","薰").replace("虚","虛").replace("蝉","蟬").replace("説","說")
  .replace("躯","軀").replace("郎","郞").replace("醤","醬").replace("録","錄")
  .replace("錬","鍊").replace("間","閒").replace("頬","頰").replace("顛","顚")
  .replace("鴎","鷗").replace("麺","麵").replace("黄","黃").replace("黒","黑")
  .replace("叱","吒"))
with open("aozora.txt","r",encoding="utf-8") as r:
  with open("aozora-aug.txt","w",encoding="utf-8") as w:
    for s in r:
      s=s.strip()
      if s!="":
        t=aug(s)
        if t!=s:
          print(t,file=w)'
cat aozora.txt aozora-aug.txt | tee train.txt | fugashi -Owakati > token.txt
python3 -c '
from transformers import (DataCollatorForLanguageModeling,TrainingArguments,
  DebertaV2TokenizerFast,DebertaV2Config,DebertaV2ForMaskedLM,Trainer)
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=32000,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",
  do_lower_case=False,keep_accents=True,bos_token="[CLS]",cls_token="[CLS]",
  pad_token="[PAD]",sep_token="[SEP]",unk_token="[UNK]",mask_token="[MASK]",
  vocab_file="/dev/null",model_max_length=512,split_by_punct=True)
t=tkz.convert_tokens_to_ids(["[PAD]","[CLS]","[SEP]"])
cfg=DebertaV2Config(hidden_size=768,num_hidden_layers=12,num_attention_heads=12,
  intermediate_size=3072,max_position_embeddings=tkz.model_max_length,
  tokenizer_class=type(tkz).__name__,vocab_size=len(tkz),
  pad_token_id=t[0],bos_token_id=t[1],eos_token_id=t[2])
arg=TrainingArguments(num_train_epochs=3,per_device_train_batch_size=32,
  warmup_steps=10000,learning_rate=2e-04,weight_decay=0.01,adam_beta1=0.9,
  adam_beta2=0.98,adam_epsilon=1e-04,seed=1,save_steps=5000,
  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("KoichiYasuoka/deberta-base-japanese-aozora")
tkz.save_pretrained("KoichiYasuoka/deberta-base-japanese-aozora")'
```

プログラム 3: esupar を用いた係り受け解析モデル作成ならびに評価

```
#! /bin/sh
S=KoichiYasuoka/deberta-base-japanese-aozora
T=KoichiYasuoka/deberta-base-japanese-luw-upos
URL=https://github.com/UniversalDependencies/UD_Japanese-GSDLUW
D=`basename $URL`
test -d $D || git clone --depth=1 $URL
for F in train dev test ; do cat $D/*-$F*.conllu > $F.conllu ; done
python3 -m esupar.train $S $T $D
URL=https://github.com/KoichiYasuoka/deplacy
D=`basename $URL`
test -d $D || git clone --depth=1 $URL
cp $D/demo/*/UD-KyotsuTest2022Kokugo/question*.conllu .
URL=http://universaldependencies.org/conll18/conll18_ud_eval.py
C=`basename $URL`
test -f $C || curl -LO $URL
mkdir -p result/$T
for F in dev test question1-1 question1-2
do python3 -c '
import esupar
nlp=esupar.load("'$T'")
with open("'$F'.conllu","r",encoding="utf-8") as r:
  with open("'result/$T/$F.conllu'","w",encoding="utf-8") as w:
    for s in r.read().split("\n"):
      if s.startswith("# text = "):
        print(s,file=w)
        print(nlp(s[9:]),file=w)'
done
( for F in dev test question1-1 question1-2
  do echo '***' $T $F
     python3 $C -v $F.conllu result/$T/$F.conllu
  done ) | tee result/$T/result
```

プログラム 4: Question Answering による係り受け解析モデル作成ならびに評価

```
#! /bin/sh
S=KoichiYasuoka/deberta-base-japanese-aozora
T=KoichiYasuoka/deberta-base-japanese-aozora-ud-head
```

```
URL=https://github.com/UniversalDependencies/UD_Japanese-GSDLUW
D=‘basename $URL‘
test -d $D || git clone --depth=1 $URL
for F in train dev test ; do cat $D/*-$F*.conllu > $F.conllu ; done
python3 -c 'src,tgt="'$S'","'$T'"
from transformers import (AutoTokenizer,AutoModelForQuestionAnswering,
  AutoModelForTokenClassification,AutoConfig,DefaultDataCollator,
  DataCollatorForTokenClassification,TrainingArguments,Trainer)
class HEADDataset(object):
  def __init__(self,conllu,tokenizer,augment=False,length=384):
    self.qa,self.pad,self.length=[],tokenizer.pad_token_id,length
    with open(conllu,"r",encoding="utf-8") as r:
      form,head=[],[]
      for t in r:
        w=t.split("\t")
        if len(w)==10 and w[0].isdecimal():
          form.append(w[1])
          head.append(len(head) if w[6]=="0" else int(w[6])-1)
        elif t.strip()=="" and form!=[]:
          v=tokenizer(form,add_special_tokens=False)["input_ids"]
          for i,t in enumerate(v):
            q=[tokenizer.cls_token_id]+t+[tokenizer.sep_token_id]
            c=[q]+v[0:i]+[[tokenizer.mask_token_id]]+v[i+1:]+[[q[-1]]]
            b=[len(sum(c[0:j+1],[])) for j in range(len(c))]
            if b[-1]<length:
              self.qa.append((sum(c,[]),head[i],b))
            if augment and [1 for x in v if t==x]==[1]:
              c[i+1]=t
              b=[len(sum(c[0:j+1],[])) for j in range(len(c))]
              if b[-1]<length:
                self.qa.append((sum(c,[]),head[i],b))
          form,head=[],[]
  __len__=lambda self:len(self.qa)
  def __getitem__(self,i):
    (v,h,b),k=self.qa[i],self.length-self.qa[i][2][-1]
    return {"input_ids":v+[self.pad]*k,"attention_mask":[1]*b[-1]+[0]*k,
      "token_type_ids":[0]*b[0]+[1]*(b[-1]-b[0])+[0]*k,
      "start_positions":b[h],"end_positions":b[h+1]-1}
class UPOSDataset(object):
  def __init__(self,conllu,tokenizer,fields=[3]):
    self.ids,self.upos=[],[]
    label,cls,sep=set(),tokenizer.cls_token_id,tokenizer.sep_token_id
    with open(conllu,"r",encoding="utf-8") as r:
      form,upos=[],[]
      for t in r:
        w=t.split("\t")
        if len(w)==10 and w[0].isdecimal():
          form.append(w[1])
          upos.append("|".join(w[i] for i in fields))
        elif t.strip()=="" and form!=[]:
          v,u=tokenizer(form,add_special_tokens=False)["input_ids"],[]
          for x,y in zip(v,upos):
            u.extend(["B-"+y]*min(len(x),1)+["I-"+y]*(len(x)-1))
          if len(u)>tokenizer.model_max_length-4:
            self.ids.append(sum(v,[])[0:tokenizer.model_max_length-2])
            self.upos.append(u[0:tokenizer.model_max_length-2])
          elif len(u)>0:
            self.ids.append([cls]+sum(v,[])+[sep])
            self.upos.append([u[0]]+u+[u[0]])
```

```
          label=set(sum([self.upos[-1],list(label)],[]))
          form,upos=[],[]
    self.label2id={l:i for i,l in enumerate(sorted(label))}
  def __call__(*args):
    label=set(sum([list(t.label2id) for t in args],[]))
    lid={l:i for i,l in enumerate(sorted(label))}
    for t in args:
      t.label2id=lid
    return lid
  __len__=lambda self:len(self.ids)
  __getitem__=lambda self,i:{"input_ids":self.ids[i],
    "labels":[self.label2id[t] for t in self.upos[i]]}
tkz=AutoTokenizer.from_pretrained(src)
trainDS=HEADDataset("train.conllu",tkz,True)
devDS=HEADDataset("dev.conllu",tkz)
arg=TrainingArguments(num_train_epochs=3,per_device_train_batch_size=32,
  output_dir="/tmp",overwrite_output_dir=True,save_total_limit=2,
  evaluation_strategy="epoch",learning_rate=5e-05,warmup_ratio=0.1)
trn=Trainer(args=arg,data_collator=DefaultDataCollator(),
  model=AutoModelForQuestionAnswering.from_pretrained(src),
  train_dataset=trainDS,eval_dataset=devDS)
trn.train()
trn.save_model(tgt)
tkz.save_pretrained(tgt)
trainDS=UPOSDataset("train.conllu",tkz,[7])
devDS=UPOSDataset("dev.conllu",tkz,[7])
testDS=UPOSDataset("test.conllu",tkz,[7])
lid=trainDS(devDS,testDS)
cfg=AutoConfig.from_pretrained(src,num_labels=len(lid),label2id=lid,
  id2label={i:l for l,i in lid.items()})
trn=Trainer(args=arg,data_collator=DataCollatorForTokenClassification(tkz),
  model=AutoModelForTokenClassification.from_pretrained(src,config=cfg),
  train_dataset=trainDS,eval_dataset=devDS)
trn.train()
trn.save_model(tgt+"/deprel")
tkz.save_pretrained(tgt+"/deprel")
trainDS=UPOSDataset("train.conllu",tkz,[3,5])
devDS=UPOSDataset("dev.conllu",tkz,[3,5])
testDS=UPOSDataset("test.conllu",tkz,[3,5])
lid=trainDS(devDS,testDS)
cfg=AutoConfig.from_pretrained(src,num_labels=len(lid),label2id=lid,
  id2label={i:l for l,i in lid.items()})
trn=Trainer(args=arg,data_collator=DataCollatorForTokenClassification(tkz),
  model=AutoModelForTokenClassification.from_pretrained(src,config=cfg),
  train_dataset=trainDS,eval_dataset=devDS)
trn.train()
trn.save_model(tgt+"/tagger")
tkz.save_pretrained(tgt+"/tagger")'
URL=https://github.com/KoichiYasuoka/deplacy
D=`basename $URL`
test -d $D || git clone --depth=1 $URL
cp $D/demo/*/UD-KyotsuTest2022Kokugo/question*.conllu .
URL=http://universaldependencies.org/conll18/conll18_ud_eval.py
C=`basename $URL`
test -f $C || curl -LO $URL
mkdir -p result/$T
for F in dev test question1-1 question1-2
do python3 -c '
```

```
class TransformersUD(object):
  def __init__(self,bert):
    import os
    from transformers import (AutoTokenizer,AutoModelForQuestionAnswering,
      AutoModelForTokenClassification,AutoConfig,TokenClassificationPipeline)
    self.tokenizer=AutoTokenizer.from_pretrained(bert)
    self.model=AutoModelForQuestionAnswering.from_pretrained(bert)
    x=AutoModelForTokenClassification.from_pretrained
    if os.path.isdir(bert):
      d,t=x(os.path.join(bert,"deprel")),x(os.path.join(bert,"tagger"))
    else:
      from transformers.file_utils import hf_bucket_url
      c=AutoConfig.from_pretrained(hf_bucket_url(bert,"deprel/config.json"))
      d=x(hf_bucket_url(bert,"deprel/pytorch_model.bin"),config=c)
      s=AutoConfig.from_pretrained(hf_bucket_url(bert,"tagger/config.json"))
      t=x(hf_bucket_url(bert,"tagger/pytorch_model.bin"),config=s)
    self.deprel=TokenClassificationPipeline(model=d,tokenizer=self.tokenizer,
      aggregation_strategy="simple")
    self.tagger=TokenClassificationPipeline(model=t,tokenizer=self.tokenizer)
  def __call__(self,text):
    import numpy,torch,ufal.chu_liu_edmonds
    w=[(t["start"],t["end"],t["entity_group"]) for t in self.deprel(text)]
    z,n={t["start"]:t["entity"].split("|") for t in self.tagger(text)},len(w)
    r,m=[text[s:e] for s,e,p in w],numpy.full((n+1,n+1),numpy.nan)
    v,c=self.tokenizer(r,add_special_tokens=False)["input_ids"],[]
    for i,t in enumerate(v):
      q=[self.tokenizer.cls_token_id]+t+[self.tokenizer.sep_token_id]
      c.append([q]+v[0:i]+[[self.tokenizer.mask_token_id]]+v[i+1:]+[[q[-1]]])
    b=[[len(sum(x[0:j+1],[])) for j in range(len(x))] for x in c]
    d=self.model(input_ids=torch.tensor([sum(x,[]) for x in c]),
      token_type_ids=torch.tensor([[0]*x[0]+[1]*(x[-1]-x[0]) for x in b]))
    s,e=d.start_logits.tolist(),d.end_logits.tolist()
    for i in range(n):
      for j in range(n):
        m[i+1,0 if i==j else j+1]=s[i][b[i][j]]+e[i][b[i][j+1]-1]
    h=ufal.chu_liu_edmonds.chu_liu_edmonds(m)[0]
    if [0 for i in h if i==0]!=[0]:
      i=([p for s,e,p in w]+["root"]).index("root")
      j=i+1 if i<n else numpy.nanargmax(m[:,0])
      m[0:j,0]=m[j+1:,0]=numpy.nan
      h=ufal.chu_liu_edmonds.chu_liu_edmonds(m)[0]
    u="# text = "+text.replace("\n"," ")+"\n"
    for i,(s,e,p) in enumerate(w,1):
      p="root" if h[i]==0 else "dep" if p=="root" else p
      u+="\t".join([str(i),r[i-1],"_",z[s][0][2:],"_","|".join(z[s][1:]),
        str(h[i]),p,"_","_" if i<n and w[i][0]<e else "SpaceAfter=No"])+"\n"
    return u+"\n"
nlp=TransformersUD("'$T'")
with open("'$F'.conllu","r",encoding="utf-8") as r:
  with open("'result/$T/$F.conllu'","w",encoding="utf-8") as w:
    for s in r.read().split("\n"):
      if s.startswith("# text = "):
        w.write(nlp(s[9:]))'
done
( for F in dev test question1-1 question1-2
  do echo '***' $T $F
     python3 $C -v $F.conllu result/$T/$F.conllu
  done ) | tee result/$T/result
```