日本語 Tokenizer の違いは下流タスク性能に影響を与えるか？

藤井 巧朗 [1*] 柴田 幸輝 [2*] 山口 篤季 [3] 十河 泰弘 [3]
[1] 横浜国立大学 [2] 筑波大学 [3] 株式会社日立製作所 研究開発グループ
[1]tkr.fujii.ynu@gmail.com [2]s1811496@klis.tsukuba.ac.jp
[3]{atsuki.yamaguchi.xn, yasuhiro.sogawa.tp}@hitachi.com

概要

本研究では、形態素解析器やサブワード分割手法の違いが大規模言語モデルに与える影響を調査した。種々の形態素解析器とサブワード分割手法を考慮した全 18 種のトークナイザについて、トークナイザの学習と BERT モデルの事前学習、7 個の下流タスクでのファインチューニングを行い、下流タスクの性能を比較した。また、形態素解析器の有無やその違い、サブワード分割手法の違い、単語分かち書き辞書の違い、トークナイザ辞書の類似性の観点から各種トークナイザの与える影響を評価した。

1 はじめに

近年、BERT [1] をはじめとした大規模言語モデルの登場により、自然言語処理における様々なタスクの性能が飛躍的に向上した。現在に至るまで、様々な日本語事前学習済み言語モデルが公開されている。例えば、東北大学の乾研究室は事前学習済み BERT モデル[1)]を公開しているほか、早稲田大学の河原研究室は事前学習済み RoBERTa モデル[2)]を公開している。

こうした日本語の事前学習済み言語モデルが用いるトークナイザは、モデルにより異なる。前述の東北大 BERT モデルは、形態素解析器に MeCab [2]、サブワード分割手法に WordPiece [3] を用いたトークナイザを活用している。一方で、早稲田大学 RoBERTa モデルは、形態素解析器に Juman++ [4]、サブワード分割手法に Unigram [5] を用いたトークナイザを活用している。

トークナイザが異なると、入力文の分割粒度が異なるため、言語モデルの学習する潜在表現に何らかの影響を与えると考えられる。このため、トークナイザの違いは下流タスクの性能に影響する可能性がある。実際にトークナイザの下流タスクに与える影響については、築地ら [6] や井上ら [7] が調査している。しかし、前者は事前学習時とファインチューニング時で異なるトークナイザを用いている点、後者はサブワード分割手法に BPE のみ考慮している点や 1 個の下流タスクでのみ分析を行っている点から、先行研究はトークナイザの下流タスクの性能に与える影響について十分な検討を行えていない。

そこで本研究では、形態素解析器を使わない場合も考慮した 6 種の形態素解析器と 3 種のサブワード分割手法の組合せに対して、同一のデータによりトークナイザの学習および BERT の事前学習を行い、JGLUE [8] や固有表現抽出 [9]、構文解析 [10] タスクにおけるファインチューニング時の性能を測定することで、形態素解析器やサブワード分割手法の違いが与える影響を網羅的に分析した。

本論文の貢献は次の 2 点である。(1) 複数の日本語形態素解析器とサブワード分割手法の組合せについて、統一的な実験設定で事前学習、およびファインチューニングを実施し、様々な下流タスクにおける性能評価を行った。(2) 各種形態素解析器、およびサブワード分割手法が下流タスク性能に与える影響を 5 つの観点から解析した。

2 日本語トークナイザ

日本語の事前学習済み言語モデルが用いるトークナイザは、形態素解析器とサブワード分割手法により構成される。トークナイズ手順は、まず形態素解析器により入力文を単語単位に分割する。次に、分割された各単語に対して、サブワード分割手法を用いてサブワード単位に分割する。言語モデルへの入力は、サブワード単位に分割された文を用いる。

以下、本章では本稿において考慮する形態素解析器とサブワード分割手法について解説する。

* 株式会社日立製作所でのインターンシップ中の成果。
1） https://huggingface.co/cl-tohoku/bert-base-japanese
2） https://huggingface.co/nlp-waseda/roberta-base-japanese

2.1 形態素解析器

MeCab MeCab [2] は辞書をもとにラティスを構築し、ビタビアルゴリズムにより累積コストが最小となる組合せを選択することでトークナイズを行う。コストの計算は、CRF による系列ラベリングの際に素性関数を用いて行われる。また、辞書には IPAdic や NEologd [11][3] などがあり、NEologd は新語・固有表現が追加され、語彙数が多い辞書である。

Juman++ Juman++ [4] は人手で調整された基本語彙辞書と Wikipedia や Web コーパスから自動作成した辞書をもとにラティスを構築し、ビームサーチによりシーケンススコアが最大となる組合せを選択することでトークナイズを行う。シーケンススコアの計算は、RNN 言語モデルを用いて行われる。

Sudachi Sudachi [12] は、UniDic [13] と NEologd をベースに人手で調整された辞書に基づいてトークナイズを行う。また、Sudachi は分割粒度の異なる 3 つのトークナイズが可能であり、目的に応じて使い分けられる。使用できる辞書としては Small、Core (Sudachi のデフォルト辞書)、Full の 3 種類の辞書が存在する。本研究では、基本的な語彙を収録した Core を辞書に用いた。

Vaporetto Vaporetto [14] は一定幅の窓を設定し、窓内の文字列から素性を取り出し、各文字間の単語境界の有無を線形分類モデルにより判別することでトークナイズを行う。辞書には UniDic を用いる。

2.2 サブワード分割手法

Byte-Pair-Encoding BPE (Byte-Pair-Encoding) [15] は、文字ペアの出現頻度が高いものから結合リストに追加し、結合した文字列を辞書に追加することで学習される。トークナイズは、文字分割後に結合リストを参照し、出現頻度の高い文字ペアから順に結合することで行われる。

WordPiece WordPiece [3][4] は、文字ペアの頻度が高く、個々のパーツの出現頻度が低いものから結合した文字列を辞書に追加することで学習される。トークナイズは、先頭の文字から順に、辞書にある最長のサブワードにより分割することで行われる。

Unigram Unigram [5] は、出現した全文字パターンを辞書に追加し、Unigram 言語モデルによりその語彙を失ったときの損失を計算し、損失の小さい語彙を削除することで学習される。トークナイズは、全文字パターンを列挙し、トークンの出現確率が高い語彙の組み合わせを選択することで行われる。

3 実験設定

比較対象 形態素解析器には §2.1 で示した MeCab、MeCab+NEologd、Juman++、Sudachi、Vaporetto の 5 種、サブワード分割手法には §2.2 で示した BPE、WordPiece、Unigram の 3 種を用いた計 15 種、および、形態素解析器を用いずサブワード分割手法のみを用いた 3 種の計 18 種のトークナイザを比較する。また、参考値として、`bert-base-japanese`[1] の性能も測定する。

評価観点 本稿では、トークナイザが言語モデルに与える影響を以下の 5 観点から調査する。

- 形態素解析器を用いるべきか
- 形態素解析器の違いが与える影響
- どのサブワード分割手法を用いるべきか
- 形態素解析器の辞書の違いが与える影響
- トークナイザ辞書の類似性が与える影響

実験データ トークナイザの学習は、Wikipedia[5] からランダムに 1000 万文を抽出し、実行した。ただし、"Category"および表が入っている文と 30 文字未満の文を除いた。事前学習は Wikipedia と CC-100[6] を用い、それぞれ 512 トークンに近づくように文を結合してから、220 万、110 万個のデータを抽出し、マスク言語モデリング [1] により実行した。ファインチューニングには、JGLUE、ストックマーク株式会社による日本語 NER データセット、UD-Japanese-GSD を用いた。[7] ただし、JGLUE は test セットが公開されていないため、train セットにより 5 分割交差検証を行い、dev セットで評価を行った。また、NER データセットは dev セットと test セットが存在しないため、train セットを 9:1 に分割し、前者で 5 分割交差検証を行い、後者で評価を行った。

ハイパーパラメータ トークナイザは語彙数を 3 万として学習した。モデルには、BERT-base (L=12、H=768、A=12) を採用した。事前学習では、オプティマイザに AdamW を用い、学習率を 1e-4、バッチサイズを 128、ステップ数を 50 万ステップとした。ファインチューニングでは、学習率を 3e-5、バッチサイズを 32、エポック数を 5 とした。また、最

3) https://github.com/neologd/mecab-ipadic-neologd
4) 本研究での WordPiece は BERT の最長一致の実装に基づく。
5) https://www.tensorflow.org/datasets/catalog/wikipedia#wikipedia20201201ja
6) https://metatext.io/datasets/cc100-japanese
7) 各タスクの説明は付録 §A.1 に記載。

Approach		MARC-ja Acc	JSTS Pearson / Spearman	JNLI Acc	JSQuAD F1	JCQA Acc	NER F1	UD LAS	Avg.
bert-base-japansese		95.5 (0.1)	89.6 (0.2) / 85.3 (0.3)	86.8 (0.6)	86.4 (0.2)	76.6 (0.8)	85.6 (0.2)	93.3 (0.1)	87.4
MeCab	BPE	95.4 (0.2)	88.8 (0.2) / 84.2 (0.1)	88.0 (0.4)	90.1 (0.3)	74.1 (0.7)	83.7 (0.8)	93.6 (0.1)	87.2
	WordPiece	95.5 (0.1)	86.8 (0.4) / 82.4 (0.5)	87.5 (0.3)	89.2 (0.3)	69.8 (0.7)	84.0 (0.9)	93.6 (0.1)	86.1
	Unigram	95.4 (0.3)	88.6 (0.3) / 84.6 (0.4)	88.3 (0.4)	89.5 (0.3)	74.5 (0.8)	83.1 (1.0)	93.4 (0.2)	87.2
MeCab	BPE	95.4 (0.1)	89.0(0.2) / 84.7 (0.2)	87.9 (0.4)	89.7 (0.1)	74.0 (0.6)	83.0 (0.5)	93.5 (0.1)	87.2
+ NEologd	WordPiece	95.3 (0.6)	87.2 (0.3) / 83.0 (0.3)	87.7 (0.2)	89.3 (0.2)	69.6 (0.4)	82.7 (0.2)	93.5 (0.1)	86.0
	Unigram	95.4 (0.2)	88.7 (0.2) / 84.5 (0.3)	88.0 (0.4)	89.7 (0.3)	75.0 (0.6)	84.1 (0.8)	93.3 (0.1)	87.3
Juman++	BPE	95.5 (0.1)	88.9 (0.4) / 84.6 (0.4)	87.6 (0.4)	90.1 (0.2)	73.8 (0.3)	85.1 (0.6)	93.6 (0.1)	87.4
	WordPiece	95.3 (0.3)	87.5 (0.3) / 83.3 (0.3)	87.7 (0.2)	89.8 (0.3)	71.1 (0.6)	84.7 (0.5)	93.6 (0.1)	86.6
	Unigram	95.4 (0.2)	88.5 (0.3) / 84.3 (0.3)	87.8 (0.3)	89.9 (0.2)	74.9 (1.2)	84.1 (0.4)	93.4 (0.1)	87.3
Sudachi	BPE	95.5 (0.1)	88.6 (0.3) / 84.2 (0.2)	88.2 (0.3)	90.2 (0.2)	74.2 (0.6)	83.5 (0.6)	93.8 (0.1)	87.3
	WordPiece	95.3 (0.2)	87.8 (0.2) / 83.7 (0.3)	87.2 (0.4)	89.6 (0.1)	70.0 (0.9)	82.4 (0.6)	94.0 (0.1)	86.3
	Unigram	95.6 (0.2)	88.9 (0.4) / 84.8 (0.5)	88.4 (0.3)	89.9 (0.1)	74.5 (0.6)	83.0 (1.3)	93.7 (0.1)	87.4
Vaporetto	BPE	95.6 (0.1)	88.9 (0.2) / 84.8 (0.2)	87.5 (0.3)	89.9 (0.2)	74.2 (1.1)	84.1 (0.9)	93.7 (0.1)	87.3
	WordPiece	95.3 (0.2)	87.7 (0.2) / 83.6 (0.1)	88.0 (0.4)	89.7 (0.2)	71.0 (0.4)	84.0 (0.8)	93.8 (0.1)	86.6
	Unigram	95.5 (0.3)	88.7 (0.1) / 84.6 (0.2)	87.9 (0.3)	89.9 (0.1)	74.3 (0.8)	84.1 (0.4)	93.7 (0.1)	87.3
なし	BPE	95.4 (0.2)	87.3 (0.2) / 82.8 (0.2)	87.2 (0.2)	88.7 (0.3)	72.8 (0.8)	62.9 (1.1)	93.4 (0.1)	83.8
	WordPiece	85.5 (0.0)	Nan / Nan	55.3 (0.0)	10.1 (0.1)	20.0 (0.8)	0.0 (0)	63.8 (0.9)	29.3
	Unigram	95.4 (0.4)	88.2 (0.3) / 83.9 (0.3)	87.7 (0.8)	89.3 (0.1)	74.6 (0.4)	76.9 (1.0)	93.2 (0.2)	86.2

表 1　JGLUE、NER、UD データセット上での評価結果。各数値は、JGLUE と NER では 5 分割交差検証の平均値 (標準偏差) を、UD では 5 シードの平均値 (標準偏差) を示している。JCQA は JCommonsenseQA データセットの略である。

Approach	MARC-ja Sequence	JSTS Sequence	JNLI Sequence	JSQuAD Token	JCQA Sequence	NER Token	UD Token	Avg. Overall	Avg. Token	Avg. Sequence
BPE	95.5 / 95.4	86.7 / 85.1	87.8 / 87.2	90.0 / 88.7	74.1 / 72.8	83.9 / 62.9	93.6 / 93.4	87.3 / 83.8	89.2 / 81.7	86.1 / 85.1
WordPiece	95.3 / 85.5	85.3 / Nan	87.6 / 55.3	89.5 / 10.1	70.3 / 20.0	83.6 / 0.0	93.7 / 63.8	86.3 / 46.9	88.9 / 24.6	84.8 / 32.2
Unigram	95.5 / 95.4	86.6 / 86.1	88.1 / 87.7	89.8/ 89.3	74.6 / 74.6	83.7 / 76.9	93.5 / 93.2	87.3 / 86.2	89.0 / 86.5	86.3 / 86.0
Avg.*	95.4 / 95.4	86.2 / 85.6	87.8 / 87.5	89.8 / 89.0	73.0 / 73.7	83.7 / 69.9	93.6 / 93.3	87.0 / 85.0	89.0 / 84.1	85.7 / 85.5

表 2　形態素解析器ありの場合となしの場合の性能比較。数値は各サブワード分割手法における各タスクでの性能の平均値であり、{形態素解析器ありの性能平均}/{形態素解析器なしの性能}を示している。また、全タスク、Token-level タスク、Sequence-level タスクの平均値も示している。ただし **Avg*** は形態素解析器なしの WordPiece を計算から除外した。

大シーケンス長に関しては、MARC-ja と UD では 512、JSTS、JNLI と NER では 128、JSQuAD では 348、JCommonsenseQA では 64 とした。[8)]

4　実験結果

各トークナイザの下流タスク性能を表 1 に示す。

形態素解析器を用いるべきか　まず、形態素解析器の有無が与える影響について分析する。具体的には、形態素解析器ありの場合となしの場合で、サブワード分割手法ごとに下流タスクの性能を比較する。また、形態素解析器の有無による影響は、タスクの粒度やサブワード分割手法によって異なる可能性があるため、これらの観点からも性能を比較する。表 2 は、形態素解析器ありの場合となしの場合において、サブワード分割手法ごとのタスク別平均値および、全タスク (Avg. / Overall)、Token-level タスク、Sequence-level タスクの平均値を示している。ただし、形態素解析器なしの場合の WordPiece は明らかに性能が低いため、平均値 (Avg.*) の計算から除外した。Token-level タスクとは各トークンに対応する出力を用いて解くタスクであり、JSQuAD、NER、UD が該当する。Sequence-level タスクとは [CLS] や [SEP] トークンを用いて解くタスクであり、MARC-ja、JSTS、JNLI、JCQA が該当する。

表 2 より、全タスク平均 (Avg. / Overall, Avg.*) において、形態素解析器ありの方がなしよりも 2.0 ポイント上回り、各タスクでの平均 (Avg.*) においても、7 タスク中 5 タスクで形態素解析器ありの方が性能が高い結果となった。さらに、サブワード分割手法別で比較した場合、どのサブワード分割手法においても形態素解析器ありの方が性能が高く (Avg. / Overall)、タスク粒度単位で比較した場合、Token-level タスクでは形態素解析器ありの方が平均で 4.9 ポイント上回っており (Avg. / Token)、Sequence-level タスクでは形態素解析器ありの方が

8)　ハイパーパラメータの詳細は付録 §A.2 に記載。

平均で 0.2 ポイント上回った (Avg. / Sequence)。以上から、サブワード分割手法によらず形態素解析器を用いた方が良く、特に Token-level タスクで形態素解析器を用いた方が良いと考えられる。

形態素解析器の違いが与える影響　表 1 の形態素解析器ありの手法について比較する。形態素解析器の違いが与える影響を調査するため、単語辞書だけが MeCab と異なる、MeCab+NEologd を除いて比較を行った。比較の結果、MARC-ja、JSTS、JNLI、JSQuAD および JCQA の計 5 タスクの性能に有意差はなかった ($p \geq .05$, Kruskal-Wallis test) が、NER と UD の 2 タスクでは有意差が確認された ($p < .05$, Kruskal-Wallis test)。これらの結果から、少なくとも Sequence-level タスクでは形態素解析器の違いが与える影響は小さいと考えられる。

どのサブワード分割手法を用いるべきか　表 2 の各サブワード分割手法における全タスク平均 (Avg. / Overall) より、形態素解析器ありの場合に関して、Unigram と BPE の性能は 87.3 で等しく、WordPiece の性能は 86.3 と 1.0 ポイント悪化している。特に、Sequence-level タスク (Avg. / Sequence) においては、WordPiece の性能が低くなる傾向がある。[9] この結果は、英語データセット上でのサブワード分割手法の比較 [16] における結果と同様の傾向である。したがって、サブワード分割手法には BPE か Unigram を用いるべきと考える。

単語分かち書き辞書の違いが与える影響　佐藤ら [17] は、形態素解析器に豊富な語句を含む単語分かち書き辞書を活用することで、文書分類タスクの一部の性能が向上することを示している。そこで、より豊富な語を含んだ単語分かち書き辞書（NEologd）を考慮することで、下流タスク性能に差異が生じるかを分析した。表 1 の MeCab と MeCab+NEologd を比較すると、タスクごとの性能に有意差があるとは言えず ($p \geq .05$, Mann–Whitney U test)、NEologd による明確な性能向上は見られなかった。NEologd は新語・固有表現に強いとされており、本研究では特に NER や JSQuAD タスクの性能に影響を与えると想定されたが、性能差は見られなかった。

トークナイザ辞書の類似性が与える影響　トークナイズ結果はサブワード分割手法のトークナイザ辞書に大きく影響するため、辞書の類似度が高いものはそのアルゴリズムによらずトークナイズ結果も類似し、下流タスクの性能も類似するのではないかと考えた。そこで、トークナイザ辞書の類似度と下流タスクの関係について調査する。トークナイザ辞書の類似度と下流タスクの関係を図 1 に示す。横軸はトークナイザ辞書の類似度を示しており、トークナイザ辞書に出現する単語の重複率により算出した。また、縦軸は各トークナイザの性能の差を示しており、表 1 の各トークナイザの全タスク平均 (Avg.) の差により算出した。つまり、ある 1 点は 2 つのトークナイザの (辞書の類似度, 性能の差) を示している。図 1 より、下流タスクの性能の差とトークナイザ辞書の類似度の間には強い負の相関（-0.821）があり、トークナイザ辞書の類似度が高ければ高いほど下流タスクの性能差は小さくなる傾向がある。したがって、トークナイザ辞書の類似度が高いと下流タスクの性能も類似すると考えられる。

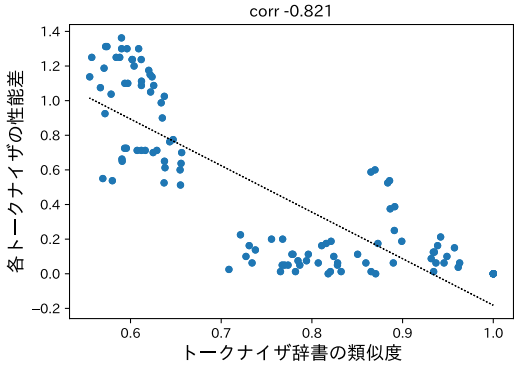

図 1　トークナイザ辞書の類似度と性能の関係。横軸はトークナイザ辞書の類似度、縦軸は各トークナイザの性能の差を表している。

5　結論

本稿では、全 18 種のトークナイザについて BERT モデルを事前学習し、様々な下流タスクでファインチューニングを行うことで、日本語トークナイザの違いが下流タスクの性能に与える影響について調査した。実験の結果、形態素解析器ありの方がなしよりも、全タスクの性能の平均値において 2.0 ポイント向上することが確認されたが、形態素解析器の違いや単語分かち書き辞書の違いによる性能の向上は見られなかった。また、サブワード分割手法に WordPiece を用いた場合、BPE および Unigram と比較して、性能が 1.0 ポイント低下することが確認された。最後に、トークナイザ辞書の類似性が下流タスクの性能の差と相関があることを示した。今後は、エンコーダ・デコーダモデルやデコーダモデルにおけるトークナイザが与える影響を調査したい。

9)　考察を A.3 に記す。

謝辞

株式会社日立製作所の清水正明氏には、本研究での実験に活用した大規模計算機資源の維持管理をしていただきました。ここに御礼申し上げます。

参考文献

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In **Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)**, pp. 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[2] T. KUDO. Mecab : Yet another part-of-speech and morphological analyzer. **http://mecab.sourceforge.jp**, 2006.

[3] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. **arXiv preprint arXiv:1609.08144**, 2016.

[4] Arseny Tolmachev, Daisuke Kawahara, and Sadao Kurohashi. Juman++: A morphological analysis toolkit for scriptio continua. In **Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations**, pp. 54–59, Brussels, Belgium, November 2018. Association for Computational Linguistics.

[5] Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. In **Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)**, pp. 66–75, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[6] 築地俊平, 新納浩幸. Tokenizer の違いによる日本語 bert モデルの性能評価. 言語処理学会第 27 回年次大会, 2021.

[7] 井上誠一, Nguyen Tung, 中町礼文, 李聖哲, 佐藤敏紀. 日本語 gpt を用いたトークナイザの影響の調査. 言語処理学会第 28 回年次大会, 2022.

[8] 栗原健太郎, 河原大輔, 柴田知秀. Jglue: 日本語言語理解ベンチマーク. 言語処理学会第 28 回年次大会, 2022.

[9] 近崇宏江. Wikipedia を用いた日本語の固有表現抽出のデータセットの構築. 言語処理学会第 27 回年次大会, 2021.

[10] Takaaki Tanaka, Yusuke Miyao, Masayuki Asahara, Sumire Uematsu, Hiroshi Kanayama, Shinsuke Mori, and Yuji Matsumoto. Universal dependencies for japanese. In **Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)**, pp. 1651–1658. European Language Resources Association (ELRA), May 2016.

[11] 佐藤敏紀, 橋本泰一, 奥村学. 単語分かち書き辞書 mecab-ipadic-neologd の実装と情報検索における効果的な使用方法の検討. 言語処理学会第 23 回年次大会, 2017.

[12] Sakamoto Miho, Kawahara Noriko, Hisamoto Sorami, Takaoka Kazuma, and Uchida Yoshitaka. Large scale dictionary development for sudachi. 言語資源活用ワークショップ発表論文集.

[13] 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. 伝康晴 and 小木曽智信 and 小椋秀樹 and 山田篤 and 峯松信明 and 内元清貴 and 小磯花絵. 日本語科学, 第 22 巻, pp. 101–123.

[14] 赤部晃一, 神田峻介, 小田悠介, 森信介. Vaporetto: 点予測法に基づく高速な日本語トークナイザ. 言語処理学会第 28 回年次大会, 2022.

[15] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In **Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)**, pp. 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics.

[16] Rastislav Hronsky and Emmanuel Keuleers. **Does the Choice of a Segmentation Algorithm Affect the Performance of Text Classifiers?** Masterpieces of Swiss Entrepreneurship, Swiss SMEs Competing in Global Markets.

[17] 佐藤敏紀, 橋本泰一, 奥村学ほか. 単語分かち書き用辞書生成システム neologd の運用-文書分類を例にして. 研究報告自然言語処理 (NL), Vol. 2016, No. 15, pp. 1–14, 2016.

[18] Timothy Dozat and Christopher D. Manning. Deep biaffine attention for neural dependency parsing. In **International Conference on Learning Representations**, 2017.

[19] Chengyue Gong, Di He, Xu Tan, Tao Qin, Liwei Wang, and Tie-Yan Liu. Frage: Frequency-agnostic word representation. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, **Advances in Neural Information Processing Systems**, Vol. 31. Curran Associates, Inc., 2018.

[20] Kaj Bostrom and Greg Durrett. Byte pair encoding is suboptimal for language model pretraining. In **Findings of the Association for Computational Linguistics: EMNLP 2020**, pp. 4617–4624, Online, November 2020. Association for Computational Linguistics.

A 付録

A.1 下流タスク

下記に、下流タスクに用いた 7 種類のタスクの説明をする。MARC-ja、JSTS、JNLI、JSQuAD、JCommonsenseQA は JGLUE[10] に含まれており、NER は日本語の固有表現抽出データセット[11]、UD は UD-Japanese-GSD[12] を用いた。また、各データセットの統計情報を付録の表 3 に示す。

MARC-ja 商品レビューを入力としてポジティブ/ネガティブをを推定する 2 値分類タスク。

JSTS 2 文を入力として 5 から 0 の範囲で類似度を推定するタスク。

JNLI 2 文を入力として含意関係を推定する 3 値分類タスク。

JSQuAD コンテキストの中から質問文の回答となる部分を抽出する質問応答タスク。

JCommonsenseQA 質問文に対して 5 つの選択肢の中かから 1 つを選択する質問応答タスク。

NER (Named Entity Recognition) 1 文の各トークンに対して固有表現タグを割り当てるタスク。

UD (Universal Dependencies) 各トークンに対して主辞となるトークンを当てる構文解析タスク。

Dataset		Task	train	dev	test
JGLUE	MARC-ja	Text Classification	187,528	5,654	-
	JSTS	Sentence Pair Classification	12,451	1,457	-
	JNLI		20,073	2,434	-
	JSQuAD	QA	62,859	4,442	-
	JCommonsenseQA		8,939	1,119	-
NER		Named Entity Recognition	5,343	-	-
UD		Dependency Parsing	7,050	507	543

表 3 データセット統計情報

A.2 実装

トークナイザの実装には、`tokenizers`[13] を用いた。モデルの実装には、PyTorch[14] と `transformers`[15] を用いた。UD は、Deep Biaffine Attention Parser (BAP) [18] を事前学習済みモデルの出力層上に構築し、学習を行った。BAP の実装には、SuPar[16] を用いた。実験は NVIDIA Tesla V100 GPU (SXM2 - 32GB) を用い、事前学習では 4 個、ファインチューニングでは 1 個で行った。また、すべての学習を fp16 でおこなった。AdamW のパラメータは事前学習、ファインチューニング共に Adam $\beta 1$ を 0.9、Adam $\beta 2$ を 0.999、Adamϵ を 1e-08 とした。

A.3 サブワード分割手法に WordPiece を用いると性能が低下する原因

サブワード分割手法に WordPiece を用いた場合に性能が低下する理由を考察する。まず、"##"による区別の影響が考えられる。例えば、MeCab・WordPiece の辞書に「田」と「##田」が存在するように、同じ文字でも"##"の有無により 2 回出現する場合があり、辞書に採録できるトークン数が実質的に減るため、`[UNK]` が増加してしまう。次に、出現頻度による影響が考えられる。「田」と「##田」は出現頻度に差がある。出現頻度の異なるトークンは、言語的意味が類似していても、埋め込み表現の類似度は低くなる [19]。最後に、デッドゾーンが影響すると考えられる。デッドゾーンとは、辞書内の他の語彙より出現頻度の低いトークンである。これは中国語や日本語の漢字を含む出現頻度の稀な文字 (鍜、鰺など) が多数含まれることで生じる [20]。デッドゾーンが多いと、`[UNK]` が増加してしまう。実際に、MeCab を用いた場合、WordPiece、BPE、Unigram の順でデッドゾーンが多いことが観測された。上記の統計的解析については、今後の課題としたい。

10) https://github.com/yahoojapan/JGLUE
11) https://github.com/stockmarkteam/ner-wikipedia-dataset
12) https://github.com/UniversalDependencies/UD_Japanese-GSD
13) https://github.com/huggingface/tokenizers
14) https://pytorch.org/
15) https://github.com/huggingface/transformers
16) https://github.com/yzhangcs/parser