令和 3 年度茨城大学工学部情報工学科卒業研究論文

Tokenizer の違いにおける日本語 BERT モデルの性能評価

所属 情報工学科

著者 築地俊平 (17T4055G)

指導教員 新納浩幸教授

令和 3 年 2 月 5 日 (金)

令和 3 年度茨城大学工学部情報工学科卒業研究論文

Tokenizer の違いにおける日本語 BERT モデルの性能評価

著者

築地俊平 (17T4055G)

指導教員

新納浩幸教授

論文要旨

近年，事前学習モデルを利用し，自然言語処理システムの性能が飛躍的に向上している．その事前学習モデルの中でも，BERT は特に大きく性能の向上を示している．BERT の日本語版である日本語 BERT に限らず，日本語を扱う事前学習モデルは，英語など明確な区切りのある言語と違い，単語や文字、サブワードなどの分割方法によって文書を分割をするステップが必要である．現在、日本語 BERT は様々な種類がある. 異なる点はいくつかあるが，その一つに、日本語 BERT によって使用している Tokenizer が異なるということが挙げられる．日本語での学習をするときは、各々の日本語 BERT によって指定されている Tokenizer で単語を分割するのが一般的である．

本研究では，訓練データとテストデータを日本語 BERT で指定されている Tokenizer とは別の Tokenizer にした場合の精度がどの程度変化するのかを実験し，性能の評価を行い，変化を考察した．

目次

第 1 章　序論 5

1.1　概要 5

1.2　構成 6

第 2 章　Tokenizer 7

2.1　Tokenizer 7

2.2　Tokenizer を使用して文章を語彙 ID 列にする 8

第 3 章　BERT を用いた文書分類 10

3.1　文書分類タスク 10

3.2　BERT の概要 11

3.3　BERT の構造 13

第 4 章　実験 15

4.1　実験手法 15

4.2　実験環境 15

4.3　性能評価に使用する Tokenizer 17

4.4　実験結果 19

第 5 章　考察 21

第 6 章　結論 22

参考文献 24

付録 25

A ソースコード 25

第 1 章

序論

1.1 概要

近年，事前学習済みモデル BERT を利用することで自然言語処理システムの性能が飛躍的に向上している．当初，実質英語だけが対象であった BERT だが，現在は日本語 BERT も数多く構築・公開され，一般に利用できるようになってきた．様々な日本語 BERT がある中で，どの BERT を利用すべきかは明らかではないが，選択の要因の一つは単語分割にあると考えられる．日本語 BERT は各々の Tokenizer を持ち、そこで利用される語彙リストが異なっている．そのためタスクの目的に合った Tokenizer を持つ BERT を利用した方がよいと考えられる．一方，利用する BERT を固定したい場合も多いと思われる．その場合、その BERT の持つ Tokenizer とは異なる Tokenizer を利用した際に，性能への影響がどの程度あるのかは興味深い．ここでは東北大版の ’bert-base-japanese’ と呼ばれる BERT を利用して，上記の調査を行った.’bertbase-japanese’ の BERT は Tokenizer として MeCab が利用されている．入力された文は MeCab により単語分割され，各単語が token id に変換される．この際、語彙リストに登録されていない単語については WordPiece が起動されて単語分割が行われる．つまり BERT への入力として別の Tokenizer を利用して単語分割を行っても、token id 列に変換され，以後の処理は正常に行われる．本研究では BERT を利用した文書分類のタスクを行う．利用する BERT は東北大版日本語 BERT である．文書から BERT の入力となる token id 列を得るのに，様々な Tokenizer を利用して、Tokenizer の違いによる性能の評価を行う．

1.2　構成

本研究では BERT のモデルの入力を様々な Tokenizer を使用し語彙 ID 列にし性能の評価を行う．2 章，3 章では本研究で実際に使われる項目である，BERT，Tokenizer についての説明を行う．4 章では実際に実験をし，5 章と 6 章では提案した手法についての結果の考察と，本研究における問題点と改善点，今後の課題について述べる．

第 2 章

Tokenizer

2.1 Tokenizer

日本語は英語と比べると単語間に明確な区切りが存在しない．そのため、日本語や明確な区切りのない言語の解析をする場合には単語，文字など何らかの部分に分ける必要がある．さまざま分け方はあるが，分割された文字列のことをトークンという．Tokenizer は文書を単語分割をして，それに ID という数値に変換するモジュールである．単語を分割する方法としては

- N-gram を使って単語に分割
- MeCab，Juman などを使って形態素解析して形態素に分割
- 事前に単語単位に分割して各単語の頻度を求め，高頻度で現れる単語を一つの単語とするサブワードとして分割

があげられる．また、サブワードについてくわしくは 4.3 で後述する．

2.1.1 形態素解析

形態素解析は自然言語処理において単語を分割する手法の中で最もメジャーである．言語上の最小単位である最小単位である形態素に分解し，形態素の品詞などの属性をつける処理である．形態素解析器の多くは，文法規則や単語の要素を集めた辞書データを使用する．自然言語は複数の解釈が可能であり，形態素解析をしても分解の方法が一つに定まらないものもある．形態素解析器はアルゴリズムや辞書データの違いによって出

力が変化してくる．

2.1.2　N-gram

N-gram とは自然言語処理において単語を分割する手法の一つ．N-gram は連続する N 個の文字，もしくは N 個の単語で分割する手法である．辞書が不必要であるという特徴がある．また，形態素解析と比べ単語の数が多くなってしまう．特に N の値が小さいほど分割した単語の数が多くなる．

N=1 の場合はユニグラム (uni-gram)，N=2 の場合はバイグラム (bi-gram)，N=3 の場合はトライグラム (tri-gram) と呼ぶ．例えば，コンビニ店員になりきる という文章を N-gram にしてみると．

N=1 uni-gram

コ/ン/ビ/ニ/店/員/に/な/り/き/る

N=2 bi-gram

コン/ンビ/ビニ/ニ店/店員/員に/にな/なり/りき/きる

N=3 tri-gram

コンビ/ンビニ/ビニ店/ニ店員/店員に/員にな/になり/なりき/りきる

単語の単位で N-gram を使うこともある．

N=1 uni-gram

コンビニ/店員/に/ないきる

N=2 bi-gram

コンビニ店員/店員に/になりきる

N=3 tri-gram

コンビニ店員に/店員になりきる

2.2　Tokenizer を使用して文章を語彙 ID 列にする

表 2.1 は今回の実験で使用する各 Tokenizer ごとに単語分割したものであり，表 2.2 は表 2.1 の文章を id 化したものである．どの分割の仕方も若干ではあるが違いがあり，各 Tokenizer によって分割するアプローチが少しづつ異なるためである．

表 2.1: Tokenizer による単語分割

Tokenizer	分割
生データのまま	コンビニ店員になりきってお釣りを返していこう！
MeCab	コンビニ 店員 に なり きっ て お釣り を 返し て いこ う ！
MeCab+NEologd	コンビニ店 員 に なり きっ て お釣り を 返し て いこ う ！
Juman++	コンビニ 店員 に なりきって お釣り を 返して いこう ！
Sudachi	コンビニ 店員 に なり きっ て お釣り を 返し て いこう ！
nagisa	コンビニ 店 員 に なり きっ て お 釣り を 返し て いこう！
SentencePiece	▁コンビ ニ 店 員 になり き って お 釣り を 返し てい こう！

表 2.2: Tokenizer による token id 列

Tokenizer	語彙 ID 列
生データのまま	2, 20617, 20333, 7, 297, 2551, 16, 73, 30406, 28477, 11, 13043, 16
MeCab	2, 20617, 20333, 7, 297, 2551, 16, 73, 30406, 28477, 11, 13043, 16
MeCab+NEologd	20617, 805, 1151, 7, 297, 2551, 16, 73, 30406, 28477, 11, 13043, 16
Juman++	2, 20617, 20333, 7, 297, 2551, 16, 73, 30406, 28477, 11, 13043, 16
Sudachi	20617, 20333, 7, 297, 2551, 16, 73, 30406, 28477, 11, 13043, 16
nagisa	2, 20617, 805, 1151, 7, 297, 2551, 16, 73, 8639, 11, 13043, 16
SentencePiece	1, 3900, 353, 805, 1151, 7, 297, 322, 6172, 73, 8639, 11, 13043, 16

第 3 章

BERT を用いた文書分類

3.1 文書分類タスク

自然言語処理における文書分類とは，一般に文書を特定の分類体系に自動的に割り当てる処理のことを指す．分類にも種類があり，一例としてはトピック分類，評判分析，属性推定などがある．例えば，ある文書 X をカテゴリ A，B，C，のいずれかに分けるとする場合を考える．この際に，各カテゴリにはすでに複数の文書が分類済みであることとする．

このときの分類の手順としては，各カテゴリとある文書 X に含まれる文書から何らかの特徴をあらかじめ抽出しておく．その後，文書 X の特徴と似た特徴を数多く含む文書の糧折に文書 X を分類する．この分類を近年では機械学習を用いることが一般的とされており，様々な方法が考案されている．

文書分類は分類問題の一種であり，一般に教師あり学習を用いることで解決できる．そのため従来より数多くの研究がある．またディープラーニングを利用する場合でも，CNN や RNN を利用するなどの多くの研究がある．

文書分類を含め自然言語処理の多くのタスクにおいて，事前学習モデルを利用する有効性が示されている．事前学習モデルを利用する場合，大きく 2 つの利用法がある．一つは fine-tuning である．これは事前学習モデルが出力する情報を，タスクを解決するためのネットワークの入力とし，その事前学習モデルを含めたネットワーク全体を学習の対象とするものである．この場合，事前学習モデルの部分はすでに大量のデータから学習できた形となっているため，比較的少量のデータを用いるだけで，連結したネットワークを学習できる．

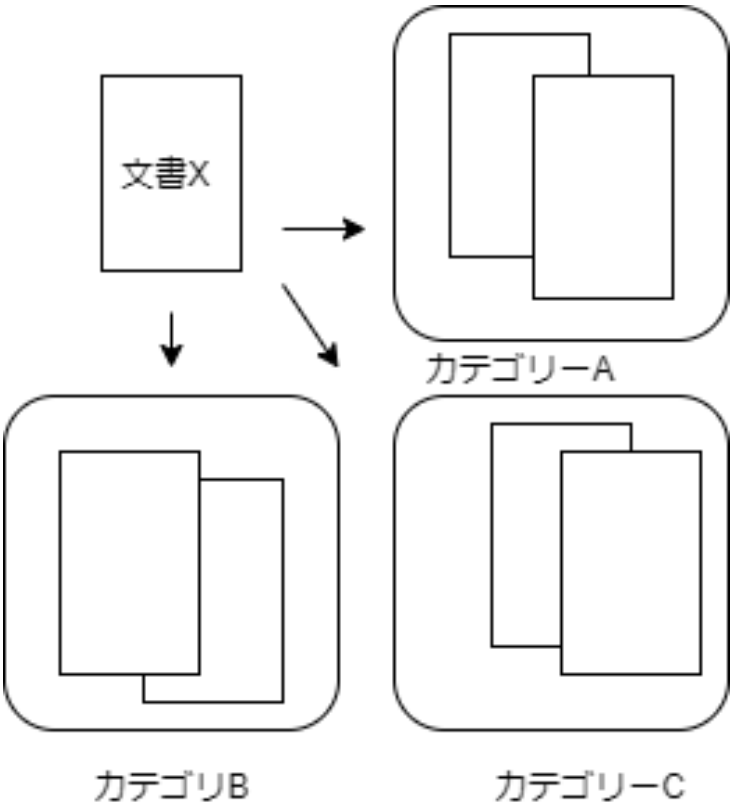

図 3.1: 文書分類の例

[1]

3.2　BERT の概要

BERT (Bidirectional Encoder Representations from Transformers) は Google の研究チームが 2018 年 10 月に公開した論文 [2] で，文書分類，質問応答，固有表現抽出等の多様なタスクで公開当時の最高性能 (SOTA: State of the Art) を達成した画期的なモデルである．自然言語処理の分野の中に様々なタスクがあり，各々のタスクを解くために，それぞれに設計されていたモデルが存在したが，そのスコアを超えた．その後、BERT のモデルを分析，改善，パラメータの数を増減させたりといった手法で，BERT の派生のモデルが出現した．

3.2.1　BERT モデル

様々なタスクへの応用を前提とし，その結果として得られたパラメータを用いて，狙いのタスクを解くために，有効な特徴量を学習することを事前学習という．

事前学習のアプローチは特徴量ベースのアプローチと fine-tuning のアプローチがある．

特徴量ベースのアプローチは，事前学習で得られたパラメータ一方 fine-tuning というのはパラメータを固定しない

BERT モデルは後者の fine-tuning のアプローチによる事前学習モデルであり，汎用的な言語モデルの事前学習を行う手法の一つである．BERT は双方向 Transformer をベースとした汎用言語モデルであり，大量のテキストデータを事前学習したモデルに少量の教師データを追加学習させることで，テキスト分類などの様々なタスクで先行研を超える分類性能を達成している．

自然言語処理に限らず、近年の大規模化した深層学習モデルは非常に大量の学習データを必要とする．大規模モデルの学習に十分な量のラベル付き日本語データセットは少ないという現状がある．

しかし，BERT はラベルなしデータによる事前学習を用いることで、大規模モデルに不可欠な大量データという課題に対応しうる手法になることが予想される．

3.2.2　事前学習

BERT は入力に対し、以下の二つの目的関数 (Masked LM, Next Sentence Prediction) を合わせた事前学習を行う．

1 つは Masked Language Model である．これは, 入力シーケンスから”’[CLS]’, ”’[SEP]’ を省いて無作為抽出した 15 %を以下のように置き換えたうえで，置き換える前の語を予測する．

- 80 %を”[MASK]” に差し替える
- 10 %をランダムな語彙に差し替える
- 10 %はオリジナルのまま

もう 1 つは Next Sentence Prediction である．これは，二つの入力分に対して 50 % を連続したもの、50 %を不連続なものとする．先頭の “[CLS]” に対応する出力から連続か不連続かを予測する．

BERT は事前学習で，文章の穴埋めの能力と二つの入力分が連続しているか否かを予測する能力を得ている．

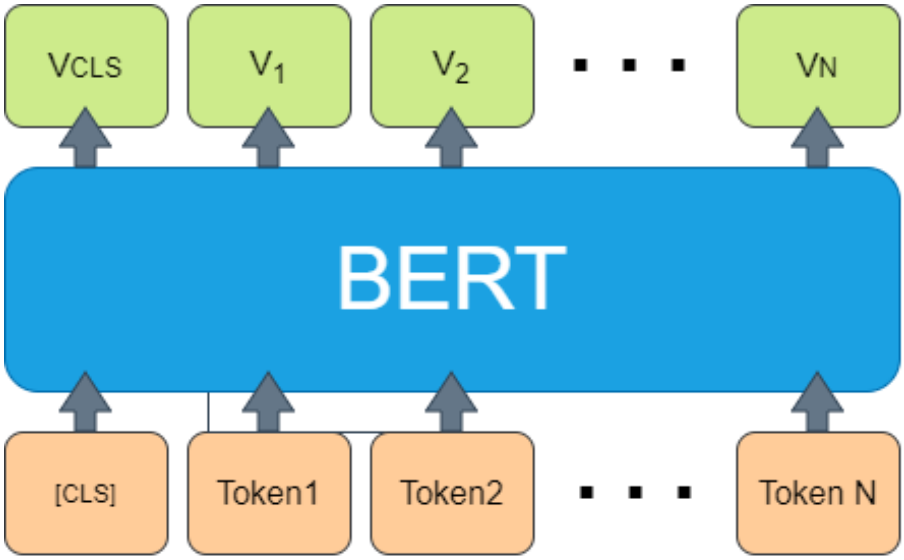

図 3.2: BERT モデル

3.3　BERT の構造

ベクトルのシーケンス (Token1 … TokenN）を入力とし、ベクトルのシーケンス (V1 … VN) が出力されるモデルである．入力層と出力層の間に隠れ層を複数重ねた構造となる．自然言語処理のタスクを行うニューラルネットワークには RNN　や CNN　がよく利用されていた．Transformer と呼ばれる構造が提案されたことで、BERT の隠れ層には Transformer が利用され，自然言語処理のタスクのスコアが向上した．

BERT のモデル構造の主な部分は，Transformer の Encoder を複数重ねたものである．”Attention is All You Need” [3] で表記されている Transformer は Encoder と Decoder の二つで構成されているが，Transformer の Encoder と表記せず，Transformer と表記することが多い．

Encoder 部分は入力と出力は配列であり，同じ長さになる．モデルに文章を入力する．文章をトークンに分割し，トークン列として入力するそして、この Encoder は，そこのトークン列の長さはそのままに，個別のトークンをベクトルに変換する．

これは学習するタスクによって変わる．文章のトークン列の位置を利用するものは，その対応するベクトルを後のレイヤーに渡す．

分類問題のために特別に文頭に [CLS] というトークンをつけて学習する．分類問題を解く場合には，対応する先頭のベクトルを，分類問題を解くためのレイヤーに渡す．

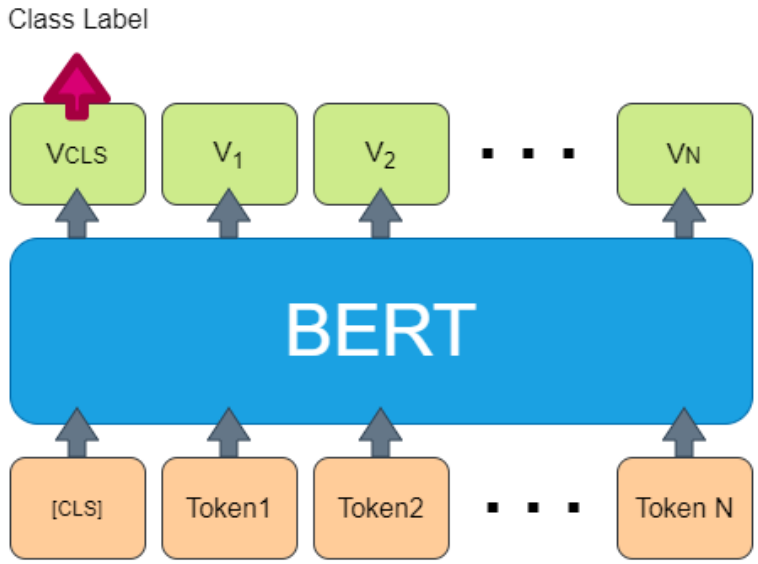

図 3.3: 分類タスクを解く場合

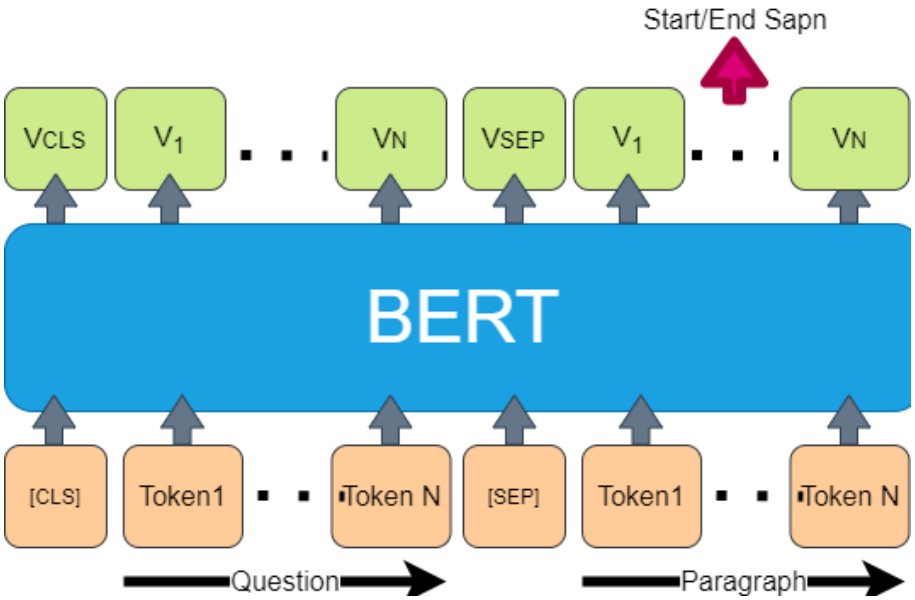

図 3.4: 質問から回答を得るタスクの場合

事前学習で 2 つのタスクや，fine-tuning のときに，そのタスクに応じてこのレイヤーが切り替えられる．

第 4 章

実験

4.1 実験手法

標準的な手法では，訓練データ，テストデータの文書の埋め込み表現列は，それぞれの日本語 BERT 固有の Tokenizer を使って単語分割し，語彙 ID 列にしたものを埋め込み表現列とし，タスクを実行している．本研究では，日本語 BERT モデルを使用して文書分類を行う．文書分類をする際に異なる Tokenizer で単語分割し，語彙 ID 列にする．その際に，Tokenizer ごとに評価に違いがあるのかを調べる．

4.2 実験環境

本研究では 6 種類の Tokenizer，事前学習済みもでるには東北大学乾・鈴木研究室が公開している日本語 BERT を使う．

その中で Toknizer の SentencePiece は日本語 Wikipedia データを使用する．使用する日本語 BERT の Tokenizer は MeCab+WordPiece である．

本研究では訓練データ、テストデータを livedoornews から使用した．テストデータは 736 文、訓練データは 5887 文が用意し，それぞれ以下の 9 つのカテゴリに属するものを収集したデータセットである．

- トピックニュース
- SportsWatch
- IT ライフハック
- 家電チャンネル

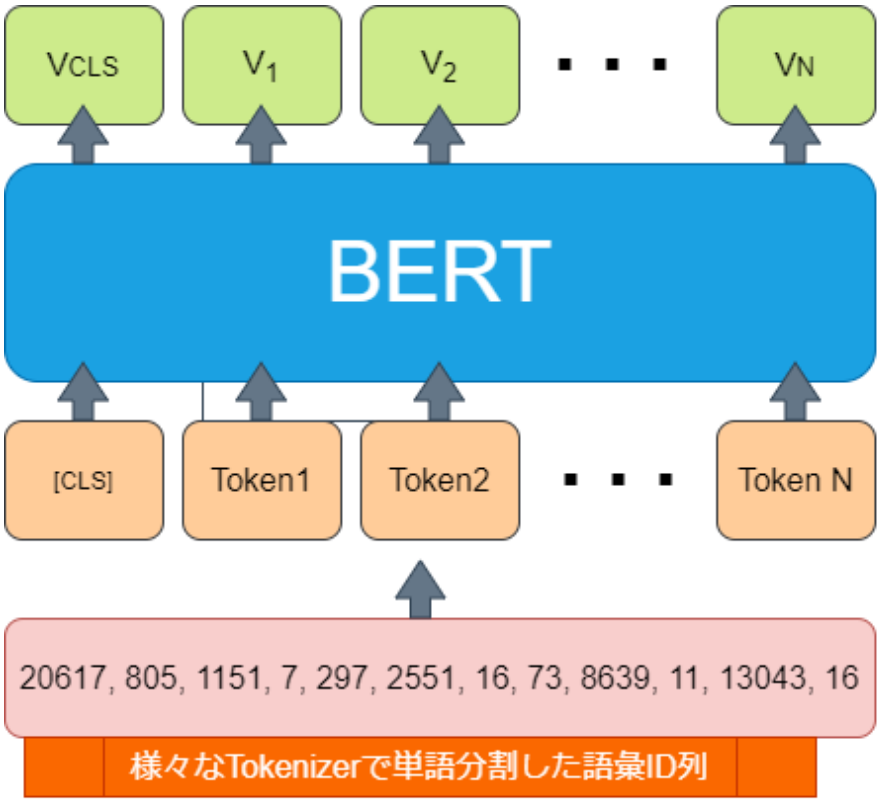

図 4.1: BERT モデルに語彙 ID 列を入力

- MOVIE ENTRY
- 独女通信
- エスマックス
- livedoor HOMME
- Peachy

本実験では 9 種類の文書を文書分類をするタスクを行う．まずは通常の単語分割がされていない生データの文書を訓練データとして学習させ，単語分割していない生データであるテストデータを実行し精度を確認した．次に各 Tokenizer で訓練データ，テストデータを単語分割した文書を使い精度を確認した．続けて訓練データのみ Tokenizer で単語分割した文書を使用し，テストデータは単語分割のない生データの文書で精度を確認．最後に訓練データは単語分割されていない生データを使用し，テストデータは各 Tokenizer で単語分割した文章を使いそれぞれ精度を確認した．

今回の精度の判定は、9 つのカテゴリーごとに分かれた訓練データから学習し，その後

テストデータの 736 文の文書を分類したのちにいくつの文書がカテゴリーと一致しているかを判別するというものである．

4.3　性能評価に使用する Tokenizer

今回使用する Tokenizer の特徴は以下の通りである．Tokenizer ごとにアルゴリズムが異なっているため，特徴と違いについて触れる．

■MeCab+IPAdic、MeCab+IPAdic+NEologd　一般的に使用されている形態素解析器．ラティスを構築して、その中から最適なパスを選択する．

辞書である NEologd が追加されたため複数の形態素に分割されてしまう固有表現をもっている．ニュース記事から抽出した新語や固有名詞や道語，ネット上で流行した単語や慣用句やハッシュタグなども行っている．文書クラスが定義する以下についての変更は禁止とする．

■Juman++　京都大学黒橋研究室で作成された,RNN を使用した形態素解析機である．ニューラル LM を用いることで精度が高くなっている．辞書は基本辞書，Wikipedia, Wikiionary，Web text であり、合計で 90 万語ほどになっている．

■Sudachi　ワークスアプリケーションズ　ワークス徳島人工知能 NLP 研究所が開発した形態素解析機である．

UniDic 由来の単単位語句と NEologd 由来の膨大な固有名称をベースに大規模な辞書データをを構築している. 約 280 万語を超えている．最適な形態素の長さは，アプリケーションによって異なるという考えから表 4.1 のように，3 種類の形態素単位が用意されている．A 単位とはほぼ UniDic の単単位規定と同じであるが，一部のものは更に短くしているものを A 単位としている．B 単位はとは A 単位に説示及び漢字 1 文字の名詞が結合したもの，及び複合動詞である．C 単位とは更に多くの語句が結合したもので，複合名詞や固有名称，慣用句などがこれに相当し，アプリケーションごとに形態素単位を選択している．

Neologd 由来の語句は長単位のものが多いため，基本的にすべて分割情報を付与している．ただし，量が膨大なため，まず機械的に分割情報を付与し，人手でチェックする．

さまざまなツールがある中，検索システムでの利用を主に，商業利用に耐えうる，より高品質で使い勝手の良い形態素解析器であり，辞書の継続的なメンテナンスがされるこ

表 4.1: Sudachi の分割情報と 3 種類の形態素解析結果

登録見出し	選挙/管理/委員会	カンヌ/国際/映画祭
A 単位	選挙—管理—委員—会	カンヌ—国際—映画祭
B 単位	選挙—管理—委員会	カンヌ—国際—映画祭
C 単位	選挙管理委員会	カンヌ国際映画祭

とも大きな特徴である．

精度は MeCab と同じくらいであり、速度は MeCab に劣るが，メモリをあまり使わない工夫がされている．[4]

■nagisa　Bidrectional-LSTM を用いた日本語単語分割と品詞推定が特徴である．Bi-LSTM を用いた文字単位の系列ラベリングによる解析を行うので，顔文字や URL，ネット用語に対し頑健な解析ができる．また，多言語に対応している．[5]

■SentencePiece　サブワードの考え方を用いた分割方法である．

サブワードとはテキストの中で低頻度で現れる単語は文字や部分文字列に分解するという考え方である．事前にテキストを単語分割し，各単語の頻度を求めておく．高頻度の単語は 1 単語として扱い，低頻度後は短い単位に分割する．最終的なトークン数が事前に与えられた数千から数万のサイズになるように分割をする．サブワードのメリットは未知語がなくなるという点にある．低頻度語も最終的には文字に変換されるため，語彙数を小さくすることができる．[6] 形態素解析だと辞書に登録されていない単語は未知語になるが，サブワードはそうではなく，単語の意味を考慮した分解にはならないという特徴がある．

サブワードは単語や文字を部分文字列に分解する．しかし，英語やヨーロッパ言語の場合であれば，単語の同定は容易だが，日本語や中国語にサブワードを適用すると，事前に形態素解析で単語分割の処理を行う必要がある．SentencePiece は単語列からではなく，生分から直接分割を学習する．

SentencePiece は文書をサブワードに分割するモデルであり，コーパスから教師なし学習が可能になる．特徴の一つとして，End to End 処理に向けたテキストの可逆分割ができる．通常の形態素解析の単語分割では，分割はできてもそれを復元することは困難である．この単語分割の逆を脱トークン化という. 脱トークン化を可能にするために，

SentencePiece ではスペースを通常のトークンとして扱う．便宜的にスペースをメタ文字 (_) に書き換える．言語依存の処理がなく，完全な End-to-End 処理が可能になる．[7]

4.4　実験結果

今回の実験では，訓練データとテストデータの文書を生データのまま東北大版 BERT で文書分類し，評価値を調べたところ，結果は 87.44 となった．これを基準として，3 つの実験結果を見ていく．

訓練データとテストデータを各々同じ Tokenizer で単語分割したデータでタスクを行った. 文書分類をした結果は表 4.2 となった．全体を通じて，生データのまま文書分類をした場合と比べて，評価値が飛躍的に高くなる，低くなることはなかった．SentencePiece で単語分割したデータの評価値がほかと比べて低くなっていた.

訓練データは生データ，各 Tokenizer で単語分割したテストデータで文書分類のタスクを行った結果は表 4.3 となった. ここでも生データのままタスクを行ったときの評価値と大きく差は出なかったが，SentencePiece のみ 84.96 と大幅に評価値を落としていた.

テストデータは生データのままで，各 Tokenizer で単語分割した訓練データでの文書分類のタスクを行った結果は表 4.4 となった. 同じように 88.00 を超えることはなく，SentencePiece が評価値落としていた．Tokenizer が同じで，訓練データやテストデータで比較すると，大きな違いはなく，SentencePiece 以外は 87.00 から 87.90 の間の評価値であった.

表 4.2: 訓練データとテストデータを Tokenizer で単語分割

両方のデータに使用した Tokenizer	accuracy
生データのまま	87.44
MeCab+IPAdic	87.39
MeCab+IPAdic+NEologd	87.85
Juman++	87.76
Sudachi	87.59
nagisa	87.71
SentencePiece	86.37

表 4.3: テストデータを Tokenizer で単語分割

テストデータに使用した Tokenizer	accuracy
生データのまま	87.44
MeCab+IPAdic	87.77
MeCab+IPAdic+NEologd	87.26
Juman++	87.77
Sudachi	87.26
nagisa	87.45
SentencePiece	84.96

表 4.4: 訓練データを Tokenizer で単語分割

訓練データに使用した Tokenizer	accuracy
生データのまま	87.44
MeCab+IPAdic	87.21
MeCab+IPAdic+NEologd	87.53
Juman++	87.80
Sudachi	87.58
nagisa	87.30
SentencePiece	85.11

第 5 章

考察

東北大版の日本語 BERT の Tokenizer は MeCab+WordPiece である．この実験で使用した他の Tokenizer では分割が異なるため，語彙 ID 列が変化する．そのため，MeCab で単語分割したタスクは生データの評価値と近くなり、それ以外の Tokenizers で単語分割分割し，タスクをを個なったときは少し評価がさがることが予想されていた．しかし，結果としては，SentencePiece で分割した時を除いて，評価値は生データでタスクを行ったときと大きな変化はなかった．これは，多少 Tokenizer の仕組みが変わっても，ID 化したときにはタスクにはあまり影響が出ない程度しか語彙 ID 列に違いがなかったと考えられる．Tokenizer で分割した文書を語彙 ID 列にしたときの違いの割合と関係していると考察する．SentencePiece に関しては，辞書を使い形態素解析をしているわけではないので他の Tokenizer と比べると語彙 ID 列に変化があったと考えられる．

第 6 章

結論

本研究では，訓練データとテストデータの文書を様々な Tokenizer を使用することによって，日本語 BERT の評価値の変化を確認することができた．単語ごとに分割する形態素解析器での文章の分割の仕方によって，文章のトークン化に多少の違いはあるが，日本語 BERT の Tokenizer(今回は MeCab+NEologd) でトークン化した ID と結果が類似したためか精度は大きく変化はしなかった．単語を ID にしてもところどころの変化することがあるものの，その程度であれば学習に影響が出にくくなっている．もしくは単語分割，今回であれば形態素解析器らは精度が上がり，どの形態素解析器でも正確に同じように単語分割ができ，語彙 ID 列はが類似しており，その結果，評価値に変化が小さくなってると考える．そうであれば単語分割のアプローチは様々あれど形態素解析であれば，結局はあまり変わらないことが予想され，逆に形態素として解析せずサブワードでの単語分割は少しの変化があることが予想される．

課題としては，今回は日本語 BERT の東北大版のみを実験で使用したが，日本語事前学習モデル全体でも今回の実験と同じように結果の差が出るかはわからないことがあげられる．

今後は，異なる日本語 BERT での精度の変化を調べたい．SentenceBART など Tokinizer がサブワードで単語分割されている日本語 BERT では違いがでる可能性がある．

謝辞

最後に、本論文を作成するにあたり、指導、助言を頂いた、指導教官の新納浩幸教授に心より感謝申し上げます．また, 日常の議論を通じて多くの知識や示唆を頂いた新納研究室の皆様に深く感謝いたします.

参考文献

[1] 難波英嗣. 人工知能による文書分類. 2016. https://www.jstage.jst.go.jp/article/jkg/66/6/66_277/_pdf.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018. https://arxiv.org/abs/1810.04805.

[3] Niki Parmar Jakob Uszkoreit Llion Jones Aidan N. Gomez Lukasz Kaiser Illia Polosukhin Ashish Vaswani, Noam Shazeer. Attntion is all you need. 2017. https://arxiv.org/pdf/1706.03762.pdf.

[4] 坂本美保, 川原典子, 久本空海, 一馬, 内田 佳孝（株式会社ワークスアプリケーションズ ワークス徳島人工知能 NLP 研究所）. 形態素解析器『sudachi』のための大規模辞書開発. 言語資源活用ワークショップ 2018, 2018.

[5] 池田大志. Python による日本語自然言語処理 系統ラベリングによる実世界テキスト分析 . 2019. https://speakerdeck.com/taishii/pycon-jp-2019.

[6] Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. 2018. https://www.aclweb.org/anthology/P18-1007.pdf.

[7] John Richardson Taku Kudo. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. 2018. https://arxiv.org/abs/1810.04805.

付録

A ソースコード

実験で使用したソースコードを A.1 と A.2 に示す.

ソースコード A.1: train.py

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pad_sequence
from transformers import BertModel, BertConfig, BertJapaneseTokenizer

import numpy as np
import pickle
import sys
import re

argvs = sys.argv
argc = len(argvs)

config = BertConfig.from_json_file('../tohoku/config.json')
bert = BertModel.from_pretrained('../tohoku/pytorch_model.bin',config=
    config)
tknz = BertJapaneseTokenizer.from_pretrained('cl-tohoku/bert-base-
    japanese')

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu"
```

```
    )

# DataLoader

class MyDataset(Dataset):
    def __init__(self, xdata, ydata):
        self.data = xdata
        self.label = ydata
    def __len__(self):
        return len(self.label)
    def __getitem__(self, idx):
        x = self.data[idx]
        y = self.label[idx]
        return (x,y)

def my_collate_fn(batch):
    images, targets= list(zip(*batch))
    xs = list(images)
    ys = list(targets)
    return xs, ys

# Data setting

xdata, ydata = [],[]
with open(argvs[1],'r',encoding='utf-8') as f:
    for line in f:
        line = line.rstrip()
        result = re.match('^(\d+)\t(.+?)$', line)
        ydata.append(int(result.group(1)))
        sen = result.group(2)
        tid = tknz.encode(sen)
        if (len(tid) > 512):
            tid = tid[:512]
        xdata.append(tid)

#xdata = []
#with open('xtrain.pkl','br') as fr:
# xdata = pickle.load(fr)

#ydata = []
#with open('ytrain.pkl','br') as fr:
```

```
65 # ydata = pickle.load(fr)
66 print("dataset␣end")
67
68 batch_size = 4
69 dataset = MyDataset(xdata,ydata)
70 dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True,
       collate_fn=my_collate_fn)
71
72 # Define model
73
74 class DocCls(nn.Module):
75     def __init__(self,bert):
76         super(DocCls, self).__init__()
77         self.bert = bert
78         self.cls=nn.Linear(768,9)
79     def forward(self,x1,x2):
80         bout = self.bert(input_ids=x1, attention_mask=x2)
81         bs = len(bout[0])
82         h0 = [ bout[0][i][0] for i in range(bs)]
83         h0 = torch.stack(h0,dim=0)
84         h1 = self.cls(h0)
85         return h1
86
87 # model generate, optimizer and criterion setting
88
89 model = DocCls(bert)
90 model.to(device)
91
92 optimizer = optim.SGD(model.parameters(),lr=0.001)
93 criterion = nn.CrossEntropyLoss()
94
95 # Learn
96
97 epoch_num = 30
98
99 model.train()
100 for ep in range(epoch_num):
101     i = 0
102     for xs, ys in dataloader:
103         xs1, xmsk = [], []
104         for k in range(len(xs)):
```

```
105             tid = xs[k]
106             xs1.append(torch.LongTensor(tid))
107             xmsk.append(torch.LongTensor([1] * len(tid)))
108         xs1 = pad_sequence(xs1, batch_first=True).to(device)
109         xmsk = pad_sequence(xmsk, batch_first=True).to(device)
110         outputs = model(xs1,xmsk)
111         ys = torch.LongTensor(ys).to(device)
112         loss = criterion(outputs, ys)
113         print(i, loss.item())
114         optimizer.zero_grad()
115         loss.backward()
116         optimizer.step()
117         i += 1
118 #   Add model
119     outfile = "./mecab/model" + "/" + str(ep) + ".model"
120     torch.save(model.state_dict(),outfile)
121     print(outfile,"␣saved")
```

ソースコード A.2: test.py

```
1  #!/usr/bin/python
2  # -*- coding: utf-8 -*-
3
4  import torch
5  import torch.nn as nn
6  import torch.optim as optim
7  import torch.nn.functional as F
8  from torch.utils.data import Dataset, DataLoader
9  from torch.nn.utils.rnn import pad_sequence
10 from transformers import BertModel, BertConfig, BertJapaneseTokenizer
11
12 import numpy as np
13 import pickle
14 import sys
15 import re
16
17 argvs = sys.argv
18 argc = len(argvs)
19
20 config = BertConfig.from_json_file('/../tohoku/config.json')
21 bert = BertModel.from_pretrained('/../tohoku/pytorch_model.bin',config=
```

```
    config)
tknz = BertJapaneseTokenizer.from_pretrained('cl-tohoku/bert-base-
    japanese')

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu"
    )

# DataLoader

class MyDataset(Dataset):
    def __init__(self, xdata, ydata):
        self.data = xdata
        self.label = ydata
    def __len__(self):
        return len(self.label)
    def __getitem__(self, idx):
        x = self.data[idx]
        y = self.label[idx]
        return (x,y)

def my_collate_fn(batch):
    images, targets= list(zip(*batch))
    xs = list(images)
    ys = list(targets)
    return xs, ys

# Data setting

xdata, ydata = [],[]
with open(argvs[1],'r',encoding='utf-8') as f:
    for line in f:
        line = line.rstrip()
        result = re.match('^(\d+)\t(.+?)$', line)
        ydata.append(int(result.group(1)))
        sen = result.group(2)
        tid = tknz.encode(sen)
        if (len(tid) > 512):
            tid = tid[:512]
        xdata.append(tid)

#xdata = []
```

```
#with open('xtest.pkl','br') as fr:
# xdata = pickle.load(fr)

#ydata = []
#with open('ytest.pkl','br') as fr:
# ydata = pickle.load(fr)
print("dataset_end")

batch_size = 20
dataset = MyDataset(xdata,ydata)
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True,
    collate_fn=my_collate_fn)

# Define model

class DocCls(nn.Module):
    def __init__(self,bert):
        super(DocCls, self).__init__()
        self.bert = bert
        self.cls=nn.Linear(768,9)
    def forward(self,x1,x2):
        bout = self.bert(input_ids=x1, attention_mask=x2)
        bs = len(bout[0])
        h0 = [ bout[0][i][0] for i in range(bs)]
        h0 = torch.stack(h0,dim=0)
        h1 = self.cls(h0)
        return h1

# model generate, optimizer and criterion setting

model = DocCls(bert)
model.load_state_dict(torch.load(argvs[2]))
model.to(device)

# Learn

model.eval()
with torch.no_grad():
    ok = 0
    n = 0
    for xs, ys in dataloader:
```

```
100         n += len(ys)
101         xs1, xmsk = [], []
102         for k in range(len(xs)):
103             tid = xs[k]
104             xs1.append(torch.LongTensor(tid))
105             xmsk.append(torch.LongTensor([1] * len(tid)))
106         xs1 = pad_sequence(xs1, batch_first=True).to(device)
107         xmsk = pad_sequence(xmsk, batch_first=True).to(device)
108         ans = model(xs1,xmsk)
109         ans = ans.to('cpu')
110         ans1 = torch.argmax(ans,dim=1)
111         ys1 = torch.LongTensor(ys)
112         ok += (ys1 == ans1).sum().float().item()
113     print(ok/n, int(ok), n)
```