令和 4 年度茨城大学大学院理工学研究科情報工学専攻

修士学位論文

平仮名 BERT 構築と平仮名文の分割

所属　情報工学専攻

著者　井筒順 (21NM707H)

指導教員　新納浩幸教授

令和 4 年 2 月 3 日 (金)

令和 4 年度茨城大学大学院理工学研究科情報工学専攻　修士学位論文

平仮名 BERT 構築と平仮名文の分割

著者

井筒順 (21NM707H)

指導教員

新納浩幸教授

論文要旨

既存の日本語の形態素解析システムの性能は非常に高いが、これらのシステムは漢字仮名混じりの文を対象にしているため、平仮名で書かれた文を単語分割することは困難である。これは一般的な形態素解析モデルを構築する際に利用されている学習データと、解析対象であるデータの構成が大きく異なっているためである。一方で、日本語学習者などに向けた平易な日本語のシステムのために、平仮名文の平仮名文を分割するシステムが望まれている。本研究では、平仮名、数字、記号に特化した漢字を使用しない文の単語分割を行うシステムを開発する。そのために、我々は unigram BERT と bigram BERT の 2 種類の BERT(Bidirectional Encoder Representations from Transformers) による平仮名文の単語分割モデルを作成した。2 種類の平仮名 BERT モデルの作成に際し、事前学習用データとして Wikipedia のデータを用い、単語分割のためのファインチューニングのデータとして国立国語研究所の『現代日本語書き言葉均衡コーパス』(BCCWJ) のコアデータを利用した。さらに、作成した 2 種類の BERT による平仮名文の単語分割における F 値と比較するため、KyTea を用いた平仮名文 KeyTea 単語分割モデルを作成した。BCCWJ のコアデータを用い 5 分割交差検証を行ったところ、unigram BERT 単語分割システムでは F 値が 94.30%、bigram BERT 単語分割システムでは F 値が 92.58% の結果をとなり、両システムは共に平仮名 KeyTea 単語分割システムの F 値の 89.66% を上回った。また、実験により、ファインチューニングに利用するデータは、大量のドメインの異なった疑似データよりも、少量のドメインの等しい、テストデータと整合性の取れたデータの方がよいことが分かった。

Master's Thesis in Scholastic 2023, Major in Computer and Information Sciences,
Granduate School of Science and Engineering, Ibaraki University

Structure of Hiragana BERT and Word Segmentation of Hiragana Sentences

Author : Jun Izutsu (21NM707H)

Adviser : Professor Hiroyuki Shinnou

Abstract

In Japanese, word segmentation is necessary when parsing sentences due to the need for clear word boundaries. While many highly effective morphological analyzers are available for Japanese, it can be challenging to accurately segment sentences written mostly in Hiragana, a more straightforward Japanese writing system, as there are fewer clues to aid in the segmentation process. In this study, we developed a word segmentation model for Hiragana sentences using two versions of Bidirectional Encoder Representations from Transformers (BERT): unigram and bigram BERT models. To create these models, we employed Wikipedia data for pre-training and BCCWJ core data to fine-tune the word segmentation task. In order to compare with the performances of the two types of BERT models in word segmentation of Hiragana sentences, we created a word segmentation model of Hiragana sentences using KyTea, a toolkit developed for analyzing text, with a focus on languages requiring word segmentation. We performed a 5-fold cross-validation on the Balanced Corpus of Contemporary Written Japanese (BCCWJ) core data to assess the performance of our unigram and bigram BERT-based word segmentation models. The F values were 94.30% and 92.58% for the unigram and bigram BERT-based word segmentation models whereas 89.66% for the KyTea word segmentation model. Our experiments have also indicated that, for fine-tuning, it is generally more effective to use a small amount in-domain of data that is consistent with the test data rather than a large amount of diverse pseudo-data from various domains.

目次

第 1 章

序論

日本語には MeCab[*1]や Chasen[*2]等の形態素解析システムが存在している。これらのシステムの性能は非常に高いが、漢字仮名混じりの文を対象にしているため、ほとんど全てが平仮名で書かれた文[*3]を単語分割することは難しい。

日本語初学者が最初に習得するのは平仮名であり平仮名で構成された文章を読む機会が多い。しかし平仮名で構成された文章を滞ることなく読むことは日本語の母語話者であっても難しい。

本稿では BERT を利用した 2 種類の平仮名文の分割システムを作成しその性能評価を行う。さらに、KyTea[*4]を用いた平仮名文 KyTea 単語分割システムを作成し、作成した 2 種類の BERT による平仮名文の単語分割システムと比較する。

本研究の目的は、ほとんどが平仮名で構成された文章を単語分割可能なシステムを開発する事である。上記でも述べた通り、日本語の形態素解析器の性能は非常に高精度であるが、平仮名と記号で構成された文章を高精度に解析することはできない。一方で、日本語初学者の外国人の方が使用する文章は平仮名で構成されている事が多いので、ほとんどが平仮名で構成された文章を形態素に解析することは可読性の観点から重要であるといえる。また、未就学児や小学校低学年の生徒が作成する文章を単語分割したい場合、文章の多くが平仮名で構成されているため既存の形態素解析器では単語分割を行うことが難しい。

[*1] https://taku910.github.io/mecab/

[*2] https://chasen-legacy.osdn.jp

[*3] 数字や記号は含まれている。

[*4] http://www.phontron.com/kytea/

上記の問題を解決するために、本研究では、ほとんどが平仮名で構成された文章を単語分割可能なシステムを開発する。

第 2 章

関連研究

平仮名文を分割するための複数の研究が報告されている。工藤ら [1] は，平仮名交じり文が生成される過程を生成モデルを用いてモデル化した。そして、そのパラメータを大規模 Web コーパスと EM アルゴリズムで推定することで、平仮名交じり文の解析性能を向上させる手法を提案している。また、林ら [2] は平仮名語の単語を辞書に追加することで形態素解析の精度が向上することを報告している。井筒ら [3] は MeCab の ipadic 辞書を平仮名に変換し、平仮名のみで構成されたコーパスを用いることで平仮名のみの文での形態素解析を行っている。さらに、井筒ら [4] は Bi-LSTM CRF モデルを用いた平仮名文の形態素解析を行い、複数のジャンルの文に対して複数回、学習とファインチューニングを行うことで形態素解析の性能にどのように変化を与えられるかを報告している。また、森山ら [5] は RNNLM を用いたべた書きかな文の形態素解析を行ない、その性能が単語分割と単語素性の全てを正解とする最も厳しい基準において従来手法を有意に上回ることを報告している。さらに、森山ら [6] は RNN とロジスティック回帰を用いた平仮名文の逐次的な形態素解析手法を提案し、平仮名文における形態素解析の性能向上とシステムの高速化を報告している。

分野に特化した BERT を作成する研究として代表的なものには鈴木ら [7] の研究がある。これは、金融文書を用い金融に関する文に特化した BERT を作成したことを報告する論文である。鈴木らは、汎用言語コーパスを用いて事前学習を行った BERT モデルに対して、金融コーパスを用いてファインチューニングを行うことが有効であるかの検証を行なっている。

これまでに、日本語を形態素解析するために複数のツールが開発されてきた。MeCab は京都大学学術情報科-日本電信電話株式会社コミュニケーション科学基礎研究所共同ユ

ニットプロジェクトを通じて開発されたオープンソースな形態素解析エンジンである [8]。MeCab ではパラメータ推定に Lafferty ら [9] によって提案された CRF（Conditional Random Fields）を利用している。KyTea は京都大学森研究室が開発したツールキットである。KyTea には単語分割機能、品詞推定機能、読み推定機能が含まれている、単語分割を必要とする言語のための一般的なテキスト解析機である。

第 3 章

BERT

本章では BERT（Bidirectional Encoder Representations from Transformers）について説明する。BERT は Devlin ら [10] によって考案された事前学習モデルである。このモデルは Vaswani ら [11] によって設計された Transformer を双方向に接続することで構成されている点に特徴がある。BERT が登場するまでは RNN（Recurrent Neural Network）を利用したモデルが利用されていた。RNN は再帰型ニューラルネットワークとも呼ばれている。RNN は再起的に系列データを入力することにより内部情報を更新することができる。しかし、RNN では学習の際に 1 つ前のデータのみを用いることから、学習を繰り返すことで長期的な依存関係が喪失してしまうという問題点が存在していた。これは、現在の入力よりもよりも遠い過去の情報を残し続けることが難しいことを意味している。

入力された情報よりも遠い過去の情報を残し続けることが難しいという問題を解決するために Sepp ら [12] によって LSTM（Long short-term memory）モデルが提案された。しかしながら LSTM モデルを利用したとしても入力された情報よりも遠い過去の情報を残し続けることが難しいという問題を完全に解決することは難しい。前方から順に処理を行う RNN モデルや LSTM モデルと比較し、BERT ではあるトークンの処理を行う際に、他のトークンに対すして参照を行う部分に大きな違いが存在している。RNN では過去の情報を残し続けることができなかったが、BERT では他のトークンを参照することにより RNN よりも文脈を考慮した分散表現を得ることが可能になる。

3.1　BERT の構造

まず、BERT の構造を表した図を図 3.1 として以下に示す。また、図 3.1 の内、Transformer Encoder 部分の各層における処理を表した図を図 3.2 として以下に示す。

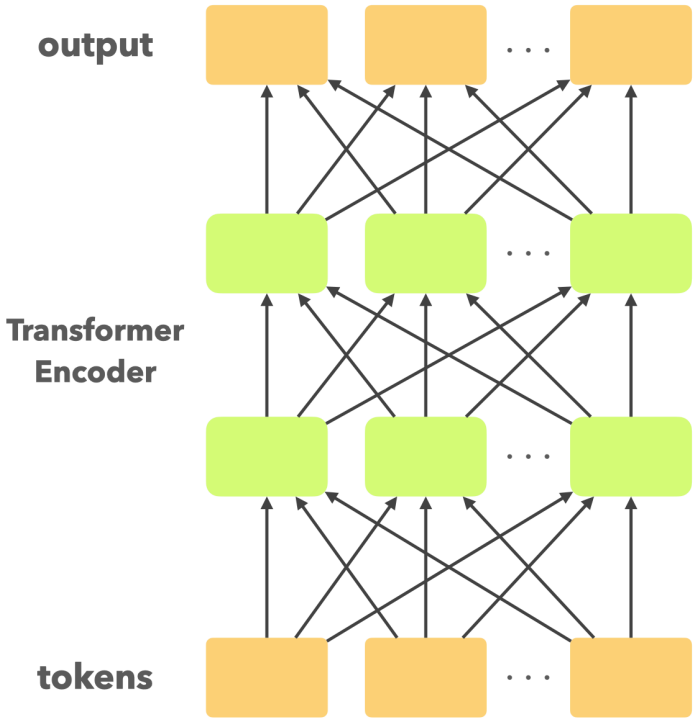

図 3.1: BERT の構造

図 3.2 より、Transformer Encoder における各層の処理には、Multi-Head Attention、Feed-forward Network、Layer Normalization の 3 つの処理が存在していることが分かる。

3.2　Multi-Head Attention

Multi-Head Attention は、入力された情報に対してどの情報に注目するべきかを判断して情報を処理する仕組みである。Multi-Head Attention の構成の論述を行う前に、

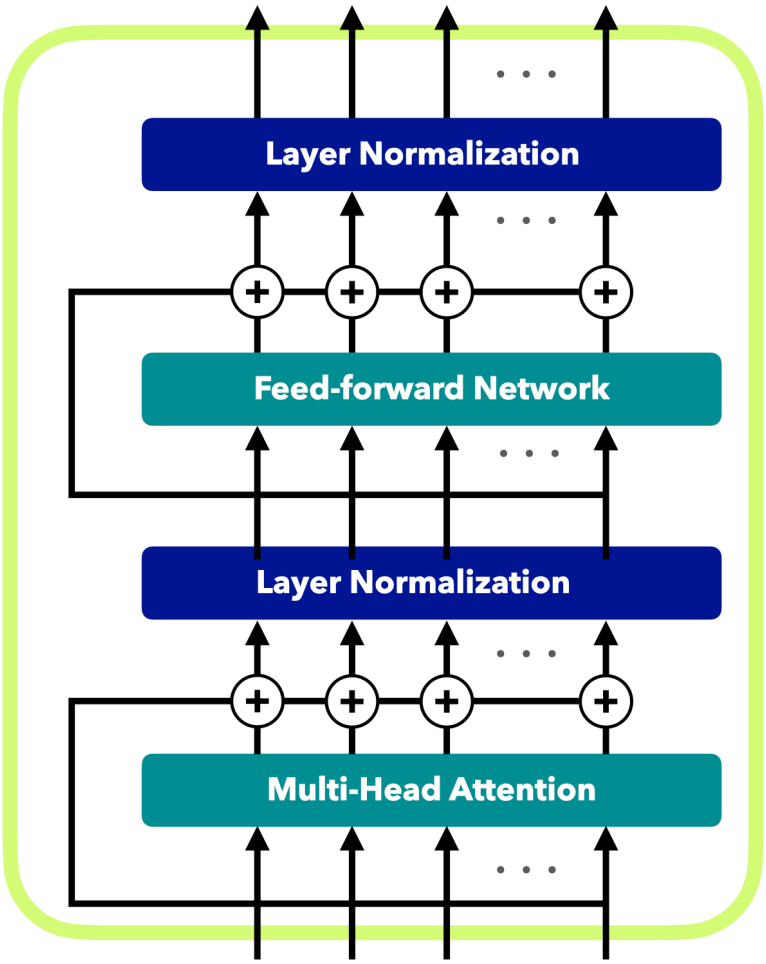

図 3.2: Transformer Encoder における各層の処理

Multi-Head Attention の構成要素となっている Scaled Dot-Product Attention の構成について論述する。まずは Scaled Dot-Product Attention を表す式を式 3.1 から式 3.5 に表す。

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{3.1}$$

$$Q = \begin{pmatrix} \vec{q_1} \\ \vdots \\ \vec{q_n} \end{pmatrix} \tag{3.2}$$

$$K = \begin{pmatrix} \vec{k_1} \\ \vdots \\ \vec{k_n} \end{pmatrix} \tag{3.3}$$

$$V = \begin{pmatrix} \vec{v_1} \\ \vdots \\ \vec{v_n} \end{pmatrix} \tag{3.4}$$

$$softmax = \frac{\exp(y_i)}{\sum_{j=1}^{n}\exp(y_j)} \quad (i = 1,2,...,n) \tag{3.5}$$

式 3.5 は Softmax 関数と呼ばれている関数である。合計が 1 となるように正規化を行う。式 3.2 における Q はクエリーを表す。また、式 3.3 における K はキーを表し、式 3.4 における V はバリューを表す。K と V はキー・バリューの名前の通り辞書の役割を持っているので、任意の k_n に対して任意の v_n を取得することができる。式 3.2 から式 3.4 に存在する n は入力の個数を表している。式 3.1 における d は式 3.2 から式 3.4 におけるベクトルの次元を表している。本式の右辺における QK^T はあるクエリ q_n に対して転置された K が掛けられていることが分かる。QK^T は与えられたクエリーに対してどのキーが似ているかの類似度を並べたベクトルを意味している。ゆえに、QK^T を利用することによりあるクエリーに対して最も類似度の高いキーを取得することが可能となる。QK^T は類似度を並べたベクトルであり、一般的に高次元のベクトルは長くなる傾向があるので $\sqrt{d}$ で割ることで長くなりすぎる部分に対して補正をかけベクトルの長さを減らしている。そして、補正がかけられた類似度が並んだベクトルを softmax 関数に入力することで類似度を合計 1 の重みに変換する。最後に重みに変換された行列とベクトル V の積を加算する。以上が Scaled Dot-Product Attention の式の説明である。Scaled Dot-Product Attention ではクエリーとキーの類似度の計算を行い、類似度に対する重みを算出し、算出した重みに対してバリューを加算している。

次に Multi-Head Attention を表す式を式 3.6 から式 3.7 に表す。

$$Multi\text{-}Head(Q,K,V) = concat(head_i)W^o \tag{3.6}$$

$$head_i = Attension(QW_i^Q, KW_i^K, VW_i^V) \tag{3.7}$$

式 3.6 と式 3.7 において、QW_i^Q、KW_i^K、VW_i^V はそれぞれ入力の行列を掛けることにより、どの部分に注目を行うのか、及び、注目の方法を決定している。この操作によりパラメータを作成することが可能となる。

3.2.1　Feed-forward Network

Feed-forward Network を表す式を式 3.8 に示す。

$$FFN(x) = ReLU(xW_1 + b_1)W_2 + b_2 \tag{3.8}$$

式 3.8 における x は入力を表す。この x は図 3.2 における 1 度目の Layer Normalization における出力にあたる。$ReLu$ は活性化関数を表している。

3.2.2　Layer Normalization

Layer Normalization は、Ioffe ら [13] によって発案された Batch Normalization を改良したものである。Batch Normalization ではバッチサイズが小さい場合に平均と分散が不安定になる問題が存在している。Layer Normalization は、層単位で、次の処理への入に対して正規化を行う。Layer Normalization を利用することで、Batch Normalization と比較して学習が収束するまでの時間を短縮することができ、また、精度を向上させることができる [14]。

3.3　事前学習

事前学習は大量のデータ[*1]を用いることで言語のパターンを学習することであり、Pre-Traning とも呼ばれている。事前学習に利用するデータにはラベルを付与する必要はない。ラベル無しデータを用いる利点として多量のデータを収集しやすいことが挙げられる。もし事前学習に利用するデータにラベルが必要となる場合はラベル付けを人手

[*1] コーパスと呼ばれる

で行うことが多くデータの作成に時間がかかり、かつ、ラベル無しデータと比較してデータ量が減少することがある。

BERT では Masked Language Model と Next Sentence Prediction という 2 つのタスクを事前学習に用いる。

3.3.1　Masked Language Model

Masked Language Model はある単語を隠し、隠した単語における周辺の単語から隠した単語を予測するタスクである。BERT では本タスクを用いて事前学習を行う。隠される単語は [MASK] というトークンに置き換えられる。以下、本タスクの流れについて論述する。まず入力された単語のうち 15% の単語をランダムに選出する。次に、ランダム選出された単語の内、80% の単語を [MASK] に置き換え、10% の単語をランダムに選出された他の単語に置き換える。残り 10% の単語については置き換えを行わない。置き換えが完了したのちに置き換えられた単語を BERT に入力し、[MASK] の位置に元々存在した単語を [MASK] 周辺の単語情報を元に予測させる。以上が Masked Language Model の流れである。このタスクを行うことによりモデルは、学習させる言語の文法や単語の意味を理解できるようになる。

3.3.2　Next Sentence Prediction

Next Sentence Prediction は入力された 2 つの文章が連続した文章であるかどうかを予測するタスクである。BERT では本タスクを用いて事前学習を行う。以下、本タスクの流れについて説明する。まず、事前学習時に利用するコーパスにおいてに 2 つの連続した文章をランダムに抽出する。抽出する量は事前学習のコーパスの 50% である。次に、残された 50% 文章から元の文章で連続しないように 2 つの文章を抽出し連結させる。以上の操作で作成した文章群のうち、前者の 50% の文章群は元のコーパスでも連続した文章の集合であり、後者の 50% の文章群は元のコーパスで連続していなかった文章の集合である。そして、作成した文章をそれぞれ BERT に入力として与え、入力された 2 つの文章が元のコーパスにおいて連続していたかどうかを判定させる。以上が Next Sentence Prediction の流れである。このタスクを行うことにより、モデルは、学習させる言語の文脈や文章の意味を理解できるようになる。

3.4　ファインチューニング

ファインチューニングは、あるモデルを作成する際に学習前のモデルに対して事前学習済みのモデルの重みを初期値として再学習させる手法のことである。ファインチューニングを利用することにより、事前学習によって作成されたモデルを事前学習の際に利用したデータセットとは異なるデータセットを用いて再学習することでモデルの精度を向上させることができる。また、事前学習の際に利用したデータセットとは異なるドメインのデータセットを利用することで別ドメインの知識を作成するモデルに対して持たせることができる。

第 4 章

平仮名 BERT

本研究では平仮名文に特化した 2 種類の平仮名 BERT モデルを生成し、それぞれを利用して平仮名文の単語分割システムを構築した。平仮名 BERT モデルのうち、1 つ目のモデルは unigram BERT モデルである。これは平仮名の文字 unigram で構成された文集合を事前学習に利用して生成した BERT モデルである。2 つ目のモデルは bigram BERT モデルである。これは平仮名の文字 bigram で構成された文集合を事前学習に利用して生成した BERT モデルである。我々は上記 2 つのモデルを平仮名文の単語分割のデータを使ってファインチューニングすることで、平仮名の文の単語分割システムを作成した。本研究では、これら 2 つの平仮名文の単語分割システムの F 値を比較する。さらに、KyTea による平仮名文の単語分割のモデルを作成し、上記 2 つの平仮名文の単語分割システムの F 値と比較する。

4.1 unigram BERT 単語分割モデル

unigram BERT は、事前学習用データとして平仮名の文字 unigram で構成された文を利用した BERT モデルである。Wikipedia の漢字仮名交じり文を平仮名に変換し、さらに文字 unigram に分かち書きしたデータを事前学習用のデータとして使用した。Wikipedia には平仮名のみのデータが存在しないので、MeCab の解析結果における読みのデータを疑似的な正解として利用している。この unigram BERT に対し、平仮名文の単語分割のデータを使ってファインチューニングすることで、平仮名文の単語分割システムを作成した。これを以降、unigram BERT 単語分割システムと呼ぶ。ファインチューニングのデータには、実験によって BCCWJ または Wikipedia のいずれかのデータを利

用している。

4.2　bigram BERT 単語分割モデル

bigram BERT は、事前学習用データとして、平仮名の文字 bigram で構成された文を利用した BERT モデルである。Wikipedia の漢字仮名交じり文を平仮名に変換し、さらに文字 bigram に分かち書きしたデータを事前学習用のデータとして使用した。bigram BERT 単語分割システムの事前学習に使用する Wikipedia は、unigram BERT 単語分割システムの事前学習に使用した Wikipedia と同様に平仮名のみのデータが存在しないので、MeCab の解析結果における読みのデータを疑似的な正解として利用している。この bigram BERT に対し、平仮名文の単語分割のデータを使用してファインチューニングをすることで、平仮名文の単語分割システムを作成した。これを以降、bigram BERT 単語分割システムと呼ぶ。ファインチューニングのデータには、実験によって BCCWJ または Wikipedia のいずれかのデータを利用している。

4.3　平仮名 KyTea 単語分割モデル

京都大学森研究室の開発したテキスト解析ツールキット KyTea を用いて平仮名文の単語分割システムを作成した。KyTea は単語分割および読み推定の機能を持つシステムである。部分的アノテーションから学習をすることが可能であり、点予測を利用して文の解析を行う。KyTea を利用して作成した平仮名文の単語分割システムを以降、平仮名 KyTea 単語分割システムと呼ぶ。学習データは、実験によって BCCWJ または Wikipedia のデータを利用している。

第 5 章

データ

5.1 Wikipedia による事前学習用データ

2 種類の平仮名 BERT を作成するための事前学習用のデータとして、Wikipedia [*1][*2] を利用した。

本データの作成方法を論述する。まず、Wikipedia の漢字仮名交じり文を MeCab を利用して形態素解析を行い、形態素解析結果における読み部分を利用することで平仮名のみで構成された文を得る。MeCab の辞書には Unidic [15] を利用した。MeCab の読みデータから作成しているため、出力される平仮名文は、正確な平仮名文ではなく、疑似的な平仮名文である。次に、平仮名のみで構成された文を文字 unigram の形と文字 bigram の形に変換した。文字 unigram の形に変換したデータは unigram BERT の事前学習用データとして利用し、文字 biram の形に変換したデータは bigram BERT の事前学習用データとして利用する。ただし、bigram の終端における文字列は、unigram との文字数を揃えるために、句点に文字種「*」を追加した「。*」としている。最後に、文の行頭と行末に対してそれぞれ [CLS] タグと [SEP] タグを付与した。unigram BERT と bigram BERT の事前学習用データ例を表 5.1 に示す。

上記の操作により 300 万文の Wikipedia による事前学習用データを得た。作成したデータの中身に関しては文字 bigram で表現されているのか文字 unigram で表現されているのかを除けば同一のデータを利用している。

[*1] `https://dumps.wikimedia.org/jawiki/latest/`

[*2] jawiki-latest-pages-articles.xml.bz2

表 5.1: 事前学習用データ例

元データ	冬が来た。
unigram 用	[CLS] ふ ゆ が き た 。 [SEP]
bigram 用	[CLS] ふゆ ゆが がき きた た。 。* [SEP]

5.2　Wikipedia による平仮名文の単語分かち書きデータ

unigram BERT と bigram BERT におけるファインチューニングに利用するデータとして、Wikipedia による平仮名文の単語分かち書きデータを作成した。本データは、MeCab の単語分割結果を信じ、その平仮名表記を利用して作成した。また、単語分割を二値分類として処理するため、分割対象位置の文字を 1、それ以外の文字を 0 とするタグデータを作成した。Wikipedia による平仮名文の単語分かち書きデータ例を 5.2 に示す。

表 5.2: Wikipedia による平仮名文の単語分かち書きデータタ例

元データ	冬が来た。
unigram 用	ふ ゆ が き た 。
bigram 用	ふゆ ゆが がき きた た。 。*
タグデータ	101101

上記の操作により 100 万文の Wikipedia による平仮名文の単語分かち書きデータを得た。作成したデータの中身に関しては文字 unigram で表現されているのか文字 bigram で表現されているのかを除けば同一のデータを利用している。

5.3　BCCWJ による平仮名文の単語分かち書きデータ

BCCWJ のコアデータは人手により形態素に分けられたデータである。Wikipedia による事前学習用データと Wikipedia による平仮名文の単語分かち書きデータでは正確ではない擬似的な平仮名文を作成したが、BCCWJ のコアデータを利用することにより正確な平仮名文を作成することが可能となる。BCCWJ コアデータを平仮名の分かち書きに変換したデータを平仮名文の単語分割システムのファインチューニング用のデータお

よびテストデータとして利用する。実験による使用データの違いについての詳細は 5 節で述べる。

本データの作成方法は以下である。まず BCCWJ のコアデータの読み情報を利用し、ほぼ平仮名で構成された文に変換した。次にそれらの文を文字 unigram の形と文字 bigram の形に変換した。文字 unigram の形に変換したデータは unigram BERT のファインチューニング用のデータとして利用し、文字 biram の形に変換したデータは bigram BERT のファインチューニング用のデータとして利用する。また BCCWJ のコアデータの単語区切りを利用し単語分割を二値分類として処理するための、分割対象位置の文字を 1、それ以外の文字を 0 とするタグデータを作成した。上記の操作により 40928 文の BCCWJ による平仮名の分かち書きデータを得た。

BCCWJ による平仮名文の単語分かち書きデータ例を 5.3 に示す。

表 5.3: BCCWJ による平仮名文の単語分かち書きデータ

元データ	冬が来た。
unigram 用	ふ ゆ が き た 。
bigram 用	ふゆ ゆが がき きた た。 。*
タグデータ	101101

上記の操作により 40928 文の BCCWJ による平仮名の分かち書きデータを得た。

5.4　平仮名 KyTea 単語分割システムのデータ

平仮名文を単語する分割ができる KyTea モデルの作成を行うために、学習データとして BCCWJ のコアデータによる平仮名の分かち書きデータを利用した。本データの読みの部分を利用しほぼ平仮名のみで構成された分かち書きデータを得た。

平仮名 KyTea 単語分割システム作成に使用したデータ例を 5.4 に示す。

表 5.4: 平仮名 KyTea 単語分割システム作成に使用したデータ

元データ	冬が来た。
学習データ	ふゆ が きた 。

5.5　平仮名 BERT の語彙

unigram BERT 作成において使用した語彙の総数は 300 である。語彙には平仮名、片仮名、アルファベット、数字、複数の記号が含まれている。また、bigram BERT 作成において使用した語彙の総数は 80956 である。語彙には平仮名、片仮名、アルファベット、数字、複数の記号の任意の 2 種類の組み合わせが含まれている。また、複数の記号には [CLS] や [SEP] 等のタグが含まれている。

第 6 章

実験

作成した 2 種類の平仮名 BERT 単語分割システムにおける平仮名文の単語分割の F 値がファインチューニング時のデータ量とデータの種類によりどのように変化するのかを検証するために 2 つの実験を行った。

6.1 実験 1：BCCWJ によるファインチューニングの実験

この実験では、正確な平仮名文の単語分割情報のデータを利用して、2 種類の BERT 単語分割システムの F 値を平仮名 KyTea 単語分割システムの F 値と比較する。

まず、平仮名 BERT 単語分割システムを作成するために必要となる平仮名 BERT を作成する手順について論述する。初めに 300 万文の Wikipedia による事前学習用データを利用して 2 種類の平仮名 BERT を作成する。300 万文の Wikipedia による事前学習用データには bigram 用のデータと unigram 用のデータの 2 種類が存在するので、unigram BERT 単語分割システムを作成するための事前学習データとして 300 万文の unigram 用の Wikipedia による事前学習用データを利用し、bigram BERT 単語分割システムを作成するための事前学習データとして 300 万文の bigram 用の Wikipedia による事前学習用データを利用した。この 2 つの事前学習データを用いて 2 種類の平仮名 BERT を作成を行った。

次に平仮名 BERT 単語分割システムを作成するために、上記で作成した 2 種類の平仮名 BERT それぞれに対して BCCWJ による平仮名文の単語分かち書きデータを用いてファインチューニングを行った。unigram 用の平仮名 BERT に対しては unigram 用の BCCWJ による平仮名文の単語分かち書きデータを利用し、bigram 用の平仮名 BERT

に対しては bigram 用の BCCWJ による平仮名文の単語分かち書きデータを利用した。unigra 用、bigram 用に関わらず、ファインチューニングに利用した BCCWJ による平仮名文の単語分かち書きデータのデータ数は 24556 文である。BCCWJ による平仮名文の単語分かち書きデータのデータ数自体は 40928 文存在しているが 5 分割交差検定を行うのでファインチューニングに利用するデータ数は全体の 5 分の 3 である 24556 文となっている。また、40928 文の BCCWJ による平仮名文の単語分かち書きデータの内、5 分の 1 のデータを検証用データに利用し、残りの 5 分の 1 のデータがテストデータに利用されている。つまり、全体の 5 分の 3 のデータがファインチューニング用でターであり、全体の 5 分の 1 のデータが検証用データであり、全体の 5 分の 1 のデータがテスト用データとして利用されている。このファインチューニングにより 2 種類の平仮名 BERT 単語分割システムを作成した。

この実験を可視化した図を図 6.1 に示す。

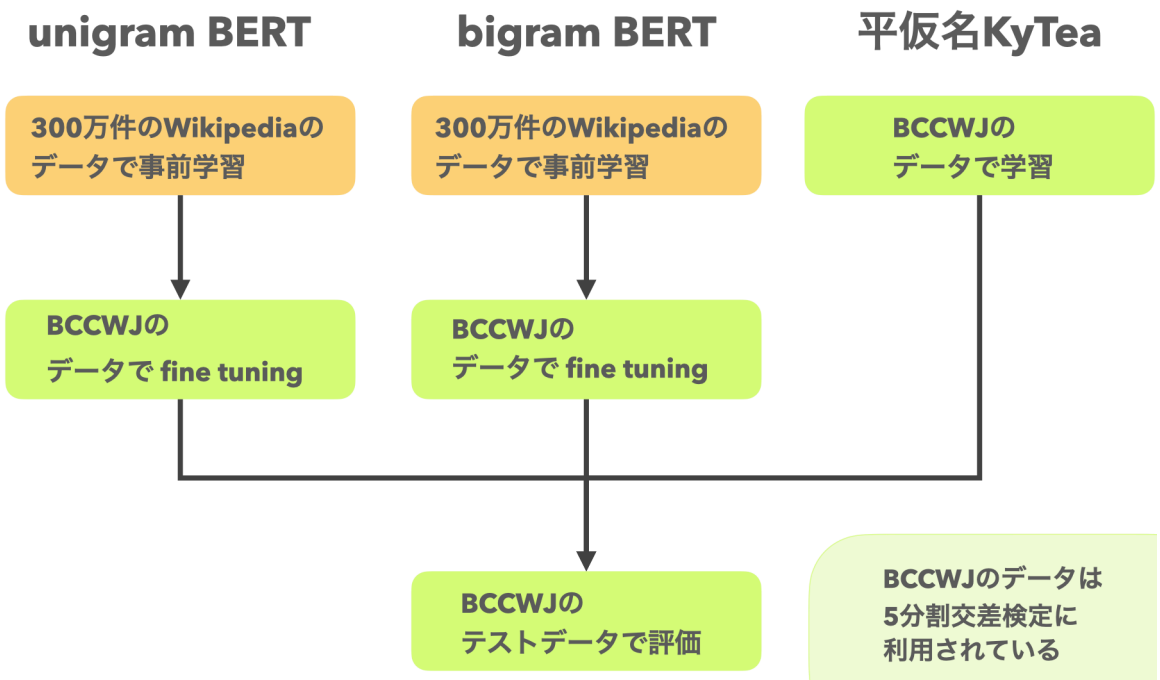

図 6.1: 実験 1：BCCWJ によるファインチューニングの実験

次に、本実験において BERT の事前学習で使用したパラメータを表 6.1 に、ファインチューニングで使用したパラメータを表 6.2 に示す。

表 6.1: 実験 1：BCCWJ によるファインチューニングの実験における事前学習のパラメータ

レイヤー数	12
隠れ層の次元数	120
ステップ数	1000000
学習率	1e-4
バッチサイズ	8

表 6.2: ファインチューニングにおけるパラメータ

ラベル数	12
学習率	1e-5
エポック数	50

6.2　実験 2：Wikipedia によるファインチューニングの実験

この実験では、疑似データである Wikipedia の単語分割情報を大量に利用した、3 つの単語分割システムの F 値を比較する。

300 万文の Wikipedia による事前学習用データを利用して平仮名 BERT を作成し、ファイチューニングのデータとして 100 万文の Wikipedia による平仮名文の単語分かち書きデータを利用して単語分割システムのファインチューニングを行った。なお、事前学習用のデータとファインチューニング用のデータの重複はない。一方で、実験 1 における事前学習データと実験 2 における事前学習データは同一のものを利用している。また、unigram BERT 単語分割システムおよび bigram BERT 単語分割システムに利用した 100 万文の Wikipedia による平仮名文の単語分かち書きデータを利用し、平仮名 KyTea 単語分割システムを作成し、評価した。テストデータには 40 万文の Wikipedia のデータと 40928 文の BCCWJ による平仮名文の単語分かち書きデータをそれぞれ利用した。テストデータに利用した Wikipedia のデータは、BERT の事前学習およびファインチューニング用の学習データと重複がないデータである。この実験を可視化した図を図 6.2 に示す。

実験 2 において BERT の事前学習で使用したパラメータおよびファインチューニング

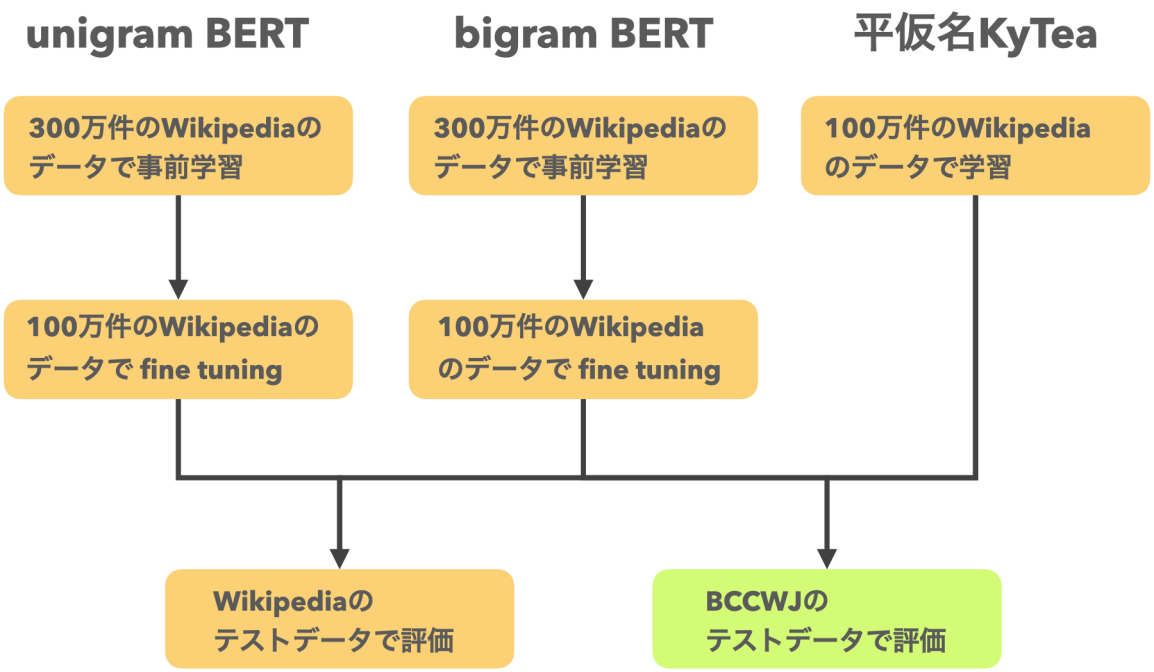

図 6.2: 実験 2：Wikipedia によるファインチューニングの実験

で使用したパラメータはどちらも epoch 数以外は実験 1 で利用したパラメータと同一である。実験 2 における epoch 数は 24 とした。なお、実験 2：Wikipedia によるファインチューニングの実験において作成される 2 種類の平仮名 BERT および 2 種類の平仮名 BERT 単語分割システムは、実験 1：BCCWJ によるファインチューニングの実験で作成された 2 種類の平仮名 BERT および 2 種類の平仮名 BERT 単語分割システムとは異なるものであることに留意されたい。

6.3　評価手法

unigram BERT 単語分割システムと bigram BERT 単語分割システムに対して、テストデータを入力として与えると上記 2 つのシステムはタグデータを出力する。出力されたタグの一致率を正答率として評価し、実際に単語分割として一致したものを適合率として評価する。さらに、文中に含まれる単語分割の内、各システムが正く出力することができた単語分割の割合を再現率として評価し、適合率と再現率の調和平均を F 値として評価する。

例えばテストデータとして「きょうはさむい」という文言である場合、単語区切りは「きょう/は/さむい」であるので、正解データは「1001100」である。このテストデータ

をシステムの入力として与えた時のシステムの出力が「1011101」であったとする。これはシステムがテストデータの単語区切りを「きょ/う/は/さむ/い」と理解したことを意味している。この時、正解率は正解データと出力データの 0 と 1 の一致率を見れば良いので正解率は 71.43% となる。また正解データに含まれる単語数は 3 つ（「きょう」「は」「さむい」の 3 つ）であり、システムが正しく分割できた単語数は 1 つ（「は」の部分のみ）であることから適合率は 33.33% となる。さらにシステムが出力したデータに含まれる単語数は 5 つ（「きょ」「う」「は」「さむ」「い」の 5 つ）であり、その中で正しく分割できている数は 1 つ（「は」の部分のみ）であることから再現率は 20% となる。そして F 値は適合率の 33.33% と再現率の 20% の調和平均なので 25% となる。

平仮名 KyTea 単語分割システムに関しては平仮名のみで構成された文字データを入力として与えるとモデルが推定する単語分割情報が出力される。出力された単語分割情報に対し単語分割の先頭を「1」に、それ以降を「0」と変換する。そして予め作成しておいたタグに対して正解率、適合率、再現率、F 値を評価する。

第 7 章

実験結果

7.1 実験 1：BCCWJ によるファインチューニングの実験結果

実験 1：BCCWJ によるファインチューニングの実験における各システムに対する 5 分割交差検定の正解率、適合率、再現率、F 値をまとめた表をそれぞれ、表 7.1、表 7.2、表 7.3、表 7.4 に示す。

表 7.1: 実験 1：BCCWJ によるファインチューニングの実験における各システムの正解率

	unigram BERT 単語分割システム	bigram BERT 単語分割システム	平仮名 KyTea 単語分割システム
1 / 5	97.43	96.64	97.07
2 / 5	97.61	96.93	**97.21**
3 / 5	**98.19**	97.16	96.68
4 / 5	97.50	**96.77**	91.13
5 / 5	97.97	97.38	97.06
平均	**97.74**	96.98	95.83

次に、実験 1：BCCWJ によるファインチューニングの実験における各システムに対する 5 分割交差検定における正解率、適合率、再現率、F 値の平均のみを表した表を表 7.5 に示す。以下、実験 1 に関しては平均のみを表した表 7.5 を参照しながら論述を行う。

表 7.2: 実験 1：BCCWJ によるファインチューニングの実験におけ各システムの適合率

	unigram BERT 単語分割システム	bigram BERT 単語分割システム	平仮名 KyTea 単語分割システム
1 / 5	93.44	91.64	92.15
2 / 5	93.79	92.05	**92.73**
3 / 5	**95.57**	93.19	91.98
4 / 5	94.00	92.36	85.48
5 / 5	94.99	**93.56**	92.30
平均	**94.36**	92.56	90.93

表 7.3: 実験 1：BCCWJ によるファインチューニングの実験における各システムの再現率

	unigram BERT 単語分割システム	bigram BERT 単語分割システム	平仮名 KyTea 単語分割システム
1 / 5	93.42	91.68	92.03
2 / 5	93.75	92.27	**92.64**
3 / 5	**95.38**	92.99	92.04
4 / 5	93.93	92.46	79.22
5 / 5	94.72	**93.57**	92.30
平均	**94.24**	92.60	88.56

表 7.5 から、unigram BERT 単語分割システムは平仮名 KyTea 単語分割システムと比較し、F 値が 4.64 point 向上していることが分かる。また bigram BERT 単語分割システムは平仮名 KyTea 単語分割システムと比較して F 値が 2.92 point 向上している。さらに unigram BERT 単語分割システムは bigram BERT 単語分割システムよりも F 値が 1.72 point 高いことがわかる。

表 7.4: 実験 1：BCCWJ によるファインチューニングの実験における各システムの F 値

	unigram BERT 単語分割システム	bigram BERT 単語分割システム	平仮名 KyTea 単語分割システム
1 / 5	93.43	91.66	92.09
2 / 5	93.77	92.16	**92.64**
3 / 5	**95.47**	93.09	92.04
4 / 5	93.96	92.41	79.22
5 / 5	94.85	**93.57**	92.30
平均	**94.30**	92.58	89.66

表 7.5: 実験 1 における各システムの結果

	unigram BERT 単語分割システム	bigram BERT 単語分割システム	平仮名 KyTea 単語分割システム
正解率	**97.74**	96.98	95.83
適合率	**94.36**	92.56	90.93
再現率	**94.24**	92.60	88.56
F 値	**94.30**	92.58	89.66

7.2　実験 2：Wikipedia によるファインチューニングの実験結果

実験 2：Wikipedia によるファインチューニングの実験の結果を表 7.6 に示す。

表 7.6 から unigram BERT 単語分割システムは、平仮名 KyTea 単語分割システムと比較し、Wikipedia をテストデータとした場合は F 値が 5.69point 向上し、BCCWJ のコアデータをテストデータとして利用した場合は F 値が 4.59point 向上していることが分かる。また bigram BERT 単語分割システムは、平仮名 KyTea 単語分割システムと比較し、Wikipedia をテストデータとした場合は F 値が 4.99point 向上し、BCCWJ をテストデータとして利用した場合には F 値が 3.77point 向上していることが分かる。さらに、unigram BERT 単語分割システムは bigram BERT 単語分割システムより高い F 値となった。F 値の差は、Wikipedia をテストデータとした場合は 0.70point であり、

表 7.6: 実験 2：Wikipedia によるファインチューニングの実験における各システムの結果

テストデータを Wikipedia にした場合			
	unigram BERT 単語分割システム	bigram BERT 単語分割システム	平仮名 KyTea 単語分割システム
正解率	**99.32**	99.08	97.17
適合率	**98.14**	97.41	92.83
再現率	**97.83**	97.15	91.76
F 値	**97.98**	97.28	92.29
テストデータを BCCWJ にした場合			
	unigram BERT 単語分割システム	bigram BERT 単語分割システム	平仮名 KyTea 単語分割システム
正解率	**95.65**	95.36	93.96
適合率	**90.85**	89.94	86.68
再現率	**86.67**	85.93	81.72
F 値	**88.71**	87.89	84.12

BCCWJ をテストデータとした場合は 0.82point であった。

第 8 章

考察

8.1 実験 1：BCCWJ によるファインチューニングの実験に対する考察

実験 1：BCCWJ によるファインチューニングの実験（表 7.5）より、作成した unigram BERT 単語分割システムおよび bigram BERT 単語分割システムの F 値は共にベースラインとなる平仮名 KeyTea 単語分割システムの F 値を上回った。特に平仮名 KeyTea の F 値に対する unigram BERT 単語分割システムの F 値の差は 4.64point であり、平仮名 KeyTea の F 値に対する bigram BERT 単語分割システムの F 値の差である 2.92point よりも 1.59 倍の結果向上を果たしている。

作成した 2 種類の平仮名 BERT 単語分割システムである unigram BERT 単語分割システムと bigram BERT 単語分割システムは共にファインチューニング時のデータとして BCCWJ コアデータのデータを利用している。その数は約 4 万分であり、これは平仮名 KeyTea 単語分割システムを構築する際の学習データと同一である。一方で 2 種類の平仮名 BERT 単語分割システムの精度が平仮名 KeyTea 単語分割システムの精度を上回った要因として 2 種類の平仮名 BERT 単語分割システムには事前学習が存在していたことが挙げられる。この事前学習には 100 万文の Wikipedia のデータが利用されており、平仮名 KeyTea 単語分割の構築にはこのような事前学習データが必要ないことから、この事前学習の存在が 2 平仮名 KeyTea 単語分割システムの精度を種類の平仮名 BERT 単語分割システムの精度上回った要因の 1 つであると考えられる。

8.2　実験 2：Wikipedia によるファインチューニングの実験に対する考察

実験 2：Wikipedia によるファインチューニングの実験（表 7.6）より、作成した unigram BERT 単語分割システムおよび bigram BERT 単語分割システムの F 値は共にベースラインとなる平仮名 KeyTea 単語分割システムの F 値を上回った。これはテストデータを Wikipedia にした場合であっても、BCCWJ にした場合でも同様であった。さらに、表 7.6 よりテストデータを BCCWJ にした場合よりも Wikipedia とした場合の方が各システムの結果が向上している事が分かる。これは実験 2 において各モデルを作成する際にデータを Wikipedia にしたことが要因であると考えられる。さらに、unigram BERT 単語分割システムと bigram BERT 単語分割システムに関しては実験 2 においてファインチューニングを行った際のデータも Wikipedia であったことからテストデータを Wikipedia にした際は高精度に単語分割ができている事が分かる。

8.3　文字種の差に着目した追加実験と考察

実験 1 と実験 2 の結果において、2 種類の平仮名 BERT 単語分割システムの F 値を比較すると、unigram BERT 単語分割システムの F 値がより良い結果であることが確認できる。bigram の方が unigram より情報量が多くなるため、我々は bigram BERT 単語分割システムの方が、unigram BERT 単語分割システムを上回ることを予想していたが、結果は逆であった。この理由としては、モデルの大きさに対応して必要になる学習データの差が考えられる。本研究において使用した平仮名 BERT の語彙数は unigram BERT 作成では 300 であり、bigram BERT 作成では 80956 であった。つまり語彙数に約 270 倍の差が存在する。その分、モデルは大きくなるため、必要な学習データも多くなると考えられる。ところが、2 種類の平仮名 BERT 単語分割システムにおける事前学習で利用したデータ数はどちらも 300 万文であった。つまり、モデルの大きさに必要な学習データ量に対して bigram BERT には十分な学習データではなかった可能性があり、それが unigram BERT 単語分割システムの F 値が bigram BERT の F 値を上回った要因であると考えられる。

語彙数に大幅な差があることに着目し、我々は、実験 1 のテストデータから記号等の文

字種を除いたデータを利用し、各システムの評価を再度算出した。実験 1 のテストデータから取り除かなかった文字種は、平仮名・片仮名・句点・読点・長音・符空白である。実験 1 のテストデータの各文に対して上記以外の文字種が含まれる文は評価せず、上記のみの文字種で構成された文を各システムに入力し、評価した。本追加実験における各システムに対する 5 分割交差検定の正解率、適合率、再現率、F 値をまとめた表をそれぞれ、表 8.1、表 8.2、表 8.3、表 8.4 に示す。

表 8.1: 実験 1：BCCWJ によるファインチューニングの実験においてテストデータから記号等の文字種を取り除いた場合における各システムの正解率

	unigram BERT 単語分割システム	bigram BERT 単語分割システム	平仮名 KyTea 単語分割システム
1 / 5	97.77	97.11	**97.4748**
2 / 5	98.04	97.44	97.69
3 / 5	**98.39**	97.51	**97.4721**
4 / 5	97.99	97.40	92.22
5 / 5	98.12	**97.59**	97.41
平均	**98.06**	97.41	96.45

表 8.2: 実験 1：BCCWJ によるファインチューニングの実験においてテストデータから記号等の文字種を取り除いた場合における各システムの適合率

	unigram BERT 単語分割システム	bigram BERT 単語分割システム	平仮名 KyTea 単語分割システム
1 / 5	94.21	92.63	93.07
2 / 5	94.82	93.25	**93.88**
3 / 5	**95.85**	93.53	93.74
4 / 5	95.07	93.55	86.94
5 / 5	95.24	**93.94**	93.11
平均	**95.04**	93.38	92.15

次に本追加実験において、正解率、適合率、再現率、F 値の平均のみを表した表を表 8.5 に示す。以下、追加実験に関する考察については表 8.5 を参照しながら論述を行う。

表 8.3: 実験 1：BCCWJ によるファインチューニングの実験においてテストデータから記号等の文字種を取り除いた場合における各システムの再現率

	unigram BERT 単語分割システム	bigram BERT 単語分割システム	平仮名 KyTea 単語分割システム
1 / 5	94.23	92.79	92.69
2 / 5	94.63	93.24	**93.51**
3 / 5	**95.73**	93.57	93.46
4 / 5	94.95	93.64	76.44
5 / 5	95.13	**94.02**	92.93
平均	**94.93**	93.44	89.81

表 8.4: 実験 1：BCCWJ によるファインチューニングの実験においてテストデータから記号等の文字種を取り除いた場合における各システムの F 値

	unigram BERT 単語分割システム	bigram BERT 単語分割システム	平仮名 KyTea 単語分割システム
1 / 5	94.22	92.71	92.88
2 / 5	94.72	93.24	**93.70**
3 / 5	**95.79**	93.57	93.60
4 / 5	95.01	93.64	81.35
5 / 5	95.18	**94.02**	93.02
平均	**94.99**	93.44	90.91

表 8.5 より文字種を制限することにより全体的に F 値が向上していることが確認できる。特に、unigram BERT 単語分割システムと bigram BERT 単語分割システムにおける F 値の差は 1.55point である。表 7.5 における 2 つの単語分割システムの F 値の差が 1.72 であったことから、テストデータにおける文字種を制限することで F 値の差が縮まっていることがわかる。

表 8.5: 実験 1 においてテストデータから記号を除いた場合の各システムの結果

	unigram BERT 単語分割システム	bigram BERT 単語分割システム	平仮名 KyTea 単語分割システム
正解率	**98.06**	97.41	96.45
適合率	**95.04**	93.38	92.15
再現率	**94.93**	93.50	89.81
F 値	**94.99**	93.44	90.91

8.4　全体考察

表 7.5 と表 7.6 より、実験 1 と実験 2 でそれぞれ作成した 2 つの平仮名 BERT 単語分割システムの F 値は、平仮名 KyTea 単語分割システムの F 値よりも高いことが確認できる。これにより、unigram BERT 単語分割システムと bigram BERT 単語分割システムが有効であるといえる。

次に、実験 1 と実験 2 の結果を比較する。BCCWJ をテストデータにした実験結果同士（表 7.5 と表 7.6 の BCCWJ の結果）を比較すると、実験 1 における 2 種類の平仮名 BERT 単語分割システムの F 値の方が高い。これは、実験 1 のファインチューニングのデータはテストデータと同じ BCCWJ であるが、実験 2 では Wikipeda のデータを利用しているためであると考えられる。特に、BCCWJ では正確な読みと単語分割の区切りの情報を利用しているが、Wikipedia は疑似データゆえに、Wikpiedia データの質は BCCWJ よりも低いと考えられる。実験 1 で利用した BCCWJ は約 4.5 万件であるのに対して実験 2 で利用した Wikipedia のデータは 100 万文であることを考えると、ファインチューニングにおける疑似データの量を増やしても、テストデータと同ドメインの正確なデータには及ばないことが分かる。

一方で、大量の Wikipedia の疑似データを与えた際、同 Wikipedia のテストデータに対する正解率は 99% を超える（表 7.6)。そのため、テストデータと同じドメインで、なおかつテストデータと整合性のある単語分割の情報をもつ大量のデータを利用してファインチューニングした場合には、かなり単語分割の評価値が高くなることが分かる。

最後に、本研究で利用した BERT 作成用の事前学習のデータ量は 300 万件であるが、この量を増やすことで、平仮名 BERT 単語分割システムの F 値が向上する可能性がある。この点に関しては今後の課題と考えている。

第 9 章

結論

本研究では、平仮名文に特化して学習した 2 種類の BERT を利用した文単語分割システム、unigram BERT 単語分割システムと bigram BERT 単語分割システムを作成した。このシステムは BERT の事前学習として、MeCab を利用して Wikipedia の平仮名文のデータから作成した文字 unigram または文字 bigram のデータを利用し、平仮名文の単語分かち書きのデータでファインチューニングを行うことで作成したものである。BCCWJ のコアデータを利用した五分割交差実験と、Wikipedia のデータを利用したファインチューニングによる実験において、平仮名文の単語分割の F 値は共に KyTea を用いた平仮名文単語分割システムの F 値を上回った。また、unigram BERT 単語分割システムと bigram BERT 単語分割システムの F 値を比較すると、unigram BERT 単語分割システムの方が F 値が向上した。これにはモデルの大きさに対する事前学習のデータ数が影響したと考えられる。

また、実験により、ファインチューニングに利用するデータは、大量のドメインの異なった疑似データよりも、少量のドメインの等しい、テストデータと整合性の取れたデータの方がよいことが分かった。

本研究は、情報処理学会第 253 回自然言語処理研究会で発表済みである [16]。また、言語処理学会第 29 回年次大会に投稿済みである [17]。

謝辞

指導教員である新納浩幸教授には、本研究を進めるにあたって多大な助言を頂きました。そのほかにも研究や論文に対して数多くのご指導をして頂きました。感謝申し上げます。

東京農工大学の古宮嘉那子准教授には大学 3 年次から大学 4 年生まで、茨城大学の古宮研究室にてご指導を頂きました。2021 年に茨城大学から東京農工大学に移られた後もなお、研究に対する助言や論文に対する多くのレビューを頂きました。ありがとうございました。

旧古宮研究室の先輩である平林さんには研究に必要とはる様々な技術や手法について何度もアドバイスを頂いたり、研究室のサーバを立てる際にご尽力頂きました。また、新納研究室の先輩である藤井さん、田中さんには研究室の運営に対する舵取りをして頂きました。それにより、新納研究室の中で不自由なく研究をさせていただく事ができました。お礼を申し上げます。

研究室の同級生である岸野さん、小林君にはオンラインやオフラインでのゼミを通して一緒に研究を進め、複数の学会および研究会に登壇する事ができました。また、私が 2021 年に新納研究室に移動になった際に同期となった伊藤君、内間君、平野君、築地君、菊田君、南濱さんには数多くのアドバイスを貰いました。ありがとうございました。

最後に、家族や友人を含め本研究に携わって頂いた全ての方々に感謝申し上げます。

参考文献

[1] 工藤拓, 市川宙, David Talbot, 賀沢秀人. Web 上のひらがな交じり文に頑健な形態素解析. 言語処理学会第 18 回年次大会発表論文集, pp. 1272–1275, 2012.

[2] 林聖人, 山村毅. ひらがな語の追加と形態素解析の精度についての考察析. 愛知県立大学情報科学部平成 28 年度卒業論文要旨, 2017.

[3] 井筒順, 明石陸, 加藤涼, 岸野望叶, 小林汰一郎, 金野佑太, 古宮嘉那子. Mecab による平仮名のみの形態素解析. 言語処理学会第 26 回年次大会発表論文集, pp. 65–69, 2020.

[4] Jun Izutsu and Kanako Komiya. Morphological analysis of japanese hiragana sentences using the bi-lstm crf model,. *10th International Conference on Natural Language Processing (NLP 2021)*, 2021.

[5] 森山柊平, 大野誠寛, 増田英孝, 絹川博之ほか. Recurrent neural network language model を用いたべた書きかな文の形態素解析. 情報処理学会論文誌, Vol. 59, No. 10, pp. 1911–1921, 2018.

[6] 森山柊平, 大野誠寛. Rnn とロジスティック回帰を用いた平仮名文の逐次的な形態素解析. 自然言語処理, Vol. 29, No. 2, pp. 367–394, 2022.

[7] 鈴木雅弘, 坂地泰紀, 和泉潔, 石川康. 金融文書を用いた追加事前学習言語モデルの構築と検証. 言語処理学会 第 28 回年次大会発表論文集, pp. 588–592, 2020.

[8] Taku Kudo. Mecab: Yet another part-of-speech and morphological analyzer. 2005. `https://taku910.github.io/mecab/`.

[9] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-

training of deep bidirectional transformers for language understanding. 2018.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *In Advances in Neural Information Processing Systems*, pp. 6000–60010, 2017.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. In *Neural computation*, pp. 1735–1780.

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. pmlr, 2015.

[14] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. 2016.

[15]「unidic」国語研短単位自動解析用辞書. `https://unidic.ninjal.ac.jp`.

[16] 井筒順, 古宮嘉那子, 新納浩幸. 平仮名 bert による平仮名文の分割. 研究報告自然言語処理（NL）, Vol. 2022-NL-253, No. 1, pp. 1–7, 2022.

[17] 井筒順, 古宮嘉那子, 新納. 平仮名 bert を用いた平仮名文の分割. 言語処理学会第 29 回年次大会発表論文集. to appear（2023）.

[18] Pre-training language models for japanese. `https://github.com/retarfi/language-pretraining`.

[19] 近江崇宏, 金田健太郎, 森長誠, 江間見亜利. Bert による自然言語処理入門: Transformers を使った実践プログラミング. ストックマーク株式会社, 2021.

付録

A　プログラム

まず、Wikipedia のデータを MeCab を利用して MeCab の出力形式に変換するプログラムを A.1 として以下に示す。

ソースコード A.1: Wikipedia のデータを MeCab を利用して MeCab の出力形式に変換するプログラム

```
1  import sys
2  import MeCab
3
4  inputData = open("wikiData_NoBrackets.txt", "r")
5  outputData = open("WikiToMecabCorpas_NoBrankets.txt", "w+")
6
7  mecab = MeCab.Tagger('-d␣unidic-cwj-2.3.0/')
8
9  limitCount = 0
10
11 for line in inputData:
12         mecablist = mecab.parse(line)
13         outputData.write(mecablist)
14
15         limitCount += 1
16         if limitCount >= 10000: break
17
18 inputData.close()
19 outputData.close()
```

次に、事前学習で利用する学習データを作成するプログラムを A.2 及び A.3 として以下に示す。

ソースコード A.2: 事前学習で利用する学習データを作成するプログラム

```
1  from pykakasi import kakasi
2
3  inputFile = open("WikiToMecabCorpas_NoBrankets.txt", "r")
4  output_all_File = open("intermediate.txt", "w+")
5  output_string_File = open("bigram_input.txt", "w+")
6  output_tag_File = open("bigram_input_tag.txt", "w+")
7
8  kakasi = kakasi()
9  kakasi.setMode('K', 'H')
10 conv = kakasi.getConverter()
11
12 count = 0
13 countOfRaw = 0
14 countOfLine = 0
15 countOfEOS = 0
16 countOfBrank = 0
17
18 oneLineStr = ""
19 oneLineTag = ""
20
21 maxLine = 3000000
22
23 for line in inputFile:
24     count += 1
25
26     if countOfLine >= maxLine: break
27     if line == "":
28         countOfBrank += 1
29         continue
30     if line == "EOS\n":
```

```
31          countOfEOS += 1
32          continue
33
34      countOfRaw += 1
35
36      split = line.split('\t')
37      pronunciationArray = split[1].split(',')
38      pronunce = pronunciationArray[0]
39      readingHiragana = split[0]
40
41      # は ipadic9, は bcc7
42      if (len(pronunciationArray) >= 9) and pronunce != "記号":
43          # は ipadic7, は bcc6
44          readingHiragana = conv.do(pronunciationArray[7])
45
46      # print(pronunce, readingHiragana)
47
48      if readingHiragana == "。":
49          oneLineStr += readingHiragana
50          output_all_File.write(oneLineStr)
51          output_all_File.write("\n")
52          output_string_File.write(oneLineStr)
53          output_string_File.write("\n\n")
54          oneLineTag += "1\n\n"
55          output_all_File.write(oneLineTag)
56          output_tag_File.write(oneLineTag)
57          oneLineStr = ""
58          oneLineTag = ""
59          countOfLine += 1
60          if countOfLine % 5000 == 0: print(countOfLine, 'completed
                ')
61          continue
62
63      strArray = list(readingHiragana)
```

```
64      for tmp in range(len(strArray)):
65          if tmp == 0:
66              oneLineStr += strArray[tmp]
67              oneLineTag += "1"
68          else:
69              oneLineStr += strArray[tmp]
70              oneLineTag += "0"
71
72
73  print(count)
74  print(countOfRaw)
75  print(countOfLine)
76  print(countOfEOS)
77  inputFile.close()
78  output_all_File.close()
79  output_string_File.close()
80  output_tag_File.close()
```

ソースコード A.3: 事前学習で利用する学習データを作成するプログラム

```
1   inputFile = open("bigram_input.txt", "r")
2   outputFile = open("bigram_output.txt", "w+")
3
4   CLS = '[CLS]␣' # 文頭
5   SEP = '␣[SEP]' # 文末
6
7   lineCount = 0
8
9   print("start␣bigram␣tokenizer")
10  for line in inputFile:
11      lineCount += 1
12      if lineCount % 5000 == 0: print(lineCount, 'completed')
13      if line == '\n':
14          outputFile.write(line)
15          continue
```

```
16
17      newLine = CLS
18      counter = -1 # をつけているので CLSから始める-1
19      for string in line:
20          if string == '\n':
21              newLine = newLine[:-2]
22              continue
23          counter += 1
24          if counter == 0:
25              newLine += string
26              continue
27          newLine += string + '␣' + string
28
29      newLine += SEP
30      newLine += '\n'
31      outputFile.write(newLine)
32
33  inputFile.close()
34  outputFile.close()
35
36  print("finish␣bigram␣tokenizer")
```

次に、ファインチューニングで利用する学習データ、検証用データ、及び、平仮名 BERT 単語分割システムの Wikipedia 用の評価データを作成するプログラムを A.4 として以下に示す。

ソースコード A.4: ファインチューニングで利用する学習データ、検証用データ、及び、平仮名 BERT 単語分割システムの評価用データを作成するプログラム

```
1  from pykakasi import kakasi
2
3  inputFile = open("WikiToMecabCorpas_NoBrankets.txt", "r")
4  output_all_File = open("fine_tuneing.txt", "w+")
5
6  output_val_file = open('tf_val.txt', 'w+')
7  output_test_file = open('tf_test.txt', 'w+')
```

```
8
9
10
11 kakasi = kakasi()
12 kakasi.setMode('K', 'H')
13 conv = kakasi.getConverter()
14
15 count = 0
16 countOfRaw = 0
17 countOfLine = 0
18 countOfEOS = 0
19 countOfBrank = 0
20
21 oneLineStr = ""
22 oneLineTag = ""
23
24 ftMaxLine = 1000000
25 valMaxLine = 600000
26 testMaxLine = 200000
27
28 ignoreLine = 10000000
29
30 # MARK: - FT
31 for line in inputFile:
32     # print(line)
33     count += 1
34     if ignoreLine >= count:
35         if count % 500000 == 0: print(count, 'ignore')
36         continue
37
38     if countOfLine >= ftMaxLine: break
39     if line == "":
40         countOfBrank += 1
41         continue
```

```
42      if line == "EOS\n":
43          countOfEOS += 1
44          continue
45
46      countOfRaw += 1
47
48      split = line.split('\t')
49      pronunciationArray = split[1].split(',')
50      pronunce = pronunciationArray[0]
51      readingHiragana = split[0]
52
53      # は ipadic9, は bcc7
54      if (len(pronunciationArray) >= 9) and pronunce != "記号":
55          # は ipadic7, は bcc6
56          readingHiragana = conv.do(pronunciationArray[7])
57
58      # print(pronunce, readingHiragana)
59
60      if readingHiragana == "。":
61          oneLineStr += readingHiragana
62          newLine = ""
63          isFirest = True
64          for string in oneLineStr:
65              if isFirest:
66                  newLine += string
67                  isFirest = False
68                  continue
69              newLine += string + '␣' + string
70          newLine = newLine[:-2]
71          output_all_File.write(newLine)
72          output_all_File.write("\n")
73          oneLineTag += "\n"
74          output_all_File.write(oneLineTag)
75          oneLineStr = ""
```

```
76          oneLineTag = ""
77          countOfLine += 1
78          if countOfLine % 5000 == 0: print(countOfLine, 'completed
                ')
79          continue
80
81      strArray = list(readingHiragana)
82      for tmp in range(len(strArray)):
83          if tmp == 0:
84              oneLineStr += strArray[tmp]
85              oneLineTag += "1"
86          else:
87              oneLineStr += strArray[tmp]
88              oneLineTag += "0"
89
90  print(count)
91  print(countOfRaw)
92  print(countOfLine)
93  print(countOfEOS)
94  output_all_File.close()
95
96  count = 0
97  countOfRaw = 0
98  countOfLine = 0
99  countOfEOS = 0
100 countOfBrank = 0
101
102 oneLineStr = ""
103 oneLineTag = ""
104
105
106 # MARK: - val
107 for line in inputFile:
108     # print(line)
```

```
109     count += 1
110     if ignoreLine + ftMaxLine >= count:
111         if count % 500000 == 0: print(count, 'ignore')
112         continue
113 
114     if countOfLine >= valMaxLine: break
115     if line == "":
116         countOfBrank += 1
117         continue
118     if line == "EOS\n":
119         countOfEOS += 1
120         continue
121 
122     countOfRaw += 1
123 
124     split = line.split('\t')
125     pronunciationArray = split[1].split(',')
126     pronunce = pronunciationArray[0]
127     readingHiragana = split[0]
128 
129     # は ipadic9, は bcc7
130     if (len(pronunciationArray) >= 9) and pronunce != "記号":
131         # は ipadic7, は bcc6
132         readingHiragana = conv.do(pronunciationArray[7])
133 
134     # print(pronunce, readingHiragana)
135 
136     if readingHiragana == "。":
137         oneLineStr += readingHiragana
138         newLine = ""
139         isFirest = True
140         for string in oneLineStr:
141             if isFirest:
142                 newLine += string
```

```
143                 isFirest = False
144                 continue
145             newLine += string + '␣' + string
146         newLine = newLine[:-2]
147         output_val_file.write(newLine)
148         output_val_file.write("\n")
149         oneLineTag += "\n"
150         output_val_file.write(oneLineTag)
151         oneLineStr = ""
152         oneLineTag = ""
153         countOfLine += 1
154         if countOfLine % 5000 == 0: print(countOfLine, 'completed
                ')
155         continue
156
157     strArray = list(readingHiragana)
158     for tmp in range(len(strArray)):
159         if tmp == 0:
160             oneLineStr += strArray[tmp]
161             oneLineTag += "1"
162         else:
163             oneLineStr += strArray[tmp]
164             oneLineTag += "0"
165
166
167 print(count)
168 print(countOfRaw)
169 print(countOfLine)
170 print(countOfEOS)
171 output_val_file.close()
172
173 count = 0
174 countOfRaw = 0
175 countOfLine = 0
```

```
176 countOfEOS = 0
177 countOfBrank = 0
178
179 oneLineStr = ""
180 oneLineTag = ""
181
182
183
184 # MARK: - test
185 for line in inputFile:
186     # print(line)
187     count += 1
188     if ignoreLine + ftMaxLine + valMaxLine >= count:
189         if count % 500000 == 0: print(count, 'ignore')
190         continue
191
192     if countOfLine >= testMaxLine: break
193     if line == "":
194         countOfBrank += 1
195         continue
196     if line == "EOS\n":
197         countOfEOS += 1
198         continue
199
200     countOfRaw += 1
201
202     split = line.split('\t')
203     pronunciationArray = split[1].split(',')
204     pronunce = pronunciationArray[0]
205     readingHiragana = split[0]
206
207     # は ipadic9, は bcc7
208     if (len(pronunciationArray) >= 9) and pronunce != "記号":
209         # は ipadic7, は bcc6
```

```
210         readingHiragana = conv.do(pronunciationArray[7])
211
212     # print(pronunce, readingHiragana)
213
214     if readingHiragana == "。":
215         oneLineStr += readingHiragana
216         newLine = ""
217         isFirest = True
218         for string in oneLineStr:
219             if isFirest:
220                 newLine += string
221                 isFirest = False
222                 continue
223             newLine += string + '␣' + string
224         newLine = newLine[:-2]
225         output_test_file.write(newLine)
226         output_test_file.write("\n")
227         oneLineTag += "\n"
228         output_test_file.write(oneLineTag)
229         oneLineStr = ""
230         oneLineTag = ""
231         countOfLine += 1
232         if countOfLine % 5000 == 0: print(countOfLine, 'completed
                ')
233         continue
234
235     strArray = list(readingHiragana)
236     for tmp in range(len(strArray)):
237         if tmp == 0:
238             oneLineStr += strArray[tmp]
239             oneLineTag += "1"
240         else:
241             oneLineStr += strArray[tmp]
242             oneLineTag += "0"
```

```
243
244
245 print(count)
246 print(countOfRaw)
247 print(countOfLine)
248 print(countOfEOS)
249 output_test_file.close()
250 inputFile.close()
```

次に、平仮名 BERT 単語分割システムを BCCWJ で 5 分割交差検定を行うための評価データを作成するプログラムを A.5 として以下に示す。

ソースコード A.5: 平仮名 BERT 単語分割システムを BCCWJ で 5 分割交差検定を行うための評価データを作成するプログラム

```
1  from pykakasi import kakasi
2  import re
3
4  kakasi = kakasi()
5  kakasi.setMode('K', 'H')
6  conv = kakasi.getConverter()
7
8  inputFile = open('core_SUW.txt', 'r+')
9  outputFile = open("bccwj_core_test_data.txt", "w+")
10
11 lineString = ""
12 tagString = ""
13 count = 0
14
15
16 def create(inputFile, outputFile):
17     lineString = ""
18     tagString = ""
19     lineCount = 0
20
21     for line in inputFile:
```

```
22          str = line.split('\t')[-1]
23
24          # 最後から番目に漢字が含まれているかを判定する 2
25          pattern = re.compile(r'^[\u4E00-\u9FD0]+$')
26          isContainsChineseStr = False
27          for string in line.split('\t')[-2]:
28              if pattern.match(string): isContainsChineseStr = True
29
30          # 漢字が含まれていなかったら最後から番目を採用する 2
31          # 含まれている場合は使えないので番最後のカタカナのを採用する 1
32          if isContainsChineseStr == False: str = line.split('\t')
                [-2]
33
34          wordClass = line.split('\t')[16].split("-")[0]
35          if wordClass == "補助記号": str = line.split('\t')[12]
36
37
38          # 「ヒコーキ」みたいなのは伸ばし棒を置換してヒコウキに帰る
39          # 漢字が含まれていた場合のみ通る
40          if 'ー' in str:
41              resutlStr = ""
42              coveredStr = line.split('\t')[-6]
43              for str1, str2 in zip(list(str), list(coveredStr)):
44                  if str1 == "ー":
45                      resutlStr += str2
46                  else:
47                      resutlStr += str1
48              str = resutlStr
49
50          # カタカナを平仮名に変換
51          str = conv.do(str)
52
53          for index, tmp in enumerate(list(str)):
54              if tmp == "\n": continue
```

```
55              # 全角文字を半角文字に変換
56              tmp = tmp.translate(str.maketrans({chr(0xFF01 + i):
                    chr(0x21 + i) for i in range(94)}))
57              lineString += tmp
58              if index == 0:
59                  tagString += "1"
60              else:
61                  tagString += "0"
62
63          if str != "。": continue
64          # if str != ？"": continue
65
66          bigramLine = ""
67          isFirest = True
68          for string in lineString:
69              if isFirest:
70                  bigramLine += string
71                  isFirest = False
72                  continue
73              bigramLine += string + '␣' + string
74          bigramLine = bigramLine[:-2]
75          # タグに関しては最後の「。」入らないので削除
76          # なので↓のような形になる
77          # ごららんんくくだささいい。
78          # 1001000
79          tagString = tagString[:-1]
80
81          if len(bigramLine) == 0: continue
82
83          # 「。」のみで構成されている場合はスルー
84          if bigramLine.strip('。␣') == '':
85              continue
86
87
```

```
88          bigramLine += '\n'
89          tagString += '\n'
90          outputFile.write(bigramLine)
91          outputFile.write(tagString)
92          print(bigramLine)
93          print(tagString)
94          lineString = ""
95          tagString = ""
96
97          lineCount += 1
98          # if count >= 100: break
99      print('行数', lineCount)
100
101 create(inputFile, outputFile)
102
103 inputFile.close()
104 outputFile.close()
```

次に、事前学習を行うプログラムを A.6 として以下に示す [18]。このプログラムを利用することにより、平仮名 BERT を作成することが可能となる。

ソースコード A.6: 事前学習を行うプログラム

```
1  import argparse
2  import datetime as dt
3  from fractions import Fraction
4  import json
5  import logging
6  import os
7  from typing import Tuple
8  import warnings
9
10 import torch
11 from torch.utils.data.dataset import Dataset
12 from tokenizers import BertWordPieceTokenizer,
       SentencePieceBPETokenizer, normalizers
```

```
13 from tokenizers.processors import BertProcessing
14 import transformers
15 from transformers.data.data_collator import
       DataCollatorForLanguageModeling
16 from transformers import (
17     BertConfig,
18     BertForPreTraining,
19     ElectraConfig,
20     PreTrainedModel,
21     TrainingArguments
22 )
23 transformers.logging.set_verbosity_info()
24 transformers.logging.enable_explicit_format()
25
26 import utils
27 TrainingArguments._setup_devices = utils._setup_devices
28
29 warnings.simplefilter('ignore', UserWarning)
30 assert any(v in torch.__version__ for v in ['1.8.0', '1.8.1', '
       1.8.2', '1.9.0']), f'This␣file␣is␣only␣guranteed␣with␣
       pytorch␣1.9.0,␣but␣this␣is␣{torch.__version__}'
31 assert transformers.__version__ in ['4.7.0'], f'This␣file␣is␣
       only␣guranteed␣with␣transformers␣4.7.0,␣but␣this␣is␣{
       transformers.__version__}'
32
33
34 def load_tokenizer(tokenizer_dir:str, max_length:int,
35                    tokenizer_type:str,
36                    mecab_dic_type:str,
37                    ) -> transformers.tokenization_utils_base.
                           PreTrainedTokenizerBase:
38     # load tokenizer
39     if tokenizer_type=='sentencepiece':
40         tokenizer = SentencePieceBPETokenizer(
```

```
41              os.path.join(tokenizer_dir, "vocab.json"),
42              os.path.join(tokenizer_dir, "merges.txt"),
43              unk_token="[UNK]",
44              add_prefix_space=False, # 文頭に自動でスペースを追加しない
45          )
46          # 改行がにあるとも改行がついてくるので input_filetokenstrip
47          # cf. https://github.com/huggingface/tokenizers/issues/231
48          tokenizer.normalizer = normalizers.Sequence([
49              normalizers.Strip(),
50              normalizers.NFKC()
51          ])
52          # post process tokenizer
53          tokenizer._tokenizer.post_processor = BertProcessing(
54              ("[SEP]", tokenizer.token_to_id("[SEP]")),
55              ("[CLS]", tokenizer.token_to_id("[CLS]")),
56          )
57          tokenizer.enable_truncation(max_length = max_length)
58          # convert to transformers style
59          tokenizer = transformers.PreTrainedTokenizerFast(
60              tokenizer_object = tokenizer,
61              model_max_length = max_length,
62              unk_token = "[UNK]",
63              sep_token = "[SEP]",
64              pad_token = "[PAD]",
65              cls_token = "[CLS]",
66              mask_token = "[MASK]",
67          )
68      elif tokenizer_type==’wordpiece’:
69          tokenizer = transformers.BertJapaneseTokenizer(
70              os.path.join(tokenizer_dir, "vocab.txt"),
71              do_lower_case = False,
72              word_tokenizer_type = "mecab",
```

```
73              subword_tokenizer_type = "wordpiece",
74              tokenize_chinese_chars = False,
75              mecab_kwargs = {'mecab_dic': mecab_dic_type},
76              model_max_length = max_length
77          )
78      else:
79          raise ValueError(f'Invalid␣tokenizer_type␣{tokenizer_type
                ␣}.')
80
81      return tokenizer
82
83
84  def make_dataset_model_bert(
85          tokenizer:transformers.tokenization_utils_base.
                PreTrainedTokenizerBase,
86          input_file:str,
87          param_config:dict,
88          overwrite_cache:bool,
89      ) -> Tuple[Dataset, PreTrainedModel]:
90
91      dataset = utils.TextDatasetForNextSentencePrediction(
92          tokenizer = tokenizer,
93          file_path = input_file,
94          overwrite_cache = overwrite_cache,
95          block_size = param_config['sequence-length'],
96          short_seq_probability = 0.1, # default
97          nsp_probability = 0.5, # default
98      )
99      bert_config = BertConfig(
100         vocab_size = tokenizer.vocab_size,
101         hidden_size = param_config['hidden-size'],
102         num_hidden_layers = param_config['number-of-layers'],
103         num_attention_heads = param_config['attention-heads'],
104         intermediate_size = param_config['ffn-inner-hidden-size'
```

```
            ],
105         max_position_embeddings = param_config['sequence-length'
            ],
106     )
107     model = BertForPreTraining(config=bert_config)
108 
109     return dataset, model
110 
111 
112 def make_dataset_model_electra(
113         tokenizer:transformers.tokenization_utils_base.
            PreTrainedTokenizerBase,
114         input_file:str,
115         param_config:dict,
116         overwrite_cache:bool,
117     ) -> Tuple[Dataset, PreTrainedModel]:
118 
119     dataset = utils.LineByLineTextDataset(
120         tokenizer = tokenizer,
121         file_path = input_file,
122         overwrite_cache = overwrite_cache,
123         block_size = param_config['sequence-length'],
124     )
125     frac_generator = Fraction(param_config['generator-size'])
126     config_generator = ElectraConfig(
127         vocab_size = tokenizer.vocab_size,
128         embedding_size = param_config['embedding-size'],
129         hidden_size = int(param_config['hidden-size'] *
            frac_generator),
130         num_attention_heads = int(param_config['attention-heads']
             * frac_generator),
131         num_hidden_layers = param_config['number-of-layers'],
132         intermediate_size = int(param_config['ffn-inner-hidden-
            size'] * frac_generator),
```

```
133     )
134     config_discriminator = ElectraConfig(
135         vocab_size = tokenizer.vocab_size,
136         embedding_size = param_config['embedding-size'],
137         hidden_size = param_config['hidden-size'],
138         num_attention_heads = param_config['attention-heads'],
139         num_hidden_layers = param_config['number-of-layers'],
140         intermediate_size = param_config['ffn-inner-hidden-size'
                ],
141     )
142     model = utils.ElectraForPretrainingModel(
143         config_generator = config_generator,
144         config_discriminator = config_discriminator,
145     )
146 
147     return dataset, model
148 
149 
150 def run_pretraining(
151         tokenizer:transformers.tokenization_utils_base.
                PreTrainedTokenizerBase,
152         input_file:str,
153         model_name:str,
154         model_dir:str,
155         fp16_type:int,
156         param_config:dict,
157         do_whole_word_mask:bool,
158         do_continue:bool,
159         overwrite_cache:bool,
160         node_rank:int,
161         local_rank:int,
162         run_name:str
163     ) -> None:
164     if run_name == '':
```

```
165         run_name = dt.datetime.now().strftime("%y%m%d") + "_" +
                os.path.basename(os.path.dirname(model_dir))
166     os.makedirs(model_dir, exist_ok=True)
167
168     # training argument
169     if do_continue:
170         training_args = torch.load(os.path.join(model_dir, "
                training_args.bin"))
171         per_device_train_batch_size = training_args.
                per_device_train_batch_size
172     else:
173         if torch.cuda.device_count() > 0:
174             per_device_train_batch_size = int(param_config['batch
                    -size'][str(node_rank)] / torch.cuda.device_count
                    ())
175         else:
176             per_device_train_batch_size = param_config['batch-
                    size'][str(node_rank)]
177     # initialize
178     training_args = transformers.TrainingArguments(
179         output_dir = model_dir,
180         do_train = True,
181         do_eval = False, # default
182         per_device_train_batch_size = per_device_train_batch_size
                ,
183         learning_rate = param_config['learning-rate'],
184         adam_beta1 = 0.9, # same as BERT paper
185         adam_beta2 = 0.999, # same as BERT paper
186         adam_epsilon = 1e-6,
187         weight_decay = 0.01, # same as BERT paper
188         warmup_steps = param_config['warmup-steps'],
189         logging_dir = os.path.join(os.path.dirname(__file__), f"
                runs/{run_name}"),
190         save_steps = param_config['save-steps'] if 'save-steps'
```

```
            in param_config.keys() else 50000, #default:500
191         save_strategy = "steps", # default:"steps"
192         logging_steps = param_config['logging-steps'] if 'logging
                -steps' in param_config.keys() else 5000, # default
                :500
193         save_total_limit = 20, # optional
194         seed = 42, # default
195         fp16 = bool(fp16_type!=0),
196         fp16_opt_level = f"O{fp16_type}",
197         #:"O1":Mixed Precision (recommended for typical use), "O2
                ":Almost " FP16 Mixed Precision, "O3":FP16 training
198         disable_tqdm = True,
199         max_steps = param_config['train-steps'],
200         gradient_accumulation_steps = 1 if 'accumulation-steps'
                not in param_config.keys() else param_config['
                accumulation-steps'],
201         dataloader_num_workers = 3,
202         dataloader_pin_memory=False,
203         local_rank = local_rank,
204         report_to = "tensorboard"
205     )
206     if not do_continue:
207         if local_rank != -1:
208             if torch.cuda.device_count() > 0:
209                 training_args.per_device_train_batch_size = int(
                        param_config['batch-size'][str(node_rank)] /
                        torch.cuda.device_count())
210             else:
211                 training_args.per_device_train_batch_size =
                        param_config['batch-size'][str(node_rank)]
212         torch.save(training_args, os.path.join(model_dir, "
                training_args.bin"))
213
214     # dataset and model
```

```
215     if model_name == 'bert':
216         train_dataset, model = make_dataset_model_bert(tokenizer,
                input_file, param_config, overwrite_cache)
217     elif model_name == 'electra':
218         train_dataset, model = make_dataset_model_electra(
                tokenizer, input_file, param_config, overwrite_cache)
219     logger.info('Dataset␣was␣complete.')
220
221     # data collator
222     if model_name == 'bert':
223         mlm_probability = 0.15
224     elif model_name == 'electra':
225         mlm_probability = param_config['mask-percent']/100
226     if do_whole_word_mask:
227         if model_name == 'bert':
228             data_collator = utils.DataCollatorForWholeWordMask(
229                 tokenizer = tokenizer,
230                 mlm = True,
231                 mlm_probability = mlm_probability
232             )
233         elif model_name == 'electra':
234             # https://github.com/google-research/electra/issues
                    /57
235             data_collator = utils.DataCollatorForWholeWordMask(
236                 tokenizer = tokenizer,
237                 mlm = True,
238                 mlm_probability = mlm_probability,
239                 rate_replaced = 0.85,
240                 rate_random = 0,
241                 rate_unchanged = 0.15
242             )
243     else:
244         if model_name == 'bert':
245             data_collator = DataCollatorForLanguageModeling(
```

```
246                 tokenizer = tokenizer,
247                 mlm = True,
248                 mlm_probability = mlm_probability
249             )
250         elif model_name == 'electra':
251             data_collator = utils.
                    DataCollatorForLanguageModelingWithElectra(
252                 tokenizer = tokenizer,
253                 mlm = True,
254                 mlm_probability = mlm_probability
255             )
256     logger.info('Datacollator was complete.')
257
258     trainer = utils.MyTrainer(
259         model = model,
260         args = training_args,
261         data_collator = data_collator,
262         train_dataset = train_dataset,
263         node_rank = node_rank
264     )
265     trainer.batch_config = param_config['batch-size']
266     trainer.real_batch_size = sum(param_config['batch-size'].
            values())
267
268     logger.info('Pretraining starts.')
269     resume_from_checkpoint = True if do_continue else None
270     trainoutput = trainer.train(resume_from_checkpoint=
            resume_from_checkpoint)
271
272
273 if __name__ == "__main__":
274     # arguments
275     parser = argparse.ArgumentParser()
276     parser.add_argument('--input_file', type=str, required=True)
```

```
277     parser.add_argument('--tokenizer_dir', type=str, required=
            True)
278     parser.add_argument('--model_dir', type=str, required=True)
279     parser.add_argument('--parameter_file', type=str, required=
            True)
280     parser.add_argument('--model_type', type=str, required=True)
281     parser.add_argument('--fp16_type', type=int, default=0,
            choices=[0,1,2,3],
282                                     help='default:0(disable),␣
                                            see␣https://nvidia.
                                            github.io/apex/amp.html␣
                                            for␣detail')
283     parser.add_argument('--tokenizer_type', type=str, choices=['
            sentencepiece', 'wordpiece'])
284     parser.add_argument('--mecab_dic_type', type=str, default='',
             choices=['', 'unidic_lite', 'unidic', 'ipadic'])
285     parser.add_argument('--run_name', type=str, default='')
286     parser.add_argument('--do_whole_word_mask', action='
            store_true')
287     parser.add_argument('--do_continue', action='store_true')
288     parser.add_argument('--disable_overwrite_cache', action='
            store_true')
289     parser.add_argument('--node_rank', type=int, default=-1)
290     parser.add_argument('--local_rank', type=int, default=-1)
291
292     args = parser.parse_args()
293     assert not (args.tokenizer_type=='sentencepiece' and args.
            do_whole_word_mask), 'Whole␣Word␣Masking␣cannot␣be␣
            applied␣with␣sentencepiece␣tokenizer'
294
295     # get root logger
296     # logging.basicConfig(format='%(asctime)s %(message)s',
            datefmt='%Y/%m/%d %H:%M:%S', level=logging.INFO)
297     logger = logging.getLogger(__name__)
```

```
298     sh = logging.StreamHandler()
299     sh.setLevel(logging.INFO)
300     formatter = logging.Formatter(
301         fmt='%(asctime)s %(levelname)s: %(message)s',
302         datefmt='%Y/%m/%d %H:%M:%S'
303     )
304     sh.setFormatter(formatter)
305     logger.addHandler(sh)
306
307     # parameter configuration
308     with open(args.parameter_file, 'r') as f:
309         param_config = json.load(f)
310     if args.model_type not in param_config:
311         raise KeyError(f'{args.model_type} not in parameters.json
                ')
312     if 'electra-' in args.model_type.lower():
313         model_name = 'electra'
314     elif 'bert-' in args.model_type.lower():
315         model_name = 'bert'
316     else:
317         raise ValueError('Argument model_type must contain
                electra or bert')
318     param_config = param_config[args.model_type]
319     set_assert = {
320         'number-of-layers', 'hidden-size', 'sequence-length', '
                ffn-inner-hidden-size', 'attention-heads',
321         'warmup-steps', 'learning-rate', 'batch-size', 'train-
                steps'
322     }
323     if model_name == 'electra':
324         set_assert = set_assert | set(['embedding-size', '
                generator-size', 'mask-percent'])
325     if param_config.keys() < set_assert:
326         raise ValueError(f'{set_assert-param_config.keys()} is(
```

```
            are)␣not␣in␣parameter_file')
327     if str(args.local_rank) not in param_config['batch-size'].
            keys():
328         raise ValueError(f'local_rank␣{args.local_rank}␣is␣not␣
                defined␣in␣batch-size␣of␣parameter_file')
329     logger.info(f'Config[{args.model_type}]␣is␣loaded')
330
331     # tokenizer
332     tokenizer = load_tokenizer(
333         tokenizer_dir = args.tokenizer_dir,
334         tokenizer_type = args.tokenizer_type,
335         max_length = param_config['sequence-length'],
336         mecab_dic_type = args.mecab_dic_type,
337     )
338     run_pretraining(
339         tokenizer = tokenizer,
340         input_file = args.input_file,
341         model_name = model_name,
342         model_dir = args.model_dir,
343         fp16_type = args.fp16_type,
344         param_config = param_config,
345         do_whole_word_mask = args.do_whole_word_mask,
346         do_continue = args.do_continue,
347         overwrite_cache = not args.disable_overwrite_cache,
348         node_rank = args.node_rank,
349         local_rank = args.local_rank,
350         run_name = args.run_name,
351     )
```

次にファインチューニングを行い平仮名 BERT 単語分割システムを作成するプログラムを A.7 として以下に示す [19]。このプログラムを実行することで、平仮名 BERT 単語分割システムを作成することができる。

ソースコード A.7: ファインチューニングを行い平仮名 BERT 単語分割システムを作成するプログラム

```
import itertools
import random
import json
from tqdm import tqdm
import numpy as np
import unicodedata

import torch
from torch.utils.data import DataLoader
from transformers import BertJapaneseTokenizer,
    BertForTokenClassification, BertConfig, BertTokenizer
import pytorch_lightning as pl


MODEL_NAME = 'model/bert/checkpoint-1000000/pytorch_model.bin'
config = BertConfig.from_json_file('model/bert/checkpoint
    -1000000/config.json')


class BertForTokenClassification_pl(pl.LightningModule):

    def __init__(self, model_name, num_labels, lr):
        super().__init__()
        self.save_hyperparameters()

        self.bert_tc = BertForTokenClassification.from_pretrained
            (
            model_name,
            config=config
        )

    def training_step(self, batch, batch_idx):
        output = self.bert_tc(**batch)
```

```
31          loss = output.loss
32          self.log('train_loss', loss)
33          return loss
34
35      def validation_step(self, batch, batch_idx):
36          output = self.bert_tc(**batch)
37          val_loss = output.loss
38          self.log('val_loss', val_loss)
39
40      def configure_optimizers(self):
41          return torch.optim.Adam(self.parameters(), lr=self.
                hparams.lr)
42
43  def load_dataset():
44      file = open("share/tokenizer/fine_tuning.txt", "r")
45      datset_tokens = []
46      dataset_labels = []
47      tokenLine = True
48      for line in file:
49          if tokenLine:
50              datset_tokens.append(line.rstrip('\n').split('␣'))
51          else:
52              dataset_labels.append(list(map(int, line.rstrip('\n')
                    )))
53          tokenLine = not tokenLine # toggle
54      return datset_tokens, dataset_labels
55
56  def load_dataset_val():
57      file = open("share/tokenizer/tf_val.txt", "r")
58      datset_tokens = []
59      dataset_labels = []
60      tokenLine = True
61      for line in file:
62          if tokenLine:
```

```
63              datset_tokens.append(line.rstrip('\n').split('␣'))
64          else:
65              dataset_labels.append(list(map(int, line.rstrip('\n')
                    )))
66          tokenLine = not tokenLine # toggle
67      return datset_tokens, dataset_labels
68
69  print('ファイルからデータの読み込み開始')
70  dataset_tokens, dataset_labels = load_dataset()
71  dataset_tokens_val, dataset_labels_val = load_dataset_val()
72
73
74  def create_dataset(tokenizer, tokens_array, labels_array,
        max_length):
75      dataset_for_loader = []
76      count = 0
77      for (tokens, labels) in zip(tokens_array, labels_array):
78          count += 1
79          if count % 10000 == 0: print('create_dataset:␣', count)
80          # 符号化を行いに入力できる形式にする。BERT
81          input_ids = tokenizer.convert_tokens_to_ids(tokens)
82          encoding = tokenizer.prepare_for_model(
83              input_ids,
84              max_length=max_length,
85              padding='max_length',
86              truncation=True
87          ) # をに変換 input_idsencoding
88          # 特殊トークン [CLS、][SEPのラベルをにする。]0
89          labels = [0] + labels[:max_length - 2] + [0]
90          # 特殊トークン [PADのラベルをにする。]0
91          labels = labels + [0] * (max_length - len(labels))
92          encoding['labels'] = labels
93
94          encoding = {k: torch.tensor(v) for k, v in encoding.items
```

```
            ()}
95          dataset_for_loader.append(encoding)
96
97      return dataset_for_loader
98
99  # トークナイザのロード
100 tokenizer = BertTokenizer('share/tokenizer/vocab.txt',
        do_lower_case=False, do_basic_tokenize=False)
101
102 # データセットの作成
103 max_length = 256
104 dataset_train_for_loader = create_dataset(
105     tokenizer, dataset_tokens, dataset_labels, max_length
106 )
107 dataset_val_for_loader = create_dataset(
108     tokenizer, dataset_tokens_val, dataset_labels_val, max_length
109 )
110
111 # データローダの作成
112 dataloader_train = DataLoader(
113     dataset_train_for_loader, batch_size=32, shuffle=True
114 )
115 dataloader_val = DataLoader(dataset_val_for_loader, batch_size
        =256)
116
117 checkpoint = pl.callbacks.ModelCheckpoint(
118     monitor='val_loss',
119     mode='min',
120     save_top_k=1,
121     save_weights_only=True,
122     dirpath='model/'
123 )
124
125 trainer = pl.Trainer(
```

```
126     gpus=1,
127     max_epochs=50,
128     callbacks=[checkpoint]
129 )
130
131 model = BertForTokenClassification_pl(
132     MODEL_NAME,
133     num_labels=2,
134     lr=1e-5
135 )
136 trainer.fit(model, dataloader_train, dataloader_val)
137 best_model_path = checkpoint.best_model_path
```

次に平仮名 BERT 単語分割システムの性能の評価を行うプログラムを A.8 として以下に示す。

ソースコード A.8: 平仮名 BERT 単語分割システムの性能の評価を行うプログラム

```
1  import itertools
2  import random
3  import json
4  from tqdm import tqdm
5  import numpy as np
6  import unicodedata
7
8  import torch
9  from torch.utils.data import DataLoader
10 from transformers import BertJapaneseTokenizer,
       BertForTokenClassification, BertConfig, BertTokenizer
11 import pytorch_lightning as pl
12 from transformers import BertTokenizer, BertModel, BertConfig,
       BertForMaskedLM
13 from transformers import BertConfig, BertTokenizer,
       BertForTokenClassification
14 from transformers import pipeline
15
```

```
16
17 class BertForTokenClassification_pl(pl.LightningModule):
18
19     def __init__(self, model_name, num_labels, lr):
20         super().__init__()
21         self.save_hyperparameters()
22
23         self.bert_tc = BertForTokenClassification.from_pretrained
               (
24             model_name,
25             config=config
26         )
27
28     def training_step(self, batch, batch_idx):
29         output = self.bert_tc(**batch)
30         loss = output.loss
31         self.log('train_loss', loss)
32         return loss
33
34     def validation_step(self, batch, batch_idx):
35         output = self.bert_tc(**batch)
36         val_loss = output.loss
37         self.log('val_loss', val_loss)
38
39     def configure_optimizers(self):
40         return torch.optim.Adam(self.parameters(), lr=self.
               hparams.lr)
41
42 def load_test_dataset():
43     file = open("share/bccwj_test_data/bccwj_core_test_data.txt",
           "r")
44     datset_tokens = []
45     dataset_labels = []
46     tokenLine = True
```

```
47      for line in file:
48          if tokenLine:
49              datset_tokens.append(line.rstrip('\n').split('␣'))
50          else:
51              dataset_labels.append(list(map(int, line.rstrip('\n')
                    )))
52          tokenLine = not tokenLine # toggle
53      return datset_tokens, dataset_labels
54
55  print('ファイルからテストデータの読み込み開始')
56  dataset_tokens, dataset_labels = load_test_dataset()
57
58  best_model_path = 'model/resultOfBigramBert/epoch=23-step
        =750000.ckpt'
59
60  max_length = 128
61
62  def predict(tokens_array, labels_array, tokenizer, bert_tc):
63      num_of_correct = 0
64      num_of_incorrect = 0
65      num_of_all_correct_line = 0
66      num_of_line = 0
67
68      N = 0
69      n0 = 0
70      n1 = 0
71
72      for (tokens, labels) in zip(tokens_array, labels_array):
73          if len(tokens) >= 512:
74              continue
75
76          input_ids = tokenizer.convert_tokens_to_ids(tokens)
77          encoding = tokenizer.prepare_for_model(
78              input_ids,
```

```
79              padding='max_length' if max_length else False,
80              truncation=True if max_length else False
81          ) # をに変換 input_idsencoding
82
83          encoding = { k: torch.tensor([v]) for k, v in encoding.
                items() }
84          encoding = { k: v.cuda() for k, v in encoding.items() }
85
86          # ラベルの予測値の計算
87          with torch.no_grad():
88              output = bert_tc(**encoding)
89              scores = output.logits
90              labels_predicted = scores[0].argmax(-1).cpu().numpy()
                    .tolist()
91
92          labels_predicted.pop(-1)
93          labels_predicted.pop(0)
94
95          if len(labels) != len(labels_predicted):
96              continue
97
98          num_of_line += 1
99          if num_of_line % 5000 == 0: print('prediction␣Line',
                num_of_line)
100         all_correct = True
101         for (label, label_predicted) in zip(labels,
                labels_predicted):
102             if label == label_predicted:
103                 num_of_correct += 1
104             else:
105                 num_of_incorrect += 1
106                 all_correct = False
107
108         N += labels.count(1)
```

```
109         n0 += labels_predicted.count(1)
110
111
112         joinedArray = [x + y for (x, y) in zip(labels,
                labels_predicted)] # つの配列を加算
                2
113         joinedArray = [tmp for tmp in joinedArray if tmp != 0] #
                を削除して左つめ
                0
114
115         for index, targetNum in enumerate(joinedArray):
116             if index + 1 == len(joinedArray): break
117             # が連続している時に 2を加算 n1
118             if targetNum == 2 and joinedArray[index + 1] == 2: n1
                    += 1
119
120         if joinedArray[-1] == 2: n1 += 1 # 配列の最後がのときも 2を
                加算 n1
121
122
123
124
125
126         if all_correct: num_of_all_correct_line += 1
127
128     p = n1 / n0
129     r = n1 / N
130     f = 2 * p * r / (p + r)
131
132     print('\n--------')
133     print('試行した全ての行数:,', num_of_line)
134     print('全一致した行数    :', num_of_all_correct_line)
135     print('N␣␣␣␣:', N)
136     print('n0␣␣␣:', n0)
137     print('n1␣␣␣:', n1)
138     print('適合率:', p)
```

```
139     print('再現率:', r)
140     print('値 F␣␣:', f)
141     return num_of_correct, num_of_incorrect
142
143
144
145 tokenizer = BertTokenizer('share/tokenizer/vocab.txt',
        do_lower_case=False, do_basic_tokenize=False)
146 config = BertConfig.from_json_file('model/bert/checkpoint
        -1000000/config.json')
147
148
149 model = BertForTokenClassification_pl.load_from_checkpoint(
150     best_model_path
151 )
152
153 bert_tc = model.bert_tc.cuda()
154 num_of_correct, num_of_incorrect = predict(dataset_tokens,
        dataset_labels, tokenizer, bert_tc)
155
156 print('試行回数:', num_of_incorrect + num_of_correct)
157 print('正解数  :', num_of_correct)
158 print('不正解数:', num_of_incorrect)
159 print('正解率
        :', num_of_correct / (num_of_correct + num_of_incorrect))
```

次に、平仮名 KyTea 単語分割システムの学習データ、及び、テストデータの作成を行うプログラムを A.9 として以下に示す。ただし、A.9 は実験 1：BCCWJ によるファインチューニングの実験で作成した平仮名 KyTea 単語分割システムの学習データ、及び、テストデータである。

ソースコード A.9: 実験 1：BCCWJ によるファインチューニングの実験で作成した平仮名 KyTea 単語分割システムの学習データ、及び、テストデータを作成するプログラム

```
1 from pykakasi import kakasi
2 import re
```

```
3
4 kakasi = kakasi()
5 kakasi.setMode('K', 'H')
6 conv = kakasi.getConverter()
7
8 inputFile = open('../core_SUW.txt', 'r+')
9 # トレーニング用に使う。
10 # 行数はなので「」「」「」「」「」で分割すればお 40928818681868186818581851k
11 # 1 →「」1
12 # 2 →「」8187
13 # 3 →「」16373
14 # 4 →「」24559
15 # 5 →「」32744
16 outputFile1 = open("kytea_traning_1.txt", "w+")
17 outputFile2 = open("kytea_traning_2.txt", "w+")
18 outputFile3 = open("kytea_traning_3.txt", "w+")
19 outputFile4 = open("kytea_traning_4.txt", "w+")
20 outputFile5 = open("kytea_traning_5.txt", "w+")
21 # テスト用に使う。入力はな文なので形は普通。Plain
22 outputTextFile1 = open("kytea_test_text_1.txt", "w+")
23 outputTextFile2 = open("kytea_test_text_2.txt", "w+")
24 outputTextFile3 = open("kytea_test_text_3.txt", "w+")
25 outputTextFile4 = open("kytea_test_text_4.txt", "w+")
26 outputTextFile5 = open("kytea_test_text_5.txt", "w+")
27 # テスト用に使う。↑を入力しての出力のが↓に合ってるかを確かめる。BERT01
28 outputTagFile1 = open("kytea_test_tag_1.txt", "w+")
29 outputTagFile2 = open("kytea_test_tag_2.txt", "w+")
30 outputTagFile3 = open("kytea_test_tag_3.txt", "w+")
31 outputTagFile4 = open("kytea_test_tag_4.txt", "w+")
32 outputTagFile5 = open("kytea_test_tag_5.txt", "w+")
33
34 originLineString = ""
35 lineString = ""
36 tagString = ""
```

```
count = 0

originLineString = ""
lineString = ""
tagString = ""
lineCount = 0

for line in inputFile:
    str = line.split('\t')[-1]

    # 最後から番目に漢字が含まれているかを判定する 2
    pattern = re.compile(r'^[\u4E00-\u9FD0]+$')
    isContainsChineseStr = False
    for string in line.split('\t')[-2]:
        if pattern.match(string): isContainsChineseStr = True

    # 漢字が含まれていなかったら最後から番目を採用する 2
    # 含まれている場合は使えないので番最後のカタカナのを採用する 1
    if isContainsChineseStr == False: str = line.split('\t')[-2]

    wordClass = line.split('\t')[16].split("-")[0]
    if wordClass == "補助記号": str = line.split('\t')[12]

    # 「ヒコーキ」みたいなのは伸ばし棒を置換してヒコウキに帰る
    # 漢字が含まれていた場合のみ通る
    if 'ー' in str:
        resutlStr = ""
        coveredStr = line.split('\t')[-6]
        for str1, str2 in zip(list(str), list(coveredStr)):
            if str1 == "ー":
                resutlStr += str2
            else:
                resutlStr += str1
        str = resutlStr
```

```
71
72      # カタカナを平仮名に変換
73      str = conv.do(str)
74
75      isFirst = True
76      for index, tmp in enumerate(list(str)):
77          if tmp == "\n": continue
78          # 全角文字を半角文字に変換
79          tmp = tmp.translate(str.maketrans({chr(0xFF01 + i): chr(0
                x21 + i) for i in range(94)}))
80          originLineString += tmp
81          if isFirst == True and tmp == '/':
82              lineString += '\\' + tmp
83              isFirst = False
84          else:
85              if tmp == '&':
86                  lineString += '\\' + tmp
87              else:
88                  lineString += tmp
89
90          if index == 0:
91              tagString += "1"
92          else:
93              tagString += "0"
94
95      lineString += '␣'
96      if str != "。": continue
97      # if str != ?"": continue
98
99      lineString = lineString[:-1]
100
101     if len(lineString) == 0: continue
102
103     # 「。」のみで構成されている場合はスルー
```

```
104     if lineString.strip('。␣') == '':
105         continue
106 
107     # 先頭にスラッシュ文字（）があるとエラーになる/
108     # なぜなら文字と認識しないから。通常のコーパスは以下の通り
109     # 「コーパスの助詞/ 文ぶん// です語尾/ 。。」//
110     # 先頭の文字とスラッシュの認識ができないのでバックスラッシュをつけて
            あげる
111 
112     # 「」もバックスラッシュを付与する&
113     originLineString = originLineString.replace('&', '\\&')
114 
115     originLineString += '\n'
116     lineString += '\n'
117     tagString += '\n'
118     # 1 → 「」1
119     # 2 → 「」8186
120     # 3 → 「」16372
121     # 4 → 「」24558
122     # 5 → 「」32745
123     if lineCount <= 8186:
124         outputTextFile1.write(originLineString)
125         outputFile1.write(lineString)
126         outputTagFile1.write(tagString)
127     elif lineCount <= 16372:
128         outputTextFile2.write(originLineString)
129         outputFile2.write(lineString)
130         outputTagFile2.write(tagString)
131     elif lineCount <= 24558:
132         outputTextFile3.write(originLineString)
133         outputFile3.write(lineString)
134         outputTagFile3.write(tagString)
135     elif lineCount <= 32745:
136         outputTextFile4.write(originLineString)
```

```
137         outputFile4.write(lineString)
138         outputTagFile4.write(tagString)
139     else:
140         outputTextFile5.write(originLineString)
141         outputFile5.write(lineString)
142         outputTagFile5.write(tagString)
143     print(originLineString)
144     print(lineString)
145     print(tagString)
146     originLineString = ""
147     lineString = ""
148     tagString = ""
149 
150     lineCount += 1
151     # if count >= 100: break
152 print('行数', lineCount)
```

また、実験 2：Wikipedia によるファインチューニングの実験で作成した平仮名 KyTea 単語分割システムの学習データ、及び、テストデータを作成するプログラムを A.10 として以下に示す。

ソースコード A.10: 実験 2：Wikipedia によるファインチューニングの実験で作成した平仮名 KyTea 単語分割システムの学習データ、及び、テストデータを作成するプログラム

```
1 from pykakasi import kakasi
2 
3 inputFile = open("WikiToMecabCorpas_NoBrankets.txt", "r")
4 output_all_File = open("kytea_traning.txt", "w+")
5 
6 output_test_file = open('kytea_test.txt', 'w+')
7 output_tag_file = open('kytea_test_tag.txt', 'w+')
8 
9 kakasi = kakasi()
10 kakasi.setMode('K', 'H')
```

```
conv = kakasi.getConverter()

count = 0
countOfRaw = 0
countOfLine = 0
countOfEOS = 0
countOfBrank = 0

oneLineStr = ""

ftMaxLine = 1000000
valMaxLine = 600000
testMaxLine = 200000

ignoreLine = 10000000

for line in inputFile:

    count += 1
    if ignoreLine >= count:
        if count % 500000 == 0: print(count, 'ignore')
        continue

    if countOfLine >= ftMaxLine: break
    if line == "":
        countOfBrank += 1
        continue
    if line == "EOS\n":
        countOfEOS += 1
        continue

    countOfRaw += 1

    split = line.split('\t')
```

```
45      pronunciationArray = split[1].split(',')
46      pronunce = pronunciationArray[0]
47      readingHiragana = split[0]
48
49
50      if (len(pronunciationArray) >= 9) and pronunce != "記号":
51          readingHiragana = conv.do(pronunciationArray[7])
52
53
54      if readingHiragana == "。":
55          oneLineStr += readingHiragana
56          output_all_File.write(oneLineStr)
57          output_all_File.write("\n")
58          oneLineStr = ""
59          countOfLine += 1
60          if countOfLine % 5000 == 0: print(countOfLine, 'completed
                ')
61          continue
62
63      # 先頭にスラッシュ文字（）があるとエラーになる/
64      # なぜなら文字と認識しないから。通常のコーパスは以下の通り
65      # 「コーパスの助詞/ 文ぶん// です語尾/ 。。」//
66      # 先頭の文字とスラッシュの認識ができないのでバックスラッシュをつけて
            あげる
67      if list(readingHiragana)[0] == '/':
68          readingHiragana = '\\' + readingHiragana
69
70      # 「」もバックスラッシュを付与する&
71      readingHiragana = readingHiragana.replace('&', '\\&')
72      oneLineStr += readingHiragana
73      oneLineStr += "␣"
74
75  print(count)
76  print(countOfRaw)
```

```
77 print(countOfLine)
78 print(countOfEOS)
79 output_all_File.close()
80
81 count = 0
82 countOfRaw = 0
83 countOfLine = 0
84 countOfEOS = 0
85 countOfBrank = 0
86 oneLineStr = ""
87 oneLineTag = ""
88
89
90 # MARK: - val
91 for line in inputFile:
92     # print(line)
93     count += 1
94     if ignoreLine + ftMaxLine >= count:
95         if count % 500000 == 0: print(count, 'ignore')
96         continue
97
98     if countOfLine >= valMaxLine: break
99     countOfLine += 1
100
101 count = 0
102 countOfRaw = 0
103 countOfLine = 0
104 countOfEOS = 0
105 countOfBrank = 0
106 oneLineStr = ""
107 oneLineTag = ""
108
109
110 # MARK: - test
```

```
111 for line in inputFile:
112     count += 1
113     if ignoreLine + ftMaxLine + valMaxLine >= count:
114         if count % 500000 == 0: print(count, 'ignore')
115         continue
116 
117     if countOfLine >= testMaxLine: break
118     if line == "":
119         countOfBrank += 1
120         continue
121     if line == "EOS\n":
122         countOfEOS += 1
123         continue
124 
125     countOfRaw += 1
126 
127     split = line.split('\t')
128     pronunciationArray = split[1].split(',')
129     pronunce = pronunciationArray[0]
130     readingHiragana = split[0]
131 
132 
133     if (len(pronunciationArray) >= 9) and pronunce != "記号":
134         readingHiragana = conv.do(pronunciationArray[7])
135 
136     if readingHiragana == "。":
137         oneLineStr += readingHiragana
138         oneLineTag += "1"
139         output_test_file.write(oneLineStr)
140         output_test_file.write("\n")
141         oneLineTag += "\n"
142         output_tag_file.write(oneLineTag)
143         oneLineStr = ""
144         oneLineTag = ""
```

```
145         countOfLine += 1
146         if countOfLine % 5000 == 0: print(countOfLine, 'completed
                ')
147         continue
148
149     oneLineStr += readingHiragana
150     strArray = list(readingHiragana)
151     for tmp in range(len(strArray)):
152         if tmp == 0:
153             oneLineTag += "1"
154         else:
155             oneLineTag += "0"
156
157
158 print(count)
159 print(countOfRaw)
160 print(countOfLine)
161 print(countOfEOS)
162 output_test_file.close()
163 output_tag_file.close()
164 inputFile.close()
```

最後に、平仮名 KyTea 単語分割システムの性能の評価を行うプログラムを A.11 として以下に示す。

ソースコード A.11: 平仮名 KyTea 単語分割システムの性能の評価を行うプログラム

```
1
2
3
4 def load_test_dataset():
5     resultFile = open('result.txt', 'r')
6     tagFile = open('kytea_test_tag.txt')
7
8     dataset_tag = []
9     dataset_result = []
```

```
10
11      for line, tagLine in zip(resultFile, tagFile):
12          dataset_tag.append(list(map(int, tagLine.rstrip('\n'))))
13          # print(line)
14          newLine = line.replace('/UNK/UNK/UNK/UNK', '').replace('/
                UNK/UNK/UNK', '').rstrip('\n')
15          newLine = newLine.replace('、、/', '、
                ').replace('\&', '&').replace('\//', '\\').replace('、
                =/', '=')
16          newLine = newLine.replace(' 「」/', ' 「').replace('」
                「/␣', '」 ')
17          newLine = newLine.replace('〇△/', '〇').replace('"
                /"␣', '" ').replace(']/[␣', ']').replace('』\␣', '\\')
18          newLine = newLine.replace('\,', '\\').replace('、
                :/', ':').replace('-/-', '-')
19          newLine = newLine.replace('+/-/,:</UNK/UNK', '+').replace
                (' 、
                \/\'', '/')
20          newLine = newLine.replace('ℓ\/', '\\').replace(' 、
                /\'', ' ')
21          newLine = newLine.replace('－/-', '-')
22          newLine = newLine.replace('" ""
                \\'', '\\').replace(')\\\\', ')')
23          newLine = newLine.replace('http:\\//', 'http://')
24          stringArray = newLine.split('␣')
25          result = []
26          # print(newLine)
27          for str in stringArray:
28              if str == "、、/": str = "、"
29              if str == '\&': str = "&"
30              if str == '\\/': str = '\\'
31              # print(str)
32              isFirst = True
33              for tmp in str:
34                  if isFirst:
35                      result.append(1)
36                      isFirst = False
37                  else:
```

```
38                      result.append(0)
39          dataset_result.append(result)
40      return dataset_result, dataset_tag
41
42  print('ファイルからテストデータの読み込み開始')
43  dataset_result, dataset_tag = load_test_dataset()
44
45  num_of_correct = 0
46  num_of_incorrect = 0
47  num_of_all_correct_line = 0
48  num_of_line = 0
49
50  N = 0
51  n0 = 0
52  n1 = 0
53
54  for (tokens, labels) in zip(dataset_result, dataset_tag):
55
56      num_of_line += 1
57      if num_of_line % 5000 == 0: print('prediction␣Line',
            num_of_line)
58      all_correct = True
59
60      N += labels.count(1)
61      n0 += tokens.count(1)
62
63      for (label, label_predicted) in zip(labels, tokens):
64          if label == label_predicted:
65              num_of_correct += 1
66          else:
67              num_of_incorrect += 1
68              all_correct = False
69
70      joinedArray = [x + y for (x, y) in zip(labels, tokens)] # つ
```

```
        の配列を加算 2
71      joinedArray = [tmp for tmp in joinedArray if tmp != 0] # を削
        除して左つめ 0
72
73      for index, targetNum in enumerate(joinedArray):
74          if index + 1 == len(joinedArray): break
75          if targetNum == 2 and joinedArray[index + 1] == 2: n1 +=
                1
76
77      if joinedArray[-1] == 2: n1 += 1
78
79      if all_correct: num_of_all_correct_line += 1
80
81
82  p = n1 / n0
83  r = n1 / N
84  f = 2 * p * r / (p + r)
85
86
87  print('\n--------')
88  print('全ての行数    :,', num_of_line)
89  print('全一致した行数:', num_of_all_correct_line)
90  print('N␣␣␣␣␣␣:', N)
91  print('n0␣␣␣␣␣:', n0)
92  print('n1␣␣␣␣␣:', n1)
93  print('試行回数:', num_of_incorrect + num_of_correct)
94  print('正解数  :', num_of_correct)
95  print('不正解数:', num_of_incorrect)
96  print('正解率
        :', num_of_correct / (num_of_correct + num_of_incorrect))
97  print('適合率␣␣:', p)
98  print('再現率  :', r)
99  print('値 F␣␣␣␣:', f)
```