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

疑似訓練データを用いた同形異音語の読み推定

所属　　情報工学専攻

著者　小林汰一郎 (21NM724L)

指導教員　　新納浩幸教授

令和 5 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

疑似訓練データを用いた同形異音語の読み推定

著者

小林汰一郎 (21NM724L)

指導教員

新納浩幸教授

論文要旨

日本語には読みに曖昧性を持つ単語が多数存在する．例えば「辛い」は「カライ」のほかに「ツライ」と読むこともできる．このような単語を同形異音語と呼ぶ．日本語話者であればこのような単語の読み分けは容易だが, 日本語を母語としない者やコンピュータにとっての難易度は高い．

本論文では, 自然言語処理モデルである BERT(Bidirectional Encoder Representations from Transformers) を用いて同形異音語の読み推定を行う．本実験は全単語を対象とした系列ラベリング問題として行った．系列ラベリングとは, 文のような順序のあるデータが与えられた際に, 各要素 (単語) を対象に, 逐次的にラベル (読みなどの情報) を与える手法である．訓練・テストデータには現代日本語書き言葉均衡コーパス (BCCWJ) と日本語話し言葉コーパス (CSJ) を利用した．BCCWJ の大半を占める非コアデータの読みは, 形態素解析システム MeCab により機械的に割り振られたものである．また, BCCWJ は書き言葉であり, CSJ は話し言葉なので, ドメインのずれが想定される．CSJ のような話し言葉をターゲット領域としたとき, 通常はこの領域の訓練事例を用いて読み推定のモデルを学習・構築すればよいが, 訓練事例の構築コストが高いという問題がある．本研究では自動的に付与されたドメイン外の大量の疑似データ (BCCWJ のデータ) を利用することで, 本来必要としたターゲットの領域の訓練事例の量を大幅に削減することができた．具体的には, テストデータと同じ領域である CSJ のデータ量を 1-shot にまで減らしたモデルの正解率が 97.50% であり, CSJ のデータ全体を用いて学習したモデルの正解率は 97.97% であった．

Master's Thesis in Scholastic 2022, Major in Computer and Information Sciences,
Granduate School of Science and Engineering, Ibaraki University

Pronunciation Estimation of Homographs Using Pseudo-Training Data

Author : Taichiro Kobayashi (21NM724L)

Adviser : Prof. Hiroyuki Shinnou

Abstract

Japanese has many words with ambiguous pronunciations. For example, the word 「辛い」 can be read as "tsurai" in addition to "karai". Such words are called homographs. Although it is easy for Japanese speakers to read such words, it is very difficult for non-native speakers of Japanese or computers.

In this paper, we use BERT(Bidirectional Encoder Representations from Transformers), a natural language processing model, to estimate the pronunciation of homographs. This experiment was conducted as a sequence labeling task for all words. Sequence labeling is a method of assigning a label (pronunciation and other information) sequentially to each element in the sequential data, say each word in sentences, we used the Balanced Corpus of Contemporary Written Japanese (BCCWJ) and the Corpus of Spontaneous Japanese (CSJ) as training and test data. The pronunciations of non-core data, which make up the majority of BCCWJ, are mechanically assigned by the morphological analyzer, MeCab. In addition, since BCCWJ is a written language and CSJ is a spoken language, a domain shift is assumed. When the spoken language like CSJ is the target domain, it is usually sufficient to learn and build a model of pronunciation estimation using the training data in this domain, but the problem is that the cost of building the training data is high. By using a large amount of out-domain pseudo data (data from BCCWJ) , whose labels were the automatically assigned, the amount of training data in the target domain was greatly reduced. Specifically, the accuracy of the one-shot model of the CSJ was 97.50% and the accuracy of the model trained with the entire data in the CSJ was 97.97%.

目次

第 1 章　序論 6

第 2 章　BERT を利用した系列ラベリング 7

2.1　単語のベクトル化 7

2.2　BERT 11

2.3　系列ラベリング問題 13

第 3 章　関連研究 18

3.1　同形異音語の読み推定の関連研究 18

3.2　全単語対象の関連研究 18

3.3　疑似データを用いることの関連研究 19

第 4 章　疑似データを用いた同形異音語の読み推定 20

4.1　現代日本語書き言葉均衡コーパス 20

4.2　日本語話し言葉コーパス 21

4.3　疑似データを用いることの意義 22

4.4　モデル 22

第 5 章　実験 24

5.1　データの整形 24

5.2　実験設定 29

第 6 章　結果 31

第 7 章　考察 34

第 8 章　結論 38

参考文献 40

付録 42

第 1 章

序論

日本語には同じ表記でも違う読みをする単語が存在する．このような単語を同形異音語という．例として「辛い」は「カライ」だけでなく「ツライ」と読むこともできる．日本語話者であれば文脈から読み分けをすることは容易だが, 日本語を母語としない者やコンピュータにとっては困難である．

これまでに筆者らは同形異音語の読み推定を行ってきた．先行研究 [1] では SVM(Support Vector Machine) を用い, データセットとして現代日本語書き言葉均衡コーパス (BCCWJ) [2](4.1 節参照), 素性として one-hot ベクトル (2.1.1 節参照), nwjc2vec [3], BERT [4] による単語分散表現を用いた．また, 先行研究 [5] では事前学習モデル BERT を用いて, 全単語を対象に読み推定を行った．

本稿では, BERT を用いて全単語を対象に読み推定を行う．既存研究 [5] との違いは, 自動的にアノテーションされた疑似データを大量に用いることで人手データを 1-shot に留め, モデルの構築コストを抑えたことである．このようにして作成されたモデルが, 人手データを大量に使用して学習したモデルの精度に匹敵することを確認した．

第 2 章

BERT を利用した系列ラベリング

本稿で行った読み推定タスクは系列ラベリング問題と呼ばれる問題の一種である．そこで, 本章では系列ラベリング問題の説明を行う．まず, 系列ラベリング問題を解く際に用いる単語のベクトル化について説明し, その後, 今回の研究で用いた BERT というモデルについて述べる．最後に系列ラベリング問題について説明する．

2.1 単語のベクトル化

単語のベクトル化には大きく分けて 2 種類存在する．one-hot ベクトルと単語分散表現である．本章では上記の 2 種類のベクトル化手法について述べる．

2.1.1 one-hot ベクトル

one-hot ベクトルとは, 対応する素性のみを 1 とし, その他の要素が 0 であるベクトルのことである．そのため, 疎なベクトルになることが多い．本節では例を用いて one-hot ベクトルの作成手順を説明する．

one-hot ベクトル作成手順は以下の通りである．

1. 文を単語毎に分ける
2. 分けられた単語に, 1 から順に数値 i を割り当てる
3. 以後, 数値 i を割り当てられた単語が出現した場合, その単語を, 要素 i 番目のみを 1 としたベクトルへと変換する
4. 1 文毎にベクトルを連結する

以下の 2 つの例文を用いて上記の手順を説明する.

例文 1　「私は猫と犬が好き」

例文 2　「あなたは犬が好きではない」

まず, 2 つの例文を単語毎に分けると以下のようになる.

例文 1　「私/は/猫/と/犬/が/好き」

例文 2　「あなた/は/犬/が/好き/で/は/ない」

このように分割された文には合計 10 種類の単語が含まれている (「私」「は」「猫」「と」「犬」「が」「好き」「あなた」「で」「ない」). そこで, 出現した順に整数を与えていくと以下のようになる

私　1
は　2
猫　3
と　4
犬　5
が　6
好き　7
あなた　8
で　9
ない　10

次に, それぞれの単語を, 上で与えた数値 i 番目のみが 1 となるベクトルに変換していくと以下のようになる.

私　(1000000000)
は　(0100000000)
猫　(0010000000)
と　(0001000000)
犬　(0000100000)
が　(0000010000)

好き　(0000001000)

あなた　(0000000100)

で　(0000000010)

ない　(0000000001)

最後に, 作成したベクトルを文毎に連結すると以下のようになる.

例文 1　「私は猫と犬が好き」 (1111111000)

例文 2　「あなたは犬が好きではない」 (0100111111)

上記のような手順で作成されたベクトルが one-hot ベクトルである. 作成手順を見ると分かるように, ベクトル化したい文の数が多くなればなるほど, ベクトルの次元数も大きくなる. 先行研究 [1] では, 同形異音語の周辺 6 単語を対象に one-hot ベクトルを作成したが, その次元数は 162,314 であった.

2.1.2　単語分散表現

単語分散表現とは, 単語を数百次元程度の低次元のベクトルへ変換する技術の総称であり, 2013 年に Mikolov らが提案した word2vec[*1]というモデルが始祖である. word2vec には単語の加減算が可能であるという特徴がある. 例えば”king”を表すベクトルから”man”を表すベクトルを引き, ”woman”を表すベクトルを足す. こうすることで”queen”を表すベクトルが得られる. これは, 図 2.1 のように,

$$v(king) - v(man) = v(queen) - v(woman)$$

であるためである. ここで, *v(w)* は, 単語 w の位置ベクトルを表す. その他にも, 国と首都, 形容詞の原型と比較級と最上級, 数字と序数などにも同様の関係が見られる. このように, 単語間で構成性を持つのが大きな特徴である.

日本語版の単語分散表現モデルには, 国立国語研究所 (NINJAL:National Institute for Japanese Language and Linguistics) が開発した nwjc2vec [3] がある. nwjc2vec は国語研ウェブコーパス (NWJC:NINJAL Web Japanese Corpus) [6] から学習された単語分散表現データであり, word2vec を基礎として開発された.

word2vec や nwjc2vec では, 同じ表記を持つ単語は同じ分散表現へと変換される. 例

[*1] urlhttps://github.com/svn2github/word2vec

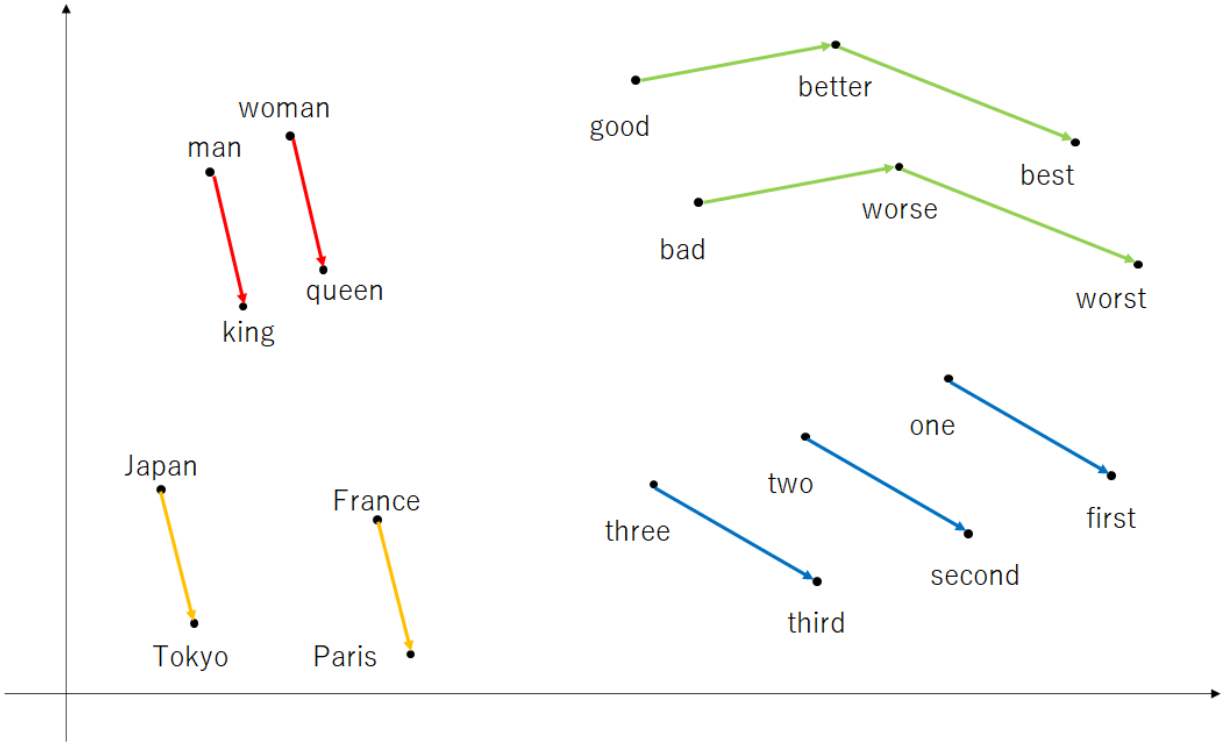

図 2.1: word2vec のイメージ図

えば, 多義語でも同じベクトルに変換される．これは同形異音語でも同様であるため, word2vec や nwjc2vec を同形異音語に適用して, 読み推定タスクに用いることはできない．ただし, 同形異音語の周辺単語にこれらを用いることは可能である．先行研究 [1] では, 同形異音語の周辺 6 単語に nwjc2vec を適用して分類実験を行った．例えば, 読みに曖昧性を持つ単語「辛い」を含む文「私はその辛いカレーを食べた」と「あなたはその辛い出来事を忘れられない」があったとする．これらを単語毎に区切るとそれぞれ「私/は/その/辛い/カレー/を/食べ/た」と「あなた/は/その/辛い/出来事/を/忘れ/られ/ない」となる．このとき, nwjc2vec を「辛い」に適用すると, 2 文とも同じベクトルが出力されてしまう．しかし, 「辛い」以外の単語 (辛いの周辺単語) にこれを適用することで, 単語と一対一対応である nwjc2vec のような単語分散表現モデルを, 読み推定タスクに利用できる．上記の例文だと, 1 文目の「辛い」の周辺 6 単語「私」「は」「その」「カレー」「を」「食べ」と 2 文目の「辛い」の周辺 6 単語「あなた」「は」「その」「出来事」「を」「忘れ」に nwjc2vec を適用することで, 2 文から別のベクトルが得られる．なお, 周辺単語を 6 単語に絞ったのは「単語の読みは周辺数単語に依存し, 同じ文中でも遠くに存在する単語は読みに寄与しない」と仮定したからある．先行研究 [1] では, このように nwjc2vec を用い

ることで実験を行った.

2.2　BERT

BERT [4] は”Bidirectional Encoder Representations from Transformers”の略称で, Jacob Devlin らが 2018 年に発表した単語分散表現及びそれを利用したモデルである. BERT の大きな特徴として, 文脈に依存した出力を行うことが挙げられる. これは, 同じ単語でも文脈によって違う単語分散表現を出力するということである. 単語毎に一対一対応の単語分散表現を持つモデル (2.1.2 節で述べた word2vec や nwjc2vec のようなモデル) では, 同形異音語をベクトル化することで読み推定を行うことはできない. これは, 本稿で読み推定を行った対象 (同形異音語) 同士が同じ表記であるためである. 例えば「辛いカレー」と「辛い修行」には同形異音語である「辛い」が含まれるが, 前者は「カライ」, 後者は「ツライ」と読む. しかし, 表記はどちらも「辛い」であるため, モデルによっては同じ単語分散表現となってしまう. 先行研究 [1] では, 同形異音語の周辺単語をベクトル化することでこの問題を解決した. 本研究では同形異音語自身をベクトル化して分類を行うため, BERT を用いて読み推定を行った.

BERT が文脈を考慮できる理由は, その事前学習方法にある. BERT の事前学習方法には 2 種類あり, それぞれ MLM(Masked Language Model) と NSP(Next Sentence Prediction) と呼ばれている. MLM とは, 文の一部を [MASK] トークンに置き換え, そのマスクされた単語を当てるというタスクである. 例文「私はバイクに乗ることが趣味であり, 休日は友人とツーリングに行く。」という文中の「バイク」を [MASK] トークンに置き換えた場合を図 2.2 に示す. この例文であれば「乗る」や「ツーリング」という周辺の単語から「バイク」という単語を予測する. このようなタスクが MLM である.

次に, NSP について説明する. NSP では, 入力された 2 文が意味的につながりを持つか否かを判定する. 2 つの例文「私は高校の時イギリスへ [MASK] したことがある。」「そこで [MASK] と英語の勉強をした。」を入力した場合を図 2.3 に, 「私は高校の時イギリスへ [MASK] したことがある。」「猫は [MASK] の哺乳類である。」を入力した例を図 2.4 に示す.

上記の 2 つの手法を用いて事前学習させることで, 他のモデルと比べて, BERT は文脈を考慮できると考えられている. また, 大量のデータを用いて事前学習しているため, 実際に BERT を利用する際には, 少量のデータを追加的に学習 (=fine-tuning) させるだけ

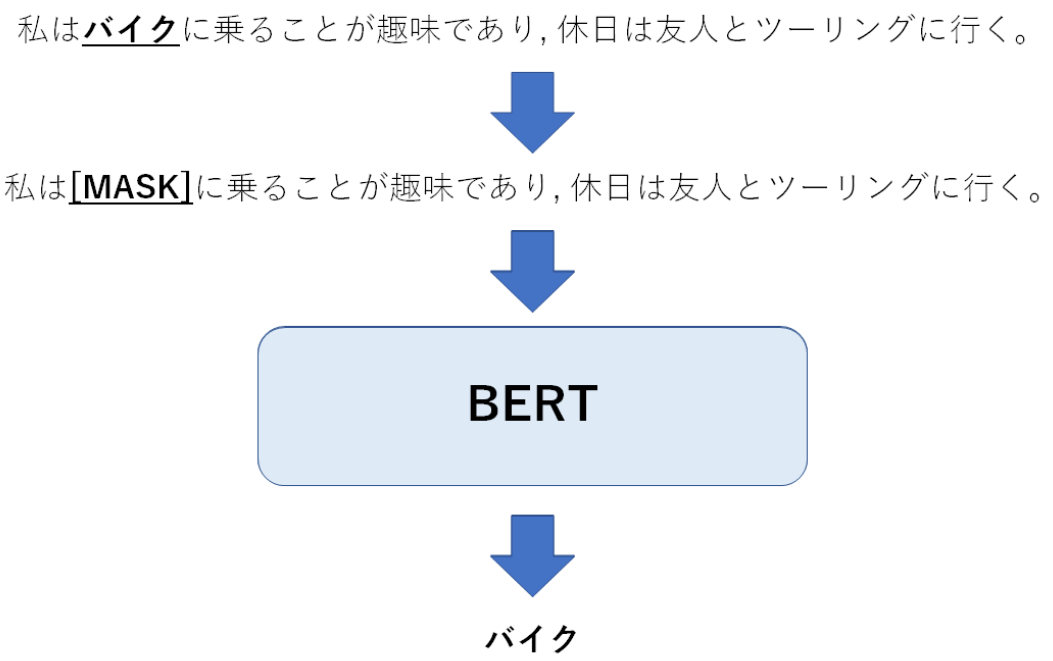

図 2.2: 事前学習 I ： Masked Language Model

でよい.

BERT には Jacob Devlin らが作成したオリジナルモデル以外にも様々な種類がある. 日本語版では京都大学から公開されているモデル[*2]や国立国語研究所が開発したモデル[*3]などがある. 本研究では東北大学から公開されている訓練済み日本語 BERT モデル[*4] を利用した.

今回利用した BERT の特徴は以下の通りである.

- モデルアーキテクチャ

  レイヤー数　12

  隠れ層のサイズ　768

- 学習データ
  - 2020 年 8 月 31 日時点の日本語版 Wikipedia

[*2] https://nlp.ist.i.kyoto-u.ac.jp/?ku_bert_japanese
[*3] https://www.gsk.or.jp/catalog/gsk2020-e
[*4] https://github.com/cl-tohoku/bert-japanese

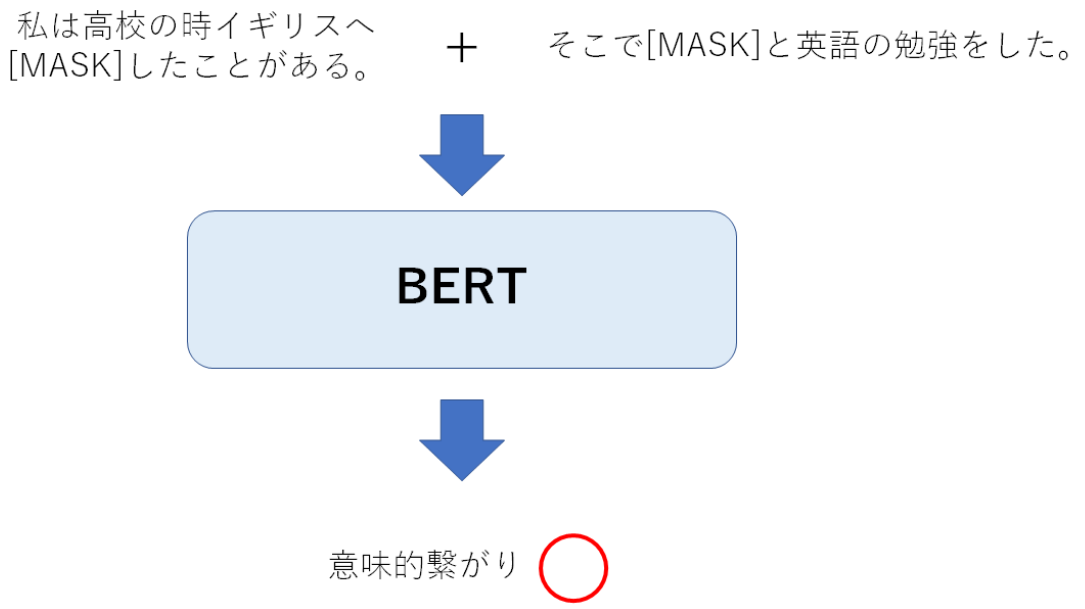

図 2.3: 事前学習 II ： Next Sentence Prediction1

サイズ　約 4.0GB

文　約 3000 万文

BERT は Transformer [7] の encoder 部分を 12 層つなぎ合わせたモデルであり, これが上述したレイヤー数である．また, 隠れ層とは入力と出力を対応付ける層のことである．その次元数がサイズであり, 今回使用したモデルでは 768 であった．

2.3　系列ラベリング問題

系列ラベリング問題とは, 文のような順序のあるデータに対して逐次的にラベルを与える手法である．ラベルは解く問題によって異なる．系列ラベリング問題を解く手法は大きく 2 つに分けられる．それは pointwise 方式か否かである．pointwise 方式とは, 個々の事例に焦点を当て, それぞれで分類する方式である．例えば, 個々の単語に焦点を当て, それぞれの読みを分類器で分類する, というのは pointwise 方式である．それに対して, pointwise 方式でない手法には隠れマルコフモデル (Hidden Markov Model:HMM) や条件付確率場 (Conditional Random Field:CRF) などがある．本研究では pointwise 方式

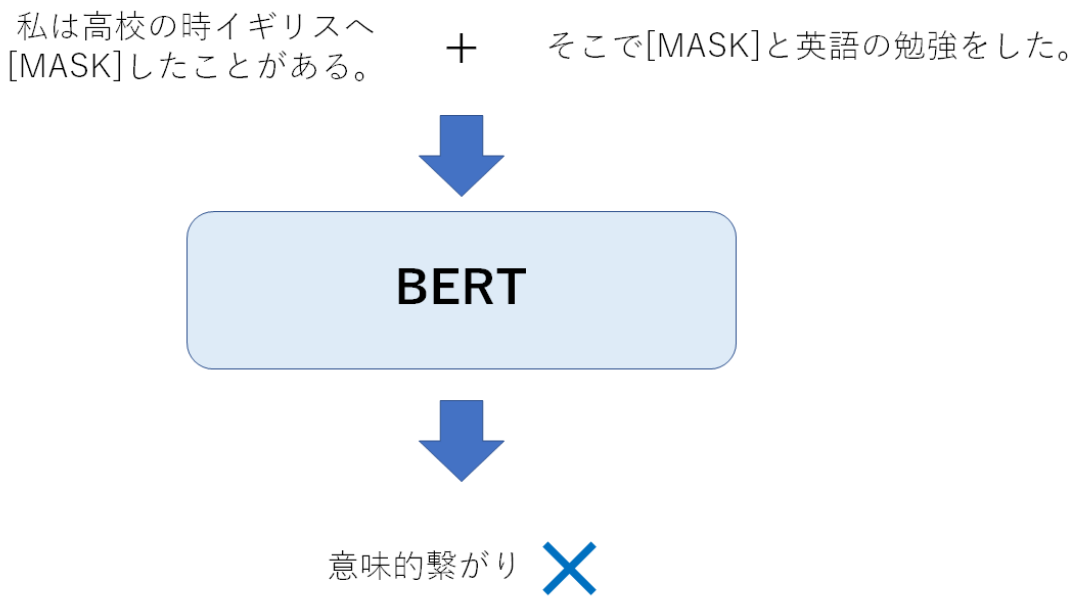

図 2.4: 事前学習Ⅱ：Next Sentence Prediction2

である BERT を採用している．2.2 節で述べた通り, BERT の事前学習には大規模なデータを使用しており, また, Masked Language Model などで文全体の学習を行っている．つまり, ある程度文脈を考慮する必要がある読み推定タスクにおいても申し分ない精度が期待できる．このような理由から本実験では BERT を用いた．

次に, 本研究で行った読み推定を例に挙げ, pointwise 方式で解く系列ラベリング問題について説明する．例文「あの出店で売っているカレーは辛い」について考える．例文中で読みに曖昧性のある単語は「出店」(「デミセ」, 「シュッテン」) と「辛い」(「カライ」, 「ツライ」) である．また, この文を形態素解析すると「あの/出店/で/売っ/て/いる/カレー/は/辛い」となる．ここで, 単語の読みは文脈 (周囲の単語) に依存すると考えられる．例えば「出店する」という文の「出店」は「シュッテン」と読むが, 例文の「出店」は「デミセ」と読む．これは「出店」の後の単語の品詞が動詞か助詞かで判断できる．「辛い」についても似たことが言える．例えば「辛い仕事」という文の「辛い」は「ツライ」と読むが, 例文の「辛い」は「カライ」と読む．これは辛いの 2 単語前が食べ物かどうかで判断できる[*5]．日本語を母語とするものであれば, 容易に読みを判断できるが, 機

[*5] ここで挙げた「出店」と「辛い」の判断方法は説明のために単純化ものであり, 実際はこれほど単純では

械は大量の文から確率を用いて判断する．図 2.5 は大量の文を学習させた読み推定システムの例である．

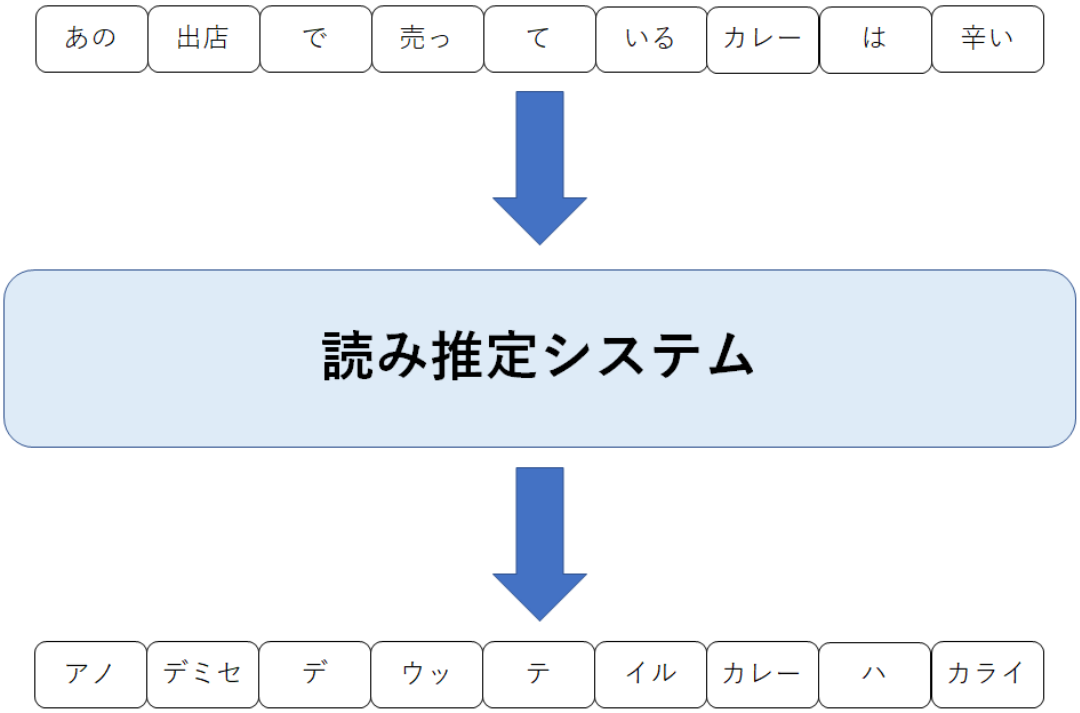

図 2.5: 大量の文を学習させた読み推定システムの例

読み推定システムでは, 文を入力することで文中の全ての単語に読みを付与する．この時, 読みに曖昧性のない単語については, コーパス中の読みをそのまま与える．読みに曖昧性のある単語の読みについては, それぞれで分類していく．図 2.6 では, 分類の様子を表している．

図 2.6 を見ると分かるように,「出店」では「シュッテン」か「デミセ」,「辛い」では「カライ」か「ツライ」の 2 値で分類を行っている．ここで, 図 2.7 のような確率でシステムが読みを判断したとする．

図 2.7 を見ると,「出店」については「デミセ」と読む確率が 0.88,「シュッテン」と読む確率が 0.12 である．よって, システムの推定結果は「デミセ」であり, それを出力する．「辛い」についても同様で,「カライ」と読む確率が 0.91,「ツライ」と読む確率が 0.09 であるため, システムの推定結果を「カライ」とし, それを出力する．つまり, 入出力は文だが, その内部では単語毎に読みを推定している．

ない．

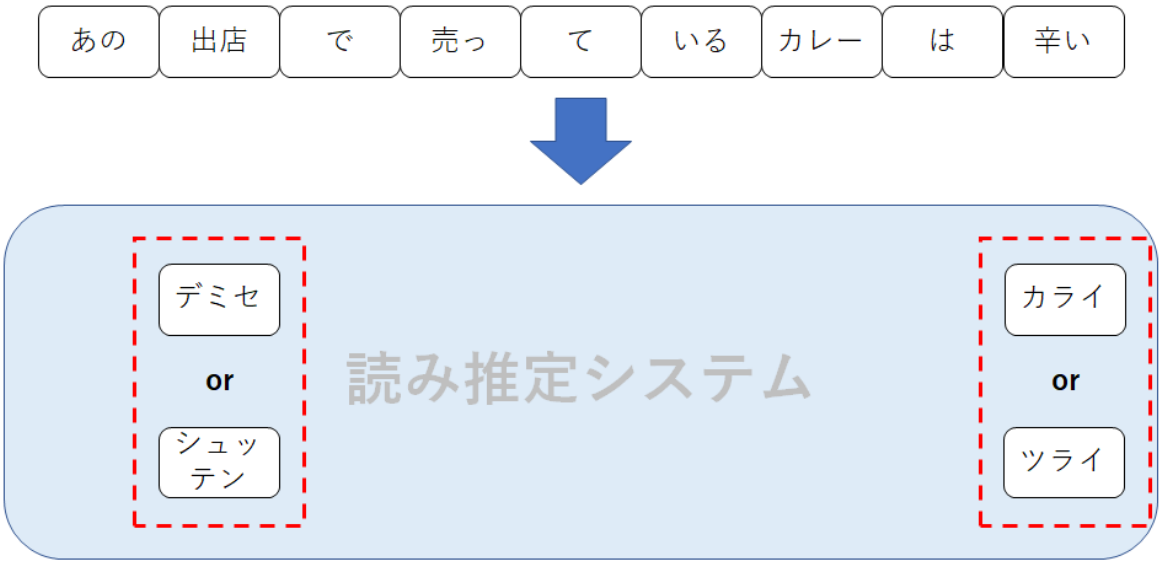

図 2.6: 読みの分類の様子

このように, 個々の事例について分類を行う手法を, pointwise 方式という.

図 2.7: 読みの判断から出力までの様子

第 3 章

関連研究

3.1 同形異音語の読み推定の関連研究

対象単語を絞って同形異音語の読み推定を行った研究には, 1 節でも述べた筆者ら [1] がある. この論文では BCCWJ 中に存在する 71 単語の読みを, さまざまな素性を用いて分類している. 全単語を対象とした読み推定を行った研究には, 筆者ら [5] がある. 本研究では 1-shot の設定としたところに違いがある.

3.2 全単語対象の関連研究

本論文ではコーパスに含まれる曖昧な読みを持つ全単語を対象に読みの推定を行った. 全単語を対象にしている点, 読みは意味が違うと異なることが多い点から見ると, 本タスクは all-words Word Sense Disambiguation (WSD, 語義曖昧性解消) と関連がある. all-words WSD とは, 文書中の全単語を対象に語義ラベルを与えるタスクである. WSD では一般に, 単語の意味は文脈に依存すると仮定している. 本研究でもまた, 単語の読みが文脈に依存すると仮定している.

新納ら [8] はテキスト解析ツール KyTea[*1] を用いて all-words WSD が解決できることを示した. KyTea は分割されたテキストデータを用いて単語分割を訓練するモデルである. 新納らは訓練データに語義データを加えて学習させることで, 語義曖昧性解消のモデルを構築した. このように, 曖昧性のある全単語にラベルを付与して学習させることで, 全単語を対象とした曖昧性解消システムを構築できる. また, 鈴木ら [9] は概念辞書を用

[*1] http://www.phontron.com/kytea/index-ja.html

いて多義語の周辺単語の分散表現を作成し, それらのユークリッド距離を計算することで多義語の語義を推測した．また, Jiaju Du et al. [10] は英語の all-words WSD タスクに BERT が有効であることを示した．具体的には, BERT を用いることで当時の最高性能を 5.2 ポイント上回る結果を残している．

3.3　疑似データを用いることの関連研究

疑似データを用いた研究には, 清野ら [11] や斎藤ら [12] の研究がある．清野らは, 文法誤り訂正において, 疑似データの生成方法や疑似データの生成元について検討している．その結果, CoNLL-2014 において当時の最高性能を記録した．斎藤ら [12] は, スペル訂正タスクにおいて, 自動生成された疑似正解データを用いて事前学習を行った後, 少量の人手正解データを用いて再度学習させる手法が有効であることを示した．また, Xiaojie Wang et al. [13] は, 疑似訓練データと語義タグ付きデータとを組み合わせることが中国語の語義曖昧性解消に効果的であることを示した．

第 4 章

疑似データを用いた同形異音語の読み推定

本章では, 読み推定に用いたコーパスやモデルの説明, 疑似データを用いることの意義を述べる.

4.1 現代日本語書き言葉均衡コーパス

現代日本語書き言葉均衡コーパス (以下 BCCWJ:Balanced Corpus of Contemporary Written Japanese) [2] は, 様々なジャンルにまたがって 1 億 430 万語のデータを格納しており, 2023 年 2 月時点で, 日本語について入手可能な唯一の均衡コーパスである. BCCWJ では，形態論情報をほとんど自動で付与しているが，その一部には人手で解析精度を高めたコアデータが含まれている[*1]．本研究の実験には, 形態論情報が自動で付与された非コアデータを疑似データとして利用した．また，BCCWJ は書き言葉であるため，例えば「日本」が「ニホン」なのか「ニッポン」なのかなど，正確な読み情報は分からない場合がある.

BCCWJ に含まれるテキストのジャンルを以下に示す．本研究では全てのジャンルを対象に実験を行った.

- 出版サブコーパス
  - 書籍

[*1] https://clrd.ninjal.ac.jp/bccwj/doc/manual/BCCWJ_Manual_02.pdf

  - 雑誌
  - 新聞
- 図書館サブコーパス
  - 書籍
- 特定目的サブコーパス
  - 白書
  - 教科書
  - 広報誌
  - ベストセラー
  - Yahoo!知恵袋
  - Yahoo!ブログ
  - 韻文
  - 法律
  - 国会会議録

4.2　日本語話し言葉コーパス

日本語話し言葉コーパス (以下 CSJ:Corpus of Spontaneous Japanese) [14] は, 日本語の自発音声を大量に集め, 品詞等の様々な形態論情報を付加した話し言葉研究用のデータベースである. 格納されているデータは語数にして約 750 万語, 時間にして約 660 時間である. 音声データの書き起こしによるコーパスであるため，正確な読み情報が付与されていると考えられる.

CSJ に含まれるテキストのジャンルを以下に示す. 本研究ではすべてのジャンルを対象に実験を行った.

- 独話
  - 学会講演
  - 模擬講演
  - その他
- 対話
  - 学会講演インタビュー

  - 模擬講演インタビュー
  - 課題指向対話
  - 自由対話
- 朗読
  - 再朗読
  - 朗読

4.3　疑似データを用いることの意義

本研究では, コーパス中の全単語を対象とした読み推定システムを作成する．本研究には，4.2 節で述べた CSJ を利用する．CSJ は音声データを書き起こしているため，正確な読み情報を得ることが可能である．しかし, CSJ のような音声情報を書き起こすコーパスは構築コストが高く, 大量に用意することは困難である.

そこで本研究では，システムによって自動的に読み情報を付与された疑似データを学習データとして追加的に利用する．大量の疑似データを用いてモデルを構築したのち, 少量の正解の読み情報が付与されたデータを用いて追加学習を行う手法と, 正解データのみを用いて構築したモデルとの正解率を比較することで, 疑似データの有効性を調査した. 本実験では追加学習に CSJ に存在する読みが曖昧な単語を一用例ずつ用い, 1-shot の設定で実験を行った．ただし, ここで用いた疑似データは書き言葉 (BCCWJ のデータ) であり, 追加学習に利用したデータは話し言葉 (CSJ のデータ) である．そのため, ドメインが異なることに注意されたい.

4.4　モデル

本研究の実験には BERT の fine-tuning を利用した．その様子を図 4.1 の模式図に示す．入力はコーパスから整形して抽出したトークン列 (5.1 節参照) であり, BERT の 12 の層を経て 768 次元のベクトルへと変換される．このベクトルを識別層 W に入力することで読みラベル R を出力する．ただし, この出力は 15,291(=読みの辞書のサイズ) 次元のベクトルである．このとき, ベクトル中の a 番目の要素は, その単語の読みがラベル a である確率を表している．この確率を参照し, 最も値の大きなラベルをモデルの推定結果とした．ただし, 参照する要素は単語によって異なる．例えば, ある単語の読みがラベル

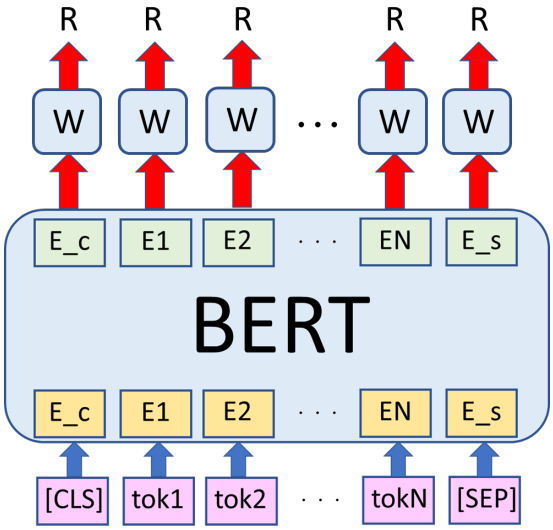

図 4.1: モデルの模式図

10, *11*, *12* に対応するのであれば, 参照する要素は *10*, *11*, *12* 番目のみである.

使用したモデルの説明を以下に示す.

modelC　CSJ で学習したモデル

modelB　BCCWJ で学習したモデル

modelB-C1s　BCCWJ で学習した後 CSJ における 1-shot 学習を行ったモデル

modelC1s　CSJ における 1-shot 学習を行ったモデル

全てのモデルは CSJ を分割したテストデータ (5.2 節参照) を用いて評価した.

第 5 章

実験

5.1 データの整形

本節では, コーパス中の単語データの整形方法を示す.

まず, 読みに曖昧性のある単語をコーパスから抽出する. これにより読みラベルを定義でき, 読みの辞書を作成することもできる. 実際に作成した辞書の一部を表したものを図 5.1 に示す.

また, 実験で用いた全データの読みに関する情報は表 5.1 の通りである.

表 5.1: BCCWJ と CSJ に含まれる読みが曖昧な単語の読み情報

読みの種類数	読みの総数	読みの候補の平均数
15,291	20,574	2.30

また, 読み以外にも単語毎に様々な情報を付与し, 単語情報 w を作成した. 作成した単語情報 w のデータ構造はソースコード 5.1 の通りである.

ソースコード 5.1: 単語情報 w のデータ構造

```
1 class WordData:
2     def __init__(self, word, tids, senid, ans, senids, cand):
3         self.word = word
4         self.tknids = tids
5         self.senid = senid
6         self.ans = ans
7         self.senids = senids
8         self.cand = cand
```

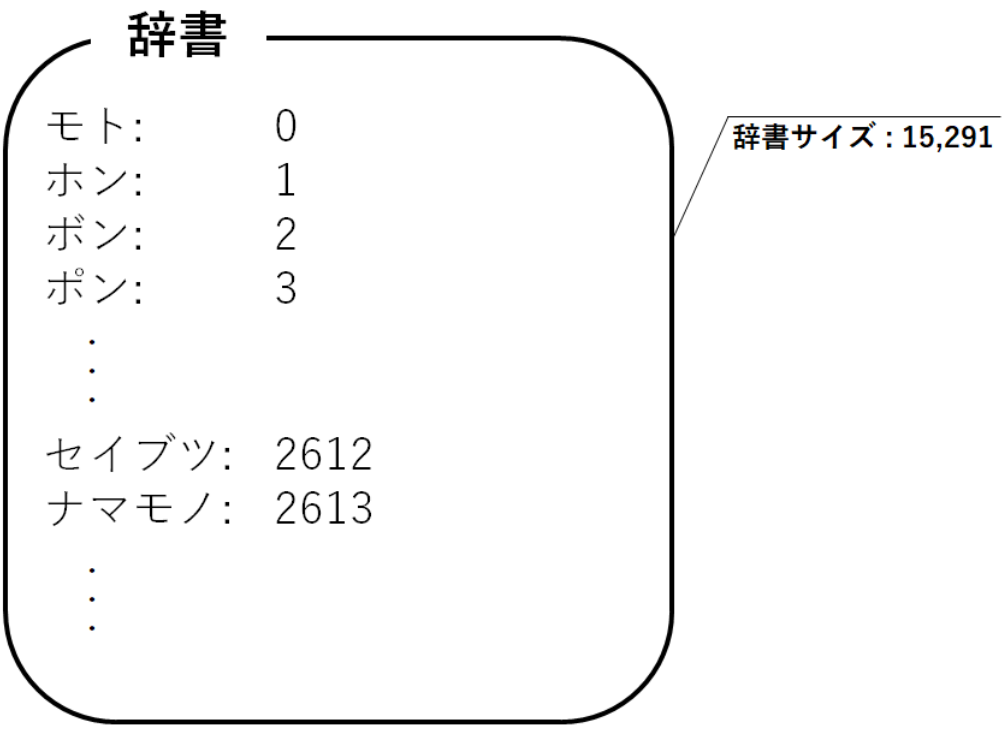

図 5.1: 実験で作成した読みの辞書

```
9         self.wsd = len(cand)
```

上記の変数が表すものは以下の通りである.

(a) word: 単語の表記

(b) tknids: BERT における単語の ID

(c) senid: 単語の読み

(d) ans: 単語の読みラベル

(e) senids: 単語の読みの候補リスト

(f) cand: 単語の読みの候補のラベルリスト

(g) wsd: 単語の読みの候補数

次にコーパス中のデータを 1 文毎に区切る. これは, 文毎に BERT へ入力し, 実験を行うためである. ただし, コーパスには単語毎にデータが格納されているため, 各コーパス

における文の区切り文字を以下のように定義した.

- BCCWJ:「。」「!」「?」
- CSJ:「です」「ます」「た」

このようにして定義された 1 文に様々な情報を付与することで, 文情報 s を作成した. 作成した文情報 s のデータ構造はソースコード 5.2 の通りである.

ソースコード 5.2: 文情報 s のデータ構造

```
class SenData:
    def __init__(self, ws):
        self.words = ws
        self.input = []
        self.y = []
        self.ys = []
        self.x = []
        self.wpos = []
```

また, s は以下に列挙する情報から構成されている.

(A) words: 文中の単語情報 w

(B) input: BERT への入力となる ID リスト

(C) y: 読みに曖昧性のある単語の読みラベルリスト

(D) ys: 読みに曖昧性のある単語の読み候補リスト

(E) x, wpos: 曖昧な読みのある単語の位置情報

文情報 s の例を, 例文「この本は多くの生物が載っている」を用いて図 5.2 に示す. なお, 図 5.2 の (B) には, [CLS] を表す 2 と [SEP] を表す 3 が挿入されている. 本実験では, 読みに曖昧性のある単語についてのみ学習及び推論を行っている. つまり, 例文においては 2 単語目の「本」(「モト」, 「ホン」, 「ボン」, 「ポン」) と 6 単語目の「生物」(「セイブツ」, 「ナマモノ」) を対象に学習・推論を行う. 推論の際には, 読み候補 (図 5.2 の (D)) の中から最も確率の高いものを選定する. この文を形態素解析すると「この/本/は/多く/の/生物/が/載っ/て/いる」となり, 合計 10 単語から構成されている. この 10 単語に対してそれぞれ単語情報 *w1*, *w2*, ..., *w10* が作成される. 例文中で読みが曖昧な単語は 2 単語目の「本」(「モト」, 「ホン」, 「ボン」, 「ポン」) と 6 単語目の「生物」(「セイブツ」, 「ナマモノ」) であるが, これらの単語情報 *w2*, *w6* は以下の通りである.

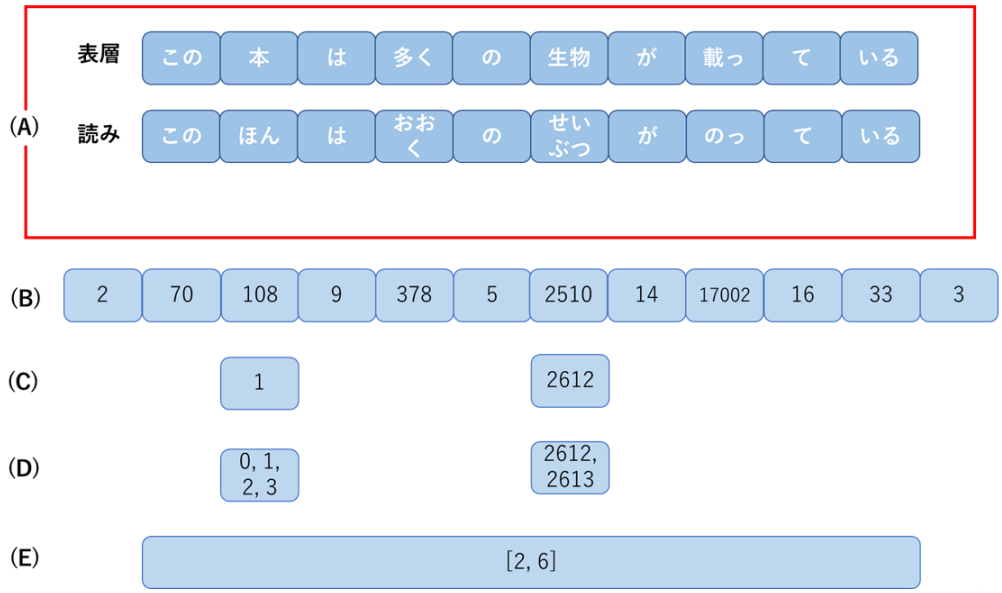

図 5.2: 「この本は多くの生物が載っている」の文情報

1. 本
   (a) 本
   (b) 108
   (c) ホン
   (d) 1
   (e) モト, ホン, ボン, ポン
   (f) 0, 1, 2, 3
   (g) 4
2. 生物
   (a) 生物
   (b) 2510
   (c) セイブツ
   (d) 2612
   (e) セイブツ, ナマモノ
   (f) 2612, 2613

（g）2

次に, 例文の文情報 s を示す.

(A) *w1*, *w2*, ..., *w10*

(B) 2, 70, 108, 9, 378, 5, 2510, 14, 17002, 16, 33, 3[*1]

(C) 1, 2612

(D) 0, 1, 2, 3, 2612, 2613

(E) 2, 6

最後に, 別の例文「私は本が好き」を用いて推論した様子を図 5.3, 出力されたベクトルの一部を図 5.4 に示す.

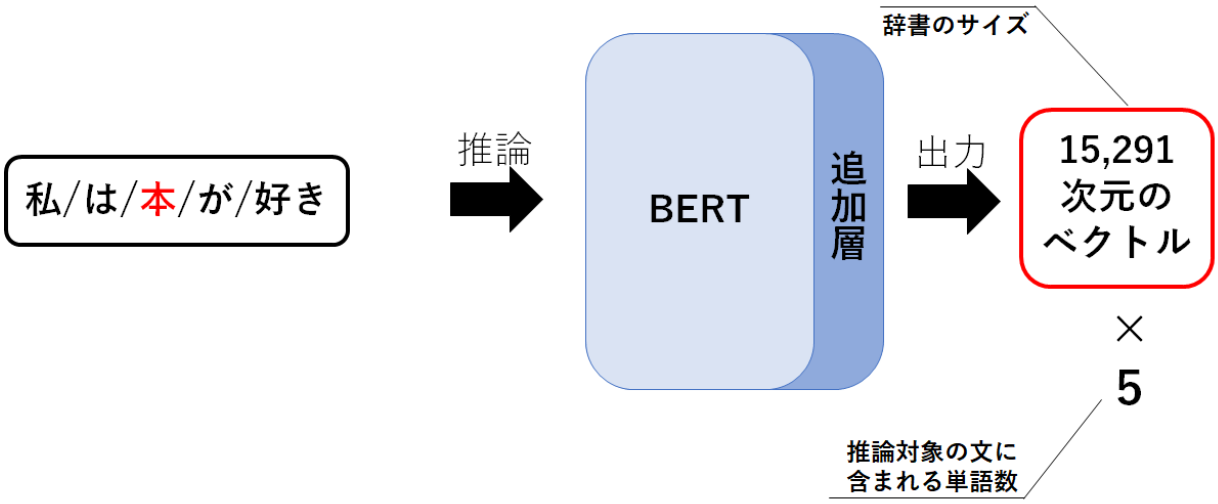

図 5.3: 推論の様子の例

図 5.3 では, fine-tuning をして追加層が付与されたモデルに例文「私は本が好き」を入力している. この例文は推論対象であるため, 読みラベルは付与されていない. 4.4 節でも述べたように, BERT の本来の出力は 768 次元である. しかし, fine-tuning をすることで作成した追加層の出力ベクトルは, 15,291 次元である. さらに, ベクトル中の a 番目の要

[*1] これが BERT の入力となるため, 元の文に [CLS] を表す 2 と [SEP] を表す 3 が加えられている

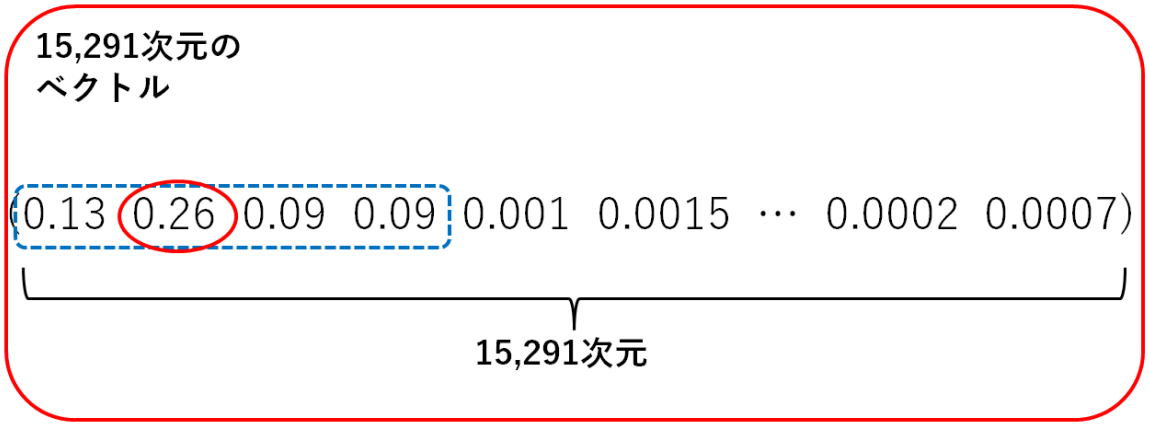

図 5.4: 出力されたベクトルの例

素は, その単語の読みがラベル a である確率を表している. 今回読み推定の対象である単語「本」に対応する辞書ラベルは, 図 5.5 を参照すると 0, 1, 2, 3 の 4 つである. そこで, 図 5.4 の 0, 1, 2, 3 番目の要素を見ると, その値はそれぞれ以下のようになっている.

0. 0.13
1. 0.26
2. 0.09
3. 0.09

よって, この中で最も値の大きい要素であるラベル 1 をモデルの推定結果とし, 読みの辞書 (図 5.1) を参照することでその読みが「ホン」であることが分かる.

5.2　実験設定

2.2 節でも述べたが, モデルには東北大から公開されている訓練済み日本語 BERT モデルを使用した. 使用したモデルのハイパーパラメータのうち, 変更したものは以下の通りである.

- 最適化関数：SGD

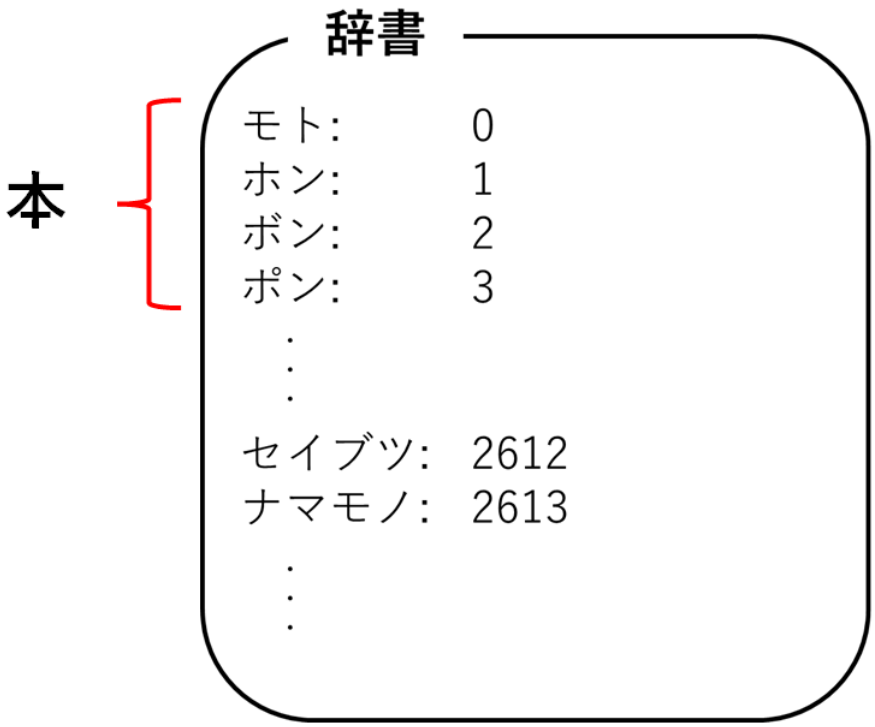

図 5.5: 辞書内での「本」のラベル

- 学習率：10^{-4}
- ミニバッチ数：1

また, CSJ はデータ全体を (訓練データ):(検証データ):(テストデータ)=1:1:8 に分割して利用した．この訓練データから, 読みに曖昧性のある単語を一用例ずつ抽出し, 1-shot 学習用のデータとした．なお, ハイパーパラメータの変更とデータの分割については, 筆者ら [5] の研究に倣った．

第 6 章

結果

追加学習に使用するデータ量毎の全単語を対象とした読み推定の正解率を表 6.1 に示す.

表 6.1: 追加学習に使用する疑似データの量ごとの全単語を対象とした読み推定の正解率

モデル名	正解率 (%)
modelC	97.97
modelB	94.22
modelB-C1s	97.40
modelC1s	58.40
MFS	64.28

まず, modelC1s, modelB, modelC の正解率がそれぞれ 58.40%, 94.22%, 97.97% であることから, 領域外の大量の疑似データを用いた学習による読み推定 (modelB) の正解率は, 対象領域の 1-shot 学習によるモデル (modelC1s) より高く, 対象領域の全データを用いた学習によるモデル (modelC) と比べると低いことが確認できる. そこで, 学習データとして, 疑似データに加えて CSJ に存在する分類対象の単語一用例ずつを追加すると (modelB-C1s), 読み推定の正解率は 97.40% となった. これは, modelC と比べて 0.57 ポイント下回りこそするが, その差はわずかである. 反対に modelB-C1s は modelB と 3 ポイントの差があることから, 書き言葉の疑似データで学習させたモデルに話し言葉データをごく少量追加学習させることで, 3 ポイントの正解率の上昇となっていることが見て取れる. つまり, 話し言葉データを一用例ずつ追加したことによる話し言葉への領域適応

の効果は高く, modelC とほとんど同程度の正解率を達成できることが確認できる. これらの実験から, 書き言葉のコーパスにある自動的に読みを付与したデータを疑似データとして利用すると, 音声書き起こしのコーパスの訓練データの量を 1-shot に減らしても, 音声書き起こしのコーパスをすべて利用する場合に比べてほとんど遜色ない読み推定の正解率が得られることが分かった.

また, 表 6.1 の MFS とは, 5.2 節で記載したテストデータの Most Frequent Score のことであり, 計算式は以下のとおりである.

$$\frac{1}{n}\sum^{n}\frac{max(wp_1, wp_2, ..., wp_l)}{\sum_k^l wp_k}$$

なお, wp_l は単語 w の l 番目の読みの出現回数を表しており, n は出現した単語の種類数を表している.

例えば, データ中に存在する同形異音語が以下の通りだったとする.

- 「辛い」

  カライの出現回数　200

  ツライの出現回数　100

- 「出店」

  カライの出現回数　150

  ツライの出現回数　50

- 「本」

  モトの出現回数　250

  ホンの出現回数　300

  ボンの出現回数　25

  ポンの出現回数　25

この時の, MFS は以下の式で求められる.

$\frac{1}{3}(\frac{max(200,100)}{200+100} + \frac{max(150,50)}{150+50} + \frac{max(250,300,25,25)}{250+300+25+25})$

$= \frac{1}{3}(\frac{200}{300} + \frac{150}{200} + \frac{300}{600})$

$= \frac{1}{3}(\frac{2}{3} + \frac{3}{4} + \frac{1}{2})$

$= \frac{23}{36}$

≈ 0.639

表 6.1 の MFS とほかの model を比較すると, modelC1s のみが MFS を下回っており,

そのほかの modelB, modelB-C1s, modelC は MFS を上回っていることが見て取れる. MFS はデータさえあれば計算できることを踏まえると, 少なくとも読み推定タスクにおいては, CSJ のデータ 1-shot のみを学習させただけのモデルにほとんど価値はないことが分かる. また, 今回使用したモデルである BERT は, 読み推定タスクにおいては, ある程度の量のデータを学習させることで真価を発揮するモデルであることが分かった.

第 7 章

考察

本章では, modelB と modelB-C1s とを比べて, どのようなデータにおいて改善が見られたのか, その傾向を考察する. 結論としては, modelB-C1s の結果から, 改善されたデータの傾向を読み取ることはできなかった.

傾向を読み取る際, 以下の 3 つを検証した.

- 品詞
- 読みの平均数
- 意味の遠さ

検証にあたって, 正解率の上昇に起因した上位 25 単語 (表 7.1) と下位 25 単語 (表 7.2) を抽出した. まず, 品詞については, 上位 25 単語のうち 23 単語が名詞, 2 単語が動詞であり, 下位 25 単語では, そのすべてが名詞であった. どちらもほとんど名詞であることから, 両モデルの品詞による傾向の差はないといえる. 次に, 読みの平均数については, 上位 25 単語では 4.12(個/単語), 下位 25 単語では 3.36(個/単語) であった. つまり, 読みの個数には, 上位と下位の間で, 1 単語あたり 0.76 個の差があることがわかる. しかし, 1 単語あたりの読みの個数の差が 1 未満であるため, 大差はないといえる. 次に, 意味の遠さについては, 筆者の主観で分類した. 例えば「開く」は「ヒラク」や「アク」と読むが, これはほとんど意味が同じである. それに対して「市場」は「シジョウ」や「イチバ」と読むが, これは文脈によって読み分けが必要であるため, 意味が遠いといえる. このように判断した時, 上位 25 単語中意味が遠い単語は「評定」(「ヒョウテイ」「ヒョウジョウ」) のみであり, 下位 25 単語中意味が遠い単語は「市場」(「イチバ」「シジョウ」) と「大勢」(「タイセイ」「オオゼイ」「タイゼイ」), 「出店」(「デミセ」「シュッテン」) の 3 単語で

ある．このような観点で上位と下位を比べても, 精度の差に傾向はみられなかった．これらのことから, modelB-C1s の正解率が modelB を上回った要因は, テストデータと同じ領域の訓練データを追加したという単純な理由によるものと考えられる．

表 7.1: 改善が見られた上位 25 単語

単語	代表的な読み		
私	ワタシ	ワタクシ	シ
後	アト	ノチ	コウ
時	ジ	トキ	ドキ
他	ホカ	タ	アダ
人	ニン	ジン	ヒト
方	カタ	ホウ	ガタ
捉え	トラエ	ツラマエ	
婆	ババ	バア	
京都	キョウト	ミヤコ	
中	ジュウ	チュウ	ウチ
節	セツ	フシ	タカシ
波形	ナミガタ	ハケイ	ナミカタ
形	ガタ	ナリ	カタチ
捉える	トラエル	ツラマエル	
九	ココノ	キュウ	ク
下	シモ	モト	シタ
行ける	イケル	ユケル	
風	フウ	カゼ	プウ
家	ヤ	イエ	カ
行け	イケ	ユケ	
評定	ヒョウテイ	ヒョウジョウ	
車	シャ	クルマ	グルマ
上	ウワ	カミ	ウエ
共	トモ	ドモ	ムタ
拍	ハク	パク	

表 7.2: 改善が見られなかった上位 25 単語

単語	代表的な読み		
日間	ジッカン	カカン	ニッカン
市場	イチバ	シジョウ	
大勢	タイセイ	オオゼイ	タイゼイ
鼻	ビ	ハナ	バナ
湖	ミズウミ	コ	ウミ
姉	アネ	ネエ	
水	ミズ	スイ	ズイ
原	ハラ	ゲン	バラ
味	ミ	アジ	
日	カ	ヒ	ジツ
開く	ヒラク	アク	
縁	ユカリ	エン	ヨスガ
六七	ロクナナ	ロクシチ	
波	ナミ	ハ	パ
梅	バイ	ウメ	
入り	ハイリ	イリ	バイリ
帯	オビ	タイ	
器	キ	ウツワ	
糞	フン	クソ	
幸	ミユキ	サイワイ	サチ
着	ギ	キ	チャク
薬	クスリ	ヤク	グスリ
酒	サケ	シュ	ザケ
肝	カン	キモ	
出店	デミセ	シュッテン	

第 8 章

結論

本稿では, BERT の fine-tuning を用いて読み推定を行った. 読み推定の対象は, BCCWJ と CSJ に存在する, 読みに曖昧性のある全単語とした. 実験では, ドメイン外の大量の疑似データ (BCCWJ の読みデータ) を用いることで, 構築コストの高い書き起こしによる読みのタグ付きデータ (CSJ の読みデータ) を減らすことができることを確認した. 具体的には, CSJ の読みデータを 1-shot まで減らしてもモデルの精度に遜色ないことが分かった.

謝辞

主指導教員である新納浩幸教授には, 論文の添削や実験の指導など, 多大なるご尽力を頂きました.

東京農工大学の古宮嘉那子准教授には, 英語論文の添削をしていただいたり, 論文を書く上で気を付けるポイントなどを教えていただいたりと, 様々な助言を頂きました.

新納研究室の先輩である田中さんには, サーバを管理する上での相談に乗っていただきました.

新納研究室の同期である伊藤君, 井筒君, 内間君, 菊田君, 岸野さん, 杉本君, 築地君, 平野君, 南濱さんとは, 研究の話から日常のコミュニケーションまで, 私の研究のモチベーション維持に欠かせない存在となってくれました.

最後に, 本研究を支えてくださった全ての方々に心よりお礼を申し上げます.

参考文献

[1] 小林汰一郎, 古宮嘉那子. SVM を用いた BCCWJ における同形異音語の読み推定. 言語処理学会第 27 回年次大会, pp. 405–409, 2021.

[2] Kikuo Maekawa, Makoto Yamazaki, Toshinobu Ogiso, Takehiko Maruyama, Hideki Ogura, Wakako Kashino, Hanae Koiso, Masaya Yamaguchi, Makiro Tanaka, and Yasuharu Den. Balanced corpus of contemporary written japanese. *Language resources and evaluation*, Vol. 48, No. 2, pp. 345–371, 2014.

[3] 新納浩幸, 浅原正幸, 古宮嘉那子, 佐々木稔. nwjc2vec: 国語研日本語ウェブコーパスから構築した単語の分散表現データ. 自然言語処理, Vol. 24, No. 5, pp. 705–720, 2017.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] 小林汰一郎, 古宮嘉那子, 新納浩幸ほか. 疑似訓練データを用いた bert による同形異音語の読み推定. 研究報告自然言語処理 (NL), Vol. 2022, No. 3, pp. 1–5, 2022.

[6] Masayuki Asahara, Kikuo Maekawa, Mizuho Imada, Sachi Kato, and Hikari Konishi. Archiving and analysing techniques of the ultra-large-scale web-based corpus project of ninjal, japan. *Alexandria*, Vol. 25, No. 1-2, pp. 129–148, 2014.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, Vol. 30, , 2017.

[8] Hiroyuki Shinnou, Kanako Komiya, Minoru Sasaki, and Shinsuke Mori. Japanese all-words wsd system using the kyoto text analysis toolkit. In *Proceedings of the 31st Pacific Asia Conference on Language, Information and Computation*, pp.

392–399, 2017.

[9] 鈴木類, 古宮嘉那子, 浅原正幸, 佐々木稔, 新納浩幸. 概念辞書の類義語と分散表現を利用した教師なし all-words wsd. 自然言語処理, Vol. 26, No. 2, pp. 361–379, 2019.

[10] Jiaju Du, Fanchao Qi, and Maosong Sun. Using bert for word sense disambiguation. *arXiv preprint arXiv:1909.08358*, 2019.

[11] 清野舜, 鈴木潤, 三田雅人, 水本智也, 乾健太郎. 大規模疑似データを用いた高性能文法誤り訂正モデルの構築. 言語処理学会第 26 回年次大会, pp. 989–992, 2020.

[12] 斉藤いつみ, 鈴木潤, 貞光九月, 西田京介, 齋藤邦子, 松尾義博. 擬似データの事前学習に基づく encoder-decoder 型 日本語崩れ表記正規化. 言語処理学会第 23 回年次大会, pp. 585–588, 2017.

[13] Xiaojie Wang and Yuji Matsumoto. Improving word sense disambiguation by pseudo-samples. In *International Conference on Natural Language Processing*, pp. 386–395. Springer, 2004.

[14] Kikuo Maekawa. Corpus of spontaneous japanese: Its design and evaluation. In *ISCA & IEEE Workshop on Spontaneous Speech Processing and Recognition*, 2003.

付録

本研究で使用した Python のソースコードを.1 に示す.

ソースコード .1: WordDataCreation.py

```
1  import pickle
2  import time
3  from tqdm import tqdm
4  from transformers import BertJapaneseTokenizer
5
6  class WordData:
7      def __init__(self, word, tids, senid, ans, senids, cand):
8          self.word = word
9          self.tknids = tids
10         self.senid = senid
11         self.ans = ans
12         self.senids = senids
13         self.cand = cand
14         self.wsd = len(cand)
15
16
17 MODEL_NAME = 'cl-tohoku/bert-base-japanese-whole-word-masking'
18 #path = '/media/sda/taichiro_kobayashi/INCL_UniDic/
       all_words_data_BCCWJ_core.pkl'
19 #path = '/media/sda/taichiro_kobayashi/INCL_UniDic/all_words_data_BCCWJ
       .pkl'
20 #path = '/media/sda/taichiro_kobayashi/INCL_UniDic/all_words_data_CSJ.
       pkl'
21 path = '/media/sda/taichiro_kobayashi/INCL_UniDic/DummyData/
       all_words_data_CSJ_dummy.pkl'
22 path2 = '/media/sda/taichiro_kobayashi/INCL_UniDic/dic_yomi_to_label.
       pkl'
23 #path3 = '/media/sda/taichiro_kobayashi/INCL_UniDic/
```

```
    word_data_BCCWJ_core.pkl'
#path3 = '/media/sda/taichiro_kobayashi/INCL_UniDic/word_data_BCCWJ1.
    pkl'
#path4 = '/media/sda/taichiro_kobayashi/INCL_UniDic/word_data_BCCWJ2.
    pkl'
#path3 = '/media/sda/taichiro_kobayashi/INCL_UniDic/word_data_CSJ.pkl'
path3 = '/media/sda/taichiro_kobayashi/INCL_UniDic/DummyData/
    word_data_CSJ_dummy.pkl'


with open(path, 'rb') as f:
    data = pickle.load(f)
with open(path2, 'rb') as f:
    dict = pickle.load(f)

tknz = BertJapaneseTokenizer.from_pretrained(MODEL_NAME)

word = []
tids = []
senid = []
ans = []
senids = []
cand = []
ws = []
c = 0
e = 0
for i, x in enumerate(tqdm(data)):
    tmp = data[i].split('\t')
    if len(tmp) == 1:
        c += 1
        continue
    word.append(tmp[0])
    tids.append(tknz(tmp[0], add_special_tokens=False).input_ids)
    senid.append(tmp[2]) # CSJ
    #senid.append(tmp[3]) # BCCWJ
    if len(tmp) == 4: # BCCWJ:5, CSJ:4
        ans.append(dict[tmp[2]]) # CSJ
        #ans.append(dict[tmp[3]]) # BCCWJ
        yomi = tmp[3].split('␣') # CSJ
        #yomi = tmp[4].split(' ') # BCCWJ
        senids.append(yomi)
```

```
61          yomils = []
62          for y in yomi:
63              yomils.append(dict[y])
64          cand.append(yomils)
65          e += 1
66      else:
67          ans.append([])
68          senids.append(tmp[2]) # CSJ
69          #senids.append(tmp[3]) # BCCWJ
70          cand.append([])
71      d = WordData(word[i-c], tids[i-c], senid[i-c], ans[i-c], senids[i
            -c], cand[i-c])
72      ws.append(d)
73
74  for i in range(10):
75      ws[i].cand, ws[i].wsd)
76
77
78  with open(path3, 'wb') as f:
79      pickle.dump(ws, f)
80
81  """
82  ws1 = ws[:int(len(ws)/2)]
83  ws2 = ws[int(len(ws)/2):]
84  with open(path3, 'wb') as f:
85      pickle.dump(ws1, f)
86  with open(path4, 'wb') as f:
87      pickle.dump(ws2, f)
88  """
```

ソースコード .2: SenDataCreation.py

```
1
2  import pickle
3  import time
4  from tqdm import tqdm
5  from transformers import BertJapaneseTokenizer
6
7  class WordData:
8      def __init__(self, word, tids, senid, ans, senids, cand):
9          self.word = word
```

```
10          self.tknids = tids
11          self.senid = senid
12          self.ans = ans
13          self.senids = senids
14          self.cand = cand
15          self.wsd = len(cand)
16
17  class SenData:
18      def __init__(self, ws):
19          self.words = ws
20          self.input = []
21          self.y = []
22          self.ys = []
23          self.x = []
24          self.wpos = []
25
26  #path = '/media/sda/taichiro_kobayashi/INCL_UniDic/word_data_BCCWJ_core
        .pkl'
27  #path = '/media/sda/taichiro_kobayashi/INCL_UniDic/word_data_CSJ.pkl'
28  path = '/media/sda/taichiro_kobayashi/INCL_UniDic/DummyData/
        word_data_CSJ_dummy.pkl'
29  #path2 = '/media/sda/taichiro_kobayashi/INCL_UniDic/SENDATA/
        sen_data_BCCWJ_core.pkl'
30  #path2 = '/media/sda/taichiro_kobayashi/INCL_UniDic/SENDATA/
        sen_data_CSJ.pkl'
31  path2 = '/media/sda/taichiro_kobayashi/INCL_UniDic/DummyData/
        sen_data_CSj_dummy.pkl'
32
33
34  with open(path, 'rb') as f:
35      data = pickle.load(f)
36
37  corpus = []
38  tmp = 0
39  for i, t in enumerate(tqdm(data)):
40      #if t.word == 。'' or t.word == '!' or t.word == ！
            '' or t.word == '?' or t.word == ？'': ##BCCWJ
41      if t.word == 'です' or t.word == 'ます' or t.word == 'た': ##CSJ
42          ws = data[tmp:i+1]
43          sd = SenData(ws)
44          sd.input.append([2])
```

```
        sd.input.extend([ws[i].tknids for i in range(len(ws))])
        sd.input.append([3])
        for j, u in enumerate(ws):
            if u.wsd >= 2:
                sd.x.append(j+1)
                sd.y.append(u.ans)
                sd.ys.append(u.cand)
                sd.wpos.append(j)
        tmp = i+1
        corpus.append(sd)


with open(path2, "wb") as f:
    pickle.dump(corpus, f)


### 全体専用 BCCWJ
"""
import pickle
import time
from tqdm import tqdm
from transformers import BertJapaneseTokenizer

class WordData:
    def __init__(self, word, tids, senid, ans, senids, cand):
        self.word = word
        self.tknids = tids
        self.senid = senid
        self.ans = ans
        self.senids = senids
        self.cand = cand
        self.wsd = len(cand)

class SenData:
    def __init__(self, ws):
        self.words = ws
        self.input = []
        self.y = []
        self.ys = []
        self.x = []
```

```
86          self.wpos = []
87
88  path_1 = '/media/sda/taichiro_kobayashi/INCL_UniDic/word_data_BCCWJ1.
        pkl'
89  path_2 = '/media/sda/taichiro_kobayashi/INCL_UniDic/word_data_BCCWJ2.
        pkl'
90
91
92  with open(path_1, 'rb') as f:
93      data1 = pickle.load(f)
94
95  with open(path_2, 'rb') as f:
96      data2 = pickle.load(f)
97  data = data1 + data2
98
99  num = 50
100 l = int(len(data)/num)
101 path = []
102 d = []
103 for i in range(num):
104     n = i + 50 path.append(f'/media/sda/taichiro_kobayashi/INCL_UniDic/
            SENDATA/sen_data_BCCWJ{n}.pkl')
105     d.append(data[i*l:(i+1)*l])
106
107
108 for k in range(num):
109     corpus = []
110     tmp = 0
111     for i, t in enumerate(tqdm(d[k])):
112         if t.word == 。'' or t.word == '!' or t.word == ！
                '' or t.word == '?' or t.word == ？'': ##BCCWJ
113         #if t.word == です'' or t.word == ます'' or t.word == た
                '': ##CSJ
114             ws = data[tmp:i+1]
115             sd = SenData(ws)
116             sd.input.append([2])
117             sd.input.extend([ws[i].tknids for i in range(len(ws))])
118             sd.input.append([3])
119             for j, u in enumerate(ws):
120                 if u.wsd >= 2:
121                     sd.x.append(j+1)
122                     sd.y.append(u.ans)
```

```
123                         sd.ys.append(u.cand)
124                         sd.wpos.append(j)
125                 tmp = i+1
126                 corpus.append(sd)
127     with open(path[k], 'wb') as f:
128         pickle.dump(corpus, f)
129 """
```