令和 2 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

BERT による単語埋め込み表現の分散値を用いた語義の広がりの分析

所属　　情報工学専攻

著者　　欧　陽恵子 (19NM705X)

指導教員　　新納浩幸　教授

令和 3 年 2 月 5 日 (金)

令和 2 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

BERT による単語埋め込み表現の分散値を用いた語義の広がりの分析

著者

欧　陽恵子 (19NM705X)

指導教員

新納浩幸　教授

論文要旨

本論文では BERT [1] が出力する単語の埋め込み表現と語義との関係を調べるために, 多義語と単義語に分けて, それら埋め込み表現の分散値, つまり語義の広がりを比較する.

BERT は言語の事前学習モデルであり, 多くの自然言語処理システムにおいてその有効性が示されている. BERT の実体は Transformer の encoder で利用される Multi-head attention を 12 層（あるいは 24 層）積み重ねたネットワークである. 各層では, 入力された単語の埋め込み表現列に対して何らかの変換を行い, 変換結果である単語の埋め込み表現列を出力する. この過程は単語の埋め込み表現が, 層を経る毎に徐々に文脈に対応した意味を表す埋め込み表現に変換されると捉えることができる.

したがって、BERT が出力する単語の埋め込み表現は, その単語が現れた文の文脈に依存している. このためある単語 w の用例を収集し, BERT により得られる単語 w に対する埋め込み表現から, それらの分散値を計算すると, その値は単語 w の語義の広がりに対応すると考えられる. そこで多義語と単義語を対象にそれら単語の用例を収集し, 分散値の比較を行った. 多義語に対しては大きな分散値, 単義に対しては小さな分散値が出ることが予想される. また BERT のどの位置の階層が最も語義の広がりに影響しているかも調査した. しかしこれら実験から, 上記の分散値では語義の広がりを測定するのは困難であることがわかった. その原因について考察する.

Master's Thesis in Scholastic 2020, Major in Computer and Information Sciences,
Granduate School of Science and Engineering, Ibaraki University

Analysis of Polysemy using Variance Values for Word Embeddings by BERT

Auther

YANGHUIZI OU (19NM705X)

Adviser

Prof. Hiroyuki Shinnou

Abstract

In this paper, to investigate the relationship between word embeddings output by BERT and semantemes, we divide words into polysemous and monosemous words. And compare the variance of the word embeddings, i.e., poly- semy.

BERT is a pre-learning model of language and has been shown to be effective in many natural language processing systems. The entity of BERT is a network of 12 layers (or 24 layers) of multi-head attention used in Transformer's encoder. In each layer Performs some conversion on the embedding string of the input word, and outputs the embedding string of the word that is the conversion result. This process can be understood as the embedding of a word is gradually transformed into an embedding that expresses the meaning corresponding to the context with each layer.

So,word embeddings output by BERT depends on the context of the sentence in which the word appears. Therefore, we collect examples of word *w* and calculate their variance values from word *w*' s embeddings obtained by BERT, and it is considered that the variance values correspond to the polysemy of word *w*. We collected examples of polysemous words and monosemous words and compared the variance values. We expected that variance values for polysemous would be large and variance values for monosemous words would be small. We also investigated which BERT position has the most influence on polysemy. However, we found that it is difficult to measure polysemy using the previous variance values, and we considered the cause of this difficulty.

目次

第 1 章　序論　7

第 2 章　関連研究　9

2.1　BERT を利用した語義曖昧性解消 9

2.2　BERT の出力する埋め込み表現をクラスタリングする研究 10

第 3 章　事前学習モデル BERT　11

3.1　自然言語処理 11

3.2　BERT 13

第 4 章　提案手法　17

4.1　手 法 17

4.2　単語埋め込み表現の分散値と語義の広がり 18

第 5 章　実験　19

5.1　単義語と多義語との語義の広がりの差 19

5.2　BERT の各階層における語義の広がりの変化 21

第 6 章　考察　23

6.1　多義語と単義語の埋め込み表現の位置関係 23

6.2　埋め込み表現間の平均距離 24

第 7 章　結論　27

参考文献　29

付録 31

A. 31

図目次

2.1 単語領域表現の可視化の例 10

3.1 BERT の Transformer ネットワーク*1 14
3.2 BERT のネットワーク構造*2 15
3.3 Masked Language Model 16

4.1 分散値 18

5.1 多義語の用例数と実験結果 20
5.2 単義語の用例数と実験結果 20
5.3 BERT の各階層における分散値 (多義語) 21
5.4 BERT の各階層における分散値 (単義語) 22

6.1 予測していた多義語と単義語の埋め込み表現の位置関係 23
6.2 現実の多義語と単義語の埋め込み表現の位置関係 24
6.3 多義語の埋め込み表現関の平均距離 25
6.4 単義語の埋め込み表現関の平均距離 25

第 1 章

序論

本論文では BERT が出力する単語の埋め込み表現と語義との関係を調べるために，多義語と単義語に分けて，それら埋め込み表現の分散値，つまり語義の広がりを比較する．BERT が出力する単語の埋め込み表現が語義を表現しているなら，多義語に対する分散値は大きく，単義語に対する分散値は小さくなることが予想できる．

BERT は言語の事前学習モデルであり，基本的には入力された単語列を対応する単語の埋め込み表現列に変換する．このとき得られる単語の埋め込み表現は word2vec [2] などから得られる分散表現のように固定したベクトルではなく，その単語が現れた文脈，つまり入力された単語列に依存している．この点からある単語 w を含む文 s を BERT に入力し w に対応する埋め込み表現 e_w を得たとき，e_w は s 内における w の語義を表していると考えられる．本論文では e_w を収集し，その分散値 V_w を得ることで w の語義の広がり（多様性）を調べることで，BERT が出力する単語の埋め込み表現と語義との関係を考察する．特に注目するのは単義語 w に対する e_w の分散値 V_w である．e_w が語義を表しているのであれば，e_w の分散値 V_w は非常に小さいはずである．これを多義語 w に対する e_w の分散値 V_w との比較から確認する．

また BERT は概略 Multi-head Attention の層を 12 層重ねたモデルであり，各層毎に単語 w に対する埋め込み表現 e_w が得られる．今第 i 層の単語 w に対する埋め込み表現を e_w^i とおく．つまり $e_w = e_w^{12}$ である．e_w がある程度の広がり（分散値）を持っていたとしても，e_w^1 は分散表現に近い形なので，その広がり（分散値）は小さく，層を経るに従い徐々に広がりが増していくと考えられる．本研究では各階層ごとに分散値 V_w^i を求め，どの階層で分散値が最も大きく増加するかを確認する．これによってどの階層が語義を特定するのに寄与しているかが考察できる．

実験では多義語として「頭」「意味」「核」「記録」「言葉」「胸」の 6 単語，単義語「生産」，「政治」，「意識」，「抗議」，「成績」の ５単語を対象にして分散値 V^i_w を求めることで，BERT の出力する単語埋め込み表現の語義の広がりの分析した．しかし実験から，上記した分散値では語義の広がりを測定するのは困難であることがわかった．その原因についても考察する．

第 2 章

関連研究

2.1　BERT を利用した語義曖昧性解消

BERT の出力する埋め込み表現と語義との関連性に関する研究として, BERT を利用した語義曖昧性解消 (Word Sense Disambiguation; WSD [3]) の研究がある.

論文 (曹鋭ほか [4]) では BERT の出力する埋め込み表現を特徴ベクトルとして利用して, 教師あり学習により WSD を行っている.

語義曖昧性解消とは文中の多義語の語義を識別する処理である. 例えば単語「犬」は, 通常, (a) 動物の犬, (b) スパイ, の 2 つの語義を持ち,「犬」を含む文「あいつは警察の犬だ」が与えられたときに, 文中の「犬」の語義が (a) か (b) かを識別する処理が WSD である.

先行の研究では各用例を Juman++ で単語分割し, その単語列を上記 BERT にかけ, 出力された単語埋め込み表現列から, WSD の対象単語の埋め込み表現を取り出し, これを WSD の特徴ベクトルとした. 学習には 2 層のニューラルネットワークを用いた, 実験データを処理する.

この実験によれば, WSD で従来の用いられ手法と本手法に比べて, 本手法の正解率が高いことが確認できる. 京都大学が公開している日本語版 BERT 事前学習モデルと SemEval-2 [5] の日本語辞書タスクデータを用いた実験では, 非常に高い正解率を示した. 本実験において分類器の学習を簡易にすませていることを考慮すると, BERT から WSD の特徴ベクトルを得る有用性が示されたと言える.

2.2　BERT の出力する埋め込み表現をクラスタリングする研究

BERT の出力する埋め込み表現をクラスタリングする研究は，語義の広がりに関する研究と関連している．

論文（山内崇史ほか [6]）では，単語の多義性と意味の広がりを捉えるために，文脈化された単語ベクトル集合から領域表現を獲得する手法を提案した．ELMo [7] や BERT によって得られる文脈化された単語ベクトル集合をクラスタリングすることにより領域表現を獲得する手法を提案する．これにより，各単語の出現文全体の文脈を考慮した単語表現が得られることを期待する．また，動的にクラスタ数を決定できるクラスタリング手法を用いることで，各単語に対して適切な語義数が割り当てられることを期待する．図 2.1 に提案手法によって得られた単語領域表現を可視化した例を示す．plant（工場，植物）がそれぞれ factory および flower と重なる領域を持ち，意味の広がりを表現できていることが分かる．

つまり、BERT の出力する埋め込み表現をクラスタリングすることにより領域表現を獲得している．この領域表現が本研究における語義のクラスタに対応している．

評価実験の結果，文脈を考慮しないタスクでは既存手法と同等以上の性能を示し，文脈を考慮するタスクでは既存手法よりも高い性能を達成した．この領域表現を単語間の意味的類似度推定タスクおよび単語間の関係推定タスクに利用して，それらの精度向上を果たしている．

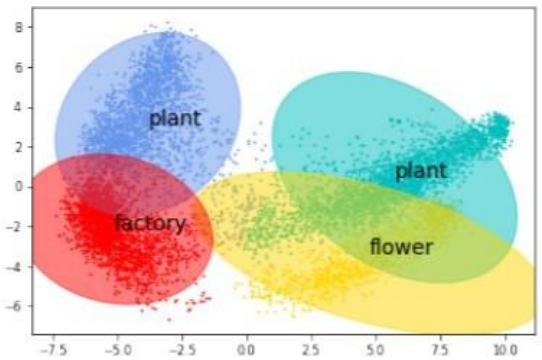

図 2.1: 単語領域表現の可視化の例

第 3 章

事前学習モデル BERT

3.1　自然言語処理

3.1.1　原理

事前学習モデル BERT を紹介する前に，自然言語処理について紹介する．

自然言語処理とは，コンピュータに人が使っている言語を処理させる技術を意味する。人間の言語（自然言語）を機械で処理し，内容を抽出することです．具体的には，言葉や文章といったコミュニケーションで使う「話し言葉」から，論文のような「書き言葉」ま での自然言語を対象として，それらの言葉が持つ意味をさまざまな方法で解析する処理技術を指す．

自然言語処理は，次のような流れで実行される。まず，形態素解析 [8] を用いて，文章を単語などの最小単位（形態素）に切り分ける．形態素解析とは，テキストデータを「意味を持つ最小限の単位（単語）」に分解し，文章やフレーズの内容を判断するために 用いる技術である．次に，データクレンジングを用いて，不要な文字列を取り除く．Bag of words [9] などを用いて，形態素解析を行ったデータをベクトルの形式に変換する．Bag of words とは，文書中に出現する単語を数え，その数を特徴とする方法である．文書における出現順序は考慮されないので、シンプルな表現である．最後に，tf-idf (term frequency – inverse document frequency) などを用いて，各単語の重要度を評価する．tf-idf は文書における単語の出現率を考慮する方法で，主に情報検索やトピック分析などの分野で用いられる．

3.1.2　自然言語処理の活用例

今, 自然言語処理は多くのところで利用される. 例えば, 開発者が製品のレビューのようなテキスト文書を分析する. 自然言語処理を使えばキーワードや地名を抽出した上でそれらの単語間の関係を分析したり, さらにはテキスト文書の内容が否定的か肯定的かが判定できる. 開発者はレビューに従って, 商品の開発を改善することができる.

自然言語処理には, 様々なビジネス事例がある. 例えば, 多くの企業は, 顧客サポートなどの電話を録音し, 書き起こして分析している. 自然言語処理は, そのようなデータを分析してより迅速に顧客のニーズに対応することに役立つ. また, 自然言語処理は, 文章のトーンを見極めるのに使われる. SNS 上で自社製品がどのように受け止められているか知りたい企業にとって, 非常に役に立ちつ.

3.2　BERT

3.2.1　原理

BERT は, 多くの NLP モデルの中でも特に精度が高いモデルと言われる. BERT で特に新規性があったのは,「事前学習」と呼ばれる学習の方法である. BERT 独特の事前学習が, BERTの汎用性を飛躍的に高めることにつながった.

従来の NLP モデルは, CNN や RNN といったニューラルネットワークを用いていった. 一方, BERT は Transformer を用いた事前学習モデルであり, 既存の実行モデルに付加することで, その実行モデルの精度を向上させることができる. 既存モデルに付加することを, Fine-tuning(転移学習) と言う.

BERT の事前学習の大きな特徴は, 文章データを基に, Transformer が文脈を双方向 (Bidirectional) に学習することである.

Transformer のネットワークアーキテクチャを図 3.1 に示す. Transformer は, 複数のエンコーダーとデコーダーをスタックすることによって形成されるエンコーダー-デコーダー構造である. 図 3.1 の左側はエンコーダーであり、Multi-Head Attention と完全な接続で構成されており, 入力コーパスを特徴ベクトルに変換するために使用される. 右側はデコーダーで, 入力はエンコーダーの出力と予測結果である. これは, Masked Multi-Head Attention、Multi-Head Attention、および最終結果の条件付き確率を出力するための完全な接続で構成される.

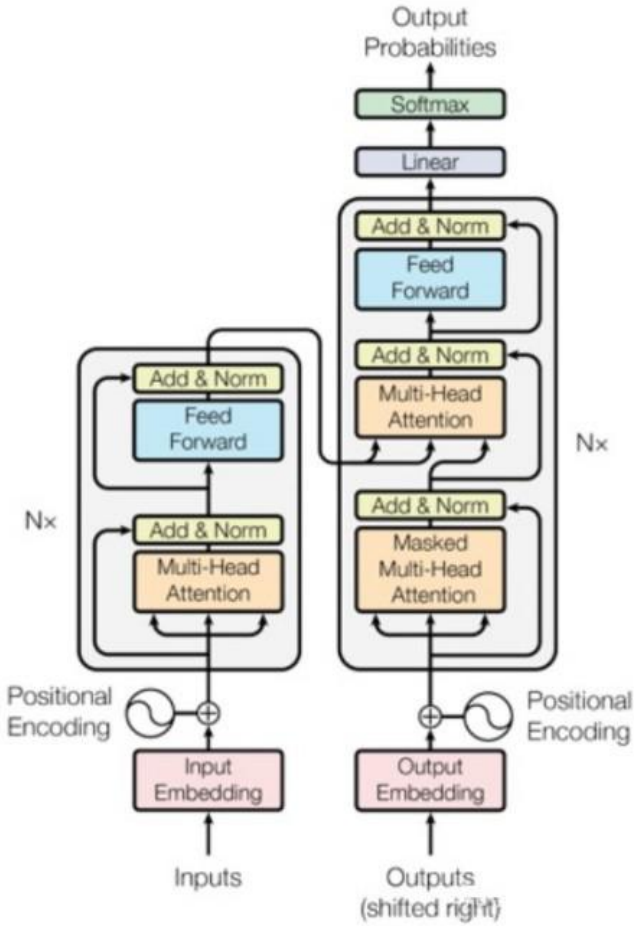

図 3.1: BERT の Transformer ネットワーク[*1]

図 3.1 の左側は、以下の図 3.2 の「Trm」に対応するTransformer Block である.

[*1] https://qiita.com/Kosuke-Szk/items/4b74b5cce84f423b7125

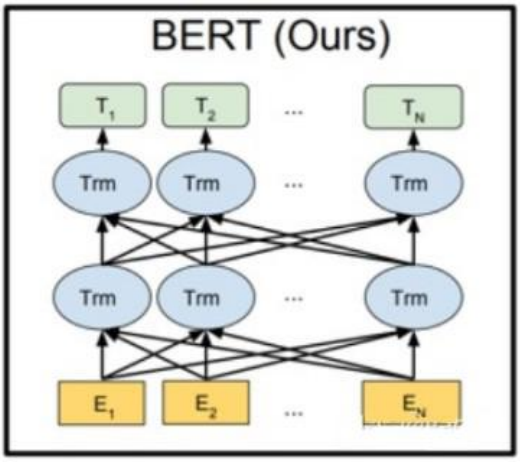

図 3.2: BERT のネットワーク構造[*2]

3.2.2　BERT の事前学習について

それでは、BERT の事前学習で用いられる，2 つの手法を見ていく．

Masked Language Model(MLM)：入力した文章から，無作為に抽出した 15 %のトークンをマスクし，以下のような割合で置換してから，置換前のトークンは何であるかを予測する穴埋め問題が MLM である．

BERT 実験では，WordPiece トークンの 15 %がランダムにマスクされる．モデルをトレーニングするとき，パラメータ学習のために文書がモデルに複数回送られますが，Google はこれらの単語を毎回マスクするわけではありませんが，マスクする単語を決定した後，80 %の確率で直接な代わる．[mask] に 10 %の確率で他の単語に置き換える．10 %の確率で、元のトークンが保持される．

例えば、以下の図 3.3 のように示す．

[*2] https://qiita.com/Kosuke-Szk/items/4b74b5cce84f423b7125

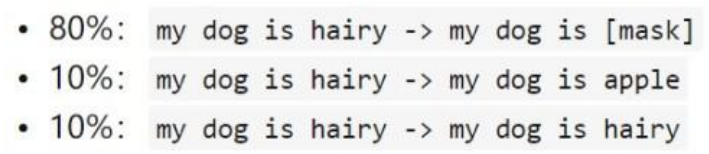

図 3.3: Masked Language Model

Next Sentence Prediction(NSP):Q & A など, 文同士の関係を考慮する問題に対しては, MLM は対応できません. そこで, 2 つの文が関係あるかを予測するのが, NSP である.

学習用データの 50 %を文章 A、B が関係あるもの, 50 %を無関係なものとする. そして, 文章B が文章 A の後に続いて, 意味が通るかどうかを予測する方法である.

第 4章

提案手法

4.1　手法

BERTは言語の事前学習モデルであり，基本的には入力された単語列を対応する単語の埋め込み表現列に変換する．このとき得られる単語の埋め込み表現は word2vec (Mikolov et al.(2013)) などから得られる分散表現のように固定したベクトルではなく，その単語が現れた文脈，つまり入力された単語列に依存している．この点からある単語 w を含む文 s を BERT に入力し，w に対応する埋め込み表現 e_w を得たとき，e_w は s 内における w の語義を表していると考えられる．本論文では e_w を収集し，その分散値 V_w を得ることで w の語義の広がり（多様性）を調べることで，BERT が出力する単語の埋め込み表現と語義との関係を考察する．特に注目するのは単義語 w に対する e_w の分散値 V_w である．e_w が語義を表しているのであれば，e_w の分散値 V_w は非常に小さいはずである．これを多義語 w に対する e_w の分散値 V_w との比較から確認する．

また BERT は概略 Multi-head Attention の層を 12 層重ねたモデルであり，各層毎に単語 w に対する埋め込み表現 e_w が得られる．今第 i 層の単語 w に対する埋め込み表現を e_w^i とおく．つまり $e_w = e_w^{12}$ である．e_w がある程度の広がり（分散値）を持っていたとしても，e_w^1 は分散表現に近い形なので，その広がり（分散値）は小さく，層を経るに従い徐々に広がりが増していくと考えられる．本研究では各階層ごとに分散値 V_w^i を求め，どの階層で分散値が最も大きく増加するかを確認する．これによってどの階層が語義を特定するのに寄与しているかが考察できる．

実験では多義語として「頭」「意味」「核」「記録」「言葉」「胸」の 6 単語，単義語「生産」，「政治」，「意識」，「抗議」，「成績」の 5 単語を対象にして分散値 V^i_w を求めること

で, BERT の出力する単語埋め込み表現の語義の広がりの分析した.

しかし実験から, 上記した分散値では語義の広がりを測定するのは困難であることが わかった. その原因についても考察する.

4.2　単語埋め込み表現の分散値と語義の広がり

単語 w を含む文 s を n 個集め, それらを $s1, s2, \cdots, sn$ とする. これらの文を BERT に入力する. s_i 中の w に対応する BERT から得られる埋め込み表現を e^i_w とする. e^i_w の平均ベクトルを e_w とし, e_w の分散値 V_w を以下図 4.1 で定義する.

$$V_w = \frac{1}{n}\sum_{i=1}^{n} ||\bar{e}_w - e_{w_i}||^2$$

図 4.1: 分散値

単語 w の語義の広がりを V_w によって測ることにする.

また BERT の第 i 層目の出力内の単語 w に対する埋め込み表現を e^i_w とおき, その分散値を $V^{\,i}_{w}$ とおく. BERT は 12 層からなるので, $V^{\,1}_{w}$ から $V^{\,12}_{w} = V_w$ が得られる. これらの値を確認することで, 語義が明確になる階層位置を考察する.

第 5 章

実験

5.1　単義語と多義語との語義の広がりの差

対象とした単語は，多義語として「頭」「意味」「核」「記録」「言葉」「胸」 の 6 単語，単義語として「生産」，「政治」，「意識」，「抗議」，「成績」の5単語である．各単語の用例は BCCWJ 及び毎日新聞の ’93 から ’98 年の記事からランダムに取り出した．

また、本研究では，日本語に対応した事前学習モデルとして，京都大学黒橋・河原研究室が以下のサイトで公開している日本語 BERT 事前学習モデルを使用する．

http://nlp.ist.i.kyoto-u.ac.jp/index.php?ku_bert_japanese

取り出した用例数を形態素解析し、形態素を subword に分割したものを基本単位とし、日本語テキストのみ (Wikipedia を利用) で pretraining しました．

その後、取り出した用例の数と得られた分散値について多義語は図 5.1，単義語は図 5.2 に示す．

単語	頭	意味	核	記録	言葉	胸
用例数	82	91	191	151	184	74
分散値	197.07	241.28	112.96	213.29	179.72	172.50

図 5.1: 多義語の用例数と実験結果

単語	生産	政治	意識	抗議	成績
用例数	110	434	82	42	33
分散値	163.81	162.73	204.19	162.00	189.38

図 5.2: 単義語の用例数と実験結果

多義語 6 単語に対する分散値の平均は 186.14, 単義語 5 単語に対する分散値の平均は 176.43 であり, 単義語の分散値の方が多義語の分散値よりも小さくなっている. ただし統計的な有意差はなく, この実験結果からは予想していた結果は得られなかった.

5.2　BERT の各階層における語義の広がりの変化

BERT の各階層における語義の広がりの変化を調べた. 多義語の結果を図 5.3 に, 単義語の結果を図 5.4 に示す.

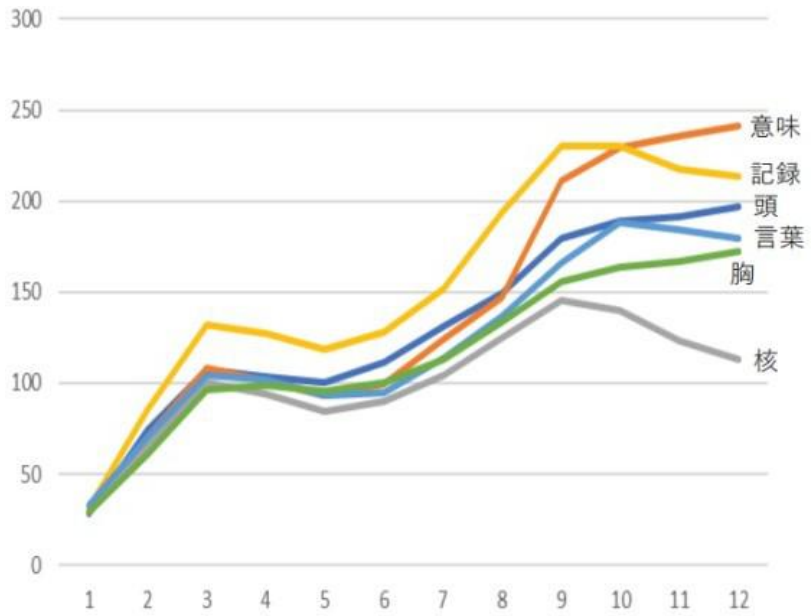

図 5.3: BERT の各階層における分散値 (多義語)

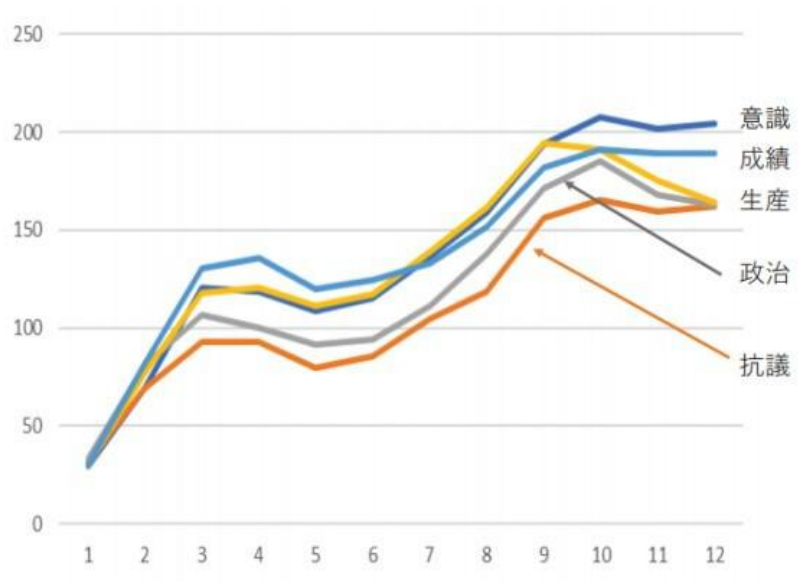

図 5.4: BERT の各階層における分散値 (単義語)

こちらの場合も多義語と単義語に大きな差は見ることができなかった. また語義の広　がりを示す分散値は階層が上がるに従って徐々に大きくなってゆくと考えられる. しかしどちらの場合も単調に分散値が上昇するという訳ではなく, 第4層目と第9層目辺りに勾配が下降したり平坦になるような様子が見られる.

BERT が出力する単語の埋め込み表現が語義を表現しているのなら, 多義語に対する分散値は大きく, 単義語に対する分散値は小さくなることが期待されたが, 実験の結果は両者の分散値に大きな差は生じなかった.

第 6 章

考察

6.1　多義語と単義語の埋め込み表現の位置関係

実験では多義語に対する分散値と単義語に対する分散値に大きな違いはなかった. こ れは語義の広がりを単に分散値から計算したことによるものだと考えられる. 当初, 語義に対する埋め込み表現は図 6.1 のような位置関係になると予想していた. 図 6.2 の場合は, 確かに多義語に対する分散値は大きく, 単義語に対する分散値は小さくなる.

図 6.1: 予測していた多義語と単義語の埋め込み表現の位置関係

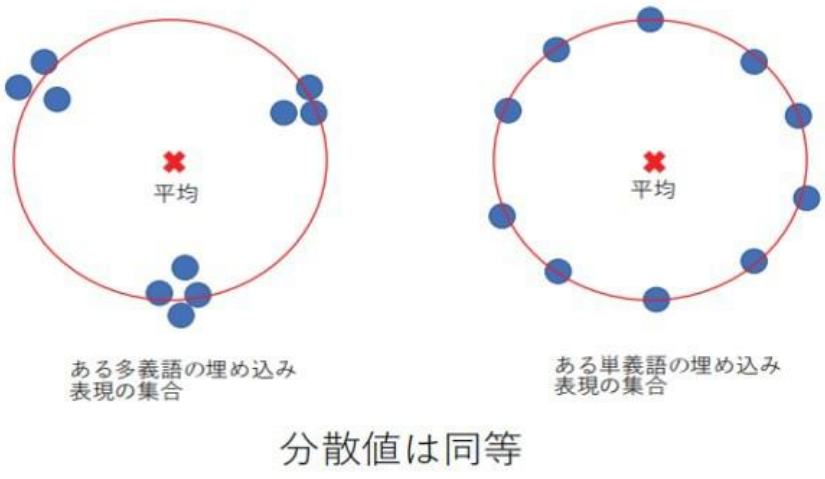

図 6.2: 現実の多義語と単義語の埋め込み表現の位置関係

しかじ実際は例えば図 6.2 のような位置関係になっていたと考えられる. 図のような場合は, 多義語に対する分散値と単義語に対する分散値に大きな違いは生じない. 言い換えれば、多義語に対する分散値と単義語に対する分散値は大体同等である.

6.2　埋め込み表現間の平均距離

語義の広がりを確認するために, BERT から出力される対象単語 w に対する埋め込み表現 e_w の集合内の全ペアに対してその距離を求め, その平均の距離を測定した. 結果を図 6.3 と図 6.4 に示すが, 埋め込み表現間の平均距離に関しても多義語と単義語間では大きな差は生じていない.

本論文では語義の広がりを埋め込み表現の位置関係で測ることができると考えていたが, 結果的にはそれができていない. 原因として以下の3つが考えられる.

1. 語義のクラスタが小さいとは限らない

   単語 w の用例を集めて, BERT から出力される w の埋め込み表現 e_w の集合を作り, そこからクラスタリングすれば語義のクラスタが作成される.

   当初, この語義のクラスタが小さいことを想定していた. 単義語 w に対しては e_w の集合が語義のクラスタ自体を表すことになるが, 前章の実験はその大きさが特

単語	頭	意味	核	記録	言葉	胸
分散値	197.07	241.28	112.96	213.29	179.72	172.50
平均距離	19.97	22.09	15.08	20.72	19.01	18.70

図 6.3: 多義語の埋め込み表現関の平均距離

単語	生産	政治	意識	抗議	成績
分散値	163.81	162.73	204.19	162.00	189.38
平均距離	18.21	18.06	20.33	18.21	19.75

図 6.4: 単義語の埋め込み表現関の平均距離

に小さくはないことを示している.

2. 語義のクラスタ間距離が大きいとは限らない

   語義のクラスタがある程度の大きさを持っていたとしても, 語義のクラスタ間距離が大きければ, 多義語 w に対する e_w の集合の分散値は大きくなるはずである.

   当初, この語義のクラスタ間距離が大きいと予想していた. しかし前章の実験で はそのような結果は示されなかった.

   本来, 語義のクラスタ間距離は語義の距離に対応しているので, それらの値も多様性があり, 一様に大きいいとは限らない.

3. コーパスから得た用例が多義語になっていない可能性もある

   本実験で用いた多義語に対して実際に語義が異なる用例を収集できているかを確認できていない.

上記の 3 点の他に，そもそも BERT が出力する埋め込み表現自体が語義を表現しているのかどうかも確認の必要がある. BERT は, 本来, 設定したタスクを利用してFine-Tuning を行って利用するものであり，タスクに応じて BERT の重みが調整され， その出力がタスクに適したものになる. このため素の BERT が出力した埋め込み表現が直接語義を表現できているのかどうかは不明である.

今後は上記の点に注意し, BERT の出力する埋め込み表現と語義との関係を調べていきたい.

第 7 章

結論

本論文では BERT が出力する単語の埋め込み表現と語義との関係を調べるために, 単義語と多義語に分けて, それら埋め込み表現のなす広がりを測ることを行った.

具体的には, 単語 w の用例を収集し, BERT により得られる単語wに対する埋め込み表現から, それらの分散値を単語 w の語義の広がりと定義した. 多義語「頭」「意味」「核」「記録」「言葉」「胸」と単義語「生産」「政治」「意識」「抗議」,「成績」を対象にそれら単語の用例を収集し, 分散値を測った.

BERT が出力する単語の埋め込み表現が語義を表現しているのなら, 多義語に対する分散値は大きく, 単義語に対する分散値は小さくなることが期待されたが, 両者の分散値に大きな差は生じなかった. 予想に反した結果に対する原因を考察した.

今後はその考察をもとに, BERT の出力する埋め込み表現と語義との関係を調べていきたい.

謝辞

本研究を進めるにあたって, 多くのご指導を頂いた指導教員の新納浩幸教授に感謝致します. また, 日常の議論を通して多くの知識, 示唆を頂いた新納研究室の皆様に感謝します.

参考文献

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. Vol. abs/1810.04805, 2018.

[2] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, Vol. 26, pp. 3111– 3119. Curran Associates, Inc., 2013.

[3] Roberto Navigli. Word sense disambiguation: A survey. *ACM Computing Surveys (CSUR)*, Vol. 41, No. 2, p. 10, 2009.

[4] 鋭曹, 裕隆田中, 静白, ブン馬, 浩幸新納, Cao Rui, Tanaka Hirotaka, Bai Jing, Ma Wen, Shinnou Hiroyuki. Word sense disambiguation using supervised learning with bert. 言語資源活用ワークショップ発表論文集 = Proceedings of Language Resources Workshop, No. 4, pp. 273–279, 2019.

[5] Manabu OKUMURA, Kiyoaki SHIRAI, Kanako KOMIYA, and Hikaru YOKONO. On semeval-2010 japanese wsd task. Vol. 18, pp. 293–307. The Asso- ciation for Natural Language Processing, jun 2011.

[6] 崇史山内, 智之梶原, 由紀荒瀬. 文脈を考慮した単語ベクトル集合からの単語領域表現. 言語処理学会年次大会発表論文集, p. 26, 2020.

[7] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. Vol. abs/1802.05365, 2018.

[8] 裕妙木, 裕治松本, 真長尾. User customizable japanese dictionary system and

morphological analyzer. 全国大会講演論文集, 第 42 回, 人工知能及び認知科学, pp. 17–18, feb 1991.

[9] Shuming Ma, Xu Sun, Yizhong Wang, and Junyang Lin. Bag-of-words as target for neural machine translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 332–338, Melbourne, Australia, July 2018. Association for Computational Linguistics.

付録

transformers のソースコードに対応した feature based 利用可能なソースコード A.1 に示す.

ソースコード A.1: extract-features.py

```
1  from __future__import absolute_import
2  from _ future__import division
3  from _ future__import print_function
4
5  import codecs
6  import collections
7  import json
8  import re
9
10 import modeling
11 import tokenization
12 import tensorflow as tf
13
14 flags = tf.flags
15
16 FLAGS = flags.FLAGS
17
18 flags.DEFINE_string("input_file", None, "")
19
20 flags.DEFINE_string("output_file", None, "")
21
22 flags.DEFINE_string("layers", "-1,-2,-3,-4", "")
23
24 flags.DEFINE_string(
```

```
25      "bert_config_file", None,
26      "The␣config␣json␣file␣corresponding␣to␣the␣pre-trained␣BERT␣model.␣
            "
27      "This␣specifies␣the␣model␣architecture.")
28
29 flags.DEFINE_integer(
30      "max_seq_length", 128,
31      "The␣maximum␣total␣input␣sequence␣length␣after␣WordPiece␣
            tokenization.␣"
32      "Sequences␣longer␣than␣this␣will␣be␣truncated,␣and␣sequences␣
            shorter␣"
33      "than␣this␣will␣be␣padded.")
34
35 flags.DEFINE_string(
36      "init_checkpoint", None,
37      "Initial␣checkpoint␣(usually␣from␣a␣pre-trained␣BERT␣model).")
38
39  flags.DEFINE_string("vocab_file", None,
40                      "The␣vocabulary␣file␣that␣the␣BERT␣model␣was␣
                            trained␣on.")
41
42 flags.DEFINE_bool(
43      "do_lower_case", True,
44      "Whether␣to␣lower␣case␣the␣input␣text.␣Should␣be␣True␣for␣uncased␣"
45      "models␣and␣False␣for␣cased␣models.")
46
47 flags.DEFINE_integer("batch_size", 32, "Batch␣size␣for␣predictions.")
48
49 flags.DEFINE_bool("use_tpu", False, "Whether␣to␣use␣TPU␣or␣GPU/CPU.")
50
51  flags.DEFINE_string("master", None,
52                      "If␣using␣a␣TPU,␣the␣address␣of␣the␣master.")
53
54 flags.DEFINE_integer(
55      "num_tpu_cores", 8,
56      "Only␣used␣if␣‘use_tpu‘␣is␣True.␣Total␣number␣of␣TPU␣cores␣to␣use."
            )
57
58 flags.DEFINE_bool(
59      "use_one_hot_embeddings", False,
60      "If␣True,␣tf.one_hot␣will␣be␣used␣for␣embedding␣lookups,␣otherwise␣
```

```
            "
61      "tf.nn.embedding_lookup␣will␣be␣used.␣On␣TPUs,␣this␣should␣be␣True␣
            "
62      "since␣it␣is␣much␣faster.")
63
64
65  class InputExample(object):
66
67    def __init__(self, unique_id, text_a, text_b):
68      self.unique_id = unique_id
69      self.text_a = text_a
70      self.text_b = text_b
71
72
73  class InputFeatures(object):
74    """A single set of features of data."""
75
76    def __init__(self, unique_id, tokens, input_ids, input_mask,
          input_type_ids):
77      self.unique_id = unique_id
78      self.tokens = tokens
79      self.input_ids = input_ids
80      self.input_mask = input_mask
81      self.input_type_ids = input_type_ids
82
83
84  def input_fn_builder(features, seq_length):
85    """Creates an `input_fn` closure to be passed to TPUEstimator."""
86
87    all_unique_ids = []
88    all_input_ids = []
89    all_input_mask = []
90    all_input_type_ids = []
91
92    for feature in features:
93      all_unique_ids.append(feature.unique_id)
94      all_input_ids.append(feature.input_ids)
95      all_input_mask.append(feature.input_mask)
96      all_input_type_ids.append(feature.input_type_ids)
97
98    def input_fn(params):
```

```
99      """The actual input function."""
100     batch_size = params["batch_size"]
101
102     num_examples = len(features)
103
104
105     d = tf.data.Dataset.from_tensor_slices({
106         "unique_ids":
107             tf.constant(all_unique_ids, shape=[num_examples], dtype=tf
                    .int32),
108         "input_ids":
109             tf.constant(
110                 all_input_ids, shape=[num_examples, seq_length],
111                 dtype=tf.int32),
112         "input_mask":
113             tf.constant(
114                 all_input_mask,
115                 shape=[num_examples, seq_length],
116                 dtype=tf.int32),
117         "input_type_ids":
118             tf.constant(
119                 all_input_type_ids,
120                 shape=[num_examples, seq_length],
121                 dtype=tf.int32),
122     })
123
124     d = d.batch(batch_size=batch_size, drop_remainder=False)
125     return d
126
127   return input_fn
128
129
130 def model_fn_builder(bert_config, init_checkpoint, layer_indexes,
        use_tpu,
131                      use_one_hot_embeddings):
132   """Returns `model_fn` closure for TPUEstimator."""
133
134   def model_fn(features, labels, mode, params):  # pylint: disable=
        unused-argument
135     """The `model_fn` for TPUEstimator."""
136
```

```
137     unique_ids = features["unique_ids"]
138     input_ids = features["input_ids"]
139     input_mask = features["input_mask"]
140     input_type_ids = features["input_type_ids"]
141
142     model = modeling.BertModel(
143         config=bert_config,
144         is_training=False,
145         input_ids=input_ids,
146         input_mask=input_mask,
147         token_type_ids=input_type_ids,
148         use_one_hot_embeddings=use_one_hot_embeddings)
149
150     if mode != tf.estimator.ModeKeys.PREDICT:
151       raise ValueError("Only␣PREDICT␣modes␣are␣supported:␣%s" % (mode))
152
153     tvars = tf.trainable_variables()
154     scaffold_fn = None
155     (assignment_map,
156      initialized_variable_names) = modeling.
             get_assignment_map_from_checkpoint( tvar
157          s, init_checkpoint)
158     if use_tpu:
159
160       def tpu_scaffold():
161         tf.train.init_from_checkpoint(init_checkpoint, assignment_map)
162         return tf.train.Scaffold()
163
164       scaffold_fn = tpu_scaffold
165     else:
166       tf.train.init_from_checkpoint(init_checkpoint, assignment_map)
167
168     tf.logging.info("****␣Trainable␣Variables␣****")
169     for var in tvars:
170       init_string = ""
171       if var.name in initialized_variable_names:
172         init_string = ",␣*INIT_FROM_CKPT*"
173       tf.logging.info("␣␣name␣=␣%s,␣shape␣=␣%s%s", var.name, var.shape
              ,
174                       init_string)
175
```

```
176     all_layers = model.get_all_encoder_layers()
177
178     predictions = {
179         "unique_id": unique_ids,
180     }
181
182     for (i, layer_index) in enumerate(layer_indexes):
183       predictions["layer_output_ %d" % i] = all_layers[layer_index]
184
185     output_spec = tf.contrib.tpu.TPUEstimatorSpec(
186         mode=mode, predictions=predictions, scaffold_fn=scaffold_fn)
187     return output_spec
188
189   return model_fn
190
191
192 def convert_examples_to_features(examples, seq_length, tokenizer):
193   """Loads a data file into a list of `InputBatch`s."""
194
195   features = []
196   for (ex_index, example) in enumerate(examples):
197     tokens_a = tokenizer.tokenize(example.text_a)
198
199     tokens_b = None
200     if example.text_b:
201       tokens_b = tokenizer.tokenize(example.text_b)
202
203     if tokens_b:
204       _truncate_seq_pair(tokens_a, tokens_b, seq_length - 3)
205     else:
206       # Account for [CLS] and [SEP] with "- 2"
207       if len(tokens_a) > seq_length - 2:
208         tokens_a = tokens_a[0:(seq_length - 2)]
209
210     tokens = []
211     input_type_ids = []
212     tokens.append("[CLS]")
213     input_type_ids.append(0)
214     for token in tokens_a:
215       tokens.append(token)
216       input_type_ids.append(0)
```

```
217     tokens.append("[SEP]")
218     input_type_ids.append(0)
219
220     if tokens_b:
221       for token in tokens_b:
222         tokens.append(token)
223         input_type_ids.append(1)
224       tokens.append("[SEP]")
225       input_type_ids.append(1)
226
227     input_ids = tokenizer.convert_tokens_to_ids(tokens)
228
229     input_mask = [1] * len(input_ids)
230
231     # Zero-pad up to the sequence length.
232     while len(input_ids) < seq_length:
233       input_ids.append(0)
234       input_mask.append(0)
235       input_type_ids.append(0)
236
237     assert len(input_ids) == seq_length
238     assert len(input_mask) == seq_length
239     assert len(input_type_ids) == seq_length
240
241     if ex_index < 5:
242       tf.logging.info("*** Example ***")
243       tf.logging.info("unique_id: %s" % (example.unique_id))
244       tf.logging.info("tokens: %s" % " ".join(
245           [tokenization.printable_text(x) for x in tokens]))
246       tf.logging.info("input_ids: %s" % " ".join([str(x) for x in
              input_ids]))
247       tf.logging.info("input_mask: %s" % " ".join([str(x) for x in
              input_mask]))
248       tf.logging.info(
249           "input_type_ids: %s" % " ".join([str(x) for x in
                input_type_ids]))
250
251     features.append(
252         InputFeatures(
253             unique_id=example.unique_id,
254             tokens=tokens,
```

```
255             input_ids=input_ids,
256             input_mask=input_mask,
257             input_type_ids=input_type_ids))
258   return features
259
260
261 def _truncate_seq_pair(tokens_a, tokens_b, max_length):
262   """Truncates a sequence pair in place to the maximum length."""
263
264   while True:
265     total_length = len(tokens_a) + len(tokens_b)
266     if total_length <= max_length:
267       break
268     if len(tokens_a) > len(tokens_b):
269       tokens_a.pop()
270     else:
271       tokens_b.pop()
272
273
274 def read_examples(input_file):
275   """Read a list of 'InputExample's from an input file."""
276   examples = []
277   unique_id = 0
278   with tf.gfile.GFile(input_file, "r") as reader:
279     while True:
280       line = tokenization.convert_to_unicode(reader.readline())
281       if not line:
282         break
283       line = line.strip()
284       text_a = None
285       text_b = None
286       m = re.match(r"^(.*)␣¥|¥|¥|␣(.*)$", line)
287       if m is None:
288         text_a = line
289       else:
290         text_a = m.group(1)
291         text_b = m.group(2)
292       examples.append(
293           InputExample(unique_id=unique_id, text_a=text_a, text_b=
                text_b))
294       unique_id += 1
```

```
295     return examples
296
297
298 def main(_):
299     tf.logging.set_verbosity(tf.logging.INFO)
300
301     layer_indexes = [int(x) for x in FLAGS.layers.split(",")]
302
303     bert_config = modeling.BertConfig.from_json_file(FLAGS.
            bert_config_file)
304
305     tokenizer = tokenization.FullTokenizer(
306         vocab_file=FLAGS.vocab_file, do_lower_case=FLAGS.do_lower_case)
307
308     is_per_host = tf.contrib.tpu.InputPipelineConfig.PER_HOST_V2
309     run_config = tf.contrib.tpu.RunConfig(
310         master=FLAGS.master,
311         tpu_config=tf.contrib.tpu.TPUConfig(
312             num_shards=FLAGS.num_tpu_cores,
313             per_host_input_for_training=is_per_host))
314
315     examples = read_examples(FLAGS.input_file)
316
317     features = convert_examples_to_features(
318         examples=examples, seq_length=FLAGS.max_seq_length, tokenizer=
                tokenizer)
319
320     unique_id_to_feature = {}
321     for feature in features:
322       unique_id_to_feature[feature.unique_id] = feature
323
324     model_fn = model_fn_builder(
325         bert_config=bert_config,
326         init_checkpoint=FLAGS.init_checkpoint,
327         layer_indexes=layer_indexes,
328         use_tpu=FLAGS.use_tpu,
329         use_one_hot_embeddings=FLAGS.use_one_hot_embeddings)
330
331     estimator = tf.contrib.tpu.TPUEstimator(
332         use_tpu=FLAGS.use_tpu,
333         model_fn=model_fn,
```

```
334         config=run_config,
335         predict_batch_size=FLAGS.batch_size)
336
337     input_fn = input_fn_builder(
338         features=features, seq_length=FLAGS.max_seq_length)
339
340     with codecs.getwriter("utf-8")(tf.gfile.Open(FLAGS.output_file,
341                                                  "w")) as writer:
342       for result in estimator.predict(input_fn, yield_single_examples=
              True):
343         unique_id = int(result["unique_id"])
344         feature = unique_id_to_feature[unique_id]
345         output_json = collections.OrderedDict()
346         output_json["linex_index"] = unique_id
347         all_features = []
348         for (i, token) in enumerate(feature.tokens):
349           all_layers = []
350           for (j, layer_index) in  enumerate(layer_indexes):
351             layer_output = result["layer_output_ %d" % j]
352             layers = collections.OrderedDict()
353             layers["index"] = layer_index
354             layers["values"] = [
355                 round(float(x), 6) for x in layer_output[i:(i + 1)].
                      flat
356             ]
357             all_layers.append(layers)
358           features = collections.OrderedDict()
359           features["token"] = token
360           features["layers"] = all_layers
361           all_features.append(features)
362         output_json["features"] = all_features
363         writer.write(json.dumps(output_json) + "¥n")
364
365
366 if __name__ == "__main__":
367     flags.mark_flag_as_required("input_file")
368     flags.mark_flag_as_required("vocab_file")
369     flags.mark_flag_as_required("bert_config_file")
370     flags.mark_flag_as_required("init_checkpoint")
371     flags.mark_flag_as_required("output_file")
372     tf.app.run()
```

引数に渡されたファイルのテキストから、Mecab 版 BERT による単語の埋め込み表現を json ファイルに出力します。モデルBERT のソースコードを A.2 に示す.

ソースコード A.2: bert-each-layer.py

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

import sys
import numpy as np
import torch
from transformers import BertConfig, BertModel, BertTokenizer,
    BertJapaneseToke$

argvs = sys.argv
argc = len(argvs)

target_word = argvs[1]
sentences_file = argvs[2]
f = open(sentences_file,'r')
sentences = f.read().strip().split('¥n')
sennum = len(sentences)
f.close()

config = BertConfig.from_json_file('/home/data/bert/tohoku/config.json'
    )
config.output_hidden_states=True
model = BertModel.from_pretrained('/home/data/bert/tohoku/pytorch_model
    .bin',co$
tknz = BertJapaneseTokenizer.from_pretrained('cl-tohoku/bert-base-
    japanese')
target_word_id = tknz.convert_tokens_to_ids(target_word)

allvec = np.zeros(12*sennum*768).reshape(12,-1,768)
mvec = np.zeros(12*768).reshape(12,768)

elenum = 0
for sentence in sentences:
    ids = tknz.encode(sentence)
```

```
31      if target_word_id in ids:
32          target_index = ids.index(target_word_id)
33      else:
34          continue
35
36      ids = torch.tensor(ids).unsqueeze(0)
37      model.eval()
38      with torch.no_grad():
39          a = model(ids)
40      for i in range(12):
41          lay = i - 1
42          vec = a[2][lay][0][target_index].numpy()
43          allvec[i][elenum] = vec
44          mvec[i] += vec
45      elenum += 1
46
47 mvec = mvec / float(elenum)
48
49 for lay in range(12):
50      vsum = 0.0
51      for ele in range(elenum):
52          d = allvec[lay][ele] - mvec[lay]
53          vsum += sum(d**2)
54
55          print(lay,vsum/elenum)
```

BERT の出力である json ファイルから、特定の単語に対する埋め込み表現のベクトルを取り出すソースコードを A.3 に示す.

ソースコード A.3: features.py

```
1 import pandas as pd
2 import numpy as np
3
4 with open('all_12.json', 'r') as json_file:
5     lst = []
6     for i, line in enumerate(json_file):
7         ds = pd.read_json(line)
8         for feature in ds['features']:
```

```
9                if feature['token'] == '成績':
10                   v = feature['layers'][0]['values']
11                   lst += [v]
12                   break
13
14 np.save('seiseki_features12', np.array(lst))
```

分散値を計算するソースコードを A.4 に示す.

ソースコード A.4: var.py

```
1 import numpy as np
2 word_list=[]
3  word1=np.load('ishiki_features1.npy')
4  word2=np.load('ishiki_features2.npy')
5  word3=np.load('ishiki_features3.npy')
6  word4=np.load('ishiki_features4.npy')
7  word5=np.load('ishiki_features5.npy')
8  word6=np.load('ishiki_features6.npy')
9  word7=np.load('ishiki_features7.npy')
10 word8=np.load('ishiki_features8.npy')
11 word9=np.load('ishiki_features9.npy')
12 word10=np.load('ishiki_features10.npy')
13 word11=np.load('ishiki_features11.npy')
14 word12=np.load('ishiki_features12.npy')
15 word_list.append(word1)
16 word_list.append(word2)
17 word_list.append(word3)
18 word_list.append(word4)
19 word_list.append(word5)
20 word_list.append(word6)
21 word_list.append(word7)
22 word_list.append(word8)
23 word_list.append(word9)
24 word_list.append(word10)
25 word_list.append(word11)
26 word_list.append(word12)
27 mean_word=[]
28 var_imi=[]
```

```
29 for i in word_list:
30     mean=np.mean(i,axis=0)
31     mean_word.append(mean)
32     var=np.var(i,axis=0)
33     var=np.sum(np.power(np.linalg.norm(i-mean,axis=1),2))/20
34     var_imi.append(var)
35 mean_word=np.array(mean_word)
36 mean=np.mean(mean_word, axis=0)
37 ave=np.sum(np.power(np.linalg.norm(mean_word-mean,axis=1),2))/12
38 var_imi=np.array(var_imi)
39 A=np.mean(var_imi,axis=0)
40 answer=A/ave
41
42 with open('ishiki_var','w') as f:
43     f.write(str(answer))
```

平均値を計算するソースコードを A.5 に示す.

ソースコード A.5: average.py

```
1 import numpy as np
2 word_list=[]
3 word1=np.load('seiseki_features1.npy')
4 word2=np.load('seiseki_features2.npy')
5 word3=np.load('seiseki_features3.npy')
6 word4=np.load('seiseki_features4.npy')
7 word5=np.load('seiseki_features5.npy')
8 word6=np.load('seiseki_features6.npy')
9 word7=np.load('seiseki_features7.npy')
10 word8=np.load('seiseki_features8.npy')
11 word9=np.load('seiseki_features9.npy')
12 word10=np.load('seiseki_features10.npy')
13 word11=np.load('seiseki_features11.npy')
14 word12=np.load('seiseki_features12.npy')
15 word_list.append(word1)
16 word_list.append(word2)
17 word_list.append(word3)
18 word_list.append(word4)
19 word_list.append(word5)
```

```
20 word_list.append(word6)
21 word_list.append(word7)
22 word_list.append(word8)
23 word_list.append(word9)
24 word_list.append(word10)
25 word_list.append(word11)
26 word_list.append(word12)
27 mean_word=[]
28 #var_imi=[]
29 for i in word_list:
30     mean=np.mean(i,axis=0)
31     mean_word.append(mean)
32     #var=np.var(i,axis=0)
33     #var=np.sum(np.power(np.linalg.norm(i-mean,axis=1),2))/20
34     #var_imi.append(var)
35 mean_word=np.array(mean_word)
36 mean=np.mean(mean_word, axis=0)
37 ave=np.sum(np.power(np.linalg.norm(mean_word-mean,axis=1),2))/12
38
39 with open('seiseki_ave_result','w') as f:
40     f.write(str(ave))
```