卒 業 研 究 報 告 題 目

ベクトル変換による
対義性を反映した単語の分散表現に関する研究

Study on Distributed Expression of Words
Reflecting Antonym by Vector Conversion

指 導 教 員　　出口 利憲 教授

岐 阜 工 業 高 等 専 門 学 校　電気情報工学科

2017E39 山田 隼斗

令 和 0 4 年（ 2 0 2 2 年 ）　　2 月 1 7 日 提 出

Abstract

The purpose of this study is to confirm the effectiveness of a word vector conversion that reflects relationships of antonyms. This method increases the distance between vectors of antonyms to reflect the relationships of antonyms. A word vectors is converted in experiments. In addition, the similarity of the antonyms before and after the word vector conversion is compared. Python is used to convert the vectors. For the pre-trained word vector model created using Word2Vec, a vector conversion using the gradient descent is performed on this word vectors. As a result of the experiment, the effectiveness of vector conversion was confirmed. However, the word vectors model obtained was incomplete. This is because there were antonyms that were not processed, and the vector conversion had a minute negative impact on other word vectors. Therefore, solving this problem is a future task.

目 次

第1章　序論

現在、人々の周辺の技術は日々進化を遂げている。またそれらの技術は、コンピュータやソフトウェアに関わりのない人々にも恩恵をもたらしている。その中には、音声認識を活用したスマートアシスタントや、テキストの入力の補助を行うオートコレクトまたはオートコンプリートといった技術がある。さらには、グローバル化が進む現代においてスムーズかつ自然な言語翻訳の技術も求められており、さまざまな場所でそれをより理想的なものにする取り組みが行われている。

このように人に近い技術において欠かせないものが、自然言語処理である。自然言語とは人間の扱う言語、つまり日本語や英語、中国語のように一般的に人々が意思疎通のために用いる言語である。そして、この自然言語をコンピュータに処理させる一連の技術を自然言語処理といい、この技術によって人の扱う言語をコンピュータが扱うことができるようになる。

自然言語処理の技術の重要な部分として、単語をベクトルとして表現するというものがある。これを単語の分散表現という。単語の分散表現を行うことは、単語の予測や文章の性質の分析等、多くの事に活用が可能である。しかし、単語の分散表現を行う手法の特性として、いくつかの課題が存在している。例えば、“コンピュータ”と“コンピューター”のような表記ゆれについて判別できない場合や、“かける”のように文脈によって意味が異なる多義語についても判別することができない場合が存在する。そして、単語の分散表現で大きく問題となるのが対義語が判別できないという問題である。

本研究では、従来の手法によって得られた単語の分散表現に対して、対義語の関係を反映する手法の実装を行う。また、それによって対義語同士の関係の変化および手法の正当性について評価を行う。提案手法は、学習済みの単語の分散表現モデルに対してベクトル変換を行うことによって対義語関係の反映を行うものである。単語の分散表現において、意味や用法の似た単語のベクトルは距離が近くなるという特性がある。そのため、提案手法ではベクトル同士の距離が近い対義語同士のベクトル間距離を遠ざけることによって対義性の反映を行う。従来のベクトル変換手法において対象とする単語の分散表現モデルは日本語であったが、本研究ではより規模の大きいデータを利用するため、英語のモデルを用いる。また、ベクトル変換の手順の差異として、ベクトル変換の対象となる単語ベクトルを選定する際に cos 類似度を用いる。

実験では、対義語が存在する単語について、その単語が何番目に類似しているかを導出する。これはベクトル変換前とベクトル変換後でそれぞれ導出し、ベクトル変換前後で類似順位を比較する。これにより、対義語同士のベクトル関係がどのように変化したかを評価する。また、対義語関係にない単語同士はベクトル関係の変化ができるだけ少ないことが理想であるため、非対義語関係の単語の関係についても評価の対象とする。これらを踏まえて、実装したベクトル変換手法の有効性および正当性の検討を行う。

第2章　自然言語処理[1)]

2.1　単語の分散表現

2.1.1　単語の分散表現

単語の分散表現とは、単語をベクトルで表現したものである。一般的に身近なベクトル表現には色の表現がある。色の表現として “赤” や “青” などの言葉の表現があるが、これには幅が存在し、1つの色には定まらない場合がある。そこで、色が Red/Blue/Green の3要素から構成されているとすると、それらの数値が定まることによって1つの色が定まる。さらには、色同士の関連性についても定量的に判断することができる。これを単語に置き換えて考えたものが単語の分散表現である。単語の意味を適切なベクトル表現にすることで、単語の意味をコンピュータに理解させることを目標としており、これを実現するために様々な手法が提案されている。

2.1.2　単語の分散表現モデル

単語の分散表現を得るために提案されている手法には、カウントベースの手法と推論ベースの手法が存在する。本研究では、そのうち推論ベースの手法によって作成された単語の分散表現を用いる。

推論ベースの手法によって得られる単語の分散表現のモデルの代表例には Word2Vec がある。他にも Glove や fastText、さらには文脈を考慮した ELMo や BERT といったモデルも提案されている。単語の分散表現に含まれる単語のベクトルは、一般的に 100 次元から数百次元程度であり、意味の近い単語のベクトルはより近くに配置される。

2.1.3　分布仮説

単語の分散表現を獲得するうえで提案されている手法のほぼ全てにおいて元となっている原理が、分布仮説である。分布仮説とは、“単語の意味は、周囲の単語によって形成される” というものである。ある単語に注目した際、その周囲の単語をコンテキストという。例として、“I eat apples.” と “I eat grapes.” という文があった際、“eat” の周囲には食べ物を表す単語が存在すると考えられる。また、ここに “I have apples.” という文が与えられた場合、“have” が “eat” と同じように “食べる” という意味で使われていると考えることができる。

多くの手法は、この分布仮説に基づいて提案がされており、これを利用することによって単語の分散表現を獲得することができる。

2.1.4 シソーラス

シソーラスとは、単語の意味を人手で定義したものである。ただし一般的な辞書とは異なり、類義語辞書となっており、同義語や類義語がグループ化されている。

シソーラスは多くの単語に対して同義語や階層構造等の関係性が考慮されている。しかし、同義語として定義されている単語であっても、微妙にニュアンスが異なっている場合は、それを表現することは困難である。そのため、これを直接コンピュータの自然言語理解に利用することは難しい。

ただし、シソーラスを用いることによって類義語および対義語を取り出すことが可能である。そのため、単語の関係の評価等の場面で用いられることがある。本研究では、シソーラスの 1 つである WordNet を用いて対義語を取り出す操作を行った。

2.1.5 WordNet

WordNet はプリンストン大学で 1985 年から開発されているシソーラスであり、これまでに多くの研究で利用されている。WordNet を用いることによって、類義語の取得や “単語ネットワーク” を利用することができる。また、“単語ネットワーク” を用いることによって、単語間の類似度を算出することも可能である。以下に、WordNet を用いて “left” の類義語と対義語を取り出した例を示す。

類義語

[‘bequeath’, ‘lead’, ‘impart’, ‘left’, ‘result’, ‘provide’, ‘give’, ‘forget’, ‘leftover’, ‘odd’, ‘depart’, ‘remaining’, ‘unexpended’, ‘allow’, ‘will’, ‘entrust’, ‘leave’, ‘leftfield’, ‘exit’]

対義語

[‘disinherit’, ‘arrive’, ‘right’, ‘enter’, ‘center’]

2.2 自然言語

2.2.1 自然言語

自然言語とは、日本語や英語、中国語にように人間が普段から使っている言葉の事である。一方で、自然言語の対をなすのが、コンピュータが扱う人工言語である。人工言

語にはPythonやC言語等のプログラミング言語やマークアップ言語が挙げられる。

一般的に、プログラミング言語はコンピュータが理解できるよう、形式が決められており、一意に解釈できるものとなっている。これに対し、自然言語は厳密に形式が定まっていない場合や、同じ文章でも異なる意味を表現している場合があるというように柔軟なものであるといえる。ただし、この柔軟性はコンピュータにとって理解が困難なものであるため、自然言語処理において大きな課題となっている。

2.2.2 自然言語の特徴

自然言語の特徴には、多義性と類義性がある。

多義性とは、同じ単語が複数の意味を持つことである。例えば、“character” という単語は “人物” を指す場合もあれば、“文字” を指す場合もあるといったように、異なる意味合いで使われることがある。

類義性は異なる単語が同じ、または似た意味を持つことである。“evaluation” と “assessment” はどちらも “評価” という意味があり、場合によっては言い換えが可能である。

加えて、自然言語処理を行う上で課題となっていることの1つとして、対義語の判別がある。一般に、自然言語において対義語となる2つの単語の品詞は同じである。つまり、対義語となる単語は文法上同様に用いられる。

第3章　実験で使用した技術・手法

3.1　単語の類似度計算

3.1.1　単語間の距離

単語間の距離はL2ノルムを用いて求める。L2ノルムでは、ベクトルの各要素の2乗和の平方根によって計算される。ある2つのP次元単語ベクトルをw_i,w_jとすると、その2単語間の距離d_{ij}は以下の式で表される。

$$d_{ij} = \sqrt{\sum_{k=1}^{P}(w_{ik} - w_{jk})^2} \tag{3.1}$$

3.1.2　cos類似度

cos類似度は、2つのベクトルの類似性を表す指標であり、単語のベクトル表現の類似度に関して用いられることが多い。cos類似度は、2つのベクトルがどれだけ同じ方向を向いているかを表す。cos類似度が1に近いほど2つのベクトルの挟む角は小さくなり、同じ方向を向いていることとなる。逆に、cos類似度が-1に近いほど2つのベクトルの挟む角は大きくなり、逆の方向を向いていることとなる。

2つのベクトルを$\vec{a} = (a_1, a_2, \ldots, a_n)$, $\vec{b} = (b_1, b_2, \ldots, b_n)$と与えたとき、cos類似度は以下の式で導出される。

$$cos(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|} = \frac{x_1 y_1 + \cdots + x_n y_n}{\sqrt{x_1^2 + \cdots + x_n^2}\sqrt{y_1^2 + \cdots + y_n^2}} \tag{3.2}$$

3.2　勾配降下法

3.2.1　勾配降下法

ある関数における勾配は、その勾配の点において値が増加する方向を指す。そのため、その関数のパラメータを勾配の逆方向に繰り返し更新することで、その関数の最小値を求めることができる。これを勾配降下法という。

勾配降下法の目的は関数の最小値を求めることである。勾配降下法は、ニューラルネッ

トワークの重み更新の手法の 1 つとして用いられている。ニューラルネットにおいて、ある重みとそれに対する損失関数が存在したとき、その損失を最小にするために最小勾配法を用いることにより、損失が最小となる重みを求めることができる。

3.2.2 SGD

SGD は確率的勾配降下法とも呼ばれ、最も基本的かつ単純な勾配降下法の手法である。SGD では、現在の重みを勾配方向へある一定の距離だけ更新を行う。これは以下の数式で表される。

$$\boldsymbol{W_{i+1}} = \boldsymbol{W_i} - \eta \frac{\partial L}{\partial \boldsymbol{W_i}} \tag{3.3}$$

ここで、$\boldsymbol{W_i}$ は更新される重みパラメータであり、$\frac{\partial L}{\partial \boldsymbol{W_i}}$ は $\boldsymbol{W_i}$ に関する損失関数の勾配である。η は学習率と呼ばれ、一度の処理における更新量を設定する。一般的に、学習率の値には 0.01 や 0.001 といった値を設定しておく。この式に従って繰り返し計算を行うことで、損失関数を最小にする重みパラメータを求めることが可能となる。SGD の不完全な点としては、解を求めるまでの収束が遅く、場合によっては発散してしまう等の問題点が挙げられる。そのため、SGD 以外にも多くの勾配降下法を用いた最適化手法が提案されている。

3.2.3 Adagrad

SGD の課題に対して考案されたものが、慣性を考慮した MomentumSGD である。これは 1 つ前の勾配の情報を計算に用いることによって振動を抑制するものである。しかし、学習率が一定であることと、予め決定しておくパラメータが学習率 η と慣性係数 α の 2 つとなることで、調整が困難であるという問題点があった。

これに対して、更に改善を加えた手法の 1 つが Adagrad である。Adagrad では、学習率のパラメータのみを予め決定しておき、繰り返し処理が進むにつれて、重みごとに学習率の値が変化していく。Adagrad の更新式を以下の式に示す。

$$\boldsymbol{h_{i+1}} = \boldsymbol{h_i} + \left(\frac{\partial L}{\partial \boldsymbol{W_i}} \right)^2 \tag{3.4}$$

$$\boldsymbol{W_{i+1}} = \boldsymbol{W_i} - \eta \frac{1}{\sqrt{h_{i+1} + \varepsilon}} \frac{\partial L}{\partial \boldsymbol{W_i}} \tag{3.5}$$

(3.4) 式より、パラメータ $\boldsymbol{h}$ の値は繰り返し処理が進むごとに増加する。これにより、(3.5) 式における学習率は減少する方向に変化していく。これによって学習率が調整され、解周辺では更新量が小さくなるため収束が速くなる。(3.5) 式中の ε は、計算を行う上でゼロ除算が発生しないようにするため、きわめて小さい値を与える。Adagrad において、学習率は減少する方向にしか変化しない。そのため、目的の関数に対して適切に η を設定しておかなければ、解に到達する前に処理が終了してしまう可能性があるため、注意が必要である。

3.3 Word2Vec

3.3.1 Word2Vec

Word2Vec とは、単語の分散表現を得るための手法である。ニューラルネットワークの重み学習を利用し、推論を行うことによって単語の意味のベクトル表現を得るものである。Word2Vec は 2013 年に Google のトマス・ミコロフ氏らによって開発・公開がされたアルゴリズムである。Word2Vec によって可能となる事には以下のようなものが挙げられる。

- 単語同士の類似度計算
- 単語同士の加算、減算
- RNN と組み合わせた文章生成

単語同士の演算について、代表的な具体例が (3.6) 式のように類推を行うものであり、アナロジー問題と呼ばれる。Word2Vec によって生成されたベクトル空間上に “king”、“man”、“queen”、“woman” という単語が存在しているとする。これらの単語は、それぞれベクトルとして値を持っているため、(3.6) 式のような計算が成り立つ。

$$\text{“king”} - \text{“man”} + \text{“woman”} = \text{“queen”} \tag{3.6}$$

(3.6) 式は空間上の単語ベクトル同士の演算によって導出されているため、近い意味の単語が存在すれば他にもいくつかの関係を導出することも可能である。こうした操作は、Word2Vec によって学習されたモデルを用いることによって実際に行うことができる。

3.3.2 Word2Vec による単語間の類似度計算

Word2Vec では、学習済みモデルに対して単語を指定することで、様々な操作を行うことができる。最も基本的な操作として挙げられるのは、指定した単語のベクトル表現の取得である。これによって、指定した単語がベクトル空間上において、どこに位置するかを数値によって判断することができる。また、取得できる値は多次元ベクトルであるため、ノルム距離や cos 類似度を用いた単語間の距離計算および類似度計算が可能となる。

3.3.3 Word2Vec の問題点

Word2Vec には、いくつかの問題点が存在する。単語の分散表現における問題点とは、つまり単語の意味がベクトル表現として適切に反映されていないという点にある。具体的にどのような点で単語の意味が反映されていないかは、以下のものが挙げられる。

- 表記ゆれに対応できない
- 多義語の分離が困難
- 対義語の判別をしていない

まず、表記ゆれについてである。表記ゆれとは、日本語では “コンピュータ” および “コンピューター” のように、同じ単語でも表現される文字列として一致しない場合のことである。コンピュータにおいて、単語の文字列は一意に定まるものであり、文字が 1 つでも異なれば別の単語として認識される。そのため、表記ゆれのある単語に対して、それを同一の単語であると判別することは困難である。

次に、多義語についてである。多義語とは、“character（登場人物・文字）” や “capital（資産・首都）” のように、複数の意味を持つ単語のことである。Word2Vec では、1 つの単語に対して与えられるベクトルは 1 つであるため、複数の意味を同時にベクトルとして対応させるのは困難である。そのため、単語の意味を正しく表現することができないという問題が発生する。多義語の判別については、文脈を用いることによって BERT 等のアルゴリズムで改善が行われている。

最後に、本研究で主題として扱う対義語の判別についてである。これは、単語の分散表現の獲得において、分布仮説を用いていることに起因する。分布仮説では、周囲の単語によってその単語の意味が形成されるとしている。この仮説により、Word2Vec では同じまたは似た文脈で登場する単語の類似度が近くなるという特徴がある。しかし、対

義語は以下の例のように似た文脈で出現することが多く存在する。

“He has the new PC.”

“He has the old PC.”

この例文において、“new” と “old” の周辺に出現している単語は同一のものであるため、Word2Vec において 2 つの単語は似た意味を持つ単語として処理される。しかし、実際には “new” と “old” は対義関係にあり、これを似た意味として捉えるのは不当であると考えられる。このため、文脈を考慮しない Word2Vec においては、単に類似した単語として捉えられてしまう。本研究では、この対義語関係に関する問題の改善についての手法について検討を行う。

3.3.4 Word2Vec のモデル

単語の分散表現を用いるために、各所で Word2Vec の学習済みモデルが公開されている。1 つのモデルにおいて扱うことのできる言語は 1 種類であり、その多くは英語で作成されている。Word2Vec は英語以外にも、日本語や中国語等について学習されたモデルも公開されている。本研究では、Google のオープンソースプロジェクトにおいて作成された学習済みの Word2Vec モデルを用いる。

3.4 Python

3.4.1 Python

Python とは、グイド・ヴァン・ロッサム氏により開発された汎用プログラミング言語である。1991 年に初のリリースがされ、現在では数百万人ものユーザが利用しているとされている。Python の特徴として以下のような点が挙げられる。

- インタプリタ形式の対話的な言語
- オブジェクト指向プログラミング言語
- 移植が容易で、多くの Unix 系 OS、Mac、Windows で動作が可能
- オープンソースで運営されている
- コードの記述がシンプルであり、可読性が高いとされる
- 汎用的なライブラリから、専門的なライブラリまで豊富に用意されている
- 豊富なフレームワークにより、アプリケーション等の作成が容易

こうした特徴から、Web アプリケーションや人工知能をはじめとした様々な分野で活

用されており、多くのユーザから支持を得ている。

3.4.2 nltk[2)]

nltk(Natural Language Toolkit) は、自然言語処理を操作する Python プログラムを構築するための主要なプラットフォームである。nltk はスティーブン・バード氏らによって開発され、多くの国や大学で使用されている。WordNet をはじめ、分類、トークン化、タグ付けや解析など自然言語処理を行う上で一連の処理をすることができる。本研究では、nltk 内の WordNet のシソーラスをインポートする。また、WordNet のシソーラスから対義語の抽出を行う。

3.4.3 gensim[3)]

gensim は、自然言語処理で多く用いられるアルゴリズムを含むオープンソースのライブラリである。gesnsim は Python および Cython で記述されており、高速なアルゴリズムの実装を実現している。Radim Řehůřek 氏らによって開発された gensim は、医療や法律に至るまで様々な分野において研究や製品として多く用いられている。gensim の主なアルゴリズムには、以下のようなものがある。

- Word2Vec
- fastText
- Doc2Vec
- LDA

本研究では、gensim を用いて Word2Vec の学習済みモデルの読み込み、およびベクトル変換後のモデルの保存を行う。また、特定の単語ペアに対しての類似度計算や、類似単語一覧等の取得についても gensim を用いて行う。

3.4.4 cos 類似度

Word2Vec 内の単語ベクトル同士の cos 類似度の計算は、モデルをインスタンスとして gensim ライブラリ中の similarity 関数を用いて導出することができる。また、ある単語に対して最も cos 類似度の高い単語または cos 類似度の高い任意の数の単語を most_similar 関数を用いて導出することができる。本研究ではこの most_similar 関数を用いて、ベクトル変換の対象とする単語ベクトルの抽出を行った。さらに、評価実験を行う際にも、類

似度の高い単語集合の抽出にこの関数を用いた。

3.5　ベクトル変換手法 [4)]

3.5.1　目的関数

本研究におけるベクトル変換の目的は、学習済みの単語の分散表現中における対義語ペアの類似度を下げることにある。ベクトル変換を行う上では、予め対義語辞書を作成しておき、その辞書に登録されている対義語ペアのベクトル間距離が遠くなるようにベクトル変換を行う。しかし、ベクトル間距離を遠ざける処理を行うにあたり、対義関係にないほかの単語ベクトルとの距離が影響を受けることは単語の意味関係を崩してしまうため、望ましくない。そのため、対義関係にある単語のベクトル間距離のみを変化させ、他の単語ベクトルとの関係は出来るだけ変化しないようにベクトル変換を行う必要がある。

よって、これを実現するためのベクトル変換手法として提案されているのが、次の手法である。単語の分散表現中の任意の単語ペアに対して、そのペアが対義語辞書中に存在する場合は、その単語ペアのベクトル間距離が、それに一定の値 α を加算した値とできるだけ等しくなるようにする。加えて、対義語辞書中に存在しない場合は、そのベクトル間距離がベクトル変換前後でできるだけ等しくなるようにする。

ここで、単語の分散表現に含まれる単語のリストを $W_1, W_2, \ldots, W_n$ とし、単語 W_i の変換前ベクトルを $\boldsymbol{w_i}$、変換後ベクトルを $\boldsymbol{w_i'}$ とする。また、単語ベクトルは P 次元であるとすると、単語ペア W_i、W_j に対し、変換前ベクトル間の距離と変換後ベクトルの距離は以下の (3.7) 式と (3.8) 式のようになる。

$$d_{ij} = \sqrt{\sum_{k=0}^{P} (w_{ik} - w_{jk})^2} \tag{3.7}$$

$$d'_{ij} = \sqrt{\sum_{k=0}^{P} (w'_{ik} - w'_{jk})^2} \tag{3.8}$$

これを基に、m 個の単語ベクトルに対して目的のベクトル変換を満たすための目的関

数を以下の (3.9) 式に示す。

$$F = \sum_{i=0}^{m-1} \sum_{j=i+1}^{m} F_{ij}$$

$$= \sum_{i=0}^{m-1} \sum_{j=i+1}^{m} (d'_{ij} - (d_{ij} + \alpha_{ij}))^2 \tag{3.9}$$

(3.9) 式中の α_{ij} について、これは式中の i, j に依存する値であり、単語のペア W_i、W_j が対義語辞書中に存在している場合には、$\alpha_{ij} > 0$ の定数とする。また、単語のペア W_i、W_j が対義語辞書中に存在していない場合は $\alpha_{ij} = 0$ とする。

この (3.9) 式で表される目的関数 F を最小化する変換後ベクトル集合 $\boldsymbol{w'}$ を、最適化手法の Adagrad を用いて導出を行う。Adagrad の処理中において、単語 W_i とその対義語 W_j に関する変換後ベクトルの更新計算を行う。それと同時に、変換前の単語ベクトルにおいて単語 W_i との類似度が上位 N 位以内の単語についても、対義関係にない単語として更新計算を行う。

以上のベクトル変換手法を用いることで、対義関係にある単語ペアのベクトル間距離を大きくしつつ、それ以外の単語のベクトル関係は出来るだけ変化させないという目的を達成することができるとする。

3.5.2 目的関数の微分

勾配降下法および Adagrad で目的関数を最小化するベクトルを求めるにあたり、目的関数の微分系である $\frac{\partial F}{\partial \boldsymbol{w'_i}}$ を求める必要がある。コンピュータで微分を求める際には数値微分という方法がある。しかし、1 つの単語ベクトルは 300 次元であり、その要素すべてに対して数値微分を求めると、計算量が膨大となってしまい、実行時間が非常に長くなってしまう。そのため、ここでは目的関数の微分形をあらかじめ求めておくこととする。ニューラルネットワークにおいては逆伝搬法が多く用いられるが、この目的関数は単一のものであるため、形式的に微分形を求めることとする。目的関数の微分形は、以下のように求められる。なお、ここで用いる文字は、(3.9) 式において述べたものと同一であるとする。

初めに、式を簡単化するために、(3.9) 式中の $(d_{ij} + \alpha_{ij})$ は定数であるため、これを T_{ij}

とおく。(3.8) 式より、(3.9) 式の目的関数は次のように表せる。

$$F = \sum_{i=0}^{m-1} \sum_{j=i+1}^{m} \left\{ \sqrt{\sum_{k=0}^{P} (w'_{ik} - w'_{jk})^2} - T_{ij} \right\}^2 \tag{3.10}$$

また、合成関数の微分より、$\frac{\partial F}{\partial w'_{ik}}$ は以下の式で表せる。

$$\frac{\partial F}{\partial w'_{ik}} = \sum_{i=0}^{m-1} \sum_{j=i+1}^{m} \left(\frac{\partial F}{\partial d'_{ij}} \cdot \frac{\partial d_{ij}}{\partial w'_{ik}} \right) \tag{3.11}$$

ここで、(3.11) 式中の微分式はそれぞれ以下のようになる。

$$\frac{\partial F}{\partial d'_{ij}} = 2 \left(d'_{ij} - T_{ij} \right) \tag{3.12}$$

$$\frac{\partial d_{ij}}{\partial w'_{ik}} = \frac{1}{2} \left\{ \sum_{k=0}^{P} \left(w'_{ik} - w'_{jk} \right) \right\}^{-\frac{1}{2}} \cdot 2 \left(w'_{ik} - w'_{jk} \right)$$

$$= \left(w'_{ik} - w'_{jk} \right) \cdot \frac{1}{d'_{ij}} \tag{3.13}$$

よって、(3.9) 式の微分系は次の (3.14) 式のように表される。この式を勾配降下法に用いることで、目的関数を最小化するベクトルを求める。

$$\frac{\partial F}{\partial w'_{ik}} = \sum_{j=1|j\neq i}^{m} 2 \left(d'_{ij} - T_{ij} \right) \cdot \left(w'_{ik} - w'_{jk} \right) \cdot \frac{1}{d'_{ij}}$$

$$= \sum_{j=1|j\neq i}^{m} 2 \left(1 - \frac{T_{ij}}{d'_{ij}} \right) \cdot \left(w'_{ik} - w'_{jk} \right) \tag{3.14}$$

第4章　実験結果と考察

4.1　実験概要

本実験では、学習済みの単語のベクトルモデルに対して対義関係にある単語のペアの距離を離すベクトル変換を行う。ベクトル変換を行った後の単語の分散表現は、新たなベクトルモデルとして保存する。ベクトル変換は以下の手順を踏んで行われる。

1. 学習済みモデルの読み込み
2. WordNet から対義語のペアを抽出
3. 対義語のペアに対してベクトル変換
4. 2 と 3 を対象とする対義語のペアに対して行う
5. ベクトル変換後の単語のベクトルモデルを保存する

また、こうして完成した単語のベクトルモデルに対して、その妥当性および正当性の有無について、評価実験を行う。評価の対象とする点は、主に 2 つ存在する。まず 1 点目に、対義語が存在するある単語に対して、その対義語とのベクトル間距離および類似度が下がっているかについてである。これにより、対義関係を反映するためのベクトル変換が行われているかを評価する。次に 2 点目として、対義関係にない単語との関係を評価する。対義関係にない単語については、もとの単語のベクトルモデルとなるべく変化がないようにすることが理想である。そのため、ベクトル変換を行う前後のベクトルモデルにおいて、ある単語と類似しているとされる単語が変化していないかについての調査を複数の単語に対して行う。

よって本実験では、単語のベクトルモデルに対してベクトル変換を行い、その前後で対義関係が反映されたかどうか、また、対義関係にない単語についてのベクトル関係の変化の程度について調査を行い、考察する。

4.2　実験準備

4.2.1　実験環境の構築

本実験を行うための準備として、以下の内容を行った。

- Python および実行環境の構築
- WordNet, gensim およびその他ライブラリの導入
- Word2Vec の学習済みモデルの取得

4.2.2 Python および実行環境の構築

本実験では、Python を実行するために Python の Windows 用インストーラを用いて PC に Python3.0 を導入した。IDE として Visual Studio Code を用い、Python プログラムの実行環境には Ubuntu 18.04 LTS をインストールし、Visual Studio Code 上のターミナルにおいて WSL を利用した。WSL を利用することで、Linux 環境での開発が可能となる。

4.2.3 WordNet, gensim およびその他ライブラリの導入

シソーラスである WordNet は、nltk のパッケージから取得した。pip3 を用いて nltk のインストールを行い、本実験では WordNet の機能のみを用いるため、プログラム内の読み込みは WordNet のシソーラスのみとした。

Word2Vec の機能を用いる上で重要な gensim、およびその他の行列演算などに用いたライブラリは pip3 を用いてインストールした。本実験で作成したプログラムで用いられている主なライブラリとそのバージョンは以下の通りである。

- gensim 3.8.3
- nltk 3.6.5
- numpy 1.19.5

4.2.4 Word2Vec の学習済みモデルの取得 [5)]

単語の分散表現に対してベクトル変換を行うためには、Word2Vec のアルゴリズムに基づいて作成された学習済みの単語ベクトルを作成する必要がある。しかし、学習に用いるテキストや単語の数から、それを基に学習を進めるためには多大な時間を要する。そのため、本実験では Web 上で公開されている Word2Vec の学習モデルを取得し、使用することとした。

本実験で用いる Word2Vec の学習済みモデルは、Google Code アーカイブにおいて公開されている学習済み単語ベクトル (GoogleNews-vectors-negative300.bin) を用いた。この単語ベクトルは Google News データセットの一部（約 1000 億語）を用いて作成されたモデルである。このモデルには 300 万の英単語が含まれており、その単語 1 つが持つベクトルは 300 次元で構成されている。

4.3 ベクトル変換

4.3.1 モデルの読込・保存

ベクトル変換を行う対象である学習済みモデルの読み込みの際には、gensim ライブラリ内のメソッドを用いる。この際、本実験で使用する学習済みモデルの読み込みには、load メソッドではなく、load_word2vec_format メソッドを用いた。これら 2 つのメソッドの違いは、学習済みモデルのフォーマットの違いによるものである。

本実験で使用する学習済み単語ベクトルは 3.5GB を超過する容量があり、これを読み込むことは困難である。また、単語の数も膨大であるため、それぞれの単語すべてに対してベクトル変換を行うとすると、非常に時間を要することとなる。そのため、本実験では計算量低減のため、学習済みモデル内から取り出す単語およびそのベクトルを限ることとした。

よって、load_word2vec_format メソッド内のオプションは、binary=True、limit=100000 とした。binary オプションは本実験で使用する学習済みモデルがバイナリファイルであるため True となっている。また、limit オプションはモデル内から取り出す単語の数の上限を定める。本実験では、実験を行うにあたり十分な量であると考えられる 10 万単語を取り出すこととした。

ベクトル変換後の単語のベクトルモデルは、評価実験を行うために新たなモデルとして保存する必要がある。そのため、新たなモデルを保存する際には、load_word2vec_format メソッドに対応させ、save_word2vec_format メソッドを用いた。この際、binary オプションは True とした。

4.3.2 WordNet からの対義語の抽出

本実験では、対義語関係にある単語のペアに対してベクトル変換を行う。そのため、単語ベクトルモデル内に存在する単語の中から、すべての対義語ペアを取り出す必要がある。対義語ペアの取り出しは、WordNet の対義語辞書を用いて行う。

初めに、対象となる Word2Vec の学習済みモデルから、gensim の index2word メソッドを用いてモデルに含まれる単語のリストを取り出す。次に、取り出した単語に対義語が存在しているかを WordNet を用いて判定する。対義語が存在していれば、取り出した単語と WordNet から対義語として出力された単語が対義語ペアとなる。

この処理を行うにあたり、対義語ペアとして判定された単語を後にモデルから取り出

した場合を考える必要がある。この処理が行われた場合には当該の対義語ペアが 2 度取り出されることとなる。この場合、同じ対義語ペアに対して 2 度ベクトル変換を行うこととなり、不必要なベクトル変化が行われてしまう。そのため、一度取り出した単語を処理済み単語リストに登録し、WordNet の対義語の出力からその単語を除外する操作を行うことで多重処理の防止を行った。

加えて、ベクトル変換の対象とする対義語ペアの選定を行った。これは、学習済み単語ベクトルモデル内において、対義語ペアのベクトルが初めから遠く配置されている場合に対しての処理である。元からベクトル間距離が大きい場合には、その単語ペアの持つ単語の対義性がある程度反映されていると考える。そのため、ベクトル間距離の大きい対義語ペアに対してはベクトル変換の対象外とした。ベクトル変換の対象とする対義語の条件は、学習済みモデルから取り出した単語に類似しているとされる単語 30 位以内に対義語が含まれている場合のみとする。なお、この際用いる類似単語は gensim ライブラリに含まれる most_similar メソッドを用いて求めた。同一単語に対して類似単語 30 位以内に対義語が 2 語以上含まれる場合には、それぞれ異なる対義語ペアであるとして、個別にベクトル変換を行う対象とした。これらの単語の出現順による影響は、ベクトル変換が対義関係に無い周囲の単語ベクトルに与える影響が極めて小さいものであるとして考慮せずに行った。

以上の条件の下に対義語ペアの抽出を行い、対義語辞書の作成を行った。その結果得られた対義語ペア数は 802 個となった。その一部を以下の Table 4.1 に示す。

Table 4.1　List of extracted antonyms (partial)

number	base word	antonym
0	on	off
1	more	less
2	more	fewer
3	up	down
4	all	some
5	first	last
6	first	second
7	there	here
8	back	ahead
⋮	⋮	⋮

4.3.3　ベクトル変換

単語のベクトルモデルに対して、対義語ペアのベクトル間距離を大きくするベクトル変換を行う。また、この変換によって対義関係にない他の単語のベクトル関係には可能な限り影響を与えないようにする必要がある。これを達成するために 3.5 節で述べた目的関数を勾配降下法である Adagrad に適用した。

ベクトル変換手法は以下の手順で実装および実行した。

1. ベクトル変換の対象となるベクトルの抽出
2. 勾配の算出
3. 重みベクトルの更新
4. 2，3 を一定数繰り返す
5. 変換後ベクトルの獲得

初めに、ベクトル変換の対象となるベクトルの抽出について、本来であれば読み込んだ学習済みモデルに含まれる 10 万単語すべてに対して演算を行うことが理想である。しかし、10 万単語すべてに対して重み更新の計算を繰り返し、かつそれを対義語 802 ペア分だけ計算を行う必要があるため、膨大な計算時間を要する。そのため、計算量を低減する必要がある。よって基準となる単語に対して gensim の most_similar メソッドを用い

て導出した類似 100 位以内の単語ベクトルに対してのみ演算を行う。従来手法ではベクトル間距離の近いものを選定していたが、本実験では cos 類似度に基づいて導出を行った。対義語は基準の単語に対して類似 30 位以内で抽出しているため、対義語の単語ベクトルは必ず演算の対象となる。

次に勾配の算出については、(3.14) 式を基にして計算を行う。ただし、この式を単純に for ループで実装すると 3 重ループとなり、1 単語当たり 300 次元の配列の計算を行うとすると計算量のオーダーが大きい。加えて Python での for ループは計算速度が比較的遅く、一度の計算時間が長くなる。そのため、本実験では numpy 行列を用いて for ループを削減することで計算量の削減を行った。

重みベクトルの更新回数は、収束に十分な回数である 100 回として固定した。

4.4 評価方法

4.4.1 対義語ペアのベクトル関係

本実験においては、ベクトル変換を行う前後の単語のベクトルモデルを比較することによって、ベクトル変換手法の有効性を評価する。ベクトル変換の妥当性を評価する点の 1 つが、対義語ペアのベクトル関係である。単語の分散表現では、ベクトルが近いほど類似している単語とされる。そのため、ここでは対義関係にある単語のペアの類似度がどれだけ下がったかを求める。それぞれの対義語ペアに対してベクトル変換前後の類似順位を求め、その差分を平均することで対義語ペアに対してどれだけ影響を与えたかを評価する。

具体的には、はじめに 4.3.2 項において抽出された対義語ペアのうち、モデルから取り出された単語を基点語と呼ぶこととする。ここで取り出された基点語に対して gensim の most_similar メソッドを適用し、出力された類似単語からその基点語の対義語が何位に位置しているかを求める。この操作をベクトル変換前後それぞれのモデルにおいて行い、その順位の変化の平均を観測する。

4.4.2 対義語-非対義語ペアのベクトル関係

ベクトル変換の妥当性の評価を行うにあたり、対義関係にない単語同士のベクトル関係に影響を与えていないことも重要な評価点の 1 つである。これについての評価は、基点語と類似しているとされる単語がどれだけ変化していないかについてを評価する。

具体的には、対義語ペアの基点語について most_similar メソッドを用いて、ベクトル変換前後のモデルにおいてそれぞれ類似 30 単語を導出し、ベクトル変換前の類似 30 単語がベクトル変換後の類似 30 単語にどれだけ含まれているかを調べる。これを比率としたものを保存率と呼ぶことにする。この計算を基点語すべてに対して行い、その平均を求めることで対義関係にない単語とのベクトル関係の変化を調査する。

加えて、ベクトル変換前の類似単語のリストには対義語が含まれていることから、ベクトル変換前後での類似単語が完全に一致することはない。そのため、ベクトル変換前の類似単語に対義語が含まれていることを考慮する。ベクトル変換前の基点語に対する類似単語のリストは、類似単語から対義語を除外した上での 30 単語として、ベクトル変換後の類似 30 単語との重複を計算する。この条件下において保存率の計算を行った結果についても導出を行う。

4.5 実験結果

4.5.1 対義語ペアのベクトル関係

はじめに、ベクトル変換を行った結果について、いくつかの基点語に対するベクトル変換前後での類似 30 単語を出力させた結果を Table 4.2 から Table 4.6 までに示す。

まず Table 4.2 では、基点語 “on” に対して “off” という対義語が存在している。ベクトル変換前は 4 位に位置していたが、変換後は 30 位以降に落ちていることが分かる。しかし、意味的には同じである “Off” という単語が依然として 11 位と比較的高い順位を保っていることも分かる。Table 4.3 では、基点語 “back” に対して “ahead” が存在しており、16 位から 30 位圏外へと順位を落としていることがわかる。またその他の各表の基点語についても、同様にベクトル変換後に対義語の順位が落ちているということが確認できる。

次に、対義語 802 ペアの基点語に対して、それぞれの対義語のベクトル変換前後のモデルにおける類似順位を導出し、その平均を求めた。また、ベクトル変換前後における対義語の類似順位の変化についても導出を行った。その結果を次に示す。

- ベクトル変換前の順位：7.709
- ベクトル変換後の順位：59.31
- ベクトル変換前後での順位の変動：51.60

この結果からベクトル変換前後において、基点語に対する対義語の順位は平均して約 52 位ほど落ちていることが確認できた。

Table 4.2 Similar words (base:“on” - antonym:“off”)

number	before	after	number	before	after
1	On	On	16	at	o_n
2	upon	upon	17	beneath	after
3	onto	onto	18	out	along
4	off	in	19	instead	following
5	the	through	20	along	where
6	Off	down	21	until	squarely
7	o_n	around	22	following	instead
8	through	before	23	ahead	beyond
9	solely	the	24	where	near
10	in	out	25	from	until
11	squarely	Off	26	after	beneath
12	down	back	27	behind	into
13	before	ahead	28	under	behind
14	around	at	29	into	when
15	back	for	30	for	below

Table 4.3 Similar words (base:“back” - antonym:“ahead”)

number	before	after	number	before	after
1	out	out	16	ahead	before
2	down	off	17	returned	start
3	off	down	18	bounce	Back
4	then	into	19	returning	right
5	again	again	20	somewhere_else	after
6	into	then	21	up	forth
7	away	away	22	right	in
8	Back	onto	23	rest	behind
9	onto	in.	24	when	returning
10	in.	up	25	closer	somewhere_else
11	return	around	26	start	starting
12	forth	through	27	regain	where
13	forward	just	28	reclaim	rest
14	finally	return	29	bounced	closer
15	just	returned	30	regroup	on

Table 4.4 Similar words (base:“best” - antonym:“worst”)

number	before	after	number	before	after
1	finest	finest	16	coolest	fantastic
2	worst	greatest	17	fastest	safest
3	greatest	smartest	18	hottest	hottest
4	strongest	strongest	19	purest	purest
5	smartest	healthiest	20	perfect	fastest
6	easiest	easiest	21	truest	dumbest
7	good	terrific	22	nicest	weakest
8	quickest	quickest	23	prettiest	truest
9	healthiest	cleanest	24	fantastic	prettiest
10	terrific	toughest	25	happiest	nicest
11	toughest	good	26	favorite	happiest
12	cleanest	better	27	sweetest	favorite
13	safest	coolest	28	smoothest	Best
14	better	great	29	excellent	hardest
15	great	perfect	30	weakest	craziest

Table 4.5 Similar words (base:“public” - antonym:“private”)

number	before	after	number	before	after
1	pubic	pubic	16	citizens	national
2	Public	Public	17	taxpayer_dollars	taxpayer_funded
3	private	PUBLIC	18	public_servants	citizens
4	PUBLIC	government	19	local	public_servants
5	citizenry	municipal	20	national	community
6	municipal	private	21	nonpublic	nonpublic
7	publics	publics	22	community	taxpayers
8	taxpayer	citizenry	23	corporate	health
9	government	taxpayer	24	health	taxpayer_dollars
10	civic	civic	25	populace	federal
11	taxpayers	general	26	stakeholder	transparency
12	publicly	governmental	27	taxpaying	stakeholder
13	governmental	publicly	28	administration	administration
14	general	corporate	29	ratepayer	privatizing
15	taxpayer_funded	local	30	publicize	populace

Table 4.6　Similar words (base:“top” - antonym:“bottom”)

number	before	after	number	before	after
1	Top	Top	16	elite	third
2	tops	tops	17	ranked	elite
3	upper_echelon	highest	18	strongest	strongest
4	highest	upper_echelon	19	hottest	No.2
5	No.1	fifth	20	Topping	ninth
6	No.	No.1	21	leading	notch
7	fifth	No.	22	ninth	Topping
8	ranking	sixth	23	No.2	rankings
9	sixth	ranking	24	rankings	leading
10	Top_Ten	best	25	topping	topping
11	bottom	seventh	26	upper_echelons	hottest
12	seventh	atop	27	topped	topped
13	atop	eighth	28	second	upper_echelons
14	best	Top_Ten	29	notch	pre_eminent
15	eighth	second	30	pre_eminent	midlevel

4.5.2　対義語-非対義語ペアのベクトル関係

基点語に対する類似30単語をベクトル変換前後で取り出し、ベクトル変換前の類似30単語が、ベクトル変換後の類似30単語にどれだけ含まれているかを調査した。ここで、保存率はベクトル変換後の類似30単語に、ベクトル変換前の類似30単語に含まれる単語がどれだけ含まれているかを示す。そのため、保存率は取り出した単語数である30中の、ベクトル変換前後で共通している単語の数の割合となる。

また、ベクトル変換前の基点語の類似30単語に対義語が含まれることを考慮した上での保存率についても計算を行った。変換前の類似30単語を、対義語を除外した上での30単語として保存率を調べた。それぞれの結果を以下に示す。

- 保存率（対義語の考慮なし）：0.8813
- 保存率（対義語の考慮あり）：0.8774

ベクトル変換前後での保存率は 9 割近くであり、つまり取り出した類似 30 単語中約 27 単語が変換前後で類似順位があまり変化していないということがわかる。また、対義語の考慮を行った場合には、保存率がわずかに減少した。

この結果から、最終的にベクトル変換前後の単語の保存率は約 9 割程度であり、本実験において行ったベクトル変換による非対義関係の単語同士のベクトル間距離は大きく変化していないことがわかる。

4.6 考察

4.6.1 対義語ペアのベクトル関係

対義語ペアのベクトル関係について、Table 4.2 から Table 4.6 までによって、基点語に対する対義語の順位およびその周囲の単語の変化についての結果が示された。また、ベクトル変換前後の基点語に対する対義語の類似順位は平均して約 52 位ほど落ちたことが確認できたことから、本実験で行ったベクトル変換によって対義関係にある単語同士のベクトルを遠ざける処理は有効なものであったと考えることができる。

従来のベクトル変換手法 [4] と比較すると、基点語に対する対義語の類似順位の変化は抑えられているように考えられる。本実験では大規模な学習済みベクトルモデルを用いたが、対義語同士のベクトル間距離をそれほど大きくできないベクトル配置となっていた可能性が考えられる。しかし、基点語に対する対義語の類似順位の下がった幅は約 52 位と少なくない値をとっているため、本実験でのベクトル変換による処理は有効なものであったと考えた。

ただし、いくつかの表内で確認できる通り、類似単語の中には “off” と “Off” や “public” と “PUBLIC” のように、大文字が区別されていないために異なる単語として処理されているものがある。その他にも、アンダーバー等によって、同じ単語であるにも関わらず、異なる単語として処理されている単語が散見される。これは、3.3.3 項において述べた、表記ゆれに関する問題である。

これらの表記ゆれの表現は、WordNet には登録されていない。そのため、対義語を抽出する際に対義語として判定されなかったと考えられる。WordNet は、人の手によって作成されているため、このような表記ゆれに自動的に対応して対義語の判別を行うことは困難であると考えられる。さらに、大文字を小文字に統一する程度であれば簡単な処理によって実装できるが、細かな単語の分散表現を獲得する際に、厳密には異なる単語

に対して同一である判定をしてしまう可能性も考えられるため、表記ゆれは対義語辞書を作成する上でも大きな課題であると考えられる。

本実験においてこのような表記ゆれが多く見られたことの原因の 1 つには、ベクトル変換に用いた学習済みの単語ベクトルモデルが大規模なものであったため、特にこのような表記ゆれのある単語がモデルに含まれることとなったと考えられる。

4.6.2 ベクトル変換による影響

対義関係にない単語同士の関係については、基点語に対して約 9 割の単語に変化が無かったことが確認できた。このことから、本実験において実装したベクトル変換手法は、ベクトル変換を行う上での周囲への影響はある程度低減することができたと考えられ、妥当なものであったと考えられる。対義語を考慮した計算を行った際に保存率が若干の低下を見せたのは、ベクトル変換後での類似 30 単語以内に、対義語が含まれている場合があるためであると考えられる。

ただし、Table 4.2 から Table 4.6 の類似単語を参照すると、Table 4.4 のようにベクトル変換前後で類似単語の出現順が大きく変動していないものもあるが、ほとんどの場合において出現順が前後していることがわかる。このことから、基点語の周囲に出現する単語に大きな変化は見られないが、詳細に出現順を確認した場合、ベクトルの関係に若干の変化が生じていると考えられる。

この問題点を緩和するために考えられる手法は、ベクトル間距離および類似度の計算に関して、cos 類似度を用いた距離を用いる事が考えられる。本実験ではベクトル間距離の計算に際してノルム距離を用いたが、その代用として 1 から cos 類似度を差し引いたものを距離として扱うことが考えられる。本実験においてはベクトル変換の対象とする単語ベクトルの選定に most_similar メソッドを用いた。これは cos 類似度に基づいたものであるため、これに即して cos 類似度を用いた距離計算を行う手法が有効ではないかと考えられる。

第5章　結論

本研究では、学習済みの単語のベクトルモデルに対して、従来よりも大規模な学習済みモデルを用いて対義関係を反映するためのベクトル変換を行った。また、ベクトル変換を行った単語ベクトルモデルに対して、対義関係にある単語ベクトル同士の類似度が離れているか、および対義関係に無い単語ベクトルとの関係に大きな影響を与えていないかという点について実験を行い、正当性の評価を行った。

手順としては、はじめに学習済みの単語のベクトルモデルの取得、WordNet を用いた対義語辞書の作成、そしてベクトルモデルに対するベクトル変換を行った。ベクトル変換の際には、gensim の most_similar メソッドを用いてベクトル変換の対象とするベクトルの選定を行った。その後、ベクトル変換を行う前後のモデルに対して、基点語の対義語の類似順位を導出し、その差分をとることで対義関係にある単語同士の類似度がどれだけ下がったかを求めた。また、ベクトル変換前後において、基点語に対してその周辺に現れる類似 30 単語にどれだけ共通の単語があるかを求めた。

ベクトル変換前後における対義語の類似順位の導出結果から、ベクトル変換によって対義関係にある単語ベクトル同士の類似度が下がっていることが分かった。これにより、本研究において実装したベクトル変換手法は、特定のベクトルの距離を大きくするために有効であることが確認された。また、基点語の類似 30 単語についても約 9 割程度が一致し、ベクトル変換前後によって周囲に与える影響は抑えることができていると考えられた。これらの事から、学習済みの単語のベクトルモデルに対して対義関係を反映するためのベクトル変換として、本件研究において実装した手法はある程度有効なものであると考えた。

しかし、本研究で実装したベクトル変換手法においては、新たに大きく 2 つの課題を確認した。まず 1 つ目に、単語のベクトルモデルに単語の表記ゆれが含まれていることである。対義語辞書を作成するにあたり本研究では WordNet を用いたが、WordNet には表記ゆれはなく、1 つの単語として存在する。しかし、ベクトルモデルにおいては大文字やアンダーバーが含まれている場合がある。これによって、対義語であっても対義語辞書に登録されずに処理を通過してしまう。特に、本研究において用いた学習済みモデルは特に大規模なものであったため、表記ゆれしている単語が多く見られたと考えられる。そのため、表記ゆれを統一する、または同じ単語を判定するメソッドを用意し、す

べての対義語に対して処理を行う必要があると考えられた。2 つ目は、基点語に対する類似単語の出現順の変化である。本研究内では、基点語に対する類似単語として得られる単語はベクトル変換前後でほとんど変化しないという結果が得られた。しかし、厳密にベクトル関係が全く変化していないわけではなく、ベクトル関係に微妙な変化が生じたために、類似単語の出現順が変化したと考えられた。そのため、距離の計算方法等を工夫する必要があると考えられた。

今後の展望として、本研究では単語の類似度に注目して評価を行ったが、単語ベクトルの差分関係が維持できているかについて評価を行う必要がある。単語ベクトルの差分関係は、単語の意味計算を行う上で重要である。この評価を行う上では、(3.6) 式に示したような単語のセットが必要である。そのため、十分な量の単語セットを用意して、ベクトル変換前後で差分関係が変化していないか評価を行うことになる。

本研究において作成された単語の分散表現モデルを用いることによって、文章分類や言い換え文の作成などのタスクにおける精度の向上が期待される。実際にベクトル変換を行ったモデルを用いることで、これらのタスクにどのような影響があるのかについても検証することが望ましいと考えられた。

謝辞

最後に、本研究を進めるにあたり、ご多忙中にも関わらず多大なご指導をしていただきました出口利憲先生、また、共に勉学に励んだ同研究室のメンバーに厚く御礼申し上げます。

参考文献

1) 斎藤 康毅 著, ゼロから作る Deep Learning 2, オライリージャパン, 2020.

2) NLTK Project, nltk, 2022. https://www.nltk.org

3) Radim Řehůřek, gensim, https://radimrehurek.com/gensim/

4) 別所 克人, 浅野 久子, 富田準二, 対義性を反映する単語ベクトル変換手法の検討, 人工知能学会研究会資料 知識ベースシステム研究会 112 回, 人工知能学会, 2017. https://www.jstage.jst.go.jp/article/jsaikbs/112/0/112_02/_article/-char/ja/

5) Google Code Archive, Word2Vec - Google Code, 2013. https://code.google.com/archive/p/word2vec/