平成２８年度茨城大学工学部情報工学科
卒業研究論文

真の負例を用いたNegative Sampling による分散表現の構築

平成29年 2月 10日提出

茨城大学 工学部情報工学科
12T4047S 冨岡 雄磨
指導教員：新納 浩幸 教授

平成 ２８ 年度茨城大学工学部情報工学科卒業研究

真の負例を用いたNegative Sampling による分散表現の構築

氏名：12T4047S 冨岡 雄磨
指導教員：新納 浩幸 教授

論文要旨

本論文では真の負例を用いたNegative Samplingにより、より適切な分散表現の構築を行う。

Word2Vecとはニューラルネットワークを用いてコーパスから各単語の意味をベクトル化する手法である。従来のBag of Wordsモデルによる意味ベクトルを次元縮約するプログラムとも見なせる。この次元縮約された単語の意味ベクトルを分散表現と呼ぶ。

Negative SamplingとはWord2Vecを高速化させる手法であり、この手法を用いることで膨大な計算量を大きく減らすことができる。だがNegative Samplingについて調べると理論とプログラムの間に乖離がある思われる部分が存在した。理論上では、ある単語の存在する文脈で出現している単語 (共起している単語) と、ランダムに抽出した出現していない単語 (共起していない単語) との差異に基づいて単語の意味ベクトルを学習していくが、プログラムではその部分で出現した回数を確認せずにランダムに単語を取得している。そこで、本論文では共起していない単語 (真の負例) のみを取得するようプログラムを修正し、Word2Vecによって作成される分散表現にどのような変化があるか、またどれだけ結果が安定するかを確認する。

実験では、Word2Vecのプログラム内のNegative Samplingの部分にフィルターを作成し、真の負例でなければ再度単語を抽出するように修正を行った。そして修正前のプログラムと修正後のプログラムで複数回Word2Vecを動かし値を確認した。その結果、全体の分散はおよそ7%ほど小さくなったが、分散が小さくなったことで値が安定していることは確認できなかった。

以上から、少しでも高い精度を求めたい場合を除き真の負例を用いる必要はないという結論に達した。

目 次

表 目 次

図 目 次

第1章　はじめに

1.1　研究の背景

Word2Vec というツールは自然言語処理の分野の最先端のツールであり、文字通り単語をベクトルとして表現することで単語の意味をとらえるという明快さとその応用性の広さから自然言語処理の分野の壁を超えて活躍している。

そんな Word2Vec について調べていた際 Negative Smapling の理論が十分に再現されていないと思われる部分を発見した。Negative Sampling とは、Skip-gram モデルの soft-max 関数で、近似式を用いることで計算回数を大幅に減らすという効果を持つ方式であり、ある単語”x” の存在する文脈で出現している単語と、アトランダムに抽出した出現していない単語との差異に基づいて単語の学習をしていくというものだ。しかし、Word2Vec として配布されているプログラムを確認すると、出現した単語と出現していない単語を区別せずに取得していたのである。

しかし、いくら調べてもプログラム上で区別していない理由についての回答は何一つ存在しなかった。そのため、自身でその理由を確認するべきだと判断し研究を行うことに決めた。

1.2　研究の目的

Negative Sampling のプログラムを理論に近づけ、、修正前と後の得られた値を照らし合わせ、修正する必要性と修正されない理由を考察することを目的とする。

1.3　研究の構成

本論文は以下のように構成される
第 1 章は本章であり、本研究の背景及び目的、構成について概説する。
第 2 章では分散表威厳及びその利点について概説する。
第 3 章では Word2Vec の特徴や性質について説明を行う。
第 4 章では Negative Sampling についての概説を行い、真の負例を選択する意義を説明する。第 5 章では実際に実験を行い、その手順や使用したデータと手法について説明する。
第 6 章では実験の結果得られた値を解析し、考察を行う。
第 7 章では本研究全体の総括を行う。

第2章　分散表現

2.1　概要

分散表現とは、その単語の意味を 200〜1000 次元程度のベクトルで表現したものである。

近年ディープラーニングなどの分野でも深く研究されている人工ニューラルネットワークの研究で提唱された考え方あり、「単語の意味はその周辺単語の分布により知ることができる」という分布仮説から、同じ文脈から出現する単語は同じ意味を持つというものだ。

そこで、その周辺単語の分布を表す最も簡単な方法として「共起単語の出現分布[1]」が存在し、共起単語の出現分布をベクトルで表現したものは BOW(bag of word) のベクトルと呼ばれる。それを次元縮約することで分散表現を得ることができる。

2.2　共起単語の出現分布

たとえば、「それは戦いの道具ではない」という文章の、「戦い」を「争い」に変えたとしても意味は通じる。そこで、「戦い」および「争い」に関する入力が「それは戦いの道具ではない」と「それは争いの道具ではない」の 2 文のみであったとすると、共起単語は以下のようになる。

上の表から分かるとおり、入力が上記の 2 文のみであった場合、「戦い」と「争い」は全く同じ意味の言葉であると判断される。

この 2 単語に以下のような多数の入力があると、
「それは戦いの道具ではない」
「今日も僕は戦いに行く」
「眠気との戦いに負けた」

[1] ある単語と同時に文章内に出現した回数を表にしたもの。

表 2.1: 共起単語の表

	それ	は	の	道具	では	ない
戦い	1	1	1	1	1	1
争い	1	1	1	1	1	1

「それは争いの道具ではない」
「母親との言い争いに負けた」

表 2.2: 次元数の増えた共起単語の表

	それ	は	の	道具	では	ない	今日	も	僕
戦い	1	2	2	1	1	1	1	1	1
争い	1	1	2	1	1	1	0	0	0

	に	行く	眠気	と	負け	た	母親	言い	
戦い	2	1	1	1	1	1	0	0	
争い	0	0	0	1	1	1	1	1	

となり、2 単語のコサイン類似度は 0.377。単純に考えると、「戦い」と「争い」は 37.7%の確率で同じ意味を持つと言える。

このようにして多数の単語の共起数から計算していくことで、各単語がどれくらい似ているかが分かるようになる。

2.3 次元縮約

先ほどのように共起数をとっていくと、最終的に単語数×単語数次元の行列ができる。しかし、そのほとんどの値は 0 であり、膨大かつスパースな行列になっている。

そこで、行列に対し次元縮約を行い、200〜1000 次元程度の行列にする。それを行うプログラムが次章で紹介する”Word2Vec” である。

第3章　Word2Vec

3.1　概要

Word2Vec とはグーグルの研究者であるトマス・ミコロフ氏が提案した手法であり、分布仮説に基づいて制作され、いくつかの問題について従来のアルゴリズムよりも飛躍的な精度向上を可能にした。

Word2Vec は単語をベクトル化して表現するする定量化手法であり、この手法によって得られるベクトル空間には、今まで定量的に捉えることの難しかった言葉の「意味」を極めて直接的に表現しているかのような性質が認められている.。

Word2Vec では各単語を 200 次元程度の分散表現として表現する。Word2Vec は人工ニューラルネットワークの研究から生まれたものである。ある単語が与えられたときに、近くに出現する他の単語 (前後 5〜10 単語ほど) を当てるという問題の解を、与えられた文章中の単語全てに対して人工ニューラルネットワークに学習させる。似た意味の言葉はお互い近くにあらわれる可能性が高いため、学習を行っていく過程で徐々に近しい方向のベクトルになっていく。そのベクトルの類似度を見ることで、この単語は○○と近しい意味を持つということが簡単にわかるようになる。

3.2 学習モデル

Word2Vec を提案したミコロフ氏は 2 つの学習モデルを提唱している。それぞれ ”CBOW ”と ”Skip-gram ”と呼ばれている。

これらはもともと NNLM(Neural Network Language Model) というニューラルネットワークを用いた学習手法の計算を簡略化させ、NNLM のもつ精度を多少下げてでも計算スピードの向上を狙ったものである。しかし、実行速度だけでなく正解率も大きく向上しており、特に Skip-gram が高いパフォーマンスを見せた。

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days x CPU cores]
			Semantic	Syntactic	Total	
NNLM	100	6B	34.2	64.5	50.8	14 x 180
CBOW	1000	6B	57.3	68.9	63.7	2 x 140
Skip-gram	1000	6B	66.1	65.1	65.6	2.5 x 125

図 3.1: 学習手法ごとのパフォーマンス

Word2Vec に採用されているのは Skip-gram であるが、関連性が高いため双方を説明する。

3.2.1 Countinuous Bag of Words モデル (CBOW)

投射層の共有により、語順の影響が無視され言葉のまとまり (Bag) として、データ群を眺めることになるため、このように呼ばれている。。（これは画像処理系のニューラルネットでよく知られた手法である。）

CBOW は、「ある単語 x が出現する文脈」で出現している「他の単語 z」を利用して x を推定することを目的としている。

具体的には、単語 x の前後 n 単語を x を推定するための特徴量として用いることとなる。つまり、

$$\mathbf{z_{i-n}}, \cdots\cdots, \mathbf{z_{i-1}}, \mathbf{z_{i+1}}, \cdots\cdots, \mathbf{z_{i+n}}$$

の中に単語 x が i 番目に出現する確率

$$\mathbf{p}(\mathbf{x}|\mathbf{z_{i-n}}, \cdots\cdots, \mathbf{z_{i-1}}, \mathbf{z_{i+1}}, \cdots\cdots, \mathbf{z_{i+n}})$$

を学習することになる。

このモデルで計算される「ある単語 x が出現する文脈」で出現している「他の単語 z」を利用して x を推定するというものは、

$$\mathbf{p}(\mathbf{x}|\mathbf{z_{i-n}}, \cdots\cdots, \mathbf{z_{i-1}}, \mathbf{z_{i+1}}, \cdots\cdots, \mathbf{z_{i+n}})$$

という条件付確率を計算することであり、v を語彙集合とし、x'を語彙集合内に含まれている単語、X'をその意味ベクトルであるとすると、条件付確率は soft-max 関数を用いて以下の式でに計算できる。

$$\frac{\exp(\mathbf{X^t X})}{\sum_{\mathbf{x'} \in \mathbf{v}} \exp(\mathbf{X^t X'})}$$

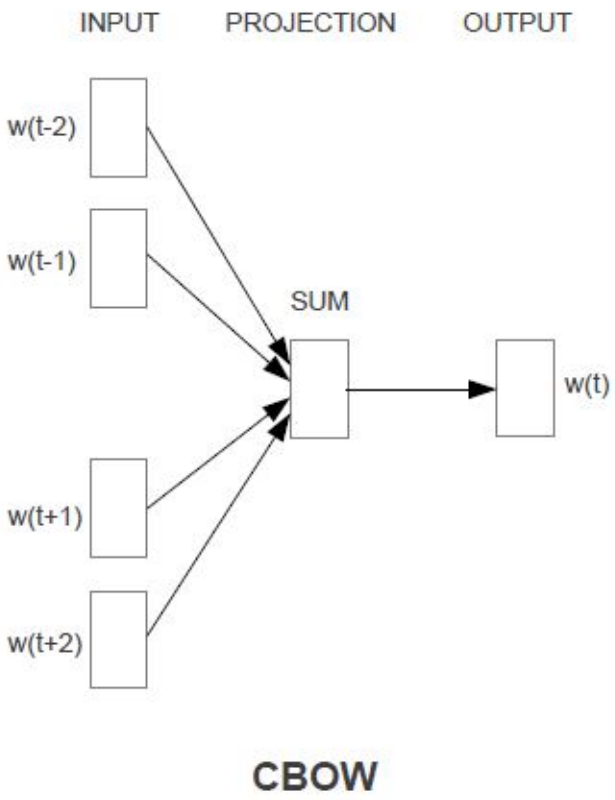

図 3.2: NN のモデル (CBOW)

3.2.2 Skip-gram

単語データを前後の文脈ではなく、前後から離れた文中の言葉の分類に用いる。N-gram は N 文字を指示するが、接頭辞 skip-との組合せから、特徴抽出の言葉のポジションが飛び飛びになっていると想像できる。

CBOW は、「ある単語 x が出現する文脈」で出現している「他の単語 z」を利用して x を推定することを目的としているのに対し、Skip-gram は「ある単語 x が出現する文脈」で、「他の単語 z」を推定することを目的としている。このモデルで求められるのは、ある単語 x が文脈中に出現している場合の別の単語 z が出現する条件付確率のため、

$$\mathbf{p}(\mathbf{z}|\mathbf{x})$$

となる。

$v(x)$ を x が出現している文脈中に出現する単語の集合とすると、条件付確率は soft-max 関数を用いて以下の式で計算できる。

$$\frac{\exp(\mathbf{X^t z})}{\sum_{\mathbf{x'} \in \mathbf{v}} \exp(\mathbf{X^t z'})}$$

身もふたもない言葉にしてしまえば、CBOW の逆のことをやりたいと考えニューラルネットワークの形や式の導出をそのまま逆にしただけである。

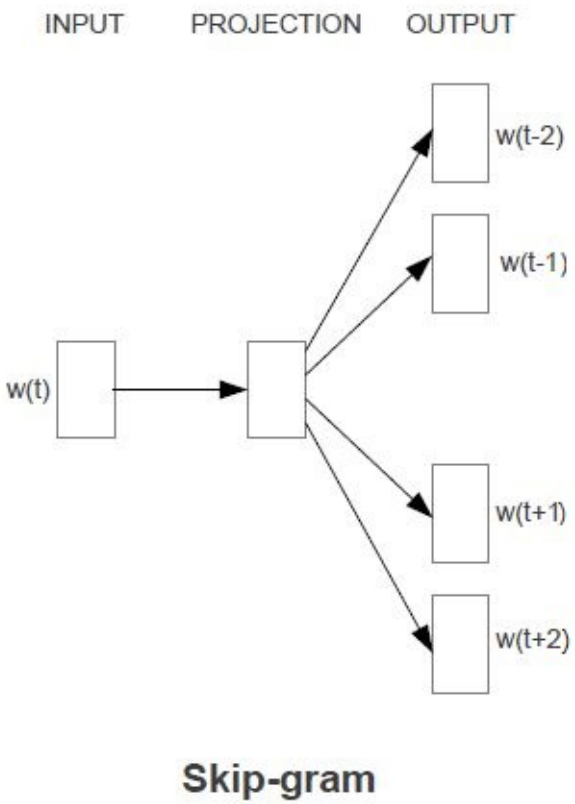

図 3.3: NN のモデル (Skip-gram)

3.3　次元縮約

Word2Vec の次元縮約はニューラルネットワークにて行われている。しかし、現在ニューラルネットワークの内部では何が起きているのかということが、実は全く分かっていない完全なブラックボックスなのである。

こんな感じのニューラルネットワークを作ると出力がこうなるということはわかっているのだが、同じ入力を行っても出力値は一意に定まらず、毎回微妙な誤差が生じてしまう。

そのため、内部で何を行っているかの逆算をすることもできず、Word2Vec のニューラルネットワークを通すと次元縮約が行われ分散表現が生成されるということしか言えない。

3.4　Word2Vec の性質

Word2Vec には他の分散表現を構築する手法とは違う、驚異的な性質が存在する。それは、関係性がベクトルにも表れるということだ。

最たる例として、単語同士の計算ができることがあげられる。例えば、

「king」-「man」＋「woman」＝「queen」

というようになる。

これは”king” から男性的な側面”man” を引き、女性的な側面”woman” を足すと”queen” になるというものだが、単語同士の関係性を理解していないとできない答えであり、これまでの分散表現を作成するアルゴリズムでは不可能だったことだ。

他には”good” と”best” のベクトルの中間に”better” が存在したりするなど、単語同士の関係性がベクトルに表れていることが分かる。データとして入力したものは文章のみで、その関係性を教えられたわけではないのに関係性を理解しているというのは驚異的である。

この、関係性がベクトルに表れるという性質を利用して、評判分析や語義曖昧性解消などの学術的なこと以外にも、様々な面白い取り組みがなされている。

第4章　Negative Sampling

4.1　概要

Negative Sampling とは、Word2Vec の動作を高速化させる手法の一つである。Word2Vec の出力層でモデル化を行っているが、その式が以下のようになる。

$$\mathbf{p(z|x)} = \frac{\exp(\mathbf{X^t z})}{\sum_{\mathbf{x'} \in \mathbf{v}} \exp(\mathbf{X^t z'})}$$

この式を見て分かるとおり、ある単語についての計算を行うために V 回の計算を行う必要がある。一般的に V の語彙数は数万〜数百万単語であり、この部分の計算コストがとても大きいものとなっているため、式 (4.1) を計算コストの低い式に近似する Negative Sampling が用いられる。

$p(D = 1|x; z)$ 及び $p(D = 0|x; z)$ という確率を考える。前者は単語 x と z が共起してコーパス D に出現する確率，後者は出現しない確率である。これら確率を用い、上の式を以下のように近似する。

$$p(z|x) \approx p(D = 1|x; z) \prod_{x' \in Ng} p(D = 0|x', x)$$

あとはこれを変形してから計算を行うと答えが出るのだが、計算回数のみに着目すると V 回必要だった計算回数が Ng 回となっていることが分かる。この Ng はもともとの単語の集合から、不正解単語 (ノイズ) として k 個だけサンプリングしてきたものとなっている。つまり、計算回数は k 回である。この k の値は通常 5〜20 程度で十分であり、またデータセットが十分に大きければ 2〜5 個程度でも十分な性能を発揮する。

この式を最小化することで softmax 関数の値と近似し、V 回の計算が必要だったものをわずか k 回の計算にすることができるのである。

4.2 計算式

$$p(D = 1|x, z)$$

上記の式は単語 x と z が共起してコーパス D に出現する確率である。逆に出現しない確率は、

$$p(D = 0|x, z) = 1 - p(D = 0|x, z)$$

である。まずはパラメータ θ を仮定し、

$$p(D = 1|x, z;\ \theta\ )$$

とする。この時目的となるのは、上記の確率を最大化することであるため、以下のようになる。

$$\arg\max_{\theta} \prod_{(x,z)\in D} p(D = 1|x, z;\ \theta\ )$$

$$= \arg\max_{\theta} \log \prod_{(x,z)\in D} p(D = 1|x, z;\ \theta\ )$$

$$= \arg\max_{\theta} \sum_{(x,z)\in D} p(D = 1|x, z;\ \theta\ )$$

$p(D = 1|x, z;\ \theta\ )$ の値は soft-max を利用することで設定できる。

$$p(D = 1|x, z;\ \theta\ ) = \frac{1}{1 + e^{-x' \cdot\ z'}}$$

よって、目的関数は以下のようになる

$$= \arg\max_{\theta} \sum_{(x,z)\in D} \log \frac{1}{1 + e^{-x' \cdot\ z'}}$$

4.3　真の負例を選択する意義

理論上では、ある単語”x” の存在する文脈で出現している単語 (共起している単語) と、ランダムに抽出した出現していない単語 (共起していない単語) との差異に基づいて単語の学習をしていくのだが、実際はランダムに取得されており、実際は正例である単語を負例として扱う可能性がある。そこで、真に負例である単語のみを利用することで理論と同じ動きをするように修正するのである。

とはいうものの、完全にランダムで不正解単語を選んでいる現在でも、例えば今回の実験で用いる”ptb.train.txt” であれば、およそ 97%の単語の組み合わせが負例であり、わずか 3 %の正例の場合のみの改善である。

第5章　実験

5.1　概要

実験では、プログラミング言語は”python”、機械学習やディープラーニングを動かすフレームワークである”chainer” を利用する。

Word2Vec のプログラムは Chainer による実践深層学習 [6] の書籍をもとに作成した。そのプログラムの次元縮約後の次元数は 200 次元である。Negative Sampling のプログラムは、chainer の example として公開されている Word2Vec のプログラムセット内のものを用いた。実験用のコーパスとしては”ptb.train.txt” という 1 万種類の単語を持つ chainer の評価用コーパスを用いた。

計測方法だが、2 単語間のコサイン類似度をとることで、間接的に分散表現の計測をすることにした。2 単語のペアは、WordSim353 という人が手作業で付けた 2 単語間の類似度を 353 種類まとめたデータセットから、ptb.train.txt で利用可能であった 226 ペア[2]を採用した。

以下実験の手順だが、まず、作成したプログラムを動かし、結果を記録することを繰り返し、それをデフォルトの場合の結果として、出てきた値の平均値と分散をとる。

次に作成したプログラムの Negative Samplling へ値を渡す部分にフィルターを作成し、真の負例でない場合は再度値を得るようにする。そして、作成したプログラムも同様に動かし、結果を記録することを繰り返し、それを修正後の場合の結果として、出てきた値の平均値や分散をとる。

最後に、出てきた値をもとに考察を行うものとする。

[2]実際は 227 ペアなのだが、除外した一つは 2 単語がともに”tiger” という同じ単語のペアであったため、コサイン類似度が必ず 1 になるという結果が分かり切っているので、この実験からは除外した。

5.2　計測方法

本来は分散表現を計測の対象としたかったが、そもそも内部でどのようにベクトルが 200 次元に圧縮されているかが全く分からないため、毎回同じような形式で分散表現が作られているのかということさえ不明である。

そのため採用された 2 単語間のコサイン類似度の分散は、同形式で同文書から作成されているためにおおよそ一定の値を示すことや、ベクトルを見せられるよりも数値として読み解きやすいという性質から、これが最善であると判断した。

5.3　使用データ

実験用コーパスとしては ptb.train.txt を用いた。

これは、chainer がテストのために用意した 1 万種類の単語を持つコーパスである。文書全体の単語数は 88 万 7 千単語ほどである。大文字は小文字に変換されており、単語の種類を減らすため、数字が N、未知語ををを全て <unk> に変換されており、N と <unk> を含めて 1 万種類の単語を持っている。

計測対象としては、WordSim353 を用いた。

これは、E. Gabrilovich. が作成した単語類似性または相関性のテストに用いられるデータセットである。353 個の単語のペアに対して、人間が手作業で付けた類似度の平均をとったものであり、人間的な感性で正しく思えるような値が振られているため、膨大なデータからこの値と近い類似度を出すことができたならおおよそ人間の感性と同じような処理を持つプログラムであると言えるだろう。当初、プログラムを修正することにより、人間の感性と近い値になる可能性を考えこのデータセットの単語のペアを利用した。

5.4　プログラムの実行

まず、Chainer による実践深層学習 [6] をもとにプログラムを作成した。

続いて作成したプログラムを、外部に各単語の共起回数を記したリストを作成しそれをもとに共起単語のペアであるか否かを判別するように修正したプログラムを作成した。

作成したプログラムは、付録として添付した。

その後、作成した 2 つのプログラムを回して分散表現を作成し、226 種類ある単語のペアの分散表現からコサイン類似度を記録した。これらをそれぞれのプログラムに対して 100 回繰り返し、それぞれのコサイン類似度の平均や分散などを計算した。

第6章　結果と考察

6.1　結果

実験の結果、図１のような値が出た。

	-0.05069	0.066915	-0.10895639	-0.09453	-0.02476	0.105756	-0.01712	0.042801
	0.082762	0.223206	-0.10982685	0.22245	0.005829	0.062499	-0.06088	-0.0248
	love,sex		tiger,cat		book,paper		plane,car	
平均	0.028409	0.050288	-0.02178473	-0.01233	-0.01904	-0.02715	0.044337	0.055766
最大	0.220783	0.26038	0.228230411	0.27075	0.186014	0.26759	0.265319	0.315312
最小	-0.23328	-0.22451	-0.2791739	-0.23485	-0.26285	-0.22279	-0.14832	-0.19112
分散	0.010362	0.009486	0.009953301	0.010491	0.008023	0.006948	0.008913	0.010574

図 6.1: 実行結果の一部

重要な部分は分散の違いだけではあるのだが、コサイン類似度の方は修正前後で何が違うのかわからないような状態であることが見て取れる。

修正前後の分散の差については表１のようになる。

表 6.1: 分散の差

	修正前	修正後	差
平均	0.00783701	0.007280835	-0.000556175
差が最小	0.00859298	0.003668206	-0.004924774
差が最大	0.00569254	0.010882681	0.005190145

平均を見ると修正後のプログラムの方が分散が小さくなっており、プログラムの修正で分散が最も小さくなったのは”drug” と”abuse” の時の-0.00492 で、最も大きくなったのが”type” と”kind” の時の 0.00519 であった。

6.2 考察

単純に考えれば、全体の分散の平均をとると値が小さくなるという想定通りの結果が出たため、この方法はより精度が求められる場合有用であるといえるだろう。しかし、最も分散が小さくなった時より、分散が大きくなった時の方が値が大きいことや、分散の平均がわずか 7%ほどしか小さくなっていないことなどから偶然の可能性もあると考えられる。しかし、有意水準の値である p 値[3]を計算したところ、$p = 0.022$ となったため、偶然このようになった可能性はとても低いと言えるだろう。今回の手法の都合上 Negative Sampling を行う度に単語のペアが共起しているか否かの確認を毎回挟むことになるため速度が落ちてしまうことも考える必要がある。(とはいえ、Negative Sampling を使わない場合よりだいぶ早い。)

以上から、今回の修正で得る利点はとても小さく、修正する必要性はほとんどないと言っても過言ではない。修正したとしても得られる効果の小ささから、作業時間が長くなるという欠点の方を重く見て良いだろう。

[3]統計上、ある事象が起こる確率が偶然とは考えにくい（有意である）と判断する基準となる確率。今回は $p = 0.022$ のため、2.2%の確率で偶然分散が小さくなった可能性があると言える。
(つまり、97.8%の確率で修正後の方が分散が小さくなりやすい。)

第7章　結論

本論文では、Negative Sampling の理論と実態の乖離に着目し、プログラムを理論に沿うように修正することで Word2Vec の持つ分散表現の揺らぎをを小さくできるという考えから実験を行った。

結果、確かにコサイン類似度の分散は小さくなった。つまり分散表現の揺らぎが小さくなったが、良い分散表現を得られやすくなったとは言い難く、分散の変化や最大値及び最小値の変化を見ても十分な効果があるとは言い難い結果となった。Negative Sampling に異なる処理が挟まることで、プログラムの実行に必要な時間が伸びてしまうのも大きな問題点だろう。

以上から、よっぽど精度を求めたい場合を除き今回の修正に効果があるとは言い難く、Word2Vec の動作に必要な時間も伸びてしまうことからプログラムは現状のままの方が良いと思われる。しかし、精度の高い結果を求めやすい方法の一つとして頭に留めておいても良いだろう。

謝辞

本研究に際して，多くのご指導を頂きました指導教員の新納浩幸教授に心より感謝致します．また，日常の議論を通じて多くの知識や示唆を頂いた新納研究室の皆様に感謝致します．

参考文献

[1] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. “Efficient Estimation of Word Representations in Vector Space.” arXiv preprint arXiv:1301.3781 (2013).

[2] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. “Distributed Representations of Words and Phrases and their Compositionality.” NIPS, pp.3111–3119 (2013).

[3] Yoav Goldberg and Omer Levy. “word2vec Explained: Deriving Mikolov et al.’s Negative-Sampling Word- Embedding Method.” arXiv preprint arXiv:1402.3722 (2014).

[4] Tomas Mikolov, Quoc V. Le, Hya Stskever. “Exploiting Similarities among Languages for Machine Translation” arXiv preprint arXiv:1309.4168 (2013)

[5] Finkelstein, L., Placing search in context: The concept revisited. Proc. of WWW 2001, pp.406-414(2001).

[6] 新納浩幸, Chainer による実践深層学習, オーム社 (2016)

付録

付録1　Chainerによる実践深層学習をもとに作成したプログラム

```
#!/usr/bin/env python

import numpy as np
import chainer
from chainer import cuda, Function, gradient_check, Variable, \
                        optimizers, serializers, utils
from chainer import Link, Chain, ChainList
import chainer.functions as F
import chainer.links as L

from chainer.utils import walker_alias
import collections

# Set data

cowords = {}
index2word = {}
word2index = {}
counts = collections.Counter()
dataset = []
```

```
with open('ptb.train.txt') as f:
    for line in f:
        for word in line.split():
            if word not in word2index:
                ind = len(word2index)
                word2index[word] = ind
                index2word[ind] = word
            counts[word2index[word]] += 1
            dataset.append(word2index[word])

n_vocab = len(word2index)
datasize = len(dataset)

cs = [counts[w] for w in range(len(counts))]
power = np.float32(0.75)
p = np.array(cs, power.dtype)
sampler = walker_alias.WalkerAlias(p)

## co-occor

for p1 in range(datasize):
    id1 = dataset[p1]
    for j in range(10):
        p2 = p1 + j
        if (p2 < datasize):
            id2 = dataset[p2]
            if (id1 < id2):
                k = str(id1) + " " + str(id2)
            else:
                k = str(id2) + " " + str(id1)
            if k not in cowords:
                cowords[k] = 1
            else:
```

```
                cowords[k] += 1

nocon = 0
con = 0
for p1 in range(datasize):
    id1 = dataset[p1]
    for id2 in sampler.sample(10):
        if (id1 < id2):
            k = str(id1) + " " + str(id2)
        else:
            k = str(id2) + " " + str(id1)
        if k not in cowords:
            nocon += 1
        else:
            con +=1

# Define model

class MyW2V(chainer.Chain):
    def __init__(self, n_vocab, n_units):
        super(MyW2V, self).__init__(
            embed=L.EmbedID(n_vocab, n_units),
        )
    def __call__(self, xb, yb, tb):
        xc = Variable(np.array(xb, dtype=np.int32))
        yc = Variable(np.array(yb, dtype=np.int32))
        tc = Variable(np.array(tb, dtype=np.int32))
        return F.sigmoid_cross_entropy(self.fwd(xc,yc), tc)
    def fwd(self, x, y):
        x1 = self.embed(x)
        x2 = self.embed(y)
        return F.sum(x1 * x2, axis=1)
```

```
# my functions

ws = 3          ### window size
ngs = 5         ### negative sample size

def mkbatset(dataset, ids):
    xb, yb, tb = [], [], []
    for pos in ids:
        xid = dataset[pos]
        for i in range(1,ws):
            p = pos - i
            if p >= 0:
                xb.append(xid)
                yid = dataset[p]
                yb.append(yid)
                tb.append(1)
                for nid in sampler.sample(ngs):
                    xb.append(yid)
                    yb.append(nid)
                    tb.append(0)
            p = pos + i
            if p < datasize:
                xb.append(xid)
                yid = dataset[p]
                yb.append(yid)
                tb.append(1)
                for nid in sampler.sample(ngs):
                    xb.append(yid)
                    yb.append(nid)
                    tb.append(0)
    return [xb, yb, tb]

# Initialize model
```

```
demb = 100
model = MyW2V(n_vocab, demb)
optimizer = optimizers.Adam()
optimizer.setup(model)

# Learn

bs = 100
for epoch in range(10):
#    print('epoch: {0}'.format(epoch))
    indexes = np.random.permutation(datasize)
    for pos in range(0, datasize, bs):
#        print epoch, pos
        ids = indexes[pos:(pos+bs) if (pos+bs) < datasize else datasize]
        xb, yb, tb = mkbatset(dataset, ids)
        model.zerograds()
        loss = model(xb, yb, tb)
        loss.backward()
        optimizer.update()

# Save model

with open('w2v.model', 'w') as f:
    f.write('%d %d\n' % (len(index2word), 100))
    w = model.embed.W.data
    for i in range(w.shape[0]):
        v = ' '.join(['%f' % v for v in w[i]])
        f.write('%s %s\n' % (index2word[i], v))
```

付録 2　修正後プログラムの改変部分

修正前のプログラムの 113〜116 行目及び 123〜126 行目を以下のように改変した。

“neglist.txt” という名で、ある単語のペアが共起したか否かを記録した三角行列を作成してある。共起していない単語のペアであった場合は linp が 0 になりプログラムが進行するが、共起している単語のペアであった場合 linp が 1 から変わらず再度抽選が行われる。

```
linp = 1
while linp == 1:
    for nid in sampler.sample(1):
        if yid < nid:
            target_line = linecache.getline('neglist.txt', nid)
            target_words = target_line.split()
            linp = target_words[yid-1]
        elif yid > nid:
            target_line = linecache.getline('neglist.txt',yid)
            target_words = target_line.split()
            linp = target_words[nid-1]
```