修 士 学 位 論 文

複数の分散表現を利用したCNNによる文書分類

平成30年度

茨城大学大学院理工学研究科

情報工学専攻

ZHANG　TIANWEI

平成 30 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

複数の分散表現を利用した CNN による文書分類

著者 : ZHANG TAINWEI (17NM712T)
指導教員 : 新納　浩幸教授

論文要旨

本論文では日本語のテキストを複数の分散表現したベクトルに対して CNN（Convolutional neural network）を用いた手法を試みる。

近年、文書分類の領域には多数研究されています、伝統的手法によって、制度を向上するのは難関であり、それで色んな新しい手法が提案されました。分散表現はテキストの中から文書を単語列を作り出し、単語に対して画像のようなベクトル化に表現します。

本論文はこの単語を画像化（ベクトル化）するために、テキストを文字列ではなく単語列としてとらえ、分散表現（nwjc2vec）を利用する。これにより上記で述へた問題か生しない。また従来学習手法として NN か゛ 用いられてきたか、近年、ディープラーニングにより画像識別の精度が飛躍的に高まっているのをわかっている。日本語文書を分散表現したものを CNN 利用して、その効果や問題を確認する。実験の結果、CNN による文書分類は同等以上の精度を示した。また訓練データの大きさかは本質的に重要であり、今後は分散表現を利用してこの問題に対処する。

本論文では，はじめ CNN による画像識別の手順に関することを紹介しました．第３章でデータの分散表現として、nwjc2vec のモデルを用いた。第 4 章で系列ラベリング問題についてまとめました。第 4 章では CNN による文書分類の手法を提案し、第 5 章では実験で利用したデータとその結果示した、第 6 章では実験結果について考察した，最後に第 7 章ではこの研究の結論を述べた。

Master's Thesis in Scholastic 2018, Major in Computer and Information Sciences, Graduate School of Science and Engineering, Ibaraki University

Document classification by Convolutional neural network using plural distributed expressions

Author:Zhang Tianwei(17NM712T)
Adviser: Prof. Hiroyuki Shinnou

ABSTRACT

In this paper, we attempt a method using CNN (Convolutional neural network) for vectors in which Japanese texts are dispersedly expressed.

In recent years, a lot has been studied in the field of document classification, it is difficult to improve the system by traditional methods, so various new methods have been proposed. Distributed representation creates word strings from texts and expresses them in a vectorized way like images for words.

In this paper, in order to image (vectorize) this word, we regard the text as a word string rather than a character string, and use a distributed expression (nwjc2vec). This will not result in the problem described above. In addition, it is known that NN has been used as a conventional learning method, or in recent years, the accuracy of image identification has dramatically increased by deep learning. Use the distributed representation of Japanese documents as CNN to confirm the effect and problem. As a result of the experiment, the document classification by CNN showed equal accuracy or better. In addition, it is essentially important whether the size of the training data is important, and in the future we will use this distributed expression to deal with this problem.

In this paper, I first introduced about the procedure of image identification by CNN. In chapter 3, the model of nwjc2vec was used as a distributed representation of data. Chapter 4 summarizes the series labeling problem. In chapter 4 we propose a CNN document classification method, chapter 5 shows the data used in the experiment and its results, chapter 6 examined the experimental results, finally in chapter 7 the conclusion of this study Mentioned.

目次

1.まえかき

近年、文書分類の領域には多数研究されています、伝統的手法によって、制度を向上するのは難関であり、それで色んな新しい手法が提案されました。

本論文では日本語のテキストを複数の分散表現したベクトルに対して CNN（Convolutional neural network）を用いた手法を試みる。

この単語を画像化（ベクトル化）するために、テキストをの文字列ではなく単語列としてとらえ、分散表現（NWJC 2 vec）を利用する。これにより上記で述べた問題かが生しじない。また従来学習手法として NN が用いられてきたかが、近年、ディープラーニングにより画像識別の精度が飛躍的に高まっているのをわかっている。本論文では日本語文書を分散表現（ベクトル化）し、CNN を利用して、その効果や問題を確認する。

本論文では，はじめ CNN による画像識別の手順に関することを紹介しました．第 3 章でデータの分散表現として、nwjc2vec のモデルを用いた。第 4 章で系列ラベリング問題についてまとめました．第 4 章では CNN による文書分類の手法を提案し、第 5 章では実験で利用したデータとその結果示した，第 6 章では実験結果について考察した，最後に第 7 章ではこの研究の結論を述べた。

実験の結果、CNN による文書分類は同等以上の精度を示した。また訓練データの大きさかは本質的に重要であり、今後は分散表現を利用してこの問題に対処する.。

2.システム設計

2.1 構造の設計

普通 CNN による画像識別の手順につきましては紹介します。

CNN の手順①　画像から特徴を抽出

フィルタを使って、入力層データの中で位置を変えながらスキャンした部分のデータと、フィルター自体の持つデータとの差異を畳み込みの結果として畳み込み層に書き込んだものを特徴量といい、入力層の全データをスキャンしてできた畳み込みの結果の値の集まりを特徴マップといいます。

複数のフィルタを用意することで、入力層のデータの特徴を捉えやすくしています。

CNN の手順②　画像を畳み込み

入力層のデータをフィルタのデータとピクセル毎に比較することで、畳み込み層にその類似度（特徴量）を書き込みます。

CNN の手順③　画像をプーリング

畳み込み層の情報はプーリング層で集約します。例えば最大プーリングという集約方法をとる場合、各ユニット（領域）のピクセルを比較し、その中の最大値をそのユニットの特徴量とします。

出力には、プーリング層のユニットすべてと全結合し、計算結果を利用して、フィルタ、重み、バイアスを更新していきます。

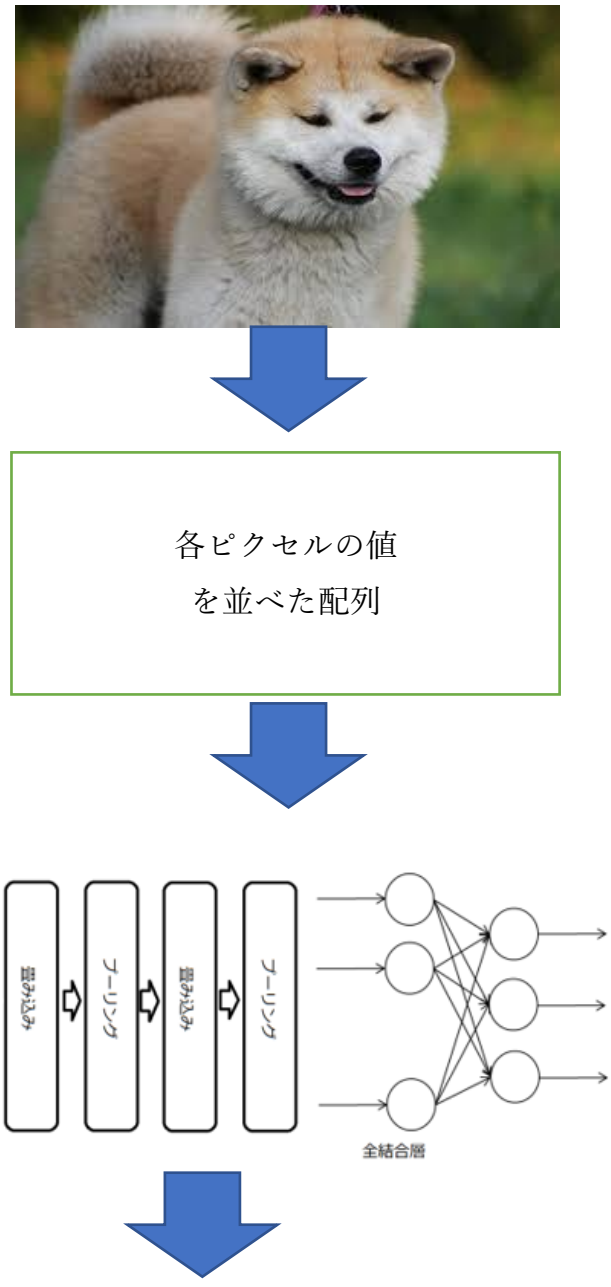
畳み込みニューラルネットワークを利用した画像処理の流れを簡単に説明すると、

①フィルターを使用し、入力画像の全体に対して畳み込み層でフィルター処理を行う。

②処理した画像をプーリング層に流し込む。

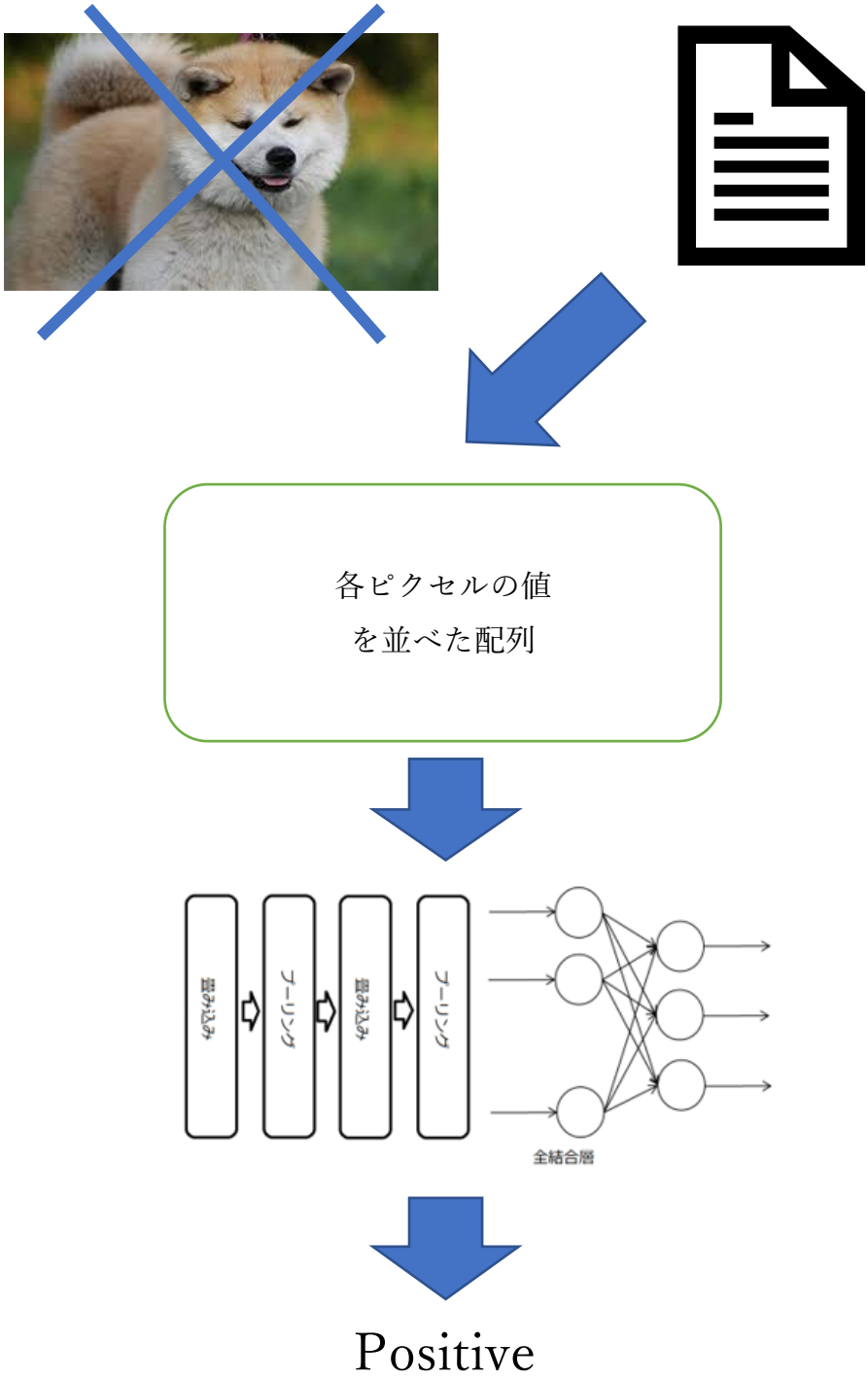
③プーリング層で画像の解像度を下げる処理を行う。

以下の図で普通の CNN の手順を示しています。



本研究では CNN による感情分析なので、画像の代わりに文書を入れ込み、そのため、文書中の単語を何らか数値ベクトルとしてに表現する必要があり。

今回の実験では nwjc2vec モデルを用いて，各領域について出現単語の単語ベクトルを作成する．単語ベクトルとは，ひとつの単語をひとつのベクトルで表した知識表現のことである．



2.2 動作環境

本システムのクライアント側は Python Framework―Chainer を用いて、使用言語は Python で開発した。文書を画像化をするために、分散表現は nwjc2vec を利用した。本システムの動作環境を表 2.1 に、開発環境を表 2.2 に示す。

表 2.1：システム動作環境

OS	Windows 10 Home
CPU	Intel(R) Core(TM) I7-7700HQ CPU @ 2.80GHz
RAM	16GB
GPU	GTX 1060 6GB

表 2.2：システム開発環境

スクリプト	Python3.7.1
フレームワーク	Chainer 5.1.0

3.畳み込みニューラルネットワーク（CNN）

3.1 基本の仕組み

CNN の特徴は,畳込み層およびプーリング層と呼ぶ特殊な層を交互に接続した構造をもっことにある。この構造を除けば, CNN は普通の順伝播型の NN に過ぎず次のような基本構造をもつ、ある層におけるノ番目のユニットに,その直前の層の出力y_i(j = 1,・・・, n)　の重み付き和にバイアスが加算された

$$x_j = b_j + \sum_{i=1}^{m} y_i w_{ij}$$

が入力される.このもを,活性化関数と呼ばれる非線形関数に入力したときの応答

$$y_j = f\ (x_j)$$

がこのユニットの出力となり,次の層の各ユニットに入力される。

通常, CNN の出力層付近には隣接層のユニット間をすべて結合した(全結合型)の層を 1 層以上配置する。クラス分類が目的の場合、一番最後の出力には、目的のクラス数と同数のユニット個を配置し、そしてこれらのユニットへの入力も x_j(j = 1,・・・, n)を,上の活性化関数ではなく多項ロジスティック関数

$$p_j = \frac{e^{xj}}{\sum_{k=1}^{n} e^{xk}}$$

を用い($\sum_{j=1}^{n} p_j = 1$になる) ,出力とする.認識実行時にはp_jが最大値をとるユニットのインデックス j= argmax$x_j p_j$而を推定クラスとする。

3.2 畳み込み層

畳込み層の基本構造とは、入力N枚(以下，この枚数をチャネルと呼ぶ)の W x W サイズの画像の形をとる。以下ではこれを$\mathrm{W}\times\mathrm{W}\times\mathrm{N}$と記す.最初の入力層では,チャネル数は入力画像がグレースケールなら $\mathrm{N}=1$、カラーなら RGB 計 3 枚 $\mathrm{N}=3$である。それ以降の中間層では,直前の畳込み層の出力チャネル数(すなわち後述のフィルタ数)と一致する.以下では,この $\mathrm{W}\times\mathrm{W}\times\mathrm{N}$の入力を$y_{ijk}((i,j,k)[1,\mathrm{W}]\times[1,\mathrm{W}]\times[1,N])$と書く。

畳込み層では,この入力にフィルタを畳み込む計算を行う.これは,一般的な画像処理でのフィルタの畳込み，すなわち小サイズの画像を入力画像に二次元的に畳み込んで,画像をばかしたり,あるいは工ッジを強調するものと基本的に同じである。

具体的には,入力の$\mathrm{W}\times\mathrm{W}$サイズの各チャネルごとに$\mathrm{w}\times\mathrm{w}$のサイズの二次元フィルタを畳み込み,その結果をNチャネルにわたって加算するのが一般的である．フィルタを$h_{ijk}((i,j,k)[1,\mathrm{w}]\times[1,\mathrm{w}]\times[1,N])$と書くと、この計算の結果は 1 チャネル分の出力となる。

これにより,一っのフィルタh_{ijk}につき,入力y_{ijk}と縦横サイズが同じ$\mathrm{W}\times\mathrm{W}$の1チャネル分の出力を得る．同様のフィルタを$N$個用意し,それぞれ独立に上の計算を行えば,$N$チャネル分の$\mathrm{W}\times W$の出力，すなわち$\mathrm{W}\times\mathrm{W}\times N$のサイズの$y_{ijk}((i,j,k)[1,\mathrm{W}]\times[1,\mathrm{W}]\times[1,N])$を得る。$N$個あるフィルタのうちの一つについての計算をする。

3.3 プーリング層

プーリング層は畳込み層と対で使われ,基本的には畳込み層の出力がプーリング層への入力となる。それにしたがって,その入力はやはりW × W × Nの形をとる。プーリング層の目的は画像のどの位置でフィルタの応答が強かったかという情報を一部捨てることで、画像内に現れる特徴の微小な位置変化に対する応答の不変性を実現することにある.なお,畳込み層とは違って、プーリング層での計算内容はネットワークの設計時に決まり、学習によって変化しない。

プーリング層のユニット(i, j)は,畳込み層の局所受容野の構造と同じく、入力層(畳込み層の出力層)の一部の小領域p_{ij}について,この小領域内部のユニット(p,q)p_{ij}の出力y_{pq}を集約し一つの出力とする。この小領域p_{ij}は,プーリング層のユニット(i, j)と同じ相対位置(あるいはその「真下」) にある畳込み層のWx Wユニットである(サイズは畳込み層のフィルタサイズとは通常無関係です)。また入力が複数チャネルある場合、チャネルごとに以上の処理を独立して行うのが一般的である(つまり、畳込み層のチャネル数(=フィルタ数)とプーリング層のそれが一致する)。

通常,このプーリングの処理は,画像の方向に間引いて行う。つまりストライドsを1より大きな値とする。例えばs = 2とすると出力は入力の縦横半分のサイズとなる。sと小領域のサイスW × Wいかんによって, 隣接する出力ユニットの受容野が互いに重なりあうこともある。プーリング層の出力は入力よりサイズが縮小されるのが普通だが、その縮小比を決めるのはストライドであることに注意する。

4.文書の画像化(分散表現)

4.1 基本のアイデア

①単語の意味を特徴的なベクトルで表そう
②意味は前後に出てくる単語で特徴化
③そのベクトルが低次元、密ベクトルになっていることも特徴です

最近の自然言語処理では、文字レベルの言語処理が行われることがあります。 これら文字レベルの言語処理は、ユーザ生成コンテンツに有効であると言われています。 その理由として、ユーザ生成コンテンツのような崩れたテキストでは、形態素解析の性能が大幅に低下し、単語レベルの処理が上手くいかなくなることが挙げられています。

文字レベルの言語処理を行うなら、単語レベルの場合と同じく文字分散表現を事前学習したくなります。 文字分散表現を事前学習しておくことで、①良い文字ベクトルの初期値を得られる、②学習データに現れない文字のベクトルを教師なしで得られる、といったメリットガわかっています。

本研究では、日本語の文字に対して文字分散表現を学習してみます。 本文は次の 2 つの内容で構成されています。

- 学習データの準備
- 文字分散表現の学習

後程の第 5 章の実験のところに詳しく説明させていただきます。

4.2 単語ベクトルの生成

併合コーパスに対して nwjc2vec モデルを用いて，各領域について出現単語の単語ベクトルを作成する．単語ベクトルとは，ひとつの単語をひとつのベクトルで表した知識表現のことである．nwjc2wec モデルは入力層，中間層，出力層からなり，対象単語の周辺単語を入力とし，対象単語を予測するニューラルネットワークである．入力層と出力層は辞書中の単語と一対一に対応したノードから成るベクトルを表している．中間層は指定した数のノードを隠れ変数として与えることができる．入力層と中間層の間には，それらをつなぐ重みがあり，重み行列として表現する．この重み行列を更新することによって，周辺単語から対象単語を予測するように学習を行う．nwjc2vec は文書集合全体で学習した重み行列を用いて各出現単語の単語ベクトルを出力する。

5.実験

5.1 実験設定

本研究における実験用の文書データには，国立国語研究所が開発した現代日本語書き言葉均衡コーパスを利用する。このコーパスは日本語の様々なジャンルの文書を収録した，書き言葉の全体像を把握するために構築されたコーパスである。

文書集合から word2vec に入力可能な単語列への変換を行う．文書集合に対して mecab を用いて形態素解析を行って単語の基本形に分割する。

コーパスのデータ数

Train	2139 個
test	535 個

nwjc2vec の実行のパラメータ

次元数	size	200
文脈長	window	233
負サンプリング数	negative	25
階層化 softmax	hs	0
最低頻度値	sample	Le-4
反複回数	iter	10

5.2 学習データの準備

まずは、文字分散表現の学習に用いるデータを用意します。 以前、以下の記事で単語分散表現を学習するためのコーパスを作成したことがありました。 今回は、そのコーパスを文字分散表現の学習用に変換して使うことにしましょう。

はじめに、コーパスのダウンロードと解凍を行います。 コーパスをダウンロードし、ファイルを開くすると、`train.dat や test.dat` という名前のコーパスが現れます。

次に、単語単位で区切られている `train.dat や test.dat` を文字単位に区切りましょう。文字単位で空白に区切っておくことで、コーパスの読み込みが簡単にできるようになります。 以下のコーパスを書いて、文字単位の区切りに変換しましょう。

上記のスクリプトを実行することで、文字単位に区切られたコーパス `ichigyo.txt` が生成されます。 `Ichigyo.txt` の中身をみると、以下のように空白を区切り文字として、文字単位に区切られていることがわかります。

```
難 を 言え ば 湯船 が 狭い こと と 喫煙 室 で あっ た ため 、白い
壁 が ヤニ で 茶色く 変色 し て い た こと 。
……
……
```

続きまして、形態素解析を行った単語列は特徴化を表現されますが、その中から単語―品詞という形で取り出しすると、以下の形式になっています。

```
難-名詞　を-助詞

言え-動詞　ば-助詞

湯船-名詞　が-助詞

狭い-形容詞　こと-名詞
```

と-助詞　、-補助記号

喫煙-名詞　室-接尾辞

で-助動詞　あっ-動詞

た-助動詞　ため-名詞

、-補助記号　白い-形容詞

壁-名詞　が-助詞

ヤニ-名詞　で-助詞

茶色く-形容詞　変色-名詞

し-動詞　て-助詞

い-動詞　た-助動詞

こと-名詞　。-補助記号

……

……

取り出した単語―品詞の形の単語列にをデータ数 2139 に合わせて 1 ずつファイルを保存します。

学習用のデータが用意できたので、文字分散表現を学習しましょう。

5.3 文字分散表現の学習

作成したコーパスと `nwj c 2vec` を使って文字分散表現を学習させてみます。同じディレクトリで、以下のコードを実行して学習させましょう。 文字分散表現の学習は数分で終わるはずです。

今回はハイパーパラメータとして、文字分散表現の次元数を `200`、window サイズを `5` にしています。 次元数が比較的低次元なのは、文字は単語と比べて語彙数が少ないため、次元数を大きくする必要がないからです。 また、window サイズを広めに取ったのは、トピックの近い文字が空間上で近くなるように学習させる意図があります。

```
難-名詞　0.93366 -6.509989 -5.906303 0.214714 …….382745 3.625399

を-助詞　2.578197 -6.678836 5.235908 6.892576 …….547395 -0.345394

言え-動詞　-0.707921 2.65009 -0.679349 0.115215……2.928199 0.806332

ば-助詞　 0.916211 1.657758 0.285238 3.529637…….466444 0.556584

湯船-名詞　-1.885426 -1.137049 6.066681 0.54647……8.441523 -3.495977

が-助詞　-0.477284 1.859203 0.269751 1.271799……0.531279 0.019086

狭い-形容詞 -5.283065 -0.44486 1.752915 -5.702563……722811 -1.796834

こと-名詞　-8.688087 -0.716002 -0.736279 -1.289141……220378 4.516357

と-助詞　1.814621 -3.797831 2.075994 -2.326933……5.710362 0.104591

、-補助記号　0.316277 -2.050302 -1.515985 -1.895723……269326 1.306579

喫煙-名詞　-5.113933 3.999117 -5.042511 -3.102235……326726 -6.602059

室-接尾辞　2.456984 -0.292772 4.536817 -4.508796……11.3925 -6.368083

で-助動詞　1.062979 -0.721476 -0.186454 0.863662……0.895759 0.451862

あっ-動詞　-0.772696 7.14905 3.220794 -6.877355……2.842664 -3.93723
```

た-助動詞　0.029988 0.118986 -1.792013 1.637438……8.291867 0.332228

ため-名詞　0.86678 0.169713 -0.194803 -0.86143……0.12663 0.358269

、-補助記号 -0.316277 -2.050302 -1.515985 -1.895723……2.269326 1.306579

白い-形容詞 -3.326147 0.320356 0.517596 -0.430277……-1.516196 -3.94105

壁-名詞　1.753178 -5.28514 -0.530058 0.338219……-2.048427 -3.386847

が-助詞　-0.477284 1.859203 0.269751 1.271799……0.531279 0.019086

ヤニ-名詞 -0.273265 0.417926 -0.025901 -0.11147……0.933303 -0.48075

で-助詞 -3.280993 2.437669 -1.257072 5.492289……0.183656 1.305152

茶色く-形容詞 1.024377 0.890364 1.584021 2.121734……7.563321 -4.853026

変色-名詞　0.070704 -4.308506 0.39464 1.54591……6.830524 -9.622178

し-動詞　-0.002801 1.220201 -4.237149 1.617499……-0.936314 -1.463345

て-助詞　0.677148 4.701918 -0.588208 -0.514052……-10.474952 4.102994

い-動詞　-3.2197 4.763291 0.758203 -3.50346……-0.939065 0.136638

た-助動詞　0.029988 0.118986 -1.792013 1.637438……8.291867 0.332228

こと-名詞　-8.688087 -0.716002 -0.736279 -1.289141……1.220378 4.516357

。-補助記号　0.972159 2.599617 -2.659817 -2.104068……4.521084 -1.731269

……

……

学習が終わったら、実際に試してみると。

2139 個の文章各自の長さが違うので、それを一番長いの文に合わせて、足らない部分が 0 を入れ込みます。上記の表に示している 200（次元数）×233（一番長い文章に単語数）のベクトルが生成されました。

5.4 実験結果

今回の実験ではかなり計算量が多くで、16GB のメムリは足りませんでした。念のためサーバーの環境に移動してから、実験を行いました。

train データ（2139 個）を用いて、モデルを生成されます。

tset データを 535 個として入力され、精度は以下に示しています。

325 / 535　＝　0.6074766355140186

何か改良するといくつ精度を上がる方法があるはずです。

まず、思われているのはモデルのエポック数を増やす、それに対して実験を行いました。

エポック数	正解率
4	0.607
5	0.616
6	0.613
10	0.644
20	0.635

結果により、モデルのエポック数を変えと精度が良くなりましたが、ただ単純にエポック数を上がって精度がよくなるわけではありません。

6 考察

本論文では，テキストを文字列ではなく単語列として捉え，単語のまま CNN には処理できない問題としては単語列を画像化する。単語ベクトル化したものを長さを同じく揃える処理が必要があるとなっています。分散表現した単語ベクトルを CNN を用いて，現在多数使用されておる手法より良い精度が期待できると考え，結果に NN と同等以上の性能を示した. 実験の評価が不十分である。今後の実験で充実させます。 今回に実験でモデルのエポック数を増やすでけ、精度がいくつ良くなったのが、それに加えて，別のコーパスを用いたり、モデルの構造を変えたりフィルターの枚数やサイズを変えるとか）を改良すると精度がよくなれると思われます。結果を比較して、効果を確認に行くと考えています。

7 結論

本論文では分散表現を用いた文書分類のシステムを作成し，正解率を向上の改善を試した．テキストを文字列ではなく単語としてとらえ，ベクトル化の問題が生じない。実験の結果、CNN は普通 NN より精度が上がったのを示した。また訓練データの大きさが本質的に重要であり、今後は CNN を利用してこの問題に対処する。

参考文献

[1] Chainer
https://chainer.org/
(2019 年 1 月 22 日最終アクセス)
[2]『国語研日本語ウェブコーパス』 に基づく単語の分散表現データ
http://www.anlp.jp/proceedings/annual_meeting/2017/pdf_dir/E1-5.pdf
(2018 年 11 月 21 日最終アクセス)
[3] Conventional natural network
https://ja.wikipedia.org/wiki/畳み込みニューラルネットワーク
(2019 年 1 月 29 日最終アクセス)
[4] NumPy
http://www.numpy.org/
(2019 年 2 月 1 日最終アクセス)
[5] MeCab 形態素解析システム
https://dev.classmethod.jp/server-side/mecab-using-python3-ja/
(2018 年 10 月 24 日最終アクセス)

謝辞

本研究ならびに論文作成に当たり、指導教員として最後までご指導いただきました茨城大学工学部情報工学科 新納　浩幸　教授に心より感謝申し上げます。

研究室の皆様には、研究の場/日常生活を問わず、あらゆる面においてお世話になりました。研究室の西　友佑様はチューターとして、右も左もわからなかった私を導いてくださったことに大変感謝いたします。本研究に関してご協力をいただいた茨城大学の各位に深く感謝致します。

最後に、これまでお世話になったすべての方々に、改めて感謝の意を表します。

ソースリスト

モデルの生成：

```
import numpy as np
import chainer
from chainer import cuda, Function, report, training, utils, Variable, ¥
                    datasets, iterators, optimizers, serializers, Link, Chain, ChainList
import chainer.functions as F
import chainer.links as L

# Data setting
xtrain = np.load('train x.npy').reshape((2139,1,233,200)).astype(np.float32)
ytrain = np.load('train y.npy').astype(np.int32)
# xtest = np.load('test_x.npy').astype(np.float32)
# yans = np.load('test_y.npy').astype(np.int32)

# Define model

class MyModel(Chain):
    def __init__(self):
        super(MyModel, self).__init__()
        with self.init_scope():
            self.cn1=L.Convolution2D(1,20,5)
            self.cn2=L.Convolution2D(20,50,5)
            self.fc1=L.Linear(50*56*47,500)
            self.fc2=L.Linear(500,2)

    def __call__(self, x,t):
        return F.softmax_cross_entropy(self._fwd(x),t)

    def fwd(self, x):
        return F.softmax(self._fwd(x))
```

```
    def _fwd(self, x):
        h = F.max_pooling_2d(F.relu(self.cn1(x)),2)
        h = F.max_pooling_2d(F.relu(self.cn2(h)),2)
        h = F.dropout(F.relu(self.fc1(h)))
        return self.fc2(h)

# Initialize model

model = MyModel()
optimizer = optimizers.Adam()
optimizer.setup(model)

# Learn

n = len(ytrain)
bs = 1000

for j in range(10):
    idx = np.random.permutation(n)
    for i in range(0, n, bs):
        x = Variable(xtrain[idx[i:(i+bs) if (i+bs) < n else n]])
        y = Variable(ytrain[idx[i:(i+bs) if (i+bs) < n else n]])
        model.cleargrads()
        loss = model(x,y)
        loss.backward()
        optimizer.update()
    outfile = "cnn-" + str(j) + ".model"
    serializers.save_npz(outfile, model)
    print(outfile," saved")
```

精度を試みる：

```
import sys
import numpy as np
import chainer
from chainer import cuda, Function, report, training, utils, Variable, ¥
                    datasets, iterators, optimizers, serializers, Link, Chain, ChainList
import chainer.functions as F
import chainer.links as L

argvs = sys.argv
argc = len(argvs)

# Data setting

# xtrain = np.load('train x.npy').reshape((2139,1,233,200)).astype(np.float32)
# ytrain = np.load('train y.npy').astype(np.int32)
xtest = np.load('test x.npy')# .reshape((535,1,233,200)).astype(np.float32)
yans = np.load('test y.npy')# .astype(np.int32)

# Define model

class MyModel(Chain):
    def __init__(self):
        super(MyModel, self).__init__()
        with self.init_scope():
            self.cn1=L.Convolution2D(1,20,5)
            self.cn2=L.Convolution2D(20,50,5)
            self.fc1=L.Linear(50*56*47,500)
            self.fc2=L.Linear(500,2)

    def __call__(self, x,t):
        return F.softmax_cross_entropy(self.fwd(x),t)
```

```
    def fwd(self, x):
        h = F.max_pooling_2d(F.relu(self.cn1(x)),2)
        h = F.max_pooling_2d(F.relu(self.cn2(h)),2)
        h = F.dropout(F.relu(self.fc1(h)))
        return self.fc2(h)

# Initialize model

model = MyModel()

# Load model

serializers.load_npz('cnn-5.model', model)

# Test

xt = Variable(xtest)
yy = model.fwd(xt)

ans = yy.data
nrow, ncol = ans.shape
ok = 0
for i in range(nrow):
    cls = np.argmax(ans[i,:])
#     print(ans[i,:], cls)
    if cls == yans[i]:
        ok += 1

print(ok, "/", nrow, " = ", (ok * 1.0)/nrow)
```