CNN の畳み込み層の相違による文章のランク分類精度の評価

岡山理科大学
総合情報学部情報科学科
椎名研究室
I13I006 井上　佳祐

目次

1. はじめに

人工知能を使ったロボットや自動返答ボットなどが数多く登場した．SoftBank の Pepper[1]や Microsoft の女子高生 AI りんな[2]などが，その典型的な例である．このように人工知能という技術が広く使われるようになった背景としては，近年のインターネットの普及で情報収集が容易になったことや，コンピュータの，収集した情報を処理する性能が向上したことなどが挙げられる．それと同時に，ディープラーニング[3]を含め機械学習を簡単に行うための便利なツールなどが無償で公開され誰でも容易に利用できるようになった．その例としては，Google の TensorFlow[4]やそのライブラリの Keras[5]などである．ディープラーニングの中でも畳み込みニューラルネットワーク（Convolutional Neural Network，以下 CNN）は，画像データの分類に広く用いられている．ディープラーニングにおける文章分類など，自然言語処理の分野では，一般的に再帰型ニューラルネットワーク(Recurrent Neural Network，RNN)が用いられるが，近年では自然言語処理に CNN を用いる研究[6]も行われている．

本研究では，CNN によるコメントのランク分類を行う上で，CNN のネットワーク構成の最適化について考察することを目的とする．

2. 実験データ

本研究では，国立情報学研究所[7]から提供された，2011 年 7 月 28 日までの楽天市場[8]の商品レビューデータを利用した．17 個の項目が登録されており，その中の評価ポイントとレビュー内容の 2 項目のデータを学習および評価に使用した．評価ポイントは 0〜5 の 6 段階評価である．本研究では 2 分割されているデータのうち, 1,297,679 件を学習データ, 1,216,726 件を評価データとした．登録されている項目を表 1 に示す．

表 1：データの登録項目一覧

	項目	説明
1	投稿者	「user1」のようにマスクしたユーザ名
2	年齢	10 歳台など
3	性別	0：男 1：女 2：不明
4	商品コード	店舗コード:商品 ID
5	商品名	商品の名前
6	店舗名	販売店舗の名前
7	商品 URL	http://item.rakuten.co.jp/[商品コード]
8	商品ジャンル ID	商品のジャンル ID
9	商品価格	商品購入時の価格
10	購入フラグ	0：購入あり 1：購入なし
11	内容	[実用品・普段使い]などの文字列
12	目的	[自分用]などの文字列
13	頻度	[はじめて]などの文字列
14	評価ポイント	0〜5 の 6 段階評価
15	レビュータイトル	レビューのタイトル
16	レビュー内容	レビューの内容
17	レビュー登録日時	レビュー登録年月日 (yyyy/mm/dd HH:MM:SS)

3. 分類処理の CNN による実装

3.1. 分類処理の概要

購入者が投稿したレビュー内容のコメントをその商品に対する購入者の評価ポイントで分類する．分類処理の概要を図 1 に示す．

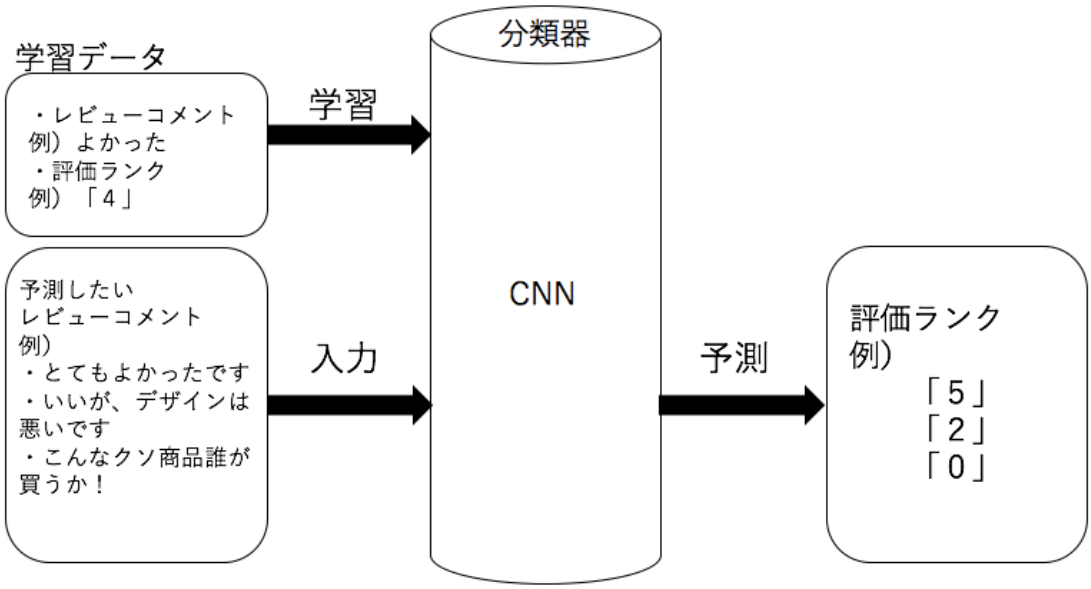

図 1：分類処理の概要

3.2. 具体的な実装方法

自然言語処理における CNN で問題なのは, ニューラルネットワークへの文章の入力方法である．画像はピクセルの集合であり（縦のピクセル数）×（横のピクセル数）であると考えられるので, 文章もこれに類似した行列に変換して入力する．したがって, 埋め込み層で単語の埋め込み表現を学習し，そして畳み込みを行う．この分類処理の CNN による実装を以下に示す．

(1) コメント分の前処理（ト―クン化）
(2) CNN の定義
　入力層，埋め込み層，畳み込み層，プーリング層，全結合層，出力層の定義
(3) 目的関数の設定
(4) データの学習

全体的な CNN の処理の流れを図 2 に示す.

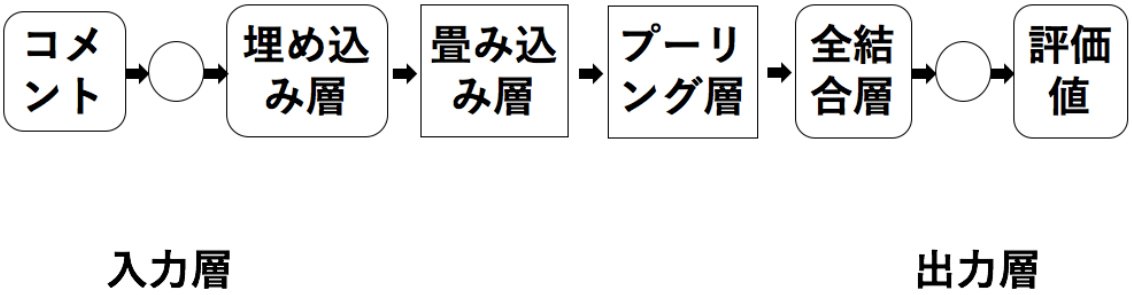

図 2 : 定義する CNN

3.2.1. コメント文の前処理

畳み込み層で単語の埋め込み表現を学習するために，形態素解析器を利用し文を形態素に分割する. 次に，単語をトークンとし，単語を辞書としてあらかじめ決定しておいた単語番号に基づいて変換する. 文を形態素に分割する例を図 3 に，単語番号への変換を図 4 に示す.

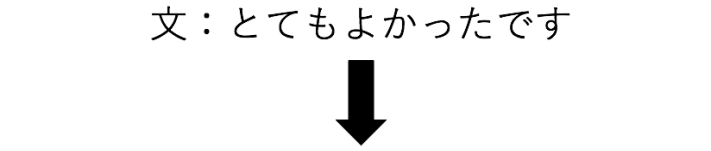

図 3 : 形態素に分割

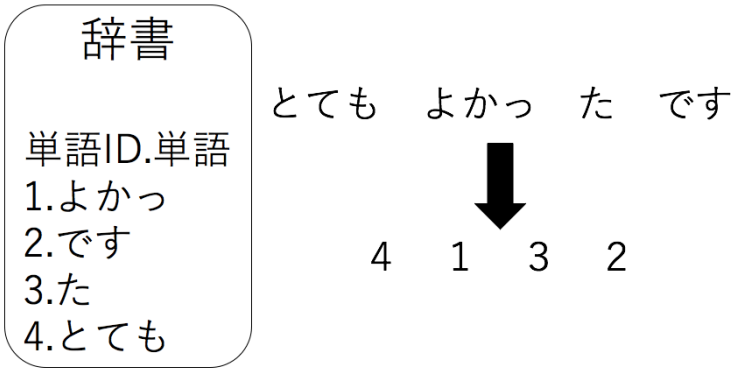

図 4 : 単語番号への変換

3.2.2. 埋め込み層

埋め込み層では，トークン化された入力文の単語の高次元ベクトルである埋め込み表現を学習する（図 5）．これはフレームワークが提供している埋め込み表現学習の Word Embedding 関数を使うことで，任意の大きさのベクトル表現が学習できる．Word Embedding 関数は，単語から文脈を推定する Skip-gram を利用したフィードフォワードモデルである．

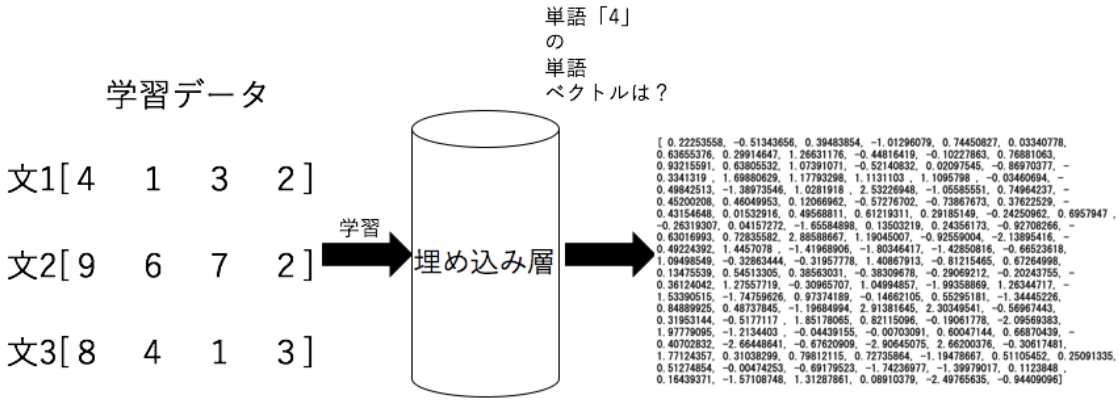

図 5：学習した単語の埋め込み表現（単語ベクトル）

3.2.2.1. Skip-gram

Skip-gram のニューラルネットワークは，時刻tにおける入力単語$w(t)$に対して前後の単語を予測するモデルである．このモデルのネットワーク構成を図 6 に示す．

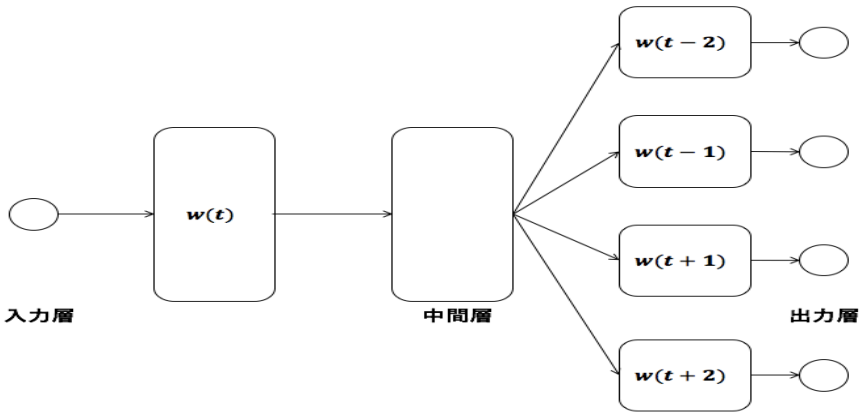

図 6 : Skip-gram の構成

このモデルは，単語$w(t)$が入力されたとき，前後の単語の条件付き生起確率$p(w_{t+j}|w_t)$の対数尤度最大化モデルと考える事ができ，次式で表される．

$$\frac{1}{T}\sum_{t\in T}\sum_{-c\leqq j\leqq c\,,j\neq 0}\log p(w_{t+j}|w_t)$$

出力の確率pはソフトマックス関数を用いて次式で表される．

$$p(w_o|w_i)=\frac{e^{v_{w_o}^T v_{w_i}}}{\sum_{w\in W} e^{v_w^T v_{w_i}}}$$

このモデルで学習される中間層の重みが単語の埋め込み表現であり単語ベクトルである．単語の前後を予測し関連付けることで，文中での単語の前後関係（文脈）から単語を意味空間上に投影し，類似度や距離が計算可能な単語ベクトルが学習できる．

3.2.2.2. 単語から単語ベクトルへの変換例

単語から単語ベクトルへ変換するとき，変換対象となる単語が学習する単語に含まれている必要がある．未知語に対しては，単語ベクトルは取得出来ない．単語「とても」「よかっ」「た」「です」が学習に含まれているモデルを利用し，例文「とてもよかったです」の単語ベクトルへの変換例を表 2 に示す．

表 2：単語から単語ベクトルへの変換例

t	単語	単語ベクトル
1	とても	[0.74023992,-2.51178455,0.52828914,-3.87759161,-0.7623468,-0.12430308,-1.30062199,0.35881296,2.33546543,-0.44892228,0.03466128,-2.18267274,-0.6401751,-1.1779443,-1.38583469,2.44884634,-0.12147604,-1.02365065,-1.73230898,-0.62814975,0.58600289,2.56747985,0.45956364,2.88063622,-1.21871197,1.05206513,-1.87785792,2.73015094,-1.49648595,0.9818731,0.10075828,-1.56679475,0.48930106,-2.0745337,0.26465741,-0.79339516,-1.86364758,-0.46492884,-2.50687146,-0.40318996,4.23239994,2.24182439,-1.23619366,0.05503944,-1.67498279,-2.70853829,-0.97387218,1.66325855,0.25824124,0.0332576,4.26040745,1.76179314,0.76561219,-0.04736669,-2.25630403,-3.10131621,0.97167253,1.20494413,-0.35476032,-0.07962036,0.78034478,2.99121308,0.09805541,0.55545098,0.70966339,-0.69973546,0.60939348,-0.18820749,-1.30080295,-1.61083436,1.69017088,-0.7856288,0.93478554,-0.73058558,-0.92143661,0.98265964,1.85705984,0.26552632,0.77052295,-1.0382123,-1.43124521,-2.30691338,0.99380386,0.23051222,0.19535917,-0.63923931,1.74829304,2.89443922,-2.61825848,0.68912613,0.4660196,1.34688628,1.53243208,-1.40657151,-1.79708624,-0.16325141,-0.35532215,1.09166741,-0.09121776,2.32841277,-0.50790823,-1.19950593,-1.42303896,0.18700232,3.87823892,1.06758094,1.48003674,0.67594904,-1.71335852,0.97312313,0.13084777,-2.78179955,-0.08380438,-0.3727468,2.51441383,-0.23988865,-0.6154632,0.12517564,1.13341475,-0.09872803,0.15996732,-0.59440923,2.41723919,-0.97201657,1.21544933,1.23479843,-0.04192927,-1.27740812]
2	よかっ	[-0.24045505,-2.58204865,-0.30464473,-3.60575008,-1.35347724,1.18492985,-0.13709317,0.73134089,-0.72242093,-1.10831439,-0.19799447,-1.90351164,-1.83036876,-0.12406625,-2.73072052,0.40414235,2.24749041,1.83439171,-0.72089994,0.52379066,-0.05816922,0.39443794,-1.31098723,-0.46211743,-3.77450013,2.10486293,-3.53243065,-0.26737437,-2.53475642,3.62842679,0.88555741,0.29424116,2.11353683,0.14752108,0.34147254,0.62645096,-0.82789421,-1.61289334,-0.69029582,5.0437355,0.96862149,3.37072563,-0.05954974,-0.04465029,0.18329166,-1.06584764,-0.7875421,0.28435591,-0.75786924,-1.17141032,0.37021941,0.48282909,-2.79085255,0.72000533,-1.95668316,-0.16892885,1.08886898,-0.47505945,0.29664424,-0.14874411,1.23907161,0.50385189,-0.56764728,2.14612627,-0.9655776,0.01241853,1.14950109,-2.86055326,0.7247597,-1.47937965,-0.90046829,-0.75494534,-0.88512683,-2.72958159,0.53628719,-0.16067772,2.23313737,2.4472785,-2.29082918,-0.52025521,-0.651748,1.15375769,1.26089978,1.26877284,1.75008035,0.53923148,1.07843006,2.08223462,0.91476488,4.75389624,-1.0965693,0.2580145,1.7866993,-0.58893293,1.33949864,-0.47677928,0.8713997,-3.22563243,0.4508107,-0.33051574,-2.73127174,-0.93049353,-2.58384156,1.16490638,4.09575558,-0.67349184,-2.33997321,3.23774242,-1.86051655,3.22237325,0.44404241,-3.72186208,-0.17063603,-0.39209324,2.10702848,0.91172409,0.26886269,0.05401034,0.15095685,-1.39151311,1.74299228,0.15223226,0.64816606,-1.26028883,-2.50266051,0.53852451,-2.40141916,1.99199378]
3	た	[4.92648602e-01,-3.44954824e+00,1.75932562e+00,-9.94901371e+00,-3.25897312e+00,-1.06773806e+00,-2.08065867e+00,-4.40738297e+00,3.21644235e+00,-1.12433088e+00,7.06372917e-01,-7.84276783e-01,4.98833847e+00,-3.43766356e+00,-1.88748670e+00,-2.43164968e+00,-3.77355665e-02,-4.52091455e+00,1.97734082e+00,-8.39039087e-01,-3.33686471e+00,7.62408376e-01,-1.88337173e-02,-2.63784230e-01,3.13092637e+00,4.07807082e-01,1.30742431e+00,1.62933493e+00,3.06787562e+00,2.99571037e+00,1.04494357e+00,2.65404910e-01,4.01050627e-01,1.05323935e+00,2.50037599e+00,1.05254757e+00,4.16552591e+00,-3.52236462e+00,3.24860740e+00,-1.06952667e+00,1.59271741e+00,1.49074078e-01,1.68612027e+00,4.46590614e+00,-4.89251089e+00,-3.86099666e-01,-3.33819687e-02,9.16493893e-01,-3.71577048e+00,3.09008151e-01,1.91114068e+00,1.09202242e+00,1.87600529e+00,1.83794236e+00,-1.76024985e+00,-3.65964800e-01,-8.87224637e-03,-4.69665384e+00,5.11932671e-01,2.84534955e+00,-2.09720182e+00,-3.35681415e+00,-4.49589586e+00,1.30368829e+00,-1.28890824e+00,-9.79331374e-01,-3.83152175e+00,5.03954828e-01,-5.06401682e+00,-3.23063993e+00,3.26303220e+00,3.58314800e+00,-1.18603432e+00,-4.77634048e+00,-2.60696530e+00,2.66176295e+00,-2.27782631e+00,2.97537446e+00,-3.54460382e+00,-2.77794862e+00,-5.92049696e-02,-3.64458770e-01,6.13554764e+00,-9.31412876e-01,2.22369361e+00,-1.96775770e+00,4.38837767e+00,5.94229460e+00,1.82035112e+00,-1.96260262e+00,-1.31979501e+00,3.15607929e+00,8.83626461e-01,-3.03904384e-01,-6.04191780e+00,-1.00449467e+00,4.40628260e-01,3.22497487e+00,1.70996654e+00,4.96030951e+00,1.44370067e+00,-3.69741678e-01,-7.68287718e-01,6.80177259e+00,5.10622501e+00,2.04904699e+00,7.22328901e-01,-8.30633044e-01,4.01792049e+00,2.17188883e+00,-1.13010359e+00,-2.44239712e+00,1.84571958e+00,2.80299711e+00,4.94452620e+00,-1.05403662e+00,-8.58421743e-01,-3.30013609e+00,-4.97999954e+00,-4.68475533e+00,-3.04823852e+00,-2.14978456e+00,2.12303972e+00,-3.63352680e+00,1.34133375e+00,4.81157827e+00,-4.71917772e+00,7.57227838e-01]
4	です	[-1.22904348,-3.15517473,0.62350208,-4.34242964,0.12550627,3.09658813,-2.58200264,0.1271107,0.92197549,-0.85984671,-2.74237919,-1.36493576,-2.6991179,-0.33201241,-1.90670204,4.45194435,2.5322969,1.76093268,-1.14779794,1.46115971,1.82584608,1.26389885,-1.15937197,3.10487533,-1.86538756,1.91611052,0.80607134,1.06936252,-0.94381696,0.94173717,-0.62227029,1.32698476,1.92570519,2.68873382,0.33926046,-0.3278847,-5.26786995,-2.71357703,-1.79410398,1.43648899,1.61420894,3.16579103,0.25718328,-0.17878909,-1.3362782,-1.9128722,-1.76344073,0.9241761,-0.10687412,-1.63001978,1.42778444,1.65897286,-1.10422945,3.70355177,-0.8163268,-1.65933013,3.2560904,-2.5542593,-3.17362976,-2.01132321,2.2587986,2.09472799,1.53540242,0.744349,-1.947505,-1.80929446,1.53011572,-1.09125412,-1.89576364,-2.68801188,-0.28127751,-0.10942721,-0.24425578,-3.09260988,0.33587676,0.88369423,1.15633523,4.23930836,-0.64375252,-1.94829822,-0.96596593,-0.87990361,-0.72057098,3.46869373,-0.80532104,-1.19538367,-0.10057927,2.3617847,0.93279225,1.59492481,-0.93803108,-0.87561005,1.22911298,-0.47780457,-3.15592194,0.01553896,0.28115341,-0.66475624,0.56237483,0.56730461,-1.63155782,0.89841837,-1.86444616,0.7924521,4.33098125,0.21147837,-1.57101631,4.28514338,-1.33715081,0.23645453,0.09555952,-3.62355185,-0.15483497,1.15735507,1.07042992,3.04047942,1.63568783,0.85249078,-1.29040515,-0.74887043,-0.64946038,-0.45408264,2.38152885,-0.01483009,0.25731644,0.46044177,-3.20428705,2.16508675]

3.2.3. 畳み込み層

畳み込み層へは，埋め込み層から出力される単語のベクトル表現を縦に並べ，（文の単語数）×（単語ベクトルの大きさ）の行列を作り，1 つの文ごとにひとまとめにして入力する．

畳み込み層の処理では，行列に対して指定した高さと幅のフィルタごとに畳み込みを行う．フィルタの幅の設定は，入力行列の各行が 1 つの単語トークンとなっているため，それに対応させ，埋め込み表現の次元数と同じ大きさにする．またフィルタの高さを任意に設定する．フィルタの高さおよび幅は図 7 で示す．

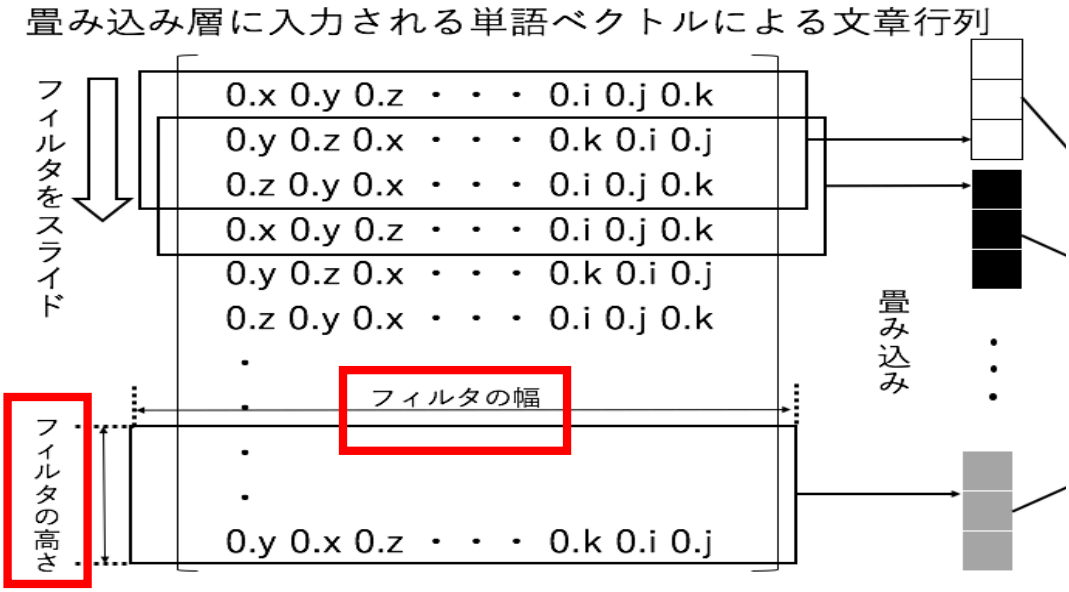

図 7：畳み込みフィルタの幅と高さ

畳み込み層は，畳み込みで得られるデータを特徴マップとしベクトルを出力する．

3.2.4. プーリング層

プーリング層ではマックスプーリングのルールに従い，畳み込み層の各フィルタから出力された特徴マップの行列から最大値を取得し，次元削減後，それらを結合して 1 つのベクトルにする．これが文章の特徴ベクトルとなる．特徴ベクトルの大きさは学習する畳み込みフィルタの数に依存し，フィルタの数と同

じ次元のベクトルとなる．この特徴ベクトルは，文章の長さが変わっても常に同じ次元数のベクトルを作る．畳み込み層とプーリング層の一連の処理の流れを図 8 に示す．

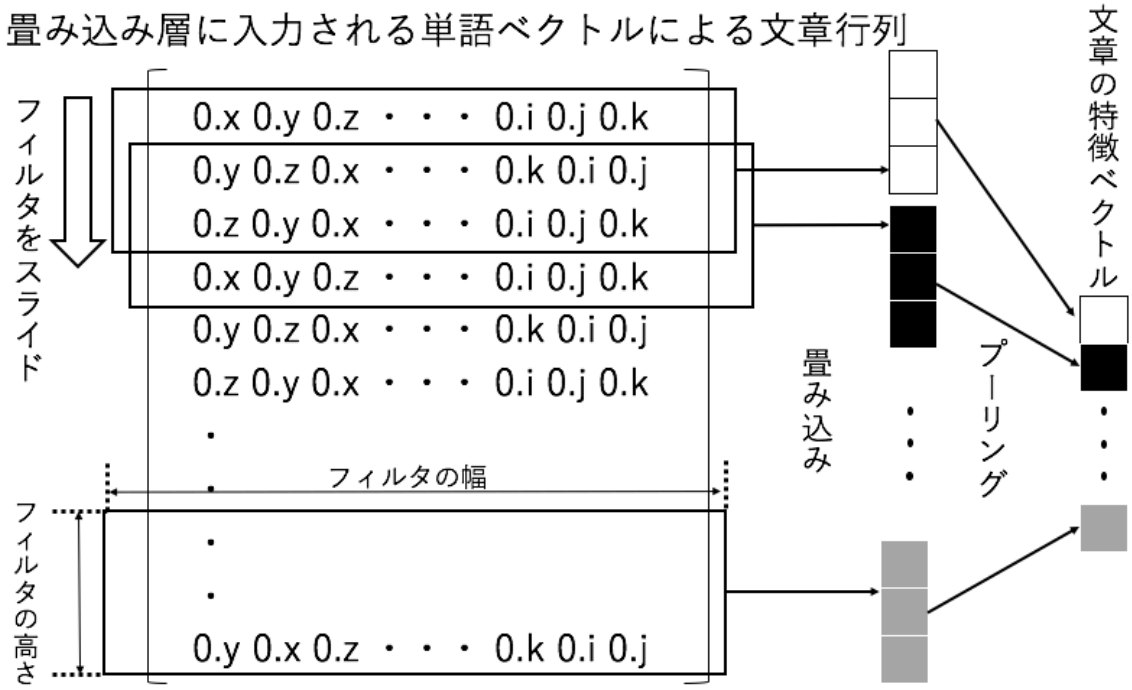

図 8：畳み込みとプーリングイメージ

3.2.5. 出力層

出力層では，全結合相からの出力に，ソフトマックス関数を適用させて分類する各クラスに属する確率を出力する．入力行列が畳み込み層・プーリング層を通って全結合層から出力されるときの値を，文章の行列ベクトル$X_n = (x_{1,n}, x_{2,n}, \cdots x_{M,n})$と入力文章が評価値$k$である確率を学習する重み$W_k = (w_{0,k}, w_{1,k}, w_{2,k}, \cdots w_{M,k})$を使って表すと次の式で表される．

$$f_k(X_n, W_k) = w_{0,k} + w_{1,k}x_{1,n} + w_{2,k}x_{2,n} + \cdots + w_{M,k}x_{M,n}$$

また，評価値kである確率P_kを次に表すソフトマックス関数を用いて示すと以下の式になる．

$$\mathrm{P}_k(X_n, W_k) = \frac{e^{f_k(X_n, W_k)}}{\sum_{k'=1}^{K} e^{f_{k'}(X_n, W_{k'})}}$$

3.2.6. 目的関数の設定

ニューラルネットワークの学習は，学習時に出力層から出力される確率値と教師ラベルから，正解ラベルに属する確率値を大きくするように重みを学習する．その指標となるのが目的関数である．出力層でソフトマックス関数を利用しているため，目的関数に交差エントロピーを用いた．入力文章X_nが教師ラベル$t_{k',n}$に属する確率値$P_{k'}(X_n)$を最大化するには，次式に表す目的関数 E を最小化する．

$$\mathrm{E(W)} = -\sum_{n=1}^{N}\sum_{k'=1}^{K} t_{k',n} \log P_{k'}(X_n, W_{k'}), \qquad \mathrm{W};\ W_1, \cdots, W_K$$

目的関数 min(E)

3.2.7. データの学習

定義したニューラルネットワークの学習を複数回繰り返し行う．データの学習は，目的関数の微分による勾配を利用したアルゴリズムに従って目的関数の値を最小化するように各層の重みを学習する．

4. 本研究での提案

ニューラルネットワークの実装とアルゴリズムに対して，畳み込み層でのフィルタの高さと畳み込み層とプーリング層の並列数（ノード数）を変更した．畳み込み層のフィルタの高さの設定イメージを図 9 に，畳み込み層とプーリング層のノード数のイメージを図 10 に示す．

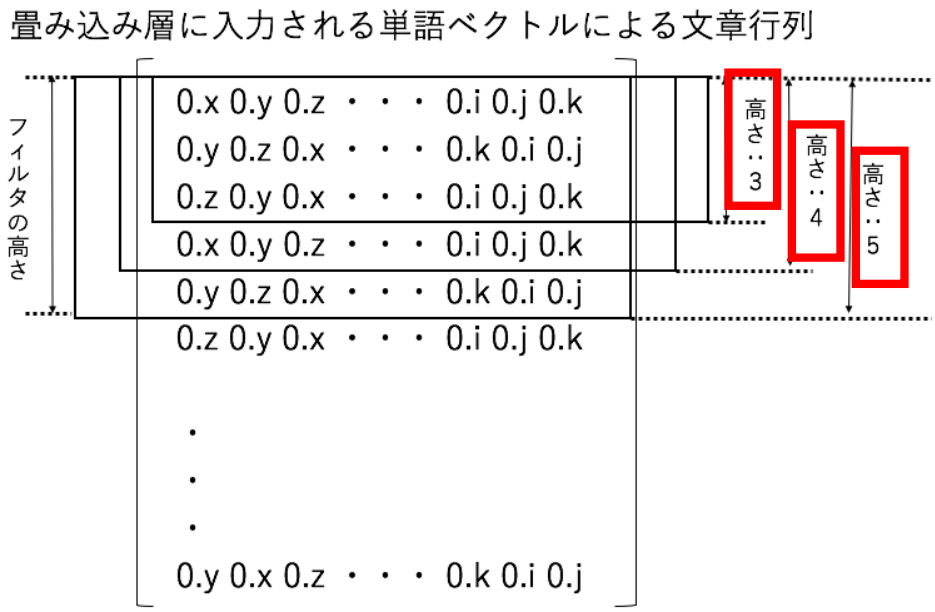

図 9：畳み込みフィルタの高さイメージ

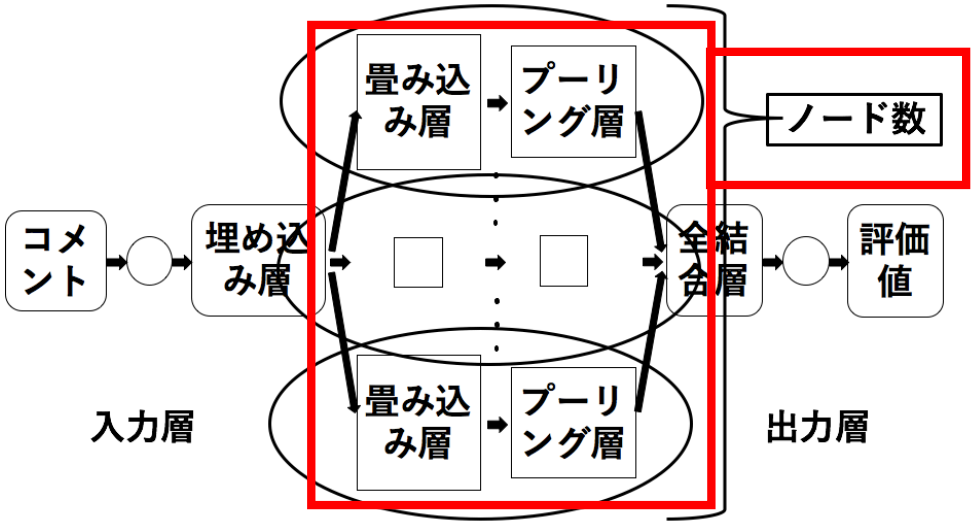

図 10：ノード数イメージ

5. 実験結果

5.1. 使用した CNN モデル

フィルタの高さおよびノード数を変更した 7 つのモデルで実験を行った．使用したモデルを図 11～13 に示す．

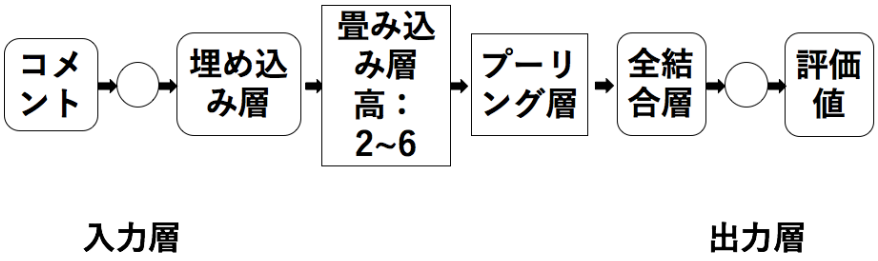

図 11：畳み込み層 1 ノードモデル

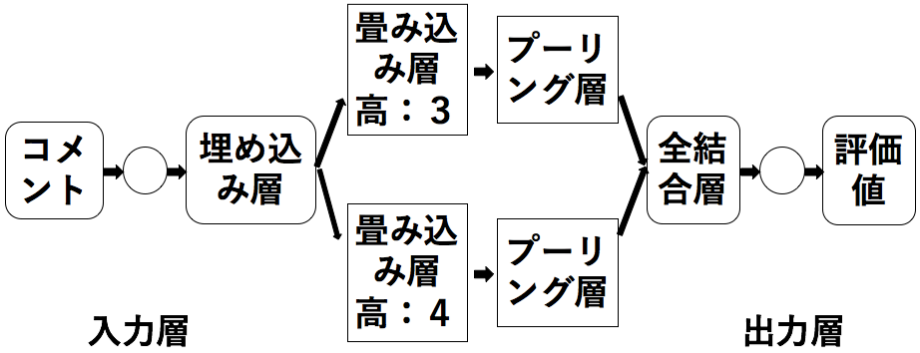

図 12：畳み込み層 2 ノードモデル

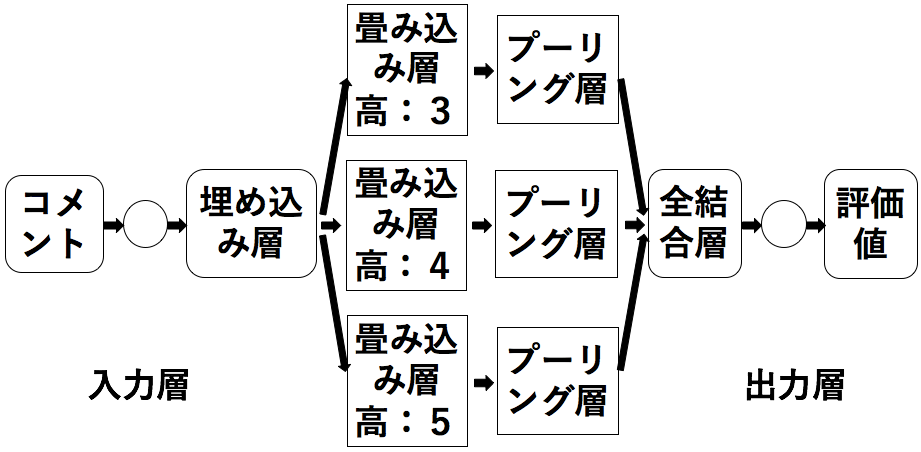

図 13：畳み込み層 3 ノードモデル

5.2. 精度および考察

モデルを 5 回ずつ学習させ，学習回数ごとで精度を評価した．その精度を表 3，損失を表 4 に示す．参考に SVM での分類精度も記載する．

表 3：精度一覧(H：フィルタの高さ，N：ノード数)

回数	精度 H:2 N:1	精度 H:3 N:1	精度 H:4 N:1	精度 H:5 N:1	精度 H:6 N:1	精度 H:3,4 N:2	精度 H:3,4,5 N:3	SVM
1	0.6852	0.6869	0.6867	0.6881	0.6891	0.6884	0.6908	
2	**0.6876**	**0.6903**	**0.6912**	0.6906	**0.6927**	**0.6921**	**0.6935**	
3	0.6866	0.6898	0.6911	**0.6925**	0.6926	0.6908	0.6897	0.44
4	0.6812	0.6840	0.6864	0.6880	0.6875	0.6837	0.6779	
5	0.6663	0.6703	0.6751	0.6764	0.6736	0.6729	0.6620	

※**赤字**は各モデルの最高精度

表 4：　損失一覧(H：フィルタの高さ，N：ノード数)

回数	損失 H:2 N:1	損失 H:3 N:1	損失 H:4 N:1	損失 H:5 N:1	損失 H:6 N:1	損失 H:3,4 N:2	損失 H:3,4,5 N:3
1	0.7374	0.7335	0.7303	0.7304	0.7259	0.7272	0.7174
2	0.7286	0.7223	0.7186	0.7195	0.7151	0.7182	0.7123
3	0.7308	0.7240	0.7200	0.7169	0.7164	0.7222	0.7300
4	0.7490	0.7442	0.7381	0.7352	0.7369	0.7462	0.7910
5	0.8002	0.7887	0.7760	0.7721	0.7805	0.7917	0.8835

5.2.1. 畳み込み層 1 ノードの CNN

畳み込み層とプーリング層が 1 ノードで高さが 2〜6 とそれぞれ違う畳み込みフィルタを用いた CNN（図 11）である.

5 回の学習における各回の損失と精度を比較してみると，2,3 回学習時での精度が最も高く, それを境に損失も大きくなり精度も低下している. 最も高い精度と 5 回学習時の精度を比較すると, およそ 1.5%〜2.0%も精度が低下している. また，5 回学習時の精度が 5 回の中でも一番低く，過学習していると考えられる．このモデルでの学習回数は 2,3 回が適切でありそれ以上学習する必要はない．また，フィルタの高さを 2〜6 と変えて学習したが，高さを高くし，より多くの単語を一度に畳み込むほど精度が高くなっている事が分かる. しかし, 高さ 5 と高さ 6 の精度はそこまで差がみられないため，これ以上高くしても効果は見られないと考える．しかし，高さ 6 のほうが高さ 5 よりも 1 回少ない学習回数で同程度の精度に到達している.

5.2.2. 畳み込み層 2 ノードの CNN

畳み込み層とプーリング層を並列に 2 ノードにし，高さ 3 と高さ 4 の畳み込みフィルタを用いた CNN（図 12）である.

5 回の学習における各回の損失と精度を比較してみると，1 ノードの時と同様に 2 回学習時での精度が最も高く，それを境に損失も大きくなり精度も低下している. 1 ノードの高さ 3 と高さ 4 での精度と比較すると, 0.1%程度であるが, それぞれ 1 ノードずつで分類するよりも高い精度が得られているので，ノードを複数個にすることで，精度が向上する可能性が考えられる.

5.2.3. 畳み込み層 3 ノードの CNN

畳み込み層とプーリング層を並列に 3 ノードにし，高さ 3 と高さ 4 と高さ 5 の畳み込みフィルタを用いた CNN（図 13）である.

5 回の学習における各回の損失と精度を比較してみると，1 ノード・2 ノードの時と同様に 2 回学習時での精度が最も高く，それを境に損失も大きくなり精度も低下している．このモデルでは，1 回目の学習で既に精度が 69%を超えてきており，実験に用いた CNN のなかで一番速い．また精度もどの CNN よりも高い値が得られており，ノード数の複数化による精度向上に多少期待できる.

6. 実験評価

6.1. 7 つのモデルの総合評価

7 つのモデルにおいて学習回数ごとの精度変化を図 14 のグラフ(H：フィルタの高さ，N：ノード数)に示す.

※赤点（・）は各モデルの最高精度

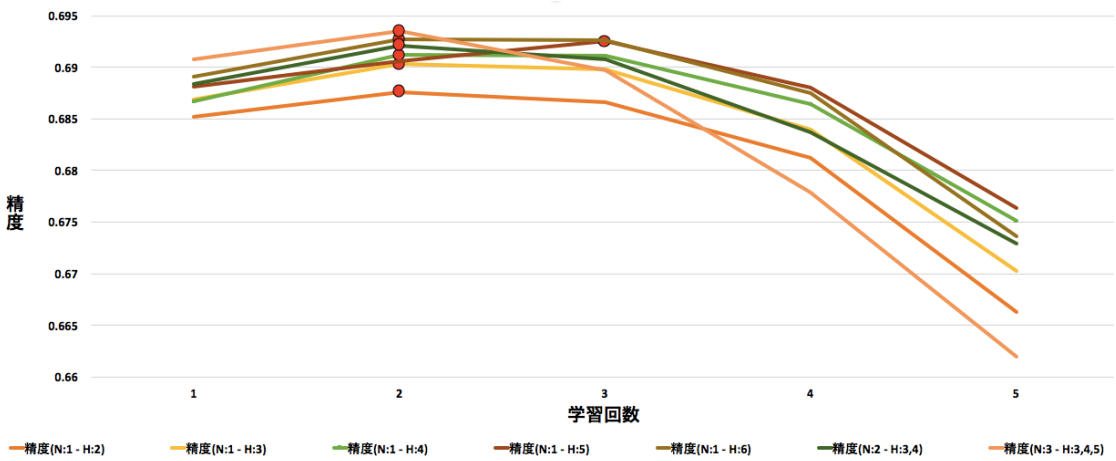

図 14：学習回数ごとの精度変化グラフ

どの種類のモデルでも類似した値の変化になっているが，中でも 3 ノードのモデル(N:3－H:3,4,5)は学習を重ねるに連れて，精度の低下度合いが大きく劣化しやすい事がわかる.

6.2. CNN 構成の最適化についての考察

CNN によるコメント分類での精度向上の最適化は次の手順で行う.

(1) 1 ノードの高さが小さいフィルタで学習する.
(2) 精度が低下するまで学習し，精度が悪くなったら 1 つ前のモデルを使い局所最適化を行う.
(3) フィルタの高さを大きくし，同様に学習する.
(4) フィルタの高さを大きくしても精度に変化が見られなければ，ノード数を増やす.
(5) 1 ノードの時の精度を参考にフィルタの高さを設定し同様に学習する.

この手順を 2 ノード, 3 ノードと繰り返すことで多少の精度向上は見込めると考えられる.

7. 結論

実験結果とその評価から，今回の実験データでより高い精度を得るためには，畳み込み層の高さを変え，ノードを複数並列にした CNN を用いるのがよい．また，学習回数は 2,3 回にするのがよい．しかし，畳み込みを行う処理が多ければ多いほど，より時間がかかるので精度と学習時間との兼ね合いが大切である

8. まとめ

本研究では，CNN での自然言語処理として，商品レビューのコメントデータの評価分類を行う CNN の実装方法を示した. このようなニューラルネットワークはパラメータの数が多く学習には膨大な時間が必要である．本来ならばフィルタ数や次元数などをもっとチューニングすべきであるが，時間の都合上その点が疎かになってしまっている．学習時間を短くするには学習データをもっと小さいデータにし，学習回数を少なくすれば良いと考えられる．今後は，少ない学習データに対しても同程度の精度が得られるかどうか，パラメータやノード数の違いによる精度への影響を評価する必要がある.

謝辞

本研究を進めるにあたり，楽天市場の商品レビューデータを提供していただいた国立情報学研究所，楽天株式会社の皆様に感謝します.

参考文献

[1] Life with Pepper | ソフトバンク，http://www.softbank.jp/robot/special/pepper/，（2017/01/01 参照）

[2] りんな，http://rinna.jp/，（2017/01/01 参照）

[3] NTT コムウェア | ディープラーニング，https://www.nttcom.co.jp/research/keyword/dl/，（2017/01/01 参照）

[4] TensorFlow，https://www.tensorflow.org/，（2017/01/01 参照）

[5] Keras Documentation，https://keras.io/，（2017/01/01 参照）

[6] Yoon Kim，Convolutional Neural Networks for Sentence Classification，EMNLP(2014)，1746-1751

[7] 国立情報学研究所，http://www.nii.ac.jp，（2017/01/01 参照）

[8] 楽天市場，http://www.rakuten.co.jp，（2017/01/01 参照）

[9] 新納浩幸，Chainer による実践深層学習，オーム社（2016）

[10] 浅川伸一，Python で体験する深層学習，コロナ社（2016）

[11] 斎藤康毅，ゼロから作る DeepLearning，オライリー・ジャパン（2016）

[12] 中井悦司，TensorFlow で学ぶディープラーニング入門，マイナビ（2016）

[13] 清水亮，はじめての深層学習プログラミング，技術評論社（2016）