単語の分散表現と word2vec

Nature Insight
ネイチャーインサイト 株式会社

自然言語処理 応用編の目的

Nature Insight

自然言語処理・応用編の目的

従来の自然言語処理は、言語特有の性質、言語学一般やそれぞれの国の言葉に特化した学問の成果をモデル作成の軸としていました。

さらに、自然言語処理では基礎編で行ったように、Bag-of-Wordsや共起行列のままだと、スパース(0の成分が多い)な行列であるため、無駄な情報が多く、処理がうまく回らず、解釈も難しいです。

そこで、近年ではDeep Learningの発展などにより、特定の言語、目的に依存せず、スパースなデータをうまく解釈し、画期的な精度向上が行われてきたモデルがあります。

今回はその中のうち、以下の3つの手法を紹介しようと思います。

- ニューラルネットワークベースの単語の分散表現である**word2vec**
- トピックを用いた文書分類モデルの**トピックモデル**
- Deep Learningベースで、事前学習済みモデルである**BERT**

自然言語処理・応用編の目的は、以上の3つのモデルの特徴・性質を理解し、実際に使えるようになってもらうことです。

「単語の分散表現とword2vec」の目的

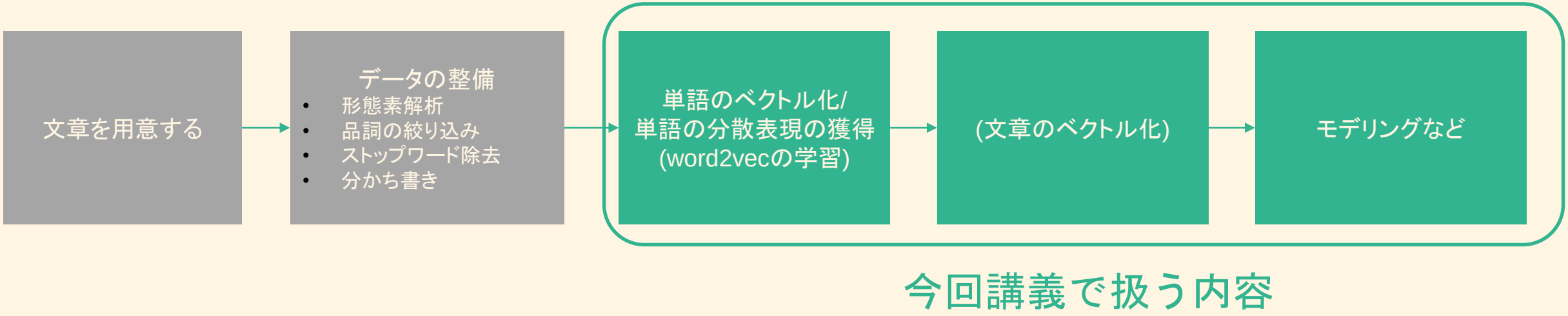
Nature Insight

「単語の分散表現とword2vec」の目的

今回の講義では、単語のベクトル化手法である「word2vec」と、それを用いた応用例(モデリング)を紹介します。

データの整備は取り扱いませんが、この部分は既知として進めますので、

もし不安な方は自然言語処理の基礎講義の受講で復習してからこの講義を受けてください。



目次

1. 単語の分散表現と概要

単語の分散表現とは

Nature Insight

単語の分散表現

自然言語処理では、機械学習で扱うために自然言語をベクトル化することが一つの大きな目標となってきます。

この講義では、**単語のベクトル化**に注目していきます。

自然言語は単語に意味があり、例えば「美味しい」、「美味い」など、似た意味をもつ単語があります。 しかし、Bag-of-WordsやTF-IDFなどの単純に単語をカウントする手法は、これらの単語を別単語として認識してしまします。したがって、単語の分散表現では、**「似た意味をもつ」**という関係性を踏まえたベクトル化を目指します。

このようなベクトル化手法を「**単語の分散表現(Word Embedding)**」と呼びます。

単語の分散表現の例

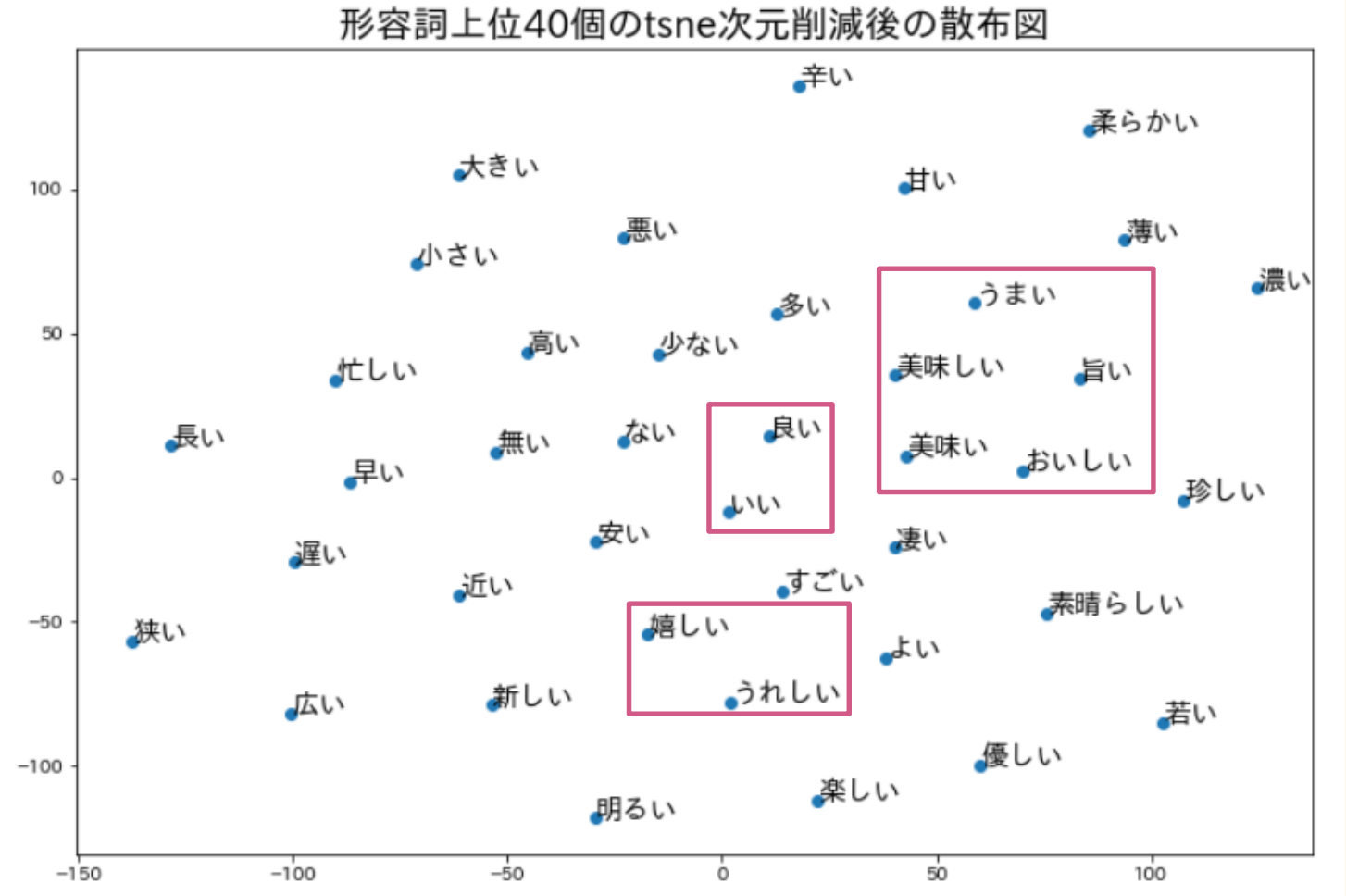
	特徴量1	特徴量2	特徴量3	特徴量4	特徴量5
美味しい	0	1	0	0	1
美味い	0	1	0	0	0

単語を数値化して、「似ている」関係を表現することを目指す。

単語の分散表現とは

単語の分散表現ができると嬉しいこと1

単語の分散表現を獲得すると、単語同士の近さを表すことができます。



「美味しい、美味い」

「良い、いい」

「嬉しい、うれしい」など、

似た意味の単語のベクトルが近くなっています

単語の分散表現とは

単語の分散表現ができると嬉しいこと2

単語の分散表現を獲得すると、単語の足し算・引き算ができるようになります。

例：「ラーメン」－「中華」＋「和風」に近い単語

単語	類似度
ベース	0.687342
醤油	0.649936
味噌	0.630444
オーソドックス	0.600499
透明	0.592406
白味噌	0.588429
透き通る	0.586029

ラーメンの和風要素として、
「醤油」・「味噌」が
上位に来ている。

2. 単語の分散表現の基礎知識

単語の分散表現の基礎知識

分布仮説

単語の分散表現の獲得において、基礎となる重要な仮説があります。

それは、「**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、「分布仮説」と呼ばれます。

考え方として、**その単語に注目するのではなく、文脈によって意味が形成される**という発想です。

例えば、以下の文章の「美味しい」と「美味い」は同じような文脈で出てきます。

この 店 に 美味しい ラーメン が ある。
この 店 に 美味い　ラーメン が ある。

単語の分散表現は、この周囲の単語によって作成されます。

また、単語の分散表現を作成する際、周囲の単語の使用する個数のことを、「**ウィンドウサイズ**」と呼びます。

ウィンドウサイズが2であれば、青色の単語を使用することになります。

単語の分散表現の基礎知識

Nature Insight

カウントベースの手法

分布仮説に基づいて単語をベクトル化する最も簡単な方法は、周囲の単語をカウントすることです。これを「カウントベースの手法」と名付けます。

例えば、以下では「この店に美味しいラーメンがある」「この店に美味いラーメンがある」という2つの文章を形態素解析して、ウィンドウサイズを1として共起をカウントし、合計します。

	この	店	に	美味しい	美味い	ラーメン	が	ある
この	0	2	0	0	0	0	0	0
店	2	0	2	0	0	0	0	0
に	0	2	0	1	1	0	0	0
美味しい	0	0	1	0	0	1	0	0
美味い	0	0	1	0	0	1	0	0
ラーメン	0	0	0	1	1	0	2	0
が	0	0	0	0	0	2	0	2
ある	0	0	0	0	0	0	2	0

「ラーメン」の周囲の単語は以下である。
前者の文章：「美味しい」「が」
後者の文章：「美味い」「が」

そのため、共起行列は以下のようにカウントされる。
「美味しい」：1
「美味い」　：1
「が」　　：2

このように単語の頻度をカウントして共起した行列は共起行列と呼ばれます。

カウントベースによる単語の分散表現は、行成分を取ってきます。

単語の分散表現の基礎知識

Nature Insight

cos類似度1

単語の分散表現を獲得できたので、次に類似度を計算します。

自然言語処理でよく使用される類似度の指標として「cos類似度」があります。

$$cos類似度 = \frac{x \cdot y}{|x||y|} = \frac{x_1y_1 + x_2y_2 + \cdots + x_ny_n}{\sqrt{x_1^2 + \cdots + x_n^2}\sqrt{y_1^2 + \cdots + y_n^2}}$$

ベクトルを正規化してから内積を取ることで、頻度が高い/低いという影響をできるだけ排除して、ベクトルの向きがどれだけ似ているかを考慮した指標を作成することができます。

また、cos類似度は-1~1の値を取り、1に近いほど向きが同じ方向を向いています。特にベクトルの値がすべて正の場合は、cos類似度は0~1の値を取ります。

単語の分散表現の基礎知識

Nature Insight

cos類似度2

実際にcos類似度を計算してみましょう。

先ほど作成した、「店」と「美味しい」のcos類似度を計算してみましょう。

	この	店	に	美味しい	美味い	ラーメン	が	ある	長さ
店	2	0	2	0	0	0	0	0	$2\sqrt{2}$
美味しい	0	0	1	0	0	1	0	0	$\sqrt{2}$
店×美味しい	0	0	2	0	0	0	0	0	

$$\text{「店」と「美味しい」の}cos\text{類似度} = \frac{0+0+2+0+0+0+0+0}{2\sqrt{2}\times\sqrt{2}} = 0.5$$

単語の分散表現の基礎知識

Nature Insight

cos類似度3

「美味しい」とそれ以外の単語も計算してみます。

	この	店	に	美味しい	美味い	ラーメン	が	ある
美味しいとの類似度	0	0.5	0	1	1	0	0.5	0

「美味しい」と同じ分散表現を持つ「美味い」はcos類似度は1となっています。

また、完全一致はしないものの、「美味しい」とベクトルの共通部分を持つ「店」と「が」も類似度は0.5となります。

したがって、この例では「美味しい」、「美味い」の類似度が高いといえます。

単語の分散表現の基礎知識

Nature Insight

カウントベースの問題点

カウントベースの単語の分散表現はシンプルかつ分かりやすいですが、欠点もあります。

それは、「文章量が多いとき、ベクトルの次元が膨大になる」点です。

使用する単語がそのままベクトルの次元となるため、計算も大変ですし、メモリに載らない場合もあります。

SVD(特異値分解)による次元削減方法も考えられますが、こちらも文章量が多いときにメモリエラーとなる場合があります。

これらを解決するために、ニューラルネットワークを使用した、推論ベースの手法を学んでいきます。

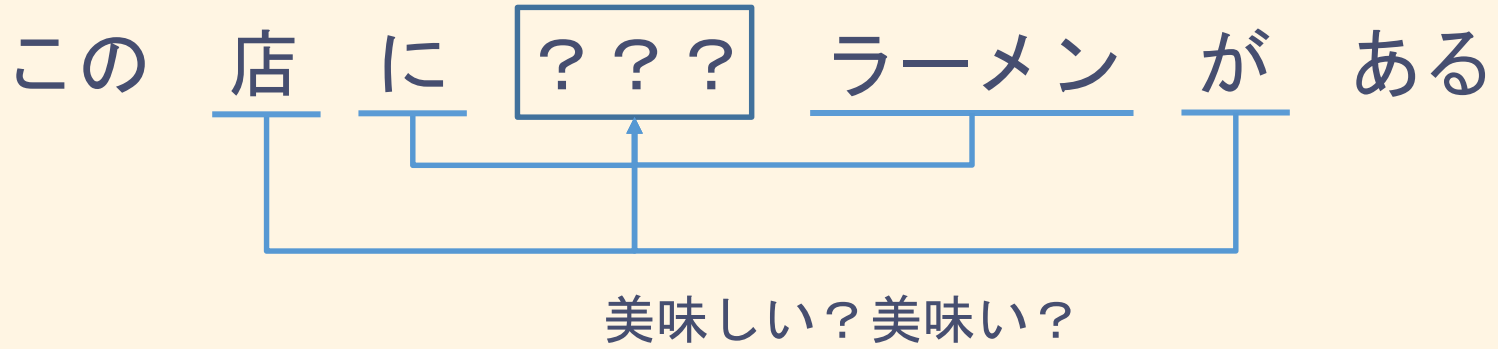
3. 推論ベースの手法「word2vec」

推論ベースの手法「word2vec」

Nature Insight

推論ベースの手法

推論ベースの手法では、以下のように周囲の単語が与えられたときに、???にどのような単語が出現するかを推測します。



推論にはニューラルネットワークを用います。

この手法を**word2vec**とよびます。

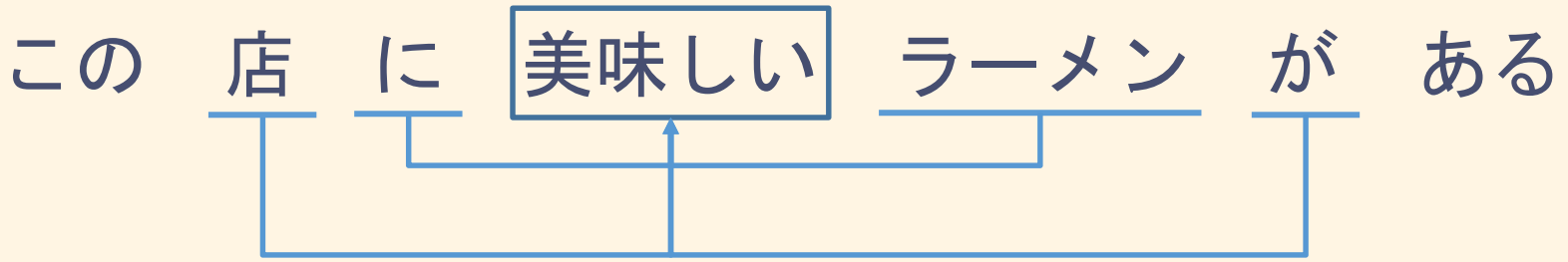
word2vecには「CBOW」と「SkipGram」の2種類の手法が存在します。

推論ベースの手法「word2vec」

Nature Insight

word2vecのCBOW

CBOWとは、**「周囲の単語からある単語を予測するタスク」**をニューラルネットワークで解き、そのときのパラメータを単語のベクトルとする手法です。



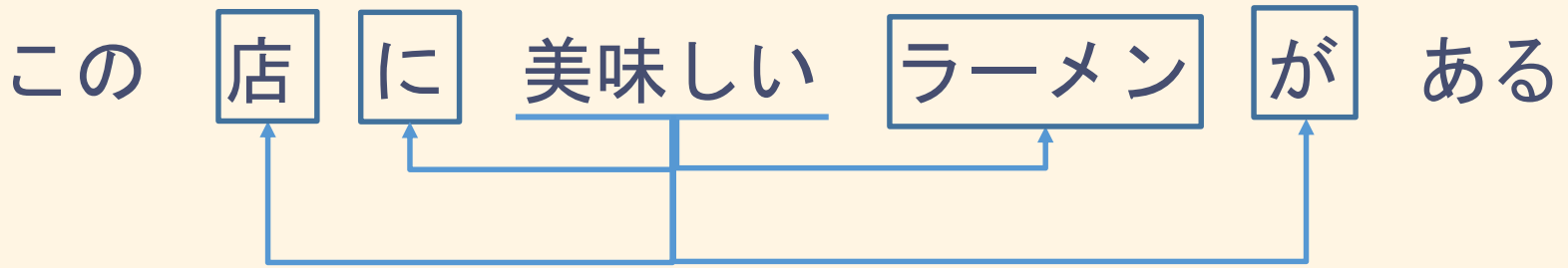
美味しいの周辺単語から「美味しい」を予測

推論ベースの手法「word2vec」

Nature Insight

word2vecのSkip-Gram

Skip-Gramとは、「**ある単語から周辺単語を予測するタスク**」をニューラルネットワークで解き、そのときのパラメータを単語のベクトルとする手法である。



「美味しい」から
美味しいの周辺単語を予測

推論ベースの手法「word2vec」

Nature Insight

単語のID化・One Hot Encoding

自然言語処理では単語を数値化することが必要なため、まずは単語を何らかの数値に置き換えます。

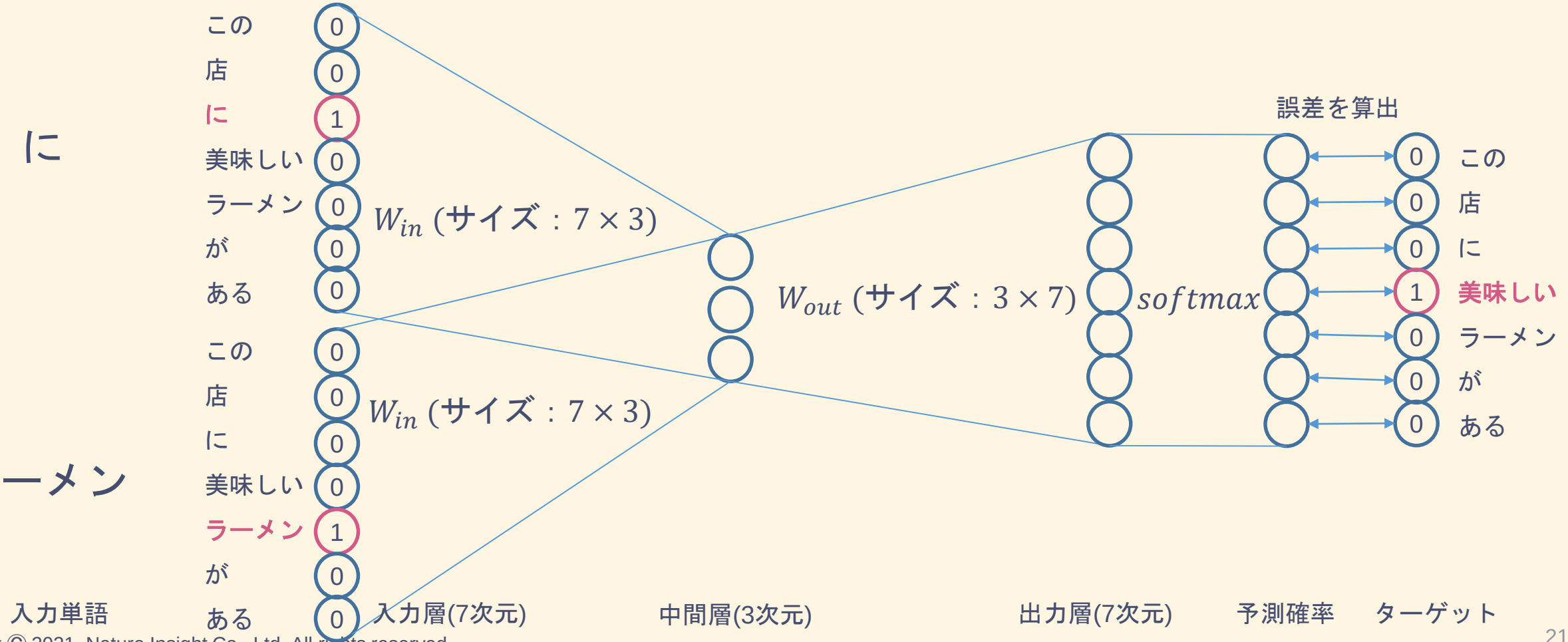
基本となる数値化にID化とOne Hot Encoding(OHE)があります。

	ID	One Hot Encoding
この	1	(1,0,0,0,0,0,0)
店	2	(0,1,0,0,0,0,0)
に	3	(0,0,1,0,0,0,0)
美味しい	4	(0,0,0,1,0,0,0)
ラーメン	5	(0,0,0,0,1,0,0)
が	6	(0,0,0,0,0,1,0)
ある	7	(0,0,0,0,0,0,1)

推論ベースの手法「word2vec」

word2vecの中身

以降、CBOWのウィンドウサイズ1で学習します。以下では、隠れ層の次元を3として、重みW_in、W_outを学習します。入力は周辺単語のOHEを使用します。

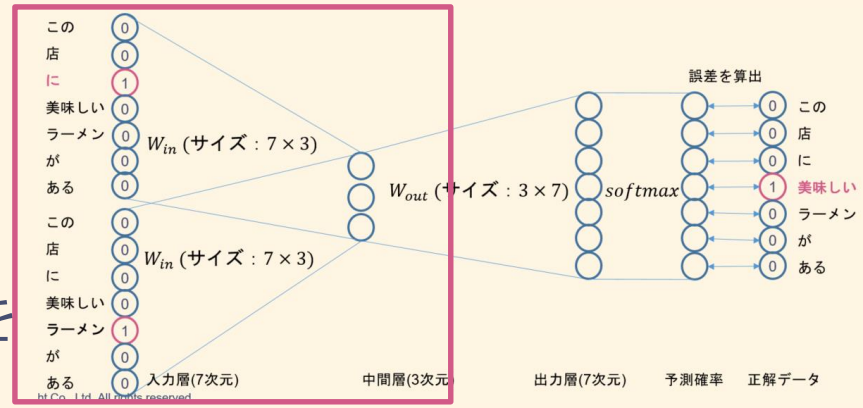


4. word2vecの中身

word2vecの中身

入力層と中間層の計算

W_inと中間層について詳しく見てみます。
Inputと重み「W_in」を**全結合**で処理します。
全結合とは、「Input」と「W_in」の行列の積を取ることです。
今回、InputはOHEなので、W_inのうち、1が立っている場所を
取り出しているのと同等の処理になっています。



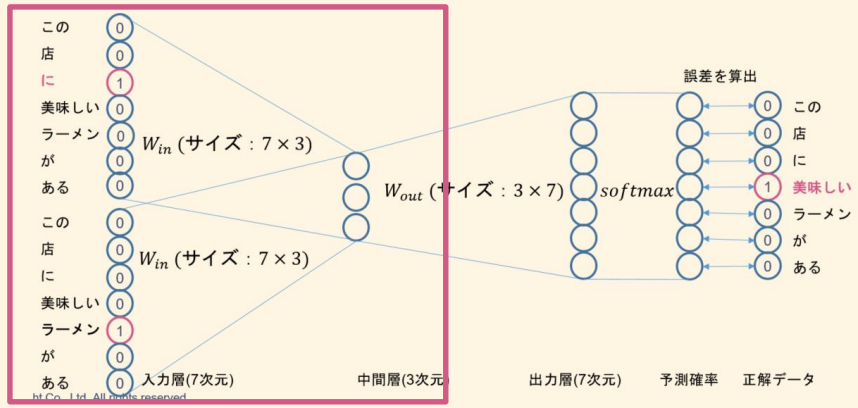
$$(0\ \ 0\ \ 1\ \ 0\ \ 0\ \ 0\ \ 0) \begin{pmatrix} 0.5 & 0.2 & 0.3 \\ 0.4 & 0.6 & 0.4 \\ 0.6 & 0.2 & -0.1 \\ -0.4 & -1.1 & -2.0 \\ 1.0 & 0.0 & -0.1 \\ 0.5 & 0.6 & 0.7 \\ 0.3 & 0.7 & 0.8 \end{pmatrix} = (0.6\ \ 0.2\ \ -0.1)$$

Input(7次元)　　W_{in} (サイズ：7×3)　　中間層(3次元)

word2vecの中身

Embedding Layer

実際に単語の数が多くなると、行列の積の計算量が
膨大になります。
OHEの全結合は、W_inの単語のidの行を取得することと
同等の処理なので、実装では行を取得する処理にします。
このLayerのことを**Embedding Layer**と呼びます。



単語「に」
(単語ID=3)

$$\begin{pmatrix} 0.5 & 0.2 & 0.3 \\ 0.4 & 0.6 & 0.4 \\ 0.6 & 0.2 & -0.1 \\ -0.4 & -1.1 & -2.0 \\ 1.0 & 0.0 & -0.1 \\ 0.5 & 0.6 & 0.7 \\ 0.3 & 0.7 & 0.8 \end{pmatrix}$$

単語IDである
3行目を抽出

$=$ $(0.6 \quad 0.2 \quad -0.1)$

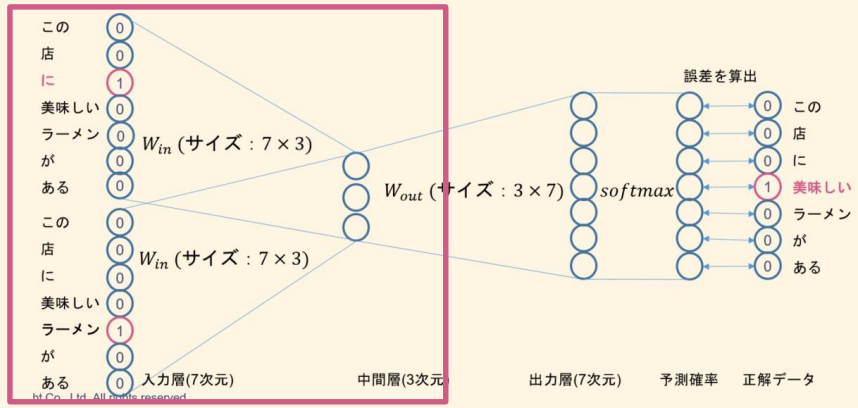
Input(単語)　　W_{in} (サイズ：7×3)　　中間層(3次元)

word2vecの中身

中間層の作成

Inputは複数単語ある(今回は「に」と「ラーメン」)ため、中間層のベクトルも複数できます。

中間層では、この複数のベクトルの合計、または平均をとります。(今回は平均を取ります。)



単語「に」の
中間層のベクトル

$(0.6 \quad 0.2 \quad -0.1)$

単語「ラーメン」の
中間層のベクトル

$(1.0 \quad 0.0 \quad -0.1)$

平均をとる

$(0.8 \quad 0.1 \quad -0.1)$

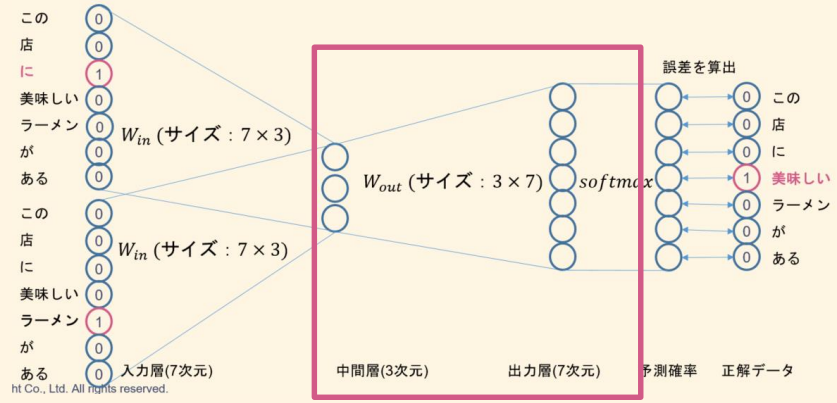
Input(単語)

中間層(3次元)

word2vecの中身

中間層と出力層の計算

中間層とW_outについて詳しく見てみます。
入力層と同様に全結合を行い、出力層を計算します。
(tは転置を表します。)



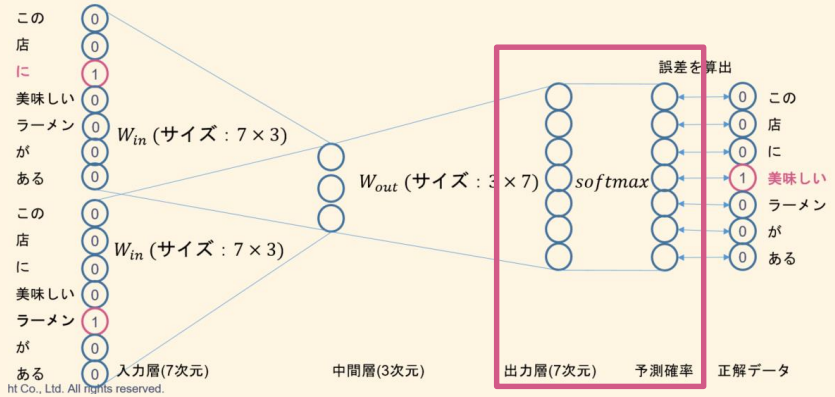
$$t\begin{pmatrix}0.8\\0.1\\-0.1\end{pmatrix}\begin{pmatrix}0.2 & 0.4 & 0.1 & 2.5 & 0.4 & 0.5 & 0.4\\0.3 & -0.1 & -0.4 & 0.5 & 0.4 & 0.4 & 0.7\\0.1 & 0.4 & 0.8 & -1.0 & -0.4 & 0.4 & -1.1\end{pmatrix} = t\begin{pmatrix}0.18\\0.27\\-0.04\\2.15\\0.4\\0.4\\0.5\end{pmatrix}$$

中間層(3次元)　　W_{out} (サイズ：3×7)　　出力層(7次元)

word2vecの中身

出力層の確率化

7次元の出力層は単語の確率分布になってほしいですが、
実際の出力層は0~1の範囲には入っていませんし、足しても
1になりません。そこで足して1になるように、
出力層にsoftmax*1関数を使用して、確率として表します。



$$t \begin{pmatrix} 0.18 \\ 0.27 \\ -0.04 \\ 0.4 \\ 0.4 \\ 0.72 \\ 0.6 \end{pmatrix} \xrightarrow{softmax^{*1}} t \begin{pmatrix} 0.07 \\ 0.08 \\ 0.06 \\ 0.51 \\ 0.09 \\ 0.09 \\ 0.10 \end{pmatrix} \begin{matrix} \text{この} \\ \text{店} \\ \text{に} \\ \text{美味しい} \\ \text{ラーメン} \\ \text{が} \\ \text{ある} \end{matrix}$$

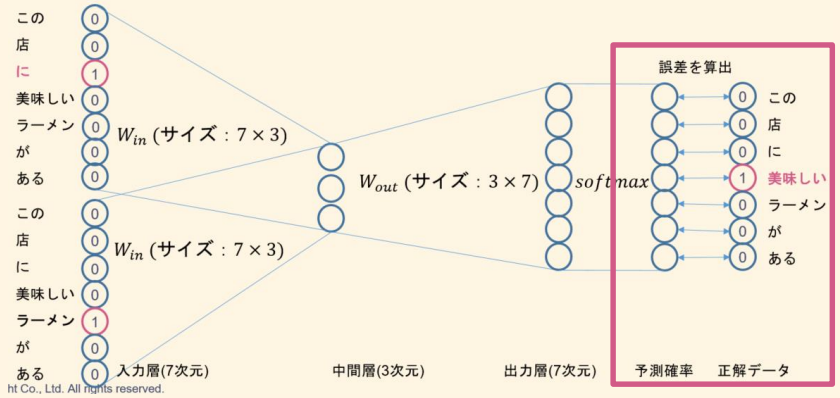
0~1の範囲で、
足して1になる

$$*1 \cdots softmax(x_i) = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}}$$

word2vecの中身

出力層と正解データの誤差1

重みを学習するために予測確率とターゲットの誤差を表す
関数として、cross entropy関数を定義します。
yを正解データ、pを予測値とすると、cross entoropyは
以下のように表されます。(Kはラベル数、kはラベルの番号)



$$CrossEntropy(\boldsymbol{y}, \boldsymbol{p}) = -\sum_{k=1}^{K} y_k \log p_k$$

y_kは0か1の値しかとらないので、次のようにも表すことができます。

$$CrossEntropy(\boldsymbol{y}, \boldsymbol{p}) = \begin{cases} -\log p_k & (y_k = 1) \\ 0 & (y_k = 0) \end{cases}$$

word2vecの中身

出力層と正解データの誤差2

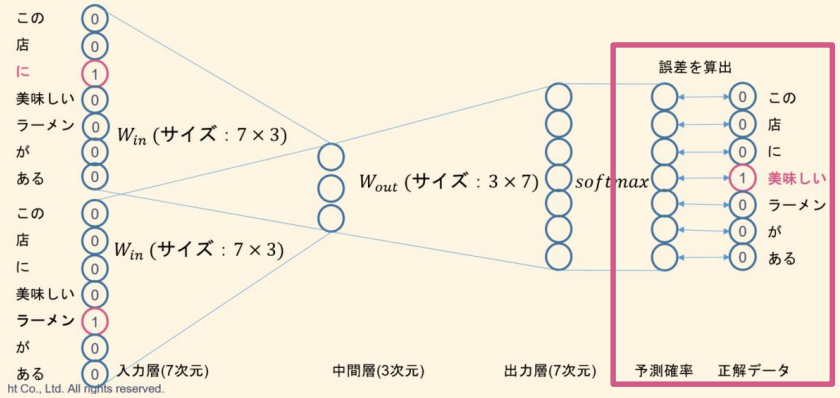
実際にcross entropy関数で誤差を算出してみます。

$$t \begin{pmatrix} 0.07 \\ 0.08 \\ 0.06 \\ 0.51 \\ 0.09 \\ 0.09 \\ 0.10 \end{pmatrix}$$

予測値：**p**

$$t \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \begin{matrix} \text{この} \\ \text{店} \\ \text{に} \\ \text{美味しい} \\ \text{ラーメン} \\ \text{が} \\ \text{ある} \end{matrix}$$

正解データ：**y**



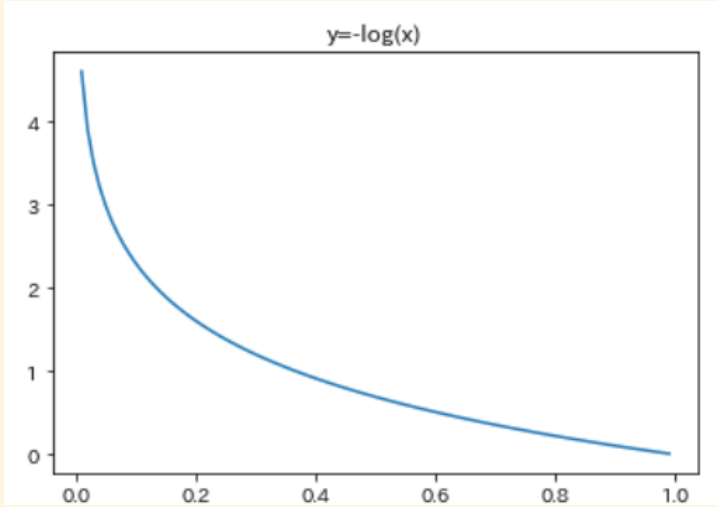
$$CrossEntropy(\boldsymbol{y}, \boldsymbol{p}) = -\log 0.51 = -0.67$$

計算方法を見てわかる通り、

-log(x)は右の図のような関数になるので、

「美味しい」の予測値が1に近ければ誤差は0に近くなり、

「美味しい」の予測値が0に近ければ誤差は大きくなります。



word2vecの中身

Nature Insight

勾配降下法

損失関数Lをすべてのデータの CrossEntropyの和で表し、それが最小となるようなパラメータW_inとW_out(以下ではまとめてWと書きます)を求めます。

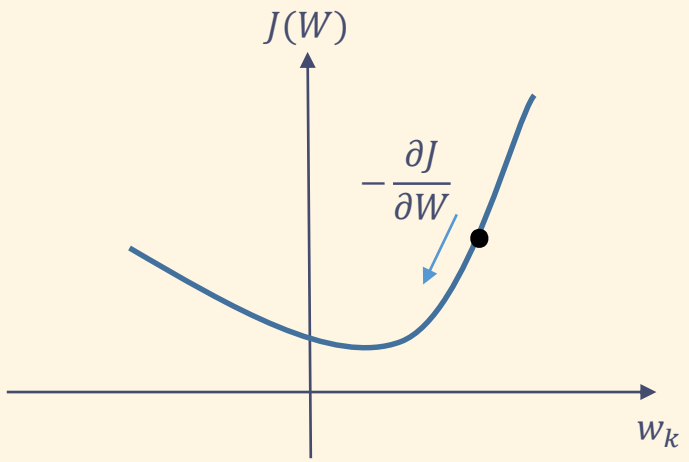
$$J(W) = \sum_{n=1}^{N} CrossEntropy(y_n, p_n)$$

本当は「損失関数の微分=0」のときのパラメータを求めたいですが、
解析的にパラメータが求まらないため、以下のようにパラメータを更新します。

$$W = W - \alpha \frac{\partial J}{\partial W}$$

αは学習率と呼ばれ、どれくらい移動するかを決めます。

Wの要素をw_kとおくと$-\frac{\partial J}{\partial W}$は傾きの逆方向、
すなわち極小値に向かうようにパラメータが更新されます。
これを繰り返し、損失関数の最小化を目指します。



word2vecの中身

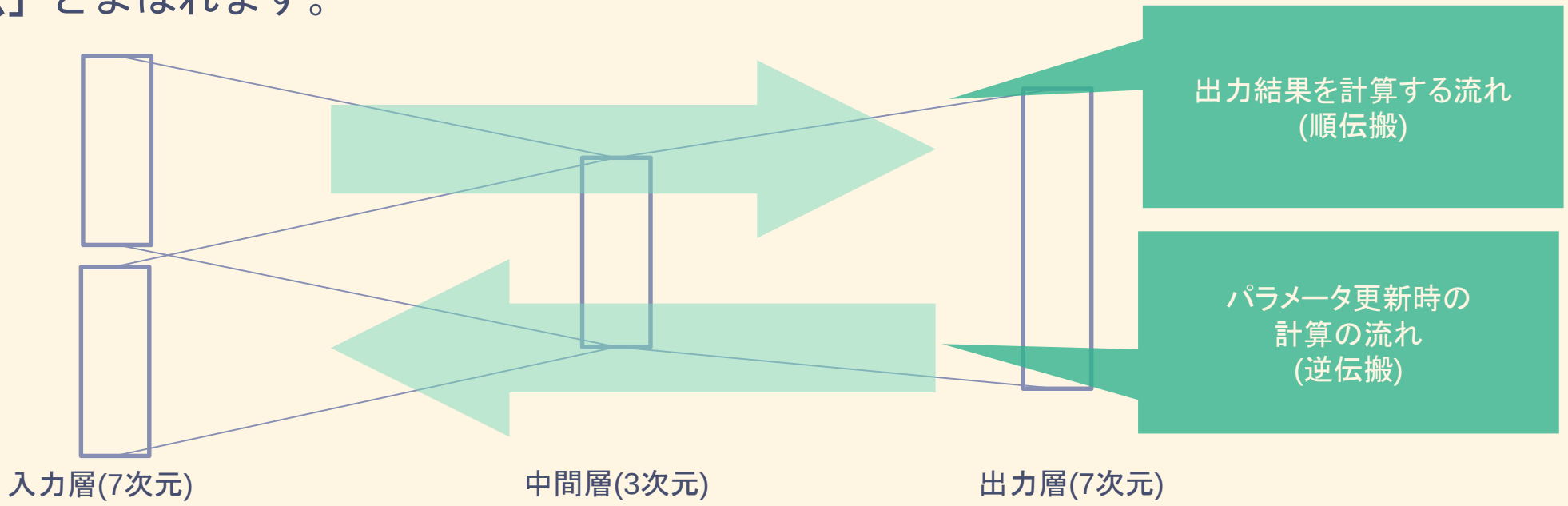
誤差逆伝播法

ここでは詳しくは計算しませんが、$\frac{\partial J}{\partial W_{in}}$を求めようとすると、$\frac{\partial J}{\partial W_{out}}$の計算結果の一部が出現します。

一般に、ニューラルネットワークのパラメータ更新では一つ先の層の計算結果の一部を使用するため、パラメータ更新をする際は出力層に近いパラメータから計算します。

通常の計算を行う方向(順伝搬)とは逆の方向にパラメータ更新していくため、

「誤差逆伝搬法」とよばれます。



word2vecの中身

単語の分散表現

単語の分散表現には、W_inを使用します。

これにより、単語の分散表現を隠れ層(今回は3次元)で表すことができます。

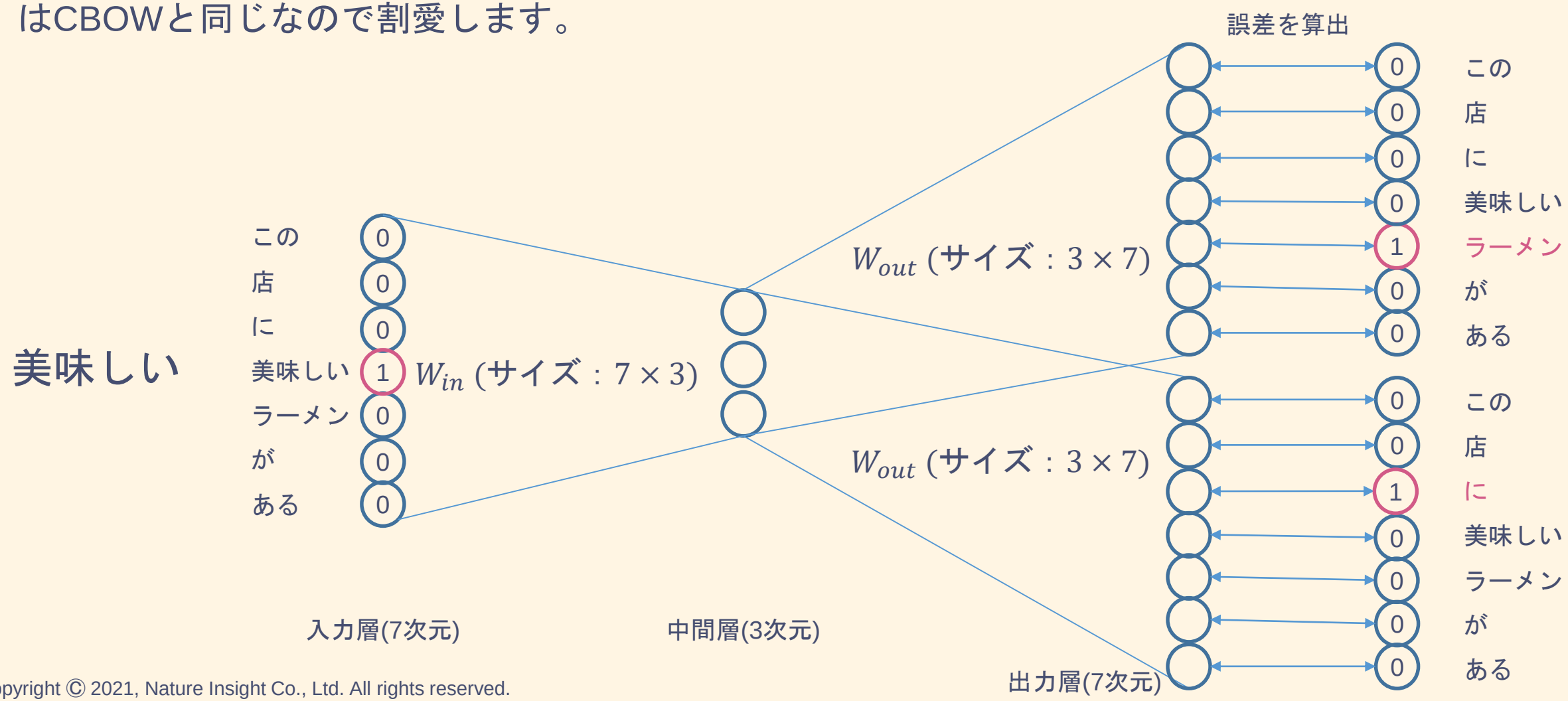
この	$0.5 \quad 0.2 \quad 0.3$
店	$0.4 \quad 0.6 \quad 0.4$
に	$0.8 \quad 0.1 \quad -0.1$
美味しい	$-0.4 \quad -1.1 \quad -2.0$
ラーメン	$1.0 \quad 0.5 \quad 0.4$
が	$0.5 \quad 0.6 \quad 0.7$
ある	$0.3 \quad 0.7 \quad 0.8$

W_{in} (サイズ：7×3)

word2vecの中身

word2vecの中身(Skip Gram)

今まではCBOWを考えましたが、Skip Gramについても考えてみます。パラメータ計算方法などはCBOWと同じなので割愛します。



5. word2vecの高速化

word2vecの中身

Nature Insight

CBOWとSkip Gram

CBOWとSkip Gramどちらを使用するべきでしょうか？

精度という観点で見ると、より難しいタスクを解いているSkip Gramのほうが優れている場合が多いです。

特に、コーパスが大規模になるにつれて、低頻出の単語や類推の問題の性能の点において、Skip-Gramのほうが優れた結果が得られる傾向にあります。

逆に学習速度の点で見ると、CBOWのほうがSkip Gramよりも高速です。これは、計算回数はCBOWのほうが少ないため、学習の計算コストが少なくなるためです。

word2vecの高速化

Nature Insight

多値分類の問題点

例えば単語数が100万であると仮定すると、softmax関数は以下のようになります。

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{k=1}^{1000000} e^{x_k}}$$

分母を毎回100万回足すのはかなり計算量が膨大になります。

さらに、中間層の計算でも100万次元×隠れ層の次元の積を行うため、
多くの時間やメモリを使用してしまいます。

そこで、Negative Samplingの実装を行います。

Negative Samplingとは負例をサンプリングして、多値分類を2値分類として学習する方法です。

特に100万の値を学習する際、正例(1)は1つで負例(0)はそれ以外のため、ほとんど負例を学習してしまい、無駄になりがちです。そのため、負例はサンプリングして学習させようとするのがNegative Samplingです。

word2vecの高速化

Nature Insight

多値分類から二値分類へ

通常のCBOWでは以下の問題を解くため、多値分類を解く必要があります。

『「に」「ラーメン」が周辺単語である単語は何か？』

二値分類にするには、以下のような問題にします。

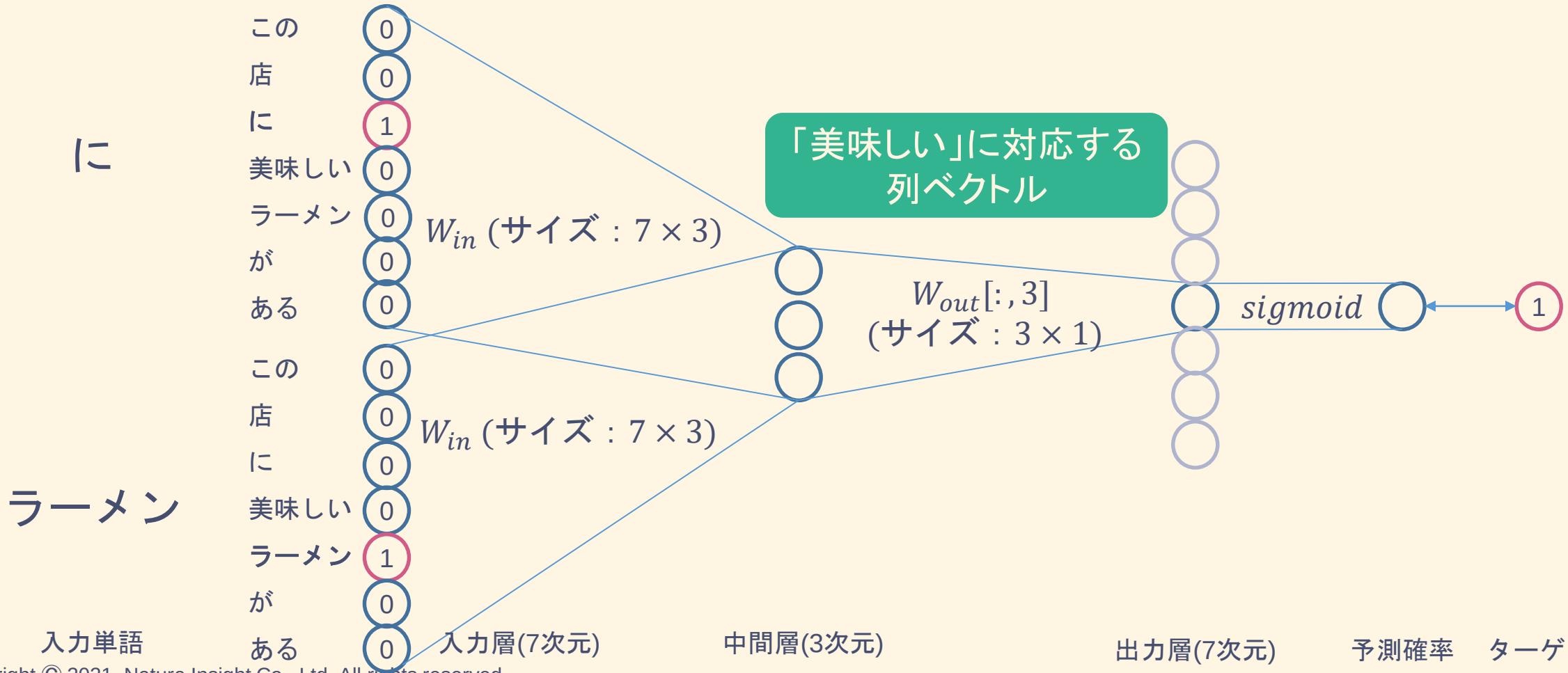
『「に」「ラーメン」が周辺単語である単語は「美味しい」か？』

この「美味しい」などの単語を適切にサンプリングすることにより、二値分類として解くことができます。

word2vecの高速化

CBOW(二値分類)

CBOWによる二値分類では、以下のようなモデルになります。

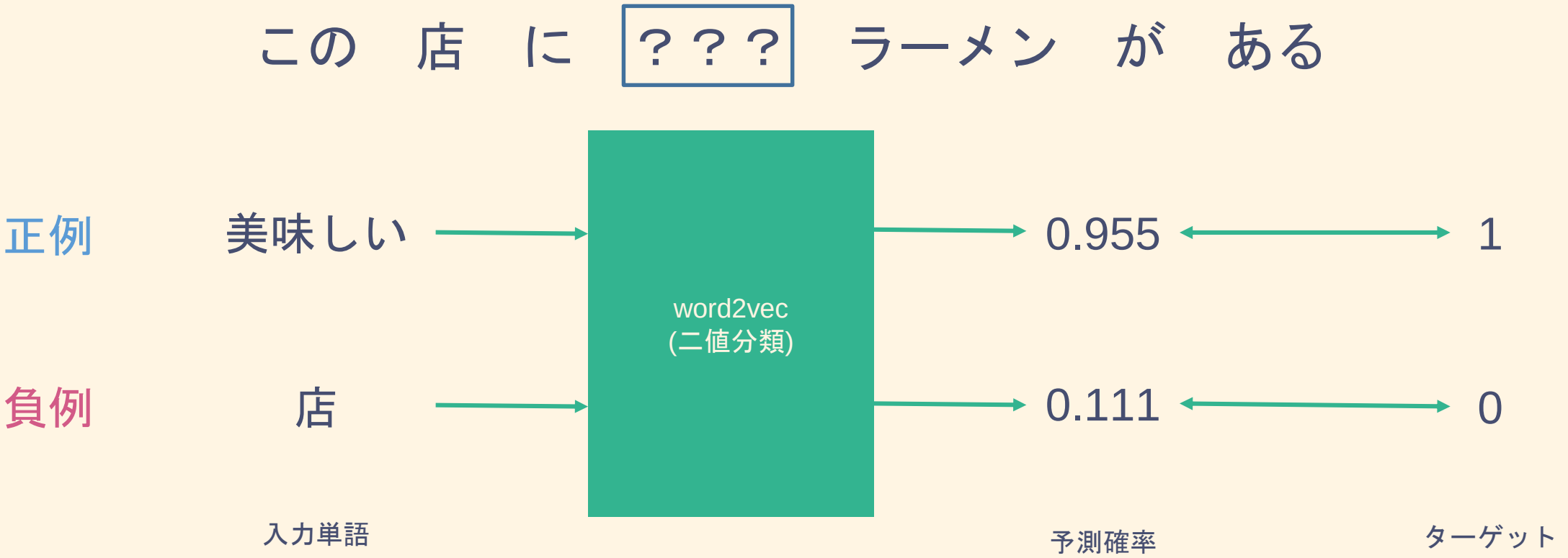


word2vecの高速化

Negative Sampling1

正例(正しいデータ)はターゲットを1として学習しますが、負例(間違ったデータ)はどのように学習すればよいでしょうか。

Negative Samplingでは、負例をいくつかサンプリングして、ターゲットを0として学習します。



word2vecの高速化

Nature Insight

Negative Sampling2

負例のサンプリングには、各単語の出現頻度を考慮します。
つまり、コーパス中で多く出現した単語ができるだけ多くサンプリングされるようにします。
実際には、以下の式の確率がサンプリングの式となります。

$$P'(w_i) = \frac{P(w_i)^{0.75}}{\sum_j P(w_j)^{0.75}}$$

ここで、w_iはi番目の単語、$P(w_i)$はw_iの出現頻度です。

0.75乗している理由は、出現頻度でサンプリングするとレアな単語は確率が極端に小さくなり、
ほとんどサンプリングされなくなってしまうため、その影響を防ぐためです。

word2vecの理論のまとめ

- word2vecは推論ベースの手法であり、推測することを目標として、
  その副産物として単語の分散表現が得られる。
- word2vecにはCBOWとskip-gramがある。
- Embedding Layerは、単語の分散表現を格納し、単語IDのベクトルを抽出する。
- word2vecでは語彙数が増加すると計算量が増えるため、Negative Samplingなどの手法を使用する。

6. word2vecの実装・結果

word2vecの実装・結果

Nature Insight

word2vecの実装方法

具体的にword2vecを実行するには以下の手順を踏みます。

このページ以降、実際のプログラムコードを交えて解説します。

実際のプログラミングコード・実装方法は以下のノートブックを参照してください。
「word2vec_yahooreview.ipynb」

word2vecの実装・結果

Nature Insight

word2vecの実行結果1

word2vecをyahooのレビューコーパスで学習させた結果を見てみます。
以下は、word2vecで作成されたベクトルをcos類似度で近さを算出した結果です。

「美味しい」と類似度が高い単語

単語	類似度
おいしい	0.788562
味つけ	0.701902
極上	0.653509
最高	0.630449
点心	0.62832
いただく	0.622383
南蛮	0.621619

「美味しい」と意味が似ている
(周辺単語が似ている)単語が出ている。

特に、ひらがな・漢字などの表記ゆれが上位に来ることが多く、word2vecに意味が似ていると判断できる。

word2vecの実装・結果

Nature Insight

word2vecの実行結果2

word2vecでは、ベクトルの加算・減算をして意味の追加・削除を行うこともできます。
以下の例では、「ラーメン」－「中華」＋「和風」=「???」を考えます。

「ラーメン」－「中華」＋「和風」に近い単語

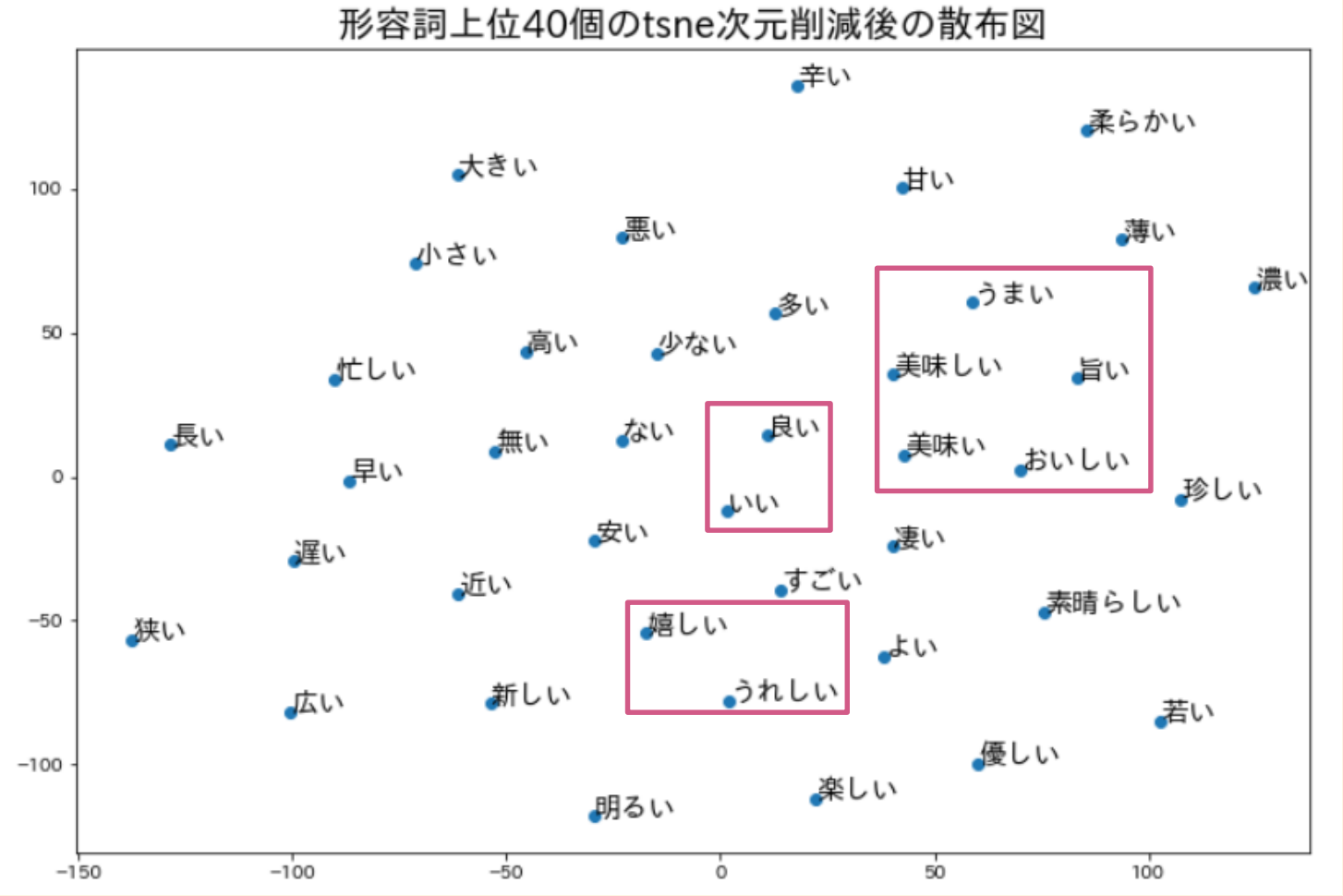
単語	類似度
ベース	0.687342
醤油	0.649936
味噌	0.630444
オーソドックス	0.600499
透明	0.592406
白味噌	0.588429
透き通る	0.586029

ラーメンの和風要素として、
「醤油」・「味噌」が
上位に来ている。

word2vecの実装・結果

word2vecの実行結果3

以下は形容詞上位40個に対して、tsneで次元削減を行い、散布図でプロットした結果です。

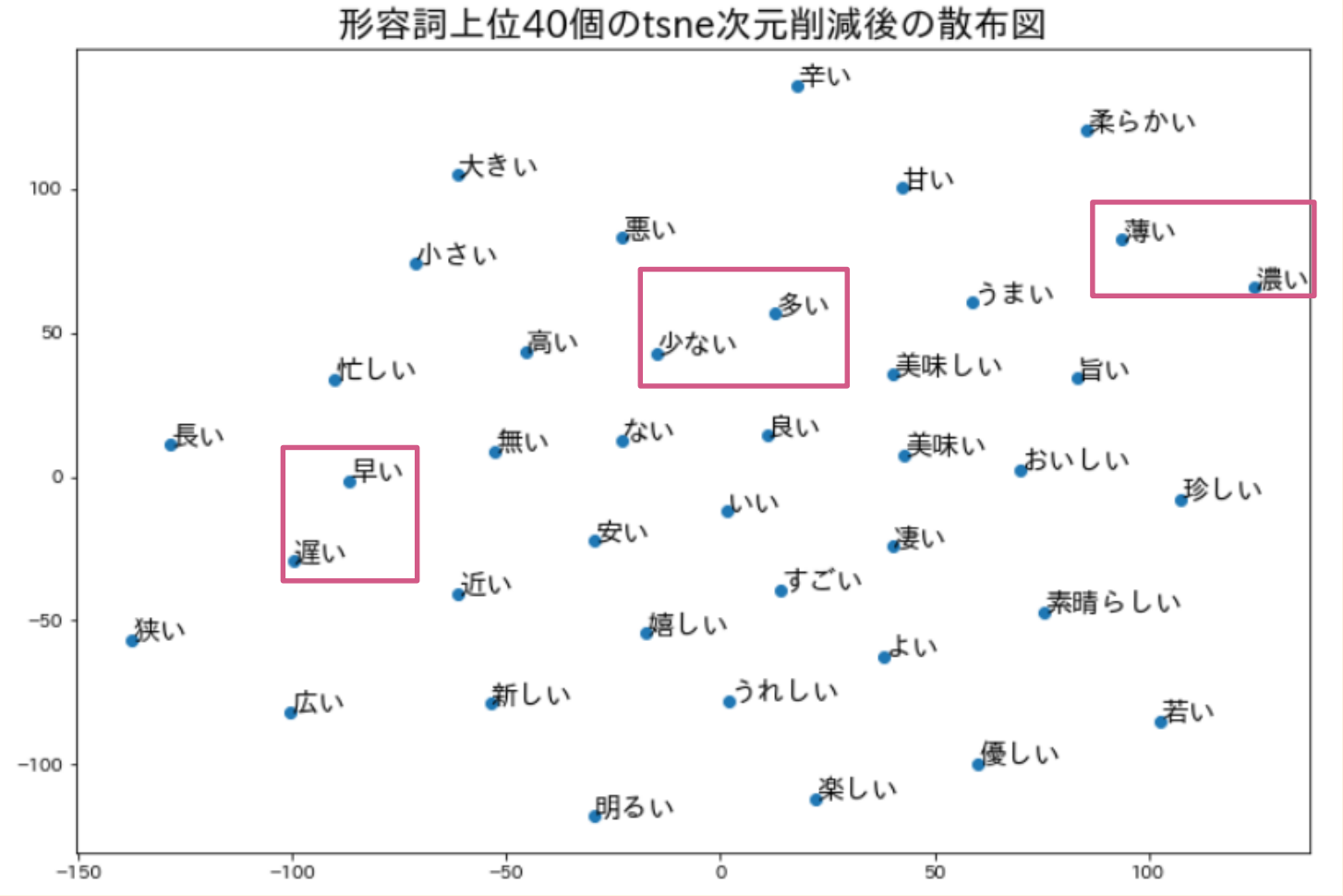


「美味しい、美味い」
「良い、いい」
「嬉しい、うれしい」など、
似た意味の単語のベクトルが近くなっています

word2vecの実装・結果

word2vecの実行結果4

以下は形容詞上位40個に対して、tsneで次元削減を行い、散布図でプロットした結果です。



「濃い、薄い」

「多い、少ない」

「早い、遅い」など、

対義語の単語のベクトルも近くなってしまいます。

これは、word2vecでは「周辺単語が同じものを類似度が高い」と判断していて、対義語の周辺単語は似通ってしまうことに起因します。

word2vecの実装・結果

Nature Insight

word2vecの実行結果5

学習データによって、word2vecの類似度は全く異なる結果になります。
以下はyahooレビューとlivedoorニュースに対してword2vecを学習した結果を比較します。

「ラーメン」と類似度が高い単語
(yahooレビュー)

単語	類似度
とんこつ	0.78812
ワンタン	0.780341
メン	0.777975
味噌	0.768431
赤味噌	0.765783
多目	0.763873
醤油	0.759327

ラーメンの種類が多い

「ラーメン」と類似度が高い単語
(livedoorニュース)

単語	類似度
ごはん	0.568596
丼	0.548891
カレー	0.546346
麺	0.545936
キムチ	0.528051
うどん	0.525322
豆乳	0.517648

料理の種類が多い

word2vecの実装・結果

Nature Insight

word2vecの評価方法

一つの判定方法として同義語判定があります。これは、公開されている単語同士の類似度付与済みデータを用いて、スピアマンの順位相関係数を測ります。

ただし、作成した目的や、正解データによって相関が異なるので、あくまで指標として考えるようにしましょう。

正解データ：
日本語単語類似度・関連度データセット JWSAN

word1	word2	similarity
か細い	弱い	3.36
きつい	甚だしい	1.92
きつい	悲しい	1.51
けばけばしい	どぎつい	3.64
さもしい	醜い	2.33

Spearmanの
順位相関係数

予測データ：
word2vecモデル

word1	word2	cos類似度
か細い	弱い	0.8
きつい	甚だしい	0.7
きつい	悲しい	0.6
けばけばしい	どぎつい	0.5
さもしい	醜い	0.7

7. word2vecの実用例

word2vecの実用例

Nature Insight

文章のベクトル化

文章中に出現した単語に対して、word2vecで作成されたベクトルを平均することで文章ベクトルを作成することができる。使用する単語は、内容語(名詞、動詞、形容詞、形容動詞、副詞など)を用いる場合が多い。

例：「この店に美味しいラーメンがある」の文章ベクトルの作成(word2vecの次元数は200)

word	特徴量1	特徴量2		特徴量200
店	1.11	-1.12	…	0.51
美味しい	0.47	0.54	…	-0.12
ラーメン	-0.55	0.45	…	-0.08
ある	0.45	-0.16	…	0.05

文章中に出現した単語の特徴量の平均を取る

文章	特徴量1	特徴量2		特徴量200
この店に美味しいラーメンがある	0.37	-0.07	…	0.09

word2vecの実用例

Nature Insight

文章の類似度(cos類似度)

文章ベクトルを作成した後にcos類似度を使用することにより、文章の類似度を計算することができます。

例：「この店に美味しいラーメンがある」とyahooレビューの類似度

文章	cos類似度
ラーメンはまあまあ美味しかったです(^-^)福重アピロスの近くにあります(o^-')b	0.9533
以前はよくランチを食べに行きましたが、昼の営業がなくなってしまい非常に残念です。どの種類のラーメンもおいしいですが、私は特にしおラーメンが最高です。開店当時はランチにもしおラーメンがあり、途中から夜だけの限定販売になってしまったラーメンです。一度試してみたらいかがでしょうか？焼酎もありますよ。	0.8797
親切、丁寧、で信頼できるドライバーでした。	-0.2062
梱包と荷物の取り扱いが雑でパソコン1台とハードディスク2台を壊されました。荷物に関する保証はあまりしてくれません。この業者は無責任でオススメできません。	-0.3381

「おいしい」と「美味しい」の類似度が高いため、文章の類似度が高い。

指定の文章とはかけ離れた意味の文章となっている。

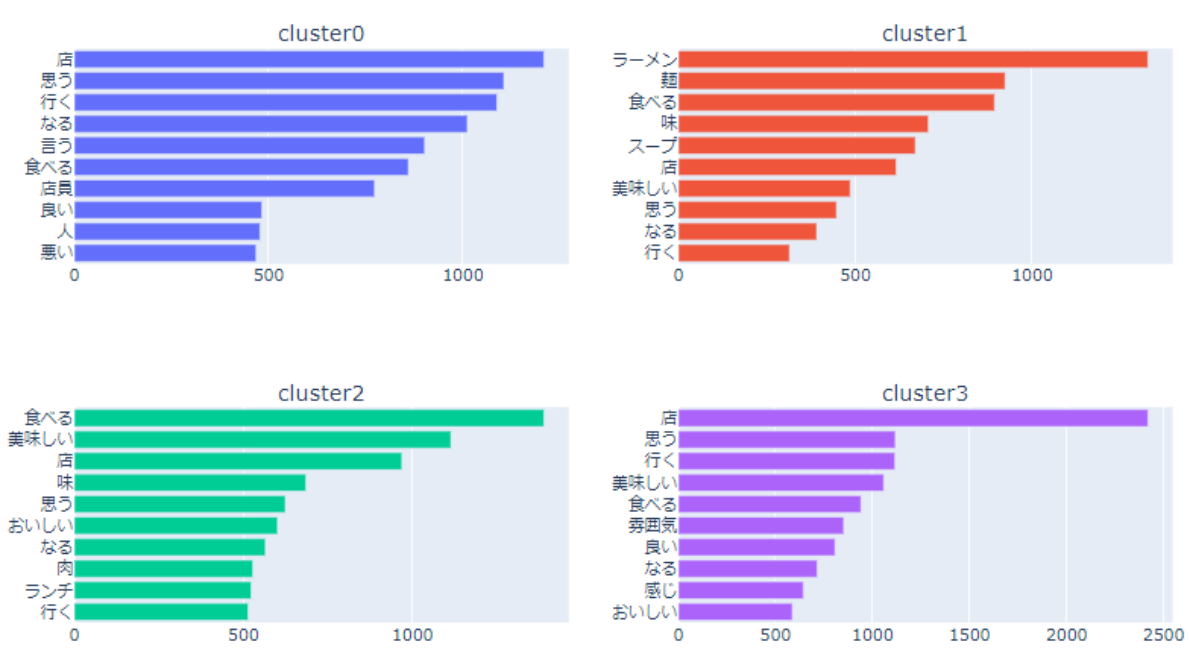
word2vecの実用例

文章のクラスタリング

例えば、レビューからお店ごとのクラスタリングを実施したい場合もあります。
その際は、お店ごとの文章ベクトルを算出し、KMeansを使用します。

例：店ごとにKMeans(k=4)でクラスタリング

クラスタ毎の単語分布



クラスタ毎の店

cluster0

DEAN & DELUCA
シェ・アオタニ
グリル&ワイン 倉庫

cluster1

味よし
西麻布五行(ラーメン屋)
銀座食堂(ラーメン屋)

cluster2

そうりの食卓
ジャクソンビル
ブラッスリー・グー

cluster3

和処 よし田 本店
ドワ
梨庵

word2vecの実用例

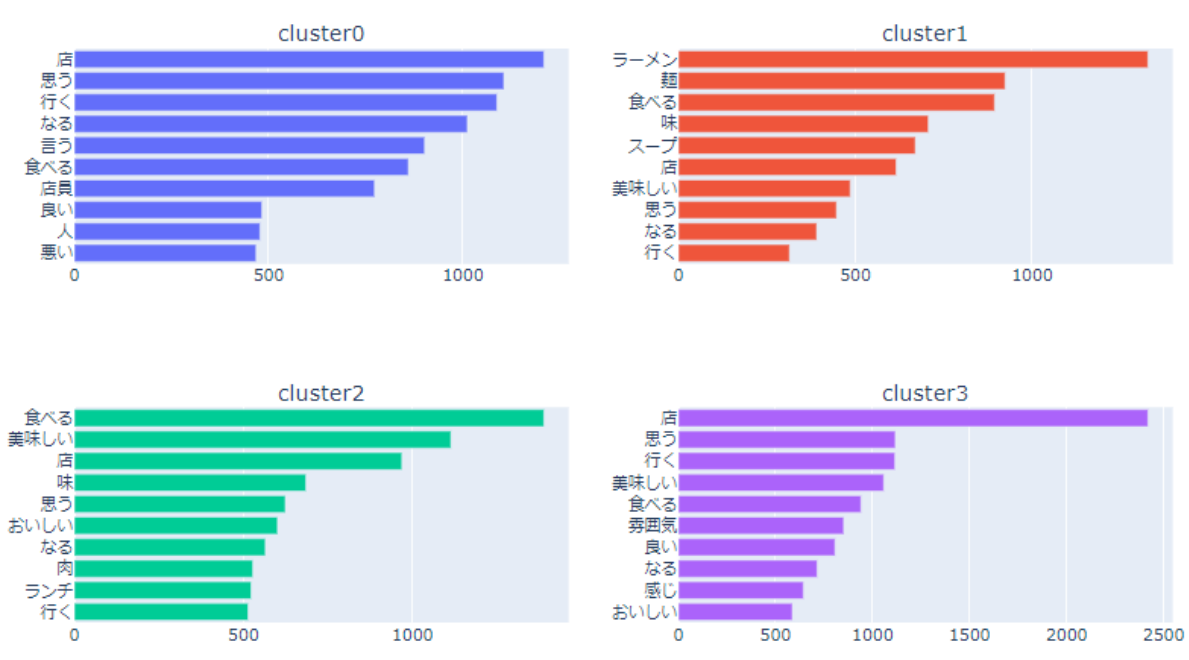
文章のクラスタリング

例えば、レビューからお店ごとのクラ…ります。
その際は、お店ごとの文章ベクトルを…

cluster0：店員のレビュー
cluster1：ラーメンのレビュー
cluster2：肉・ランチのレビュー
cluster3：雰囲気がよい店のレビュー

例：店ごとにKMeans(k=4)でクラスタリン…

クラスタ毎の単語分布



クラスタ毎の店

cluster0
DEAN & DELUCA
シェ・アオタニ
グリル&ワイン 倉庫

cluster1
味よし
西麻布五行(ラーメン屋)
銀座食堂(ラーメン屋)

cluster2
そうりの食卓
ジャクソンビル
ブラッスリー・グー

cluster3
和処 よし田 本店
ドワ
梨庵

word2vecの実用例

Nature Insight

単語と文章の類似度

単語と文章のベクトルに対してcos類似度を使用することで、単語と文章の類似度も計算することができます。これを応用すると、ある単語を入力して意味の近い店を出力する検索機能を作成できます。

例：「ラーメン+安い」と類似度の高い店

店舗名	類似度	代表的なレビュー
べんてん小関	0.863120	以前一度行ったことがあります。ラーメンがシンプルで美味しかったです。昔ながらのラーメンと言った感じでした。
博多ラーメン　勝	0.840792	友達に連れられて初めて行ってみました。今まで数あるラーメンを食べてきましたが、私はあっさりスープの細麺が好みでした。（長浜系）ここのラーメンはあっさりした中に、クリーミーさがあり、かといって豚骨の癖の強いにおいはなく、コクもありまた食べたくなる味だ！！と思いました。 ラーメン好きでもそうでない方も一度は行ってみる価値あると思います。値段も安いしね（笑） ラーメン330円　替え玉100円　3000餃子250円　その他セットメニューが沢山ありました。
長浜ラーメン福重家	0.839434	豚骨ラーメンは数あれど、これぞ博多ラーメンの味です。¥r¥r¥n通は「固めん」もしくは「バリ固」で頼むべし！ 価格も安いし、必ず替え玉はしましょう。駐車場はいつも満杯なので、時間をずらして行ってみましょう。

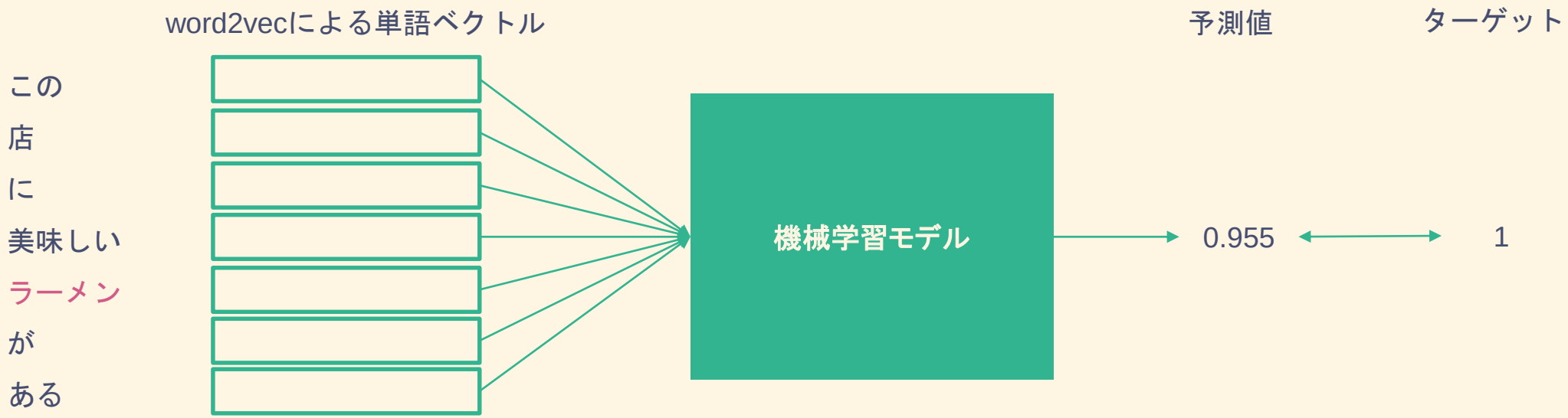
word2vecの実用例

機械学習モデルのインプットとして特徴量を使用

単語ベクトルや文章ベクトルを使用して、分類モデルや回帰モデルを構築できます。

特に、Deep Learning系の手法である1次元CNNやRNN/LSTMなどを用いると、単語の順番も考慮したモデルを作成することが可能です。

例：word2vecと機械学習モデルによる教師あり学習



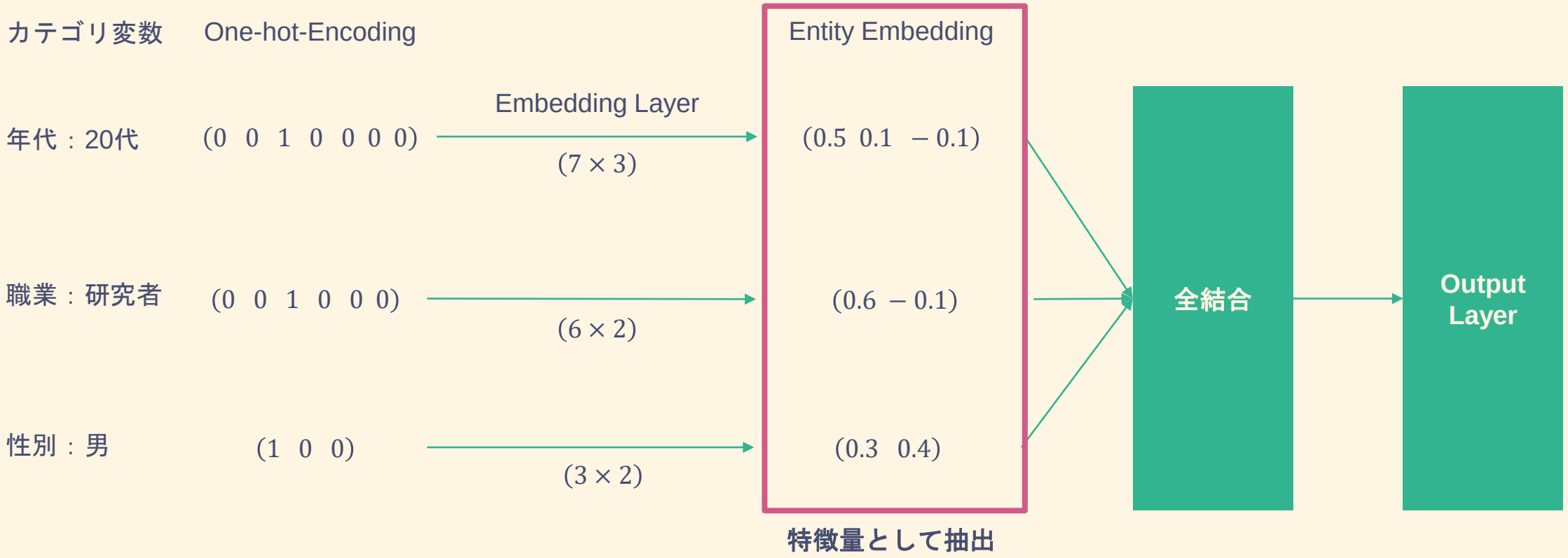
word2vecの実用例

Entity Embedding

Entity Embeddingはword2vecのカテゴリ変数版で、カテゴリ変数の分散表現を学習する方法です。

One-hot-EncodingからEmbedding Layerを用いて任意の次元の特徴量を作成することができます。

例：Embedding Layerの仕組み



8.まとめ

word2vecの理論のまとめ

Nature Insight

まとめ

- word2vecは推論ベースの手法であり、推測することを目標として、
  その副産物として単語の分散表現が得られる。
- word2vecにはCBOWとskip-gramがある。
- Embedding Layerは、単語の分散表現を格納し、単語IDのベクトルを抽出する。
- word2vecでは語彙数が増加すると計算量が増えるため、Negative Samplingなどの手法を使用する。

word2vecの実用例のまとめ

Nature Insight

まとめ

- word2vecを使用することで単語の類似度・文章の類似度・単語と文章の類似度を計算することができる。
- 類似度が高いとは、周辺単語が似ているという意味であり、対義語も上位に出てきてしまう可能性がある。
- 単語ベクトルの平均を取ることによって文章ベクトルを作成することができる。
- 文章ベクトルから文章のクラスタリングを実施することができる。
- 単語ベクトルや文章ベクトルをインプットとして機械学習モデルを構築することができる。

東京都千代田区神田小川町３-３
HF神田小川町ビルディング５階

03-3518-6061(代)

http://www.n-insight.co.jp/