

# ニューラルネットワークで自然言語処理をかじってみた

市毛 竣

芝浦工業大学 数理科学研究会

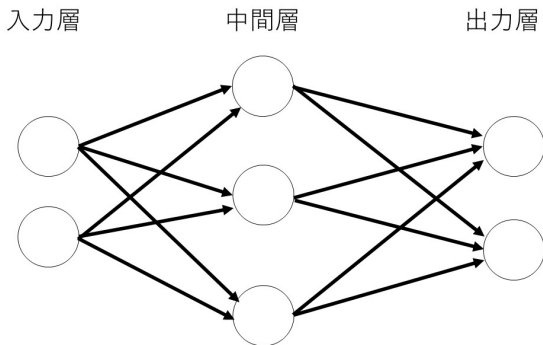
2019/11/2

## ニューラルネットワークとはなにか

ニューラルネットワークとはなにか ニューラルネットワークとは、コンピュータが行う機械学習の一種であり、人間の脳の神経(nueron)を参考にし、極力人の介入を避けられる学習を可能としたモデルである。AIが流行っている所以はこのニューラルネットワークの概念を拡張したディープラーニング(深層学習)が発達したためである。

# ニューラルネットワークの基本構造

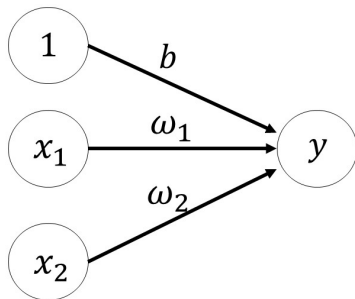
ニューラルネットワークを図に表すと下の図のようになる. 左の列を入力層, 一番右の列を出力層, 中間の列を中間層と呼ぶ (または隠れ層). 集めたデータを左の入力層のノードのから中間層を伝い, 出力層で結果を出す. その結果と実際のデータと比較をし, 学習していく. また, 各層の値を格納する場所をノードと呼ぶ.



# ノードの伝い方

ここでは、簡単に入力された値が二つの場合に出力される結果を考える。入力された2つのデータを  $x_1, x_2$  と表す。また、その際に重みと呼ばれる値を入力されたデータに乗算する。それぞれ重みを  $\omega_1, \omega_2, b$  とし、出力結果を出すために次のような計算が行われる。

$$y = \omega_1 x_1 + \omega_2 x_2 + b$$

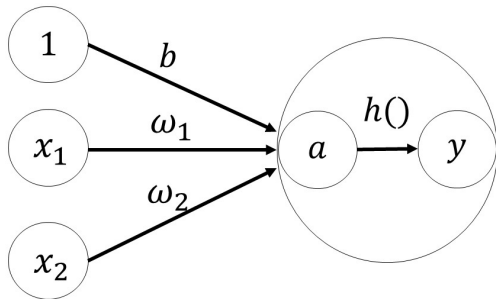


そして、ここで一度得られた計算結果  $y$  を  $a$  と置く. そしてその  $a$  に対してある関数  $f()$  を使い値を変換したものを  $y$  とする. この関数は活性化関数と呼ばれ, いくつかの関数が挙げられている.

$$a = \omega_1 x_1 + \omega_2 x_2 + b \quad (1)$$

$$y = h(a) \quad (2)$$

これら二つの式を使い, ノード間の計算は下の図のような形になる.



# 行列でニューラルネットのノードの伝い方を表現

ニューラルネットワーク上のノード間の計算は図 3 のように前の層の全ノードに対応する重み  $\omega$  をかけて, それにバイアス  $b$  を足したものが次のノードに入る.

今度は入力層のノードを 4, 中間層のノードを 3 で計算を行うと,

$$a_1 = \omega_{11}x_1 + \omega_{12}x_2 + \omega_{13}x_3 + \omega_{14}x_4 + b_1$$

$$a_2 = \omega_{21}x_1 + \omega_{22}x_2 + \omega_{23}x_3 + \omega_{24}x_4 + b_2$$

$$a_3 = \omega_{31}x_1 + \omega_{32}x_2 + \omega_{33}x_3 + \omega_{34}x_4 + b_3$$

のようになる.

これらに活性化関数を適用したものが出力となる.

$$z_j = f(a_j)(j = 1, 2, 3) \quad (3)$$

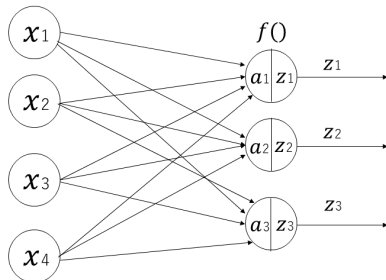


Figure: 前の層のノードから次の層のノードへ向かう際の動き



第1層のノード番号を  $i = 1, \dots, I$ , 第2層のを  $j = 1, \dots, J$  で表すと, 第1層のノードから第2層のノードの出力が決まるまでの計算は次のように一般化される.

$$a_j = \sum_{i=1}^I \omega_{ji} x_i + b_j \quad (4)$$

$$z_j = f(a_j) \quad (5)$$

ベクトルと行列を用いて表記すると, これは

$$\mathbf{a} = \mathbf{W}\mathbf{x} + \mathbf{b} \quad (6)$$

$$\mathbf{z} = f(\mathbf{a}) \quad (7)$$

のように表せる. ただし, 各ベクトルと行列は資料のように定義する.

ここで層をもう一つ増やして考える. 中間層 ( $l = 2$ ) の出力は, 式 (3)(4) にならって次のようにあらわせる.

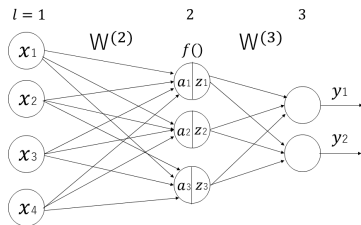


Figure: 層をもう一つ増やした NN

$$\mathbf{a}^{(2)} = \mathbf{W}^{(2)}\mathbf{x} + \mathbf{b}^{(2)}$$

$$\mathbf{z}^{(2)} = \mathbf{f}(\mathbf{a}^{(2)})$$

$\mathbf{W}^{(2)}$  は入力層と中間層間の結合の重みで,  $3 \times 4$  行列になる. また,  $\mathbf{b}^{(2)}$  は中間層の各ノードに与えられたバイアスである

次に出力層 ( $l = 3$ ) のノードの出力は  $\mathbf{x}$  を中間層の出力  $\mathbf{z}^{(2)}$  に置き換えて,

$$\mathbf{a}^{(3)} = \mathbf{W}^{(3)}\mathbf{z}^{(2)} + \mathbf{b}^{(3)}$$

$$\mathbf{z}^{(3)} = \mathbf{f}(\mathbf{a}^{(3)})$$

のように計算される.  $\mathbf{W}^{(3)}$  は中間層と出力層間の重みで, 図 6 のネットワークでは  $2 \times 3$  行列である. また,  $\mathbf{b}^{(3)}$  は出力層に与えられたバイアスである. 以上から, 任意の層数  $L$  のネットワークに一般化できる. 層  $l+1$  のユニットの出力  $\mathbf{z}^{(l+1)}$  は 1 つ前の層  $l$  の出力  $\mathbf{z}^{(l)}$  から

$$\mathbf{a}^{(l+1)} = \mathbf{W}^{(l+1)}\mathbf{z}^{(l)} + \mathbf{b}^{(l+1)} \quad (8)$$

$$\mathbf{z}^{(l+1)} = \mathbf{f}(\mathbf{a}^{(l+1)}) \quad (9)$$

# 活性化関数の種類

主な活性化関数はシグモイド関数と RELU 関数がある。

シグモイド関数

$$h(x) = \frac{1}{1 + \exp(-x)} = \frac{1}{1 + e^{-x}} \quad (10)$$

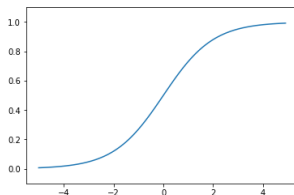


Figure: シグモイド関数のグラフ

値の範囲が  $[0; 1]$  なので確率としても表せる。

## RELU 関数

$$h(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases} \quad (11)$$

RELU 関数のグラフは下の図のようになっている. 近年ではこの関数を使ったほうがより良い成果が得られていることが知られている.

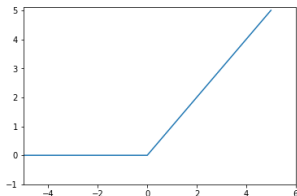


Figure: RELU 関数のグラフ

# 出力層の活性化関数

機械学習 (ニューラルネットワークを使う深層学習はこのひとつ) の問題は、'分類問題' と '回帰問題' に分けることができる。

## 分類問題

データがどのクラスに属するか. という問題

例) 人の写った画像から, そのような人が男性か女性のどちらであるかを分類する

## 回帰問題

ある入力データから (連続的な) 数値の予測をする. という問題

例) 人の写った画像から, その人の体重を予測する (57.4kg など)

ニューラルネットワークは, 以上の分類問題と回帰問題の両方に用いることができる. ただし, どちらの問題を用いるかによって出力層の活性化関数を変更する必要がある. 一般的には分類問題にはソフトマックス関数. 回帰問題には恒等関数を使う.

# 恒等関数

回帰問題に使われる恒等関数は、そのまま出力する。入ってきたものに対して何も手を加えない関数、それが恒等関数である。そのため、出力層で恒等関数を用いるときは、入力信号をそのまま出力することになる。プロセスは下の図のようになる。(σは活性化関数...今の状態は恒等関数を表している)

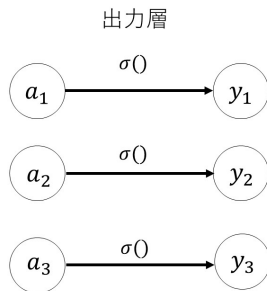


Figure: 恒等関数

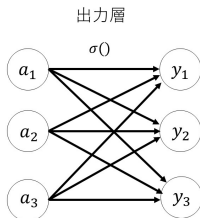
# ソフトマックス関数

分類問題で使われるソフトマックス関数は、次の式で表す。

ソフトマックス関数

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^n \exp(a_i)} \quad (12)$$

ここでは出力層のノードが全部で  $n$  個あるとして、 $k$  番目の出力  $y_k$  を求める計算式を表している。図の通りソフトマックス関数はすべての入力信号から矢印による結びつきがある。





# ニューラルネットワークの動きまとめ

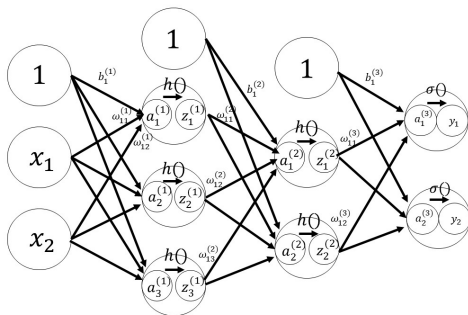


Figure: 入力データ 2 個 出力データ 2 個のニューラルネットワークの動き

ニューラルネットワーク上で行われている計算を上図のままとめておく。  
 $h(\cdot)$ ,  $\sigma(\cdot)$  は活性化関数. 各値の右上の  $()$  は層の番号, 右下の二桁目の数は次層のノードの番目, 一桁目は前層のノードの番目を表している.

# ニューラルネットワークの学習

## —— 学習とは ——

よく言われる機械学習とはその名の通り人の介入を極力避け、集められたデータから答え (パターン) を見つけようとする、試みである。その一つであるニューラルネットの学習とは先に説明した重みやバイアスを決定することである。

## —— 訓練データとテストデータ ——

機械学習の問題では、訓練データとテストデータにわけて学習や実験を行うのが一般的である。訓練データで学習 (重みやバイアスを決定) し、そしてテストデータでそのモデルを評価する。なぜテストデータが必要かというと、求められているのは汎用的な能力 (汎化能力)、つまりまだ見ぬデータに対しての能力であるためである。

例) 手書き文字をある人だけのデータを使い学習したとしても、他の人にも使えなかったら実用性がない。

学習をするために、損失関数というものを設定する。ニューラルネットワークではその損失関数を指標にして現在の状態を表し、(どれだけ精度が良いかわかり) その指標を基準にして最適な重みパラメータを探索する。それが損失関数である。重みを探索する際その損失関数が小さくなるようにする。一般的には2乗和誤差や交差エントロピー誤差などが使われる。

## — 2乗和誤差 —

2乗和誤差は損失関数を  $E$ , 出力された値を  $y_k$ , 教師データ (訓練データ) の値を  $t_k$  とすると次のようにあらわされる。

$$E = \frac{1}{2} \sum_k (y_k - t_k)^2 \quad (13)$$

例えば出力された値 (ソフトマックス関数で確率化) と教師データをそれぞれ

$$[y_0, y_1, y_2, \dots, y_9] = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]$$

$$[t_0, t_1, t_2, \dots, t_9] = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$$

とすると、値は  $E = 0.009750\dots$  となる。(ちなみに今回教師データの場合は 1 と書いているのを正解の部分としている。この表記を **one-hot** 表現という)

別の場合 (正解部分の確率が低い) のとき、

$$[y_0, y_1, y_2, \dots, y_9] = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]$$

$E = 0.5975000\dots$  となり、確かに正解から遠のくと、損失関数の値は高くなっている (適合していない)。

つぎに交差エントロピー誤差について述べる. 交差エントロピー誤差は次の式で表される.

交差エントロピー誤差

$$E = - \sum_k t_k \log y_k \quad (14)$$

先に説明した 2 乗和誤差の例の出力結果と教師データを用いると,

$$[y_0, y_1, y_2, \dots, y_9] = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]$$

のとき,  $E = 0.51308\dots$

$$[y_0, y_1, y_2, \dots, y_9] = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]$$

のとき,  $E = 2.3025\dots$  となり, 損失関数の値は高くなる.

# 損失関数の微分

ニューラルネットワークの学習では, 最適なパラメータ (重みとバイアス) を探索する際に損失関数の値をできるだけ小さくなるように探索する. そのために損失関数の微分 (勾配) を計算し, それを手掛かりにパラメータを更新する. 重み損失関数に対しての微分は「その重みパラメータを少しだけ変化させたとき, 損失関数がどのように変化するか」を表す. 例えば微分の値が負の値であったとき, 重みの値を正の方向に変化させることで損失関数を減少させることができる. 逆に, 微分の値が正の時重みを負の方向に変化させることで損失関数を減らすことができる. まとめると, 次の式で重みパラメータの更新を行う.

$$\omega = \omega - \eta \frac{\partial L}{\partial \omega} \quad (15)$$

$$\omega = \omega - \eta \frac{\partial L}{\partial \omega}$$

$\eta$  は学習率 (パラメータの更新量) と呼ばれる. 一回の学習でどれだけ学習すべきか, パラメータを更新するかを決めることを表す.  $\frac{\partial L}{\partial \omega}$  は重み  $\omega$  の微小な変化に対しての損失関数の  $L$  の変化の量を表している. このような重みパラメータの更新 10 回, 100 回... と繰り返す (繰り返し続けしすぎると過学習<sup>1</sup> となってしまう), 損失関数が小さくなるように重みパラメータを決めていくことを勾配法 (特に勾配降下法) と呼ぶ.

---

<sup>1</sup> 訓練データで学習しすぎて, その訓練データでは精度が高いが, まだ見ぬデータに対して (汎化能力が) 弱くなってしまうこと

# 学習アルゴリズムのまとめ

## ステップ 1

訓練データの中からランダムに一部のデータを選び出す。  
重みの初期値をランダムに決める<sup>a</sup>

<sup>a</sup>重みの初期値に関してはいろいろな方法があるがここでは議論しない。気になる人は Xavier の初期値や He の初期値などを調べてほしい。

## ステップ 2

出力されたデータと訓練データからなる損失関数を減らす重みの勾配を求める。

## ステップ 3

求めた変化の量を使い、重みパラメータを更新する。

$$\omega = \omega - \eta \frac{\partial L}{\partial \omega} \quad (16)$$



#### ステップ4

更新した重みした重みを使い, またステップ2から3を繰り返して最適な重みパラメータを探していく.

#### ステップ5

重みが決まったらそれを使いテストデータで汎化能力(まだ見ぬデータに対しての能力)を確認する.

# 学習の推移

学習の過程がどのようになっているのかの例を下の図に乗せる. これは 1 から 10 の数字の識別について学習をしたモデルである. **iteration** が学習した回数, **loss** が損失関数となっている. 図の通り学習を続けると損失関数の値はだんだんと減っていつている.

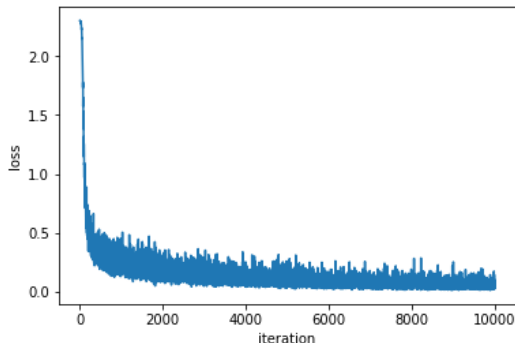


Figure: 損失関数の推移

# 汎化能力の推移

そして学習によって得られた重みパラメータを使いどのくらい適合しているのか (汎化能力があるのか) を表す図を下に乘せる. **accuracy** は精度, **epoch** は学習の途中の期間を表す. 図を見るとこのモデルは非常に高い精度を持っていることになる.

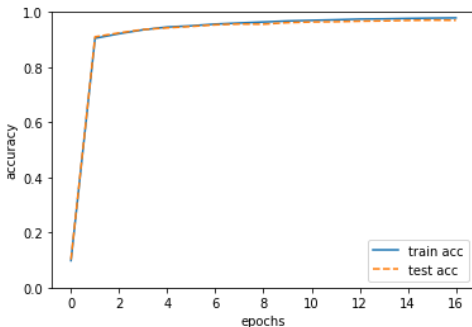


Figure: 訓練データとテストデータに対する認識精度の推移

ニューラルネットワークを使い、自然言語処理をする際に使われる手法としては、推論ベースというものが挙げられる。推論ベースの説明としては、具体的には下の文のようにある単語のコンテキスト (周囲の単語) が与えられたとき、その単語には何が入るかを推論する、ということを行う。

???に入る単語は?

you ??? goodbye and I say hello .

この場合、???のコンテキストは「you」と「goodbye」の2単語である。このように、どこにどんな単語が起こるかを機械に推論させる、というのが推論ベースの目的である。この推論問題を解くための手法が、word2vecと呼ばれるものである。word2vecには、CBOWモデルとskip-gramモデルの二通りのモデルがある。

# 単語の表現

単語をニューラルネットワーク上で計算できるように、単語 ID と one-hot 表現を使う。単語 ID とはその名の通り、単語に対応する何かしらの数字を与えたもの。one-hot 表現は先に説明した通り、データの中に一つだけ 1 を入れて、それ以外に 0 を入れるやり方である。

単語	単語 ID	one-hot 表現
you	0	(1,0,0,0,0,0,0)
say	1	(0,1,0,0,0,0,0)
goodbye	2	(0,0,1,0,0,0,0)
and	3	(0,0,0,1,0,0,0)
I	4	(0,0,0,0,1,0,0)
hello	5	(0,0,0,0,0,1,0)
.	6	(0,0,0,0,0,0,1)

上の文の場合, 語彙数が7つとして, 入力層に上の **one-hot** 表現をそのままデータとして与えることが出来る. **one-hot** 表現を使うと, 入力層のデータには下の図のように与えることが出来る. 黒の部分が1, 白の部分が0を表している.

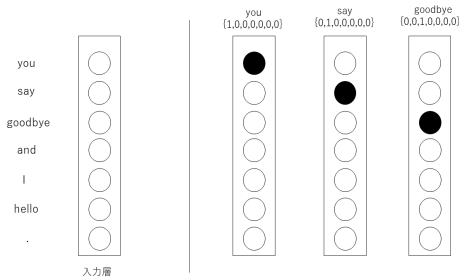


Figure: 入力層に与えられるデータ

# CBOW モデル

CBOW モデルとは、コンテキストからターゲットを推測することを目的としたニューラルネットワークである (ターゲットが中央の単語, コンテキストがその周囲の単語). CBOW モデルはコンテキストを 2 単語として下の図のように表せる.

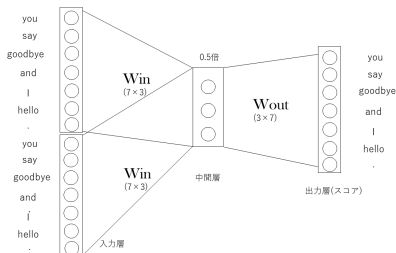


Figure: CBOW モデルの動き 1

入力層が 2 つあり, 中間層を経て出力層にたどり着く. 中間層では 2 つの計算結果を足して  $\frac{1}{2}$  をする.

出力結果にソフトマックス関数を使うことで、結果を確率として表現できる。この確率はコンテキストを与えられたとき、その中央にどの単語が来やすいかを表している。下の図の場合、入力層でコンテキストの「you」と「goodbye」が与えられ、教師ラベルがターゲットの「say」となっている。入力データから計算し、各単語が出現する確率を求め、正解ラベルと比較する。これが **CBOW** モデルの学習の流れとなっている。

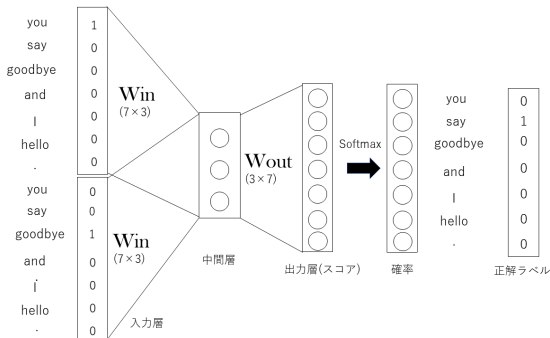


Figure: CBOW モデルの動き 2



学習する際には損失関数というのを定めなければならない. 基本的には one-hot 表現でデータを表すときには式 (14) の交差エントロピーを使うとよいことが知られている. ここで, 損失関数を確率で表現してみる.  $t$  番目の単語を求めるとして, 単語の列を  $\omega_1, \omega_2, \dots, \omega_{t-1}, \omega_t, \omega_{t+1}, \dots, \omega_T$  のようにすると, 事後確率を使って次のように表せる.

$$P(\omega_t | \omega_{t-1}, \omega_{t+1}) \quad (17)$$

これは  $\omega_{t-1}, \omega_{t+1}$  という情報が与えられた時,  $\omega_t$  が起こる確率という風に解釈することが出来る. ここで交差エントロピーの式に戻る. 式は

$$L = - \sum_k t_k \log y_k$$

で表せた.

$y_k$  は  $k$  番目に対応する事象が起こる確率 (今回の場合単語 ID の番号の単語が起きる確率),  $t_k$  は教師ラベルで入力層と同じく, **one-hot** で表されている. この時  $t$  番目の単語が正解 (教師ラベル) とすると, 先の事後確率を使い, 式 (14) の交差エントロピーは以下の式に書き換えられる.

$$L = -\log P(\omega_t | \omega_{t-1}, \omega_{t+1}) \quad (18)$$

この式を負の対数尤度という. またこの式は単に一つの単語のみに注目した物になる. そのため, 単語全体数を  $T$  とし, 全体の損失関数の値をとった後に平均をとると, 以下のようなになる.

$$L = -\frac{1}{T} \sum_{t=1}^T \log P(\omega_t | \omega_{t-1}, \omega_{t+1}) \quad (19)$$

# skip-gram モデル

skip-gram モデルとは、ターゲットからコンテキストを推測することを目的としたニューラルネットワークである。コンテキストを2単語として下の図のように表せる。

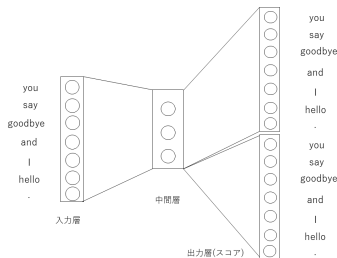


Figure: skip-gram モデルの動き 1

CBOW モデルと違い、入力層の入力1つから、出力層の出力結果を2つ出すようなモデルとなっている。

CBOW モデルと同様に, 単語の列を  $\omega_1, \omega_2, \dots, \omega_{t-1}, \omega_t, \omega_{t+1}, \dots, \omega_T$  のようにして, ターゲットの単語を  $\omega_t$ , コンテキストをそれぞれ  $\omega_{t-1}, \omega_{t+1}$  とすると, 「 $\omega_t$  が与えられたとき,  $\omega_{t-1}$  と  $\omega_{t+1}$  が同時に起きる」という (事後) 確率は次のように表せる.

$$P(\omega_{t-1}, \omega_{t+1} | \omega_t) \quad (20)$$

ここで, skip-gram モデルでは, コンテキストの単語の間に関連性がないと仮定し, 次のように分解する.

$$P(\omega_{t-1}, \omega_{t+1} | \omega_t) = P(\omega_{t-1} | \omega_t)P(\omega_{t+1} | \omega_t) \quad (21)$$

この式 (21) を交差エントロピー誤差に適用することで, skip-gram モデルの損失関数が導ける.

この式 (21) を交差エントロピー誤差に適用することで, skip-gram モデルの損失関数が導ける.

$$\begin{aligned} L &= -\log P(\omega_{t-1}, \omega_{t+1} | \omega_t) \\ &= -\log P(\omega_{t-1} | \omega_t) P(\omega_{t+1} | \omega_t) \\ &= -(\log P(\omega_{t-1} | \omega_t) + \log P(\omega_{t+1} | \omega_t)) \end{aligned}$$

またこの式は単に一つの単語のみに注目した物になる. そのため, 単語全体数を  $T$  とし, 全体の損失関数の値をとった後に平均をとると, 以下のようになる.

$$L = -\frac{1}{T} \sum_{t=1}^T (\log P(\omega_{t-1} | \omega_t) + \log P(\omega_{t+1} | \omega_t)) \quad (22)$$

# 単語の分散表現

先に求めた損失関数を用いて CBOW モデルや skip-gram モデル内では学習をしていく. つまり最終的に重みパラメータが決まっていくが, この時に決まった重みパラメータが, 単語の分散表現と呼ばれる「単語の意味」をとらえたベクトル表現となっている. 基本的に  $\text{word} \rightarrow \text{vec}$  では  $W_{in}$  を単語の分散表現として使ったほうが良いことが知られている.

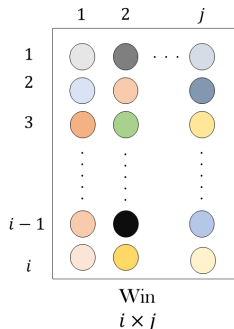


Figure: 単語の分散表現

# コサイン類似度

またその単語の分散表現を得た時に、単語間の類似度を求めることが出来る。分散表現のとある2単語のベクトルを、それぞれ  $\mathbf{x} = (x_1, x_2, x_3, \dots, x_n)$  と  $\mathbf{y} = (y_1, y_2, y_3, \dots, y_n)$  とすると、次の式で類似度を求める。

$$\text{similarity}(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} = \frac{x_1 y_1 + \dots + x_n y_n}{\sqrt{x_1^2 + \dots + x_n^2} \sqrt{y_1^2 + \dots + y_n^2}} \quad (23)$$

これをコサイン類似度と呼ぶ。1に近いほど、二つのベクトルの向きが同じ方向を向いていることを表し、-1に近いと逆の方向を向いていることを表している。つまり word2vec で得た単語の分散表現内で単語の意味が近い、遠い分かるということになる。

実際に学習をし、単語間の意味をコサイン類似度で求めた例を下の表に表す。なお、学習をする際に使ったデータは PTB データセット<sup>2</sup>である。

[word]	you	[word]	year	[word]	toyota
we	0.6105	month	0.7182	ford	0.5505
someone	0.5917	week	0.6522	instrumentation	0.5100
i	0.5543	spring	0.6269	mazda	0.4936
something	0.4900	summer	0.6258	bethlehem	0.4748
anyone	0.4734	devade	0.6030	nissan	0.4746

結果をみると、「you」と「we」, 「year」と「month」など、私たちの直感にあっていう結果と言える。




<sup>2</sup>Penn Treebank(ペン・ツリー・バンク)という word2vec の開発者, Tomas Mikolov 氏が提供している単語のデータセット。プログラムで書く際に便利な前処理をしているため、色々な場所で使われている



今回のまとめとしては

- ニューラルネットワークの基本的な構造と、学習のアルゴリズムをまとめた。学習をする際には損失関数を用いて教師データとの比較を行い、それをもとに重みパラメータの更新、すなわち学習をしていく。
- `word2vec` ではコンテキストを利用して、ターゲットを推論する `CBOW` モデルと、ターゲットからそのコンテキストを推論する `skip-gram` モデルがある。それで得た重みパラメータが単語の分散表現となり、単語の意味をもつ。

今回紹介できなかった `word2vec` の高速化方法に、`Embedding` レイヤと `Negative Sampling` というものがある。興味のある方は後で私に聞いてください。

-  斎藤 康毅, ゼロから作るディープラーニング–Pythonで学ぶディープラーニングの理論と実装–, オライリージャパン出版, 2016.
-  斎藤 康毅, ゼロから作るディープラーニング–自然言語処理編–, オライリージャパン出版, 2018.
-  岡谷 貴之, 深層学習 (機械学習プロフェッショナルシリーズ), 講談社出版, 2015.