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

芝浦工業大学 数理科学研究会

2019 年 11 月 2 日

※何か不明な点や計算ミス等がありましたら加筆修正しますので指摘をお願いします

制作: AL18018 市毛 竣

目次

1　研究背景

最近何かと話題になっている人工知能．その中でも自然言語処理について数多くの研究がなされている．特に深層学習の発達により, 以前よりも人間らしい会話ができるようになっており, これから社会にもっと普及していく技術である．今回は既に勉強していたニューラルネットワークの理論と自然言語処理はどのように組み合わさっているのかを知りたいと思い, 後の研究につなげるため発表に至る．

1.1　ニューラルネットワークの小歴史

ニューラルネットワーク自体は 1900 年代中盤にすでに発掘されたものである．ニューラルネットワークの概念はある程度確立されていたものの 1900 年代後半には冷や飯を食っている状態だった．

2　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.

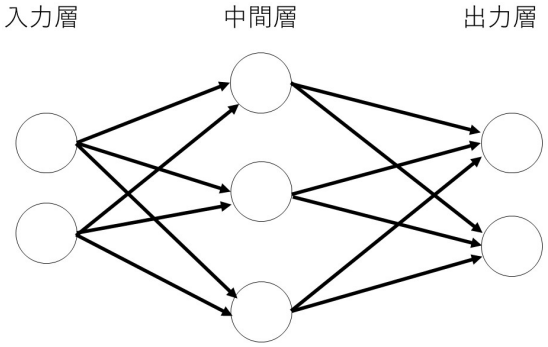

図 1　ニューラルネットの一例

2.1　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を行われる.

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

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 $f()$ 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.

$$a = \omega_1 x_1 + \omega_2 x_2 + b \tag{1}$$

$$y = h(a) \tag{2}$$

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.

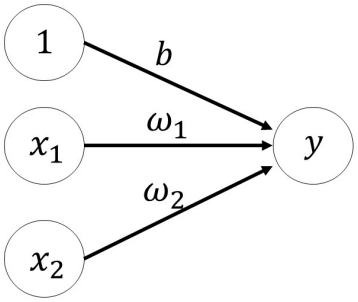

図 2　入力層から中間層の計算

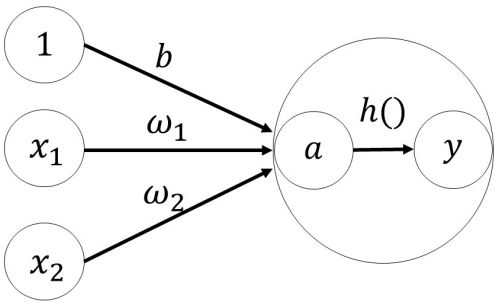

図 3　活性化関数によるプロセス

2.2　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 ω 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.

今度は下の図 5 をもとに計算を行うと,

$$
\begin{aligned}
a_1 &= \omega_{11}x_1 + \omega_{12}x_2 + \omega_{13}x_3 + \omega_{14}x_4 + b_1 \\
a_2 &= \omega_{21}x_1 + \omega_{22}x_2 + \omega_{23}x_3 + \omega_{24}x_4 + b_2 \\
a_3 &= \omega_{31}x_1 + \omega_{32}x_2 + \omega_{33}x_3 + \omega_{34}x_4 + b_3
\end{aligned}
$$

のようになり, これらに活性化関数を適用したものが出力となる.

$$z_j = f(a_j)(j = 1, 2, 3) \tag{3}$$

第 1 層のノード番号を $i = 1, \cdots, I$, 第 2 層のを $j = 1, \cdots, J$ 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.

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

$$z_j = f(a_j) \tag{5}$$

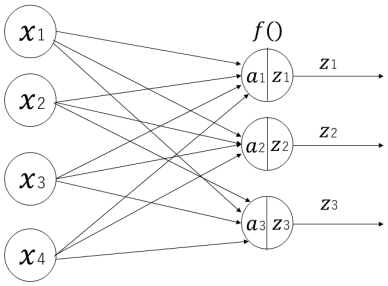

図 4　前の層のノードから次の層のノードへ向かう際の動き

ベクトルと行列を用いて表記すると, これは

$$\boldsymbol{a} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \tag{6}$$
$$\boldsymbol{z} = \boldsymbol{f}(\boldsymbol{a}) \tag{7}$$

のように表せる. ただし, 各ベクトルと行列は次のように定義する.

$$\boldsymbol{a} = \begin{pmatrix} a_1 \\ \vdots \\ a_J \end{pmatrix}, \boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_I \end{pmatrix}, \boldsymbol{b} = \begin{pmatrix} x_1 \\ \vdots \\ x_J \end{pmatrix}, \boldsymbol{z} = \begin{pmatrix} z_1 \\ \vdots \\ z_I \end{pmatrix},$$

$$\boldsymbol{W} = \begin{pmatrix} \omega_{11} & \dots & \omega_{1I} \\ \vdots & \ddots & \vdots \\ \omega_{J1} & \dots & \omega_{JI} \end{pmatrix}, \boldsymbol{f}(\boldsymbol{a}) = \begin{pmatrix} f(a_1) \\ \vdots \\ f(a_J) \end{pmatrix}$$

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.

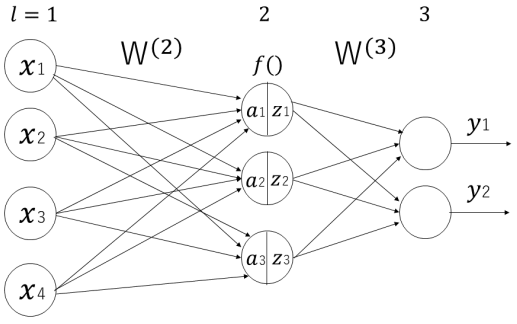

図 5　層をもう一つ増やした NN

$$\boldsymbol{a}^{(2)} = \boldsymbol{W}^{(2)}\boldsymbol{x} + \boldsymbol{b}^{(2)}$$
$$\boldsymbol{z}^{(2)} = \boldsymbol{f}(\boldsymbol{a}^{(2)})$$

$\boldsymbol{W}^{(2)}$ は入力層と中間層間の結合の重みで, 図 6 のネットワークでは, 3×4 行列になる. また, $\boldsymbol{b}^{(2)}$ 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 $\boldsymbol{x}$ を中間層の出力 $\boldsymbol{z}^{(2)}$ に置き換えて,

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

のように計算される. $\boldsymbol{W}^{(3)}$ は中間層と出力層間の重みで, 図 6 のネットワークでは 2×3 行列である. また, $\boldsymbol{b}^{(3)}$ 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 $\boldsymbol{z}^{(l+1)}$ は 1 つ前の層 l の出力 $\boldsymbol{z}^{(l)}$ から

$$\boldsymbol{a}^{(l+1)} = \boldsymbol{W}^{(l+1)}\boldsymbol{z}^{(l)} + \boldsymbol{b}^{(l+1)} \tag{8}$$
$$\boldsymbol{z}^{(l+1)} = \boldsymbol{f}(\boldsymbol{a}^{(l+1)}) \tag{9}$$

のように計算できる.

2.3 活性化関数の非線形性

なぜ活性化関数が必要なのかというと, それは線形性と非線形性に理由がある. 線形性の定義は次のようになる.

定義 2.1 関数などの演算 f が次の x, y, p を満たすとき, f のそのような性質を線形性という

1 $f(x+y) = f(x) + f(y)$
2 $f(px) = pf(x)$

さて, この線形性が活性化関数とどのような関係があるのか. 結論から言うと活性化関数は非線形の関数である必要がある. それを示すために次の証明を行う.

証明 線形性である演算 f, g があるとき, 合成関数 $f \circ g$(もしくは $g \circ f$) は線形性である.
線形性である条件を f, g に適応する

1 $f(x+y) = f(x) + f(y)$

2 $f(px) = pf(x)$

3 $g(x+y) = gf(x) + g(y)$

4 $g(px) = pg(x)$

これらを使い
$f \circ g(x+y) = f \circ (g(x) + g(y)) = f \circ g(x) + f \circ g(y)$
$f \circ g(px) = f \circ pg(x) = pf \circ g(x)$
が求められる.$g \circ f$ も同様に求めることができる. 以上より題意は満たされた.

例 2.1 活性化関数を $f(x) = cx$ とする.(線形性) 0 層目の入力を x_1, x_2, x_3 1 層目出力を y_1, y_2, 2 層目の出力を z_1, z_2. $\omega_{ji}^{(k)}$ は k 層目の i 番目の入力ノードから j 番目の出力ノードに向けての重みとすると

1 $y_1 = c(\omega_{11}^{(1)}x_1 + \omega_{12}^{(1)}x_2 + \omega_{13}^{(1)}x_3)$

2 $y_2 = c(\omega_{21}^{(1)}x_1 + \omega_{22}^{(1)}x_2 + \omega_{23}^{(1)}x_3)$

3 $z_1 = c(\omega_{11}^{(2)} y_1 + \omega_{12}^{(2)} y_2) = c^2(\omega_{11}^{(2)}(\omega_{11}^{(1)} x_1 + \omega_{12}^{(1)} x_2 + \omega_{13}^{(1)} x_3) + \omega_{12}^{(2)}(\omega_{21}^{(1)} x_1 + \omega_{22}^{(1)} x_2 + \omega_{23}^{(1)} x_3)) = c^2((\omega_{11}^{(2)}\omega 11^{(1)} + \omega_{12}^{(2)}\omega 21^{(1)})x_1 + (\omega_{11}^{(2)}\omega_{12}^{(1)} + \omega_{12}^{(2)}\omega_{22}^{(1)})x_2 + (\omega_{11}^{(2)}\omega_{13}^{(1)} + \omega_{12}^{(2)}\omega_{23}^{(1)})x_3)$

4 $z_1 = c(\omega_{21}^{(2)} y_1 + \omega_{22}^{(2)} y_2) = c^2(\omega_{21}^{(2)}(\omega_{11}^{(1)} x_1 + \omega_{12}^{(1)} x_2 + \omega_{13}^{(1)} x_3) + \omega_{22}^{(2)}(\omega_{21}^{(1)} x_1 + \omega_{22}^{(1)} x_2 + \omega_{23}^{(1)} x_3)) = c^2((\omega_{21}^{(2)}\omega 11^{(1)} + \omega_{22}^{(2)}\omega 21^{(1)})x_1 + (\omega_{21}^{(2)}\omega_{12}^{(1)} + \omega_{22}^{(2)}\omega_{22}^{(1)})x_2 + (\omega_{21}^{(2)}\omega_{13}^{(1)} + \omega_{22}^{(2)}\omega_{23}^{(1)})x_3)$

[3][4] の x_i の左の括弧内の重みをまとめて $\acute{\omega}_i$ とまとめると結局は $f(x) = c^2 x$ で表すことができる.

つまりは活性化関数が線形であると層を深くしても単層のみでできるため, 中間層で何度計算をやっても意味がない. もっと言えばニューラルネットワークでやる意味がなくなってしまう. そのためこれから紹介する活性化関数は非線形性を持つようになっている.

2.4　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

e は自然対数 2.7182...である.

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

漸近線は $h(x) = 1$ と $h(x) = 0$

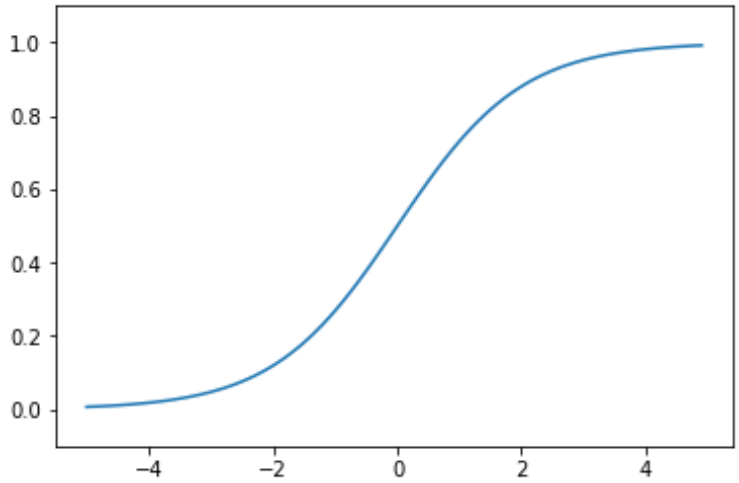

図 6　シグモイド関数のグラフ

下の図がシグモイド関数のグラフとなっている. シグモイド関数では出力される値は 0.731...,0.8456... など 0 から 1 の範囲のなかで出力される. この連続的な実数がニューラルネットワークでは重要な役割を持つ.

RELU 関数

$$h(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leqq 0) \end{cases} \tag{10}$$

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.

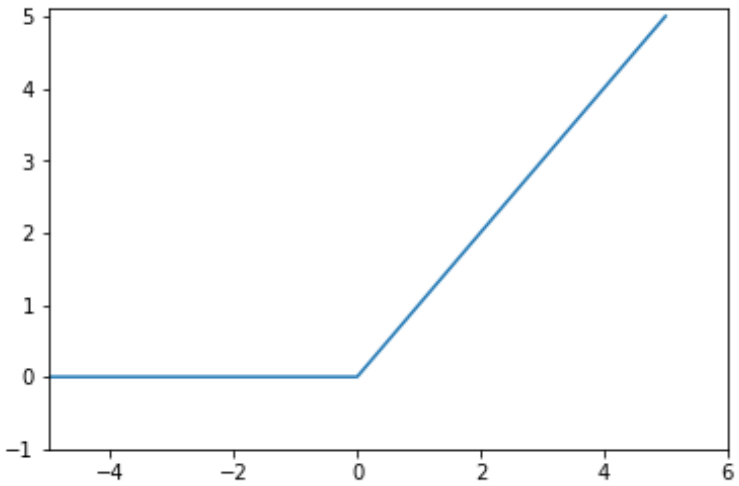

図 7　RELU 関数のグラフ

2.5　出力層の活性化関数

出力層で使われる活性化関数は中間層と違い, ソフトマックス関数と呼ばれる関数を扱う.

ソフトマックス関数

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)} \tag{11}$$

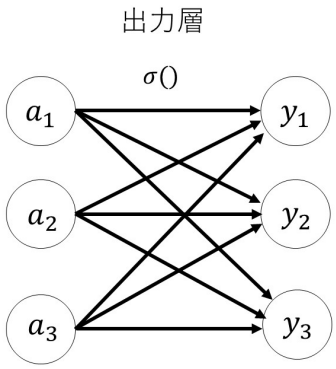

図 8　ソフトマックス関数

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 をもともめる計算式を表している. プロセスは下の図のようになっている. 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. また式から分かる通り, この式を確率としてみることができる. という重要な性質を持っている.

2.6　ソフトマックス関数の改善

知っての通り, ニューラルネットワークはコンピュータなどを使い演算される. そこでコンピュータを使う上で問題とされるものがオーバーフローである. ソフトマックス関数では, 指数関数の計算を行うことになるが, その際, 指数関数の値が容易に大きくなってしまう. 例えば e^{10} は 20000 を超え, e^{100} では正 (せい) 超し (10 の 40 乗), e^{1000} ではコンピュータ上で inf... 無限が返ってくる. このような大きな値で計算してしまうとどうしても値が不安定になっ

てしまう. そのため, 次のような式に変形する.

$$\begin{aligned} y_k &= \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)} \\ &= \frac{C \exp(a_k)}{C \sum_{i=1}^{n} \exp(a_i)} \\ &= \frac{\exp(a_k + \log C)}{\sum_{i=1}^{n} \exp(a_i + \log C)} \\ &= \frac{\exp(a_k + A)}{\sum_{i=1}^{n} \exp(a_i + A)} \end{aligned} \tag{12}$$

例えば a_i の中の最大の数が 1000 だとすると, A の値を 1005 などにするとオーバーフローすることなく計算することができる.

2.7　ニューラルネットワークの動きまとめ

ニューラルネットワーク上で行われている計算を下の図のまとめておく. 流れとしては, 入力されたデータ (x_1 と x_2) を重みと呼ばれる ω とバイアス b を使い, 計算した結果をノードに入れる. その後活性化関数で変換し, 新しく得たデータ z と, 重み (先ほど使った重みとは別の重み) とバイアス (先ほど使ったバイアスと別のバイアス) を使い, 中間層のノードを伝っていく. そして出力層で, 得られた結果をソフトマックス関数で確率化する. $h()$ は活性化関数. $\sigma()$ はソフトマックス関数. 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.

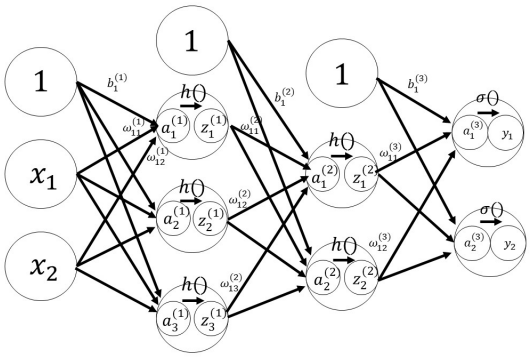

図 9　入力データ 2 個　出力データ 2 個のニューラルネットの動き

3　学習の流れ

3.1　学習

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. ニューラルネット, いわゆる深層学習は従来の機械学習よりも人の介入を避けることができるという重要な性質を持つ. データは先に説明した入力層に送られる. つまり機械学習においてデータはなくてはならないものである.

3.2　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. 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, その他の人にも使えなかったら実用性がないことを思い浮かべてみてほしい. また, 訓練データの学習をしすぎて訓練データにしか適用できない状態を過学習という. この過学習を避けることは機械学習においての重要な課題である.

3.3 損失関数

ニューラルネットワークではある一つの指標によって現在の状態を表す. 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.

3.4 2 乗和誤差と交差エントロピー誤差

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.

2 乗和誤差

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

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

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

$$[t_0, t_1, t_2, ...t_9] = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$$

とすると. 値は $E = 0..009750.....$ 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,

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

$E = 0.5975000....$ となり確かに損失関数の値はは高くなっている (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

$$E = -\sum_k t_k \log y_k \tag{14}$$

上の教師データと正解の確率が高いほうの出力結果を使うと $E = 0.51308...$ 正解の確率が低いほうの出力結果を使うと $E = 2.3025...$ となりこれまでの議論と一致する.

3.5 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 (正確には勾配) 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のか」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. これを表す式を下に記す.

$$\omega = \omega - \eta \frac{\partial L}{\partial \omega} \tag{15}$$

η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... と繰り返し (繰り返し続けしすぎると過学習となってしまう), 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.

3.6　学習アルゴリズムのまとめ

では一度どのように学習しているか整理してみよう. 基本的なステップは次のようになる.

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. (同じデータだけを使わないため)
重みの初期値をランダムに決める[a]

[a] ここでは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 (微分して変化の量を求める).

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.

$$\omega = \omega - \eta \frac{\partial L}{\partial \omega} \tag{16}$$

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. 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っている.

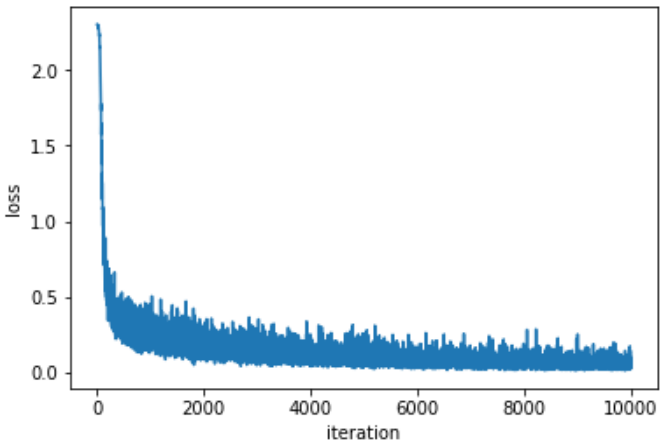

図 10　損失関数の推移

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) を表す図を下に乗せる. 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.

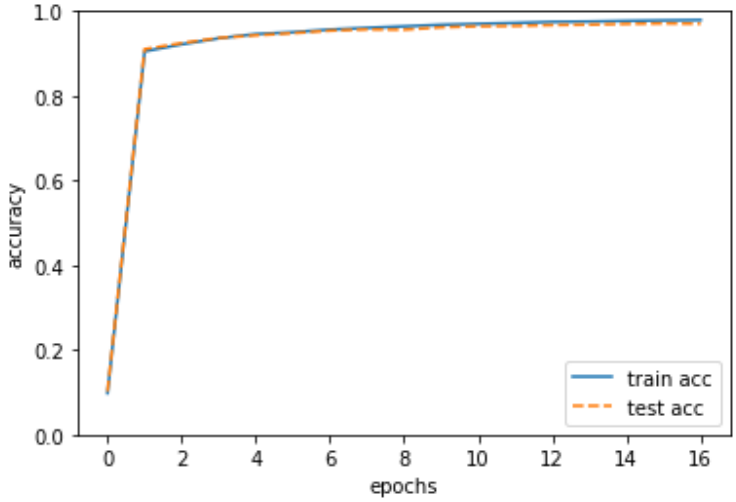

図 11　訓練データとテストデータに対する認識精度の推移

4　word2vec

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, 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 .

答えは私たちならすぐに分かるように「say」である. この場合, ???のコンテキストは「you」と「goodbye」の 2 単語である. このように周囲のコンテキストからその単語に何があるかを推論する問題を解かせ, 単語の出現パターンを学習させる, 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. その二つの紹介の前にまずは単語をどのように表現するかを説明する.

4.1　単語 ID で表現

ニューラルネットワーク上ではあくまで計算を行うので, 単語そのままを使うわけにはいかない. そのため単語を何かしらの数字に置き換えなければいけない. そのために使われるのが単語 ID と one-hot 表現である. 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.

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.

4.2　one-hot 表現データとノードの動き

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.

あとは, 先のニューラルネットワークの説明の通り, 重みを使い, 値を中間層に伝えていく.

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)

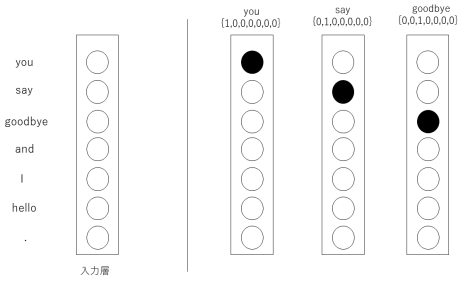

図 12 入力層に与えられるデータ

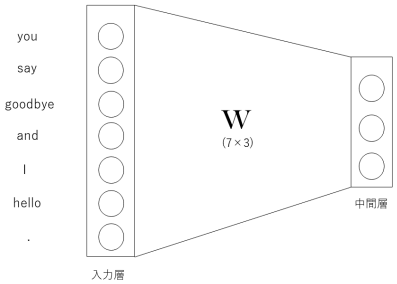

図 13 中間層に向か動き

4.3 CBOW モデルの推論処理

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). 先に説明した one-hot 表現で表したニューラルネットの図を使い, 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.

入力層が 2 つあり, 中間層を経て出力層にたどり着く. ここで中間層に注目する. この状態だと, 二つの計算結果をノードに入れてしまうことになる. そのため, 中間層では二つの計算結果を足して $\frac{1}{2}$ をする.

4.4 CBOW モデルの学習

4.4.1 ソフトマックス関数の導入

CBOW モデルでは入力層で 2 つのデータを使い, 中間層ではその結果を $\frac{1}{2}$ をし, 出力層で結果を出す. この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.

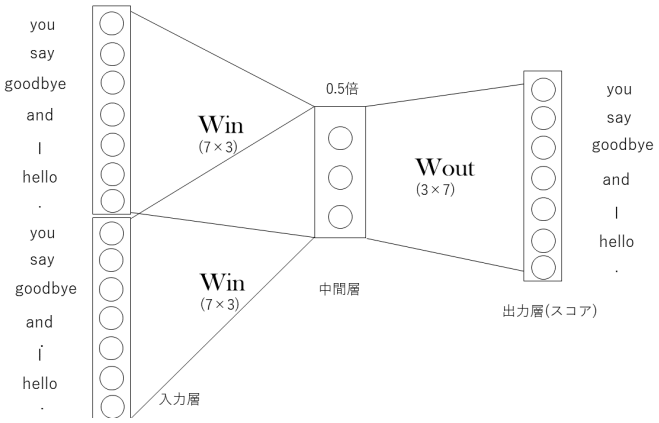

図 14 CBOW モデルの動き 1

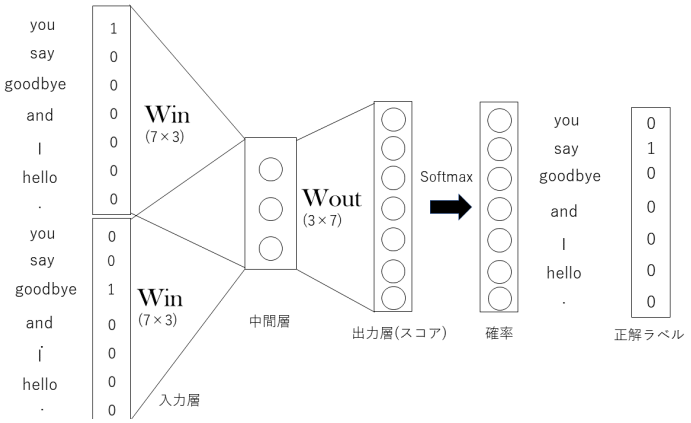

図 15 CBOW モデルの動き 2

4.4.2 CBOW モデルと確率

先のニューラルネットワークの構造について説明した通り, 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ようと思う. CBOW モデルでは, コンテキストが与えられたときにその中央の単語の推測を行う. そのため, t 番目の単語を求めるとして, 単語の列を ω_1, ω_2, ...ω_{t-1}, ω_t, ω_{t+1}, ...ω_T のようにすると, 事後確率[*1]を使って次のように表せる.

$$P(\omega_t \mid \omega_{t-1}, \omega_{t+1}) \tag{17}$$

これは ω_{t-1}, ω_{t+1} という情報が与えられた時, ω_t が起こる確率という風に解釈することが出来る. また, 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.

[*1] P(A|B) のように表す. 文字通り, 事が起こった後の確率であり, この場合「B という事象 (情報) を得た時, A が起こる確率を表している.

$$
\begin{aligned}
L &= -\sum_k t_k \log y_k \\
&= -(t_0 \log y_0 + t_1 \log y_1 + \cdots + t_t \log y_t + \cdots + t_k \log y_k) \\
&= -(0 \log y_0 + 0 \log y_1 + \cdots + \log y_t + \cdots + 0 \log y_k) \\
&= -\log y_t \\
&= -\log P(\omega_t \mid \omega_{t-1}, \omega_{t+1})
\end{aligned} \tag{18}
$$

つまり CBOW モデルの損失関数は, 単に式 (12) の確率に log をとり, それにマイナスをかけた簡単な式に表せる. 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, 以下のようになる.

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

4.5 skip-gram モデルの推論処理

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. CBOW モデルと同様に, one-hot 表現で表したニューラルネットを使い, コンテキストを 2 単語として下の図のように表せる.

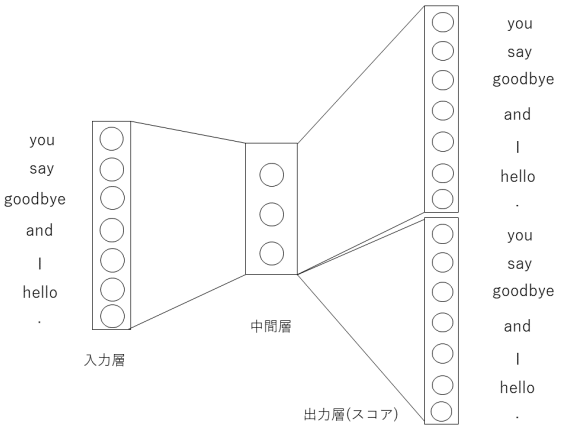

図 16 skip-gram モデルの動き 1

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.

4.6 skip-gram モデルの学習

CBOW モデルと同様に, 出力層でソフトマックス関数を使い, 確率化された出力結果を出す. 下の図の場合, 入力層でターゲットの「say」が与えられ, 教師ラベルがコンテキストの「you」と「goodbye」の 2 単語となっている.

4.6.1 skip-gram モデルと確率

CBOW モデルと同様に, 単語の列を ω_1, ω_2, ...ω_{t-1}, ω_t, ω_{t+1}, ...ω_T のようにして, ターゲットの単語を ω_t, コンテキストをそれぞれ ω_{t-1}, ω_{t+1} とすると, 「ω_t が与えられたとき, ω_{t-1} と ω_{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.

$$
P(\omega_{t-1}, \omega_{t+1} \mid \omega_t) \tag{20}
$$

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.

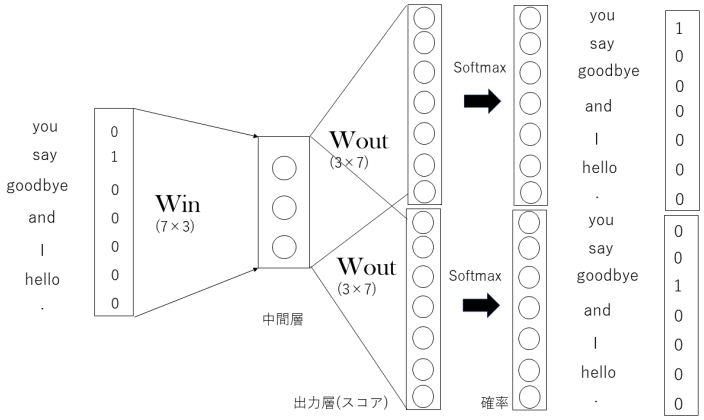

図 17 skip-gram モデルの動き 2

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

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.

$$\begin{aligned} L &= -\log P(\omega_{t-1}, \omega_{t+1} \mid \omega_t) && (22)\\ &= -\log P(\omega_{t-1} \mid \omega_t) P(\omega_{t+1} \mid \omega_t) && (23)\\ &= -(\log P(\omega_{t-1} \mid \omega_t) + \log P(\omega_{t+1} \mid \omega_t)) && (24) \end{aligned}$$

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.

$$L = -\frac{1}{T} \sum_{t=1}^{T} (\log P(\omega_{t-1} \mid \omega_t) + \log P(\omega_{t+1} \mid \omega_t)) \tag{25}$$

4.7 CBOW モデルと skip-gram モデルの比較

学習した後の精度は, CBOW モデルよりも skip-gram モデルの方が高いことが知られている. 特に, 単語の数を増やすごとにその差は顕著に表れる. しかし, 学習速度に関しては skip-gram モデルよりも CBOW モデルの方が速いことが分かっている. skip-gram モデルの場合は, コンテキストの数だけ損失を求めるため, その分計算コストが多くなるためである.

4.8 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. 入力層から中間層の重み W_{in} と中間層から出力層の重み W_{out} の二つとも意味をとらえているが, 基本的に word 2 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.

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 $\boldsymbol{x} = (x_1, x_2, x_3, \cdots, x_n)$ と $\boldsymbol{y} = (y_1, y_2, y_3, \cdots, y_n)$ とすると, 次の式で類似度を求める.

$$similarity(\boldsymbol{x}, \boldsymbol{y}) = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\| \|\boldsymbol{y}\|} = \frac{x_1 y_1 + \cdots + x_n y_n}{\sqrt{x_1^2 + \cdots + x_n^2} \sqrt{y_1^2 + \cdots + y_n^2}} \tag{26}$$

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, 遠いが分かるということになる.

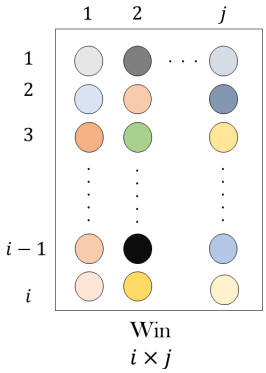

図 18　単語の分散表現

4.9　学習で得た単語の意味の類似性

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[*2]である. 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

[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

る結果と言える.

5　総括

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.

6　今後について

今回得た知識をもとに単語や文をデータとして集め, 実際にニューラルネットワークを用いて自然言語処理をしてみたい. データを集める際, 特定の人物やキャラクターの言葉を集中して学習させたら, その学習データを使って対話型の AI が作れるかどうか確認してみたい.

参考文献

[1] 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.

[2] 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.

[*2] 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, 色んな場所で使われている

[3] 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.