令和元年度後期 東京電機大学EC科3年「画像処理工学」期末レポート

提出者: 東京電機大学EC科3年 17ec084 平田智剛

第1章. はじめに

本書は、タイトルにも付した通り画像処理工学の期末レポートを目的としていることはもちろん、

それと同時に、「特別プログラミング演習」の期末テストの勉強も兼ねて執筆されたものである。

本書では第2章でmatlabによる簡単なニューラルネットワークを設計し、第3章でニューラルネットワークを用いた画像処理の数々を紹介する。

また、私は今後の研究で、マスコミの報道を定量的に評価する手法の開発や、司法判断の傾向(女性は刑が軽くなる、執行猶予が付きやすいなど)を客観的に明らかにできないかと検討している。

その上で機械学習は最適なツールであるといえる。このことを1-1および1-2節に示した。

それらのことを実現したく、本書を執筆した次第である。

1-1. 学問を侮辱する不自然科学

ハンガリー政府は同国内におけるジェンダー学の学士修士を廃止し、これを「ジェンダー学は科学ではなくイデオロギー」と説明した。[2]

これを「学問の自由の侵害だ」と危惧する声も大きいが、私個人の意見としてはこの決断は国民の権利を侵害するものではなく、むしろ多くの国民を護る物であったと評価したい。

ハンガリー政府の「科学ではなくイデオロギー」というコメントの真意を考えよう。小保方春子氏の騒動で国民が広く再確認することとなった通り、科学に不可欠な性質は「再現性」であり、これは「誰がどこで実験しても、同じような結果が得られる」というものだ。同様の説明は、魔法と科学の違いを議論する場合[3]にも行われる。

一方、イデオロギーはどうだろうか。イデオロギーは「特定の政治的立場に基づく考え」[4]等の意味を持つ言葉だ。

以上に基づいて考えると、科学とイデオロギーは客観と(主語の広い)主観という相反関係にある。

イデオロギーが不自然科学(社会カガク)として科学を騙っているような(一部の)文系学問の現状は、主観と客観を混同する詭弁を容認するものであると非難せざるを得ない。このような現状を隠蔽・温存し続け、イデオロギーの対象となった属性(ジェンダー学でいえば「女性」であることは、彼女ら自身の論理(男女比という安直すぎる考え)からも明らかだ)を贔屓するのが本当に「学問の自由」なのだろうか。私にはハンガリーの英断は、こうした彼女らの詭弁からイデオロギーの対象とならなかった男性たちを保護するものであったと思えてやまないし、これを学問の自由の侵害と叫ぶ人たちがどこか学問を侮辱しているようにすら思える。

1-2. データサイエンスによる、自然科学的分析

主観や個人的信念に直接に基づいて行われる研究・発見・拡散が一方的になされれば、それが他の人たちに多大な迷惑をかけることにつながりかねないのは明らかだ。広い視点で見れば一方的な差別とは言えないような事柄を、狭い視野で差別と言い張って撤廃させれば、全体としては別の人たちが新たに差別を受けることになってしまう。典型的な例は医大の入試で女子を減点していた事件が挙げられる。当事者にとっては明らかな女性差別であることに疑いの余地はないが、報道がこの事件ばかりを女性差別と大々的に取り上げ、けれども大阪電気通信大学で以前まで行われていた女子加点入試についてはほと

んど触れないのは異常なことだ。名古屋工業大学には女子推薦枠というものまである。女子に不利な入試はなくなり、女子に有利な入試は依然残る。皮肉なことにも、女性差別に関する報道そのものが男性差別につながりかねないのである。

以上のことから、報道がなるべく本当に近い意味で中立であるためには、客観的で(過度な感情論に基づかず論理的思考の働いている)誰もが納得せざるを得ないような―即ち(自然)科学的な手法による、報道の評価や監視が必要だ。

人工知能による報道の予測や判決の予想の正解率が十分に上がり、かつそのような人工知能に対して「説明可能AI」の技術を施すことができれば、報道の偏りを定量的に指摘し、判決の傾向を客観的に示すことができるだろう。

私の研究に対する意欲の根源はここにある。

1-3. 本書について

参考:[9]

matlabのコードを解説するにあたって、ここではいくつかの本書の「立場」を明確にしておこうと思う。但しNとMは0または自然数である。

- 0次元配列(1行1列)のことをスカラと呼ぶ
- 1次元配列(N行1列)のことを列ベクトルと呼ぶ
- 2次元配列(N行M列)のことを行列と呼ぶ
- N次元配列のことを配列と呼ぶ
- 行列のうち、行数が1であるものを行ベクトルと呼ぶ
- 行列のうち、行数と列数が共に0であるものを空と呼ぶ
- 列ベクトルや行ベクトルのことをベクトルと呼ぶ

例えば行ベクトルは「1行ベクトル」で1が省略されたものと考えることができるし、列ベクトルについても「1列ベクトル」ということができる。

同じように、「行方向」といったら、1行の向いている方向であるし、「列方向」といったら1列の向いている方向である。

なお、以上のことは。数字以外のデータを要素としている配列についても同様である。

したがって、任意の行列において、各列の要素の集まりを列ベクトル1つにまとめ、唯一の要素とすることで、「列ベクトルを要素とする行ベクトル」(ベクトルのベクトル)に変換することが可能だ。

また、本書では「パーセプトロン」、「多層パーセプトロン」、「ニューラルネットワーク」を次のように区別している。N,M,Lは自然数。

- パーセプトロンとは、(1層)N入力1出力の、次式で表される関数である

$$\text{出力} = \text{活性化関数}\left(\left(1, \overrightarrow{\text{入力}}\right) \cdot \left(\text{バイアス}, \overrightarrow{\text{重み}}\right)\right)$$

- 多層パーセプトロンとは、パーセプトロンをM台用意することで(1層)N入力M出力を実現したものや、その出力をさらに別の多層パーセプトロン(M入力L出力)に入力する(...を繰り返す)ことで2層以上を実現したものである。
- ニューラルネットワークとは、多層パーセプトロンに学習能力を与えて分類問題や回帰問題を解答する仕組みである。

第**2**章. ニューラルネットワークの設計

2-1. パーセプトロン

$$y = ax + b$$

はパラメータaとbを調節することで、xy平面上の任意の直線を表現できる。

パーセプトロンは、そういった直線より上だったら1、その直線より下だったら0を返却する・・・※。

これにより、パーセプトロンは分類やら予測やらを示すことができる(場合がある)。

「ある直線に対する上下関係を返すだけの関数が、実はそれだけで分類や予測する力を帯びている」ことを示す例を挙げよう。図2.1.1のような調査結果があったとしよう。直線はマスコミが報道するものとしないものとを分類しているといえる。

※のことをするパーセプトロンは、マスコミが報道するような事例に対して1を、報道しないような事例に対して0を返す。これが、「パーセプトロンがマスコミのとる行動を分類、予測する」例である。

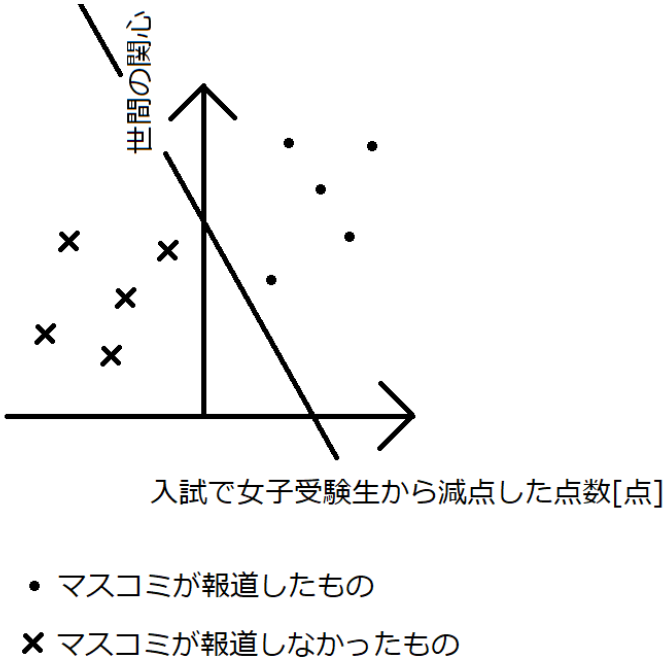


図2.1.1 報道に関するグラフ(出鱈目であることに注意)を分類する例

ニューラルネットワークによる機械学習は、パーセプトロンを何段にも重ねたものを使って「まずは嘘八百デタラメな」予測結果を出し、現実と見比べながらパラメータを再調整するのを何度も繰り返すことで、予測や分類の精度を高めている。

パーセプトロンを何段にも重ねたり、「再調整」を何度もやるというのに、「n段目のm番目のパラメータaは・・・」などと処理させるのは非効率だ。そこで、直線の式を変形して「内積」で表せるようにしよう。そうすれば、「いろいろな内積を考えなければいけない」という問題が「行列の積を計算する」ことに置き換わる。(このことは2-2節で説明している。)

$$y = ax + b$$

を変形すると、

$$y - ax - b = 0$$

$$cy - acx - bc = 0$$

$$-bc + cy - acx = 0$$

yやxをx_1やx_2と書き換え、また定数をw_iで置きなおすと

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

と書くことができる。

ここでバイアス成分として$x_0 = 1$とおくと

$$w_0 x_0 + w_1 x_1 + w_2 x_2 = 0$$

となり、これは次のようにベクトルの内積で書くことができる。

$$\begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = 0$$

今後、$\boldsymbol{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}$を重みベクトル、$\boldsymbol{x} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix}$を入力ベクトルと呼ぶこととする。

ベクトルや行列の演算はmatlabの得意とするところだ。

上記の計算は次のコードで実行できる。

```
x_0=1;
syms w_0 w_1 w_2 x_1 x_2 real;%最後の「real」により実数と仮定
dot([w_0 w_1 w_2], [x_0 x_1 x_2])
```

$$\text{ans} = w_0 + w_1 x_1 + w_2 x_2$$

※のことは、この内積の正負により判定できる。

すなわち、内積が正なら1を、負なら0を返却するようにコードを書けばよい。

しかし、そのままif文で書いてしまうと、のちのち「学習」するのに都合が悪い。

というのも、if文で場合分けなどしようものなら数学的に「単純」な式で表せないではないか。

機械学習の「学習」は、数学的な理論に裏付けられているので、プログラムが簡単であることよりも数学的に簡単である方がよい(その後のプログラミングが楽になる等)場合がある。今回がそれだ。

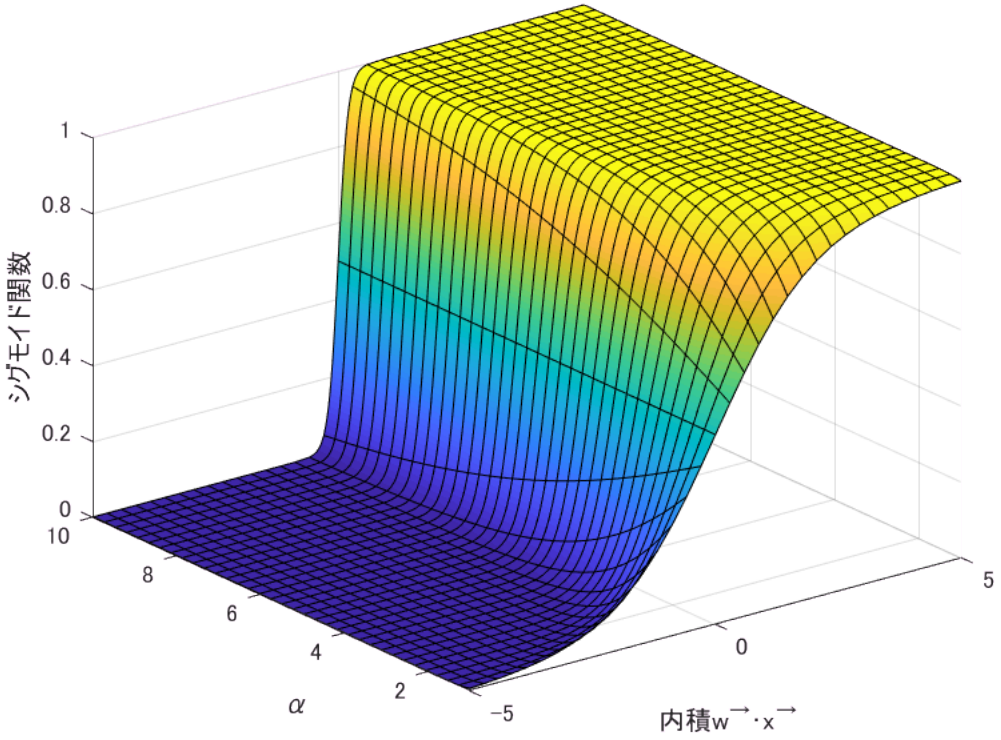
そこで登場するのが「シグモイド関数」だ。

シグモイド関数は次の式で表される。

$$\sigma(x) = \frac{1}{1 + e^{-\alpha x}}$$

これをxとαの2変数関数とみてプロットするコードを用意した。

```
syms x alpha;
xrange=[-5,5];
alpha_range=[1 10];
fsurf((1+exp(-alpha*x))^-1,[xrange alpha_range]);
xlabel('内積w^→・x^→');
ylabel('α');
zlabel('シグモイド関数');
```



結果から明らかなように、確かに内積が負ならシグモイド関数は0,正なら1に向かっていることが確認できるが、

ある程度0に近い(絶対値が小さい)とシグモイド関数は0.5など、0と1の中間的な値をとることが分かる。

そして、この「中間的な値」へのなりにくさを決めるのがαである。

αが小さいほど中間的な値をとりやすく、逆に大きいほど、0や1へ近づきやすい。

今回は$\alpha = 1$として設計していこう。

よって、パーセプトロンを数式で表すなら、次のようになる。

$$\mathrm{Perceptron}(\boldsymbol{w}, \boldsymbol{x}) = \frac{1}{1 + e^{-w \cdot x}}$$

2-2. 行列の積と内積

行列の積$\boldsymbol{AB}$のi行j列目は次の式で計算できる。[1]

$$\sum_{k=1}^{\text{Aの列数つまりBの行数}} \boldsymbol{A}(\text{i行k列目})\boldsymbol{B}(\text{k行j列目})$$

$\boldsymbol{A}$のi行目全体や$\boldsymbol{B}$のj行目全体は、ベクトルである。

1行n列のあるいはn行1列の行列はn次元ベクトルであるからだ。

そしてそういった目でみたとき、この式は「$\boldsymbol{A}$のi行目全体と、$\boldsymbol{B}$のj行目全体の内積」そのものであるといえる。

したがって、行列の積とは内積の集まりであるということができるのだ。

2-3. パーセプトロンの限界

2-1節も述べたように、ニューラルネットワークは「パーセプトロンたちの出力を、別のパーセプトロンたちに入力する」ことを繰り返すことで構成されている。

パーセプトロンを一つ動かしたところで機械学習を行うことは、一般的には残念ながらできない。

理由は簡単で、「曲線を使わなければどう頑張っても分類できない」という事態に陥ったとき、パーセプトロン1人ではどうすることも出来ないのだ。

例えば図2.3.1のように○と×があるとき、これを直線で分類するのは不可能だ。

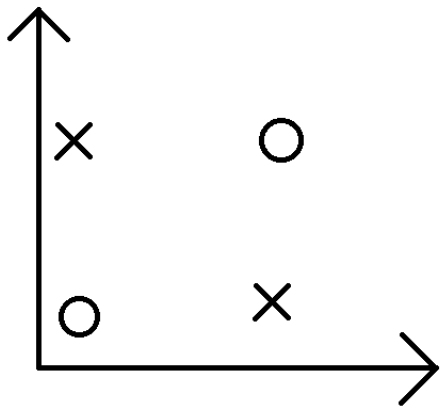


図2.3.1 パーセプトロン1つでは分類できないデータ

目で見てみても、試してみても、図2.3.1を直線で分類できないのは明らかだ。

ただ、「明らかだ」と無根拠に言い放って終わるようでは自然科学を学ぶ学生の名が廃る。

図2.3.1がなぜ直線で分類できないか、直観的で厳密性に欠けるものではあるが、説明を試みる。

まず、分類後の「×」の領域の面積が最小になるように直線を引いてみる。(赤)

次に、「○」に対しても同様な直線を引いてみる。(青)

すると、図2.3.2のようになる。

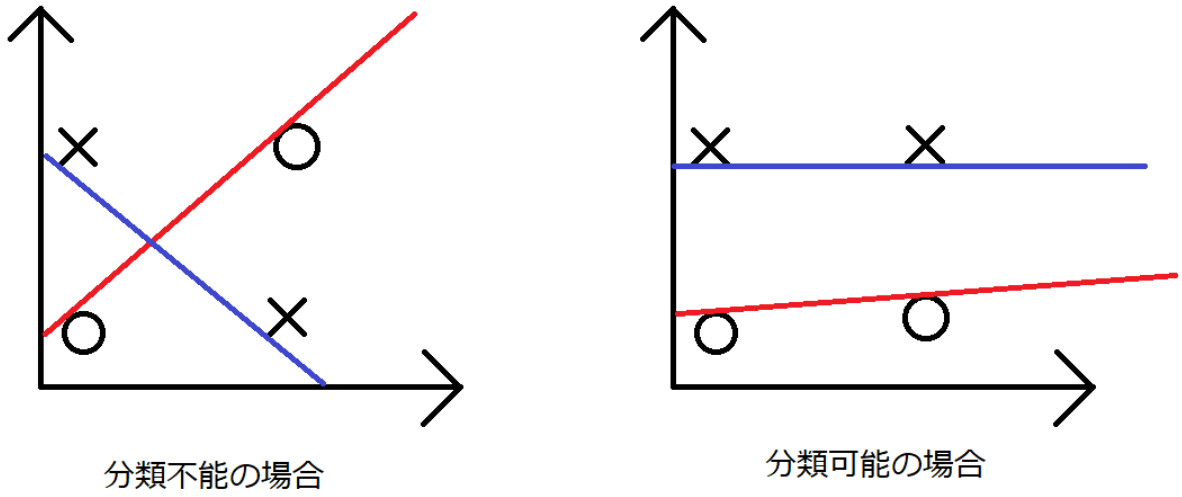


図2.3.2 各領域の面積を最小にする直線

比較すると、分類可能な場合は青線と赤線が(必要な範囲内では)交わらないのに対し、分類不能の場合は交わってしまっている。

分類可能の場合を見てほしい。2線は各領域の面積を最小にする「両極端な」直線である。

各領域の面積をこれ以上小さくできないので、「実際の直線」は青線より上に行けないし、赤線より下にいけない。

したがって、青線と赤線の間のどこかに「実際の直線」が定まるはずだ。

一方分類不能の場合については、もともと2線が交わっているため、3次元目の方向が存在しでもしない限り「2線の間」という場所が存在しない。

したがって、分類のための直線を引くことができないのだ。

2-4. 多層パーセプトロンによる限界突破

(参考:[5])

2-3節を読んで、「論理積が使えれば」とは思わないだろうか。

実は、使える。論理積自体はパーセプトロンで難なく表現できるものだからだ。

パーセプトロン1号が図2.4.1の①の直線より下を1、上を0とするとしよう。(「下を1、上を0とする」のは「上を1、下を0とする」場合の内積に-1をかけるだけで実現できる)(要はNANDを出力)

パーセプトロン2号が図2.4.1の②の直線より上を①、下を0とするとしよう。(要はORを出力)

そしてパーセプトロン3号は1号と2号の出力の論理積を出力するとしよう。③の直線より上を1、下を0とすればよい。

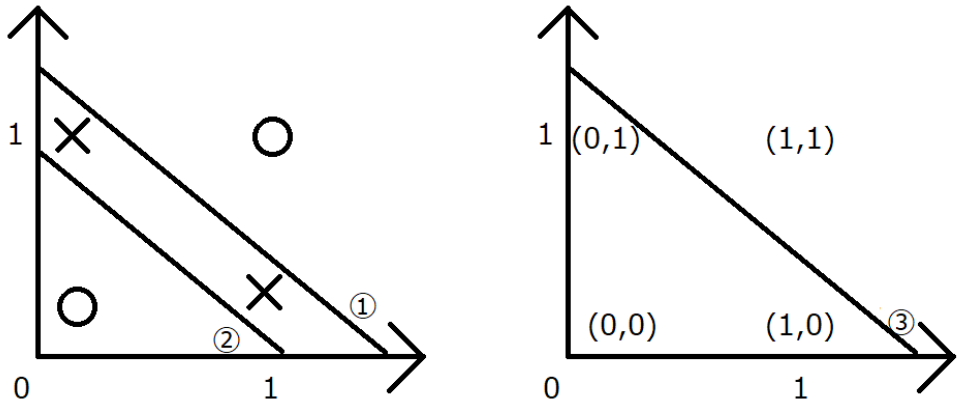


図2.4.1 3つのパーセプトロン

すると、パーセプトロン3号の出力は、直線①と②の間なら1、外なら0となり、○と×を分類できたことになる。

×なら1、○なら0を出力する。要はXORだ。

以上のことを論理式で書けば、かの有名な $A \,\mathrm{XOR}\, B = (A \,\mathrm{NAND}\, B)\mathrm{AND}(A \,\mathrm{OR}\, B)$ が導かれる。

(これからは3つのパーセプトロンの動作を思い出せば、もうXORの公式を覚える必要がないわけだ。)

また、3つのパーセプトロンは2段構成になっている。

1段目に1号と2号があって、外部から入力を受け取り2段目へと出力する。

2段目に3号があり、1段目の出力を受け取って外部へと出力している。

これを式にすると、2段目である3号が「外側」、1段目である1号、2号が「内側」のパーセプトロンとして表れる。

$$\mathrm{Perceptron}\left(\boldsymbol{w}_{\mathbf{AND}}, \begin{pmatrix} 1 \\ \mathrm{Perceptron}(\boldsymbol{w}_{\mathbf{NAND}}, \boldsymbol{x}_1) \\ \mathrm{Perceptron}(\boldsymbol{w}_{\mathbf{OR}}, \boldsymbol{x}_1) \end{pmatrix}\right) = \frac{1}{1 + e^{-\boldsymbol{w}_{\mathbf{AND}} \cdot \begin{pmatrix} 1 \\ \frac{1}{1+e^{-\boldsymbol{w}_{\mathbf{NAND}} \cdot \boldsymbol{x}_1}} \\ \frac{1}{1+e^{-\boldsymbol{w}_{\mathbf{OR}} \cdot \boldsymbol{x}_1}} \end{pmatrix}}}$$

・・・①

なお、参考サイト[5]では、パーセプトロンを多層化することにより、「3次元目の方向」を作り出しているようであると主張している。この主張は式でこそ裏付けられてはいないが、直観的にはとてもわかりやすく、また根拠(何が3次元目に当たるのか)も示されているため、「直接的な証拠がない」と片付けるにはもったいない。

そこで、[5]の主張を式で説明することを考えた。

本書による多層パーセプトロンの説明は2次元グラフ2つで行った(図2.4.1)のに対し、[5]の説明では3次元目の方向を考えることにより、3次元グラフ1つでの説明がなされている。また3次元目の方向の入力は、[5]の以下の記述より、

$$\boldsymbol{w'}_{\mathbf{AND}} \bullet \begin{pmatrix} \mathrm{Perceptron}\left(\boldsymbol{w}_{\mathbf{NAND}}, \boldsymbol{x}_1\right) \\ \mathrm{Perceptron}\left(\boldsymbol{w}_{\mathbf{OR}}, \boldsymbol{x}_1\right) \end{pmatrix}$$

であるといえる。但し$\boldsymbol{w'}_{\mathbf{AND}}$は$\boldsymbol{w}_{\mathbf{AND}}$の第1成分、すなわちバイアス成分となる物を消去したものである。

OR回路、NAND回路の入力はともに重み*w=1*なので、右のグラフ中では赤：１，青：１、灰：２となります。この値を高さと見れば、グラフの軸が３次元方向にもう１つ追加された感じでしょうか。

そして

高さが*1.5*以上であるのは灰部分のみであるので、その領域にある黒点がヒットし、XOR回路が表現されるという仕組みです

とあることより、3次元グラフを分類するとされるパーセプトロンは、平面(3次元目の方向=1.5)との上下関係で分類されるから、

$$\mathrm{Perceptron}\left(\begin{pmatrix} -1.5 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ \mathrm{Perceptron}\left(\boldsymbol{w}_{\mathbf{NAND}}, \boldsymbol{x}_1\right) \\ \mathrm{Perceptron}\left(\boldsymbol{w}_{\mathbf{OR}}, \boldsymbol{x}_1\right) \\ \boldsymbol{w'}_{\mathbf{AND}} \cdot \begin{pmatrix} \mathrm{Perceptron}\left(\boldsymbol{w}_{\mathbf{NAND}}, \boldsymbol{x}_1\right) \\ \mathrm{Perceptron}\left(\boldsymbol{w}_{\mathbf{OR}}, \boldsymbol{x}_1\right) \end{pmatrix} \end{pmatrix}\right) = \frac{1}{1 + e^{-\begin{pmatrix} -1.5 \\ 0 \\ 0 \\ 1 \end{pmatrix} \bullet \begin{pmatrix} 1 \\ \frac{1}{1+e^{-\boldsymbol{w}_{\mathbf{NAND}} \bullet \boldsymbol{x}_1}} \\ \frac{1}{1+e^{-\boldsymbol{w}_{\mathbf{OR}} \bullet \boldsymbol{x}_1}} \\ \frac{w_{\mathrm{AND},1}}{1+e^{-\boldsymbol{w}_{\mathbf{NAND}} \bullet \boldsymbol{x}_1}} + \frac{w_{\mathrm{AND},2}}{1+e^{-\boldsymbol{w}_{\mathbf{OR}}}} \end{pmatrix}}}$$

但し

$$\boldsymbol{w}_{\mathbf{AND}} = \begin{pmatrix} -1.5 \\ \boldsymbol{w'}_{\mathbf{AND}} \end{pmatrix} = \begin{pmatrix} -1.5 \\ w_{\mathrm{AND},1} \\ w_{\mathrm{AND},2} \end{pmatrix}$$

・・・②

②を計算すると

$$\frac{1}{1 + e^{1.5 - \left(\frac{w_{\mathrm{AND},1}}{1+e^{-\boldsymbol{w}_{\mathbf{NAND}} \bullet \boldsymbol{x}_1}} + \frac{w_{\mathrm{AND},2}}{1+e^{-\boldsymbol{w}_{\mathbf{OR}} \bullet \boldsymbol{x}_1}} \right)}}$$

また①式も計算すると

$$\frac{1}{1+e^{-w_{AND}\cdot\begin{pmatrix}\frac{1}{1+e^{-w_{NAND}\cdot x_1}}\\ \frac{1}{1+e^{-w_{OR}\cdot x_1}}\end{pmatrix}}}=\frac{1}{1+e^{-\begin{pmatrix}-1.5\\ w_{AND,1}\\ w_{AND,2}\end{pmatrix}\cdot\begin{pmatrix}1\\ \frac{1}{1+e^{-w_{NAND}\cdot x_1}}\\ \frac{1}{1+e^{-w_{OR}\cdot x_1}}\end{pmatrix}}}=\frac{1}{1+e^{1.5-\left(\frac{w_{AND,1}}{1+e^{-w_{NAND}\bullet x_1}}+\frac{w_{AND,2}}{1+e^{-w_{OR}\bullet x_1}}\right)}}$$

となることから、①と②は同じだ。

したがって、

$$\mathrm{Perceptron}\left(w_{AND},\begin{pmatrix}1\\ \mathrm{Perceptron}(w_{NAND},x_1)\\ \mathrm{Perceptron}(w_{OR},x_1)\end{pmatrix}\right)=\mathrm{Perceptron}\left(\begin{pmatrix}-1.5\\ 0\\ 0\\ 1\end{pmatrix},\begin{pmatrix}1\\ \mathrm{Perceptron}(w_{NAND},\\ \mathrm{Perceptron}(w_{OR},x\\ w'_{AND}\cdot\begin{pmatrix}\mathrm{Perceptron}(w_{NA}\\ \mathrm{Perceptron}(w_{O}\end{pmatrix}\end{pmatrix}\right)$$

と書くことができ、「多層パーセプトロンは、グラフの次元を一つ追加するものである」ことが式により証明されたのである。

今、図2.3.2を見てほしい。分類不能なグラフでも、奥行方向に次元を一つ増やせば、その方向に「青線と赤線の間」が生じる。即ち分類が可能になるではないか。

以上のことにより、2層のパーセプトロンがあれば、直線で分類できなくても、いろいろな分類方法を学習することができるということが示された。

2-5. 行列

2-2節で示した通り、パーセプトロンの処理(活性化関数は除く)が内積で書けるということは、パーセプトロンの集まりの処理が行列で書けることを示唆している。

まず、パーセプトロンを絵にすると、図2.5.1のようになる。

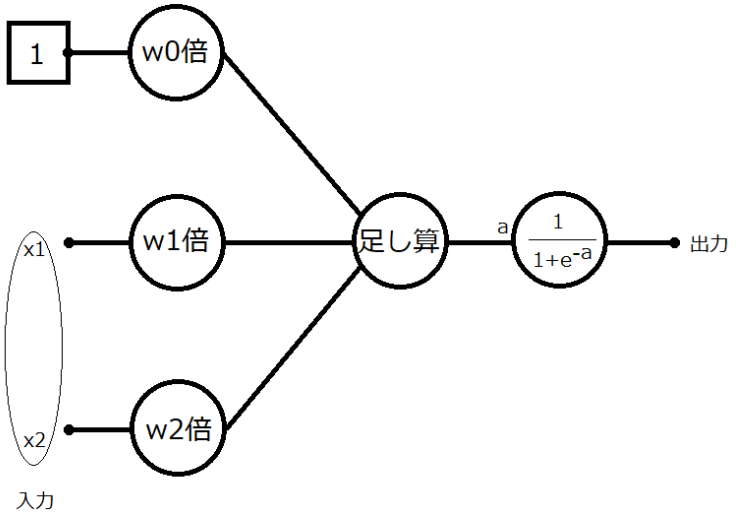


図2.5.1 パーセプトロン

パーセプトロンを図2.5.2のように2つ並べると、2つの出力を得ることができる。

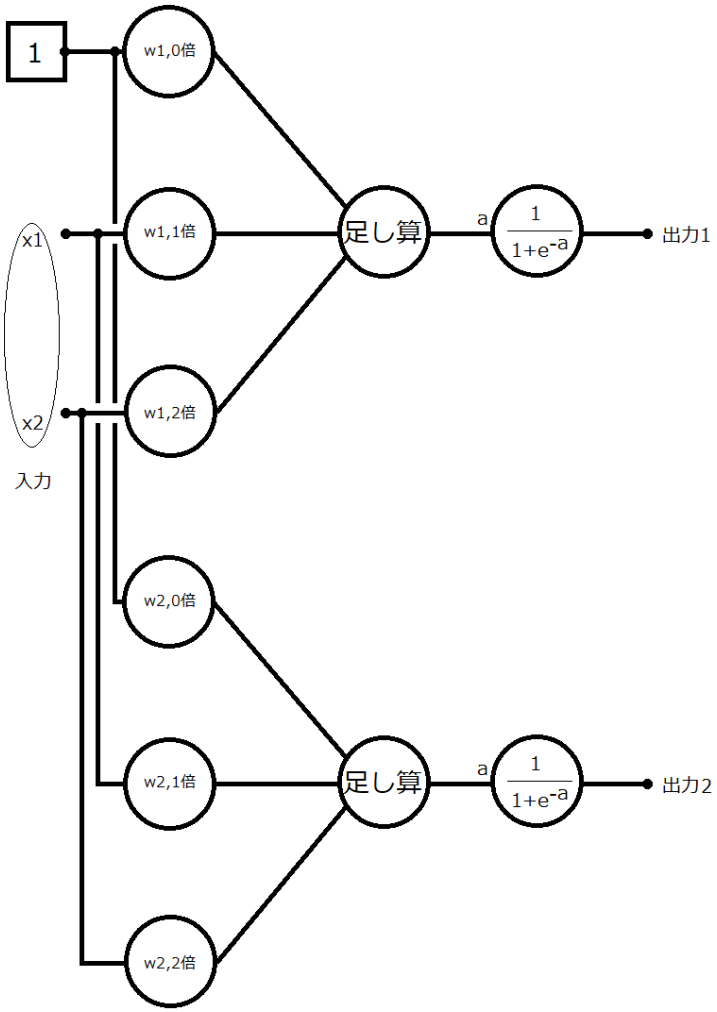


図2.5.2 入力が共通なパーセプトロンを2つ並べたもの

(第1層の)パーセプトロン2つの出力をまた別の(第2層の)パーセプトロンに入力するようにすると、図2.5.3のようになる。

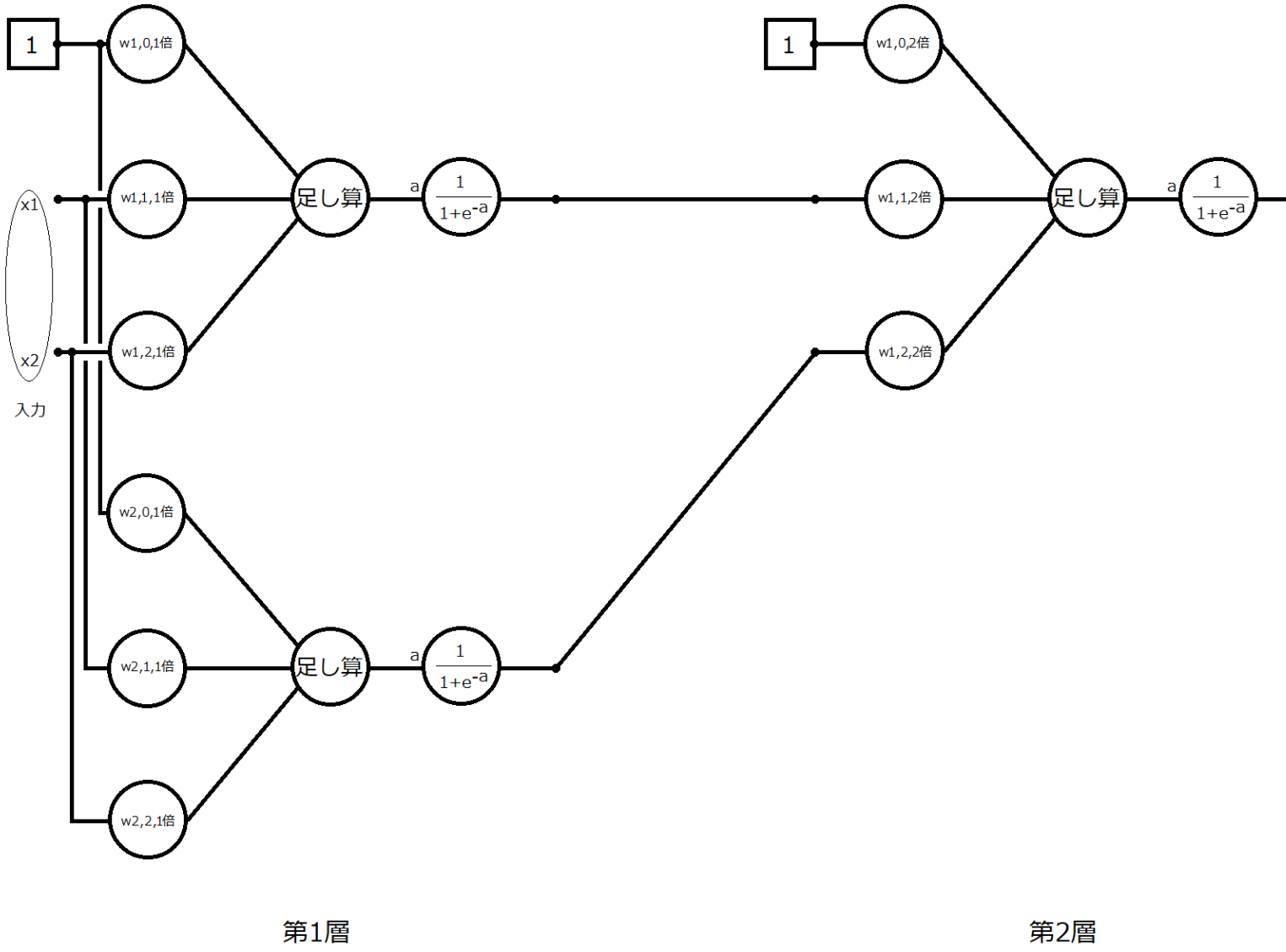


図2.5.3 2層 パーセプトロン

また重みを表す変数については

$$W_{\text{出力先パーセプトロンの番号,入力の番号,層の番号}}$$

という形で表示している。

層の番号が同じ重みを、行列でまとめて表現してみよう。

すなわち$W_{\text{任意の出力先パーセプトロンの番号,任意の入力の番号,ある層の番号}}$の集まりを$\boldsymbol{W}_{\text{ある層の番号}}$という「重み行列」で表現する。

今

$$\boldsymbol{W_1} = \begin{pmatrix} W_{1,0,1} & W_{2,0,1} \\ W_{1,1,1} & W_{2,1,1} \\ W_{1,2,1} & W_{2,2,1} \end{pmatrix}$$

を考えよう。

シグモイド関数への入力aは、いままで内積を使って

$$\begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix} \cdot \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}$$

のように表現されてきたが、これをまとめて

$$\begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix} \begin{pmatrix} w_{1,0,1} & w_{2,0,1} \\ w_{1,1,1} & w_{2,1,1} \\ w_{1,2,1} & w_{2,2,1} \end{pmatrix}$$

と表現でき...ない。

その理由を今から説明する。

行列の積は可換律が成り立たず、必ずかけられる行列の列数と、かける行列の行数が一致しなくてはならない。･･･①

かけられる行列の列数と、かける行列の行数が一致する場合、積は、かけられる行列の行数を持ち、かける行列の列数を持つ。･･･②

内積

$$\begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix} \cdot \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}$$

を行列の積で表現するときは、①の都合により

$$\begin{pmatrix} 1 & x_1 & x_2 \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ w_2 \end{pmatrix}$$

となる。そしてこの計算結果は②より、行数1、列数1の行列、すなわちスカラとなるわけだ。

以上が、先ほどの積が誤っている理由である。

実際にはシグモイド関数への入力は

$$\begin{pmatrix} 1 & x_1 & x_2 \end{pmatrix} \begin{pmatrix} w_{1,0,1} & w_{2,0,1} \\ w_{1,1,1} & w_{2,1,1} \\ w_{1,2,1} & w_{2,2,1} \end{pmatrix} = \begin{pmatrix} \text{パーセプトロン1における} a & \text{パーセプトロン2における} a \end{pmatrix}$$

とまとめて表現される。

②よりこれは1行2列の行列、すなわち行ベクトルとなる。

2-6. オブジェクト指向

注意:2-6節は著者のオブジェクト指向プログラミングに対する持論が中心に執筆されている。オブジェクト指向プログラミングの登場した背景や目的をまとめた内容ではないため、「正しい」オブジェクト指向プログラミング(privateメソッドが忌避されるなど)の説明は良書に譲る。

パーセプトロンや多層パーセプトロン、そして学習能力のあるニューラルネットワークをmatlabで実装するときは、オブジェクト指向に基づいたプログラミングをすると効率が良い。

クラスを小さなプログラムとメモリであると考えるとき、オブジェクト指向プログラミングは、

「小さなプログラムをお互いに関わり合わせ、全体的に大きなプログラムを作る手法」ということを書いていくプログラミング手法であるといえるのだ。

そしてこの「小さなプログラム」と「大きなプログラム」の関係はまさに1つのパーセプトロンと多層パーセプトロンの関係そのものではないか。

したがって、パーセプトロンや多層パーセプトロン、ニューラルネットワークをプログラミングする際は、オブジェクト指向プログラミングで行われるべきであると、著者は考えるのである。

クラスがプログラム＋メモリであるという主張の詳しい説明は2-6-1節に示した。この考え方に異論がなければ2-6-1節は読み飛ばして差し支えない。

2-6-1. クラスは小さなプログラム＋メモリであり、インスタンスはコンピュータだ！

オブジェクト指向プログラミングとは、「クラス」の集まりとしてプログラムを書くことである。

またクラスとは「プログラムにメモリを持たせたもの」だ。[0]

即ちオブジェクト指向プログラミングとは、「小さなプログラムやメモリの集まりとして大きなプログラムを書くこと」といえる。

プログラムとは、「順次、反復、分岐」の処理を集めたものである[6]。

クラス内に書かれた関数(メソッドやコンストラクタなど)でこれらの処理を行うことが可能なので、メソッドはプログラムの一種といっても論理的には正しい。

そしてメソッドを通じてプロパティ(「フィールド」と呼ぶことも。プロパティとフィールドの違いは言語によって異なることが多い)というメモリの内容が書き換わる。

また、プログラムを実行する機能(メソッド)とメモリ(プロパティ)を持っているということは、クラスをコンピュータのように考えることもできる。

正確に言い直せば、クラスに基づいた(仮想的な)コンピュータを作ることができるわけだ。

クラス(プログラム)は概念であり、実際に動いてくれるのはインスタンス(コンピュータ)であるというわけだ。

このコンピュータのことをインスタンスという。そしてインスタンスには「型」という概念があり、これはどのクラスに基づいて設計されたコンピュータであるかを示す。

例えばCat型コンピュータdoraemonは、javaでは

```
Cat doraemon = new Cat(129.3);
```

のようなコードで作ることができる。

クラスが概念であるのに対し、インスタンスは実在する「物」である。

したがって、「クラスをプログラミングして、それに従ったインスタンスを生成する」というプログラミング手法を用いると、

「物がどんな機能(UMLでは「操作」)、状態(UMLでは「属性」)を持つか」に着目したプログラミングをしやすい。機能はメソッドで、状態はプロパティで書ける。

また、クラスは「他のクラスを使って」書くことができる。

例えば「人」クラスは、「哺乳類」クラスの一種であるとか、「手」クラスを持つ、等といったことを明示的に書くことができる。また、その関係性を書くと、

「人」クラスは「哺乳類クラス」のプログラムやメモリを引き継ぐ(これを「継承」という)し、内部で「手」のコンピュータを動かすことができるようになる。

このように、オブジェクト指向プログラミングでは、クラスという小さなプログラム同士を関わり合わせることで大きなプログラムを動かす手法であるといえる。

2-6-2. コードを簡潔にし、かつ高速化する方法

参考:[7]

ニューラルネットワークの教科書[-1]では、パーセプトロンや多層パーセプトロン(ニューラルネットワーク)を作るにあたって、オブジェクト指向を用いていなかった。柔軟性(後から入力数やパーセプトロンの数、活性化関数、層の数などを、(プログラムを書いた人以外でも、)如何に簡単に変更できるか)を重視せず、短いコードで素早くプログラミングを行いたい場合は、その方が適切なのかもしれない。

しかし、本書では、最終的には「たった1行程度」のコードで多種多様なニューラルネットワークを生成できるようにすることを目指している。

そのため、オブジェクト指向を使わないコードより、ほんの少しばかり長いコードを書くことになってしまうことをご了承いただきたい。

とはいえ、その中でもできるだけ短く、できるだけ高速で動くコードを書いたほうが良いということには間違いない。

matlab上でコードを書くときに意識すべきいくつかのことを、ここで整理しておく。

1. 「各成分に対する処理」はループ文を避け、なるべくベクトルや行列としてまとめて処理せよ(ベクトル化)
2. 機能は作るよりも探せ。matlab語よりも日本語を考えよ
3. プロファイラに頼れ

1は、本質的に2と同じことだ。例えば行列の積やベクトルの内積はfor文で実装することもできるが、そうはせずmatlabが標準で定義している関数dotや演算子「*」を使うべきである。というのも、matlabの欠点として、「ループ文が遅い」という点があるそうだからだ。

また、「配列内に重複する要素があったらそれを取り除く」とか、「行列で特定の値だけ変更する」などの、マニアックな操作に対しても、「もしかしたらそういう関数が標準であるのではないか」と考え、探すことを試みるべきだ。欲しい機能をmatlabで書くのではなく、まずは日本語で表現して、ネットやドキュメンテーションで検索し、見つかったらその関数を積極的に使うべきだ。

また、欲しい機能に(ある程度)近いものがあっても、より近い機能が標準で実装されていることがある。

例えばベクトルの長さを取得することを考えよう。

size関数で各次元の大きさを取得し、その総積をとることもできるが、length関数というものがあり、これは「最大の次元の大きさを返す」ことをしてくれる。この場合は後者を使う。

以上の工夫をすれば、コードは短くて済むし、matlabの開発者たちが書いた、「最適化されたアルゴリズム」にて処理を行えるから、高速にもなる。

2-6-3. matlabのいびつなオブジェクト指向

参考:[8]

matlabを開いて「新規作成」から「クラス」を選ぶと、クラスを作成することができる。

次のようなクラスを書き、test.mとして保存しよう。

(javaなどに慣れている人が見れば、各メソッドの第1引数にあるthisが気になるだろう。

matlabのオブジェクト指向では、コンストラクタ以外では「第1仮引数に自分自身のオブジェクトを示す」ことが必要となっている。

しかも、実引数(呼び出すとき)ではthisに当たるものが不要なので、さらにややこしい。)

```
classdef test
properties
property;
end

methods
function this=test()
end

function this=set.property(this, property)%セッタ
disp("set.propertyが呼ばれました″);
this.property=property;
end

function a=get.property(this)%ゲッタ
disp("get.propertyが呼ばれました″);
a=this.property;
end
end
end
```

保存したら、コマンドウィンドウに次のように入力する。

```
t=test;
t.property=6;
```

```
set.propertyが呼ばれました
```

```
t.property
```

```
get.propertyが呼ばれました
ans = 6
```

結果から分かる通り、matlabではプロパティにアクセスするとセッタやゲッタが自動で呼び出される。

但し、propertiesの行で右に(Access = private)を追加すると、外部からpropertyというプロパティへアクセスできなくなる。

そのため、propertyを書き換える術がなくなる。

プロパティを(Access = private)にしたものをtest_.mというファイル名で保存し、次のコードを実行してみると、エラーになる。

```
t=test_;
t.property=6;
```

```
test_ の 'property' プロパティを設定できません。
```

```
t.property
```

「勝手にいじられるとプログラムが壊れてしまう」というようなプロパティに対して、このような設定をするのであるが、実際には「基本的にすべてのプロパティを(Access = private)にする」と考えてよい。このような方針のことを「カプセル化」という。

2-6-1節に示した通り、プロパティとはコンピュータで言えば「メモリ」や「状態」に当たるものであるから、外部(コマンドウィンドウのことと考えよう)に対して野晒しにすべきものではない。そのため「カプセル」の中に入れて、気安くべたべた触れなくしてしまおうというわけだ。

但し、「外部からプロパティを利用できない」ようでは、プロパティの存在する意味がない。そこで、セッタやゲッタによって、「プロパティに対するどのような触り方を許すか」を定義する。こうすることで、外部は設計者の意図通りにしかプロパティをいじれなくなるため、安全性が増すわけだ。

これもまたカプセル化の一種だ。外部は、カプセルの中に閉じ込められたプロパティをセッタやゲッタを使って操作できる。これはカプセルにキーボード、マウスやモニタなどの周辺機器を付けるようなものだ。利用者はコンピュータのメモリを直接見たり書き換えたりできないが、キーボード、マウスやモニタのおかげでそのコンピュータを「利用」することだけはできる。

以上がオブジェクト指向プログラミングの思想であるが、matlabでは「プロパティにアクセスすると、暗黙的にセッタやゲッタが呼ばれる」という仕組みがとられている。その上ゲッタやセッタを直接呼び出すことができない仕様なので、privateなプロパティへ外部からアクセスすることが全くできない。

そこで、「疑似的なセッタやゲッタを作ることで、内部からアクセスする」という方法が考えられる。

疑似的なセッタやゲッタ内でプロパティへアクセスすれば、これは内部からのアクセスであるから、本物のセッタやゲッタが連動するだろう。

これは、次のようなコードで確かめることができる。

```
classdef test2
properties (Access = private)
property;
end

methods
function obj=test2()
end

function this=set.property(this, property)
disp("set.propertyが呼ばれました″);
this.property=property;
end

function property=get.property(this)
disp("get.propertyが呼ばれました″);
property=this.property;
end

function this=setProp(this, property)
disp("setPropが呼ばれました″);
this.property=property;
end

function property=getProp(this)
disp("getPropが呼ばれました″);
property=this.property;
end
end
end
```

```
t=test2;
disp("t.setProp(9)ではうまくいかない″);
```

```
t.setProp(9)ではうまくいかない
```

```
t.setProp(9);
```

```
setPropが呼ばれました
set.propertyが呼ばれました
```

```
t.getProp()
```

```
getPropが呼ばれました
get.propertyが呼ばれました
ans =

     []
```

```
disp("  ");
```

```
disp("t=t.setProp(9)とするとうまくいく″);
```

```
t=t.setProp(9)とするとうまくいく
```

```
t=t.setProp(9);
```

```
setPropが呼ばれました
set.propertyが呼ばれました
```

```
t.getProp()
```

```
getPropが呼ばれました
get.propertyが呼ばれました
ans = 9
```

また、(疑似でなくて本物の)セッタやゲッタは必須ではなく、省略することができる。

```
classdef test2_
properties (Access = private)
property;
end

methods
function obj=test2_()
end

function this=setProp(this, property)
disp("setPropが呼ばれました″);
this.property=property;
end

function property=getProp(this)
disp("getPropが呼ばれました″);
property=this.property;
```

end

end

end

```
t=test2_;
t=t.setProp(9);
```

```
setPropが呼ばれました
```

```
t.getProp()
```

```
getPropが呼ばれました
ans = 9
```

2-7. パーセプトロンを書こう

パーセプトロンのクラスを書くにあたって、まずはクラス図を用いた設計を行った。

重みについてはインスタンス生成時にコンストラクタへ引数を渡すのではなく、疑似セッタによって設定するものとした。

これは今後設計していくニューラルネットワークが学習する際に、各パーセプトロンの重みを変更する必要があるためである。

クラス図は図2.7.1のように設計された。

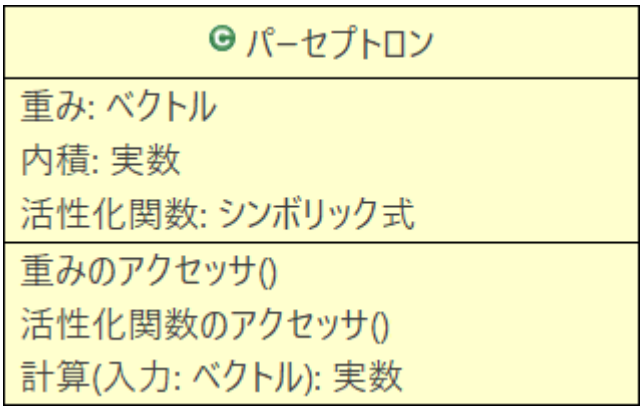


図2.7.1 パーセプトロンのクラス図

classdef Perceptron_

%PERCEPTRON_ パーセプトロン

% 詳細はリファレンスドキュメントに示した。

% リファレンスドキュメントでは、Perceptron_のインスタンスをpとする。

properties (Access = private)

w0;%バイアス

ws;%重みベクトル。長さは入力数と同じだけ必要

x;%活性化関数の入力変数(シンボリック式)

actFunc;%活性化関数(デフォルトはシグモイド関数)(シンボリック式)

end

methods

function this = Perceptron_(cnt_of_input)

%コンストラクタ(入力ノード数)

this.w0=0; this.ws=zeros(1,cnt_of_input);

syms x; this.x=x; this.actFunc=(1+exp(-x))^-1;

end

function output = run(this, inputs)

%パーセプトロンの出力を得る

output = double(this.test(inputs));

end

function output = test(this, inputs)

%runの結果が返す値を導出するためのシンボリック式を出力

if not(isvector(inputs) && length(inputs)==length(this.ws))

error('Perceptron_.testまたはPerceptron_.runに対する引数は、入力数と同じ長さのベクトルである必要があります');

end

a=this.w0+dot(inputs,this.ws);

output=subs(this.actFunc, this.x, a);

end

function this=setW(this, w)

%p=p.setW([bias wedges]);
で、バイアスと重みベクトルを設定。

if not(isvector(w) && length(w)==length(this.ws)+1)

error('Perceptron_.setWへは、次のようなベクトルを渡すようにしてください。¥n・長さは入力数+1です。¥n・1番目はバイアスです。¥n・1番目はバイアスです。','');%#ok<CTPCT>

end

this.w0 = w(1);

this.ws = w(2:end);

end

function rtn=getW(this)

%フィールドw0(バイアス)とws(重みベクトル)を持つ構造体を返却。

rtn.w0=this.w0;

rtn.ws=this.ws;

end

function this=setActFunc(this, var, actFunc)

%syms x;
y=(xを引数とする活性化関数);〈br〉 p=p.setActFunc(x,y);
で、活性化関数を設定。〈br〉デフォルトはα=1のシグモイド関数。

this.x = var;

this.actFunc=actFunc;

end

end

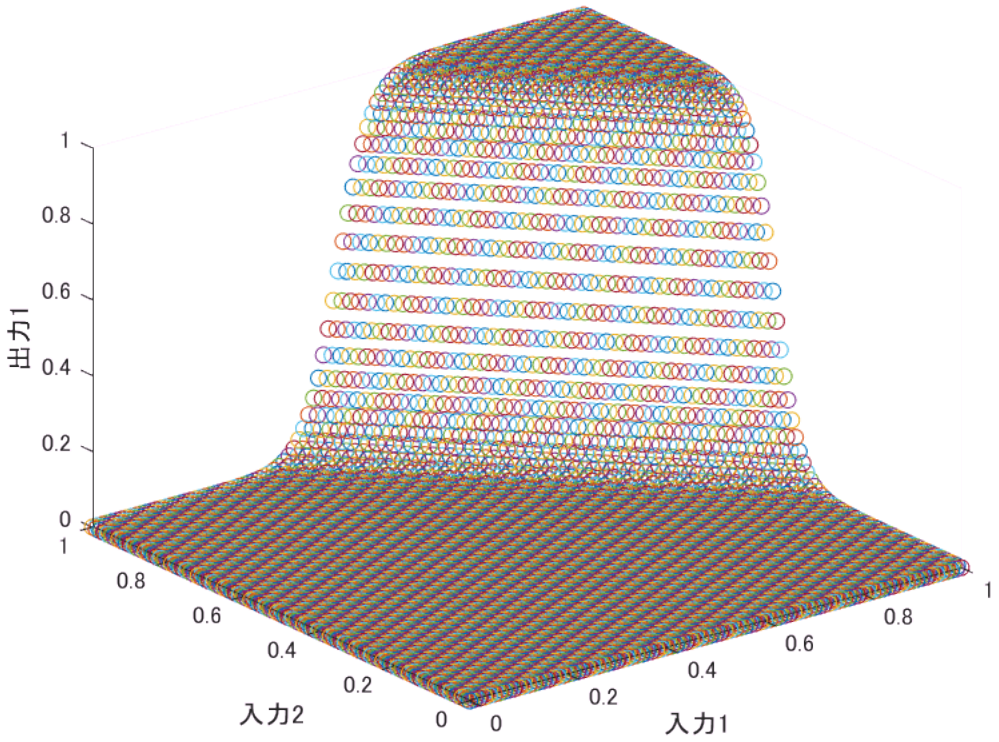
end

このパーセプトロンに重み(-15,10,10)を設定し、2つの入力をそれぞれ0～1の間で行った。

```
p=Perceptron_(2)%入力数2
```

```
p =
  Perceptron_ にはプロパティがありません。
```

```
p=p.setW(3*[-15 10 10]);
x=[0:0.01:1];
y=[0:0.01:1];
figure
hold on
for i=x
    for j=y
        scatter3(i, j, p.run([i j]));
    end
end
xlabel("入力1");
ylabel("入力2");
zlabel("出力1");
view(3);

hold off
```



すると、3×(−15+10x+10y)=0つまり−3+2x+2y=0をほぼ境に、出力が0と1で分かれていることが確認できる。

2-8. パーセプトロンを利用して作る多層パーセプトロン

多層パーセプトロンは、パーセプトロンクラスを複数回呼び出すことで実装できる。

クラス図は図2.8.1のようになった。

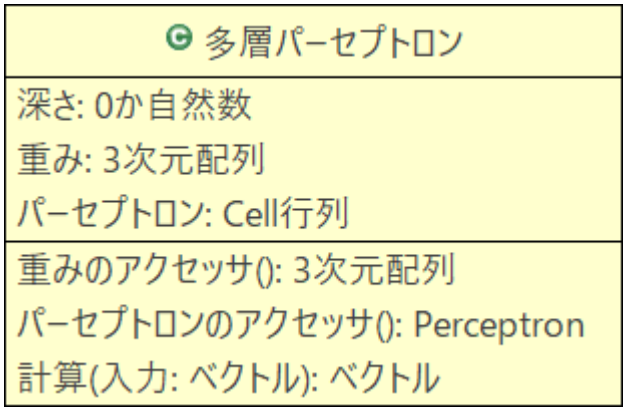


図2.8.1 多重パーセプトロン

少しばかり設計が大変になるので、コンストラクタ、重みやパーセプトロンのアクセッサ(セッタやゲッタのこと)、計算機能のそれぞれについて、見出しを分けて解説していく。

2-8-1. コンストラクタ

まず、多層パーセプトロンに関しても、途中で形状を変更することはないものとしよう。

そのため、コンストラクタに、多層パーセプトロンの形状(層の個数や、各層におけるパーセプトロンの個数や、各パーセプトロンにおける入力数)を記述することでインスタンス生成ができる仕様にしよう。

具体的には、以下のように2n個(nは層の深さに応じて可変)の成分で形状を指定する。

p=MLP(第1層目の入力数, 第1層目のパーセプトロンの個数, 第2層目の入力数, 第2層目のパーセプトロンの個数, . . .);

これを可能にするコードはMLP1.mに示した。

```
classdef MLP1
%MLP1 多層パーセプトロン(MultiLayerPerceptron)
% 試作1

properties %(Access = private)試作1では確認のためプロパティを公開
depth;%層の深さ
w;%3次元重み配列
p=cell(0);%パーセプトロンのCell行列
p_input_size;%各層でのパーセプトロンの入力数
p_cnt;%各層でのパーセプトロンの個数
end

methods
function this=MLP1(varargin)
%多重パーセプトロンの形状を決定する。<br>(第1層目の入力数, 第1層目のパーセプトロンの個数, 第2層目の入力数, 第2層目のパーセプトロンの個数, . . . )
if(mod(nargin,2)~=0 || not(isnumeric(cell2mat(varargin(1:this.depth*2)))))
error('MLPのコンストラクタへの引数は偶数個であり、かつすべて数値である必要があります' );
end

this.depth = nargin/2; this.p_input_size(this.depth)=0; this.p_cnt(this.depth)=0;
in=cell2mat(varargin(1:this.depth*2));
this.w=NaN(max(in)+1+1, max(in)+1, this.depth);%とりあえず確実に十分な大きさでwの領域を確保(高速化)

for i=1:this.depth%各層について
```

```
this.p_input_size(i) = varargin{i*2-1}; in_size=this.p_input_size(i);
this.p_cnt(i) = varargin{i*2}; p_cnt=this.p_cnt(i);
this.w(1:in_size+1, 1:p_cnt, i) = ones(in_size+1, p_cnt);%重み行列の有効範囲に1を立てておく

p_objs=repmat(Perceptron_(in_size), p_cnt, 1);
this.p(1:p_cnt, i)=num2cell(p_objs);
end

w_row_max=max(this.p_input_size); w_col_max=max(this.p_cnt);
this.w=this.w(1:w_row_max+1, 1:w_col_max, 1:this.depth);%wの余分な領域を切り落とす
end
end
end
```

```
mlp=MLP1(9,6,7,8)
```

```
mlp =
  MLP1 のプロパティ:

           depth: 2
               w: [10×8×2 double]
               p: {8×2 cell}
     p_input_size: [9 7]
           p_cnt: [6 8]
```

```
%{
第1層目パーセプトロンが入力数9, 個数6
第2層目入力数7,個数8
%}
w=mlp.w
```

```
w =

(:,:,1) =

     1     1     1     1     1     1   NaN   NaN
     1     1     1     1     1     1   NaN   NaN
     1     1     1     1     1     1   NaN   NaN
     1     1     1     1     1     1   NaN   NaN
     1     1     1     1     1     1   NaN   NaN
     1     1     1     1     1     1   NaN   NaN
     1     1     1     1     1     1   NaN   NaN
     1     1     1     1     1     1   NaN   NaN
     1     1     1     1     1     1   NaN   NaN
     1     1     1     1     1     1   NaN   NaN


(:,:,2) =

     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1
```

```
     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1
   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
```

```
p=mlp.p
```

```
p = 8×2 の cell 配列
    [1×1 Perceptron_]    [1×1 Perceptron_]
    [1×1 Perceptron_]    [1×1 Perceptron_]
    [1×1 Perceptron_]    [1×1 Perceptron_]
    [1×1 Perceptron_]    [1×1 Perceptron_]
    [1×1 Perceptron_]    [1×1 Perceptron_]
    [1×1 Perceptron_]    [1×1 Perceptron_]
                  []    [1×1 Perceptron_]
                  []    [1×1 Perceptron_]
```

重み配列wについては、第1層(:,:,1)では成分が「1」となるような行数が入力数+1つまり10となっている。1加算したのはバイアスのためである。また列数はパーセプトロンの個数である6だ。

第2層(:,:,2)についても同様に、成分が「1」となるような行数が7+1で8,列数は8となっている。

成分は「1」以外に「NaN」がある。「NaN」は余白と思えばよい。

パーセプトロンのcell配列pについても、1列目(第1層に相当)は、個数として指定した通り6個、2列目は8個、オブジェクトが生成された。

またcell配列というのは、「型にこだわらずどんなものでも入れられるデータコンテナとしての配列」のことである。

通常の配列では、要素はすべて同じ型のデータである必要があり、クラスオブジェクトを格納している場合、「空」を表現することが非常に困難であるが、cell配列を使うことで「空」(実態は0行0列のdouble行列)を容易に表現できる。

2-8-2. 疑似セッタと疑似ゲッタ

次は重みのアクセッサを作ろう

オブジェクト生成直後、コンストラクタによってプロパティwは1とNaNのみからなる3次元配列となる。

これは各層の重み行列を3次元目の方向に並べたものである。

重みの設定可能なインデックスには必ず1が入っていて、重みの設定不可能な「余白」のインデックスには必ずNaNが入っている。

そのため、3次元配列の大きさと、NaNの情報から、各層に対応する重み行列のサイズを特定できる。

これらの情報を一気にコンストラクタへ渡すのは工夫がいる。例えば3次元配列などを使うのが適切だろう。

3次元配列とは、2次元配列にさらに1次元追加したものだ。2次元配列とは行列のことであるから、3次元配列とは

行列を一列(実際は高さ方向)に並べたもの

であると考えればよい。

この行列は、(各層の)重み行列にすればよい。というのも重み行列は、(入力数+1)行(パーセプトロンの個数)列の行列であり、自身の大きさにより入力数やパーセプトロンの個数を示しているからだ。

そして層の深さは3次元目のサイズで表現している。以下、この3次元配列を「3次元重み配列」と呼び、$\boldsymbol{W}$とかくことにする。

MLP1.mに以下の重みアクセッサを追加したものをMLP2.mとして保存した。(プロパティはprivateに設定した)

```
function this=setW2(this,layer_idx, w2)
%mlp=mlp.setW2(layer_idx, w2)で、第layer_idx層の重み行列としてw2を設定する。
row_size=this.p_input_size(layer_idx)+1; col_size=this.p_cnt(layer_idx);
if not(size(w2)==[row_size col_size])
error('重み行列の大きさが異常です');
end
this.w(1:row_size, 1:col_size, layer_idx)=w2;
end
function this=setW3(this, w3)
%mlp=mlp.setW3(w3)で、第layer_idx層の重み行列としてw3を設定する。
row_size=max(this.p_input_size)+1; col_size=max(this.p_cnt); hei_size=this.depth;
if not(size(w3)==[row_size col_size hei_size])
error('3次元重み配列の大きさが異常です');
end
this.w(1:row_size, 1:col_size, 1:hei_size)=w3;
end
function this=setW0(this, input_idx, p_idx, layer_idx, w0)
%mlp=mlp.setW0(input_idx, p_idx, layer_idx, w0)で、第layer_idx層のp_idx番目のパーセプトロンについて、input_idx-1番目の入力(0番目ならバイアス)に対する重みをw0と設定
if(isnan(this.w(input_idx, p_idx, layer_idx)))
error('余白に重みを書き込むことは出来ません');
end
this.w(input_idx, p_idx, layer_idx)=w0;
end

function w2=getW2(this, layer_idx)
%第layer_idx層の重み行列を取得する。
w2 = this.w(1:this.p_input_size(layer_idx), 1:this.p_cnt(layer_idx), layer_idx);
```

end

function w3=getW3(this)

%3次元重み配列を取得する。

w3 = this.w;

end

function w0=getW0(this, input_idx, p_idx, layer_idx)

%getW0(this, input_idx, p_idx, layer_idx)
第layer_idx層のp_idx番目のパーセプトロンについて、input_idx-1番目の入力(0番目ならバイアス)に対する重みを取得する。

w0 = this.w(input_idx, p_idx, layer_idx);

if(isnan(w0))

error('重み配列の余白にアクセスしました');

end

end

```
disp('mlp=MLP2(4,3,3,2,2,1)')
```

```
mlp=MLP2(4,3,3,2,2,1)
```

```
mlp=MLP2(4,3,3,2,2,1)
```

```
mlp =
  MLP2 にはプロパティがありません。
```

```
disp('  ');
```

```
disp('mlp.getW3()');
```

```
mlp.getW3()
```

```
mlp.getW3()
```

```
ans =

(:,:,1) =

     1     1     1
     1     1     1
     1     1     1
     1     1     1
     1     1     1


(:,:,2) =

     1     1   NaN
     1     1   NaN
     1     1   NaN
     1     1   NaN
```

```
   NaN   NaN   NaN


(:,:,3) =

     1   NaN   NaN
     1   NaN   NaN
     1   NaN   NaN
   NaN   NaN   NaN
   NaN   NaN   NaN
```

```
disp('  ');
```

```
disp('mlp=mlp.setW3(mlp.getW3().*rand(size(mlp.getW3())));');
```

```
mlp=mlp.setW3(mlp.getW3().*rand(size(mlp.getW3())));
```

```
mlp=mlp.setW3(mlp.getW3().*rand(size(mlp.getW3())));
disp('  ');
```

```
disp('mlp.getW3()');
```

```
mlp.getW3()
```

```
mlp.getW3()
```

```
ans =

(:,:,1) =

    0.8147    0.0975    0.1576
    0.9058    0.2785    0.9706
    0.1270    0.5469    0.9572
    0.9134    0.9575    0.4854
    0.6324    0.9649    0.8003


(:,:,2) =

    0.1419    0.6557       NaN
    0.4218    0.0357       NaN
    0.9157    0.8491       NaN
    0.7922    0.9340       NaN
       NaN       NaN       NaN


(:,:,3) =

    0.7060       NaN       NaN
    0.0318       NaN       NaN
    0.2769       NaN       NaN
       NaN       NaN       NaN
       NaN       NaN       NaN
```

次に、パーセプトロンのアクセッサを追加する。

活性化関数を変更するなどの用途で、外部からパーセプトロンへアクセスしたい場合があるかもしれないからだ。

MLP3.mではmethod～end間に、以下を追加した。

function this=setP(this, p)

%mlp=mlp.setP(p);で、パーセプトロンのCell配列を設定する。

if not(size(p)==size(this.p)&&(cellfun(@isempty,p)==cellfun(@isempty,this.p)))

error('パーセプトロンのCell配列の大きさが異常か、または個数が矛盾しています。');

end

this.p=p;

end

function p=getP(this)

%パーセプトロンのCell配列を取得する。

p=this.p;

end

```
mlp=MLP3(4,3,3,2,2,1);
p=mlp.getP()
```

```
p = 3×3 の cell 配列
    [1×1 Perceptron_]    [1×1 Perceptron_]    [1×1 Perceptron_]
    [1×1 Perceptron_]    [1×1 Perceptron_]                   []
    [1×1 Perceptron_]                   []                   []
```

```
p{1,2}
```

```
ans =
  Perceptron_ にはプロパティがありません。
```

2-8-3. 計算実行メソッド(1/2)

そして、多層パーセプトロンを実際に動作させるための機能を書こう。

入力を受け取って、Cell行列内の各パーセプトロンを使って逐次計算し、出力を得るためには、

まず各パーセプトロンへ、3次元重み配列から切り取った重みベクトルを書き込む必要がある。2-8-3節ではこれを中心に解説し、実際に計算する過程は2-8-4節で実装することとする。

重みベクトルの書き込みを、多層パーセプトロンとしての出力を得る時(つまりrunメソッド呼び出し時)に行うとすると、MLP3.mに対して次のようにできる。

- methods~endの間に次を追加

```
function outputs=run(this, inputs)
%入力ベクトルを受け取り、多層パーセプトロンとしての出力を返却する。
if not(isvector(inputs) && length(inputs)==this.p_input_size(1))
error(char("入力は第1層パーセプトロンの入力数である"+(this.p_input_size(1))+"を長さとするベクトルで有る必要があります。"));
end
inputs=reshape(inputs,[1,length(inputs)]);%inputsが列ベクトルなら行ベクトルへ変換
if length(inputs)<max(this.p_input_size)+1
inputs(1,max(this.p_input_size)+1)=0; %inputsの大きさをmax(this.p_input_size)+1へ拡張
end
this=this.applyW();%重み配列の内容を各パーセプトロンへ反映
%★
outputs=this.p;
end
```

・methods~endの外に次を追加

```
methods (Access = private)
function this=applyW(this)
for layer_idx=1:this.depth
row_size=this.p_input_size(layer_idx)+1; col_size=this.p_cnt(layer_idx); p_cnt_=col_size;
w_vec_vec=mat2cell(this.w(1:row_size, 1:col_size, layer_idx), row_size, ones(1, col_size));
this.p(1:p_cnt_, layer_idx)=cellfun(@this.writeWtoP, this.p(1:p_cnt_,layer_idx), w_vec_vec.', 'UniformOutput', false);
end
end

function p=writeWtoP(~, elem_of_p, col_of_w2)
p=elem_of_p.setW(col_of_w2);
end
end
```

これをMLP4.mとする。

「inputsの大きさをmax(this.p_input_size)+1へ拡張」は、2-8-4節にてinputsを繰り返し利用するので、あらかじめ必要な大きさを確保するためのコードである。

privateなメソッドapplyWでは、for文で各層について順番に処理を行っている。但し、それ以外ではループ文を用いていない。これは、処理を高速化させるための工夫である。cells_o = cellfun(@func, cells_1, cells_2,..., 'UniformOutput', false)はmatlabで標準で用意された関数で、cells_o(i) へ

func(cells_1(i), cells_2(i), ...)の出力を代入することを、すべての要素(i番目)に対して行う。ここで、cells_で始まるのはすべて同じ大きさのcell配列である。

2-8-1節ではcell配列を「型にこだわらずどんなデータも入れられる」といったが、配列(普通の配列でもcell配列でもよい)を入れることも可能だ。そこで1次元配列である重み列ベクトルをcell行ベクトルの要素とすることで、2次元配列である重み行列を「列ベクトルの行ベクトル」という1次元配列に変換できる。

w_vec_vecでは、(着目中の層における)重み行列に対してこのテクニックを使い、this.p(1:p_cnt_, layer_idx)と同じ次元・各次元における大きさを持つよう、重み行列を変換したものである。(mat2cellの第2引数以降で、cell配列にする際の分割の仕方を指定している)

w_vec_vecとthis.p(1:p_cnt_,layer_idx)はどちらもcellベクトルなのでcellfunが利用可能であるというわけだ。

ただ、せっかく以上のような工夫を施したが、for文の入れ子を作った場合と比べて大した高速化は期待できない。(それどころかむしろ遅くなるようだ。)このことは[7]で詳しく説明されている。実際に高速化するためには、matlabでサポートされているGPUを用意し、プロパティpやw_vec_vecを「GPU配列」にする必要がある。(筆者の環境ではGPU配列が使えないため、使用する例を示せない。)

GPU配列を使わない場合、applyWメソッドは次のようにすればよい。

```
methods (Access = private)

function this=applyW(this)

for layer_idx=1:this.depth

p_cnt_=this.p_cnt(layer_idx);

for p_idx=1:p_cnt_

this.p{p_idx, layer_idx}=this.p{p_idx, layer_idx}.setW(this.w(:, p_idx, layer_idx));

end

end

end

end
```

これをMLP4_.mとする。

```
mlp=MLP4(4,3,3,2,2,1);
mlp_=MLP4_(4,3,3,2,2,1);
set=mlp.getW3().*rand(size(mlp.getW3()));

mlp=mlp.setW3(set);
mlp_=mlp_.setW3(set);

p=mlp.run([1 2 3 4]);
p_=mlp_.run([1 2 3 4]);

disp('----↓パーセプトロンに書き込まれた重みは両者一致する①');
```

```
----↓パーセプトロンに書き込まれた重みは両者一致する①
```

```
p{1,1}.getW.w0
```

```
ans = 0.4898
```

```
p{1,1}.getW.ws
```

```
ans =
    0.4456
    0.6463
    0.7094
    0.7547
```

```
p_{1,1}.getW.w0
```

```
ans = 0.4898
```

```
p_{1,1}.getW.ws
```

```
ans =
    0.4456
    0.6463
    0.7094
    0.7547
```

```
disp('----');
```

```
----
```

```
disp('  ');
```

```
disp('----↓速度評価②');
```

```
----↓速度評価②
```

```
tic
for i=1:10000
    p=mlp.run([1 2 3 4]);
end
disp("cellfunによるコードで10000回実行した結果:"+toc+"秒かかる");
```

```
cellfunによるコードで10000回実行した結果:9.0291秒かかる
```

```
tic
for i=1:10000
    p_=mlp_.run([1 2 3 4]);
end
disp("for文入れ子によるコードで10000回実行した結果:"+toc+"秒かかる");
```

```
for文入れ子によるコードで10000回実行した結果:3.0831秒かかる
```

①の結果より、2つの多重パーセプトロンオブジェクトが同じ重み配列を持つとき、cellfunによる重みの書き込みと、for文入れ子による書き込みを行った結果、どちらも同じ重みがパーセプトロンに書き込まれることが確認できる。

また、②の結果より、GPU配列を用いない場合、cellfunを用いるとむしろ処理が遅くなるということが分かった。

```
mlp=MLP4(100,90, 90,80, 80,70, 70,60, 60,50, 50,40, 40,30, 30,20, 20,10, 10,1);
mlp_=MLP4_(100,90, 90,80, 80,70, 70,60, 60,50, 50,40, 40,30, 30,20, 20,10, 10,1);
set=mlp.getW3().*rand(size(mlp.getW3()));

mlp=mlp.setW3(set);
mlp_=mlp_.setW3(set);

p=mlp.run(1:100);
p_=mlp_.run(1:100);

disp('----↓パーセプトロンに書き込まれた重みは両者一致する①');
```

```
----↓パーセプトロンに書き込まれた重みは両者一致する①
```

```
p{1,1}.getW.w0
```

```
ans = 0.0759
```

```
p{1,1}.getW.ws
```

```
ans =
    0.0540
    0.5308
    0.7792
    0.9340
    0.1299
    0.5688
    0.4694
    0.0119
    0.3371
    0.1622
     ⋮
     ⋮
```

```
p_{1,1}.getW.w0
```

```
ans = 0.0759
```

```
p_{1,1}.getW.ws
```

```
ans =
    0.0540
    0.5308
    0.7792
    0.9340
    0.1299
    0.5688
    0.4694
    0.0119
    0.3371
    0.1622
     ⋮
     ⋮
```

```
disp('----');
```

```
----
```

```
disp('  ');
```

```
disp('----↓速度評価②');
```

```
----↓速度評価②
```

```
tic
for i=1:10000
    p=mlp.run(1:100);
end
disp("cellfunによるコードで10000回実行した結果:"+toc+"秒かかる");
```

```
cellfunによるコードで10000回実行した結果:244.7826秒かかる
```

```
tic
for i=1:10000
    p_=mlp_.run(1:100);
end
disp("for文入れ子によるコードで10000回実行した結果:"+toc+"秒かかる");
```

```
for文入れ子によるコードで10000回実行した結果:131.3095秒かかる
```

2-8-4. 計算実行メソッド(2/2)

2-8-3節にて、それぞれのパーセプトロンへ重みを書き込んだので、後は順番にパーセプトロンを動かしていくことで、多重パーセプトロンとしての出力を得ることができる。

そのアルゴリズムは次の通り。

1. 入力ベクトルの受け取り
2. 第1層のパーセプトロンの取得
3. 取得したパーセプトロンにおけるrun(入力ベクトル)メソッド実行
4. その出力を入力ベクトルへ上書き
5. 第2層のパーセプトロンの取得
6. 取得したパーセプトロンにおけるrun(入力ベクトル)メソッド実行
7. その出力を入力ベクトルへ上書き
8. ・・・
9. (最終層の出力は、そのまま多層パーセプトロン全体としての出力として扱う)

計算を実行するコードは、MLP4.mまたはMLP4_.mの★のところへ挿入する。この時、outputs=this.p;は消去すること。

for layer_idx=1:this.depth

p_vec=this.p(1:this.p_cnt(layer_idx), layer_idx);%第layer_idx層のパーセプトロンの取得

outputs=inputs;%outputsのサイズ確保

for p_idx=1:this.p_cnt(layer_idx)

outputs(p_idx)=p_vec{p_idx}.run(inputs(1:this.p_input_size(layer_idx)));%run(入力ベクトル)実行, その出力を入力ベクトルへ上書き

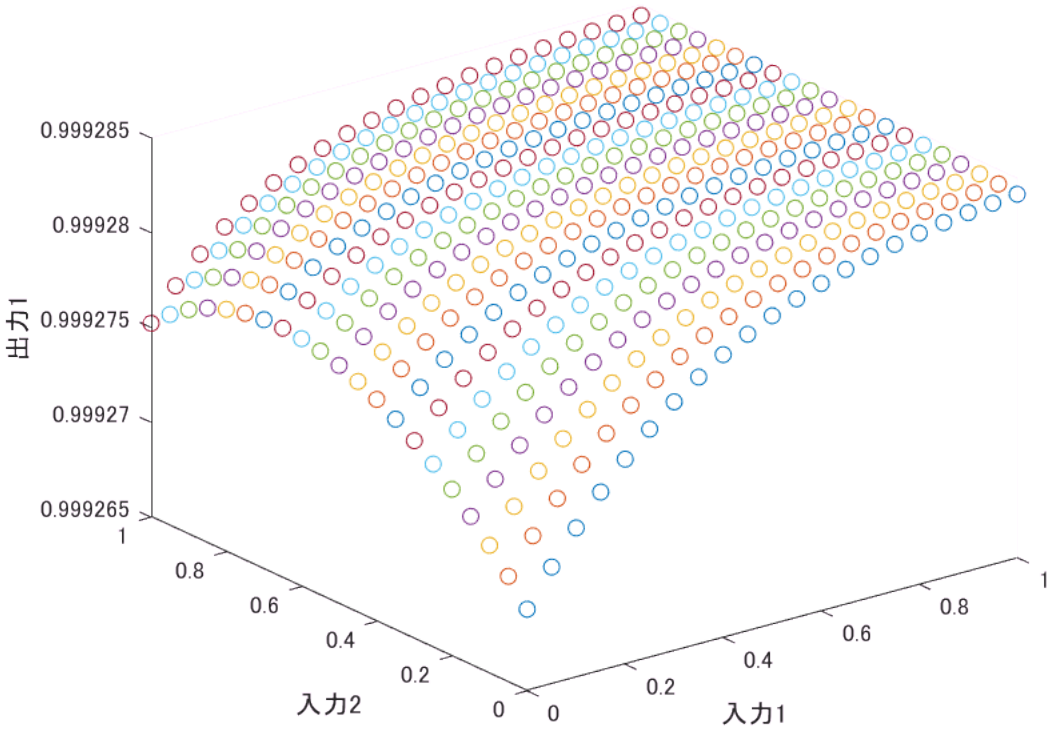
end

inputs=outputs;

end

outputs=inputs(1:this.p_cnt(this.depth));%最終層の出力は、そのまま多層パーセプトロンの全体の出力

```
mlp=MLP5(2,3,3,2,2,1);
mlp=mlp.setW3(10.*mlp.getW3().*rand(size(mlp.getW3()))-5);
x=[0:0.05:1];
y=[0:0.05:1];
figure
hold on
for i=x
    for j=y
        scatter3(i, j, mlp.run([i j]));
    end
end
xlabel("入力1");
ylabel("入力2");
zlabel("出力1");
view(3);

hold off
```



以上で、パーセプトロンを用いた多層パーセプトロン(ニューラルネットワーク)が完成した。

2-8. 行列を利用して作る多層パーセプトロン

2-7節では、パーセプトロンを複数台用いるだけで、多層パーセプトロン(ニューラルネットワーク)が実現することを、実例により示した。

但し、2-5節でも述べた通り、多層パーセプトロンの各層における計算(活性化関数除く)は、次の式の通りの行列演算として示すことができる。

$$\begin{pmatrix}1 & x_1 & x_2\end{pmatrix}\begin{pmatrix}w_{1,0} & w_{2,0}\\ w_{1,1} & w_{2,1}\\ w_{1,2} & w_{2,2}\end{pmatrix}=\begin{pmatrix}\text{パーセプトロン1における}a & \text{パーセプトロン2における}a\end{pmatrix}$$

こちらを実装する方が、パーセプトロンのクラスのオブジェクトを利用したものよりもコードが簡潔になり、また高速に動作する。ここでは2-7節で作ったMLP5.mを改造し、内部計算にパーセプトロンのオブジェクトを利用しない仕様のMLP6.mを設計しよう。

2-8-1. プロパティの変更

MLP5.mのprivateプロパティpはパーセプトロンのオブジェクトを格納するcell行列である。

しかし今回、計算にはこれらのオブジェクトを用いないため、このオブジェクトは不要である。

但し、活性化関数は必要なので、シンボリック式を格納する配列を代わりに用意する必要がある。

properties (Access = private)～endの部分は、次のように変更する。

```
properties (Access = private)
depth;%層の深さ
w;%3次元重み配列
actFunc=actFunc=struct('func',sym(0), 'var',sym(0));%funcは活性化関数のシンボリック式Cell行列、varは活性化関数の入力を表す変数
p_input_size;%各層でのパーセプトロンの入力数
p_cnt;%各層でのパーセプトロンの個数
end
```

プロパティpを取り除き、代わりにactFuncを追加した。

2-8-2. アクセッサの変更

また、pの疑似セッタや疑似ゲッタを取り除き、代わりにactFuncの疑似セッタ、疑似ゲッタを追加する

```
function this=setActFunc(this, actFunc)
```

%mlp=mlp.setActFunc(actFunc.func);で、活性化関数の行列actFunc.funcを設定する。
但し引数はactFunc.varで。

```
if not(size(actFunc.func)==size(this.actFunc.func))
error('活性化関数配列の大きさが異常です。');
end
this.actFunc=actFunc;
end
function actFunc=getActFunc(this)
%活性化関数の行列および引数を構造体で取得する。
actFunc.func=this.actFunc;
actFunc.var=this.actFunc.var;
end
```

2-8-3. コンストラクタ

次のコードは、パーセプトロンのオブジェクトを初期化するためのものであるから、取り除く。

```
p_objs=repmat(Perceptron_(in_size), p_cnt, 1);
this.p(1:p_cnt, i)=num2cell(p_objs);
```

代わりに、次のようにactFuncを初期化するコードを挿入する。

```
this.actFunc.func(1:p_cnt, i)=(1+exp(-this.actFunc.var))^-1;
```

シンボリック式で配列を作った際は、未指定のインデックスには0が自動で代入される。

また、コンストラクタの最初の処理として

```
syms x;
this.actFunc.var=x;
```

も必要だ。

2-8-4. runメソッド

applyWメソッドに関してはmethodsブロックごと消す。

runメソッドは次のように置き換える。

```
function outputs=run(this,inputs)
%入力ベクトルを受け取り、多層パーセプトロンとしての出力を返却する。
if not(isvector(inputs) && length(inputs)==this.p_input_size(1))
error(char("入力は第1層パーセプトロンの入力数である"+(this.p_input_size(1))+"を長さとするベクトルで有る必要があります。"));
```

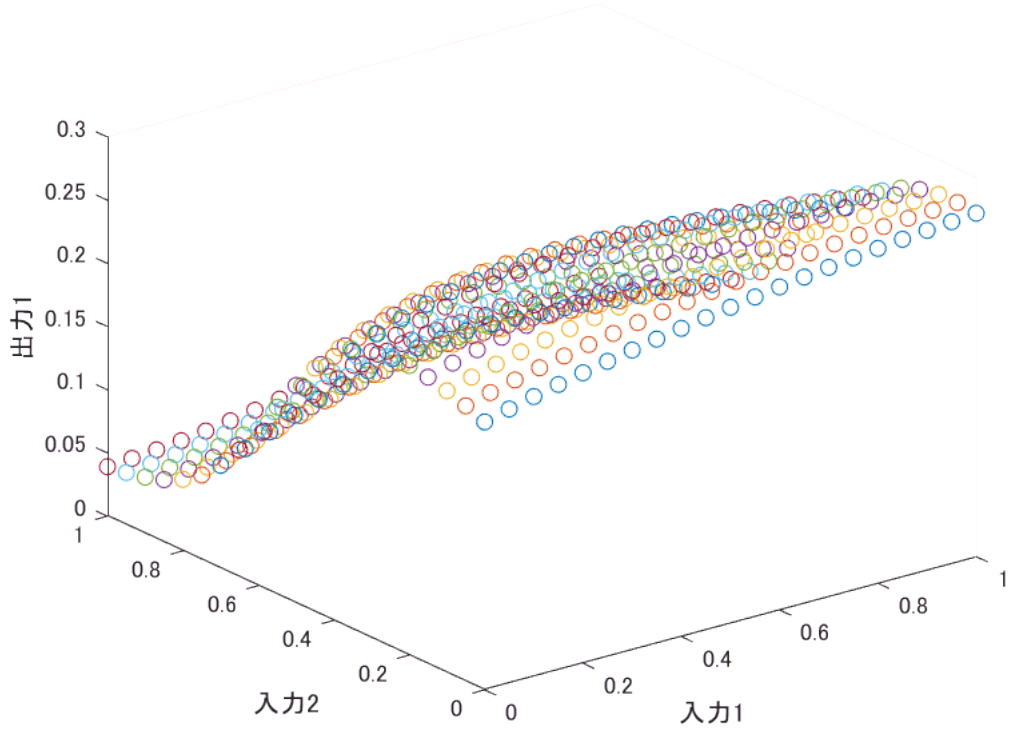
```
end
inputs=reshape(inputs, 1, length(inputs));%行ベクトル化
if length(inputs)<max(this.p_input_size)+1
inputs(max(this.p_input_size)+1)=0;%サイズ確保
end
for layer_idx=1:this.depth
row_size=this.p_input_size(layer_idx)+1; col_size=this.p_cnt(layer_idx);
mat_mult=[1 inputs(1:row_size-1)]*this.w(1:row_size, 1:col_size, layer_idx);%行列演算
for p_idx=1:this.p_cnt(layer_idx)
inputs(p_idx)=double(subs(this.actFunc.func(p_idx, layer_idx), this.actFunc.var, mat_mult(p_idx)));
end
end
outputs=inputs(1:this.p_cnt(this.depth));
end
```

```
mlp5=MLP5(2,3,3,2,2,1);
mlp6=MLP6(2,3,3,2,2,1);
w=10.*mlp5.getW3().*rand(size(mlp5.getW3()))-5
```

```
w =

(:,:,1) =

   -2.5932   -2.8069   -0.1763
   -4.4048    1.3406   -1.0353
    3.2498    4.1814   -2.1995
       NaN       NaN       NaN


(:,:,2) =

   -3.8087    2.5962       NaN
    3.9796    0.4794       NaN
    4.5303    3.8584       NaN
    2.2469   -1.8635       NaN


(:,:,3) =

   -4.7789       NaN       NaN
   -2.8451       NaN       NaN
    4.2437       NaN       NaN
       NaN       NaN       NaN
```

```
mlp5=mlp5.setW3(w);
mlp6=mlp6.setW3(w);
x=[0:0.05:1];
y=[0:0.05:1];
tic;
figure
```

```
hold on
for i=x
    for j=y
        scatter3(i, j, mlp5.run([i j]));
    end
end
xlabel("入力1");
ylabel("入力2");
zlabel("出力1");
view(3);

hold off
```

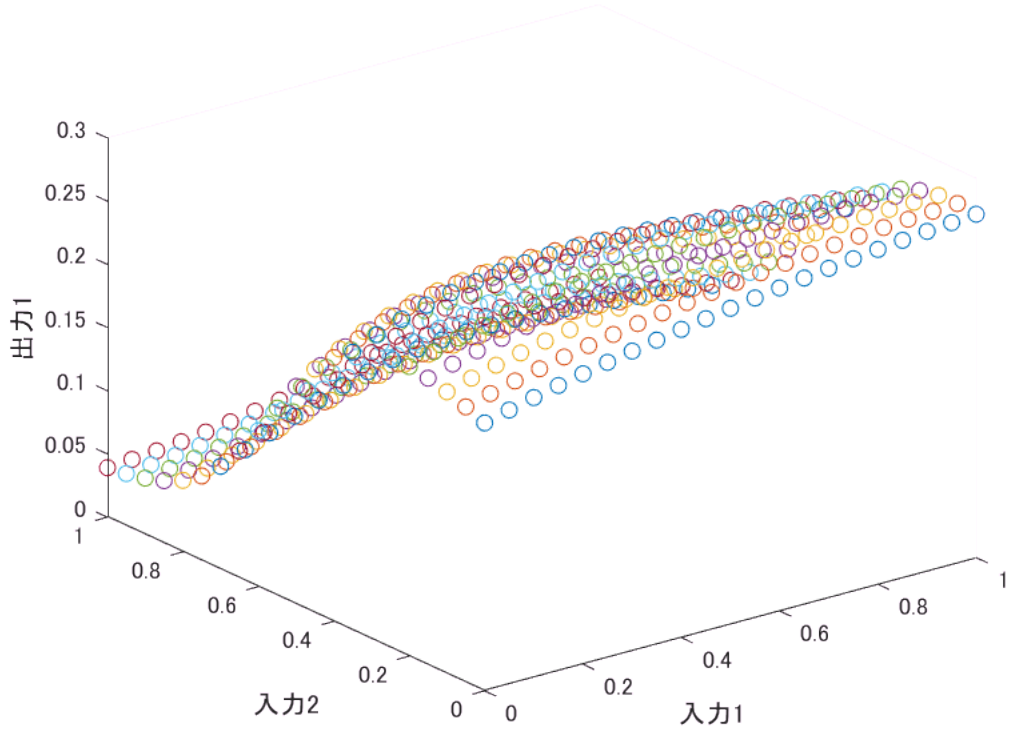



```
disp("mlp5の所要時間:"+toc);
```

```
mlp5の所要時間:29.6717
```

```
tic;
figure
hold on
for i=x
    for j=y
        scatter3(i, j, mlp6.run([i j]));
    end
end
xlabel("入力1");
ylabel("入力2");
zlabel("出力1");
view(3);
```

```
hold off
```




```
disp("mlp6の所要時間:"+toc);
```

```
mlp6の所要時間:48.4288
```

2-9. 出力層の活性化関数―分類問題と回帰問題

ニューラルネットワークの教科書によれば、機械学習の教師有り学習の問題は分類問題と回帰問題に大別できるそうだ。

多層パーセプトロンは分類問題にも回帰問題にも用いることができる。

分類問題は「クラスの境界線」を学習することで行うクラスタリングであり、回帰問題は「データが近づきそうな線」を学習することで行う数値を予測といえる。したがって、分類問題の「答え」は「選択肢の中のどれか一つ」(以下「選択」という)であり、回帰問題の「答え」は連続量であるため、問題の種類によって、答えの性質が大きく異なる。

答えの性質が異なるため、同じアルゴリズムでどちらの問題をも解けるようにするには工夫が必要だ。その工夫は2通り考えられる。1つ目は「出力の解釈を違える」ことであり、2つ目は「出力層だけ活性化関数を変更する」ことだ。

「出力の解釈を違える」例としては、0~1の範囲内に定まる連続量としての出力を、どの離散値が最も尤もらしいかを示す確率(尤度)とみなすことなどが考えられる。この場合、連続量を選択として「解釈」出来る。

しかし、そもそもの話、多層パーセプトロンが出力するのは最後の層におけるパーセプトロンの出力そのものであり、これはシグモイド関数によってほぼ0またはほぼ1と見なせる、実質2通りの値だ。これを連続量と見なすのは無理がある。また、選択肢が3つ以上ある場合には、選択とみることも不可能だ。ここでとられるのが、「出力層だけ活性化関数を変更する」という方法だ。

2-9-1. 回帰問題には恒等関数を使う

回帰問題の出力層では、活性化関数を適用せずに、行列演算の結果をそのまま出力値とする方法がとられる。即ち出力層では、活性化関数の入力をxとするなら、活性化関数の式はxそのものとなる。(これを恒等関数という)こうすることで、多層パーセプトロンは連続値を出力出来ることになる。

回帰問題用のクラスを作ろう、MLP6(またはMLP5)クラスを改編してMLP7.mを作り、これを継承するRegression1.mを書けばよい。

matlabでは、可変長引数を実現するためにはvararginという仮引数を関数頭部に宣言する必要がある。そして、可変長引数は、cellベクトルとしてvarargin に格納される。残念ながらこの方法は仮引数のみに有効で、実引数を可変長引数とする術はないようだ。

このことがスーパクラスから可変長引数を持つコンストラクタを継承してくるときに災いし、引数の受け渡しに失敗してしまう。これを解決するために、MLP7.mをまず作る。

MLP7.mはMLP6(または5)のコンストラクタにおける最初のif文を、次のように変更したものである。

前

```
if(mod(nargin,2)~=0 || not(isnumeric(cell2mat(varargin(1:this.depth*2)))))
error('MLPのコンストラクタへの引数は偶数個であり、かつすべて数値である必要があります');
end
```

後

```
if(mod(nargin,2)~=0 || not(isnumeric(cell2mat(varargin(1:this.depth*2)))))
if nargin==1 && iscell(varargin)
varargin=varargin{1};
this.depth = length(varargin)/2;
else
error('MLPのコンストラクタへの引数は偶数個であり、かつすべて数値である必要があります');
end
else
this.depth = nargin/2;
end
```

そして、MLP6.mにもともと存在するthis.depth = nargin/2;は消去する。

続いて、次のようなRegression1.mを書こう。

```
classdef Regression1 < MLP7
```

%REGRESSION1 回帰問題用の多重パーセプトロン

% MLP7の子クラス

methods

function this=Regression1(varargin)

this@MLP7(varargin);

end

function outputs=run(this,inputs)

actFunc=this.ActFunc();

func=actFunc.func; var=actFunc.var;

[p_cnt, out_layer]=size(func);

for p_idx=1:p_cnt

if(func(p_idx, out_layer)~=0)

func(p_idx, out_layer)=var;

end

end

actFunc.func=func; actFunc.var=var;

this=this.setActFunc(actFunc);

outputs=run@MLP7(this,inputs);

end

end

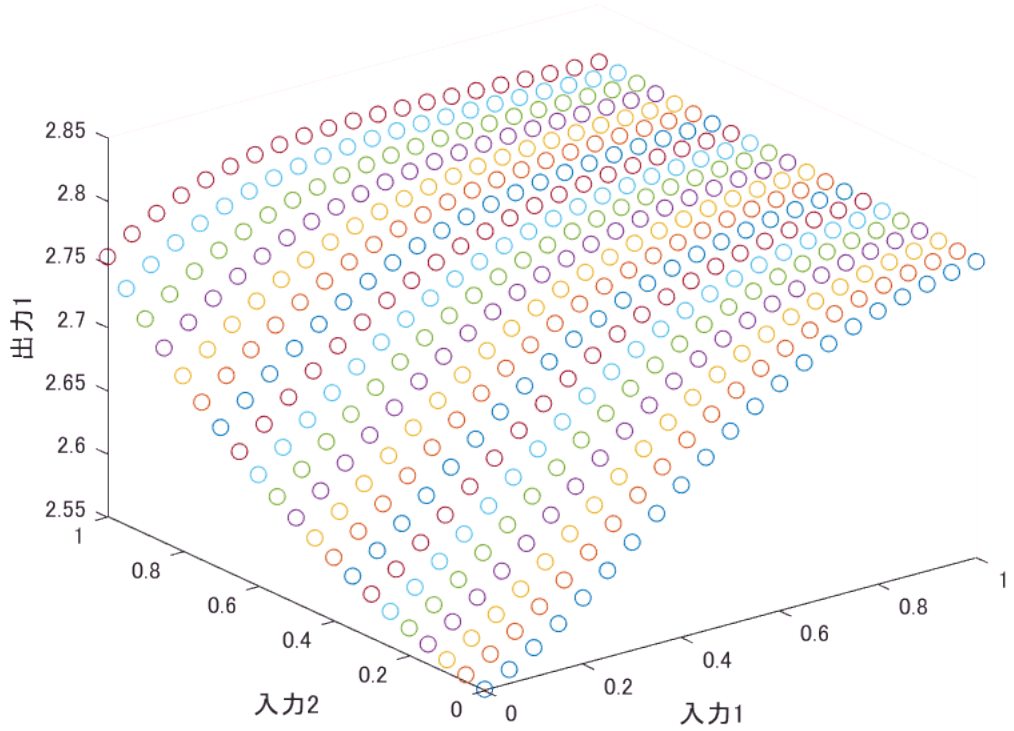
end

```
regression1=Regression1(2,3,3,2,2,1);
mlp6=MLP6(2,3,3,2,2,1);
w=10.*regression1.getW3().*rand(size(regression1.getW3()))-5
```

```
w =

(:,:,1) =

   -0.7129   -3.4876   -4.8657
    4.3884   -4.1269   -2.4913
    1.9156    4.1169    4.0589
       NaN       NaN       NaN


(:,:,2) =

    4.6281   -3.5421       NaN
    0.7903    1.9439       NaN
   -0.2283    0.1343       NaN
```

```
    0.1587    0.6699       NaN

(:,:,3) =

    4.1066       NaN       NaN
   -1.6824       NaN       NaN
    2.1802       NaN       NaN
       NaN       NaN       NaN
```

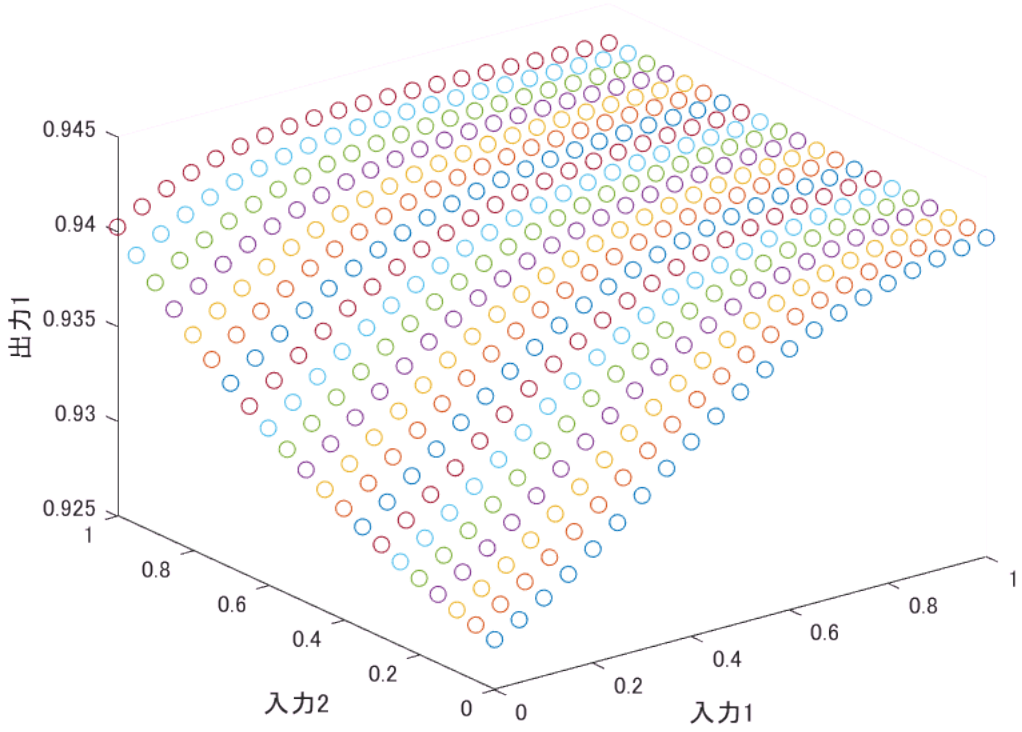
```
regression1=regression1.setW3(w);
mlp6=mlp6.setW3(w);
x=[0:0.05:1];
y=[0:0.05:1];
tic;
figure
hold on
for i=x
    for j=y
        scatter3(i, j, regression1.run([i j]));
    end
end
xlabel("入力1");
ylabel("入力2");
zlabel("出力1");
view(3);

hold off
```




```
disp("regression1の所要時間:"+toc);
```

```
regression1の所要時間:51.5598
```

```
tic;
figure
hold on
for i=x
    for j=y
        scatter3(i, j, mlp6.run([i j]));
    end
end
xlabel("入力1");
ylabel("入力2");
zlabel("出力1");
view(3);

hold off
```




```
disp("mlp6の所要時間:"+toc);
```

```
mlp6の所要時間:48.7143
```

2-9-2. 分類問題にはソフトマックス関数を使う

分類問題では、多重パーセプトロンの各出力(出力ベクトルの各成分)を、各クラスの尤度とみる。

したがって、次のような制約がある。

1. 出力層のパーセプトロンの個数(出力ベクトルの成分の数)とクラスの個数は一致する必要がある。
2. 各出力は0から1の間である

3. 各出力の総和は1となる

条件1については、ただ単にそうすればいい。

条件2と3については、ソフトマックス関数を使うことで実現できる。

ソフトマックス関数$\sigma(a_i)$は次の式で表される。

$$\sigma(a_i) = \frac{\exp(a_i)}{\sum_j \exp(a_j)}$$

これなら、分母分子が共に指数関数であるから、分母分子は共に正であるため計算結果も正になり、

また確実に分子≦分母なので計算結果は1以下であるから、条件2を満たしている。

また、総和をとると、分子が分母と同じになるため、総和の計算結果は1となり、条件3を満たす。

但し、これをそのまま計算するのではいけない。というのも、分母や分子が極端に大きく、あるいは小さくなった場合に情報が消滅してしまう。あまりにも大き(小さ)すぎて、どれくらい大き(小さ)いのか表現しきれないというわけだ。

分数を結果として返すので、このままでは例えば「大きすぎる数/大きすぎる数」や「小さすぎる数/小さすぎる数」などといった計算が必要になりかねない。「どれくらい大き(小さ)いか」の情報がすでに失われているので、これらの計算の結果を求めることなどできない(大きいのか小さいのかすらわからなくなる)。

こういった問題を避けるため、次のような工夫をする。

1. 分母が大きすぎるなら小さな数を、分母が小さすぎるなら大きな数を、Cとする。
2. $\sigma(a_i) = \frac{C \times \exp(a_i)}{C \times \sum_j \exp(a_j)} = \frac{\exp(a_i + \ln C)}{\sum_j \exp(a_j + \ln C)}$を代わりに計算する

これを実現する関数softmax_.mのコードは次の通り。

```
function outputs = softmax_( inputs , accuracy)
%SOFTMAX_ ソフトマックス関数
% 適切なバイアスを求め、NaNになることを避ける
if sum(isinf(inputs))
error('要素にinfが含まれているため、計算できません。');
end

%バイアスを求める(再帰処理)
c=compute_c(sort(inputs, 'descend'), length(inputs), 1, 1, true, false, false, accuracy);
if isnan(c)
error('エラー発生');
end
```

```
%ソフトマックス関数を計算
denominator = sum(exp(inputs+log(c)));
outputs=exp(inputs+log(c))./denominator;
end


function c=compute_c(sorted, cnt, c, b, prev_is_inf, has_been_suppressed_getting_smaller,
has_been_suppressed_getting_bigger, accuracy)
if isnan(c)
return;
end
sam=0;
for i=1:cnt
sam=sam+exp(sorted(i)+log(c));
if isinf(sam)
if prev_is_inf%前回の再帰が無限大により起こった場合
if c/(2^(b+1))==0 %cがすでに小さすぎるなら
b=b-1;%cの縮小は抑えざるを得ない
if has_been_suppressed_getting_smaller %が、縮小の抑制の履歴があるなら
disp('softmax_の入力に大きすぎる成分があります。') ;%振動してしまうので止める。
c=NaN;
return; % error関数を使うと、再帰のためエラーメッセージが大量に生じるから避ける
end
has_been_suppressed_getting_smaller=true;
else
b=b+1;%「ずっと大きすぎるまま」なのでcの縮小を加速
end
else
b=b-1;%前回の再帰が0により起こった場合、「行き過ぎた」ので慎重に
end
c=c/(2^b);
c=compute_c(sorted, cnt, c, b, true, has_been_suppressed_getting_smaller,
has_been_suppressed_getting_bigger, accuracy);
end
```

if exp(sorted(i))==0

if i~=1 && 5e-324*accuracy<sum(exp(sorted))

return;

else

if not(prev_is_inf)%前回の再帰が0により起こった場合

if isinf(c*(2^b+1)) %cがすでに大きすぎるなら

b=b-1; %cの拡大は抑えざるを得ない

if has_beem_suppressed_getting_bigger %が、拡大の抑制の履歴があるなら

disp('softmax_の入力に大きすぎる成分があります。') ;%振動してしまうので止める。

c='error';

return;

end

has_been_suppressed_getting_bigger=true;

else

b=b+1;%「ずっと小さすぎるまま」なのでcの拡大を加速

end

else

b=b-1;%前回の再帰が無限大により起こった場合、「行き過ぎた」ので慎重に

end

c=c*(2^b);

c=compute_c(sorted, cnt, c, b, false, has_been_suppressed_getting_smaller,
has_been_suppressed_getting_bigger, accuracy);

end

end

end

end

```
org=0;
mine=0;
for i=1:10
    disp(i+"回目");
    vec=rand(1,10);
tic;
    softmax(vec.')%matlab標準
org=org+toc;
tic;
    softmax_(vec, 100)%自作
mine=mine+toc;
```

```
end
```

```
1回目
ans =
    0.1157
    0.0804
    0.0782
    0.0785
    0.1291
    0.1518
    0.0860
    0.0754
    0.0763
    0.1286

ans = フィールドをもつ struct:
        c: 1
    result: [0.1157 0.0804 0.0782 0.0785 0.1291 0.1518 0.0860 0.0754 0.0763 0.1286]
2回目
ans =
    0.0804
    0.1327
    0.0925
    0.1151
    0.0901
    0.0721
    0.1245
    0.0632
    0.1321
    0.0974

ans = フィールドをもつ struct:
        c: 1
    result: [0.0804 0.1327 0.0925 0.1151 0.0901 0.0721 0.1245 0.0632 0.1321 0.0974]
3回目
ans =
    0.1164
    0.0566
    0.1097
    0.1494
    0.1326
    0.1355
    0.0648
    0.0721
    0.0901
    0.0729

ans = フィールドをもつ struct:
        c: 1
    result: [0.1164 0.0566 0.1097 0.1494 0.1326 0.1355 0.0648 0.0721 0.0901 0.0729]
4回目
ans =
    0.1420
    0.0920
    0.1170
    0.0673
    0.1271
    0.1069
    0.0910
    0.0904
    0.1004
    0.0660

ans = フィールドをもつ struct:
        c: 1
```

```
    result: [0.1420 0.0920 0.1170 0.0673 0.1271 0.1069 0.0910 0.0904 0.1004 0.0660]
5回目
ans =
    0.1150
    0.0690
    0.0971
    0.1272
    0.0765
    0.0895
    0.0634
    0.0990
    0.1513
    0.1120

ans = フィールドをもつ struct:
        c: 1
    result: [0.1150 0.0690 0.0971 0.1272 0.0765 0.0895 0.0634 0.0990 0.1513 0.1120]
6回目
ans =
    0.0833
    0.1029
    0.0901
    0.0887
    0.1141
    0.1314
    0.0903
    0.0738
    0.1129
    0.1124

ans = フィールドをもつ struct:
        c: 1
    result: [0.0833 0.1029 0.0901 0.0887 0.1141 0.1314 0.0903 0.0738 0.1129 0.1124]
7回目
ans =
    0.1167
    0.0816
    0.0988
    0.0707
    0.1486
    0.0700
    0.0841
    0.0829
    0.1029
    0.1436

ans = フィールドをもつ struct:
        c: 1
    result: [0.1167 0.0816 0.0988 0.0707 0.1486 0.0700 0.0841 0.0829 0.1029 0.1436]
8回目
ans =
    0.0735
    0.1515
    0.0572
    0.1063
    0.1017
    0.1239
    0.0951
    0.0945
    0.1042
    0.0921

ans = フィールドをもつ struct:
        c: 1
    result: [0.0735 0.1515 0.0572 0.1063 0.1017 0.1239 0.0951 0.0945 0.1042 0.0921]
```

```
9回目
ans =
    0.0670
    0.1284
    0.0841
    0.1228
    0.1125
    0.1339
    0.0818
    0.1177
    0.0653
    0.0866

ans = フィールドをもつ struct:
        c: 1
    result: [0.0670 0.1284 0.0841 0.1228 0.1125 0.1339 0.0818 0.1177 0.0653 0.0866]
10回目
ans =
    0.1300
    0.1499
    0.0947
    0.0880
    0.1181
    0.0640
    0.0706
    0.1237
    0.0882
    0.0728

ans = フィールドをもつ struct:
        c: 1
    result: [0.1300 0.1499 0.0947 0.0880 0.1181 0.0640 0.0706 0.1237 0.0882 0.0728]
```

```
disp("matlab標準関数の所要時間:"+org);
```

```
matlab標準関数の所要時間:0.27994
```

```
disp("自作関数の所要時間:"+mine);
```

```
自作関数の所要時間:0.33569
```

```
disp('エラー時の比較');
```

```
エラー時の比較
```

```
vec=[16,32,64,128,256,512,1024,2058,4096];
softmax(vec.')%matlab標準
```

```
ans =
     0
     0
     0
     0
     0
     0
     0
     0
     1
```

```
softmax_(vec, 100)%自作 精度が失われたときにエラーを返す機能を付けた
```

```
softmax_の入力に大きすぎる成分があります。
エラー: softmax_ (line 11)
エラー発生
```

matlabで標準で用意されているソフトマックス関数softmaxよりも高速かつ高機能の物を自作できたのは著者の誇りだ。

自作したソフトマックス関数を用いた、分類用多層パーセプトロンのクラスClassification1.mは次の通り。

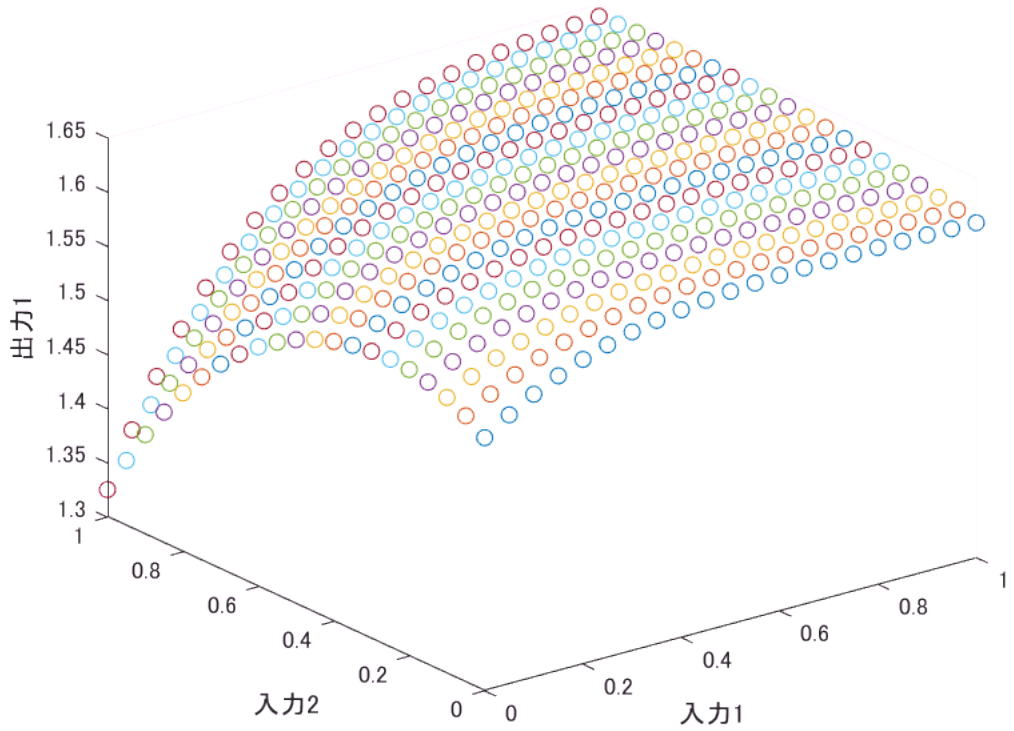
```
classdef Classification1 < Regression1

%CLASSOFOCATION1 分類問題用の多重パーセプトロン

% Regression1の子クラス。MLP7の孫クラス


methods

function this=Classification1(varargin)

this@Regression1(varargin{:});

end

function outputs=run(this,inputs)

dots=run@Regression1(this, inputs);%Regression1の出力は「出力層で活性化関数を通す前」といえる。

outputs=softmax_(dots, 100).result;%それをソフトマックス関数に入れればよい

end

end

end
```

```
regression1=Regression1(2,3,3,2,2,1);
classification1=Classification1(2,3,3,2,2,1);
w=10.*regression1.getW3().*rand(size(regression1.getW3()))-5
```

```
w =

(:,:,1) =

   -0.4100    2.7334   -0.4586
    0.9375    3.6409    4.5512
   -3.4770   -2.7635    2.4770
       NaN       NaN       NaN


(:,:,2) =

    0.9678    0.4509       NaN
   -3.4299   -0.3975       NaN
    1.1063    2.9166       NaN
```

```
    3.9943    0.1802       NaN

(:,:,3) =

   -2.5410       NaN       NaN
    0.6112       NaN       NaN
    3.6840       NaN       NaN
       NaN       NaN       NaN
```

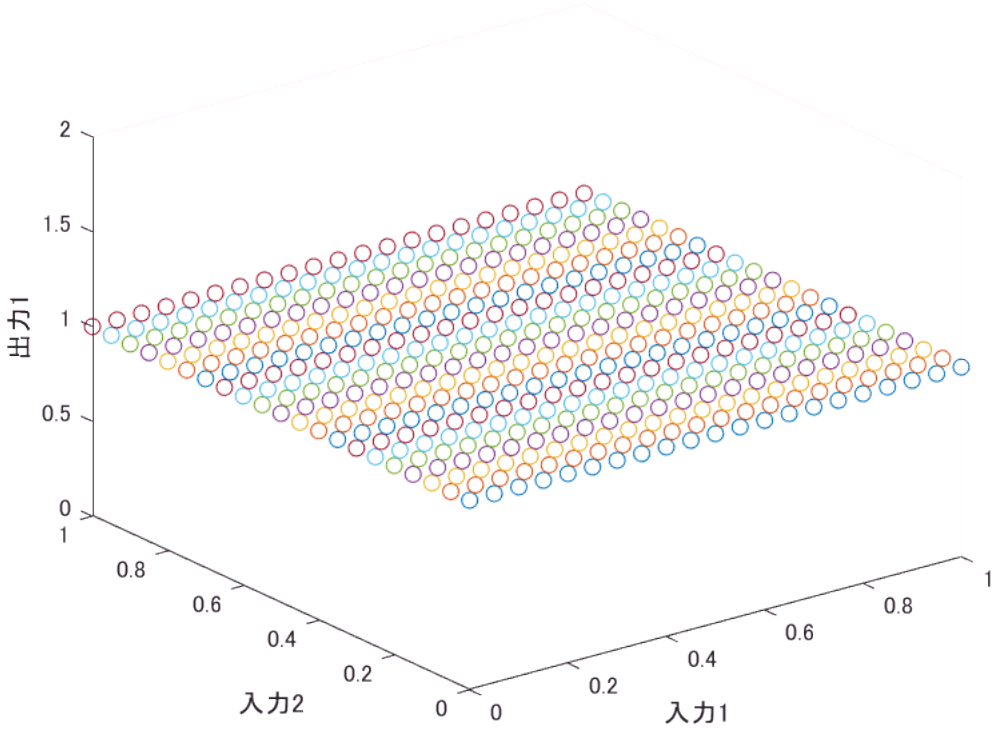
```
regression1=regression1.setW3(w);
classification1=classification1.setW3(w);
x=[0:0.05:1];
y=[0:0.05:1];
tic;
figure
hold on
for i=x
    for j=y
        scatter3(i, j, regression1.run([i j]));
    end
end
xlabel("入力1");
ylabel("入力2");
zlabel("出力1");
view(3);

hold off
```




```
disp("regression1の所要時間:"+toc);
```

```
regression1の所要時間:45.4288
```

```
tic;
figure
hold on
for i=x
    for j=y
        scatter3(i, j, classification1.run([i j]));
    end
end
xlabel("入力1");
ylabel("入力2");
zlabel("出力1");
view(3);

hold off
```




```
disp("classification1の所要時間:"+toc);
```

```
classification1の所要時間:59.938
```

出力が1つしかないため、ソフトマックス関数の定義より、常に1となる。

2-10. バッチに対応する

ニューラルネットワークの教科書通りにpythonで多層パーセプトロンを構築する場合、入力ベクトルを行列にまとめることで高速化が期待できるそうだ。(但し、行列演算は行わず、各入力ベクトルと垂直な方向にfor文を回す)

これはつまり、「複数のデータを束ねたもの(バッチ)を入力して、それぞれの出力を同時に求める」ことである。

これを可能にするBMLP1.mは、MLP7を継承し、runメソッドをオーバライドすることで実現する。

ここで、実際にBMLP1を動かし、高速化に成功したか検討してみよう。

2-11. 多層パーセプトロンにP(DCA)+サイクルで学習させるには

さて、2-10節までで多層パーセプトロンの基本的な設計は完了したといってよい。

ニューラルネットワークを「P(DCA)+サイクル」に例えるなら、そのうちのP(計画)がすでに終了した。

ここで、P(DCA)+サイクルとは、図2.11.1(b)のようにPは一度のみ行った後はDCAを繰り返すようなローテーションのことである。

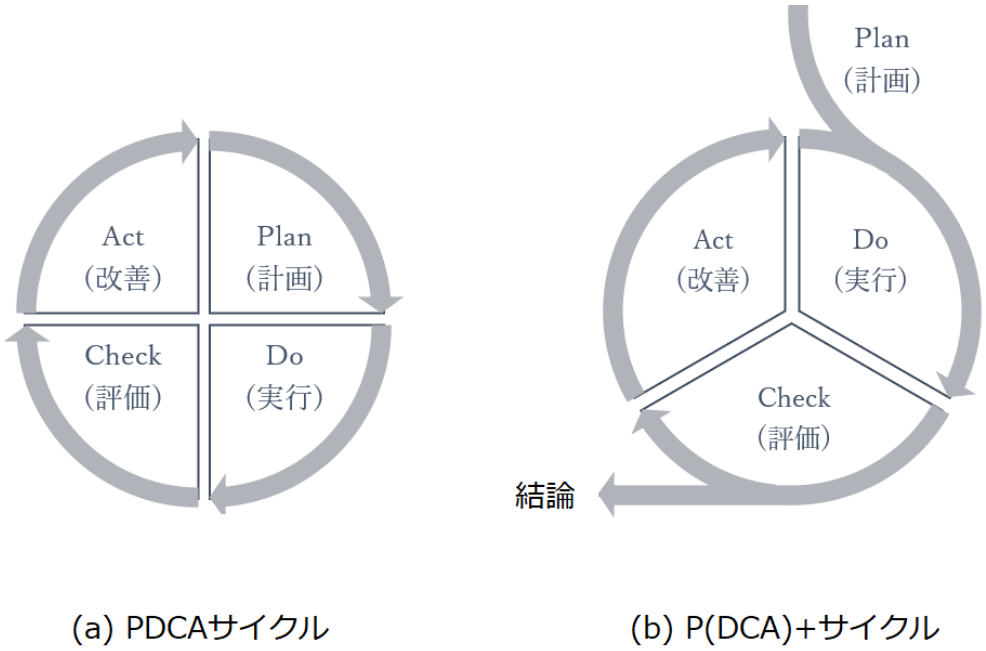


図2.11.1 PDCAサイクルとP(DCA)+サイクルの比較

また、P(DCA)+サイクルは正規表現で厳密にかくなら

P(DCA)+ではなくPDC(ADC)*である。但し、これでは名前としてあまりにも冗長であると考え、前者の名前とした。

2-11節以降は、多層パーセプトロンに学習能力を与えることでニューラルネットワークを作っていく。

大まかな流れは以下の1~3を繰り返すといったものである。

1. データを入力し、多層パーセプトロンに出力させる(D, 2-12節)
2. 「出力と答えの間のギャップの大きさ」(損失関数)の勾配を求める(C, 2-13節)
3. 重みを勾配の下り方向に少しだけ変化させる(A, 2-13節)

2-12

節以降では、データをベクトル $\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \\ y \end{pmatrix}$ と考える。またニューラルネットワークに $\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$ を与えたとき、yの値を予測できるようにすることを目標とする。

2-12. ミニバッチ(P(DCA)+の「D」)

学習時に入力するデータのことを教師データという。

例えば手書き画像 $\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$ から数字 y を認識するニューラルネットワークを作りたいのであれば、数万枚ほどの大量の画像を用いる必要がある。

但し、何万枚もの画像を1つのバッチにして入力するのは流石に難しい。そこで、ランダムに100枚ほど選び出し、それを学習に用いるという方法をとる。ここで選ばれた100枚(ほど)を1つのバッチにしたもの

$$\begin{pmatrix} x_{1,\text{画像}1} & x_{1,\text{画像}2} & \cdots & x_{1,\text{画像}100} \\ x_{2,\text{画像}1} & x_{2,\text{画像}2} & \cdots & x_{2,\text{画像}100} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,\text{画像}1} & x_{n,\text{画像}2} & \cdots & x_{n,\text{画像}100} \end{pmatrix}$$

を「ミニバッチ」という。

2-13. 損失関数と勾配(P(DCA)+の「C」および「A」)

損失関数とは、「予測(「D」での出力)がどれだけ正解(教師データ $(y_{\text{画像}1} \ y_{\text{画像}2} \ \cdots \ y_{\text{画像}100})$)とかけ離れているか」を表す指標である。

最小2乗法やクロスエントロピー誤差などの関数を各データについて適用し、その総和を適用することで求めることができる。

ここで、東西南北(緯度経度)の方向を重み、高度方向が損失関数になるような3次元グラフを考えてみよう。

重み配列(重みの値とその組み合わせ)によって出力が変化するということは、重み配列によって損失関数が変化する。

つまり、3次元グラフは緯度経度によって高度が異なる、(複雑な)坂道を(たくさん)持っている。

このようなグラフ上のどこかの点にボールを置くと、坂道を下っていくだろう。つまり、ボールは高度の「少しでも」小さいような緯度経度へ移動する。

このような理屈で、「損失関数を小さくするような重み配列を探す」ことができるのである。損失関数が小さいということは、予測と正解があまりかけ離れていない、すなわちニューラルネットワークの正解率が高くなったということであるわけだ。

ここで、ボールが坂道を下っていくことを、どのようにして再現しようか。代表的な手法2つを2-13-1節や2-13-2節で取り上げる。

2-13-1. 差分法を用いた素朴な方法

ボールが坂道を下っていくのを素朴に再現するには、次の手順を繰り返す。

1. その地点におけるグラフの傾きを(中心)差分法で求める
2. 各層ごとに、重み行列を「重み行列−微小量(スカラ)×求めた傾き(配列)」に更新する

このアルゴリズムを勾配降下法という。

但し、勾配降下法にも欠点はある。

- 谷のような地形(極小値)が2つ以上ある場合、どの極小値にたどり着くかわからない。
- 「微小量」とはいったいいくつなのか。(小さすぎると時間がかかるし、大きすぎると振動が起こって結局時間がかかる)

前者の問題は、「極小値の最小値は最小値だろう」とすることで解決できる。具体的には、ランダムで「ボールを置く地点」を変えることにより、なるべくすべての極小値を見つけ、それらの中での最小値を採用するというものだ。この手法を「確率的勾配降下法」という。

後者の問題については、人間が試しながら最適解を見つけるほかないようだ。こういった、ニューラルネットワークが学習しないパラメータのことを「ハイパーパラメータ」という。

2-13-2. 誤差逆伝播法を用いた高速な方法

突然だが、ここに「微分の似非定義式」を示す。※

$f(x)$に$x = a$を代入したときに微分の値を$\frac{\mathrm{d}}{\mathrm{d}x} f(x = a)$と書くとして、

$$\frac{\mathrm{d}}{\mathrm{d}x} f(x = a) = \lim_{h \to 0} \frac{f(a + h) - f(a - h)}{2h} \cdots ① = \frac{\mathrm{d}}{\mathrm{d}t_1} f(x = a) \frac{\mathrm{d}t_1}{\mathrm{d}t_2} \frac{\mathrm{d}t_2}{\mathrm{d}t_3} \cdots \frac{\mathrm{d}t_n}{\mathrm{d}x} \cdots ②$$

2-13-1節の手法では、①を計算することで損失関数の微分を求めている。

こうすると、具体的な式を知らなくても、入力から出力を得ることさえできれば微分(に近いこと)ができるというメリットがある反面、出力を2通り求めなくてはならないため、効率が悪い。

一方、これから紹介する誤差逆伝搬法では②を計算する。

こうすると、関数の式が分かっていれば、微分の式$\frac{\partial \text{損失関数}}{\partial \text{重み行列}}$自体を求めることができるため、効率が良い。

2-13-2-1. 回帰問題の場合の$\frac{\partial \text{損失関数}}{\partial \text{重み行列(出力層)}}$

今、例として出力層の重み行列を、$\begin{pmatrix} w_{0,1} & w_{0,2} & \cdots & w_{0,finP} \\ w_{1,1} & w_{1,2} & \cdots & w_{1,finP} \\ \vdots & \vdots & \ddots & \vdots \\ w_{finInput,1} & w_{finInput,2} & \cdots & w_{finInput,finP} \end{pmatrix} = \begin{pmatrix} \boldsymbol{w}_1 & \boldsymbol{w}_2 & \cdots & \boldsymbol{w}_{finP} \end{pmatrix} = \boldsymbol{W}$ としよう。損失関数をLとすると、回帰問題の場合、(損失関数の、取り出してきた列ベクトルに対する)傾きは

$$\frac{\partial L}{\partial \begin{pmatrix} \boldsymbol{w}_1 & \boldsymbol{w}_2 & \cdots & \boldsymbol{w}_{finP} \end{pmatrix}} = \begin{pmatrix} \frac{\partial L}{\partial \boldsymbol{w}_1} & \frac{\partial L}{\partial \boldsymbol{w}_2} & \cdots & \frac{\partial L}{\partial \boldsymbol{w}_{finP}} \end{pmatrix} = \begin{pmatrix} \frac{\partial L}{\partial w_{0,1}} & \frac{\partial L}{\partial w_{0,2}} & \cdots & \frac{\partial L}{\partial w_{0,finP}} \\ \frac{\partial L}{\partial w_{1,1}} & \frac{\partial L}{\partial w_{1,2}} & \cdots & \frac{\partial L}{\partial w_{1,finP}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial L}{\partial w_{finInput,1}} & \frac{\partial L}{\partial w_{finInput,2}} & \cdots & \frac{\partial L}{\partial w_{finInput,finP}} \end{pmatrix}$$

ここで、損失関数を最小2乗法によるものとしてみよう。

$$L = \frac{1}{2} \sum_{\text{出力層の各パーセプトロン}} (\text{出力} - \text{正解})^2$$

このとき、

$$\frac{\partial L}{\partial \begin{pmatrix} \boldsymbol{w}_1 & \boldsymbol{w}_2 & \cdots & \boldsymbol{w}_{finP} \end{pmatrix}}$$

=

$$\frac{\partial L}{\partial \begin{pmatrix} \boldsymbol{w}_1 \bullet \boldsymbol{x} & \boldsymbol{w}_2 \bullet \boldsymbol{x} & \cdots & \boldsymbol{w}_{finP} \bullet \boldsymbol{x} \end{pmatrix}} \frac{\partial \begin{pmatrix} \boldsymbol{w}_1 \bullet \boldsymbol{x} & \boldsymbol{w}_2 \bullet \boldsymbol{x} & \cdots & \boldsymbol{w}_{finP} \bullet \boldsymbol{x} \end{pmatrix}}{\partial \begin{pmatrix} \boldsymbol{w}_1 & \boldsymbol{w}_2 & \cdots & \boldsymbol{w}_{finP} \end{pmatrix}}$$

今回は重みと入力の内積$\boldsymbol{w}_i \bullet \boldsymbol{x}$がそのまま出力$y_i$となる。

$$\frac{\partial L}{\partial \begin{pmatrix} y_1 & y_1 & \cdots & y_{finP} \end{pmatrix}} \frac{\partial \begin{pmatrix} y_1 & y_1 & \cdots & y_{finP} \end{pmatrix}}{\partial \begin{pmatrix} \boldsymbol{w}_1 & \boldsymbol{w}_2 & \cdots & \boldsymbol{w}_{finP} \end{pmatrix}}$$

$$\begin{pmatrix} \frac{\partial L}{\partial y_1} & \frac{\partial L}{\partial y_2} & \cdots & \frac{\partial L}{\partial y_{finP}} \end{pmatrix} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{W}}$$

ここで、正解を$\boldsymbol{t}$とすると、

$L = \frac{1}{2} \sum_i (y_i - t_i)^2$

なので

$$\begin{pmatrix} y_1 - t_1 & y_2 - t_2 & \cdots & y_{finP} - t_{finP} \end{pmatrix} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{W}}$$

$$= (\boldsymbol{y} - \boldsymbol{t}) \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{W}}$$

となる。

次に$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{W}}$を求めよう。

その前に、合成微分の公式について述べる。r行c列の行列を$X(r \times c)$とかくことにする。

今、

$$\left\{\frac{\partial S(1\times 1)}{\partial \boldsymbol{M}(r\times c)}\right\}(r\times c)\ (r\geq 2,\ c\neq 1, c\neq r)$$

を考えよう。

スカラの微分公式どおり

$$\frac{\partial S}{\partial \boldsymbol{M}}=\frac{\partial S}{\partial \boldsymbol{R}}\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{M}}$$

と書きたいところだが、行列の微分では実際そうはいかない。

例えば$\boldsymbol{R}(1\times c)\ (c\geq 1)$

とすると、

$$\left\{\frac{\partial S}{\partial \boldsymbol{M}}\right\}(r\times c)=\left\{\frac{\partial S}{\partial \boldsymbol{R}}\right\}(1\times c)\left\{\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{M}}\right\}(?_1\times ?_2)$$

となってしまい、左辺と右辺の形状が一致しないという矛盾が出る。

行列の積が成立するとき、左の行列の列数と右の行列の行数が一致する（サイズ表記を（行数×列数）の形で行ったとき、内側が一致する「内側の一致」）ことより、$?_1=c$はいえるが、$?_2$をいかにしても右辺の形状$1\times ?_2$を$r\times c$に一致させることはできない。

そこで、$\frac{\partial S}{\partial \boldsymbol{R}}$と$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{M}}$の演算の順を入れ替えてみよう。すなわち、

$$\left\{\frac{\partial S}{\partial \boldsymbol{M}}\right\}(r\times c)=\left\{\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{M}}\right\}(?_1\times ?_2)\left\{\frac{\partial S}{\partial \boldsymbol{R}}\right\}(1\times c)$$

とする。内側の一致より$?_2=1$、また外側の行数、列数がそのまま積の行数列数になる「外側の採用」より$?_1=r$

なので

$$\left\{\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{M}}\right\}(r\times 1)$$

がいえる。

$$\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{M}}=\frac{\partial \boldsymbol{R}}{\partial H}\frac{\partial H}{\partial \boldsymbol{M}}\ \text{または}\ \frac{\partial H}{\partial \boldsymbol{M}}\frac{\partial \boldsymbol{R}}{\partial H}$$

と置けないか打診してみよう。前の行列の行数がrである（一般的に言えば、1でない方の大きさrが$\boldsymbol{M}$にもみられる）という条件より、

$$\left\{\frac{\partial \boldsymbol{R}}{\partial \boldsymbol{M}}\right\}(r\times 1)=\left\{\frac{\partial H}{\partial \boldsymbol{M}}\right\}(r\times c)\left\{\frac{\partial \boldsymbol{R}^T}{\partial H}\right\}(c\times 1)$$

と決まる。

このように、行列の合成微分を行う際には、形状を手掛かりに積の順序及び転置の有無を考える必要があるため、注意する必要がある。今後これをHの消極的条件と呼ぶことにする。

Hの消極的条件は行列の積の順序や転置の有無を(ほぼ)一意に特定するものに過ぎないので、積極的に用いる必要はない。そもそも積の計算が正常に行われた場合は、消極的条件によっても同じ計算式が示されるか、あるいは一意に特定できないかのいずれかであるから、原理的に役立たない。したがって、まずは素直に計算してみて、

- 途中で積が定義できなくなった
- 被微分変数と微分変数が共にベクトルで、長さが異なる
- ベクトルを行列で微分する、あるいは行列をベクトルで微分する必要が出てきた

などの問題が発生した時、初めて消極的条件を吟味するのが効率的だろう。

また、消極的条件によって積の順序、転置の有無が必ず特定できる条件をHの積極的条件と呼ぼう。

積極的条件については、研究が間に合わなかったため執筆を見合わせる。

H の消極的条件を確認する。

$$\left\{\frac{\partial L}{\partial \boldsymbol{W}}\right\}\left((finInput+1)\times finP\right)=\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{W}}\left((finInput+1)\times 1\right)\left\{\left(\boldsymbol{y}-\boldsymbol{t}\right)\right\}\left(1\times finP\right)$$

$$\left\{\frac{\partial \boldsymbol{y}\left(1\times finP\right)}{\partial \boldsymbol{W}\left((finInput+1)\times finP\right)}\right\}\left((finInput+1)\times 1\right)=\left\{\frac{\partial H}{\partial \boldsymbol{W}}\right\}\left((finInput+1)\times finP\right)\left\{\frac{\partial y}{\partial H}^{T}\right\}\left(finP\times 1\right)$$

よって

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{W}}$$

$$= \frac{\partial H}{\partial \boldsymbol{W}} \frac{\partial \boldsymbol{y}}{\partial H}^{T}$$

$$= \begin{pmatrix} \frac{\partial H}{\partial w_{0,1}} & \frac{\partial H}{\partial w_{0,2}} & \cdots & \frac{\partial H}{\partial w_{0,finP}} \\ \frac{\partial H}{\partial w_{1,1}} & \frac{\partial H}{\partial w_{1,2}} & \cdots & \frac{\partial H}{\partial w_{1,finP}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial H}{\partial w_{finInput,1}} & \frac{\partial H}{\partial w_{finInput,2}} & \cdots & \frac{\partial H}{\partial w_{finInput,finP}} \end{pmatrix} \begin{pmatrix} \frac{\partial y_1}{\partial H} \\ \frac{\partial y_2}{\partial H} \\ \vdots \\ \frac{\partial y_{finP}}{\partial H} \end{pmatrix}$$

$$= \begin{pmatrix} \frac{\partial H}{\partial w_{0,1}} & \frac{\partial H}{\partial w_{0,2}} & \cdots & \frac{\partial H}{\partial w_{0,finP}} \\ \frac{\partial H}{\partial w_{1,1}} & \frac{\partial H}{\partial w_{1,2}} & \cdots & \frac{\partial H}{\partial w_{1,finP}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial H}{\partial w_{finInput,1}} & \frac{\partial H}{\partial w_{finInput,2}} & \cdots & \frac{\partial H}{\partial w_{finInput,finP}} \end{pmatrix} \begin{pmatrix} \frac{\partial}{\partial H} \sum_{i=0}^{finInput} w_{i,1} x_i \\ \frac{\partial}{\partial H} \sum_{i=0}^{finInput} w_{i,2} x_i \\ \vdots \\ \frac{\partial}{\partial H} \sum_{i=0}^{finInput} w_{i,finP} x_i \end{pmatrix}$$

$$= \begin{pmatrix} \sum_{j=1}^{finP} \left(\frac{\partial H}{\partial w_{0,j}} \frac{\partial}{\partial H} \sum_{i=0}^{finInput} w_{i,j} x_i \right) \\ \sum_{j=1}^{finP} \left(\frac{\partial H}{\partial w_{1,j}} \frac{\partial}{\partial H} \sum_{i=0}^{finInput} w_{i,j} x_i \right) \\ \vdots \\ \sum_{j=1}^{finP} \left(\frac{\partial H}{\partial w_{finInput,j}} \frac{\partial}{\partial H} \sum_{i=0}^{finInput} w_{i,j} x_i \right) \end{pmatrix}$$

$$- \begin{pmatrix} \sum_{j=1}^{finP} \left(\frac{\partial}{\partial w_{0,j}} \sum_{i=0}^{finInput} w_{i,j} x_i \right) \\ \sum_{j=1}^{finP} \left(\frac{\partial}{\partial w_{\cdot}} \sum_{i=0}^{finInput} w_{i,j} x_i \right) \end{pmatrix}$$

よって、係数を無視すれば、

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{W}} = \boldsymbol{x}^{T}$$

なので、

$$\frac{\partial L}{\partial \boldsymbol{W}} = \boldsymbol{x}^{T} \left(\boldsymbol{y} - \boldsymbol{t} \right)$$

2-13-2-2. 分類問題の場合の $\frac{\partial 損失関数}{\partial 重み行列(出力層)}$

分類問題の場合[10][11]は、ソフトマックス関数を活性化関数としているため、傾きは

$$\frac{\partial L}{\partial \left(\boldsymbol{w}_1 \quad \boldsymbol{w}_2 \quad \cdots \quad \boldsymbol{w}_{finP}\right)}$$

$$= \frac{\partial L}{\partial \,\mathrm{softmax}\left(\boldsymbol{w}_1 \bullet \boldsymbol{x} \quad \boldsymbol{w}_2 \bullet \boldsymbol{x} \quad \cdots \quad \boldsymbol{w}_{finP} \bullet \boldsymbol{x}\right)} \frac{\partial \mathrm{softmax}\left(\boldsymbol{w}_1 \bullet \boldsymbol{x} \quad \boldsymbol{w}_2 \bullet \boldsymbol{x} \quad \cdots \quad \boldsymbol{w}_{finP} \bullet \boldsymbol{x}\right)}{\partial \left(\boldsymbol{w}_1 \bullet \boldsymbol{x} \quad \boldsymbol{w}_2 \bullet \boldsymbol{x} \quad \cdots \quad \boldsymbol{w}_{finP} \bullet \boldsymbol{x}\right)} \frac{\partial \left(\boldsymbol{w}_1 \bullet \boldsymbol{x} \quad \boldsymbol{w}_2 \bullet \boldsymbol{x}}{\partial \left(\boldsymbol{w}_1 \quad \boldsymbol{w}_2 \quad ..\right.}$$

いま

$\left(\boldsymbol{w}_1 \bullet \boldsymbol{x} \quad \boldsymbol{w}_2 \bullet \boldsymbol{x} \quad \cdots \quad \boldsymbol{w}_{finP} \bullet \boldsymbol{x}\right) = \left(a_1 \quad a_2 \quad \cdots \quad a_{finP}\right) = \boldsymbol{a}$

$\mathrm{softmax}\left(a_1 \quad a_2 \quad \cdots \quad a_{finP}\right) = \left(y_1 \quad y_2 \quad \cdots \quad y_{finP}\right)$ つまり $\mathrm{softmax}\,\boldsymbol{a} = \boldsymbol{y}$

とすると

$$\left(\frac{\partial L}{\partial y_1} \quad \frac{\partial L}{\partial y_2} \quad \cdots \quad \frac{\partial L}{\partial y_{finP}}\right) \frac{\partial \mathrm{softmax}\left(\boldsymbol{a}\right)}{\partial \boldsymbol{a}} \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{W}}$$

となる。ここで

$$\mathrm{softmax}\left(\boldsymbol{a}\right) = \frac{\exp\left(\boldsymbol{a}\right) \quad \leftarrow \text{ベクトル}}{\Sigma \exp\left(\boldsymbol{a}\right) \quad \leftarrow \text{スカラ}}$$

また

$$\frac{\partial y_i}{\partial a_j} = \begin{cases} y_i\left(1 - y_i\right) & i = j \\ -y_i y_j & i \neq j \end{cases}$$

より

$$\frac{\partial y_i}{\partial \boldsymbol{a}}$$

$$= \left(-y_i y_1 \quad -y_i y_2 \quad \cdots \quad -y_i y_{i-1} \quad y_i\left(1 - y_i\right) \quad -y_i y_{i+1} \quad \cdots \quad -y_i y_{finP}\right)$$

$$= -y_i \left(y_1 \quad y_2 \quad \cdots \quad y_{i-1} \quad y_i - 1 \quad y_{i+1} \quad \cdots \quad y_{finP}\right)$$

$$= -y_i \left\{\left(y_1 \quad y_2 \quad \cdots \quad y_{i-1} \quad y_i \quad y_{i+1} \quad \cdots \quad y_{finP}\right) - \left(0 \quad 0 \quad \cdots \quad 0 \quad 1 \quad 0 \quad \cdots \quad 0\right)\right\}$$

$$= \left(0 \quad 0 \quad \cdots \quad 0 \quad y_i \quad 0 \quad \cdots \quad 0\right) - y_i \boldsymbol{y}$$

n 次単位行列の i 行目を $\boldsymbol{I@n}_i$ とすると、これは

$y_i \boldsymbol{I@finP}_i - y_i \boldsymbol{y}$

$= y_i\left(\boldsymbol{I@finP}_i - \boldsymbol{y}\right)$

なので

となる。

実は、ソフトマックス関数には、「相性の良い」損失関数が存在する。

それが「クロスエントロピー誤差関数」

$$L = -\sum_{\text{出力層の各パーセプトロン}} t \times \ln \text{パーセプトロンの出力} \left(\text{但し } t = \begin{cases} 1 & (\text{正解})\text{番目のパーセプトロンのとき} \\ 0 & (\text{不正解})\text{番目の時} \end{cases}\right)$$

だ。このtを正解ラベルという。

別の表現をするなら

$$L = -\ln\left(\left(\text{正解}\right)\text{番目のパーセプトロンの出力}\right)$$

となる。

(

蛇足:

バッチ対応版では、次のようにするとよい。

$$L = -\sum_{\text{バッチの各々}} \ln\left(\left(\text{正解}\right)\text{番目のパーセプトロンの出力}\right)$$

)

このようなLを考えるとき、$\frac{\partial L}{\partial \boldsymbol{y}}$を求めてみよう。(まずは、バッチ非対応で、入力は$1$データとする)

簡単に考えるため、

$$\boldsymbol{y} = \begin{pmatrix} y_{\text{正解}} & y_{\text{不正解}} \end{pmatrix}$$

のように、不正解の成分を一つにまとめてしまおう。(どうせ係数0より、明らかに不正解に対応する変数で偏微分した結果は0なので)

$$\begin{aligned}
&\frac{\partial L}{\partial \boldsymbol{y}} \\
&= -\frac{\partial\left(1 \times \ln y_{\text{正解}} + 0 \times \ln y_{\text{不正解}}\right)}{\partial\begin{pmatrix} y_{\text{正解}} & y_{\text{不正解}} \end{pmatrix}} \\
&= \begin{pmatrix} -\frac{\partial\left(1 \times \ln y_{\text{正解}} + 0 \times \ln y_{\text{不正解}}\right)}{\partial y_{\text{正解}}} & -\frac{\partial\left(1 \times \ln y_{\text{正解}} + 0 \times \ln y_{\text{不正解}}\right)}{\partial y_{\text{不正解}}} \end{pmatrix} \\
&= \begin{pmatrix} -\frac{1}{y_{\text{正解}}} & 0 \end{pmatrix} \\
&= \begin{pmatrix} -\frac{0}{y_1} & -\frac{0}{y_2} & \cdots & -\frac{0}{y_{\text{正解}-1}} & -\frac{1}{y_{\text{正解}}} & -\frac{0}{y_{\text{正解}+1}} & \cdots & -\frac{0}{y_{finP}} \end{pmatrix}
\end{aligned}$$

よって、

$$
\begin{aligned}
&\frac{\partial L}{\partial \begin{pmatrix} \boldsymbol{w}_1 & \boldsymbol{w}_2 & \cdots & \boldsymbol{w}_{finP} \end{pmatrix}} \\
&= \left\{ \begin{pmatrix} \frac{\partial L}{\partial y_1} y_1 & \frac{\partial L}{\partial y_2} y_2 & \cdots & \frac{\partial L}{\partial y_{finP}} y_{finP} \end{pmatrix} - \begin{pmatrix} \frac{\partial L}{\partial y_1} & \frac{\partial L}{\partial y_2} & \cdots & \frac{\partial L}{\partial y_{finP}} \end{pmatrix} \boldsymbol{y}^T \boldsymbol{y} \right\} \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{W}} \\
&= \left\{ \begin{pmatrix} -\frac{0 y_1}{y_1} & -\frac{0 y_2}{y_2} & \cdots & -\frac{0 y_{正解-1}}{y_{正解-1}} & -\frac{1 y_{正解}}{y_{正解}} & -\frac{0 y_{正解+1}}{y_{正解+1}} & \cdots & -\frac{0 y_{finP}}{y_{finP}} \end{pmatrix} - \begin{pmatrix} -\frac{0}{y_1} & -\frac{0}{y_2} & \cdots & -\frac{0}{y_{正解-1}} & -\frac{1}{y_{正} \end{pmatrix} \right. \\
&= \left\{ -\begin{pmatrix} 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \end{pmatrix} - \begin{pmatrix} 0 & 0 & \cdots & 0 & -\frac{1}{y_{正解}} & 0 & \cdots & 0 \end{pmatrix} \boldsymbol{y}^T \boldsymbol{y} \right\} \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{W}} \\
&= \left\{ -\left((\text{正解}) \text{番目だけ1それ以外0} \right) - \begin{pmatrix} 0 & 0 & \cdots & 0 & -\frac{1}{y_{正解}} & 0 & \cdots & 0 \end{pmatrix} \boldsymbol{y}^T \boldsymbol{y} \right\} \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{W}} \\
&= \left\{ -\left(\text{正解ラベル「} \boldsymbol{t} \text{」} \right) - \begin{pmatrix} 0 & 0 & \cdots & 0 & -\frac{1}{y_{正解}} & 0 & \cdots & 0 \end{pmatrix} \boldsymbol{y}^T \boldsymbol{y} \right\} \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{W}} \\
&= \left\{ -\boldsymbol{t} + \frac{1}{y_{正解}} \begin{pmatrix} 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \end{pmatrix} \boldsymbol{y}^T \boldsymbol{y} \right\} \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{W}} \\
&= \left\{ -\boldsymbol{t} + \frac{1}{y_{正解}} \begin{pmatrix} y_1 y_{正解} & y_2 y_{正解} & \cdots & y_{正解-1} y_{正解} & y_{正解} y_{正解} & y_{正解+1} y_{正解} & \cdots & y_{finP} y_{正解} \end{pmatrix} \right\} \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{W}} \\
&= \left\{ -\boldsymbol{t} + \frac{y_{正解}}{y_{正解}} \begin{pmatrix} y_1 & y_2 & \cdots & y_{正解-1} & y_{正解} & y_{正解+1} & \cdots & y_{finP} \end{pmatrix} \right\} \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{W}} \\
&= (\boldsymbol{y} - \boldsymbol{t}) \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{W}}
\end{aligned}
$$

$\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{W}}$は$\frac{\partial \boldsymbol{xW}}{\partial \boldsymbol{W}}$であり、2-13-2-1と同様に

$$\frac{\partial L}{\partial \boldsymbol{W}} = \boldsymbol{x}^T (\boldsymbol{y} - \boldsymbol{t})$$

が求められた。

2-13-2-3. $\frac{\partial \text{損失関数}}{\partial \text{重み行列}(\text{出力層以外})}$

2-13-2-1節及び2-13-2-2節から、回帰問題であろうと分類問題であろうと、

$$
\begin{aligned}
&\frac{\partial \text{損失関数}}{\partial \text{出力層の重み行列}} \\
&= (\text{入力列ベクトル})(\text{出力行ベクトル} - \text{正解行ベクトル}) \\
&= (\text{入力ベクトル}) \bullet (\text{出力と正解の差のベクトル})
\end{aligned}
$$

であることが求められた。

出力層以外の重み行列も当然更新が必要で、これを行うために、

$$\frac{\partial \text{損失関数}}{\partial \text{出力層以外の重み行列}}$$

を求める必要がある。今、出力層からi層だけ手前における重み行列を$\boldsymbol{W}_i$とかくことにしよう。例えば出力層の重み行列は$\boldsymbol{W}_0$だ。

また損失関数、入力ベクトル、出力ベクトルなどは2-13-2-1節などと同様とする。

出力層以外の活性化関数(出力層からi層だけ手前におけるものを$\theta_i(\boldsymbol{x}\boldsymbol{W}_i)$)としてはReLUや、シグモイド関数が考えられる。

例えば、出力層の1層だけ手前の重み行列を考えるとするなら、

$$\frac{\partial L}{\partial \boldsymbol{W}_1}$$
$$= \frac{\partial L}{\partial \boldsymbol{W}_0}\frac{\partial \boldsymbol{W}_0}{\partial \theta_1(\boldsymbol{x}\boldsymbol{W}_1)}\frac{\partial \theta_1(\boldsymbol{x}\boldsymbol{W}_1)}{\partial \boldsymbol{x}\boldsymbol{W}_1}\frac{\partial \boldsymbol{x}\boldsymbol{W}_1}{\partial \boldsymbol{W}_1}$$
$$= \boldsymbol{x}^T(\boldsymbol{y}-\boldsymbol{t})\frac{\partial \boldsymbol{W}_0}{\partial \theta_1(\boldsymbol{x}\boldsymbol{W}_1)}\frac{\partial \theta_1(\boldsymbol{x}\boldsymbol{W}_1)}{\partial \boldsymbol{x}\boldsymbol{W}_1}\frac{\partial \boldsymbol{x}\boldsymbol{W}_1}{\partial \boldsymbol{W}_1}$$

θ1をシグモイド関数とする場合

$$\frac{\partial \theta_1(\boldsymbol{x}\boldsymbol{W}_1)}{\partial \boldsymbol{x}\boldsymbol{W}_1}$$
$$= \theta_1(\boldsymbol{x}\boldsymbol{W}_1)(1-\theta_1(\boldsymbol{x}\boldsymbol{W}_1))$$

ここで時間切れ。学習機能の実装に至らなかった

第3章. 画像処理

お断り、本来は、第2章でニューラルネットワークを完成させ、これを用いて処理を行おうと考えていた。

しかし、時間の都合でできなかったため、第2章の内容と無関係に、指示された課題取り組む。

3-1. 課題1

次のコードを、matlabのコマンドラインに打ち込んだ上実行すると、図3.2～3.7を得た。但し原画像D:\daigaku_note\ham.pngの内容は図3.1の通り。

図3.1 原画像

図3.2 1回目のpauseで得られた画像


100
200
300
400
500
600
700
800
100
200
300
400
500
600
700

図3.3 2回目

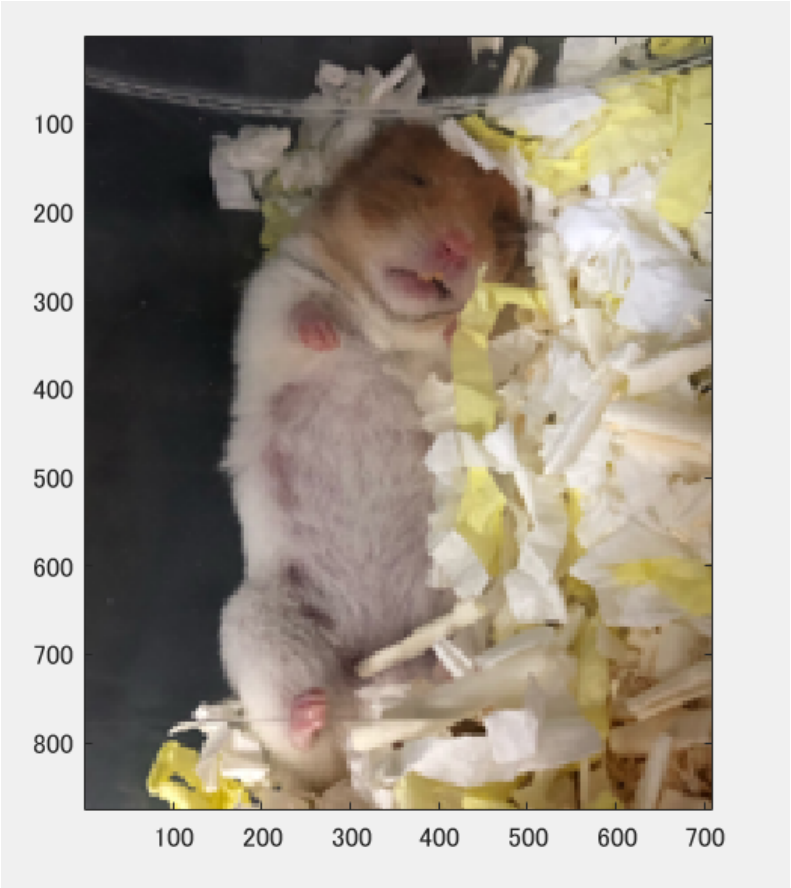

100
200
300
400
500
600
700
800
100
200
300
400
500
600
700

図3.4 3回目

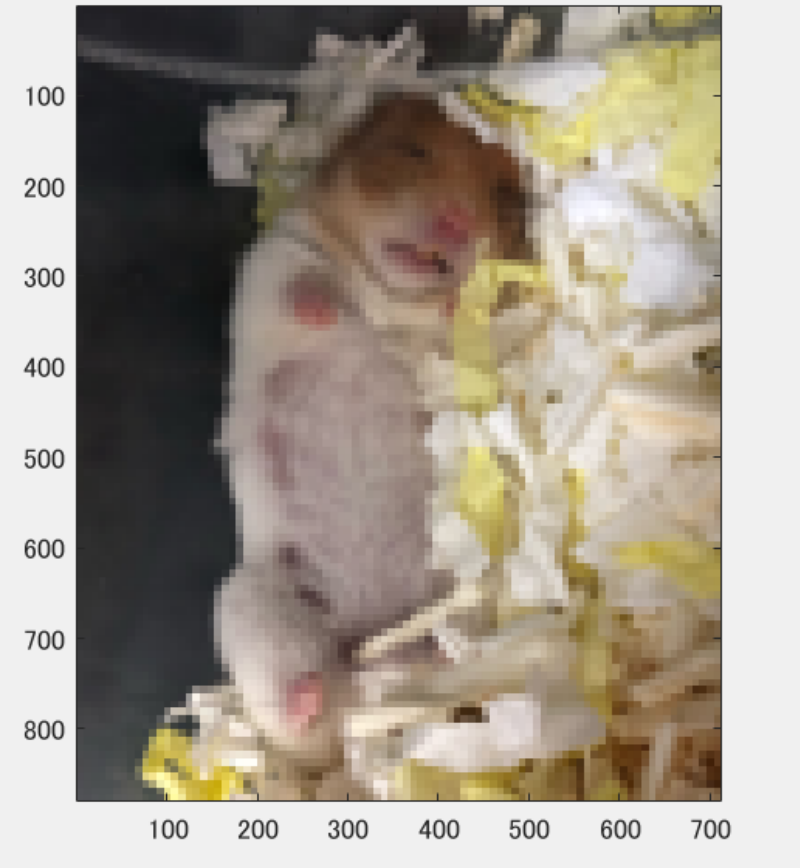


図3.5 4回目

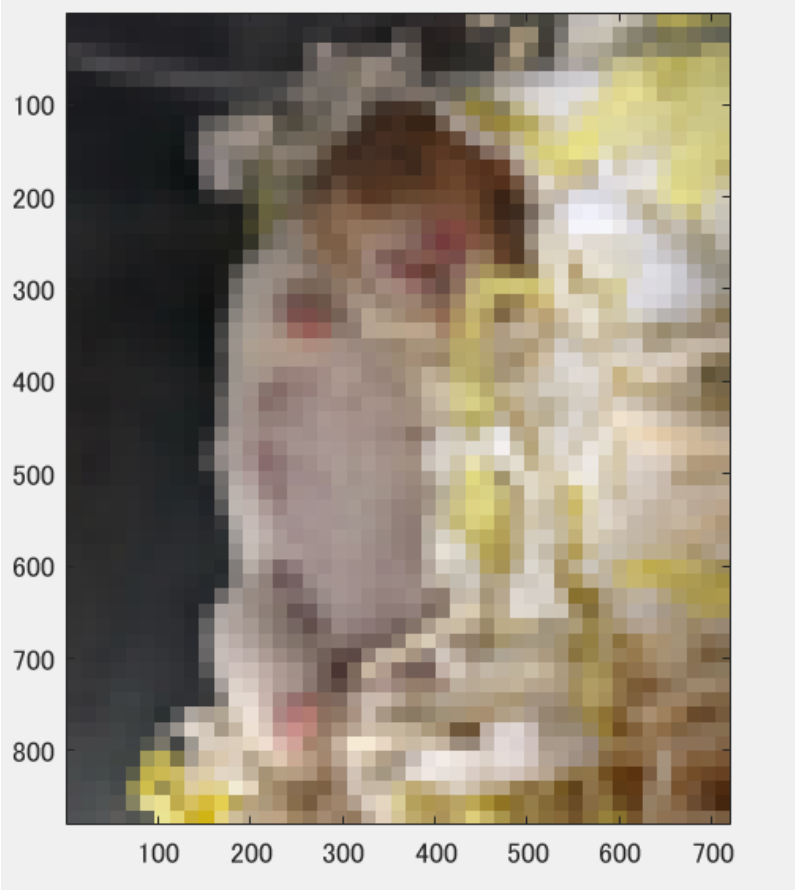

100
200
300
400
500
600
700
800
100
200
300
400
500
600
700

図3.6 5回目


100
200
300
400
500
600
700
800
100
200
300
400
500
600
700

図3.7 6回目

```
% 課題１　標本化間隔と空間解像度
% 画像をダウンサンプリングして（標本化間隔を大きくして）
% 表示せよ．．
% 下記はサンプルプログラムである．
% 課題作成にあたっては「Lenna」以外の画像を用いよ．

clear; % 変数のオールクリア

ORG=imread('D:\daigaku_note\ham.png'); % 原画像の入力
imagesc(ORG); axis image; % 画像の表示
pause; % 一時停止

IMG = imresize(ORG,0.5); % 画像の縮小
IMG2 = imresize(IMG,2,'box'); % 画像の拡大
imagesc(IMG2); axis image; % 画像の表示
pause; % 一時停止

IMG = imresize(IMG,0.5); % 画像の縮小
IMG2 = imresize(IMG,4,'box'); % 画像の拡大
imagesc(IMG2); axis image; % 画像の表示
pause; % 一時停止

IMG = imresize(IMG,0.5); % 画像の縮小
IMG2 = imresize(IMG,8,'box'); % 画像の拡大
imagesc(IMG2); axis image; % 画像の表示
pause; % 一時停止

IMG = imresize(IMG,0.5); % 画像の縮小
IMG2 = imresize(IMG,16,'box'); % 画像の拡大
imagesc(IMG2); axis image; % 画像の表示
pause; % 一時停止

IMG = imresize(IMG,0.5); % 画像の縮小
IMG2 = imresize(IMG,32,'box'); % 画像の拡大
imagesc(IMG2); axis image; % 画像の表示
```

参考:

ニューラルネットワークの教科書[-1]:ゼロから作るDeep Learning pythonで学ぶディープラーニングの理論と実装 斎藤康毅著 オライリー・ジャパン

[0]持論

[1]https://jp.mathworks.com/help/matlab/ref/mtimes.html

[2]http://www.kinyobi.co.jp/kinyobinews/2018/12/26/gender-6/

[3]http://www.sci.kyoto-u.ac.jp/ja/academics/programs/scicom/2015/201602/02.html

[4]https://dictionary.sanseido-publ.co.jp/column/%E7%AC%AC45%E5%9B%9E-%E3%82%A4%E3%83%87%E3%82%AA%E3%83%AD%E3%82%AE%E3%83%BC

[5]http://ker60.hatenablog.com/entry/2017/06/13/000609

[6]https://edtechzine.jp/article/detail/661

[7]https://blog.tokor.org/2017/12/17/rogy-Advent-Calendar-2017-%E3%80%8C%E3%81%86%E3%82%8F%E3%81%A3%E2%80%A6%E3%82%8F%E3%81%9F%E3%81%97%E3%81%AEMATLAB%E3%82%B3%E3%83%BC%E3%83%89%E3%80%81%E9%81%85%E3%81%99%E3%81%8E%E2%80%A6%EF%BC%9F%E3%80%8D/#fn2

[8]https://jp.mathworks.com/help/matlab/matlab_oop/specifying-properties.html

[9]https://jp.mathworks.com/help/comm/ug/matrices-vectors-and-scalars.html

[10]http://ipr20.cs.ehime-u.ac.jp/column/neural/chapter6.html

[11]https://www.iwanttobeacat.com/entry/2018/08/26/225822