清風学園　2019年度 シンポジウム

高校生と学ぶAI

Pythonで学ぶ ニューラルネットワーク

2019年 12月11・12日

常勤顧問 井上愛一郎

president's lunch

おじさんたちと牛丼の様々な事情

ここで問題や！

関澤君、秋草君、黒川君、野副君の4人は、お腹が空いたので、上本町に牛丼を一緒に食べに行こうとしてはります。上本町には、わるの屋ときらい屋の2軒の牛丼屋があります。牛丼は4人とも並盛で良いと言っていますが、関澤君は牛丼にはたまごはいらないけど味噌汁を1杯、秋草君はたまごを2つのせて味噌汁も1杯、黒川君は味噌汁はいらないけどたまごは1つ欲しい、野副君はたまごも味噌汁も1つずつと言ってます。たまごと味噌汁は、わるの屋ではどちらも50円、きらい屋ではたまごが40円、味噌汁が60円です。みんなお金がないので、ランチミーティングにして会社の交際費から出そうということになりました。あとで怒られたら困るのでわるの屋ときらい屋のそれぞれに行った場合に4人それぞれいくらずつ使ったのかをきちんと記録しておくのは秘書の役割です。もしかしたら後から山本君と田中君と時田君が一緒に行くというかもしれませんが牛丼は必ず並盛と言っています。さあ、あなたが有能な秘書になったつもりで、この記録を作ってください。言い忘れましたが、並盛の牛丼は、わるの屋では360円、きらい屋では350円です。

ごちゃごちゃするので表を作ってみよう

食べる数の表

表1	たまご	味噌汁
関澤	個	杯
秋草	個	杯
黒川	個	杯
野副	個	杯
(山本)		
(田中)		
(時田)		

値段の表

表2	わるの屋	きらい屋
たまご	円	円
味噌汁	円	円

なんぼの表

表3	わるの屋 牛丼並盛360円	きらい屋 牛丼並盛350円
関澤		
秋草		
黒川		
野副		
(山本)		
(田中)		
(時田)		

↑たまごと味噌汁に必要なお金を
計算しておこう

たまごと味噌汁にかかるお金を計算してみよう

関澤君がわるの屋に行くと：

関澤君がきらい屋に行くと：

秋草君がわるの屋に行くと：

秋草君がきらい屋に行くと：

黒川君がわるの屋に行くと：

黒川君がきらい屋に行くと：

野副君がわるの屋に行くと：

野副君がきらい屋に行くと：

行列を使ってあらわすと

$$
\begin{array}{c}
\\
\text{関澤} \\
\text{秋草} \\
\text{黒川} \\
\text{野副}
\end{array}
\begin{array}{c}
\begin{array}{cc} \text{たまご} & \text{味噌汁} \end{array} \\
\begin{pmatrix} 0 & 1 \\ 2 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}
\end{array}
$$

$$
\begin{array}{c}
\\
\text{たまご} \\
\text{味噌汁}
\end{array}
\begin{array}{c}
\begin{array}{cc} \text{わるの屋} & \text{きらい屋} \end{array} \\
\begin{pmatrix} 50 & 40 \\ 50 & 60 \end{pmatrix}
\end{array}
$$

$$
\begin{array}{c}
\\
\text{関澤} \\
\text{秋草} \\
\text{黒川} \\
\text{野副}
\end{array}
\begin{array}{c}
\begin{array}{cc} \text{わるの屋} & \text{きらい屋} \end{array} \\
\begin{pmatrix} 50 & 60 \\ 150 & 140 \\ 50 & 40 \\ 100 & 100 \end{pmatrix}
\end{array}
$$

行列どうしのかけ算(行列積)

$$
\begin{array}{c}
\\ \text{関澤} \\ \text{秋草} \\ \text{黒川} \\ \text{野副}
\end{array}
\begin{array}{c}
\begin{array}{cc} \text{たまご} & \text{味噌汁} \end{array} \\
\begin{pmatrix} 0 & 1 \\ 2 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}
\end{array}
\times
\begin{array}{c}
\begin{array}{cc} \text{わるの屋} & \text{きらい屋} \end{array} \\
\begin{array}{c} \text{た} \\ \text{味} \end{array}
\begin{pmatrix} 50 & 40 \\ 50 & 60 \end{pmatrix}
\end{array}
=
\begin{array}{c}
\begin{array}{cc} \text{わるの屋} & \text{きらい屋} \end{array} \\
\begin{pmatrix}
0 \times 50 + 1 \times 50 & 0 \times 40 + 1 \times 60 \\
2 \times 50 + 1 \times 50 & 2 \times 40 + 1 \times 60 \\
1 \times 50 + 0 \times 50 & 1 \times 40 + 0 \times 60 \\
1 \times 50 + 1 \times 50 & 1 \times 40 + 1 \times 60
\end{pmatrix}
\end{array}
$$

$$
=
\begin{array}{c}
\\ \text{関澤} \\ \text{秋草} \\ \text{黒川} \\ \text{野副}
\end{array}
\begin{array}{c}
\begin{array}{cc} \text{わるの屋} & \text{きらい屋} \end{array} \\
\begin{pmatrix} 50 & 60 \\ 150 & 140 \\ 50 & 40 \\ 100 & 100 \end{pmatrix}
\end{array}
$$

食べる数 × 値段 ＝ なんぼ の関係や

Python

今回のプログラムに必要な文法をかいつまんで
(ときどき牛丼屋)

Pythonの特徴

スクリプト言語：テキストで簡便に手続きを記述するプログラム言語

インタープリタ方式：逐次、解釈しながら実行(⇔コンパイラ方式)

インデント：インデントが制御に使われる

Numpyやmatplotlibなど豊富(今回は使わない)

オブジェクト指向プログラミングに対応

人工知能開発でスタンダード

PC上に無料で簡単に記述と実行の環境が作れる

https://www.python.org/からダウンロードしてインストールですぐ動く

配列(ベクトルや行列)が扱える

個人的には刹那的な印象、あらかじめ配列変数を用意するのではなく、実行の成行きで次々と作っていくのに向いている(とてもスクリプトらしい)

変数

値や変数には、次の４つの型がある

int　　整数型

float 浮動小数点型

str　　文字列型

bool　論理型

変数に対して型の明示は必要なく、代入の際に動的に変換される

(基本的に変数は代入しようとする値の型になる)

しかし入出力などで、明示的に型を指定したい場合もある

その場合、int(値や変数)、float(値や変数)のようにして、

型を変換・指定することができる

[注] " " か ' ' で囲まれた値は文字列となる

リストと配列

複数の値を一括して扱えるリストがあり、これも型の1つとされる

リストは全体を[ ]で囲み、各要素を ’ , ’（カンマ）で区切る

例えば　a =[1, 2, 3, 4, 5]とすれば、変数aは1～5の5つの値をもつ

リストを用いてpythonで配列を扱うことができる

この例の a は1次元配列=ベクトルとみなすことができる

さらにリストの中にリストを格納することもできるから、

これを利用して多次元の配列を扱うことができる

例えば　A=[[1, 2, 3], [4, 5, 6]]とすれば、

Aは2行3列の2次元配列=行列とみなすことができる

リストと配列～具体例=牛丼屋

関澤君のたまごと味噌汁の数は、[0, 1]

秋草君のたまごと味噌汁の数は、[2, 1]

黒川君のたまごと味噌汁の数は、[1, 0]

野副君のたまごと味噌汁の数は、[1, 1]

これをまとめると、[[0, 1],[2, 1],[1, 0],[1, 1]]

←これは表1

$$\begin{array}{c} \\ \text{関澤} \\ \text{秋草} \\ \text{黒川} \\ \text{野副} \end{array} \begin{array}{c} \begin{array}{cc} \text{たまご} & \text{味噌汁} \end{array} \\ \begin{pmatrix} 0 & 1 \\ 2 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \end{array}$$

4行2列の行列

いっぽう、

たまごの値段はわるの屋ときらい屋で、[50, 40]

味噌汁の値段はわるの屋ときらい屋で、[50, 60]

これをまとめると、[[50, 40],[50, 60]] ←これは表2

$$\begin{array}{c} \\ \text{たまご} \\ \text{味噌汁} \end{array} \begin{array}{c} \begin{array}{cc} \text{わるの屋} & \text{きらい屋} \end{array} \\ \begin{pmatrix} 50 & 40 \\ 50 & 60 \end{pmatrix} \end{array}$$

2行2列の行列

'[[' と ']]' どっちも2重や、いや、'[リスト]' の中に '[リスト]' や

では、いきなり、やってみよう！

まずはpythonのshellを起動

pythonのshellのウィンドウ

なんやねん！

いきなりやるで！

表1(食べる数)　X = [[0, 1],[2, 1],[1, 0],[1, 1]] を入力

表2(値段)　　W = [[50, 40],[50, 60]]　　を入力

Python 3.7.1 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:57:15) [MSC v.1915 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> X = [[0, 1],[2, 1],[1, 0],[1, 1]]
>>> W = [[50, 40],[50, 60]]
>>> X
[[0, 1], [2, 1], [1, 0], [1, 1]]
>>> W
[[50, 40], [50, 60]]
```

XとWを確認

たまごや味噌汁は安いもの

ほんまかいな？

いきなり、もっとやるで！

Numpyをインポート　import numpy as np

行列積　np.dot(X, W)

Python 3.7.1 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.1 (v3.7.1:260ec2c36a, Oct 20 2018, 14:57:15) [MSC v.1915 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> X = [[0, 1],[2, 1],[1, 0],[1, 1]]
>>> W = [[50, 40],[50, 60]]
>>> X
[[0, 1], [2, 1], [1, 0], [1, 1]]
>>> W
[[50, 40], [50, 60]]
>>> import numpy as np
>>> np.dot(X, W)
array([[ 50,  60],
       [150, 140],
       [ 50,  40],
       [100, 100]])
>>>
```

⇒　なんぼ(結果)の表がでけた！

ちゃうねん！

あんさん兵庫でっか？

牛丼屋で牛丼を食べへんのは学校で勉強しいへんよな

わるの屋	きらい屋
牛丼並盛360円	牛丼並盛350円

たまごと味噌汁の合計代金に牛丼並盛の代金(こっちのほうが高いけど)を全員に上乗せや

$$
\begin{array}{c}
\text{わるの屋 きらい屋} \\
\begin{array}{c}
\text{関澤} \\ \text{秋草} \\ \text{黒川} \\ \text{野副}
\end{array}
\begin{pmatrix} 50 & 60 \\ 150 & 140 \\ 50 & 40 \\ 100 & 100 \end{pmatrix}
\end{array}
+
\begin{array}{c}
\text{わるの屋 きらい屋} \\
\begin{pmatrix} 360 & 350 \end{pmatrix}
\end{array}
=
\begin{array}{c}
\text{わるの屋 きらい屋} \\
\begin{pmatrix} 50+360 & 60+350 \\ 150+360 & 140+350 \\ 50+360 & 40+350 \\ 100+360 & 100+350 \end{pmatrix}
\end{array}
$$

たまごと味噌汁の合計

牛丼並盛

この足し算は行方向に拡張して全要素に加算

牛丼されど牛丼！

牛丼並盛の値段表　B = [360,350] を入力

これを足し合わせる(結果を U としよう)

```
>>> X = [[0, 1],[2, 1],[1, 0],[1, 1]]
>>> W = [[50, 40],[50, 60]]
>>> X
[[0, 1], [2, 1], [1, 0], [1, 1]]
>>> W
[[50, 40], [50, 60]]
>>> import numpy as np
>>> np.dot(X, W)
array([[ 50,  60],
       [150, 140],
       [ 50,  40],
       [100, 100]])
>>> B = [360, 350]
>>> U = np.dot(X, W) + B
>>> U
array([[410, 410],
       [510, 490],
       [410, 390],
       [460, 450]])
>>> |
```

'dot' も偉いけど '+' も偉い

(ややこしいだけや！ほんまに偉いんは誰や?)

高いわ！

バイアスばかり大きうて...

⇒　これがほんとのなんぼの表ですわ！

こわい秘書室長

秘書室長の綾子さんは社長の昼飯は450円までと決めています

それを超えた場合は有無を言わせず食事の感想文を書かせます

こわいから誰がどの場合に感想文を書かなければならないか表にしておきます

$$
\begin{array}{c}
\text{わるの屋 きらい屋} \\
\begin{array}{c} \text{関澤} \\ \text{秋草} \\ \text{黒川} \\ \text{野副} \end{array}
\begin{pmatrix} 410 & 410 \\ 510 & 490 \\ 410 & 390 \\ 460 & 450 \end{pmatrix}
\end{array}
\xrightarrow[\text{450円までは '0'}\ \text{450円を超えたら '1'}]{\text{ステップ関数}\ f}
\begin{array}{c}
\text{わるの屋 きらい屋} \\
\begin{array}{c} \text{関澤} \\ \text{秋草} \\ \text{黒川} \\ \text{野副} \end{array}
\begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \end{pmatrix}
\end{array}
$$

秋さん、やばいで！

野副はんもキイつけな

そもそも、たまご2個も食うからあかんのや

pythonでちゃちゃっと、ufiesiaも使うて

```
>>> import numpy as np
>>> np.dot(X, W)
array([[ 50,  60],
       [150, 140],
       [ 50,  40],
       [100, 100]])
>>> B = [360, 350]
>>> U = np.dot(X, W) + B
>>> U
array([[410, 410],
       [510, 490],
       [410, 390],
       [460, 450]])
>>> from ufiesia import common_function_aid as cf
>>> cf.step(U, 450)
array([[0, 0],
       [1, 1],
       [0, 0],
       [1, 0]])
>>>
```

便利や！ちごた、怖いわ

高いお金を使って出る結論は？

お腹が空いた関澤君、秋草君、黒川君、野副君の４人が、「たべるたまごと味噌汁の数」＝入力に応じて、「感想文を書く書かないの結果」＝出力が決まります。そしてその関係を決定づける＝パラメタが、「わるの屋ときらい屋のたまごと味噌汁の値段と牛丼並盛の値段」です。

入力	パラメタ	出力
$\boldsymbol{X}$	$\boldsymbol{W}$	$\boldsymbol{Y}$

$$\boldsymbol{X} = \begin{array}{c} \\ \text{関澤} \\ \text{秋草} \\ \text{黒川} \\ \text{野副} \end{array} \begin{array}{c} \begin{array}{cc} \text{たまご} & \text{味噌汁} \end{array} \\ \begin{pmatrix} 0 & 1 \\ 2 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \end{array}$$

$$\boldsymbol{W} = \begin{array}{c} \\ \text{たまご} \\ \text{味噌汁} \end{array} \begin{array}{c} \begin{array}{cc} \text{わるの屋} & \text{きらい屋} \end{array} \\ \begin{pmatrix} 50 & 40 \\ 50 & 60 \end{pmatrix} \end{array}$$

$$\boldsymbol{B} = \begin{array}{c} \\ \text{牛丼} \end{array} \begin{array}{c} \begin{array}{cc} \text{わるの屋} & \text{きらい屋} \end{array} \\ \begin{pmatrix} 360 & 350 \end{pmatrix} \end{array}$$

$$\boldsymbol{Y} = \begin{array}{c} \\ \text{関澤} \\ \text{秋草} \\ \text{黒川} \\ \text{野副} \end{array} \begin{array}{c} \begin{array}{cc} \text{わるの屋} & \text{きらい屋} \end{array} \\ \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} \end{array}$$

牛丼屋でのランチで感想文を書くのは誰？

「たまごと味噌汁の数」から、「感想文を書く書かない」を求める式は？

$$
\boldsymbol{U} = \boldsymbol{X} \times \boldsymbol{W} + \boldsymbol{B}
$$

$$
\begin{array}{c} \\ \text{関澤} \\ \text{秋草} \\ \text{黒川} \\ \text{野副} \end{array}
\begin{array}{c} \begin{array}{cc} \text{わるの屋} & \text{きらい屋} \end{array} \\ \begin{pmatrix} 410 & 410 \\ 510 & 490 \\ 410 & 390 \\ 460 & 450 \end{pmatrix} \end{array}
=
\begin{array}{c} \\ \text{関澤} \\ \text{秋草} \\ \text{黒川} \\ \text{野副} \end{array}
\begin{array}{c} \begin{array}{cc} \text{たまご} & \text{味噌汁} \end{array} \\ \begin{pmatrix} 0 & 1 \\ 2 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \end{array}
\times
\begin{array}{c} \\ \text{たまご} \\ \text{味噌汁} \end{array}
\begin{array}{c} \begin{array}{cc} \text{わるの屋} & \text{きらい屋} \end{array} \\ \begin{pmatrix} 50 & 40 \\ 50 & 60 \end{pmatrix} \end{array}
+
\begin{array}{c} \\ \text{牛丼} \end{array}
\begin{array}{c} \begin{array}{cc} \text{わるの屋} & \text{きらい屋} \end{array} \\ \begin{pmatrix} 360 & 350 \end{pmatrix} \end{array}
$$

$$
\boldsymbol{Y} = \boldsymbol{f}\,(\boldsymbol{U})
$$

$$
\begin{array}{c} \\ \text{関澤} \\ \text{秋草} \\ \text{黒川} \\ \text{野副} \end{array}
\begin{array}{c} \begin{array}{cc} \text{わるの屋} & \text{きらい屋} \end{array} \\ \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} \end{array}
=
\boldsymbol{f}_{\text{step}}
\left(
\begin{array}{c} \\ \text{関澤} \\ \text{秋草} \\ \text{黒川} \\ \text{野副} \end{array}
\begin{array}{c} \begin{array}{cc} \text{わるの屋} & \text{きらい屋} \end{array} \\ \begin{pmatrix} 410 & 410 \\ 510 & 490 \\ 410 & 390 \\ 460 & 450 \end{pmatrix} \end{array}
\right)
$$

president's lunchをpythonで①

これまでのことをまとめて

```
import numpy as np
from ufiesia import common_function_aid as cf

X = [[0, 1],[2, 1],[1, 0],[1, 1]]
W = [[50, 40],[50, 60]]
B = [360, 350]

U = np.dot(X, W) + B
Y = cf.step(U, 450)

print(Y)
```

president's lunchをpythonで②

核の部分を関数にしておこう

```
import numpy as np
from ufiesia import common_function_aid as cf

def forward(X, W, B):
    U = np.dot(X, W) + B
    Y = cf.step(U, 450)
    return Y

X = [[0, 1],[2, 1],[1, 0],[1, 1]]
W = [[50, 40],[50, 60]]
B = [360, 350]

Y = forward(X, W, B)
print(Y)
```

president's lunchをpythonで③

人によって反応はまちまちなので、関数を実行するときに指定するのが良さそう

```
import numpy as np
from ufiesia import common_function_aid as cf

def forward(X, W, B, act=None):
    U = np.dot(X, W) + B
    if act == 'step450':
        Y = cf.step(U, 450)
    else:
        Y = U
    return Y

X = [[0, 1],[2, 1],[1, 0],[1, 1]]
W = [[50, 40],[50, 60]]
B = [360, 350]

Y = forward(X, W, B, 'step450')
print(Y)
```

president's lunchをpythonで④

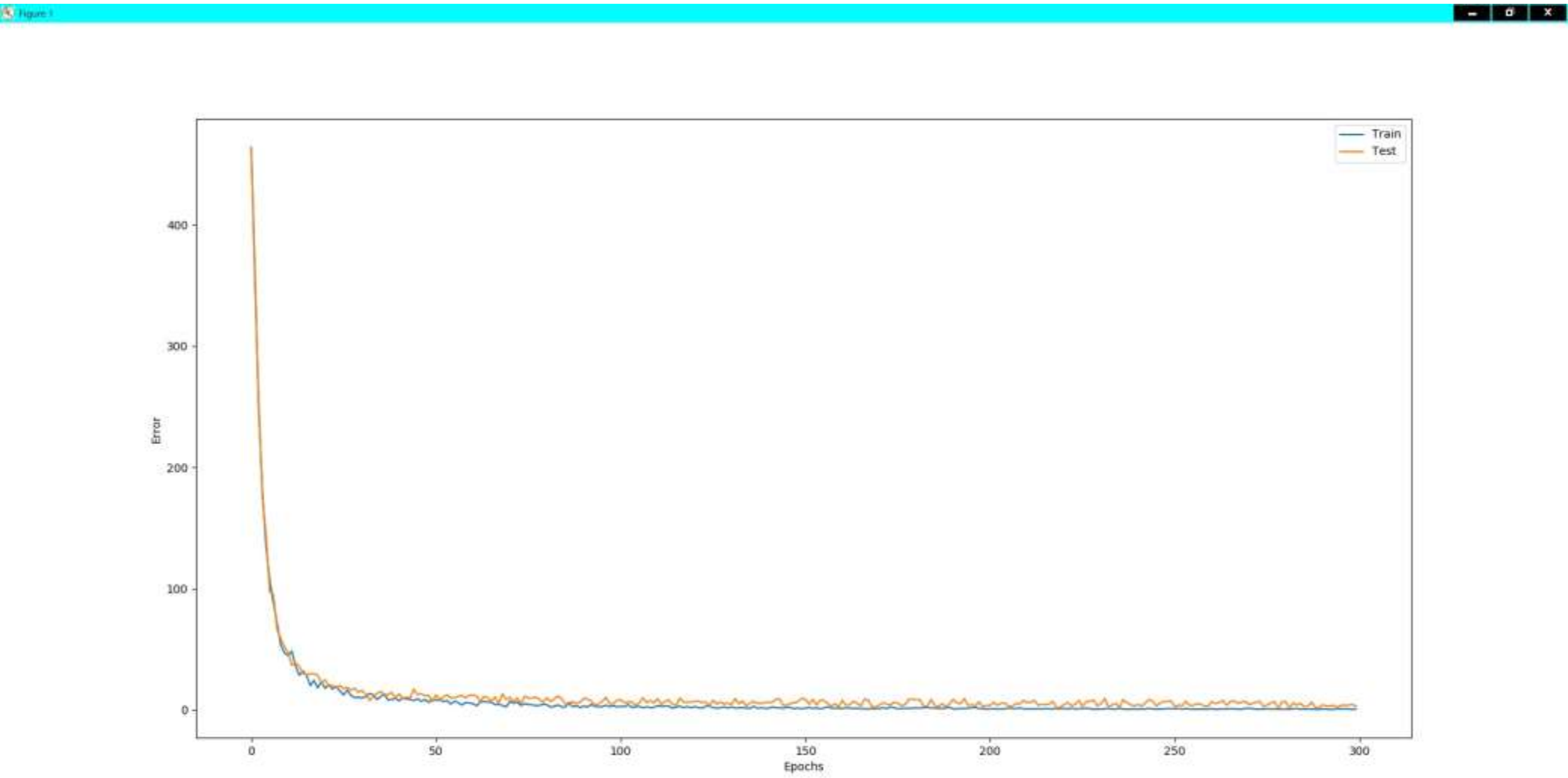
ここまで来たら一気に！オブジェクト指向プログラミング！

```
import numpy as np
from ufiesia import common_function_aid as cf

class Layer:                    # 設計図のようなもの クラス
    def __init__(self, W, B, act=None): # 価格表と反応の仕方を与えて初期化、コントラクタ
        self.W = W
        self.B = B
        self.act = act

    def forward(self, X):   # 価格表と入力の関係、それを受けて出力との関係
        U = np.dot(X, self.W) + self.B
        if self.act == 'step450':
            self.Y = cf.step(U, 450)
        else:
            self.Y = U

X = [[0, 1],[2, 1],[1, 0],[1, 1]]
W = [[50, 40],[50, 60]]
B = [360, 350]

presidents_lunch = Layer(W, B, 'step450')  # 実体、インスタンスを生成
presidents_lunch.forward(X) # 生成したインスタンスのforwardメソッドを実行
print(presidents_lunch.Y)
```

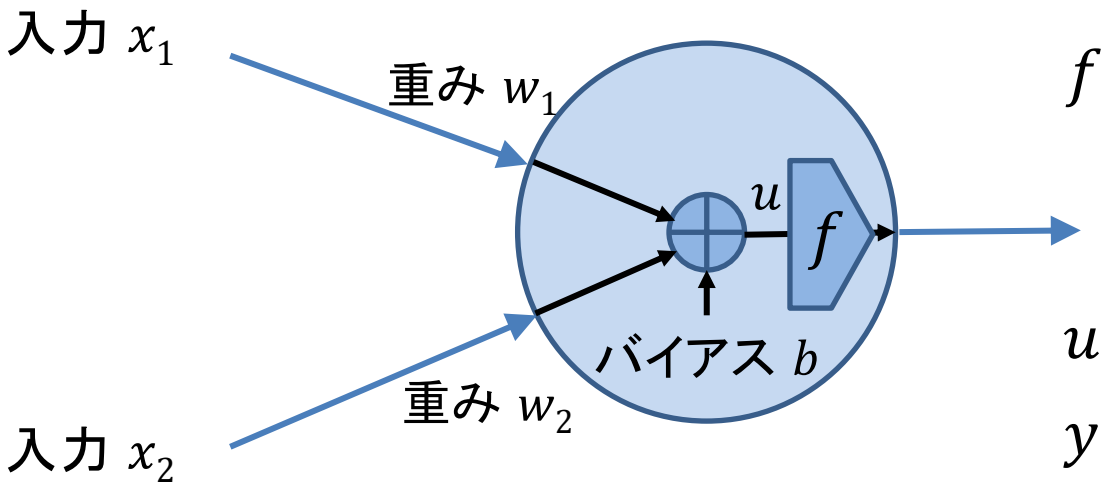
ニューラルネットワーク

牛丼屋に関係あるんかいな？

ufiesia 環境で作った CNN でsklearn-digitsを学習 2019年7月26日



ニューロン



f : 活性化関数

$$u = x_1 \times w_1 + x_2 \times w_2 + b$$

$$y = f\ (\mathrm{u})$$

あれ？、なんか見たような式？

ニューロンは、入力に反応して出力するが、その際、入力はそれぞれの重みがかかって足し合わされ、さらに、バイアスが加わった値が、ニューロンの興奮を決める
重みは文字通り、どの入力からの信号を重視するのかを決定づけ、バイアスは入力とその重みによって決まる値を、かさ上げしたり減じたりして、ニューロンの興奮度合いを調節する
そしてそこで入力とバイアスから決まる値に、出力がどう対応するかの関係を決める関数を活性化関数という

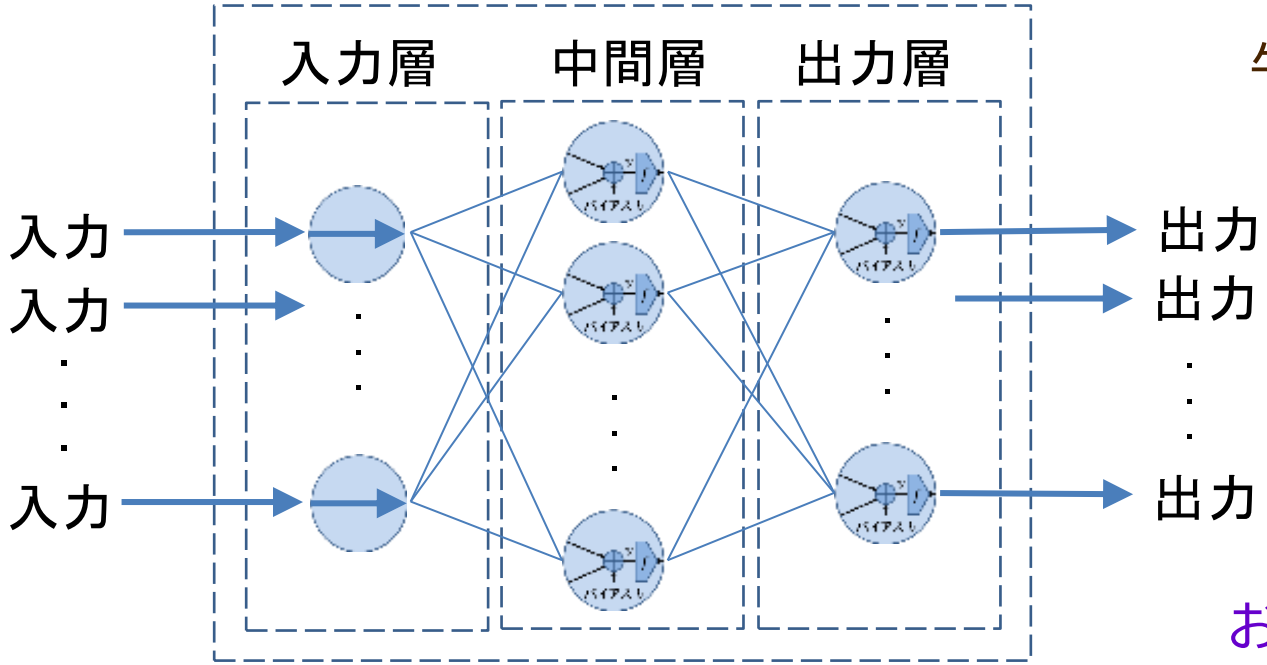
ニューラルネットワーク

ニューロンが複数集まってネットワーク状につながったものをニューラルネットワークという

通常、同時に信号が伝わるいくつかのニューロンの集まりを層として、ニューラルネットワークは、入力層と出力層のほかに1つまたは複数の中間層を持っている

中間層は外部から直接見えないため「かくれ層」と呼ばれることもある

ニューラルネットワークで入力が各層を通って出力に伝わることを順伝播という

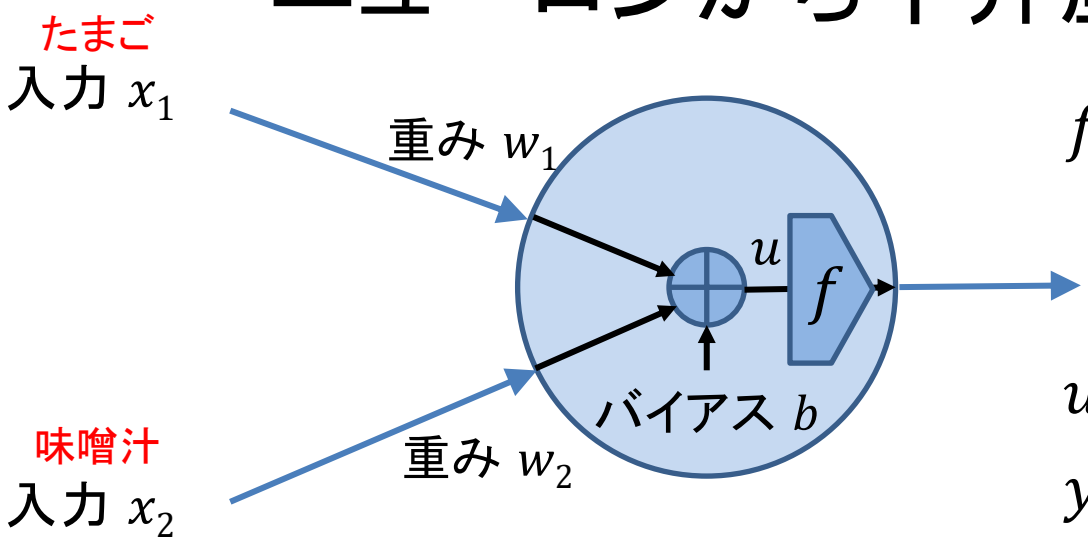


ニューラルネットワーク

牛丼屋はわるの屋ときらい屋
それだけじゃないしな

おまけに層がぎょうさんかいな
感想文の次に何する？

ニューロンから牛丼屋にまいもどる



f : 活性化関数

$$u = x_1 \times w_1 + x_2 \times w_2 + b$$

$$y = f\ (\mathrm{u})$$

たまごが x_1 で、味噌汁が x_2 ならば、

関澤君は $x_1 = 0$個、$x_2 = 1$個
秋草君は $x_1 = 2$個、$x_2 = 1$個
黒川君は $x_1 = 1$個、$x_2 = 0$個
野副君は $x_1 = 1$個、$x_2 = 1$個

4人、えらくなってもみな牛丼屋

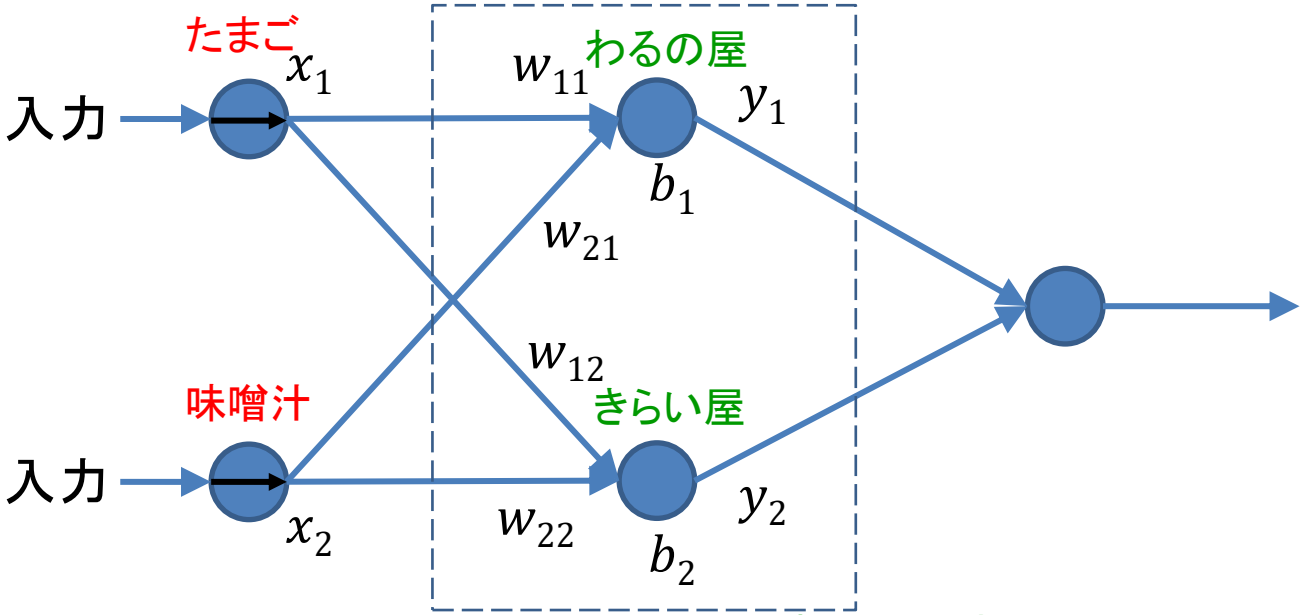
たまごの値段が w_1 で、味噌汁の値段が w_2 そして牛丼の値段が b ならば、

わるの屋なら、$w_1 = 50$円、$w_2 = 50$円、$b = 360$円
きらい屋なら、$w_1 = 40$円、$w_2 = 60$円、$b = 350$円

2店、わるの屋ときらい屋

ニューラル牛丼屋ネットワークのかくれ層

$$
\begin{array}{c|cc}
 & \text{たまご} & \text{味噌汁} \\
\hline
\text{関澤} & 0 & 1 \\
\text{秋草} & 2 & 1 \\
\text{黒川} & 1 & 0 \\
\text{野副} & 1 & 1
\end{array}
$$



2つの入力は
たまごと味噌汁

2つのニューロンは
わるの屋ときらい屋

$$
\begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix}
=
\begin{array}{c}
 \\ \text{たまご} \\ \text{味噌汁}
\end{array}
\begin{array}{c}
\begin{array}{cc} \text{わるの屋} & \text{きらい屋} \end{array} \\
\begin{pmatrix} 50 & 40 \\ 50 & 60 \end{pmatrix}
\end{array}
$$

$$
\begin{pmatrix} b_1 & b_2 \end{pmatrix}
=
\begin{array}{c}
 \\ \text{牛丼}
\end{array}
\begin{array}{c}
\begin{array}{cc} \text{わるの屋} & \text{きらい屋} \end{array} \\
\begin{pmatrix} 360 & 350 \end{pmatrix}
\end{array}
$$

どこがニューラルやねん？

振返ると

ニューラル牛丼屋ネットワークのかくれ層

この部分をモジュール化

```
import numpy as np
from ufiesia import common_function_aid as cf

class Layer:                    # 設計図のようなもの クラス
    def __init__(self, W, B, act=None): # 価格表と反応の仕方を与えて初期化、コントラクタ
        self.W = W
        self.B = B
        self.act = act

    def forward(self, X):   # 価格表と入力の関係、それを受けて出力との関係
        U = np.dot(X, self.W) + self.B
        if self.act == 'step450':
            self.Y = cf.step(U, 450)
        else:
            self.Y = U
```

```
X = [[0, 1],[2, 1],[1, 0],[1, 1]]
W = [[50, 40],[50, 60]]
B = [360, 350]

presidents_lunch = Layer(W, B, 'step450')  # 実体、インスタンスを生成
presidents_lunch.forward(X) # 生成したインスタンスのforwardメソッドを実行
print(presidents_lunch.Y)
```

neuron0

モジュール化しておけば牛丼屋に限らず色々なプログラムで使える

```
import numpy as np
from ufiesia import common_function_aid as cf

class Layer:                    # 設計図のようなもの クラス
    def __init__(self, W, B, act=None): # 初期化、コントラクタ
        self.W = W
        self.B = B
        self.act = act

    def forward(self, X):   # 入出力
        U = np.dot(X, self.W) + self.B
        if self.act == 'step450':
            self.Y = cf.step(U, 450)
        else:
            self.Y = U
```

あらためて牛丼屋は？

ニューラル牛丼屋ネットワークのかくれ層

```
import neuron0

X = [[0, 1],[2, 1],[1, 0],[1, 1]]
W = [[50, 40],[50, 60]]
B = [360, 350]

presidents_lunch = neuron0.Layer(W, B, 'step450')  # 実体、インスタンスを生成
presidents_lunch.forward(X) # 生成したインスタンスのforwardメソッドを実行
print(presidents_lunch.Y)
```

とても簡単や！

その実行結果＝かくれ層の出力

```
[[0 0]
 [1 1]
 [0 0]
 [1 0]]
```

出力層も同じようにプログラムできるはず

こじつけですが…

じつは関澤君、秋草君、黒川君、野副君の4人のランチミーティングは連日におよびます。 秘書室長の綾子さんがしらべたところ、週のうち2日はわるの屋に、3日はきらい屋に行ってはることがわかりました。
それでも彼らはみな、綾子さんに怒られるかどうかによらず、週のうち1日は休みますが、もう1日は会社で感想文を書いてはるのです。

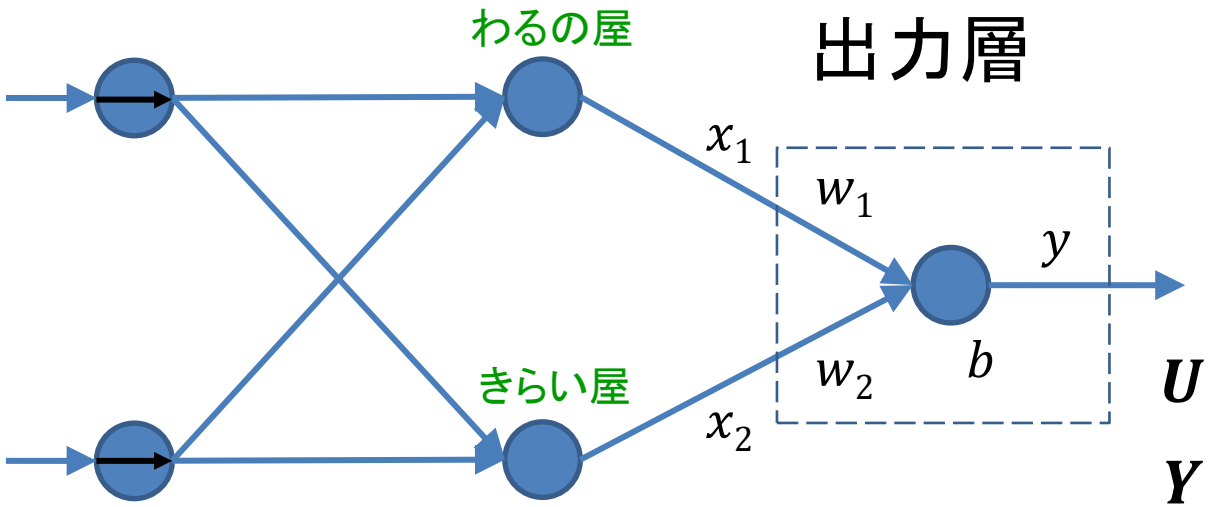
わるの屋に行く回数
きらい屋に行く回数

$$\begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

会社で感想文を書く回数

$$\begin{pmatrix} 1 \end{pmatrix}$$

牛丼屋を始めてしまったので…



$$\boldsymbol{U} = \boldsymbol{X} \times \boldsymbol{W} + \boldsymbol{B}$$

$\boldsymbol{Y} = \boldsymbol{U}$ 活性化は恒等関数

出力層への入力はかくれ層の出力

$$\begin{array}{c} \\ \text{関澤} \\ \text{秋草} \\ \text{黒川} \\ \text{野副} \end{array} \begin{array}{c} \begin{array}{cc} \text{わるの屋} & \text{きらい屋} \end{array} \\ \begin{pmatrix} 0 & 0 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} \\ \begin{array}{cc} \uparrow & \uparrow \\ x_1 & x_2 \end{array} \end{array}$$

$$\begin{array}{c} \text{わるの屋} \\ \text{きらい屋} \end{array} \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

$$\text{会社で} \begin{pmatrix} b \end{pmatrix} = \begin{pmatrix} 1 \end{pmatrix}$$

$$\begin{array}{c} \\ \text{関澤} \\ \text{秋草} \\ \text{黒川} \\ \text{野副} \end{array} \begin{array}{c} \text{感想文の数} \\ \begin{pmatrix} 1 \\ 6 \\ 1 \\ 3 \end{pmatrix} \\ \uparrow \end{array}$$

y :出力層の出力

牛丼屋に行った結果として

ねんのため、ニューラル牛丼屋ネットワークの出力層で行う計算を確認

```
>>> import numpy as np
>>> X = [[0, 0],[1, 1],[0, 0],[1, 0]]
>>> W = [[2],[3]]
>>> B = [1]
>>> U = np.dot(X, W) + B
>>> Y = U
>>> Y
array([[1],
       [6],
       [1],
       [3]])
>>> |
```

間違いなし！

これをニューラル牛丼屋ネットワークに組み込む

ニューラル牛丼屋ネットワーク全体

```
import neuron0

X  = [[0, 1],[2, 1],[1, 0],[1, 1]]
Wm = [[50, 40],[50, 60]]     # かくれ層のW
Bm = [360, 350]              # かくれ層のB
Wo = [[2],[3]]               # 出力層のW
Bo = [1]                     # 出力層のB

middle_layer = neuron0.Layer(Wm, Bm, 'step450')   # かくれ層
output_layer = neuron0.Layer(Wo, Bo)        # 出力層、出力はそのまま通過
middle_layer.forward(X)
output_layer.forward(middle_layer.Y) # かくれ層の出力が出力層の入力

Y = output_layer.Y
print(Y)
```

これで良かったんかいな？

ふたたびニューロン

neuron0では活性化関数として、閾値450のステップ関数でしたが、

```
import numpy as np
from ufiesia import common_function_aid as cf

class Layer:                    # 設計図のようなもの クラス
    def __init__(self, W, B, act=None): # 初期化、コントラクタ
        self.W = W
        self.B = B
        self.act = act

    def forward(self, X):   # 入出力
        U = np.dot(X, self.W) + self.B
        if self.act == 'step450':
            self.Y = cf.step(U, 450)
        else:
            self.Y = U
```

← この部分

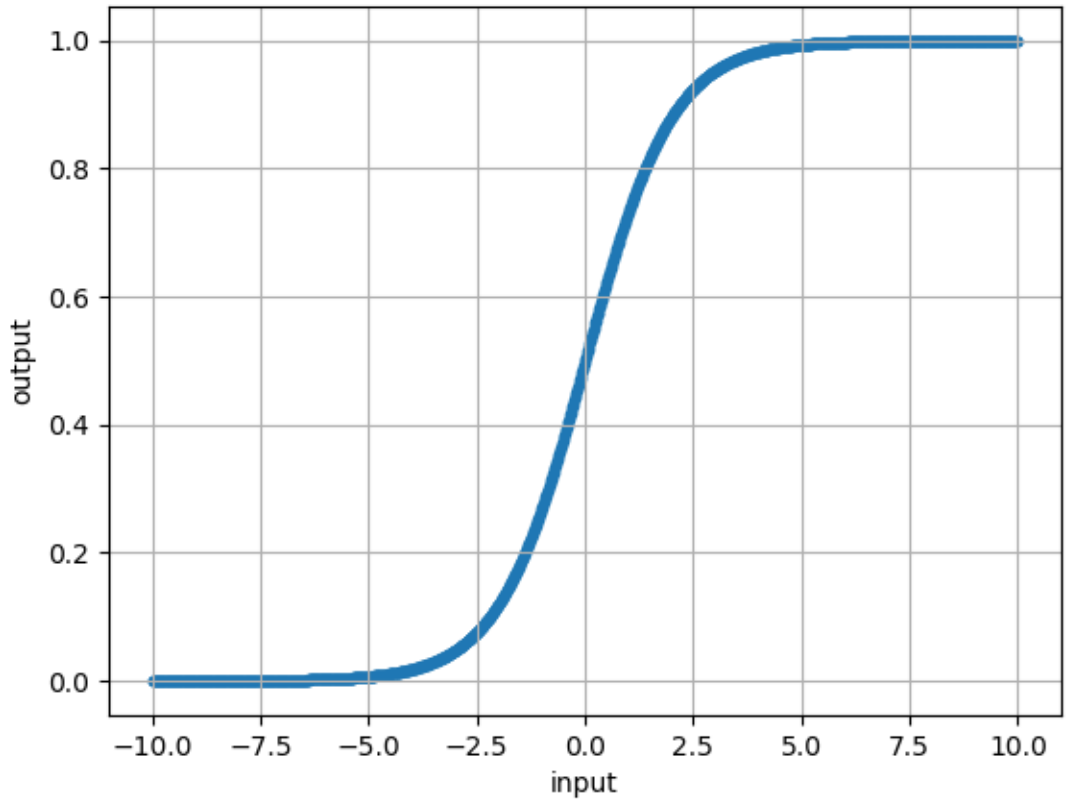
ニューラルネットワークで多用する活性化関数を使えるようにしなければ...

ニューラルネットワークで多用する活性化関数
シグモイド sigmoid

生物の神経細胞の刺激に対する反応を模した関数

グラフを見た通り0から離れると勾配が小さくなるのが欠点

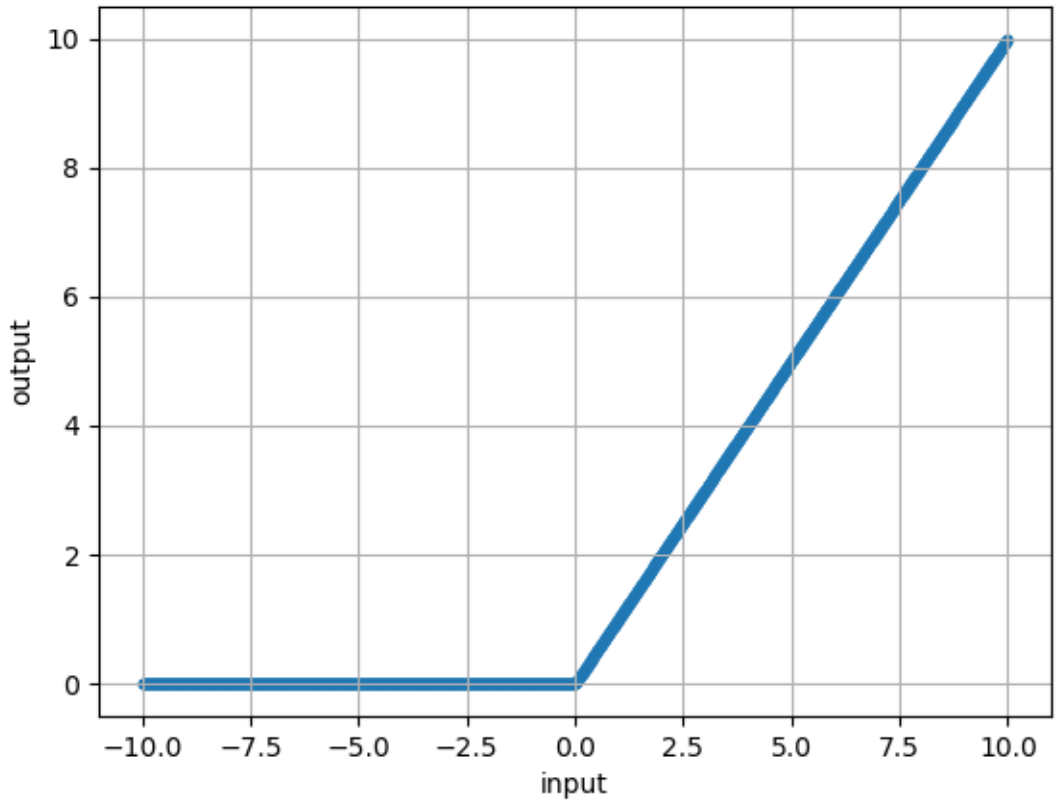
$$S(x) = \frac{1}{1 + e^{-x}}$$



ニューラルネットワークで多用する活性化関数
ランプ ramp

シグモイド関数にかわってディープラーニングでよく使われる

$$R(x) := \begin{cases} x, & x \geqq 0 \\ 0, & x < 0 \end{cases}$$

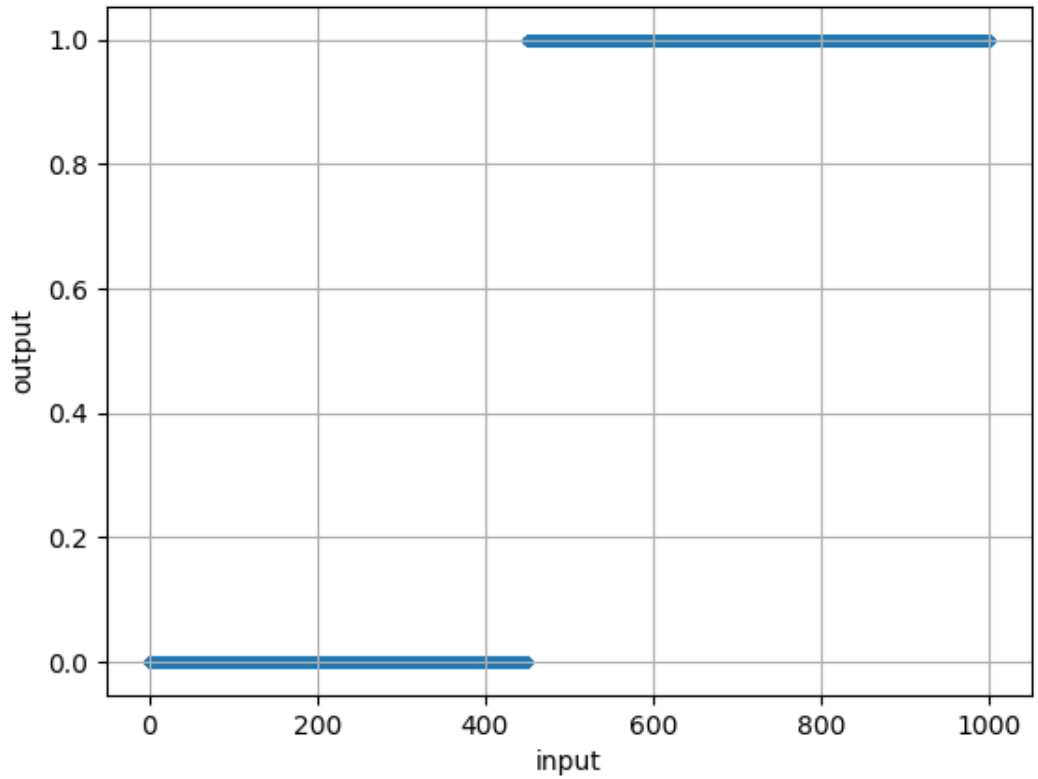


ニューラルネットワークで多用する活性化関数
ステップ step

ニューラルネットワークの起源であるパーセプトロンで使われてきた

$$Step(x) := \begin{cases} 1, & x > t \\ 0, & x \leqq t \end{cases}$$

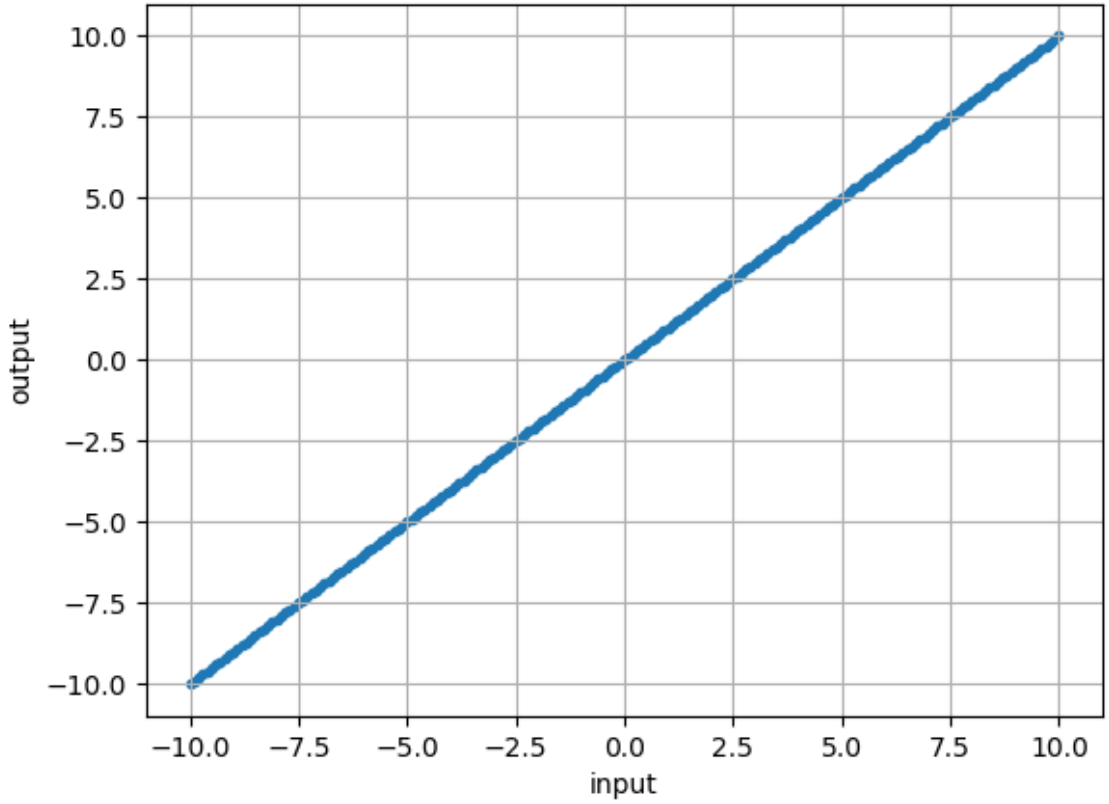
t は閾値



ニューラルネットワークで多用する活性化関数
恒等関数 identity

一応、そのまま通過させるものも恒等関数として

$$I(x) = x$$



活性化関数をまとめてpythonで

```
import numpy as np
from ufiesia import common_function_aid as cf

def sigmoid(X):
    return 1 /(1 + np.exp(-X))

def ramp(X):
    return np.where(X>0, X, 0)

def step(X, t=0):
    return np.where(X>t, 1, 0)

def identity(X):
    return X

#X = np.array([i/10 for i in range(-100, 101)])  # [[i/100]] -10～10、0.01刻みの値
X = np.linspace(-10, 10, 201)

Y = sigmoid(X)
cf.xy_graph(X, Y)

Y = ramp(X)
cf.xy_graph(X, Y)

Y = step(X)
cf.xy_graph(X, Y)

Y = identity(X)
cf.xy_graph(X, Y)
```

numpy便利やな！

neuron1

ニューラルネットワークで多用する活性化関数を加えて汎用化

```
import numpy as np

def sigmoid(X):
    return 1 /(1 + np.exp(-X))

def ramp(X):
    return np.where(X>0, X, 0)

def step(X, t=0):
    return np.where(X>t, 1, 0)

class Layer:                    # 層のクラス
    def __init__(self, W, B, act=None): # 初期値はインスタンス生成時に与える
        self.W = W
        self.B = B
        self.act = act

    def forward(self, X):   # 順伝播
        U = np.dot(X, self.W) + self.B
        if   self.act == 'step450': # 牛丼屋専用
            self.Y = step(U, 450)
        elif self.act == 'sigmoid':
            self.Y = sigmoid(U)
        elif self.act == 'ramp':
            self.Y = ramp(U)
        elif self.act == 'step':
            self.Y = step(U)
        else:
            self.Y = U
```

だんだんそれらしく...

neuron1でもニューラル牛丼屋ネットワークが動くことを確認

```
import neuron1

X  = [[0, 1],[2, 1],[1, 0],[1, 1]]
Wm = [[50, 40],[50, 60]]     # かくれ層のW
Bm = [360, 350]              # かくれ層のB
Wo = [[2],[3]]               # 出力層のW
Bo = [1]                     # 出力層のB

middle_layer = neuron1.Layer(Wm, Bm, 'step450')   # かくれ層
output_layer = neuron1.Layer(Wo, Bo)          # 出力層、出力はそのまま通過
middle_layer.forward(X)
output_layer.forward(middle_layer.Y) # かくれ層の出力が出力層の入力

Y = output_layer.Y
print(Y)
```

ひと安心

すこしだけニューラルネットワークらしく

牛丼屋ではなく、関数をなぞらせてみよう

入力は-4～4、0.1刻みの値を内包表記で用意

学習済みパラメータを使って関数回帰

結果はグラフで確認

```
import numpy as np
import neuron1
from ufiesia import common_function_aid as cf

X = [[i/10] for i in range(-40, 41)]        # -4～4、0.1刻みの値
Wm, Bm, Wo, Bo = cf.parms_for_NN141('sin')  # 学習済みパラメータを与える

middle_layer = neuron1.Layer(Wm, Bm, 'sigmoid')   # かくれ層
output_layer = neuron1.Layer(Wo, Bo)        # 出力層、出力はそのまま通過
middle_layer.forward(X)
output_layer.forward(middle_layer.Y) # かくれ層の出力が出力層の入力

Y = output_layer.Y
cf.xy_graph(X, Y)
```

パラメタを変えるだけで違う働きをしはります

ほんまでっか？

きょうはここまで

２日目

ニューラルネットワークで画像認識

そろそろ牛丼屋から離れて

手書き文字データ

手書き文字データ　ufiesiaの関数から返されるデータの内容を確認しよう

```
>>> from ufiesia import sklearn_digits as DG
>>> data = DG.get_data()
sklearn digits のデータが読み込まれました
入力と正解値のデータが用意できました
入力データの形状（1797, 64）正解値の形状（1797, 10）
>>> data
(array([[-0.8117562 , -0.8117562 ,  0.01925204, ..., -0.8117562 ,
         -0.8117562 , -0.8117562 ],
        [-0.8117562 , -0.8117562 , -0.8117562 , ...,  0.85026027,
         -0.8117562 , -0.8117562 ],
        [-0.8117562 , -0.8117562 , -0.8117562 , ...,  1.84747015,
          0.68405863, -0.8117562 ],
        ...,
        [-0.8117562 , -0.8117562 , -0.64555455, ...,  0.18545368,
         -0.8117562 , -0.8117562 ],
        [-0.8117562 , -0.8117562 , -0.4793529 , ...,  1.18266357,
         -0.8117562 , -0.8117562 ],
        [-0.8117562 , -0.8117562 ,  0.85026027, ...,  1.18266357,
         -0.64555455, -0.8117562 ]]), array([[1., 0., 0., ..., 0., 0., 0.],
        [0., 1., 0., ..., 0., 0., 0.],
        [0., 0., 1., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 1., 0.],
        [0., 0., 0., ..., 0., 0., 1.],
        [0., 0., 0., ..., 0., 1., 0.]]), array([0, 1, 2, ..., 8, 9, 8]))
>>> len(data)
3
>>> |
```

← 入力データと正解値:1797個

画像の大きさ=8×8=64

0～9の10文字

data[0]

画像データの行列

data[1]

正解の行列one hot

data[2]

正解のベクトル

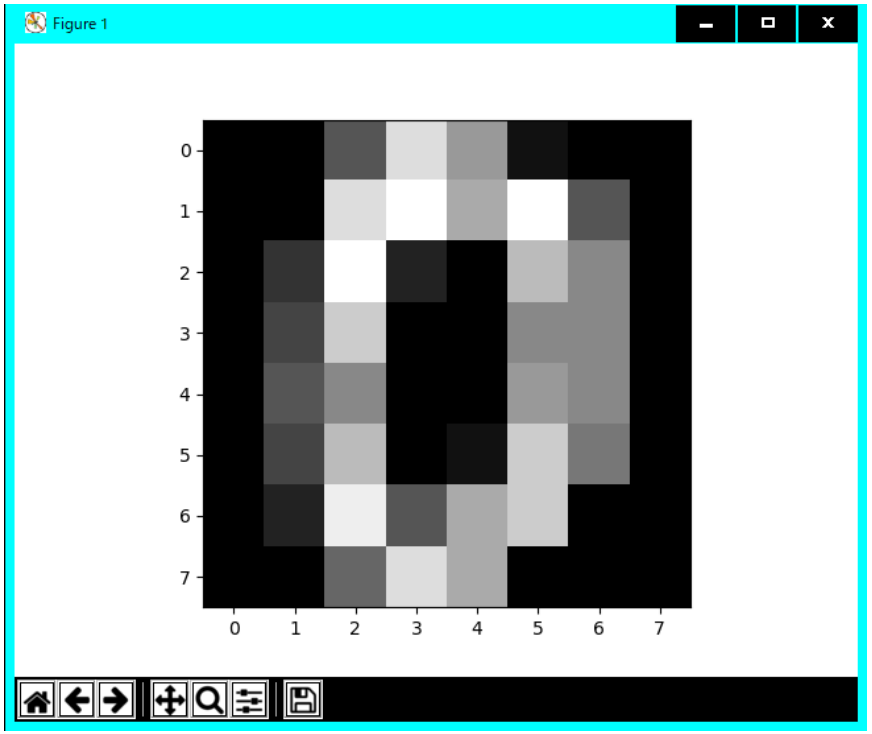
値(0～9のラベル)

← 3つの配列

手書き文字データを画像で確認

手書き文字データを表示してみよう

```
>>> DG.show_sample(data[0][0])
```



もっとニューラルネットワークらしく

手書き数字を読めるかどうかやってみよう

やり方はここまでと同じ

・入力データを用意

・パラメタを設定

s10

```
import numpy as np
import neuron1
from ufiesia import common_function_aid as cf
from ufiesia import sklearn_digits as DG

data = DG.get_data()
X = data[0] # input_data
C = data[1] # 正解値 one hot
T = data[2] # 正解値 id

Wm, Bm, Wo, Bo = cf.inherit_from('手書き文字認識NNbasicパラメタ')

middle_layer = neuron1.Layer(Wm, Bm, 'ramp')   # かくれ層
output_layer = neuron1.Layer(Wo, Bo)           # 出力層、出力はそのまま通過
middle_layer.forward(X)
output_layer.forward(middle_layer.Y) # かくれ層の出力が出力層の入力

Y = output_layer.Y
```

画像認識による

10カテゴリーの分類問題やねん

どうやって結果を見ようか？

実行したら結果が変数Yに入っている

```
sklearn digits のデータが読み込まれました
入力と正解値のデータが用意できました
入力データの形状（1797, 64）正解値の形状（1797, 10）
ヘッダー: NNbasic
入力数            = 64
中間層ニューロン数= 100
出力層ニューロン数= 10
学習済みのパラメータを提供します
>>> Y
array([[ 72.42499528, -70.84269357,   1.02500174, ...,  -7.29803033,
          4.46872672,  11.32184628],
       [-26.08372844,  59.38980494,  -3.84705777, ..., -13.23387557,
          5.04701622, -20.73895731],
       [ -7.5016654 ,  18.55122709,  58.53755004, ..., -25.02399245,
         23.38300661, -29.53092495],
       ...,
       [  5.60304564,  -1.21926681,   1.35649579, ..., -38.07124385,
         56.52607365,  -4.75196669],
       [  6.88467684,  -8.50853125, -24.00442434, ...,  -2.49488863,
          7.00514182,  46.61550954],
       [ -2.36108264, -10.66432032,  -3.20917424, ..., -27.05502774,
         46.93301921,   0.87542451]])
>>> Y.shape
(1797, 10)
>>> Y[0]
array([ 72.42499528, -70.84269357,   1.02500174,  -9.79627604,
        -8.54392048,  -3.24462172,  10.34729405,  -7.29803033,
         4.46872672,  11.32184628])
```

たしかに入ってるわ

← 1行が10個の並びでそれが1797行

最大値は左端の72.42499528
→'0'と判定

ニューラルネットワークで多用する活性化関数
ソフトマックス softmax

分類問題で多用されるトリッキーな関数

y の合計は 1 で、$0 < y < 1$ → 分類される確率

$$y = \frac{\exp(x)}{\sum_k \exp(xk)}$$

なんやけったいな計算やな

知らんがな

実際に計算してみると；

```
>>> Y[0]
array([ 72.42499528, -70.84269357,   1.02500174,  -9.79627604,
        -8.54392048,  -3.24462172,  10.34729405,  -7.29803033,
         4.46872672,  11.32184628])
>>> expY = np.exp(Y[0])
>>> smxY = expY / np.sum(expY)
>>> smxY
array([1.00000000e+00, 6.02051020e-63, 9.80340171e-32, 1.95774250e-36,
       6.84930776e-36, 1.37120658e-33, 1.09647039e-27, 2.38083828e-35,
       3.06879269e-30, 2.90562499e-27])
```

neuron2

活性化関数にソフトマックスを加えて分類問題にも便利なように改良

```
import numpy as np

def sigmoid(X):
    return 1 /(1 + np.exp(-X))

def ramp(X):
    return np.where(X>0, X, 0)

def step(X, t=0):
    return np.where(X>t, 1, 0)

def softmax(X):
    return np.exp(X) / np.sum(np.exp(X), axis=1, keepdims=True)

class Layer:                    # 層のクラス
    def __init__(self, W, B, act=None): # 初期値はインスタンス生成時に与える
        self.W = W
        self.B = B
        self.act = act

    def forward(self, X):   # 順伝播
        U = np.dot(X, self.W) + self.B
        if   self.act == 'step450': # 牛丼屋専用
            self.Y = step(U, 450)
        elif self.act == 'sigmoid':
            self.Y = sigmoid(U)
        elif self.act == 'ramp':
            self.Y = ramp(U)
        elif self.act == 'softmax':
            self.Y = softmax(U)
        elif self.act == 'step':
            self.Y = step(U)
        else:
            self.Y = U
```

こうしておいてよかったのかも...

なんやねん

モジュール化とかなんとか

偉い！

画像識別の分類問題をニューラルネットワークで

前に作ったのとほとんどまったく同じ

neuron1→neuron2、出力層の活性化関数に'softmax'を指定

s11

```
import numpy as np
import neuron2
from ufiesia import common_function_aid as cf
from ufiesia import sklearn_digits as DG

data = DG.get_data()
X = data[0] # input_data
C = data[1] # 正解値 one hot
T = data[2] # 正解値 id

Wm, Bm, Wo, Bo = cf.inherit_from('手書き文字認識NNbasicパラメタ')

middle_layer = neuron2.Layer(Wm, Bm, 'ramp')    # かくれ層
output_layer = neuron2.Layer(Wo, Bo, 'softmax') # 出力層、活性化関数はソフトマックス
middle_layer.forward(X)
output_layer.forward(middle_layer.Y) # かくれ層の出力が出力層の入力

Y = output_layer.Y
```

再び結果を確認

実行したら結果が変数Yに入っている

```
sklearn digits のデータが読み込まれました
入力と正解値のデータが用意できました
入力データの形状（1797, 64）正解値の形状（1797, 10）
ヘッダー: NNbasic
入力数            = 64
中間層ニューロン数= 100
出力層ニューロン数= 10
学習済みのパラメータを提供します
>>> Y
array([[1.00000000e+00, 6.02051020e-63, 9.80340171e-32, ...,
        2.38083828e-35, 3.06879269e-30, 2.90562499e-27],
       [7.57383958e-38, 1.00000000e+00, 3.44016656e-28, ...,
        2.88360851e-32, 2.50741792e-24, 1.58679459e-35],
       [2.08705141e-29, 4.30685836e-18, 1.00000000e+00, ...,
        5.12487409e-37, 5.40226366e-16, 5.65388905e-39],
       ...,
       [7.66319600e-23, 8.34674765e-26, 1.09687407e-24, ...,
        8.25861096e-42, 1.00000000e+00, 2.43941236e-27],
       [5.56056089e-18, 1.14797721e-24, 2.13871248e-31, ...,
        4.69488622e-22, 6.27243074e-18, 1.00000000e+00],
       [3.90700149e-22, 9.67817053e-26, 1.67310069e-22, ...,
        7.36884611e-33, 1.00000000e+00, 9.94124498e-21]])
>>> Y[0]
array([1.00000000e+00, 6.02051020e-63, 9.80340171e-32, 1.95774250e-36,
       6.84930776e-36, 1.37120658e-33, 1.09647039e-27, 2.38083828e-35,
       3.06879269e-30, 2.90562499e-27])
```

今度は少しわかりやすいやろか？

でもいちいち面倒！

どんなものか？

仮にすべて正解と一致していれば、

結果の行列 - one hotの正解の行列 ＝ ***0***（全要素が0の行列）

```
>>> Y-C
array([[ 0.00000000e+00,  6.02051020e-63,  9.80340171e-32, ...,
         2.38083828e-35,  3.06879269e-30,  2.90562499e-27],
       [ 7.57383958e-38,  0.00000000e+00,  3.44016656e-28, ...,
         2.88360851e-32,  2.50741792e-24,  1.58679459e-35],
       [ 2.08705141e-29,  4.30685836e-18, -4.44089210e-16, ...,
         5.12487409e-37,  5.40226366e-16,  5.65388905e-39],
       ...,
       [ 7.66319600e-23,  8.34674765e-26,  1.09687407e-24, ...,
         8.25861096e-42, -2.45248266e-13,  2.43941236e-27],
       [ 5.56056089e-18,  1.14797721e-24,  2.13871248e-31, ...,
         4.69488622e-22,  6.27243074e-18, -6.32827124e-15],
       [ 3.90700149e-22,  9.67817053e-26,  1.67310069e-22, ...,
         7.36884611e-33, -1.77752257e-11,  9.94124498e-21]])
>>> np.sum(Y-C)
2.2731719769313546e-15
>>> np.sum((Y-C)**2)
1.0158009482782757e-06
>>> np.sum(np.abs(Y-C))
0.0027831402375220722
```

ほとんど0に近い

← 合計しても0に近い（これはぬか喜び）

← 差の2乗でも絶対値でも合計して0に近い
つまり、すべて正解ではないか!!

まだ信じへんのやろか？

いあやそんなことは...

ニューラルネットワークの損失関数
二乗和誤差

出力と正解値の差の二乗の総和

$$E = \frac{1}{2}\sum_{k}(yk - tk)^2$$

実際に計算してみると⇒

```
>>> Y-C
array([[ 0.00000000e+00,  6.02051020e-63,  9.80340171e-32, ...,
         2.38083828e-35,  3.06879269e-30,  2.90562499e-27],
       [ 7.57383958e-38,  0.00000000e+00,  3.44016656e-28, ...,
         2.88360851e-32,  2.50741792e-24,  1.58679459e-35],
       [ 2.08705141e-29,  4.30685836e-18, -4.44089210e-16, ...,
         5.12487409e-37,  5.40226366e-16,  5.65388905e-39],
       ...,
       [ 7.66319600e-23,  8.34674765e-26,  1.09687407e-24, ...,
         8.25861096e-42, -2.45248266e-13,  2.43941236e-27],
       [ 5.56056089e-18,  1.14797721e-24,  2.13871248e-31, ...,
         4.69488622e-22,  6.27243074e-18, -6.32827124e-15],
       [ 3.90700149e-22,  9.67817053e-26,  1.67310069e-22, ...,
         7.36884611e-33, -1.77752257e-11,  9.94124498e-21]])
>>> np.sum(1/2 * np.sum((Y-C)**2))
5.079004741391378e-07
```

ニューラルネットワークの損失関数
交差エントロピー誤差

出力と正解値の分布の間のずれをあらわす

$$E = -\sum_{k} t_k \log(yk)$$

実際に計算してみると⇒

```
>>> Y
array([[1.00000000e+00, 6.02051020e-63, 9.80340171e-32, ...,
        2.38083828e-35, 3.06879269e-30, 2.90562499e-27],
       [7.57383958e-38, 1.00000000e+00, 3.44016656e-28, ...,
        2.88360851e-32, 2.50741792e-24, 1.58679459e-35],
       [2.08705141e-29, 4.30685836e-18, 1.00000000e+00, ...,
        5.12487409e-37, 5.40226366e-16, 5.65388905e-39],
       ...,
       [7.66319600e-23, 8.34674765e-26, 1.09687407e-24, ...,
        8.25861096e-42, 1.00000000e+00, 2.43941236e-27],
       [5.56056089e-18, 1.14797721e-24, 2.13871248e-31, ...,
        4.69488622e-22, 6.27243074e-18, 1.00000000e+00],
       [3.90700149e-22, 9.67817053e-26, 1.67310069e-22, ...,
        7.36884611e-33, 1.00000000e+00, 9.94124498e-21]])
>>> C
array([[1., 0., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 1., 0.]])
>>> - np.sum(C * np.log(Y))
0.0013918270123525795
```

neuron3

neuronのモジュールの中に損失関数を定義しておく

```
import numpy as np

def sigmoid(X):
    return 1 /(1 + np.exp(-X))

def ramp(X):
    return np.where(X>0, X, 0)

def step(X, t=0):
    return np.where(X>t, 1, 0)

def softmax(X):
    return np.exp(X) / np.sum(np.exp(X), axis=1, keepdims=True)

class Layer:                    # 層のクラス
    def __init__(self, W, B, act=None): # 初期値はインスタンス生成時に与える
        self.W = W
        self.B = B
        self.act = act

    def forward(self, X):    # 順伝播
        U = np.dot(X, self.W) + self.B
        if self.act == 'step450': # 牛丼屋専用
            self.Y = step(U, 450)
        elif self.act == 'sigmoid':
            self.Y = sigmoid(U)
        elif self.act == 'ramp':
            self.Y = ramp(U)
        elif self.act == 'softmax':
            self.Y = softmax(U)
        elif self.act == 'step':
            self.Y = step(U)
        else:
            self.Y = U
def loss_function(y, t, loss_f=None):  # 損失関数
    if loss_f == 'x_entropy':
        return -np.sum(t * np.log(y + 1e-7))  # 交差エントロピー誤差
    else:
        return 1/2 * np.sum(np.square(y - t)) # 2乗和誤差
```

logが−∞にならないように微小値を加算

(y − t)**2と同じ

画像識別の分類問題をニューラルネットワークで

neuron2→neuron3、損失関数('x_entropy'を指定)で結果を評価

判定結果を表示、対象画像も表示

s12

```
import numpy as np
import neuron3
from ufiesia import common_function_aid as cf
from ufiesia import sklearn_digits as DG

data = DG.get_data()
X = data[0] # input_data
C = data[1] # 正解値 one hot
T = data[2] # 正解値 id

Wm, Bm, Wo, Bo = cf.inherit_from('手書き文字認識NNbasicパラメタ')

middle_layer = neuron3.Layer(Wm, Bm, 'ramp')    # かくれ層
output_layer = neuron3.Layer(Wo, Bo, 'softmax') # 出力層、活性化関数はソフトマックス
middle_layer.forward(X)
output_layer.forward(middle_layer.Y) # かくれ層の出力が出力層の入力

Y = output_layer.Y
print('クロスエントロピー誤差 =', neuron3.loss_function(Y, C, 'x_entropy'))

while True:
    try:
        i = int(input('0から1797の数値を入れてください'))
    except:
        i = 9999 # 脱出のための便宜上
    if 0<=i<1797:
        print('判定結果=>', np.argmax(Y[i])) # 最大値のインデクスが判定結果
        DG.show_sample(X[i])
    else:
        break
```

なんか凄ぉ!

画像識別の分類問題をニューラルネットワークで

乱数を使って出鱈目なパラメタを与えてみる(ただし数だけは踏襲)

s13

```
import numpy as np
import neuron4
#from ufiesia import common_function_aid as cf
from ufiesia import sklearn_digits as DG

data = DG.get_data()
X = data[0] # input_data
C = data[1] # 正解値 one hot
T = data[2] # 正解値 id

Wm = np.random.randn(64, 100) * 0.01 # かくれ層のニューロンは100個
Bm = np.zeros(100)
Wo = np.random.randn(100, 10) * 0.01 # 出力層のニューロンは10個
Bo = np.zeros(10)

middle_layer = neuron4.Layer(Wm, Bm, 'ramp')     # かくれ層
output_layer = neuron4.Layer(Wo, Bo, 'softmax') # 出力層、活性化関数はソフトマックス
middle_layer.forward(X)
output_layer.forward(middle_layer.Y) # かくれ層の出力が出力層の入力

Y = output_layer.Y
print('クロスエントロピー誤差 =', neuron4.loss_function(Y, C, 'x_entropy'))

while True:
    try:
        i = int(input('0から1797の数値を入れてください'))
    except:
        i = 9999 # 脱出のための便宜上
    if 0<=i<len(X):
        print('判定結果=>', np.argmax(Y[i])) # 最大値のインデクスが判定結果
        DG.show_sample(X[i])
    else:
        break
```

でたらめな俺にはでたらめなほうが...

neuron4

外で定義しておいた活性化関数をLayerのクラスの中に取り込んでおく

その際に牛丼屋専用の部分は外す(もういいよね？)

```
class Layer:                    # 層のクラス
    def __init__(self, W, B, act=None): # 初期値はインスタンス生成時に与える
        self.W = W
        self.B = B
        self.act = act

    def forward(self, X):   # 順伝播
        self.X = X
        U = np.dot(X, self.W) + self.B
        self.U = U

        # 活性化関数
        if self.act == 'sigmoid':
            self.Y = 1 /(1 + np.exp(-U))
        elif self.act == 'ramp':
            self.Y = np.where(U>0, U, 0)
        elif self.act == 'softmax':
            self.Y = np.exp(U) / np.sum(np.exp(U), axis=1, keepdims=True)
        elif self.act == 'step':
            self.Y = np.where(U>0, 1, 0)
        else:
            self.Y = U
```

牛丼屋懐かしな～

まああうれしおす

そんでどこの生まれでっか？

neuron4

neuronのモジュールの中のLayerのクラスに、

機械学習に必要な逆伝播と更新の関数を定義しておく

```
def backward(self, grad_y): # 逆伝播
    self.grad_y = grad_y

    # 活性化関数の微分
    if   self.act == 'sigmoid':
        self.delta = self.grad_y * (1 - self.Y) * self.Y
    elif self.act == 'ramp':
        self.delta = self.grad_y * np.where(self.U <=0, 0, 1)
    elif self.act == 'softmax':
        self.delta = self.Y - self.grad_y
    else:
        self.delta = self.Y - self.grad_y

    self.grad_w = np.dot(self.X.T, self.delta)
    self.grad_b = np.sum(self.delta, axis=0)
    self.grad_x = np.dot(self.delta, self.W.T)

def update(self):
    self.W   -= 0.01 * self.grad_w
    self.B   -= 0.01 * self.grad_b
```

まずは動かしてみよう

neuron4

Layerクラス全体

```
class Layer:                    # 層のクラス
    def __init__(self, W, B, act=None): # 初期値はインスタンス生成時に与える
        self.W = W
        self.B = B
        self.act = act

    def forward(self, X):    # 順伝播
        self.X = X
        U = np.dot(X, self.W) + self.B
        self.U = U

        # 活性化関数
        if self.act == 'sigmoid':
            self.Y = 1 /(1 + np.exp(-U))
        elif self.act == 'ramp':
            self.Y = np.where(U>0, U, 0)
        elif self.act == 'softmax':
            self.Y = np.exp(U) / np.sum(np.exp(U), axis=1, keepdims=True)
        elif self.act == 'step':
            self.Y = np.where(U>0, 1, 0)
        else:
            self.Y = U

    def backward(self, grad_y): # 逆伝播
        self.grad_y = grad_y

        # 活性化関数の微分
        if    self.act == 'sigmoid':
            self.delta = self.grad_y * (1 - self.Y) * self.Y
        elif self.act == 'ramp':
            self.delta = self.grad_y * np.where(self.U <=0, 0, 1)
        elif self.act == 'softmax':
            self.delta = self.Y - self.grad_y
        else:
            self.delta = self.Y - self.grad_y

        self.grad_w = np.dot(self.X.T, self.delta)
        self.grad_b = np.sum(self.delta, axis=0)
        self.grad_x = np.dot(self.delta, self.W.T)

    def update(self):
        self.W   -= 0.01 * self.grad_w
        self.B   -= 0.01 * self.grad_b
```

逆伝播と更新についての説明は、参考文献に譲り、ここではざっくりと説明しておきます

ニューラルネットワークのパラメタは、はじめは根拠のない乱数です*1。そして、その状態で入力を与え順伝播して結果を出します（neuron4の関数forward）。そしてその結果から個々のパラメタを大きくした方が良さそうか、小さくした方が良さそうかを決めます（neuron4の関数backward）。どちらが良さそうかは、大雑把に言うと正解値に近づく方向です*2。そして個々のパラメタについて求めた方向に少しだけ調整します（neuron4の関数update）*3。この調整はすべてのパラメタをいっぺんに行います。そして、またふたたびデータを入力して順伝播するということを繰り返して、次第にニューラルネットワークが正解値を出力するように調整していきます。

*1：ディープラーニングでは特徴量を人が決めるのではなく、データをもとに機械学習で獲得します。そのためには、初期値として恣意的な要因が入っていると、かえって機械学習の妨げとなります。
*2：損失関数を定義して、あるべき状態との乖離を評価し、これが小さくなるようにします。
一般的に損失関数は、二乗和誤差もしくは交差エントロピー誤差に正則化項を加えたものになります。
*3：損失関数にしたがって、個々のパラメタの勾配を求めます。そしてたとえば、勾配が正のある値ならば、そのパラメタをその値に応じ学習係数を乗じた分だけ小さくし、勾配が負のある値ならば、そのパラメタをその値に応じ学習係数を乗じた分だけ大きくします。勾配は個々のパラメタの誤差に対する偏微分によって算出されますが、誤差逆伝播法によって出力から入力へと層ごとに遡って高速に計算することができます。

どこで機械学習すればよいのか？

乱数からなるでたらめなパラメータでニューラルネットワークをいったん構築して、
その後で、順伝播して結果を確認する前に、「機械学習」によってパラメータを調整

この間で「学習」

```
import numpy as np
import neuron4
#from ufiesia import common_function_aid as cf
from ufiesia import sklearn_digits as DG

data = DG.get_data()
X = data[0] # input_data
C = data[1] # 正解値 one hot
T = data[2] # 正解値 id

Wm = np.random.randn(64, 100) * 0.01 # かくれ層のニューロンは100個
Bm = np.zeros(100)
Wo = np.random.randn(100, 10) * 0.01 # 出力層のニューロンは10個
Bo = np.zeros(10)

middle_layer = neuron4.Layer(Wm, Bm, 'ramp')     # かくれ層
output_layer = neuron4.Layer(Wo, Bo, 'softmax') # 出力層、活性化関数はソフトマックス
middle_layer.forward(X)
output_layer.forward(middle_layer.Y) # かくれ層の出力が出力層の入力

Y = output_layer.Y
print('クロスエントロピー誤差 =', neuron4.loss_function(Y, C, 'x_entropy'))

while True:
    try:
        i = int(input('0から1797の数値を入れてください'))
    except:
        i = 9999 # 脱出のための便宜上
    if 0<=i<len(X):
        print('判定結果=>', np.argmax(Y[i])) # 最大値のインデクスが判定結果
        DG.show_sample(X[i])
    else:
        break
```

ほんまかいな？

機械学習の部分

入力データXは、8×8＝64の大きさの画像が、1797個 → len(X)=1797

range文で、0～len(X)の整数の並びを作り、

for文で、idxにその整数を0から順にひとつずつ取り出してlen(X)より小さい間、

繰返す

```
for idx in range(len(X)):    # データすべてをひとなめ
    x = X[idx:idx+1]
    c = C[idx:idx+1]

    middle_layer.forward(x)
    output_layer.forward(middle_layer.Y) # かくれ層の出力が出力層の入力

    output_layer.backward(c)
    middle_layer.backward(output_layer.grad_x)

    middle_layer.update()
    output_layer.update()
```

for文の中では；

入力データXと、正解値Cから、idxで指定される1個ずつをxとcに取り出して、

（インデクスの指定を[idx]とせずに[idx:idx+1]とするのは次元を保つため）

順伝播→逆伝播→更新を1セットにして行い、次のidxについて同じことを繰り返す

これでええんかいな？

データをひとなめするのを「エポック」言いますねん

画像識別の分類問題をニューラルネットワークで

機械学習を行うプログラム全体

s14

```
import numpy as np
import neuron4
from ufiesia import sklearn_digits as DG

data = DG.get_data()
X = data[0] # input_data
C = data[1] # 正解値 one hot
T = data[2] # 正解値 id

Wm = np.random.randn(64, 100) * 0.01
Bm = np.zeros(100)
Wo = np.random.randn(100, 10) * 0.01
Bo = np.zeros(10)

middle_layer = neuron4.Layer(Wm, Bm, 'ramp')     # かくれ層
output_layer = neuron4.Layer(Wo, Bo, 'softmax') # 出力層、活性化関数はソフトマックス

for idx in range(len(X)):     # データすべてをひとなめ
    x = X[idx:idx+1]
    c = C[idx:idx+1]

    middle_layer.forward(x)
    output_layer.forward(middle_layer.Y) # かくれ層の出力が出力層の入力

    output_layer.backward(c)
    middle_layer.backward(output_layer.grad_x)

    middle_layer.update()
    output_layer.update()

middle_layer.forward(X)
output_layer.forward(middle_layer.Y) # かくれ層の出力が出力層の入力
Y = output_layer.Y
print('クロスエントロピー誤差 =', neuron4.loss_function(Y, C, 'x_entropy'))

while True:
    try:
        i = int(input('0から1797の数値を入れてください'))
    except:
        i = 9999 # 脱出のための便宜上
    if 0<=i<len(X):
        print('判定結果=>', np.argmax(Y[i])) # 最大値のインデクスが判定結果
        DG.show_sample(X[i])
    else:
        break
```

だんだんえらいことに

機械学習の効果を確認

乱数を使って出鱈目なパラメタで順伝播した場合と、機械学習を行った場合では、クロスエントロピー誤差が約6.5分の1に小さくなっている

実際、番号指定(0～1797)して手書きもじ数字が読めるようになっているのを確認できる

s13:でたらめなパラメタ

```
sklearn digits のデータが読み込まれました
入力と正解値のデータが用意できました
入力データの形状 (1797, 64) 正解値の形状 (1797, 10)
クロスエントロピー誤差 = 4136.159803812766
0から1797の数値を入れてください
```

s14:データをひとなめ学習

```
sklearn digits のデータが読み込まれました
入力と正解値のデータが用意できました
入力データの形状 (1797, 64) 正解値の形状 (1797, 10)
クロスエントロピー誤差 = 631.9052676394609
0から1797の数値を入れてください
```

小さくなった

機械学習ではなんども学習するって聞いたで

よう知ってはりますなあ、そんなら何回もやってみたら...

機械学習の部分を繰り返す

データをひとなめしている機械学習の部分を繰り返すように、
さらにその外側でfor 文でループさせる

```
for i in range(100):
    for idx in range(len(X)):    # データすべてをひとなめ
        x = X[idx:idx+1]
        c = C[idx:idx+1]

        middle_layer.forward(x)
        output_layer.forward(middle_layer.Y) # かくれ層の出力が出力層の入力

        output_layer.backward(c)
        middle_layer.backward(output_layer.grad_x)

        middle_layer.update()
        output_layer.update()
```

なんや1行増えただけや！

インデントを忘れずに！

???

画像識別の分類問題をニューラルネットワークで

s15

機械学習を行うプログラム全体

```
import numpy as np
import neuron4
from ufiesia import sklearn_digits as DG

data = DG.get_data()
X = data[0] # input_data
C = data[1] # 正解値 one hot
T = data[2] # 正解値 id

Wm = np.random.randn(64, 100) * 0.01
Bm = np.zeros(100)
Wo = np.random.randn(100, 10) * 0.01
Bo = np.zeros(10)

middle_layer = neuron4.Layer(Wm, Bm, 'ramp')     # かくれ層
output_layer = neuron4.Layer(Wo, Bo, 'softmax') # 出力層、活性化関数はソフトマックス

for i in range(100):
    for idx in range(len(X)):    # データすべてをひとなめ
        x = X[idx:idx+1]
        c = C[idx:idx+1]

        middle_layer.forward(x)
        output_layer.forward(middle_layer.Y) # かくれ層の出力が出力層の入力

        output_layer.backward(c)
        middle_layer.backward(output_layer.grad_x)

        middle_layer.update()
        output_layer.update()

    middle_layer.forward(X)
    output_layer.forward(middle_layer.Y) # かくれ層の出力が出力層の入力
    Y = output_layer.Y
    print(i, 'クロスエントロピー誤差 =', neuron4.loss_function(Y, C, 'x_entropy'))

while True:
    try:
        i = int(input('0から1797の数値を入れてください'))
    except:
        i = 9999 # 脱出のための便宜上
    if 0<=i<len(X):
        print('判定結果=>', np.argmax(Y[i])) # 最大値のインデクスが判定結果
        DG.show_sample(X[i])
    else:
        break
```

ますますえらいことに

学習が進む様子を確認

```
sklearn digits のデータが読み込まれました
入力と正解値のデータが用意できました
入力データの形状（1797, 64）正解値の形状（1797, 10）
0 クロスエントロピー誤差 = 616.6512601870937
1 クロスエントロピー誤差 = 319.84351575018
2 クロスエントロピー誤差 = 230.35348579989022
3 クロスエントロピー誤差 = 182.96782138892593
4 クロスエントロピー誤差 = 153.19199501807827
5 クロスエントロピー誤差 = 133.4707062721448
6 クロスエントロピー誤差 = 118.87369104674528
7 クロスエントロピー誤差 = 107.55206208556923
8 クロスエントロピー誤差 = 100.39572634296587
9 クロスエントロピー誤差 = 94.00494411619768
10 クロスエントロピー誤差 = 86.54826356127444
11 クロスエントロピー誤差 = 81.66212446501494
12 クロスエントロピー誤差 = 75.71595524751723
13 クロスエントロピー誤差 = 71.58394332608034
14 クロスエントロピー誤差 = 65.87258162373107
15 クロスエントロピー誤差 = 62.41987911400946
              ⋮                    ⋮
91 クロスエントロピー誤差 = 1.5905569685135417
92 クロスエントロピー誤差 = 1.5677616235171588
93 クロスエントロピー誤差 = 1.5447204156305157
94 クロスエントロピー誤差 = 1.5231087917774833
95 クロスエントロピー誤差 = 1.5024561872390485
96 クロスエントロピー誤差 = 1.4815990944482431
97 クロスエントロピー誤差 = 1.461977422336666
98 クロスエントロピー誤差 = 1.442227662541089
99 クロスエントロピー誤差 = 1.4230031544141215
0から1797の数値を入れてください
```

どんどん学習する！

どうや、すごいやろ

ついでにもひとつ教えたる！

ミニバッチ学習

データをひとなめしている機械学習の部分で複数データ(ここでは100個)をまとめて扱うようにする

```
for i in range(100):
    for idx in range(0, len(X), 100):    # 0～len(X)まで100ずつ更新
        x = X[idx:idx+100]               # 100個ずつ
        c = C[idx:idx+100]               # 100個ずつ

        middle_layer.forward(x)
        output_layer.forward(middle_layer.Y) # かくれ層の出力が出力層の入力

        output_layer.backward(c)
        middle_layer.backward(output_layer.grad_x)

        middle_layer.update()
        output_layer.update()
```

なんやほとんどいっしょや！

「ミニバッチ学習」言いますねん

???

ちなみに前のように一つずつのんは「オンライン学習」

オンラインって聞いたことあるよな？

ついでに全データいっぺんにやったら「バッチ学習」

画像識別の分類問題をニューラルネットワークで

s16

機械学習を行うプログラム全体

```
import numpy as np
import neuron4
from ufiesia import sklearn_digits as DG

data = DG.get_data()
X = data[0] # input_data
C = data[1] # 正解値 one hot
T = data[2] # 正解値 id

Wm = np.random.randn(64, 100) * 0.01
Bm = np.zeros(100)
Wo = np.random.randn(100, 10) * 0.01
Bo = np.zeros(10)

middle_layer = neuron4.Layer(Wm, Bm, 'ramp')     # かくれ層
output_layer = neuron4.Layer(Wo, Bo, 'softmax') # 出力層、活性化関数はソフトマックス

for i in range(100):
    for idx in range(0, len(X), 100):     # 0～len(X)まで100ずつ更新
        x = X[idx:idx+100]                # 100個ずつ
        c = C[idx:idx+100]                # 100個ずつ

        middle_layer.forward(x)
        output_layer.forward(middle_layer.Y) # かくれ層の出力が出力層の入力

        output_layer.backward(c)
        middle_layer.backward(output_layer.grad_x)

        middle_layer.update()
        output_layer.update()

    middle_layer.forward(X)
    output_layer.forward(middle_layer.Y) # かくれ層の出力が出力層の入力
    Y = output_layer.Y
    print(i, 'クロスエントロピー誤差 =', neuron4.loss_function(Y, C, 'x_entropy'))

while True:
    try:
        i = int(input('0から1797の数値を入れてください'))
    except:
        i = 9999 # 脱出のための便宜上
    if 0<=i<len(X):
        print('判定結果=>', np.argmax(Y[i])) # 最大値のインデクスが判定結果
        DG.show_sample(X[i])
    else:
        break
```

ほとんどいっしょや！

そうどす！

もうあんたになれたわ

うれしおす

参考文献

はじめてのディープラーニング　我妻幸長 著　SBクリエイティブ株式会社

ゼロから作るDeep Learning　斎藤康毅 著　株式会社オライリー・ジャパン

入門Python3 Bill Lubanovic著/斎藤康毅 監訳/長尾高弘 訳　株式会社オライリー・ジャパン

わけがわかる機械学習　中谷秀洋 著　株式会社技術評論社

Pythonで動かして学ぶ！あたらしい数学の教科書 我妻幸長 著　株式会社翔泳社

必要な数学だけでわかるディープラーニングの理論と実装 チーム・カルボ著　株式会社秀和システム

Python　機械学習プログラミング　Sebastian Raschka・Vahid Mirjalili著/株式会社クイープ訳/福島真太朗 監訳

株式会社インプレス

人工知能は人間を超えるのか　ディープラーニングの先にあるもの　松尾豊 著　株式会社KADOKAWA

線形代数入門　斎藤正彦 著　財団法人東京大学出版会

おわり