プログラミング勉強会

やってみようぜ、ディープラーニングのプログラミング

2017年~~5月28日~~ **7月22日 改訂版**

名古屋校 2011期 越智 由浩

今日にいたる経緯
～名古屋校 プログラミング勉強会 クラブ活動～

メモ：リンクあり

9月
25日
プログラミング ハンズオン&ワークショ...
日 14:30 - グロービス 名古屋校 101教室
プログラミング勉強会主催

10月
30日
実践！Excel VBA入門／プログラミング ...
日 14:30 - グロービス 名古屋校 101教室
プログラミング勉強会主催

11月
27日
IoTコトハジメ 世界でたった一つだけ...
日 14:00 - グロービス 名古屋校 101教室
プログラミング勉強会主催

2月
19日
野呂さん公認・パクリ版＊ゼロからのR...
日 14:00 - グロービス名古屋 101教室
プログラミング勉強会主催

4月
8日
ここ中部圏をプログラミング教育のメッ...
土 14:00 - グロービス名古屋校 101教室室
プログラミング勉強会主催

5月
28日
やってみようぜ、ディープラーニングの...
日 14:00 - グロービス名古屋校 101教室室
プログラミング勉強会主催

今日の話の軸足

応用・展開

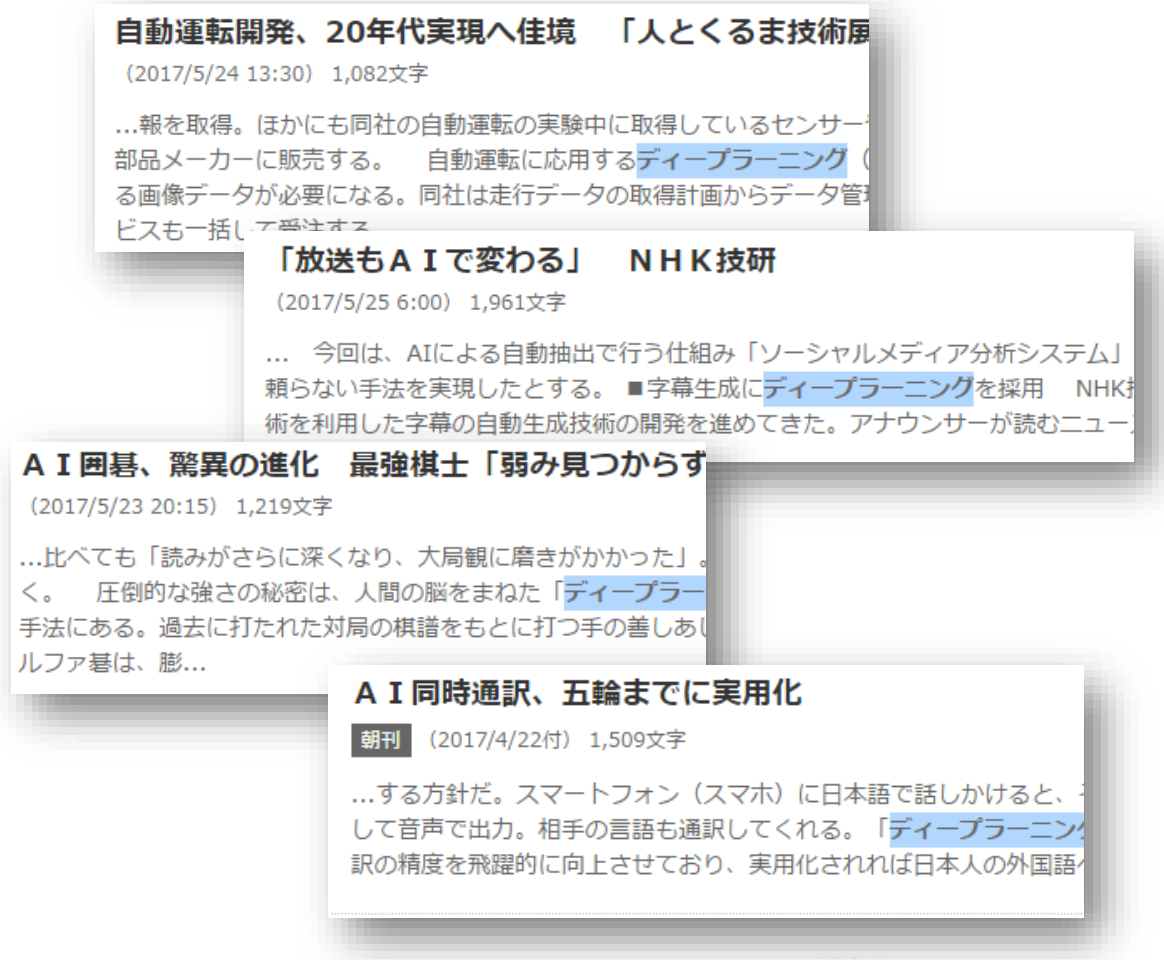
ディープラーニングで何ができるのか、世の中がどう変わるのか

ではなく、

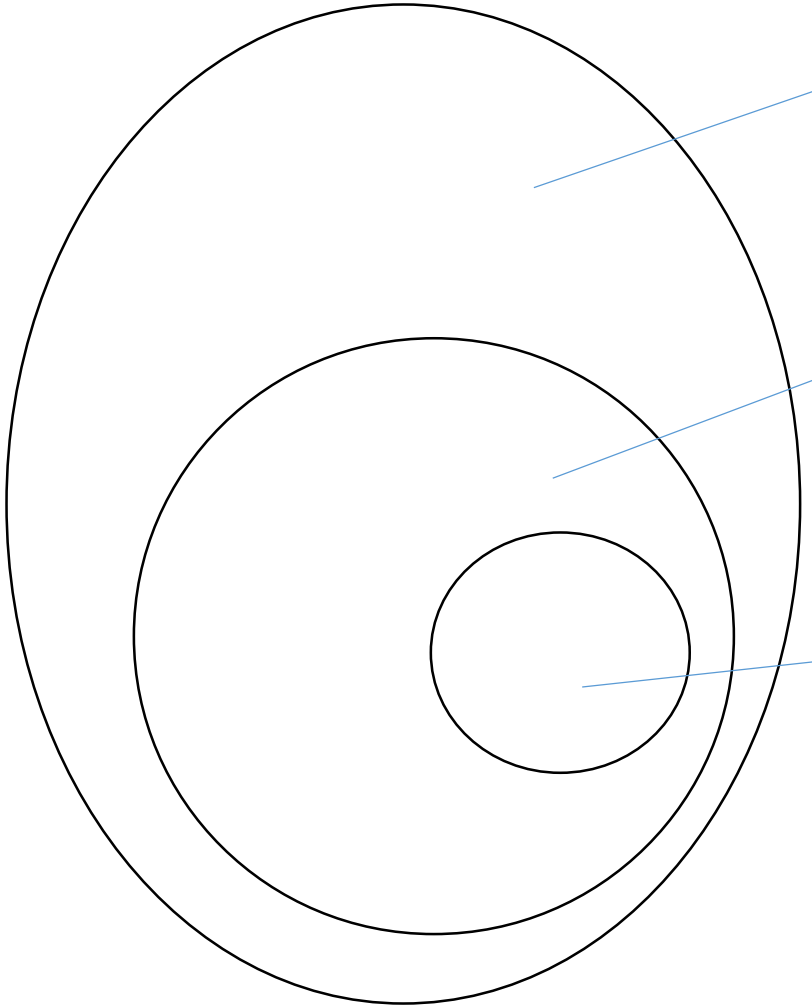
基礎(中の仕組み)

ディープラーニングってそもそも中身は何をやっているのか

今日の軸足はこっち



まず、言葉の整理



人工知能

機械があたかも知性があるかのように振る舞うこと

機械学習

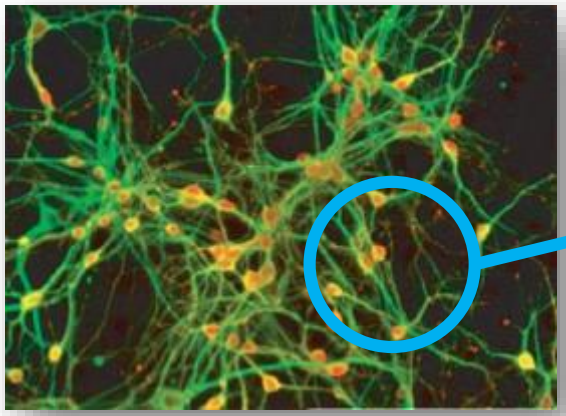
人口知能を実現するやり方のひとつ。
機械に学習させることでデータからパターンを
見い出し、新たなデータから将来を予測すること

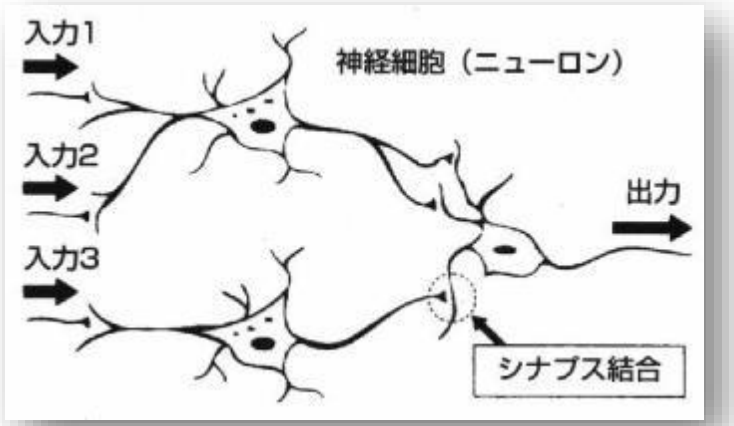
ディープラーニング

多層構造のニューラルネットワークを用いた
機械学習

ニューラルネットワークモデル

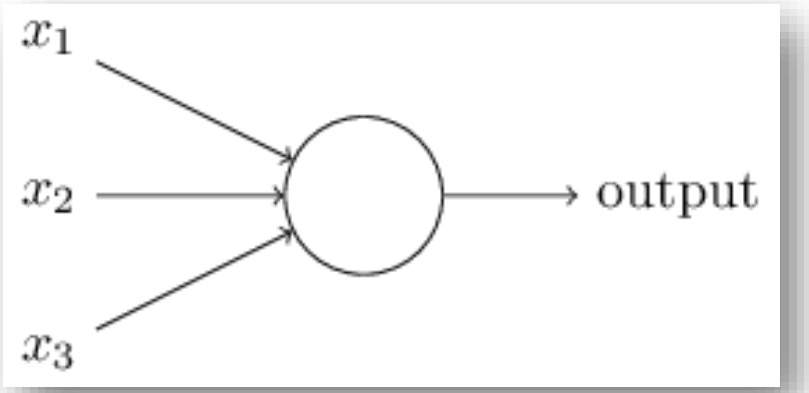
実物

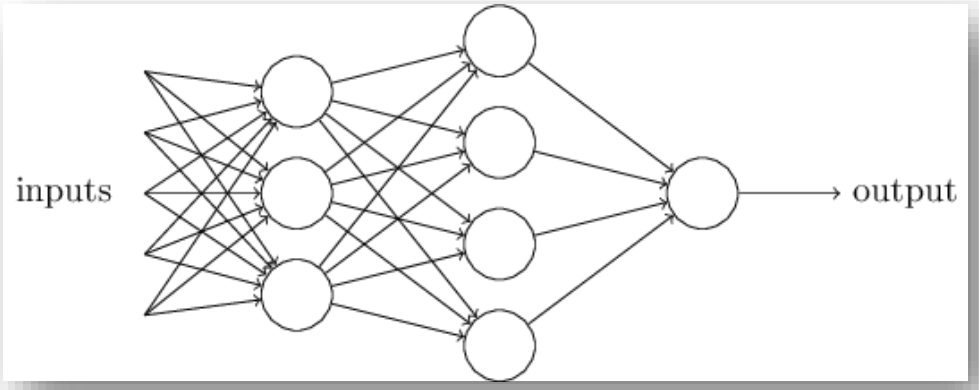




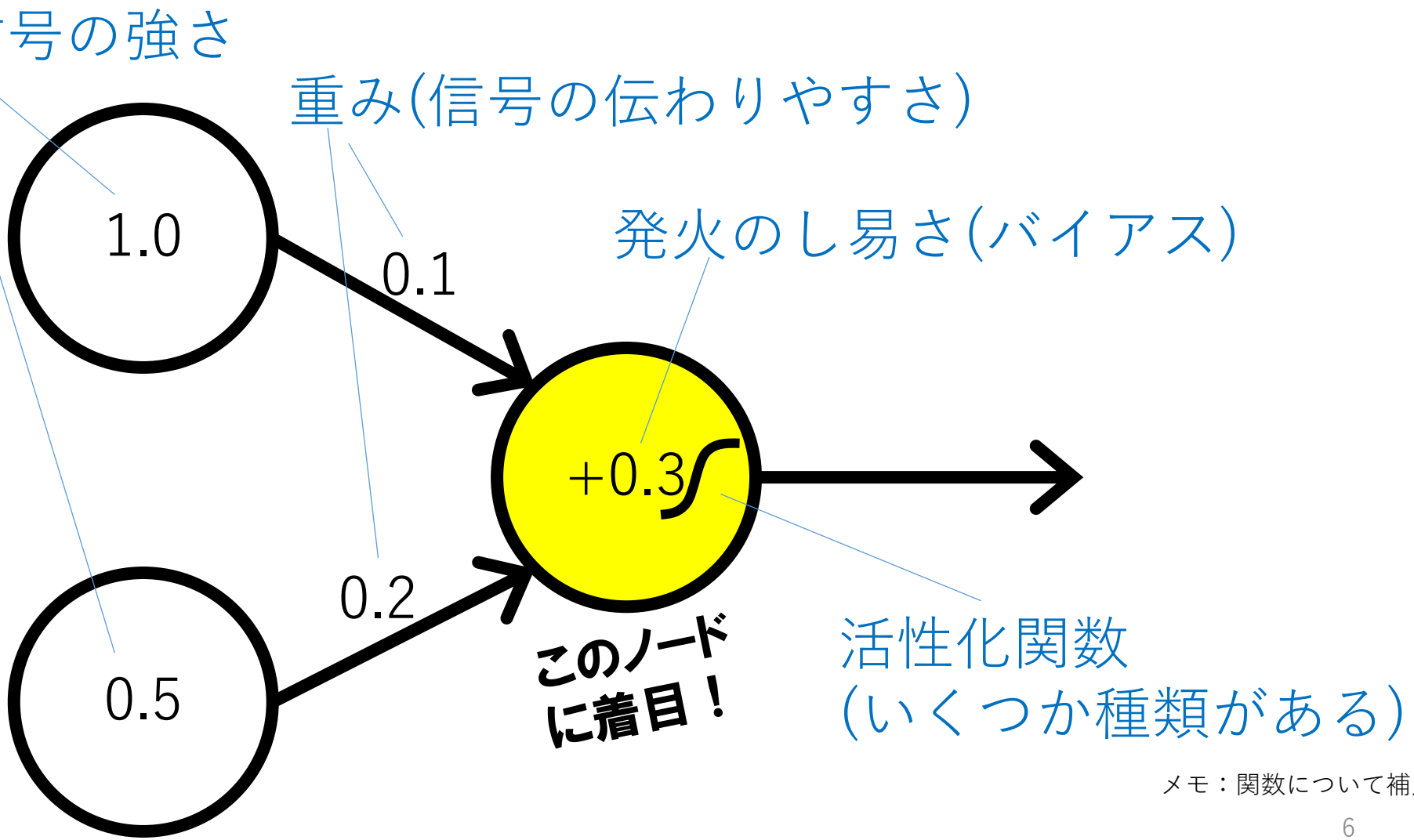
抽象的に表現

モデル
計算できる”模型”





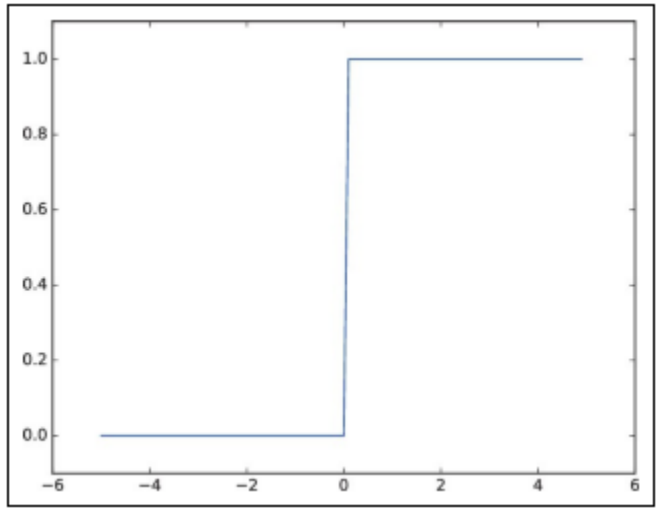
ニューラルネットワークモデルの計算ルール



メモ：関数について補足

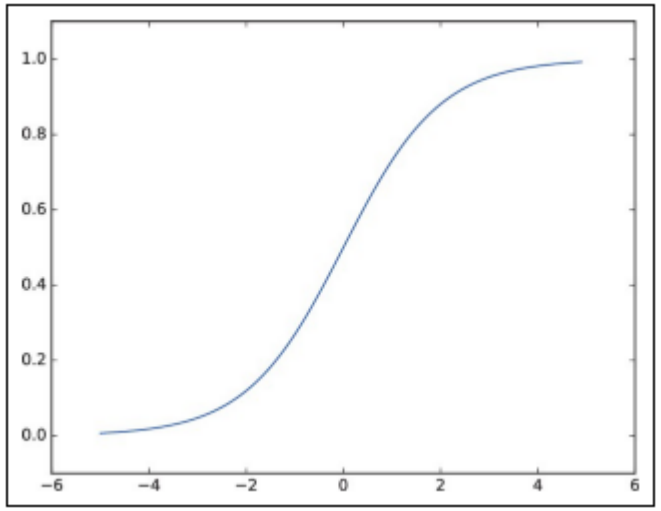
活性化関数

ステップ関数



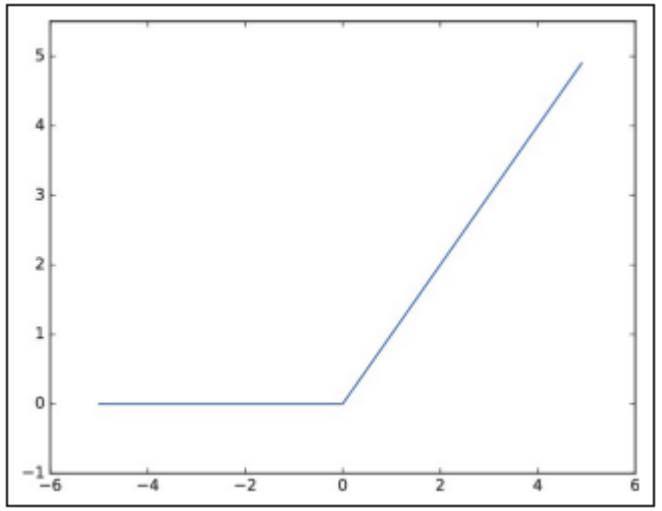
$$h(x) = \begin{cases} 0 & (x \leq 0) \\ 1 & (x > 0) \end{cases}$$

シグモイド関数



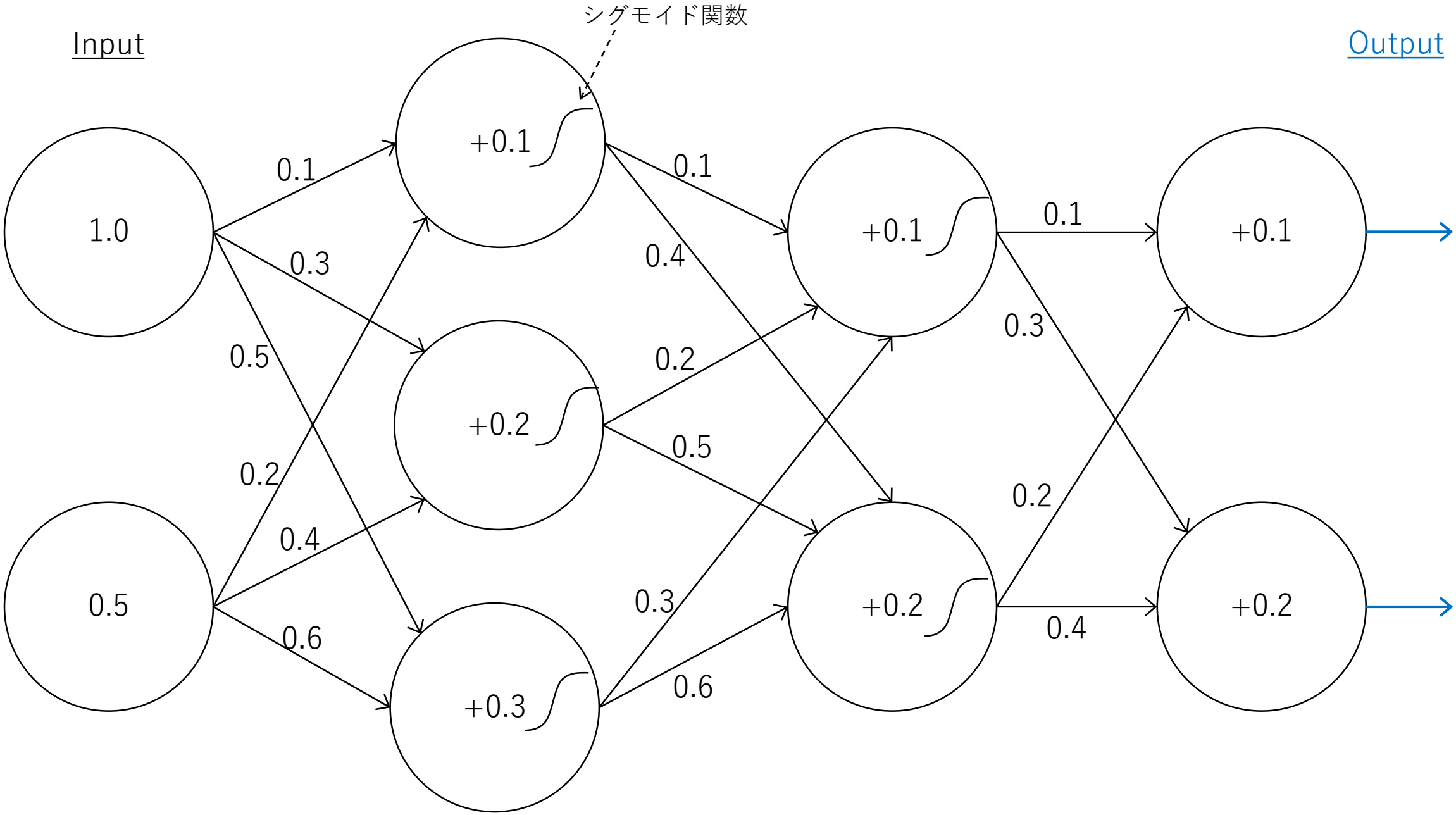
$$h(x) = \frac{1}{1 + \exp(-x)}$$

ReLU関数
(Rectified Linear Unit)

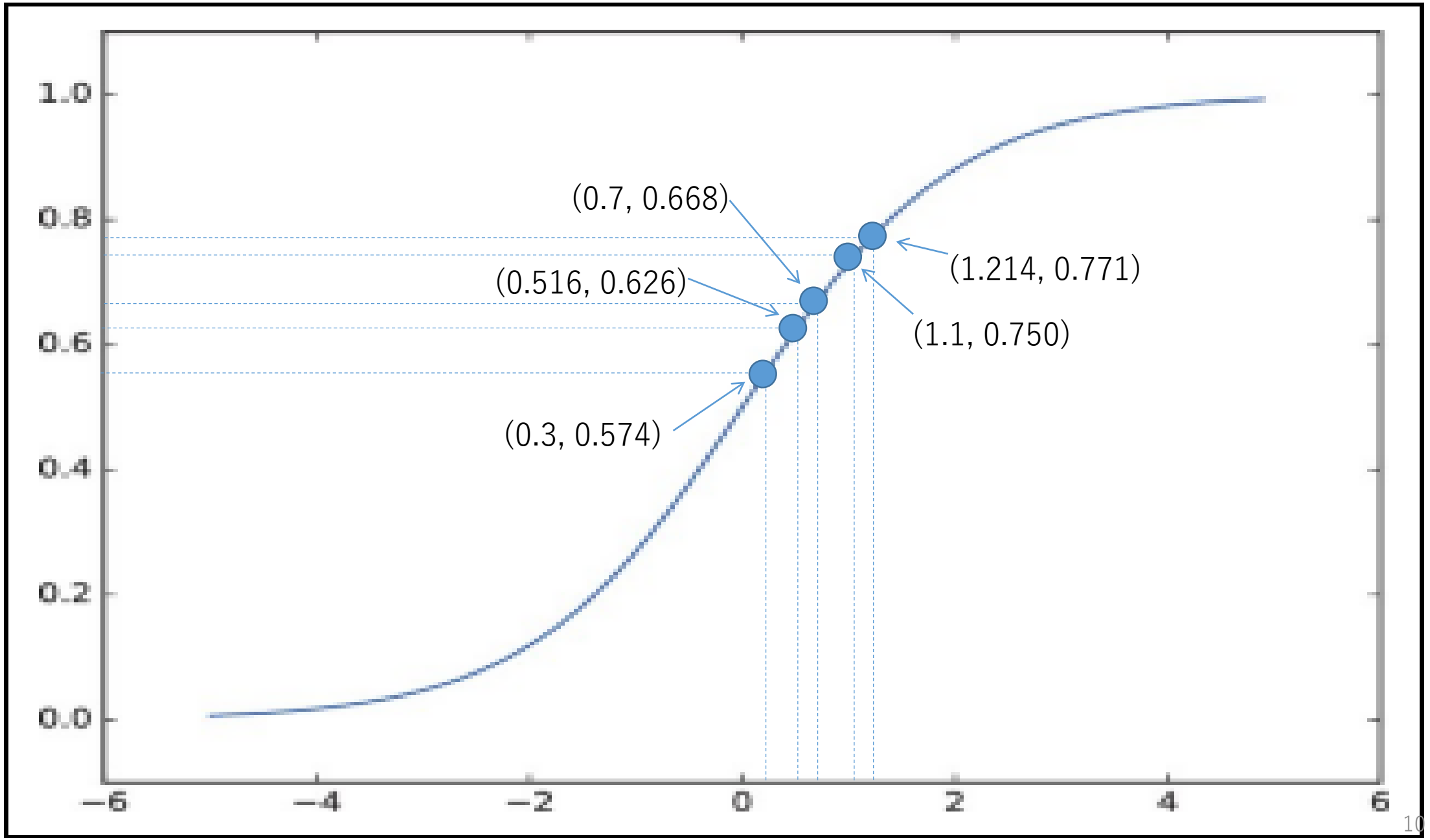


$$h(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

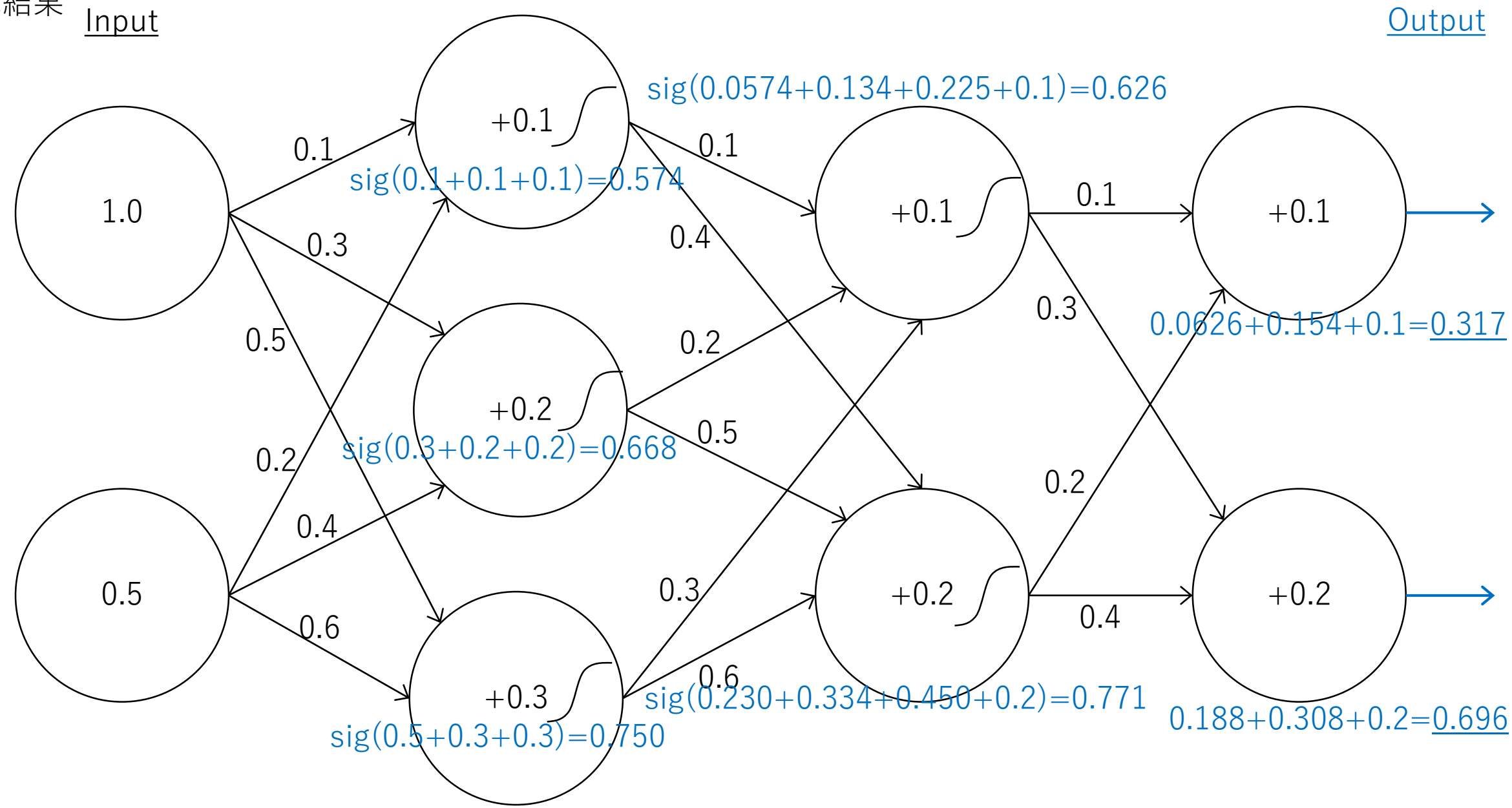
手触り感持って”模型”と慣れ親しむために
手で計算してみる


シグモイド関数
Input
Output
1.0
0.5
+0.1
+0.2
+0.3
+0.1
+0.2
+0.1
+0.2
0.1
0.3
0.5
0.2
0.4
0.6
0.1
0.4
0.2
0.5
0.3
0.6
0.1
0.3
0.2
0.4

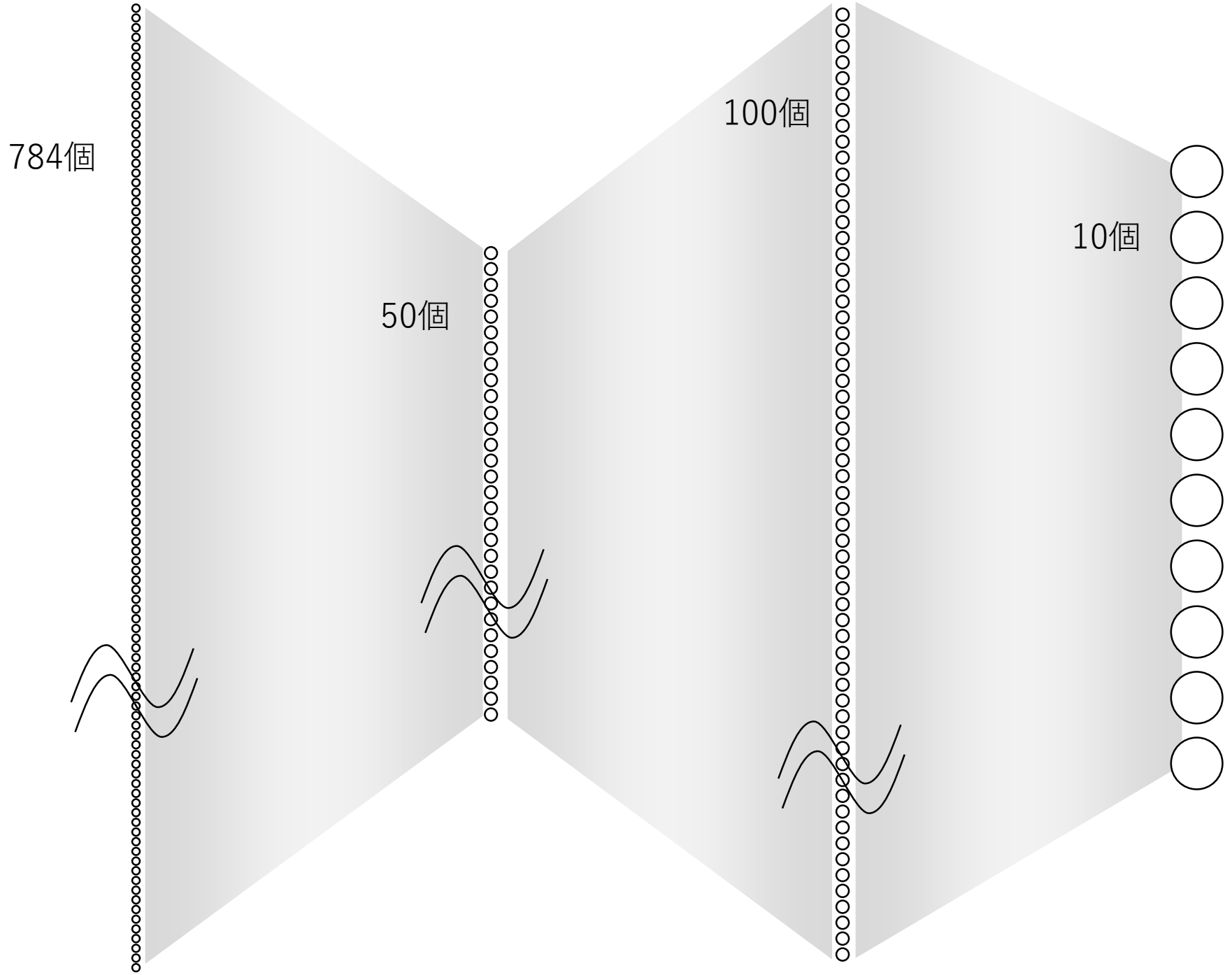
シグモイド関数、参照シート



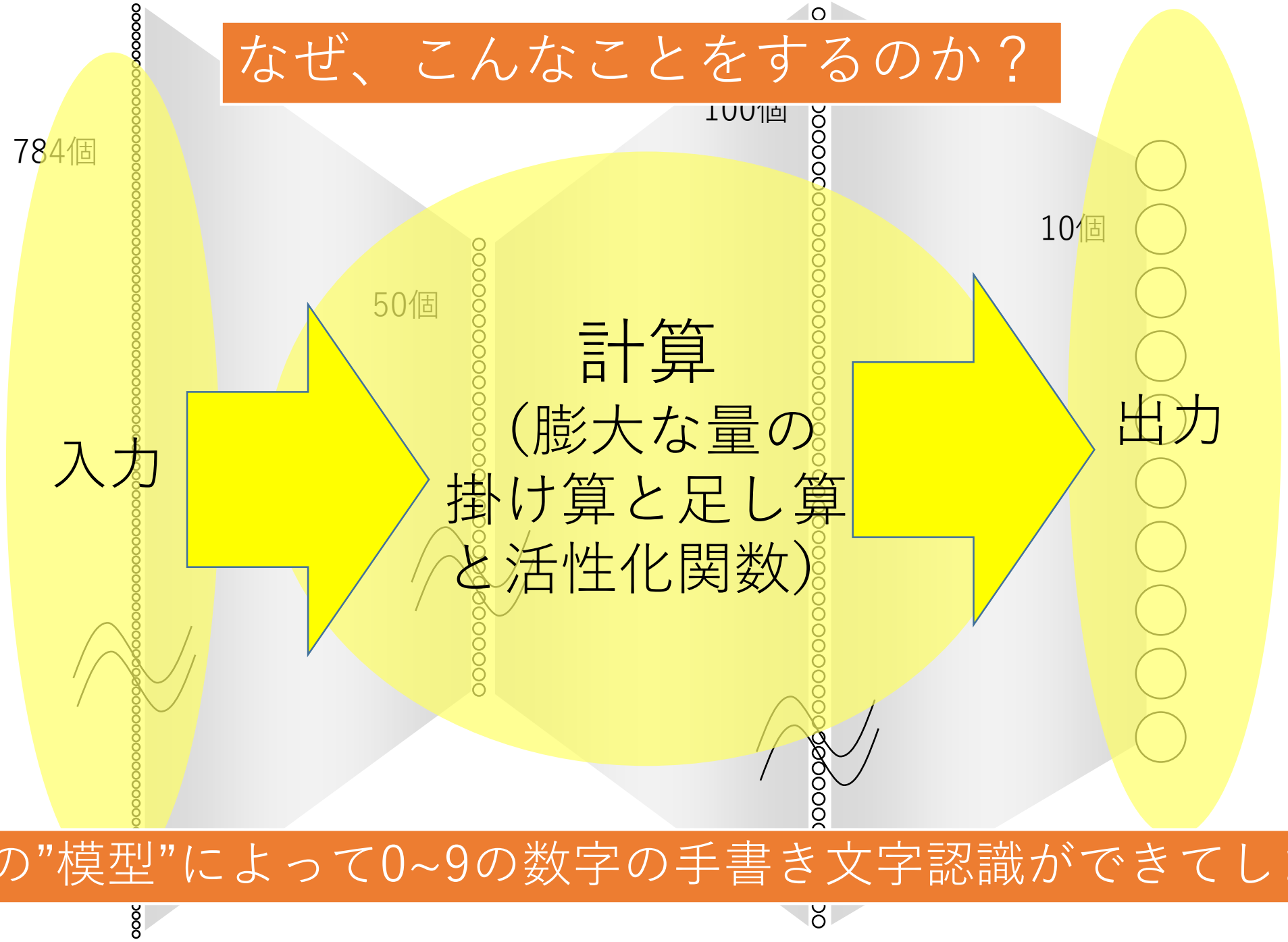
手計算結果



sig()：シグモイド関数


784個
50個
100個
10個

なぜ、こんなことをするのか？



この”模型”によって0~9の数字の手書き文字認識ができてしまう！

手計算じゃムリ！

計算
（膨大な量の
掛け算と足し算
と活性化関数）

単調な計算の繰り返しは
コンピューターの得意技。
プログラムを作って計算
させちゃえばいい！

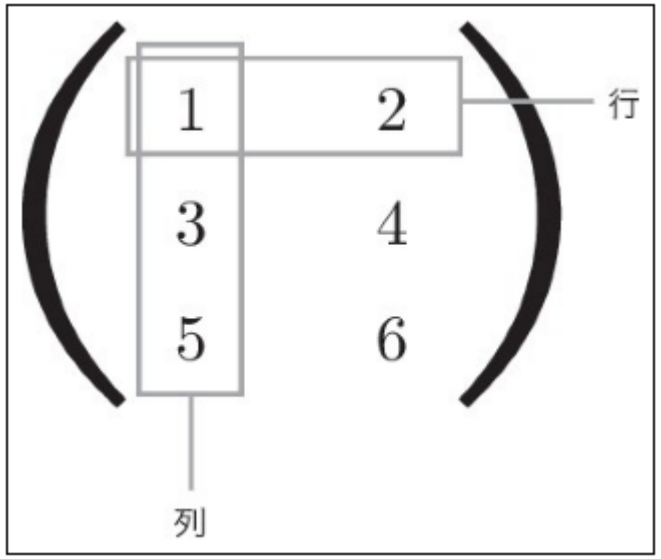
ふたつの武器を手に入れよう

- 数学の道具 – 「行列」と「行列の内積」
- プログラミング – Python（なぜPythonかは後ほど補足）

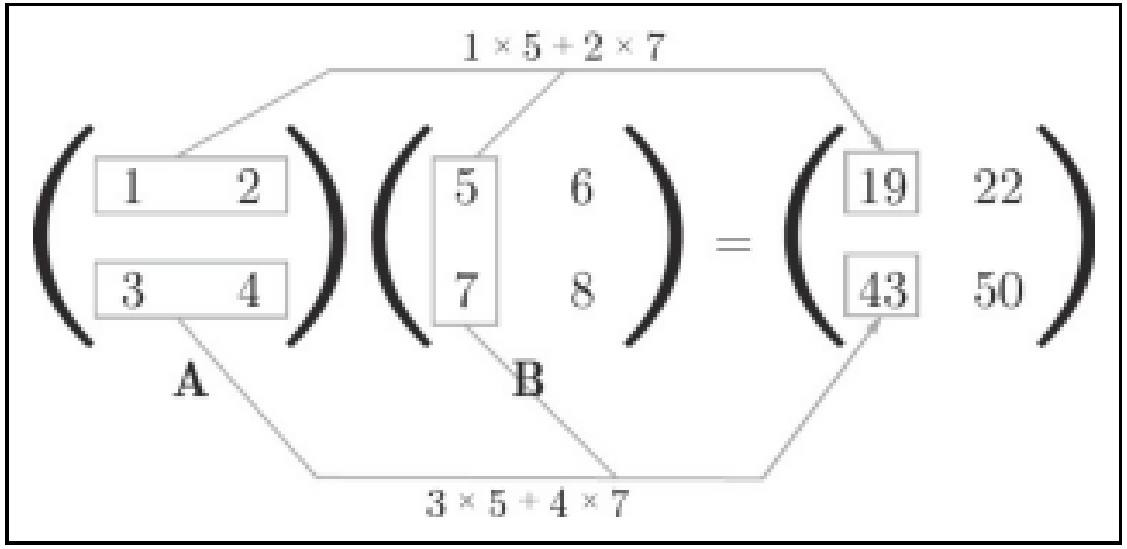
数学の便利な道具 – 行列と行列の内積

なぜ便利かは後ほどわかる

3 x 2の行列の例



行列の内積の例



行列計算に慣れ親しむ

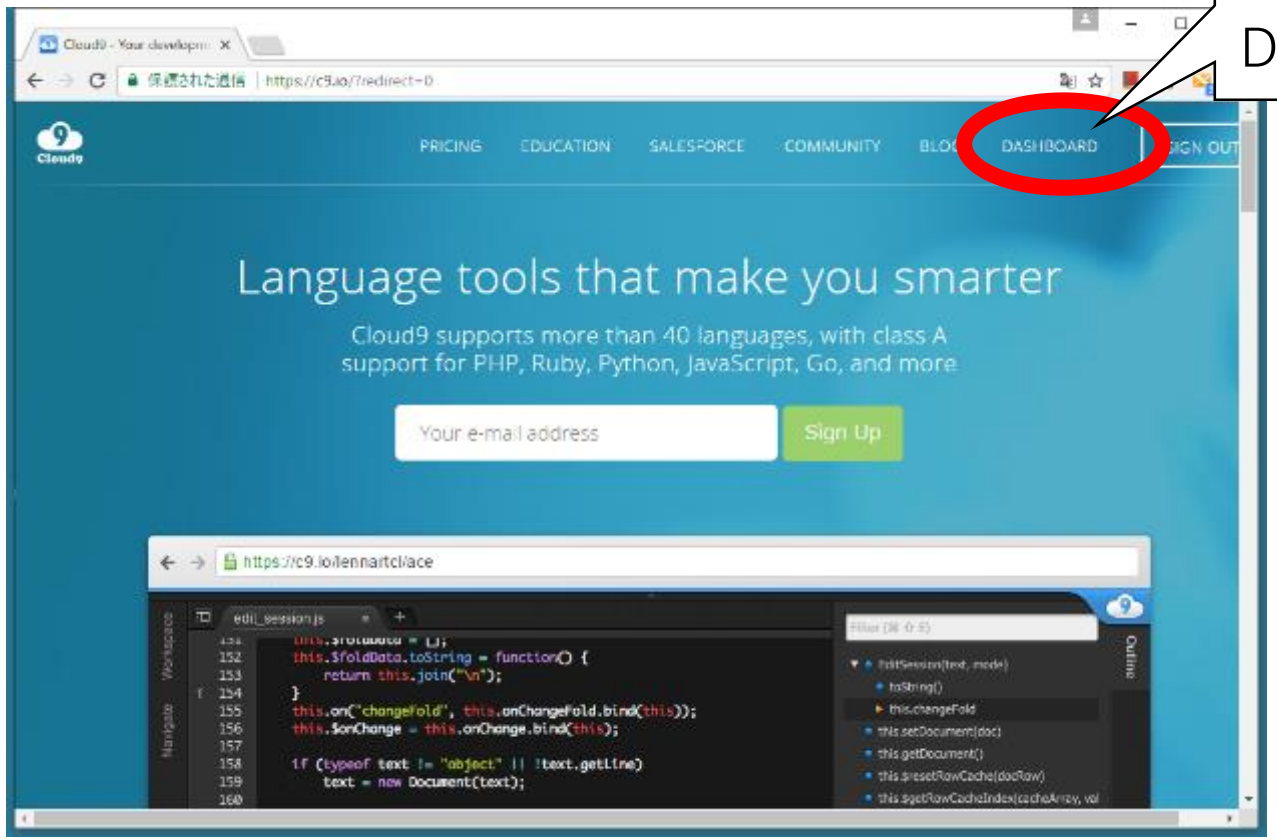
$$\begin{pmatrix} 1 & 3 \\ 5 & 7 \\ 9 & 11 \end{pmatrix} \cdot \begin{pmatrix} 11 \\ 22 \end{pmatrix} = \begin{pmatrix} 77 \\ 209 \\ 341 \end{pmatrix}$$

$$(11\ 22) \cdot \begin{pmatrix} 1 & 3 \\ 5 & 7 \\ 9 & 11 \end{pmatrix} = \boxed{?}$$

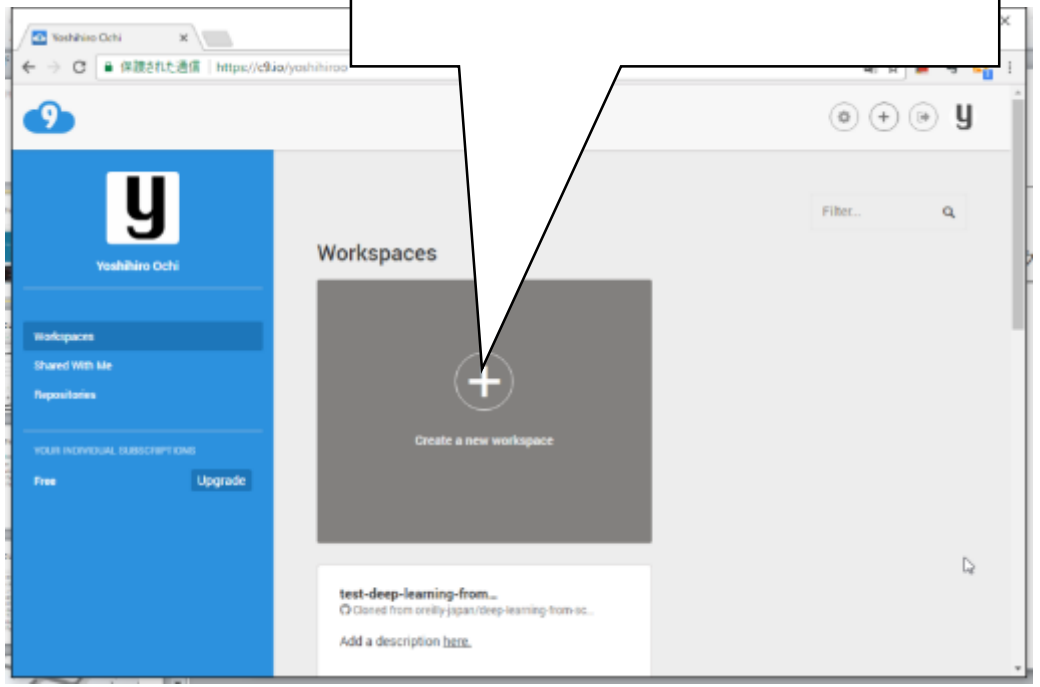
ちょっとその前に

Cloud9によるPython開発環境（１）

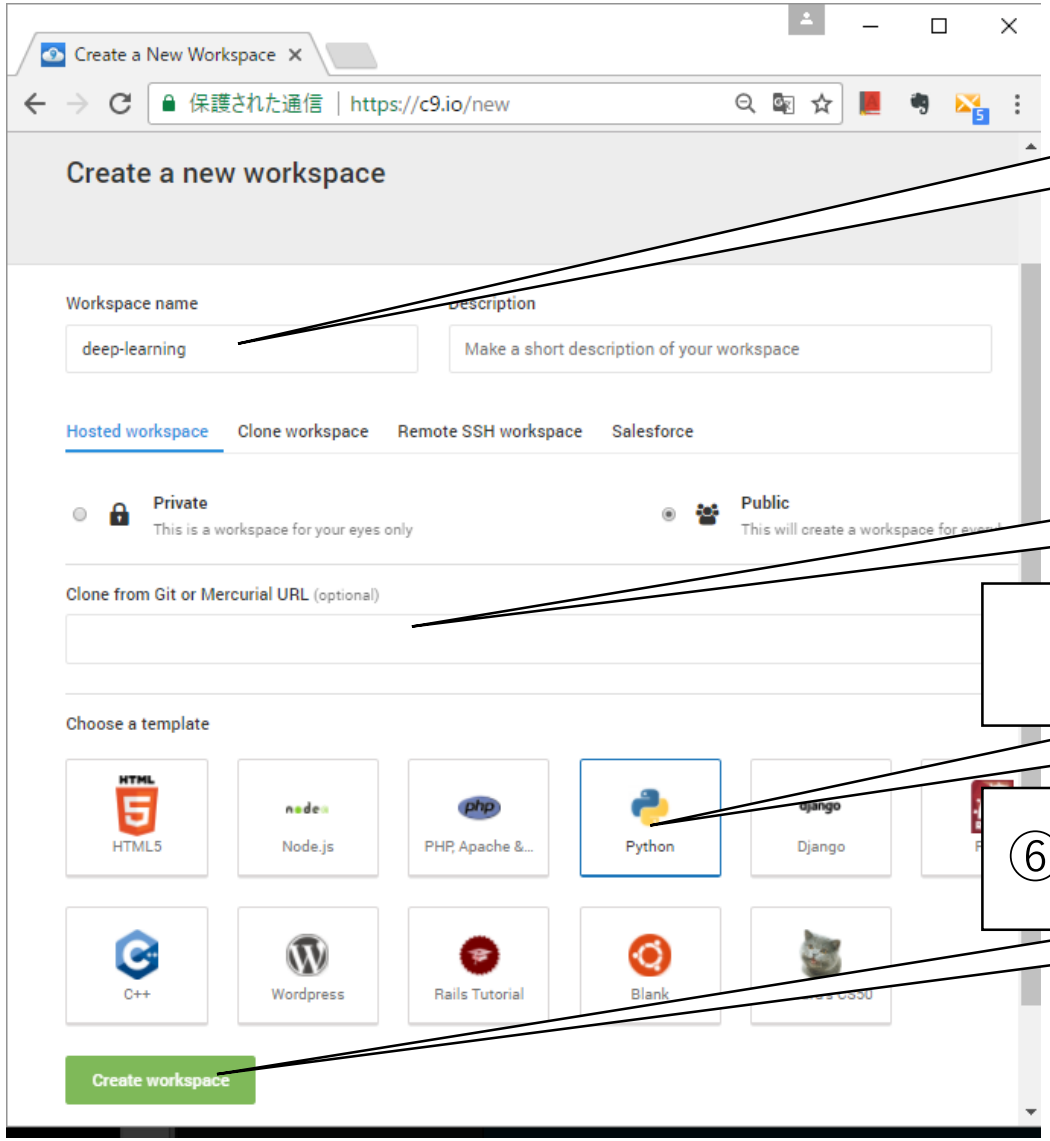
①Cloud9ログイン後、
トップページから
DASHBOARDをクリック



②「＋」をクリックする



Cloud9によるPython開発環境（２）



③「Workspace name」は
適当な名前を入力

④「Clone from Git or Mercurial URL (optional)」
空白のままでOKです！

⑤「Choose a template」は
Pythonをクリックして選択

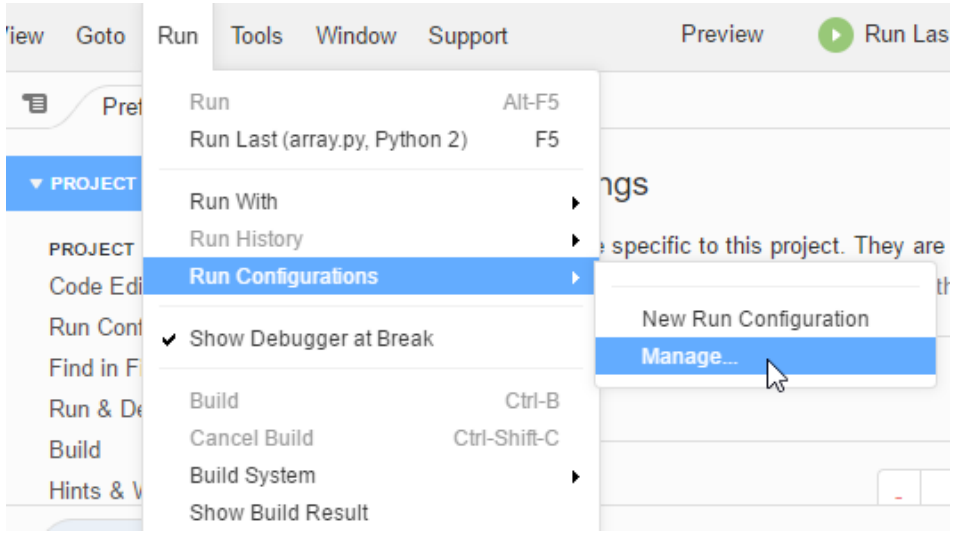
⑥Create workspaceをクリック

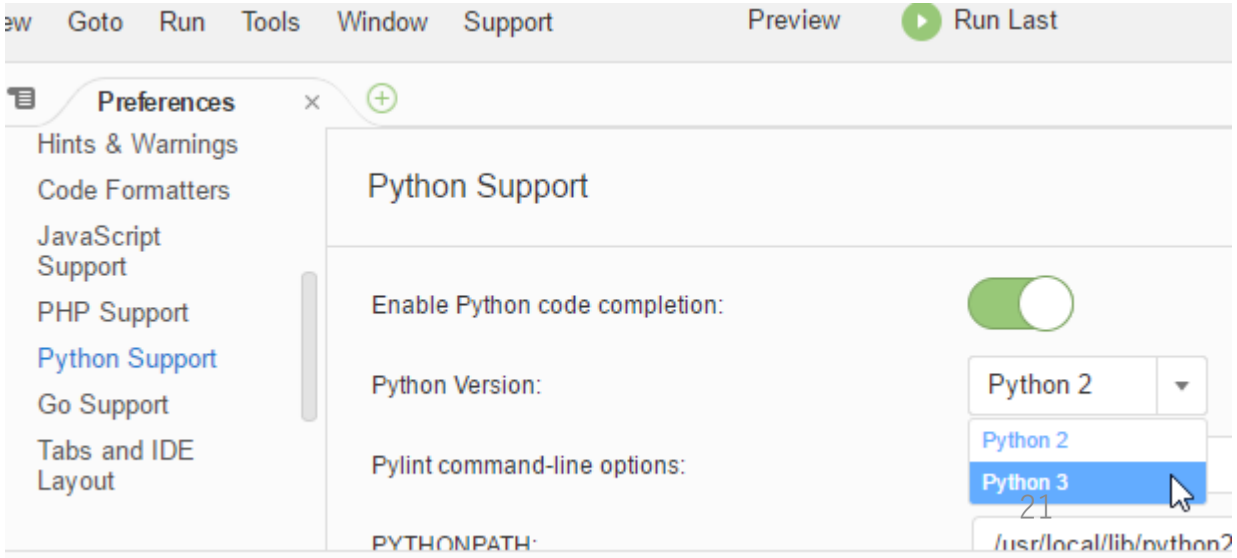
こんな画面になったら完了！→

Cloud9によるPython開発環境（３）

⑦Pythonのバージョンを3に変更

- Run -> Run Configurations -> Manage
- Preference
  - Python Support : Python Version:
  - Python 3へ変更
- Preferenceタブを閉じる

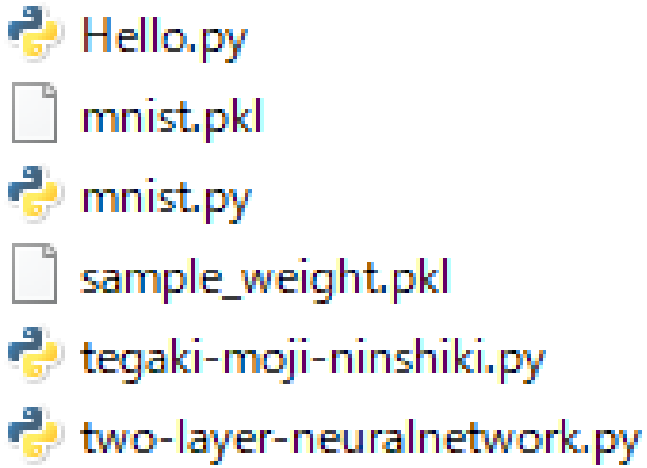




Cloud9によるPython開発環境（４）

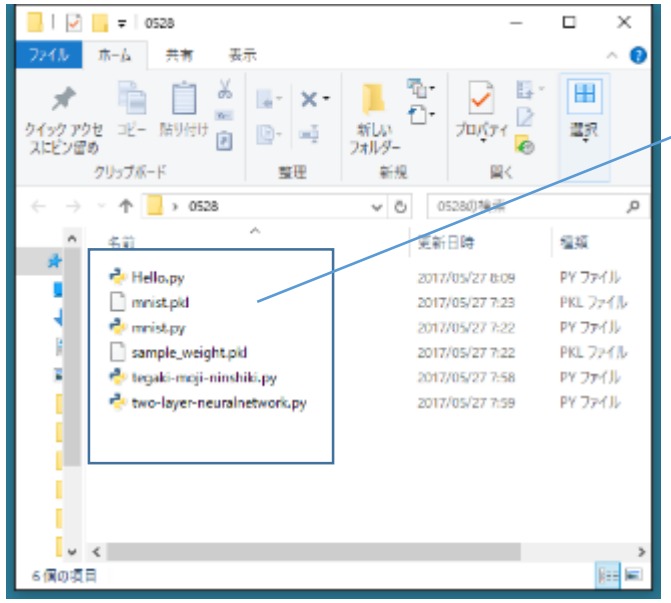
⑧Githubから必要なファイルをダウンロード

- 下記URLに行く
  https://github.com/yoshihiroo/programming-workshop/tree/master/deep-learning
- 0528.zip をダウンロードし、解凍すると下記６つのファイルができる

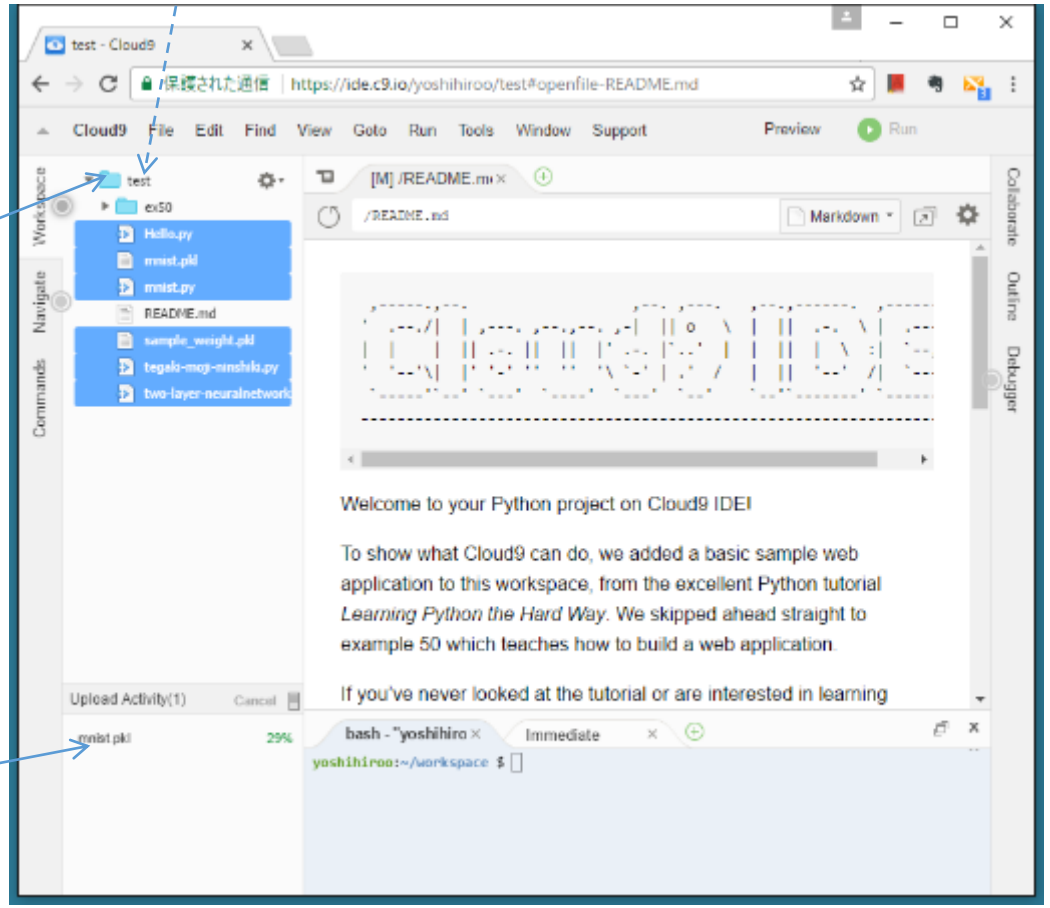


Cloud9によるPython開発環境（５）

- ⑨Cloud9にファイルをアップロード
  - 解凍した６つのファイルをドラッグ＆ドロップでCloud9上のフォルダにコピー



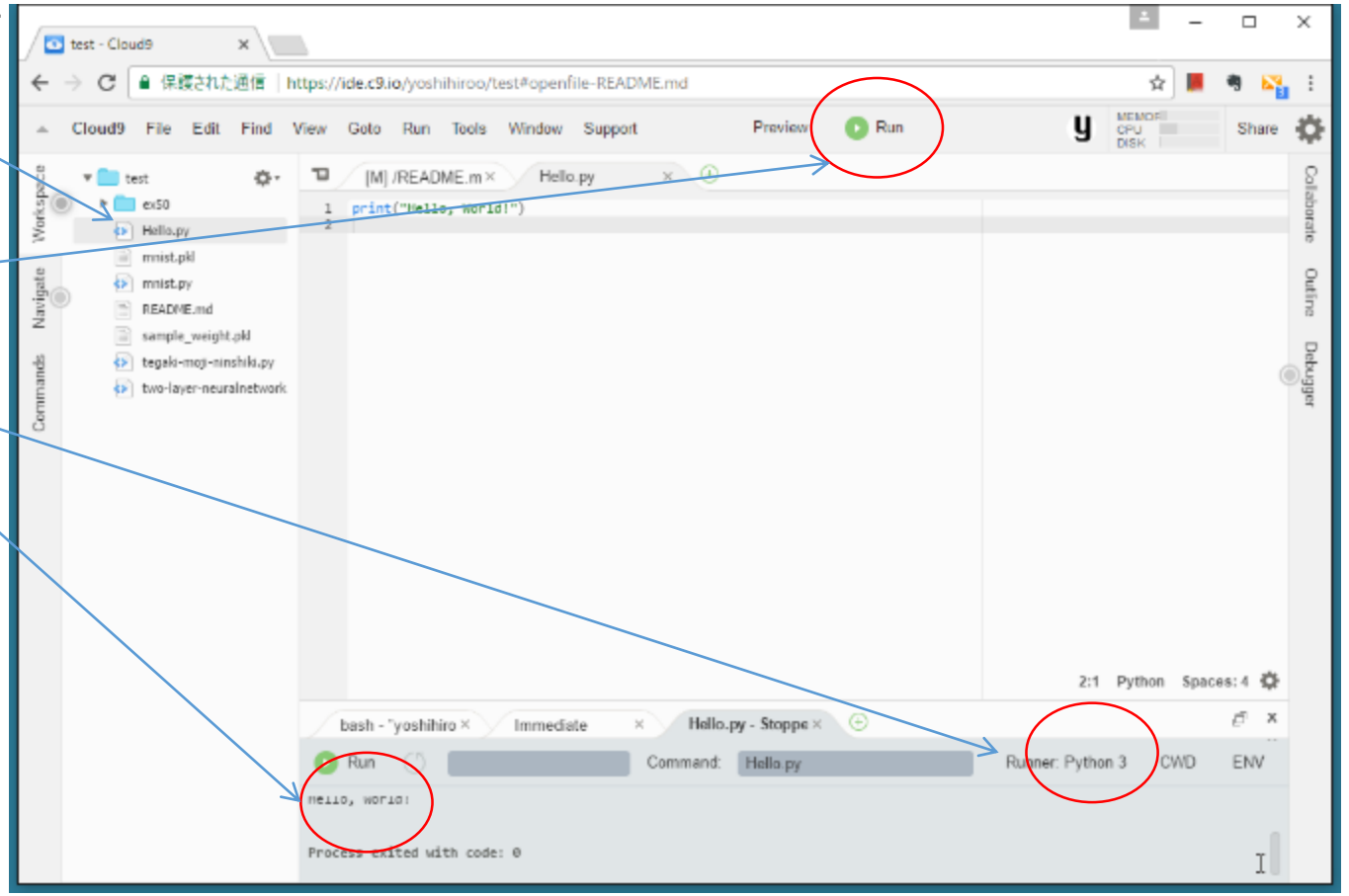
ワークスペースの作成時の名前がディレクトリ名になっています
この画面の場合「test」。



mnist.pklは重たいファイルなので時間かかります。
ここに進行状況が表示されます。

Cloud9によるPython開発環境（６）

- ⑩サンプルファイルの実行
  - Hello.pyをダブルクリック
  - ソースコードが表示される
  - 右上のRUNをクリック
  - 「Python 3」を確認
  - Hello, World!の表示を確認




delete
Break

蛇足

いろいろな呼び方

- 行列　＝　マトリックス　＝　二次元配列　＝　二次元行列

  数学っぽい　英語　プログラムのときよく使う　次元を明示的に

- ベクトル　＝　配列　＝　リスト

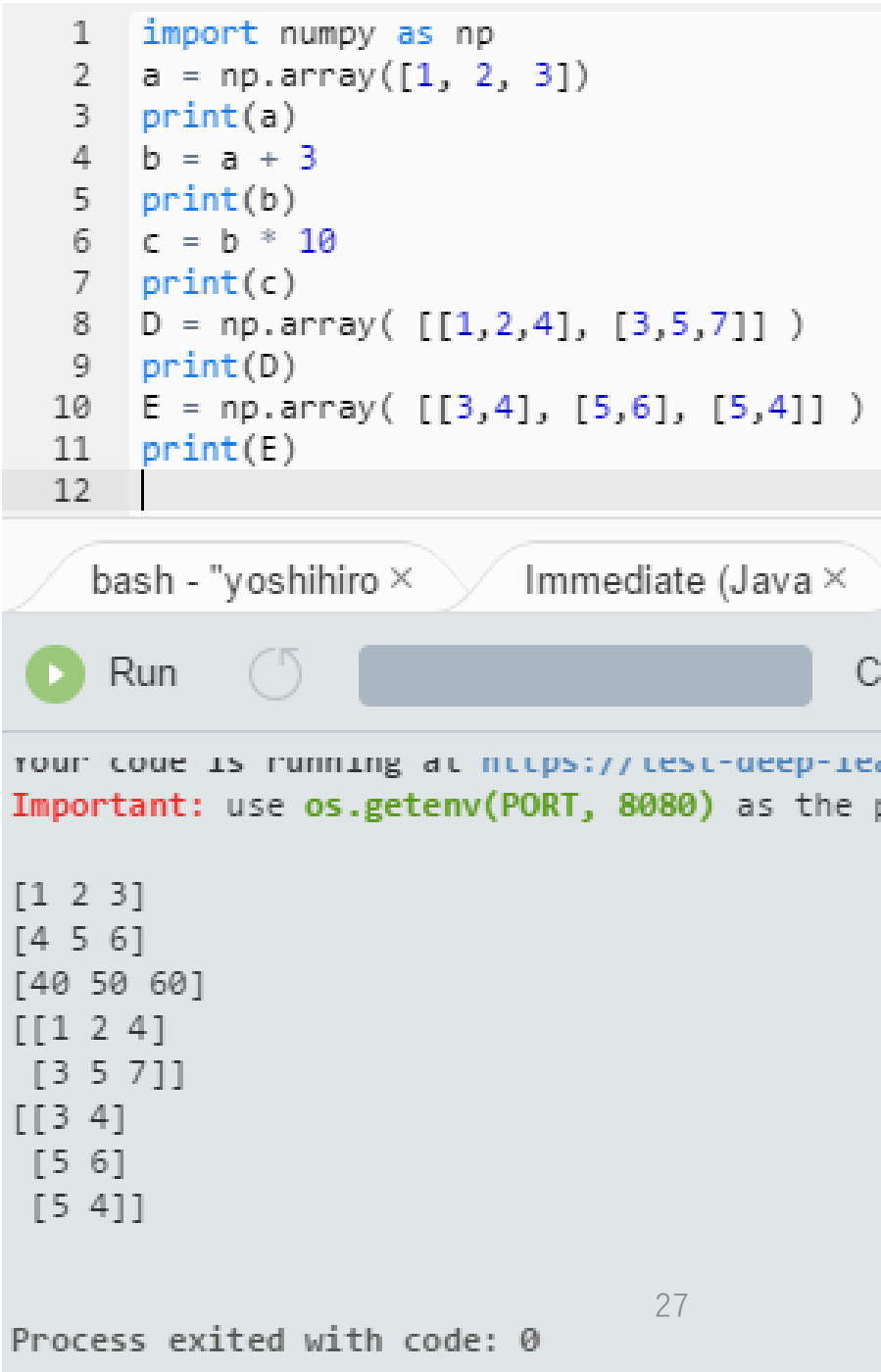
  数学っぽい　プログラム‥　特にPython

表記方法

- 行列は大文字 A, B, C..
- ベクトルは小文字 a, b, c..

Python上で行列計算を行う

```
import numpy as np
a = np.array([1, 2, 3])
print(a)
b = a + 3
print(b)
c = b * 10
print(c)
D = np.array( [[1,2,4], [3,5,7]] )
print(D)
E = np.array( [[3,4], [5,6], [5,4]] )
print(E)
```



先ほどの例をNumPyを使って計算してみましょう

$$\begin{pmatrix} 1 & 3 \\ 5 & 7 \\ 9 & 11 \end{pmatrix} \cdot \begin{pmatrix} 11 \\ 22 \end{pmatrix} = \begin{pmatrix} 77 \\ 209 \\ 341 \end{pmatrix}$$

先ほどの例をNumPyを使って計算してみましょう

$$\begin{pmatrix} 1 & 3 \\ 5 & 7 \\ 9 & 11 \end{pmatrix} \cdot \begin{pmatrix} 11 \\ 22 \end{pmatrix} = \begin{pmatrix} 77 \\ 209 \\ 341 \end{pmatrix}$$

X　　y　　z

```
import numpy as np
X = np.array( [[1,3], [5,7], [9,11]] )
y = np.array( [11,22] )
z = np.dot(X, y)
print(z)
```

← np.dotは内積を計算するコマンド

先ほどの例をNumPyを使って計算してみましょう

$$(11\ 22) \cdot \begin{pmatrix} 1 & 3 \\ 5 & 7 \\ 9 & 11 \end{pmatrix} = \boxed{?}$$

y　　X

```
import numpy as np
X = np.array( [[1,3], [5,7], [9,11]] )
y = np.array( [11,22] )
z = np.dot(y, X)
print(z)
```

← Xとyを入れ替えて内積計算したらどうなりますか？

そのほかのNumPyのコマンドをいくつか試してみましょう　青文字箇所を追加

```
import numpy as np
X = np.array( [[1,3], [5,3], [9,11]] )
y = np.array( [11,22] )
z = np.dot(X, y)
print(z)
print(X[0])
print(X[0][1])
print(X.shape)
```

X[3]と見ようとしたらどうなるでしょうか？

同じく、X[0][3]は？

Xのところをyやzに変えたらどうなるでしょうか？

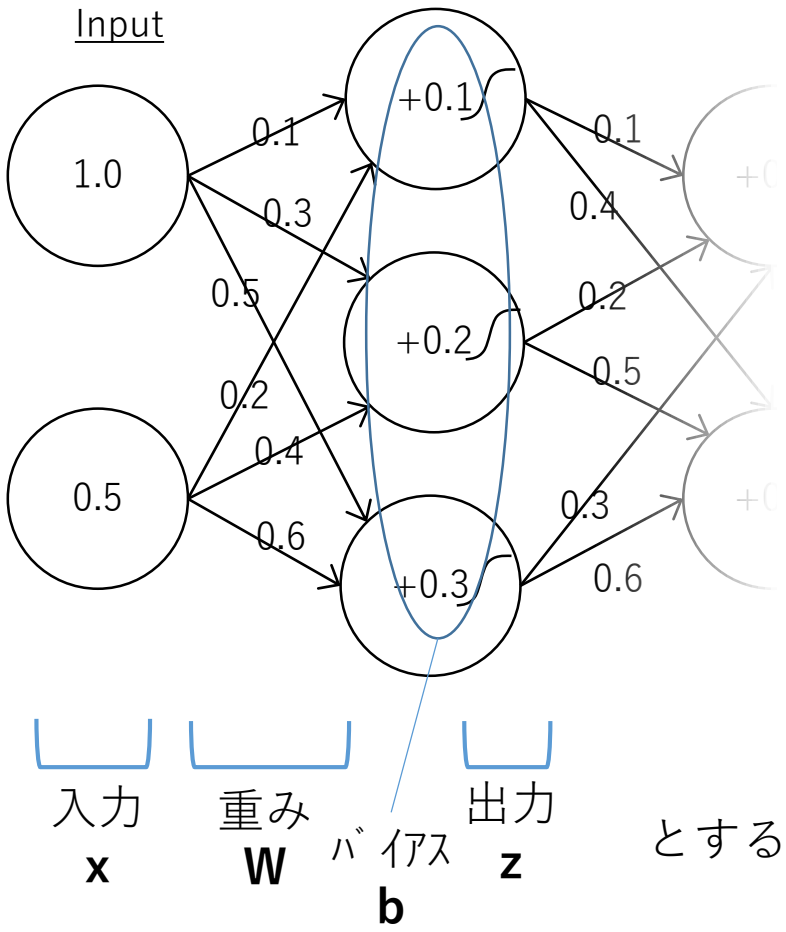
なぜ、行列（内積）なんていう道具を使うのか？

それは、行列を使うとニューラルネットワークをシンプルに記述できるから。

ニューラルネットワークを行列計算で表現してみる

sig() …シグモイド関数

先ほどの模型の前半部分



とすると、

$$\mathbf{z} = \mathbf{sig}(\ \mathbf{W}\ \cdot\ \mathbf{x}\ +\ \mathbf{b}\ )$$

$$= \mathbf{sig}\left(\begin{pmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \end{pmatrix} \cdot \begin{pmatrix} 1.0 \\ 0.5 \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0.2 \\ 0.3 \end{pmatrix} \right)$$

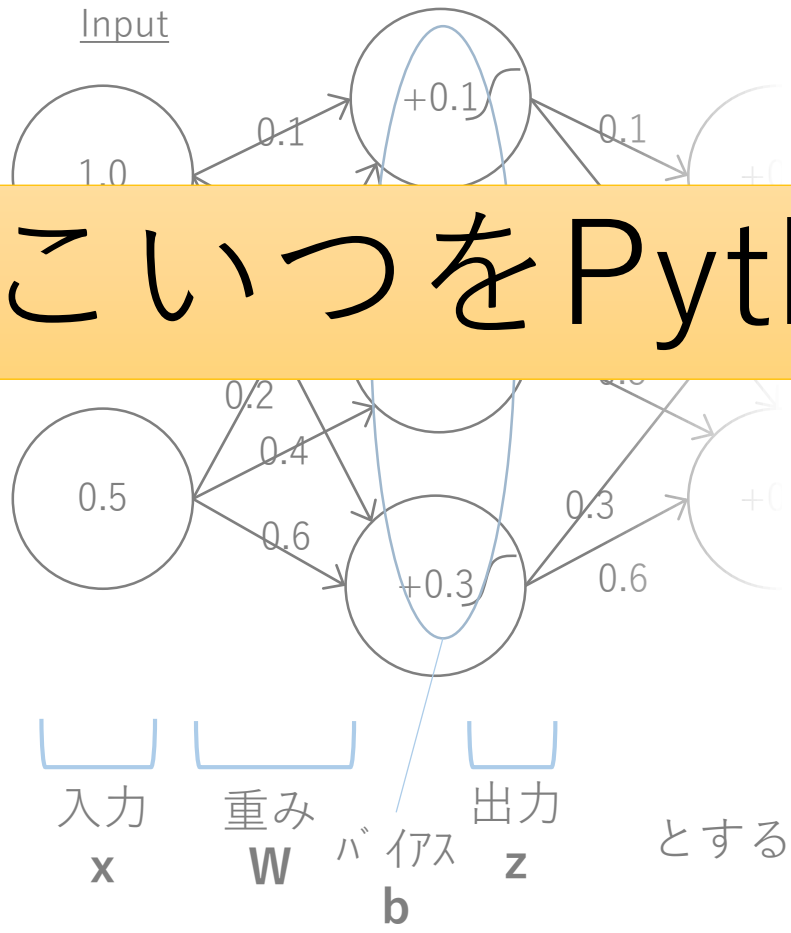
$$= \mathbf{sig}\left(\begin{pmatrix} 0.2 \\ 0.5 \\ 0.8 \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0.2 \\ 0.3 \end{pmatrix} \right)$$

$$= \mathbf{sig}\left(\begin{pmatrix} 0.3 \\ 0.7 \\ 1.1 \end{pmatrix} \right) = \begin{pmatrix} 0.574 \\ 0.668 \\ 0.750 \end{pmatrix}$$

ニューラルネットワークを行列計算で表現してみる

sig() …シグモイド関数

先ほどの模型の前半部分



とすると、

$$\mathbf{z} = \mathbf{sig}(\ \mathbf{W}\ \cdot\ \mathbf{x}\ +\ \mathbf{b}\ )$$

$$= \mathbf{sig}\left(\begin{pmatrix}0.1 & 0.2\\ \cdots & \cdots\end{pmatrix}\begin{pmatrix}1.0\\ 0.5\end{pmatrix} + \begin{pmatrix}0.1\\ \cdots\end{pmatrix}\right)$$

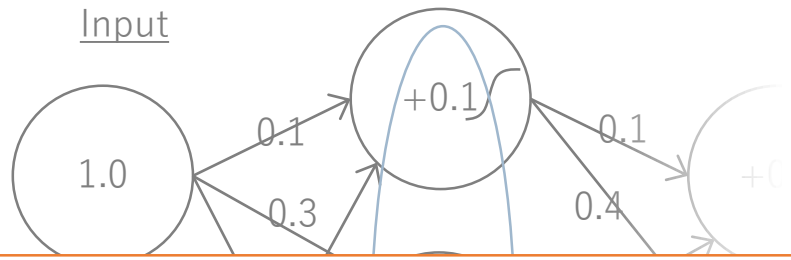
こいつをPythonで実装してみよう！

$$= \mathbf{sig}\left(\begin{pmatrix}0.2\\ 0.5\\ 0.8\end{pmatrix} + \begin{pmatrix}0.1\\ 0.2\\ 0.3\end{pmatrix}\right)$$

$$= \mathbf{sig}\left(\begin{pmatrix}0.3\\ 0.7\\ 1.1\end{pmatrix}\right) = \begin{pmatrix}0.574\\ 0.668\\ 0.750\end{pmatrix}$$

ニューラルネットワークを行列計算で表現してみる

先ほどの模型の前半部分



sig() …シグモイド関数

$$z = sig(\ W \cdot x + b\ )$$

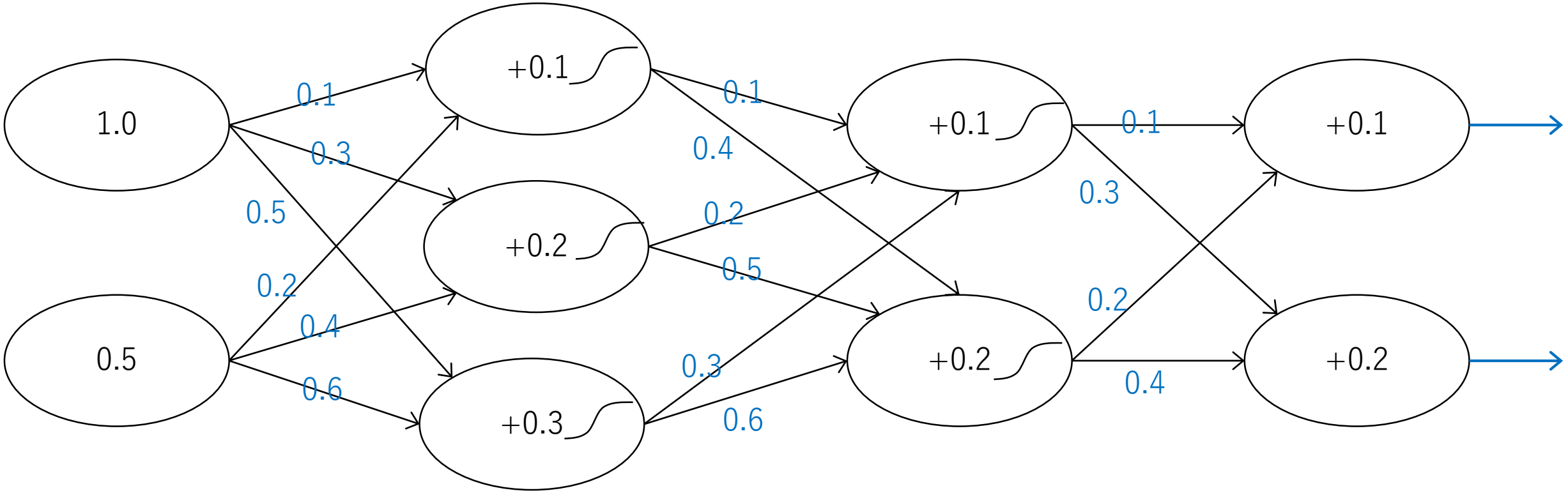
$$= sig\left(\begin{pmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \end{pmatrix} \cdot \begin{pmatrix} 1.0 \\ 0.5 \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0.2 \\ 0.3 \end{pmatrix} \right)$$

$$= sig\left(\begin{pmatrix} 0.2 \\ 0.5 \\ 0.8 \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0.2 \\ 0.3 \end{pmatrix} \right)$$

$$= sig\left(\begin{pmatrix} 0.3 \\ 0.7 \\ 1.1 \end{pmatrix} \right) = \begin{pmatrix} 0.574 \\ 0.668 \\ 0.750 \end{pmatrix}$$

```
import numpy as np

def sig(x):
    return 1 / (1 + np.exp(-x))

W = np.array([[0.1,0.2],[0.3,0.4],[0.5,0.6]])
x = np.array([1.0,0.5])
b = np.array([0.1,0.2,0.3])

z = sig( np.dot(W,x) + b )
print(z)
```

先ほどのニューラルネットの例



$$x = \begin{pmatrix} 1.0 \\ 0.5 \end{pmatrix} \quad W1 = \begin{pmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \end{pmatrix} \quad b1 = \begin{pmatrix} 0.1 \\ 0.2 \\ 0.3 \end{pmatrix} \quad z1 = \begin{pmatrix} ? \\ ? \\ ? \end{pmatrix} \quad W2 = \begin{pmatrix} 0.1 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.6 \end{pmatrix} \quad b2 = \begin{pmatrix} 0.1 \\ 0.2 \end{pmatrix} \quad z2 = \begin{pmatrix} ? \\ ? \end{pmatrix} \quad W3 = \begin{pmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{pmatrix} \quad b3 = \begin{pmatrix} 0.1 \\ 0.2 \end{pmatrix} \quad z3 = \begin{pmatrix} ? \\ ? \end{pmatrix}$$

z1=sig(W1・x+b1)

z2=sig(W2・z1+b2)

z3=W3・z2+b3

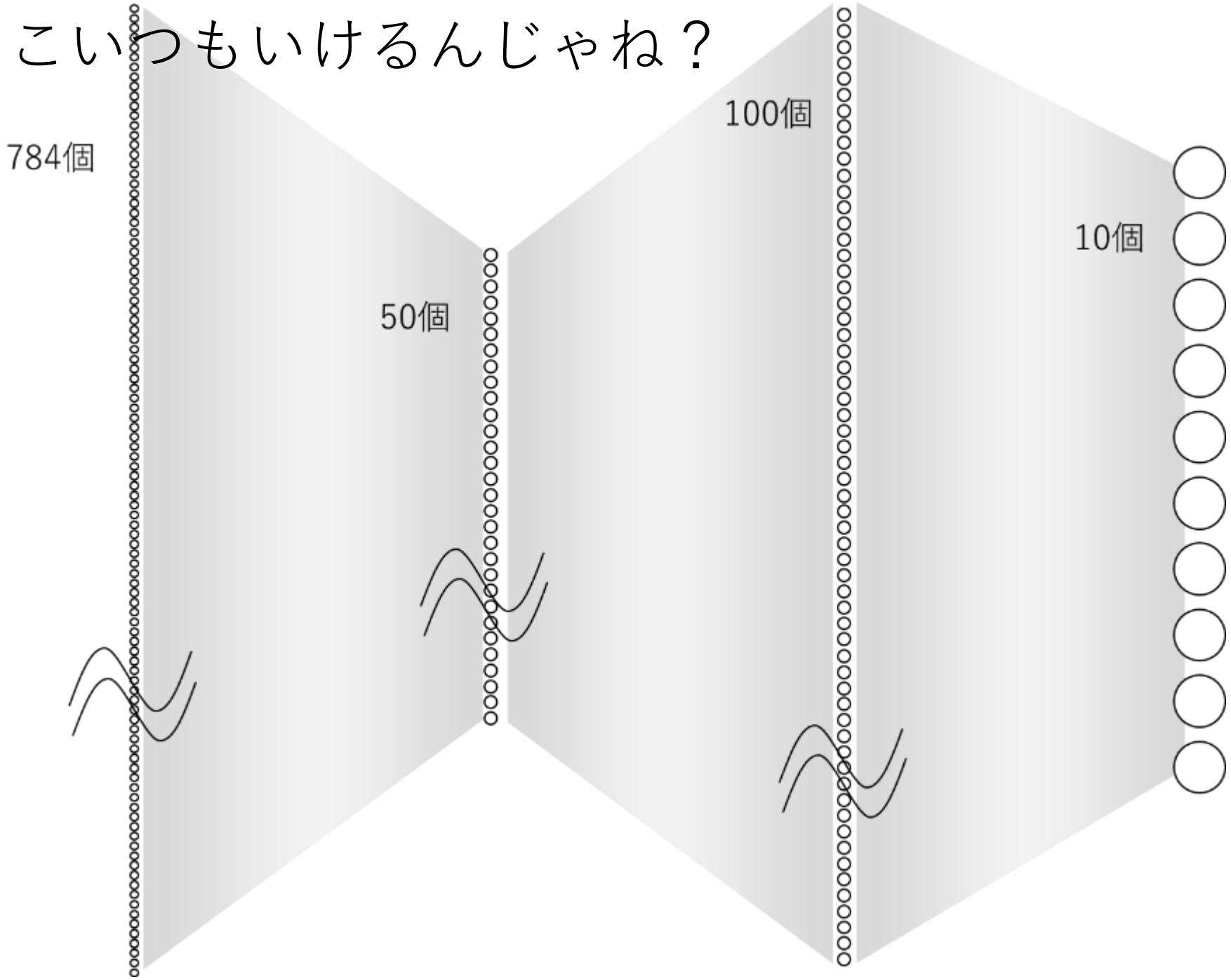
sig() …シグモイド関数

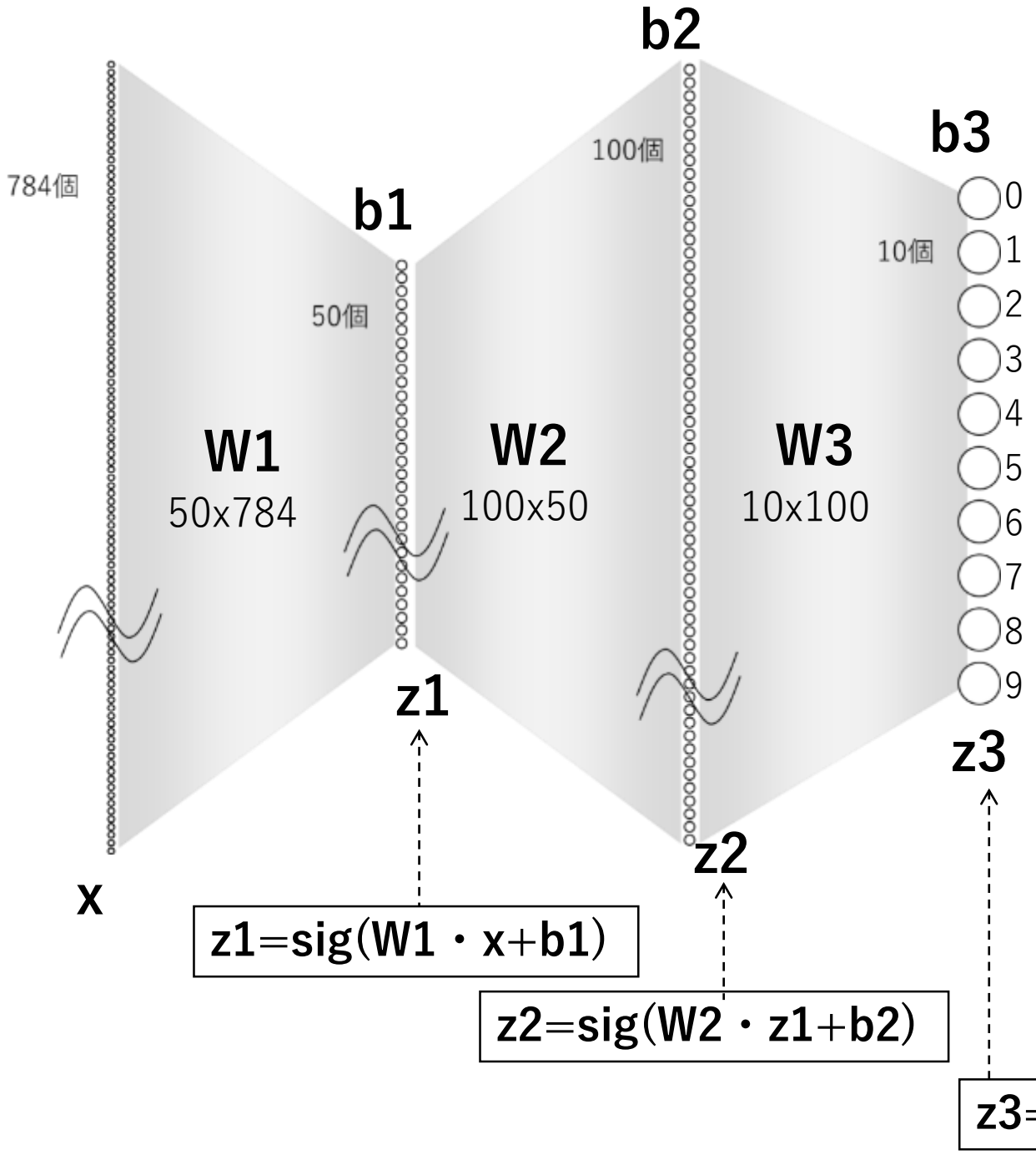
ソースコード
two-layer-neuralnetwork.py

```
import numpy as np

def sig(x):
    return 1 / (1 + np.exp(-x))

W1 = np.array([[0.1,0.2], [0.3,0.4], [0.5,0.6]])
x = np.array([1.0, 0.5])
b1 = np.array([0.1, 0.2, 0.3])
z1 = sig( np.dot(W1, x) + b1 )

W2 = np.array([[0.1,0.2,0.3], [0.4,0.5,0.6]])
b2 = np.array([0.1, 0.2])
z2 = sig( np.dot(W2, z1) + b2 )

W3 = np.array([[0.1,0.2], [0.3,0.4]])
b3 = np.array([0.1, 0.2])
z3 = np.dot(W3, z2) + b3

print(z3)
```

コンピューターにニューラルネットワークの演算をさせる方法を手に入れた！

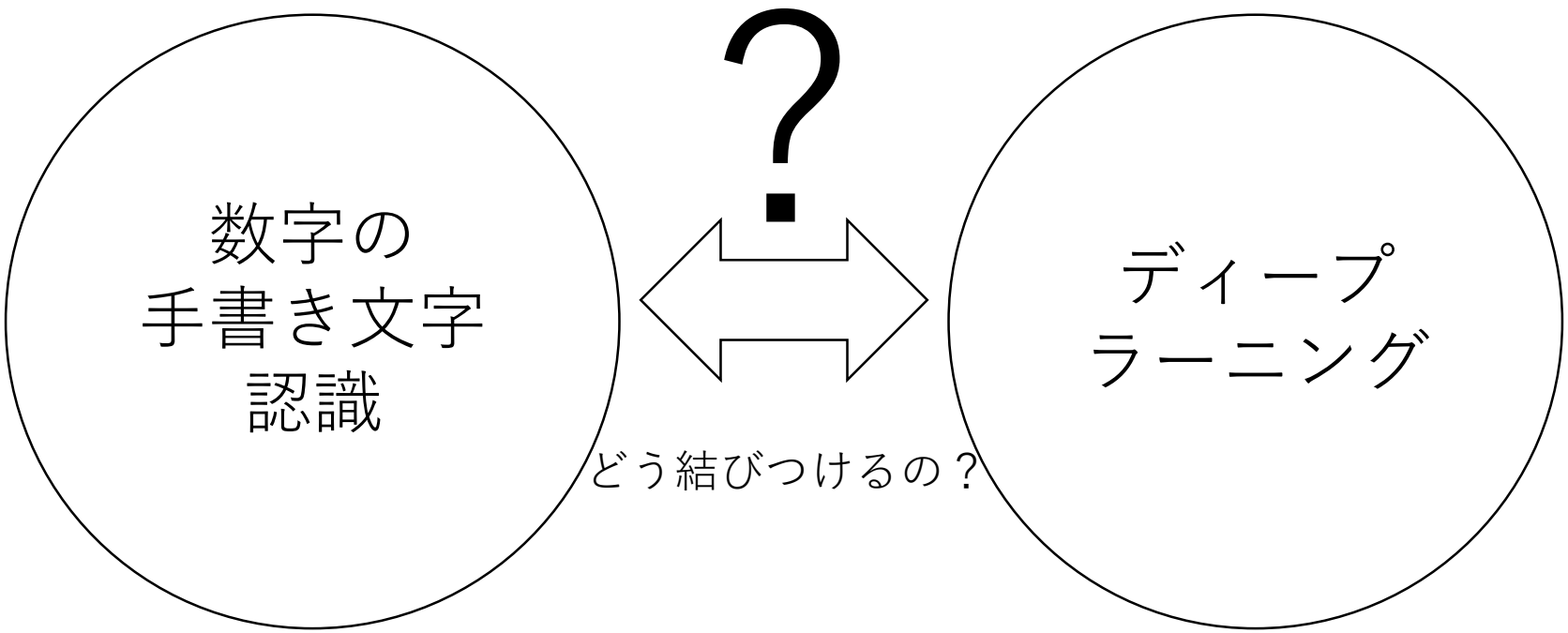
だったらこいつもいけるんじゃね？





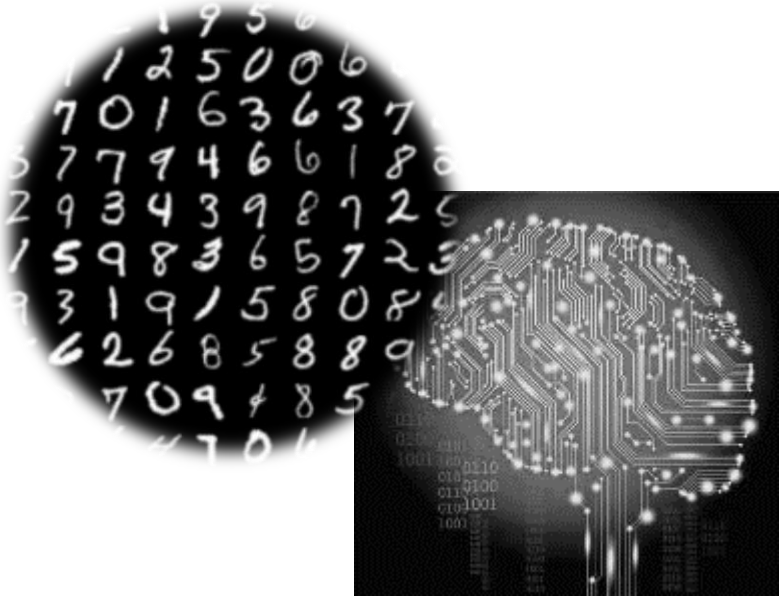
ここの理解、
今日の踏ん張りどころです

数字の手書き文字認識をさせてみる



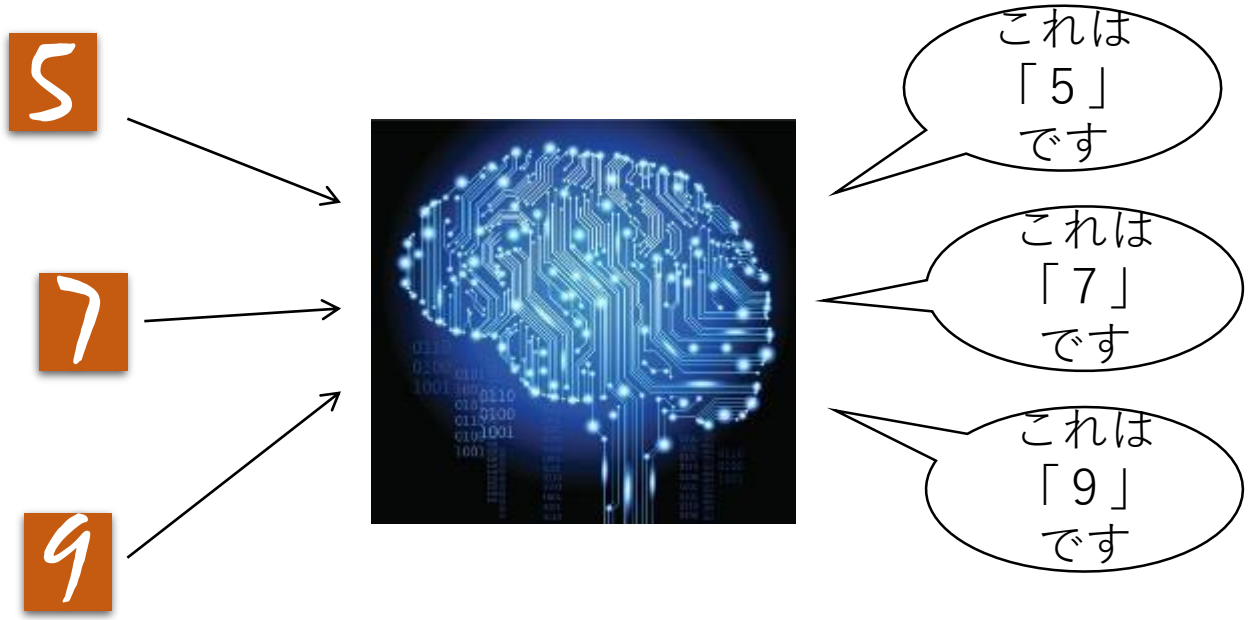
人工知能が文字認識するとはどういうことか？

訓練用データを使って学習させる

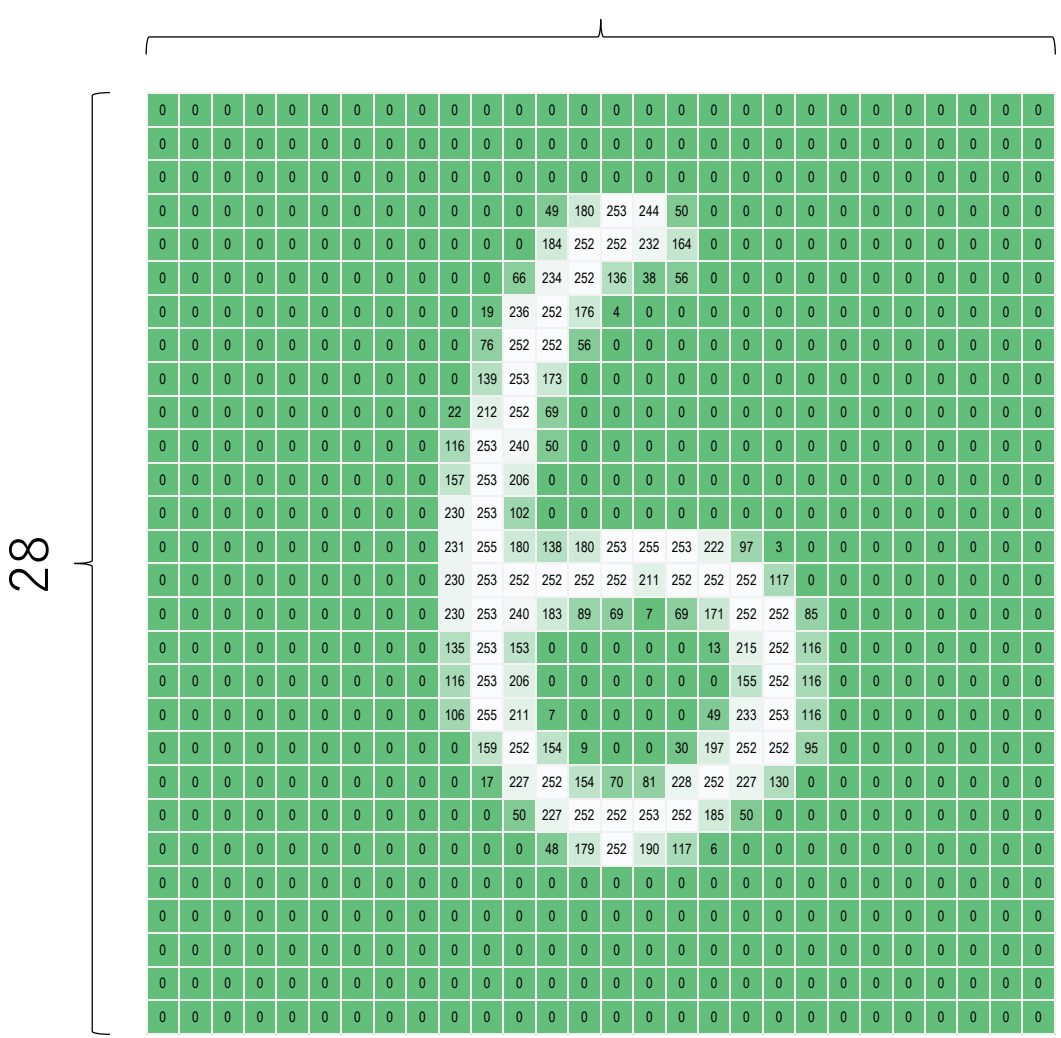


汎用データを使って判別させる

（訓練用データとは別の）



コンピューターにわかってもらうために、手書きの文字を数値化する

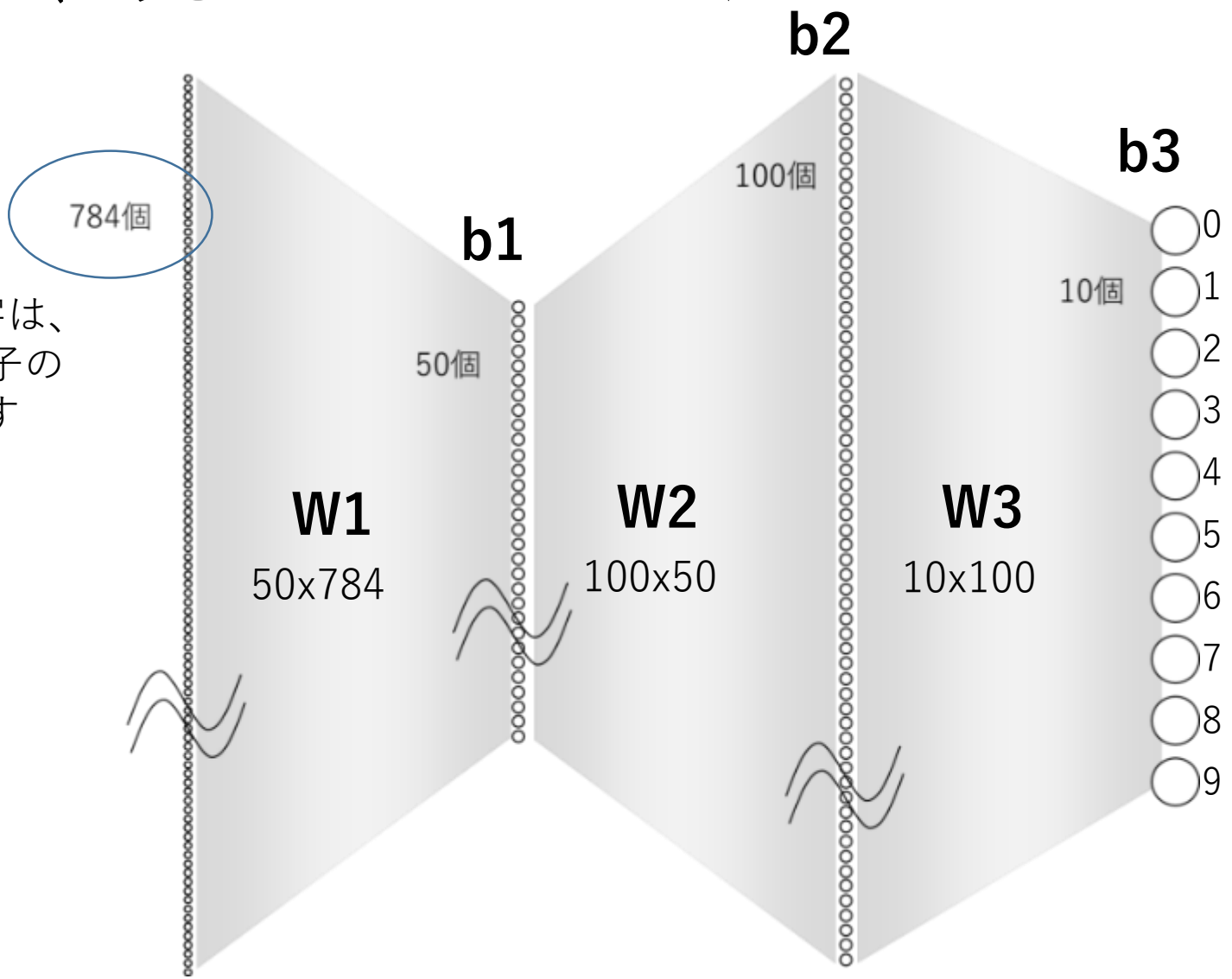


28x28=784個の格子(ピクセル)ごとに、0~255の値で明るさを表現

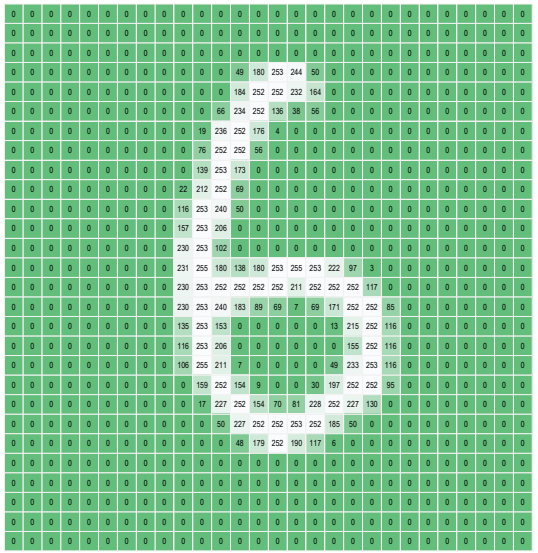
784個の数字の配列で、一つの手書き文字を表現できる

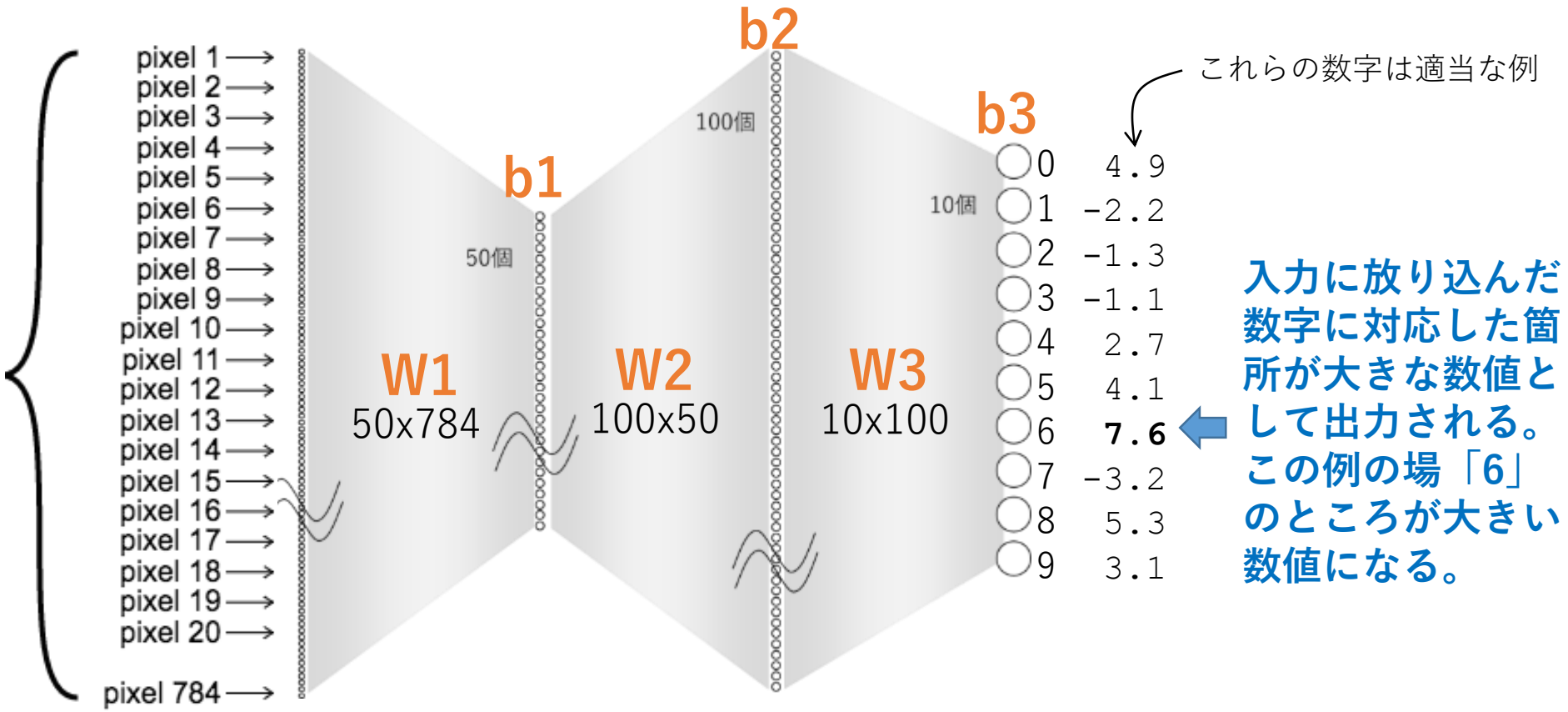
そして、先ほどのこれ↓にかませる

この784という数字は、
28x28=784個の格子の
数に対応しています



結果を先に言っておきますと、ディープラーニングによる文字認識とは、こういうことができます

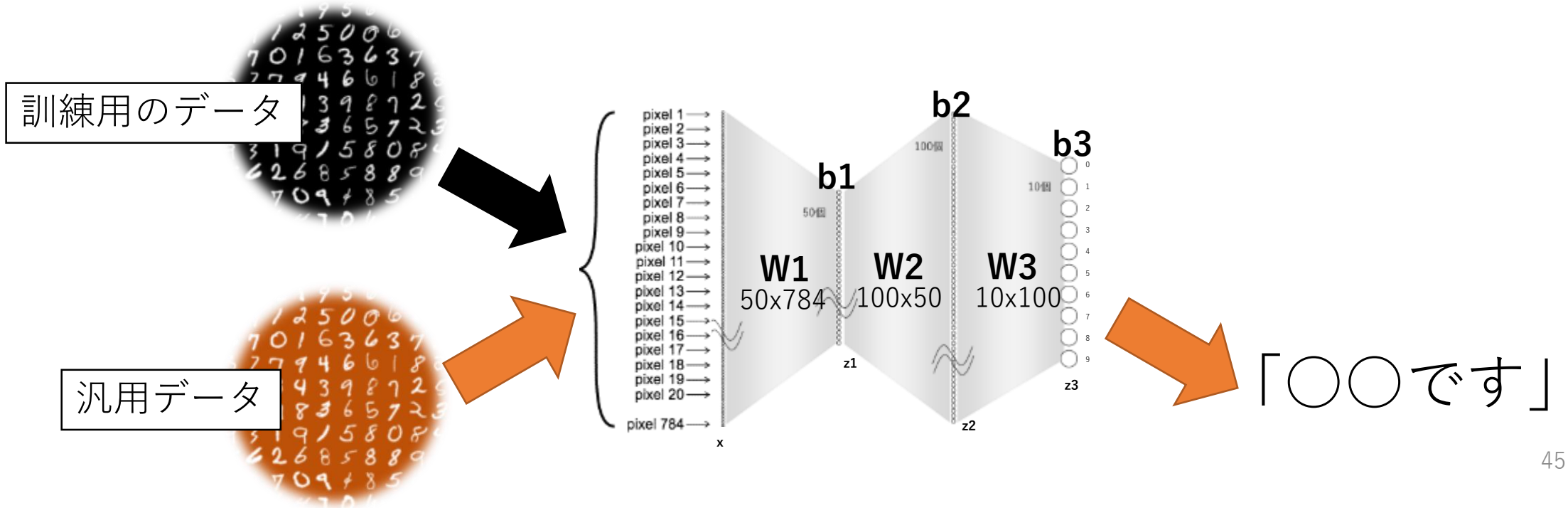




W1, W2, W3, b1, b2, b3 (全部で45,350個の数字)それぞれに
“絶妙な組み合わせの数字” を置いたら、これができてしまう！
これが、ディープな(多層構造の)ニューラルネットワークの不思議な力！

ディープラーニングにおける学習とは

多層構造のニューラルネットワークに対して、大量の訓練用データを食わせながら、**パラメータ(W1, W2, W3, b1, b2, b3)の絶妙な組み合わせを探し出す**こと。
その絶妙なパラメータとは、訓練用データとは別の汎用的なデータを入力したときに、なるべく高い精度で正解を判別できるもの。



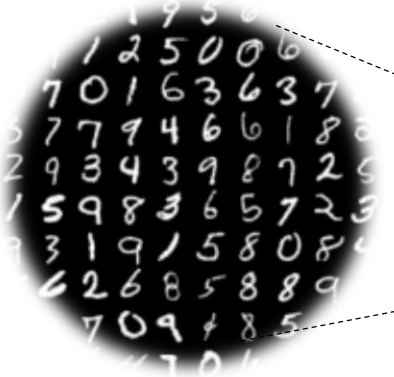
じゃ、学習って具体的になにやるの？

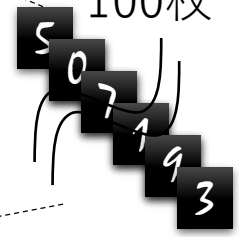
- 今日は時間の都合で学習のプログラミング実装は割愛m(__)m
- イメージでいうと、コンピューターの計算力を頼りに、ちょっとずつパラメータW1, W2, W3, b1, b2, b3 (全部で45,350個の数字)を変えながら、何度も何度も計算を繰り返して絶妙なパラメータの組み合わせを探すプロセス
- 学習(コンピューターの数値計算)を効率よくさせるための手法はそれ自体が奥深い研究テーマであり、誤差逆伝搬法(バックプロパゲーション)、SGD、Momentum、AdaGrad、Adamなど、専門用語がバンバン出てくる領域。今日はそのあたりの深入りはやめときます

当たってなさ具合を数値で表す

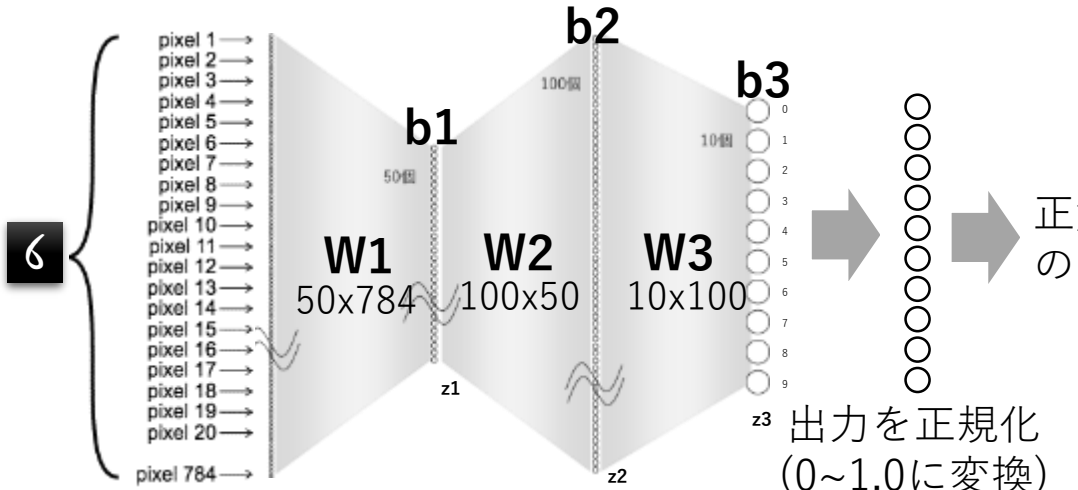
- イメージでいうと、コンピューターの計算力を頼りに、ちょっとずつパラメータW1, W2, W3, b1, b2, b3 (全部で45,350個の数字)を変えながら、何度も何度も計算を繰り返して絶妙なパラメータの組み合わせを探すプロセス

訓練用のデータ





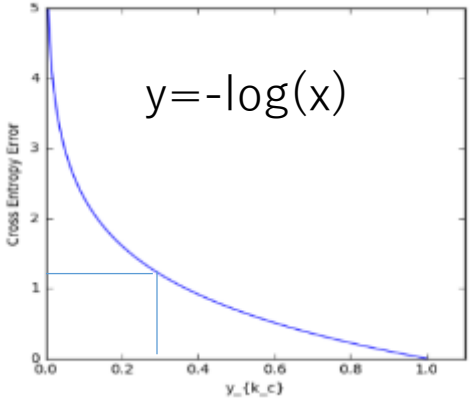
MNIST計6万枚のデータ群から、
ランダムに100枚を選び出す。



一枚ずつニューラルネットに放り込んで計算

出力を正規化
(0~1.0に変換)

正解箇所の値
の-logを計算

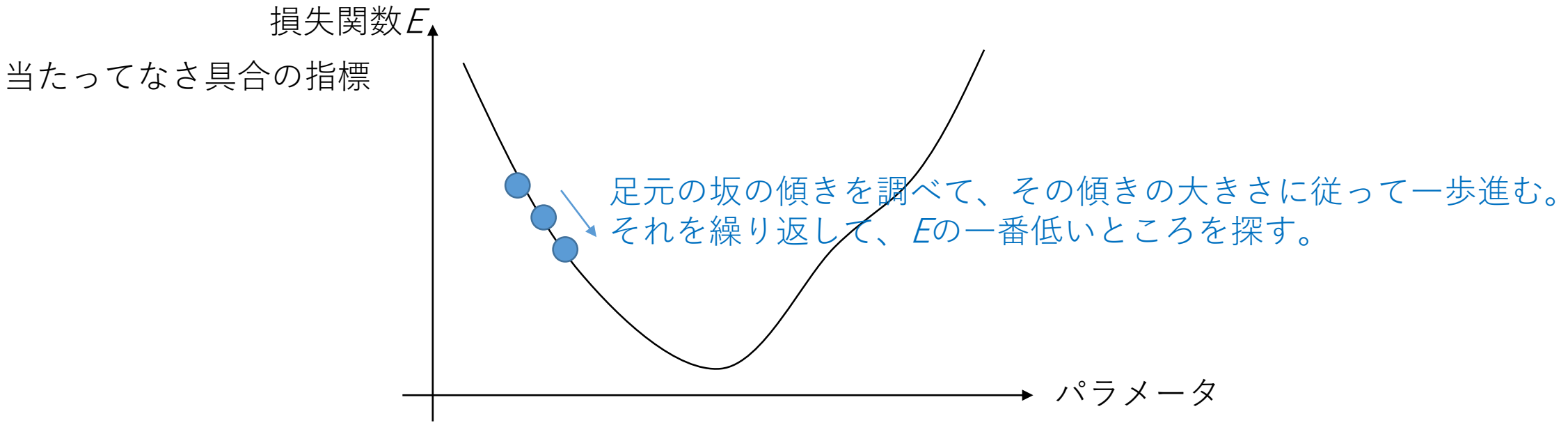


100枚分計算して平均を求める。
これが損失関数*E*の値となる。
要は当たってなさ具合の指標。

パラメータを少しずつ変えて繰り返し。*E*が十分に小さくなるパラメータを探す。

当たってなさ具合の数値が一番小さくなるところを探す

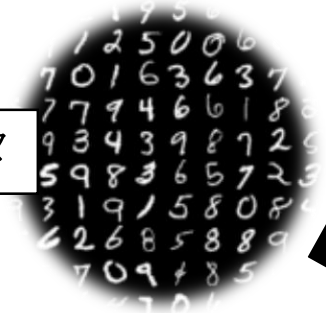
- イメージでいうと、コンピューターの計算力を頼りに、ちょっとずつパラメータW1, W2, W3, b1, b2, b3 (全部で45,350個の数字)を変えながら、何度も何度も計算を繰り返して絶妙なパラメータの組み合わせを探すプロセス

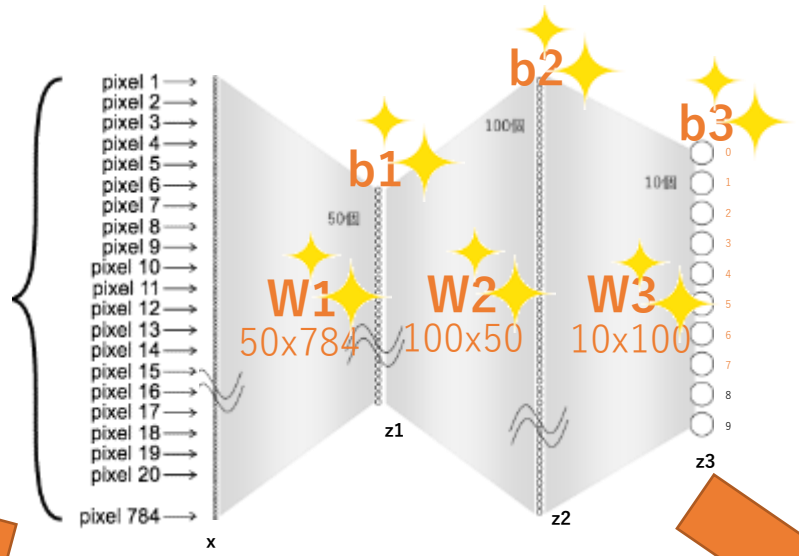


W1, W2, W3, b1, b2, b3の45,350個それぞれに対して

今日は学習の実装は割愛なので、まるで三分間クッキングのように、そこそこ絶妙なパラメータを手に入れたとする☺

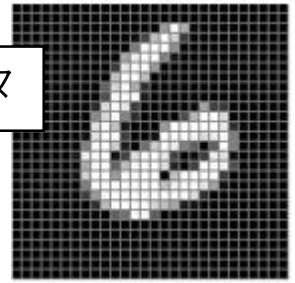
訓練用のデータ





例えば「6」

テスト用のデータ

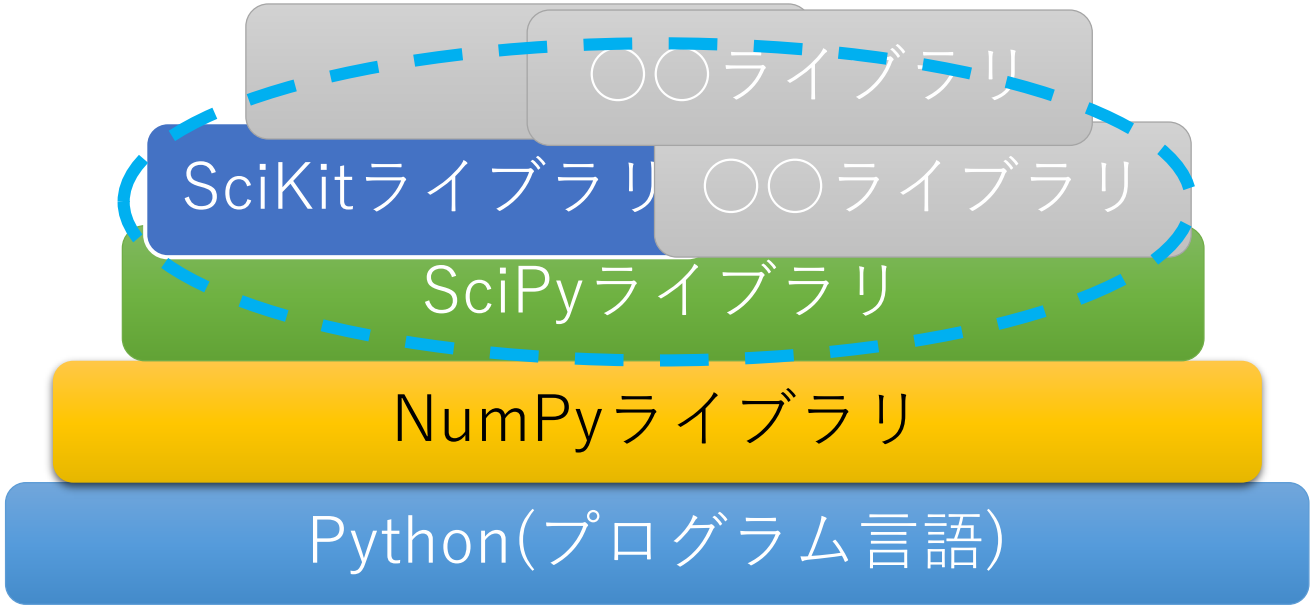


「6」と認識できるか？
実際にPythonで動かして試してみよう！

ソースコード
`tegaki-moji-ninshiki.py`

ということで、一通りのおさらい

（ちょいと脱線）
ディープラーニングでなぜPythonなのか？



Pythonにはディープラーニングのみならず、科学技術分野に有用なライブラリが豊富。
その土台を支えるのが、行列に関する演算を高速に・手軽に行うためのNumPyライブラリ。

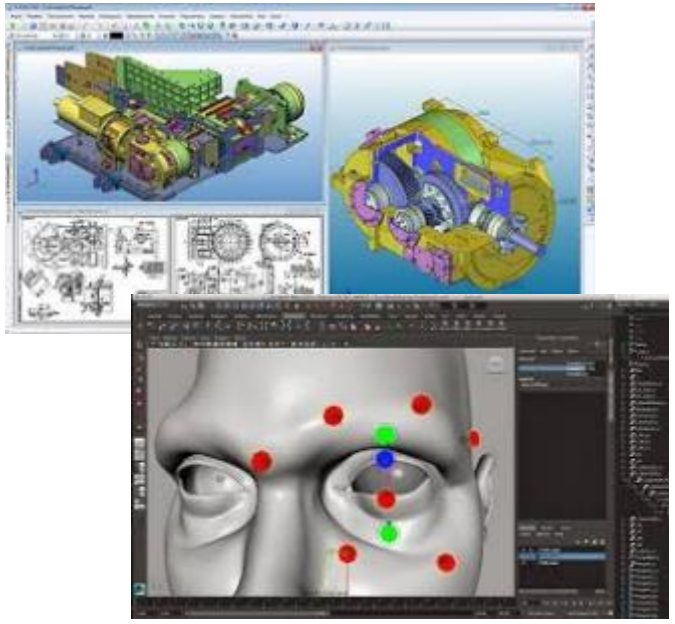
便利なコマンド・関数をパッケージにして、広く他の人にも使えるようにしたもの

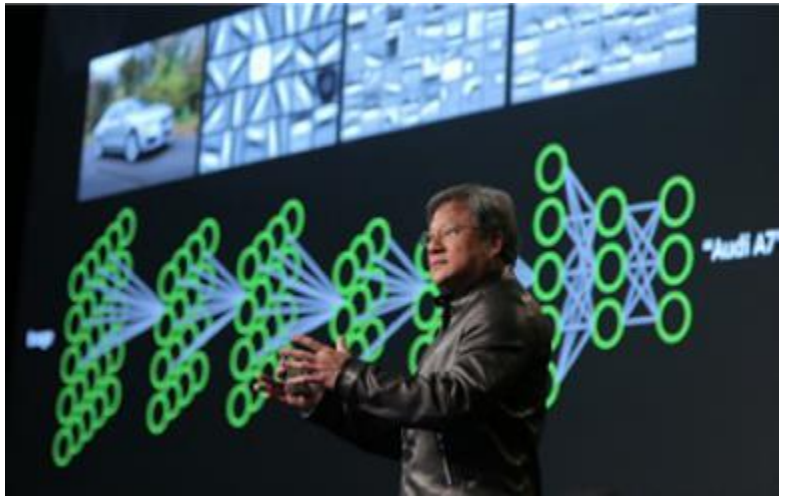
キラーアプリとしてNumPyの存在が大きい

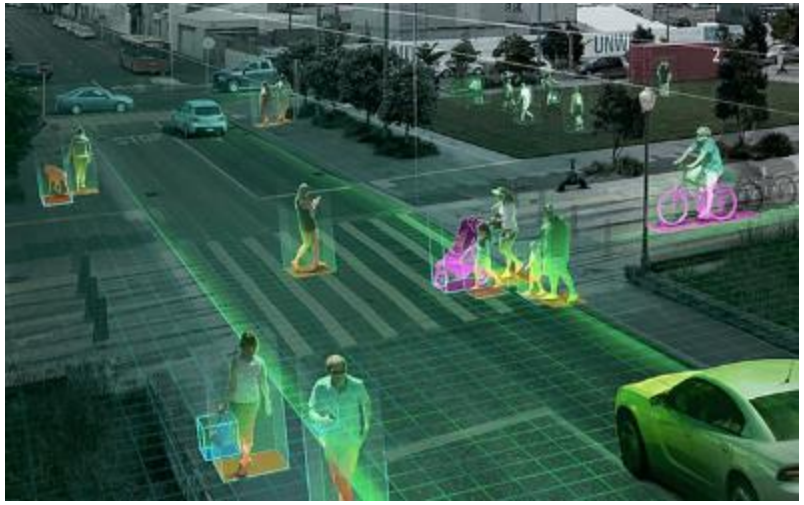
“謎のAI半導体メーカー”☺

話題の企業 – エヌビディア社

行列計算を超高速に並列処理できるチップを開発。それを組み込んだ各種ハードウェアと、それらを活用するためのソフトウェア群を提供。
CADやCG、ゲームなどの3Dグラフィックス領域から、ディープラーニングへ適応領域を拡大。いずれも膨大な行列計算を必要とするアプリケーション領域。

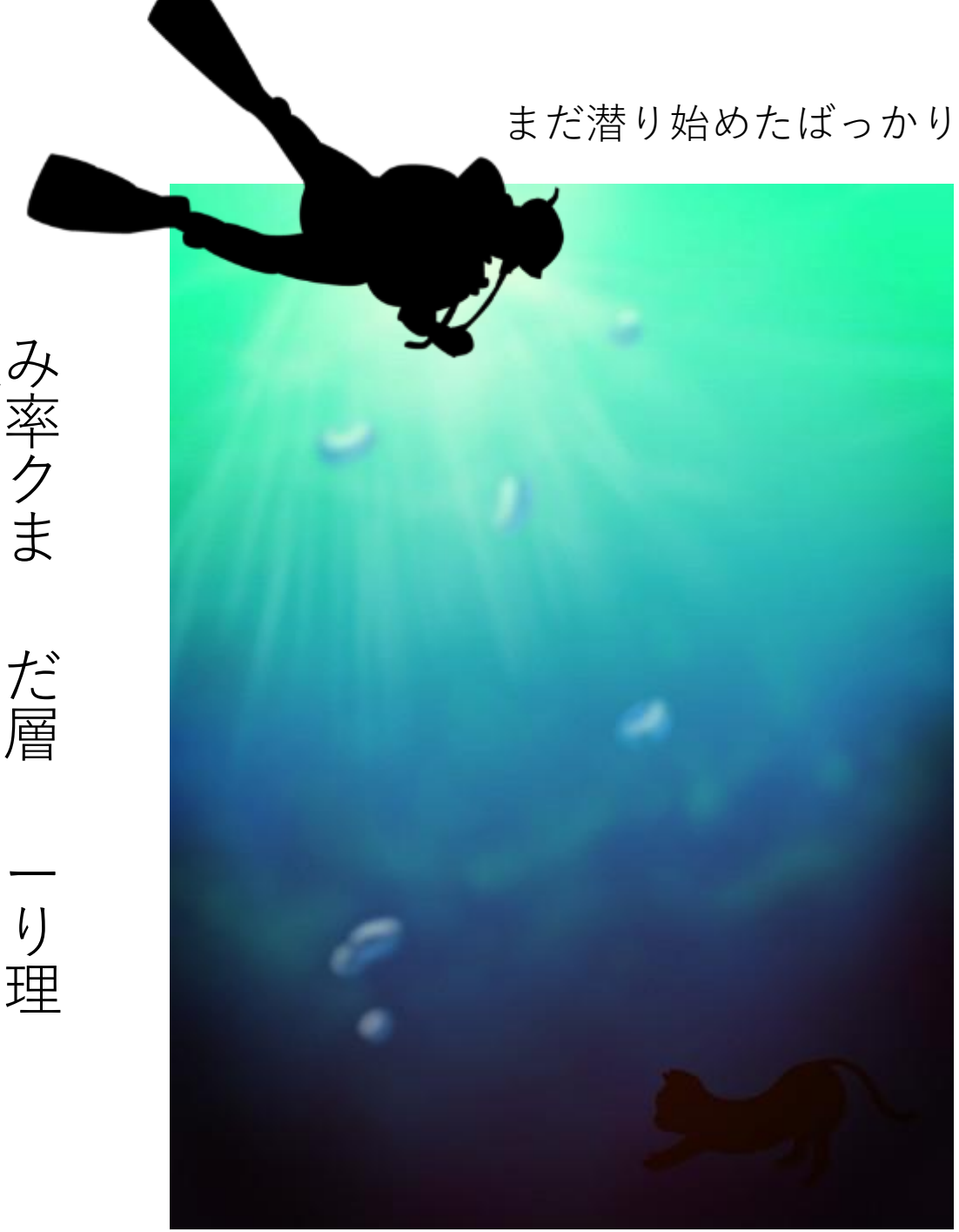






まだまだ先は深いけど

- 今日の話のさらに続きとして、畳み込みニューラルネットワークへの発展、効率よく学習させるための様々なテクニックなど、学ぶべきテーマはまだまだあります
- しかし、その基本となるのは今日学んだニューラルネットワークと、それを多層構造化したモデルです
- これからニュースや記事でディープラーニングを見かけたときに、いままでよりも多少なりとも中身に親近感を持って理解できる一助となれば幸いです



まだ潜り始めたばっかり

手書き文字認識

http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html