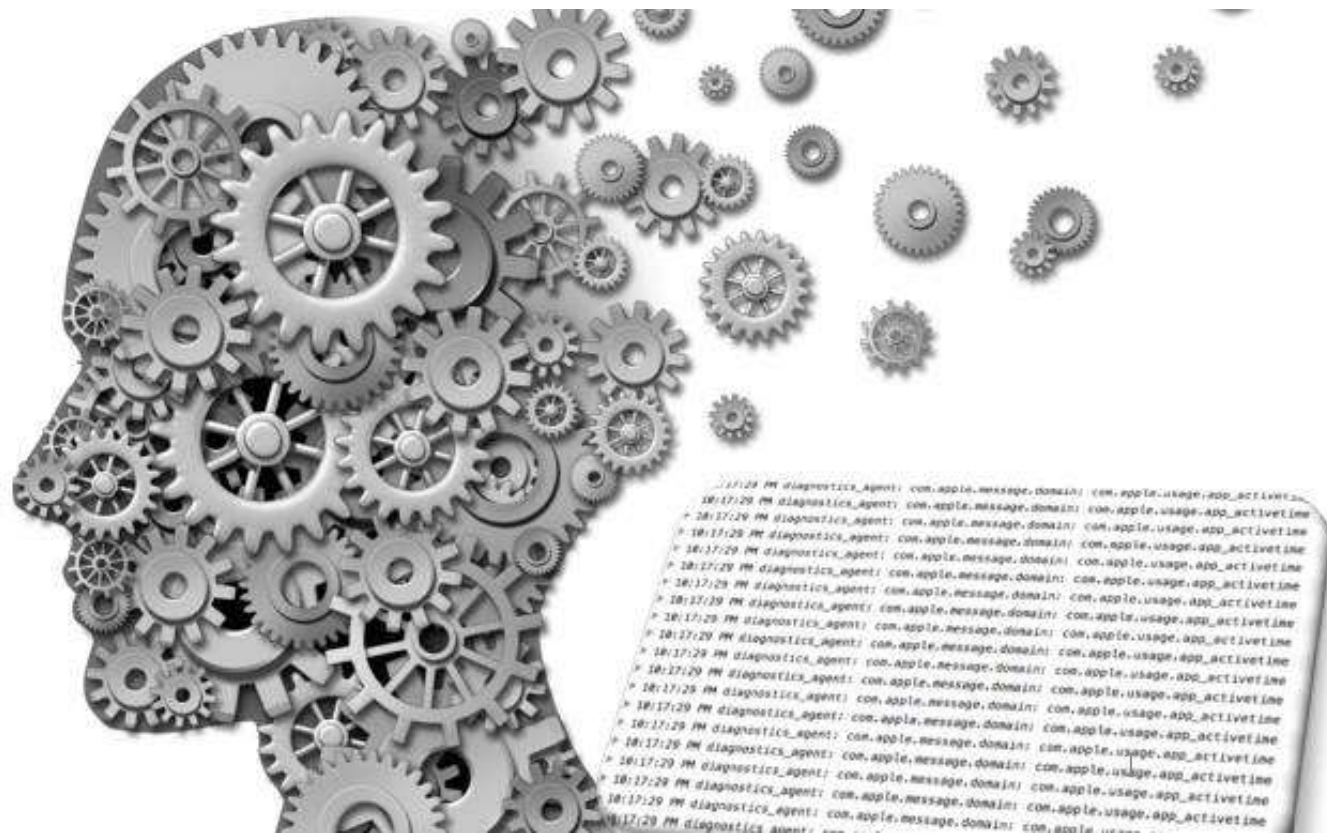


# テクノベート勉強会

人工知能/ディープラーニングのプログラミング・ワークショップ

2017年12月2日

名古屋校 2011期 越智 由浩



# 今日のアジェンダ

- 今日の立ち位置、言葉の整理
- ディープラーニングの
  - 計算のやり方
  - 計算の道具
- ディープラーニングを用いた手書き文字(数字)認識の実例

# 今日のアジェンダ

- 今日の立ち位置、言葉の整理
- ディープラーニングの
  - 計算のやり方
  - 計算の道具
- ディープラーニングを用いた手書き文字(数字)認識の実例

# 今日の立ち位置 ~ 人工知能を学ぶ中で

## 社会・ビジネス視点

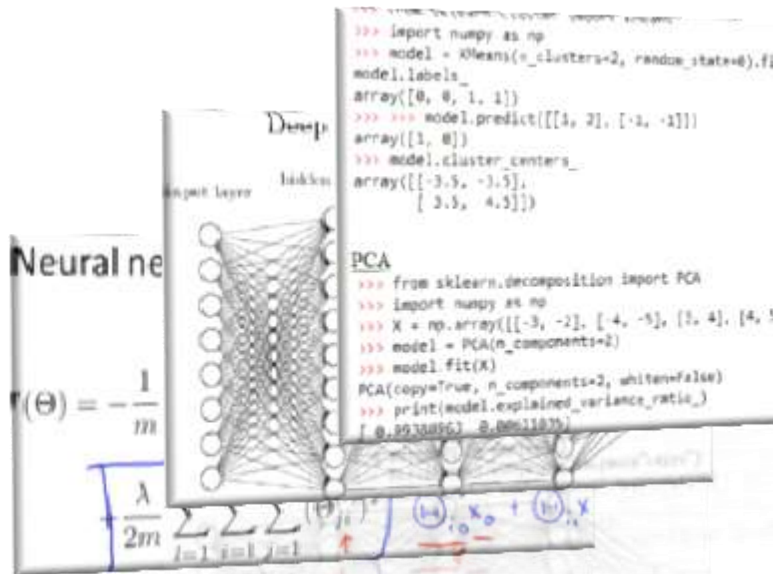
ディープラーニングで何ができるのか、世の中がどう変わるのか



コトバノチガイ

## サイエンス・テクノロジー視点

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



両方の目を持とう!



今日はこちら側の基礎的なところ

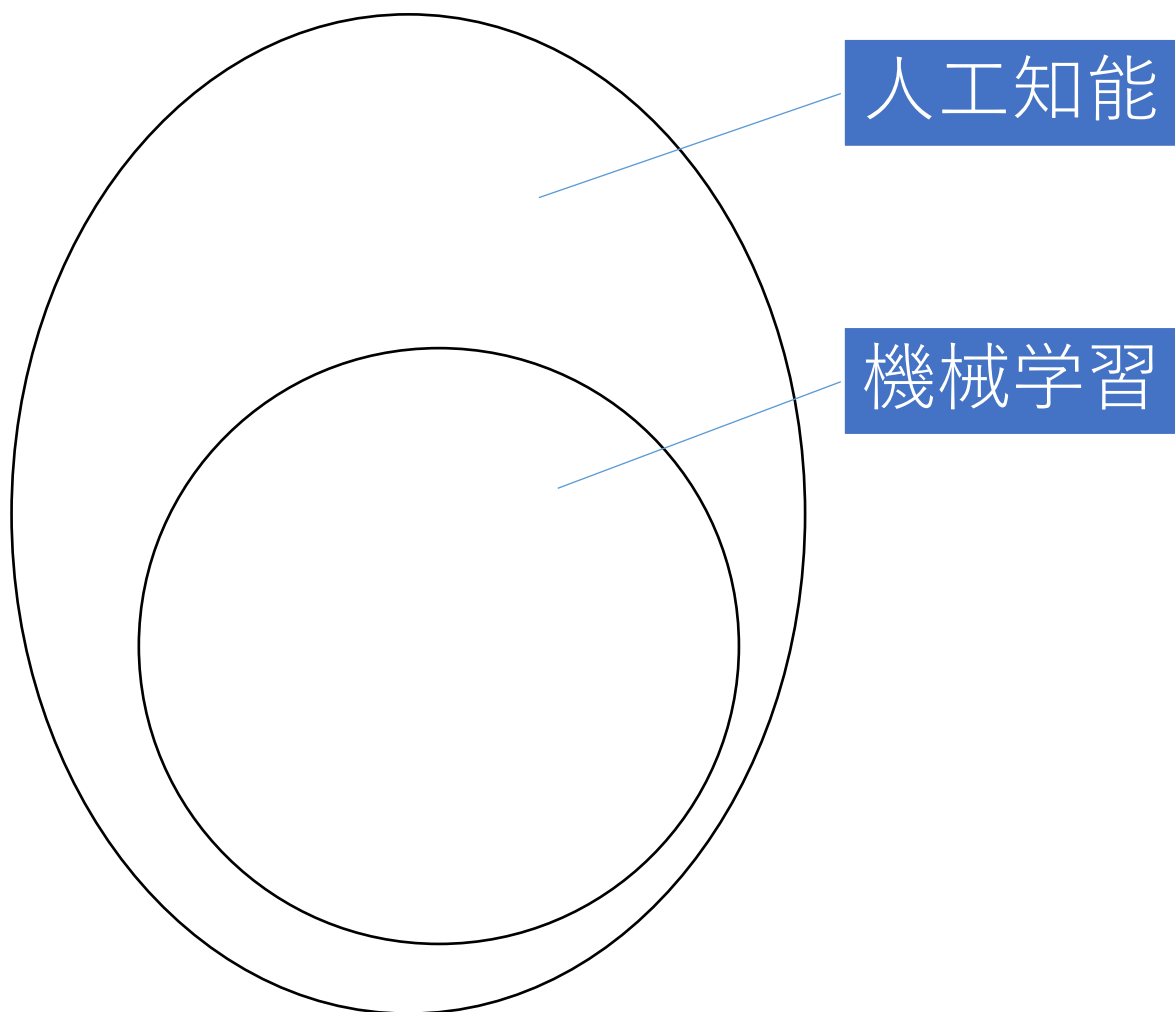
# まず、言葉の整理

## 人工知能



「お、こいつ賢いな!」と思わせるもの、ふるまい。  
またそれを探求する学問領域。

# まず、言葉の整理



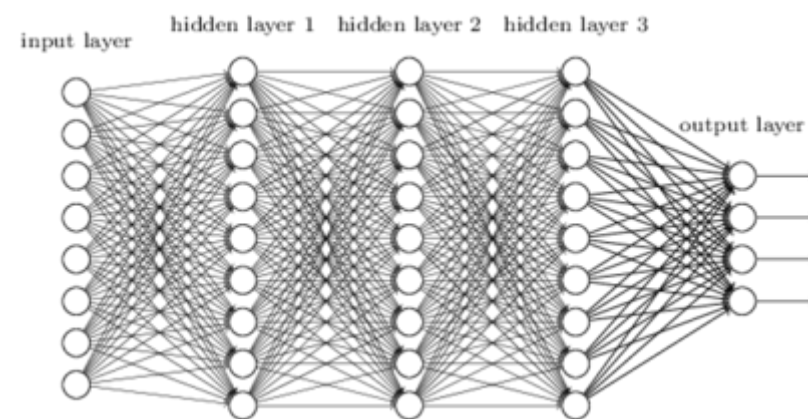
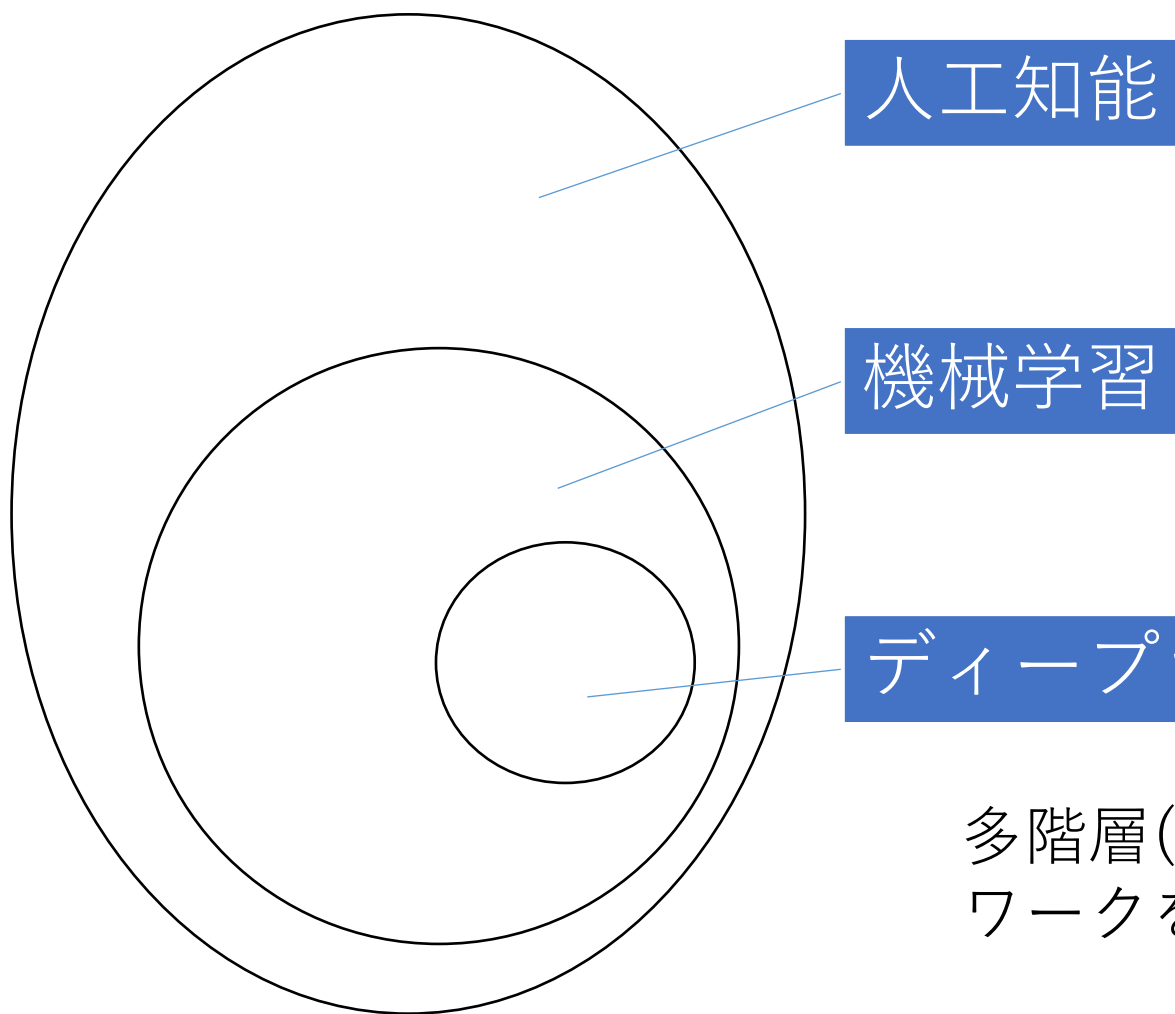
人工知能を実現する一つ的手段。  
過去のデータ(知見/経験)に基づいて：

- ・ 将来を予測する
- ・ 未知のものを分類する

例)

eコマースサイトのリコメンド  
迷惑メールの自動振り分け

# まず、言葉の整理



多階層(ディープな)ニューラルネットワークを用いた機械学習(ラーニング)

# 今日のアジェンダ

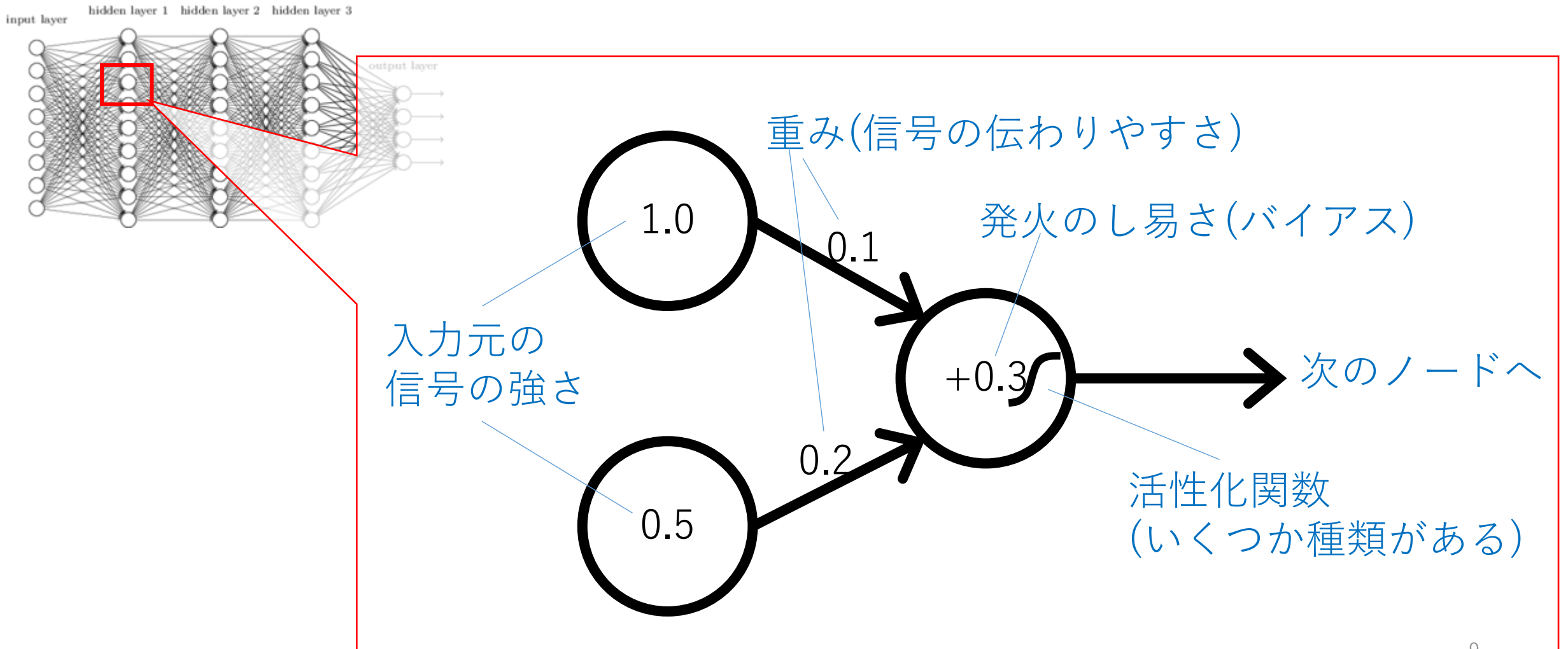
- 今日の立ち位置、言葉の整理

- ディープラーニングの
  - 計算のやり方
  - 計算の道具

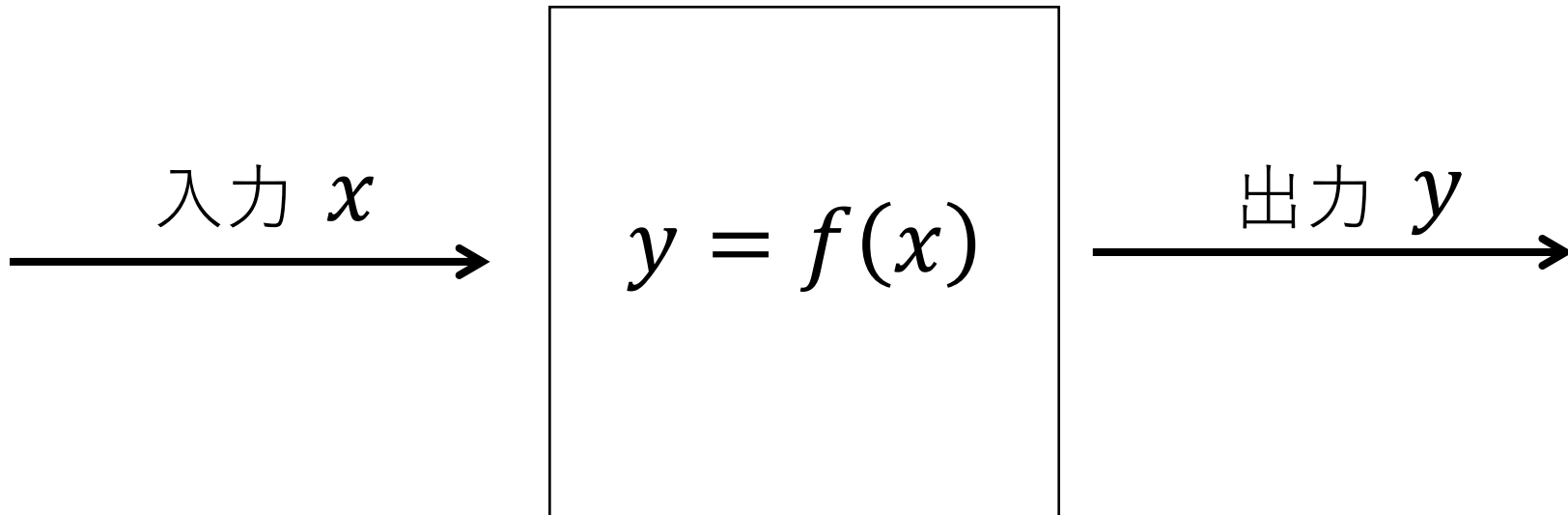
- ディープラーニングを用いた手書き文字(数字)認識の実例



# ニューラルネットワークモデルの計算 ルール

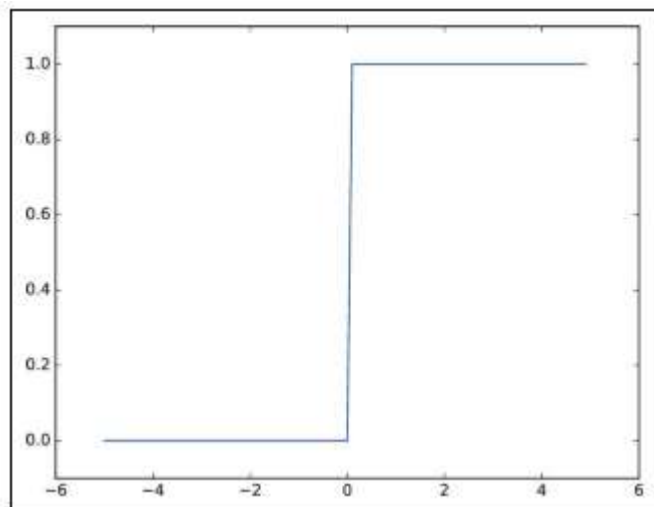


「関数」ってなに？

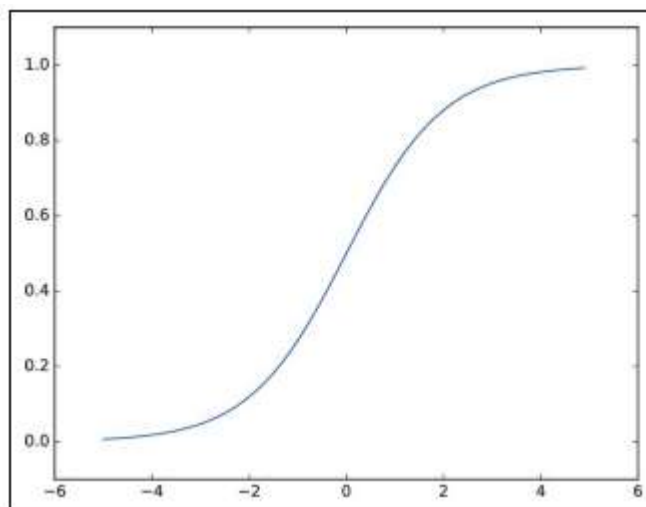


# 活性化関数

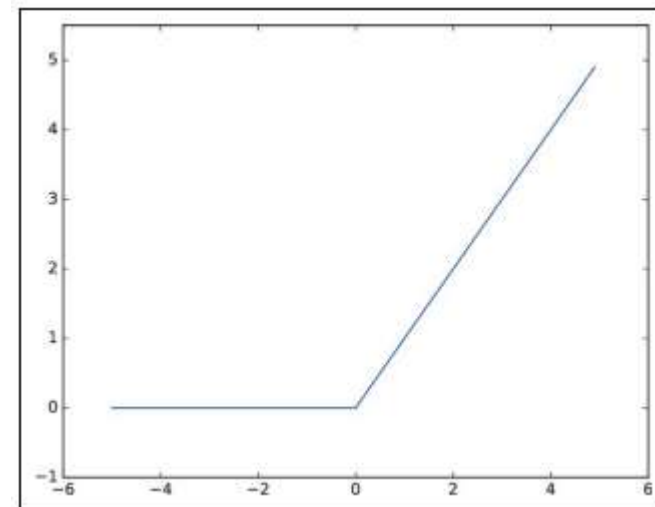
ステップ関数



シグモイド関数



ReLU関数  
(Rectified Linear Unit)



出力  
↑  
入力の総和 + バイアス  
→

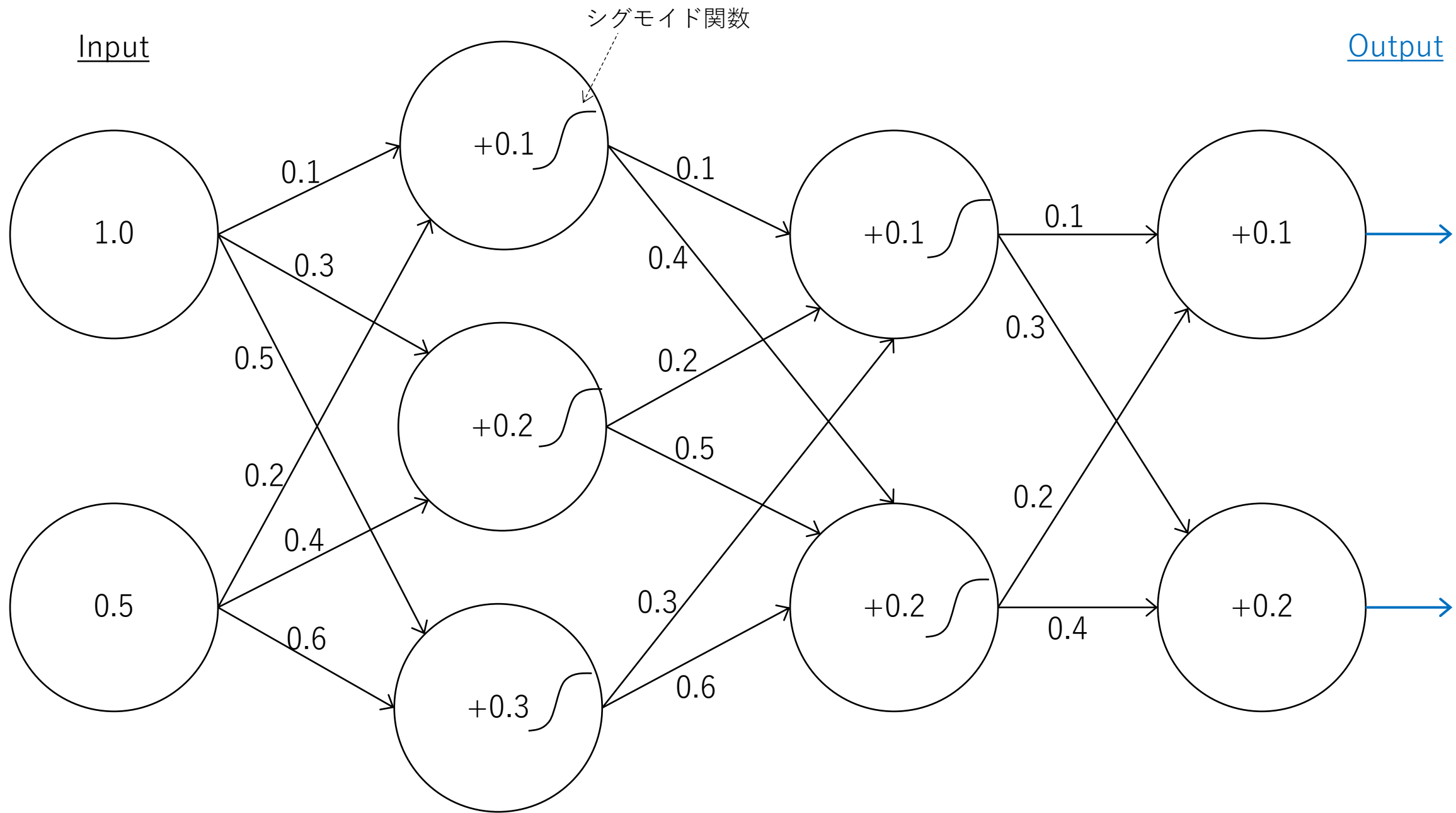
$$h(x) = \begin{cases} 0 & (x \leq 0) \\ 1 & (x > 0) \end{cases}$$

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

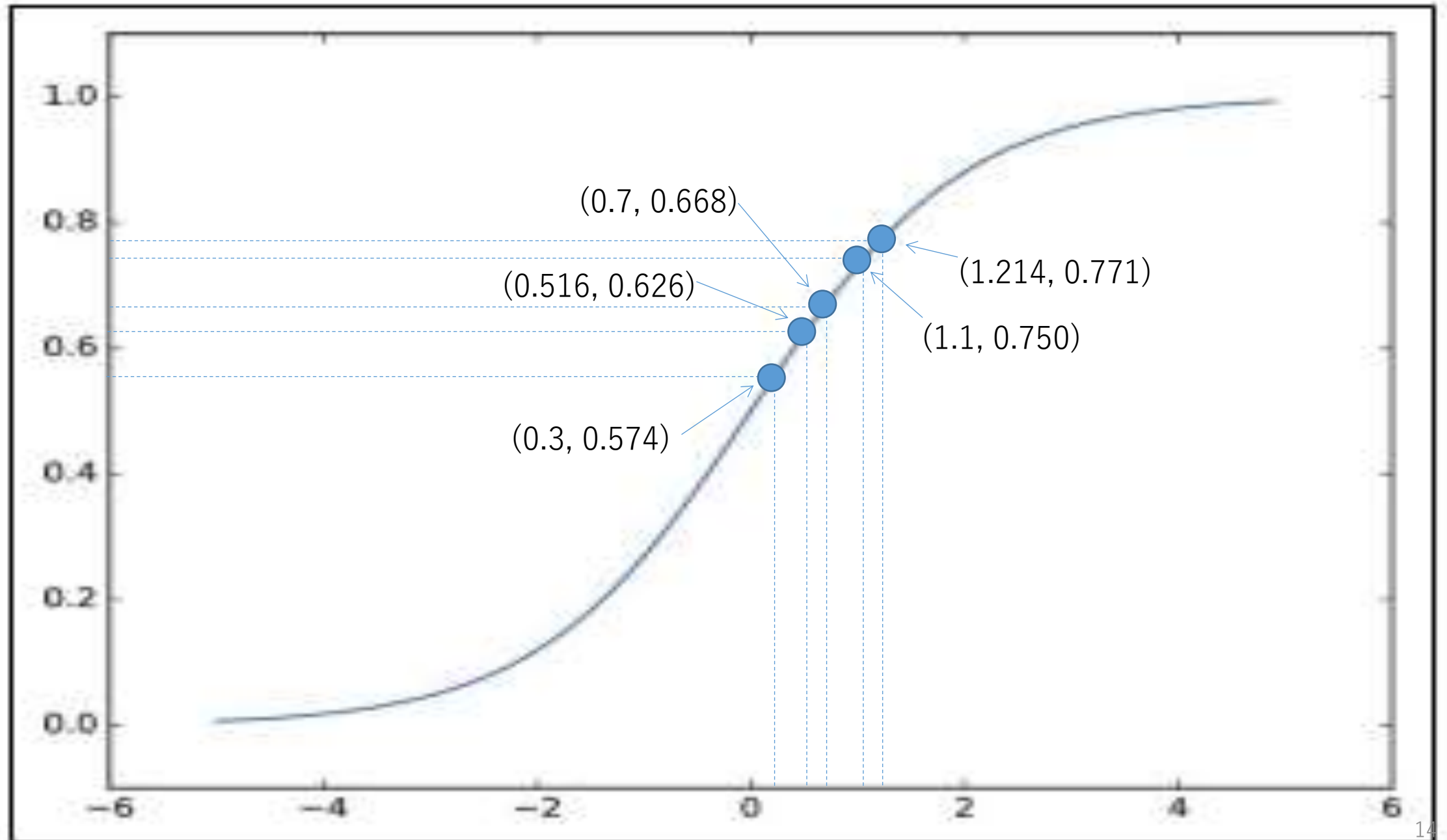
$$h(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

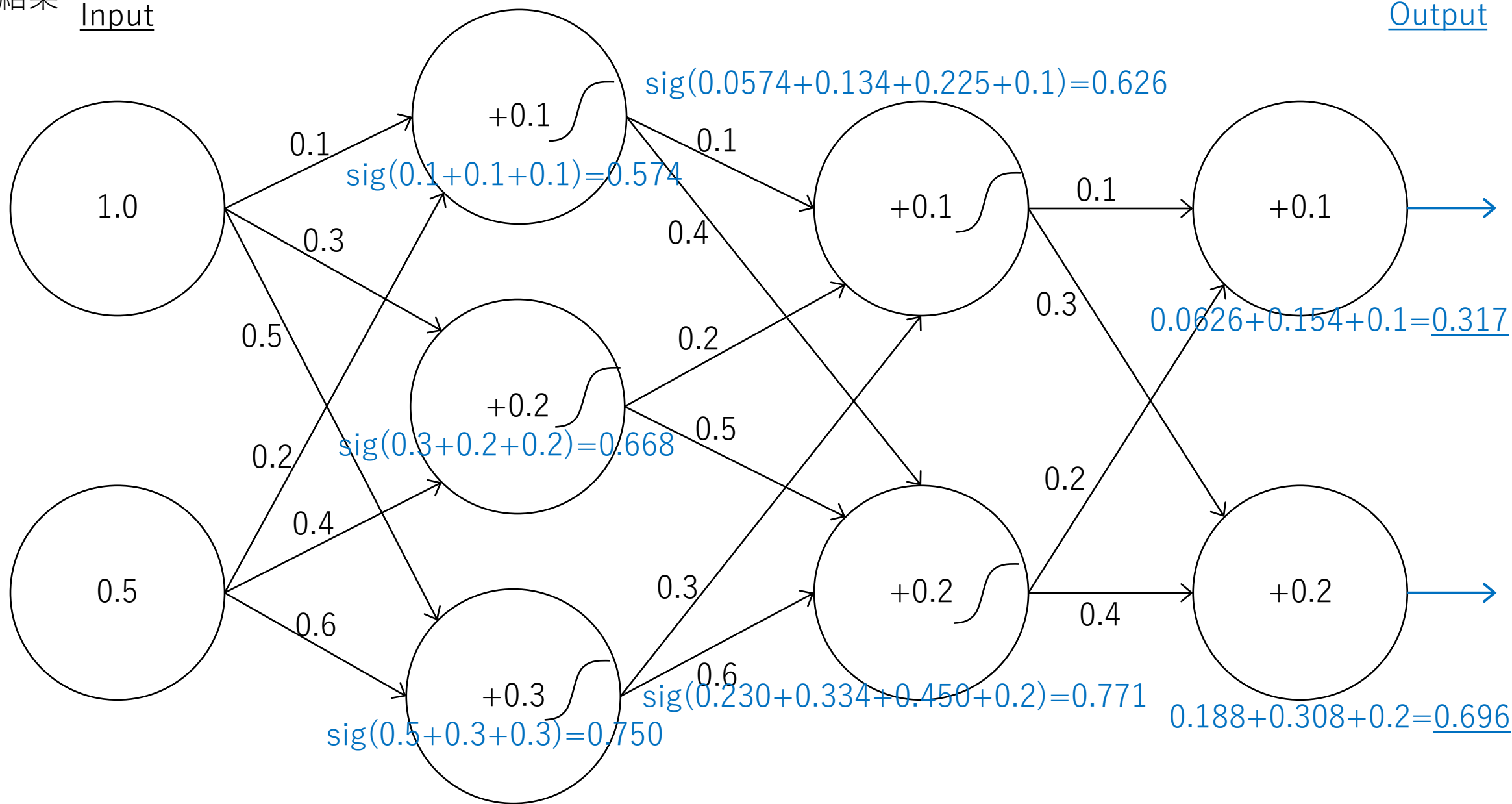
手触り感持って理解するために、さほど  
ディープじゃないニューラルネットワークモ  
デルを使って手計算してみよう



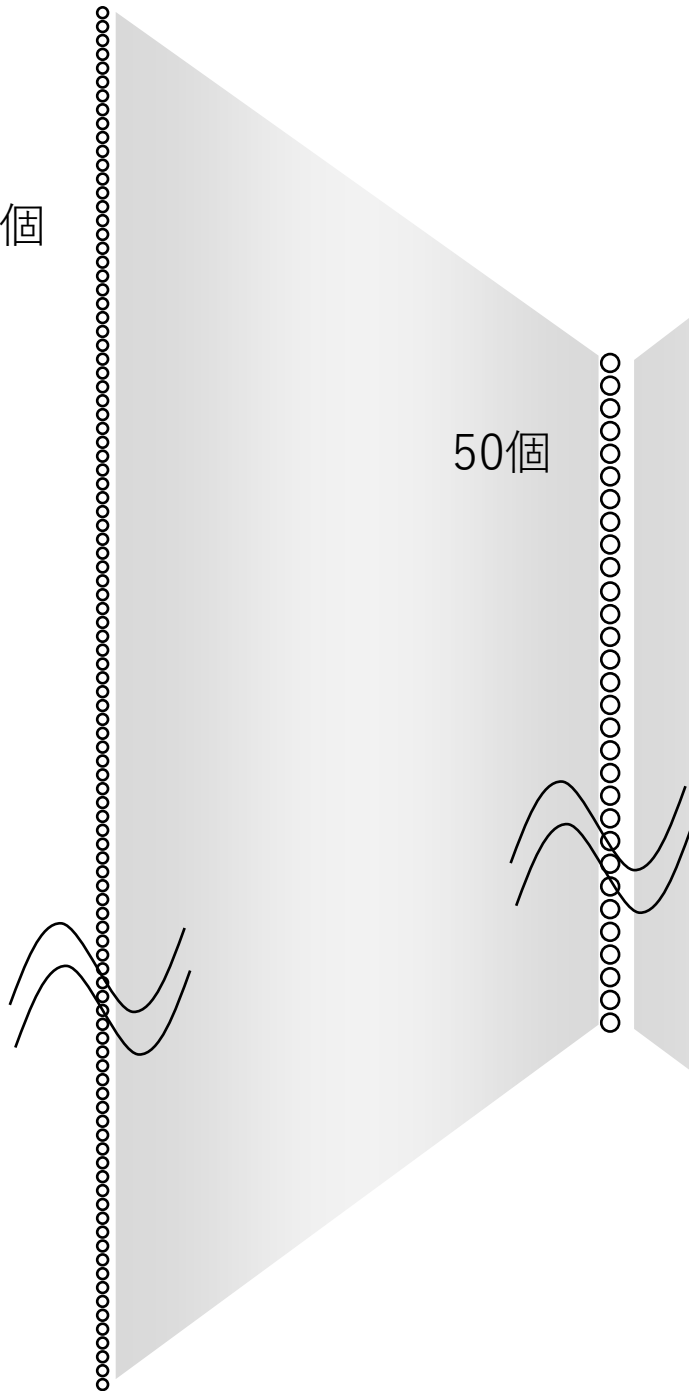


# シグモイド関数、参照シート



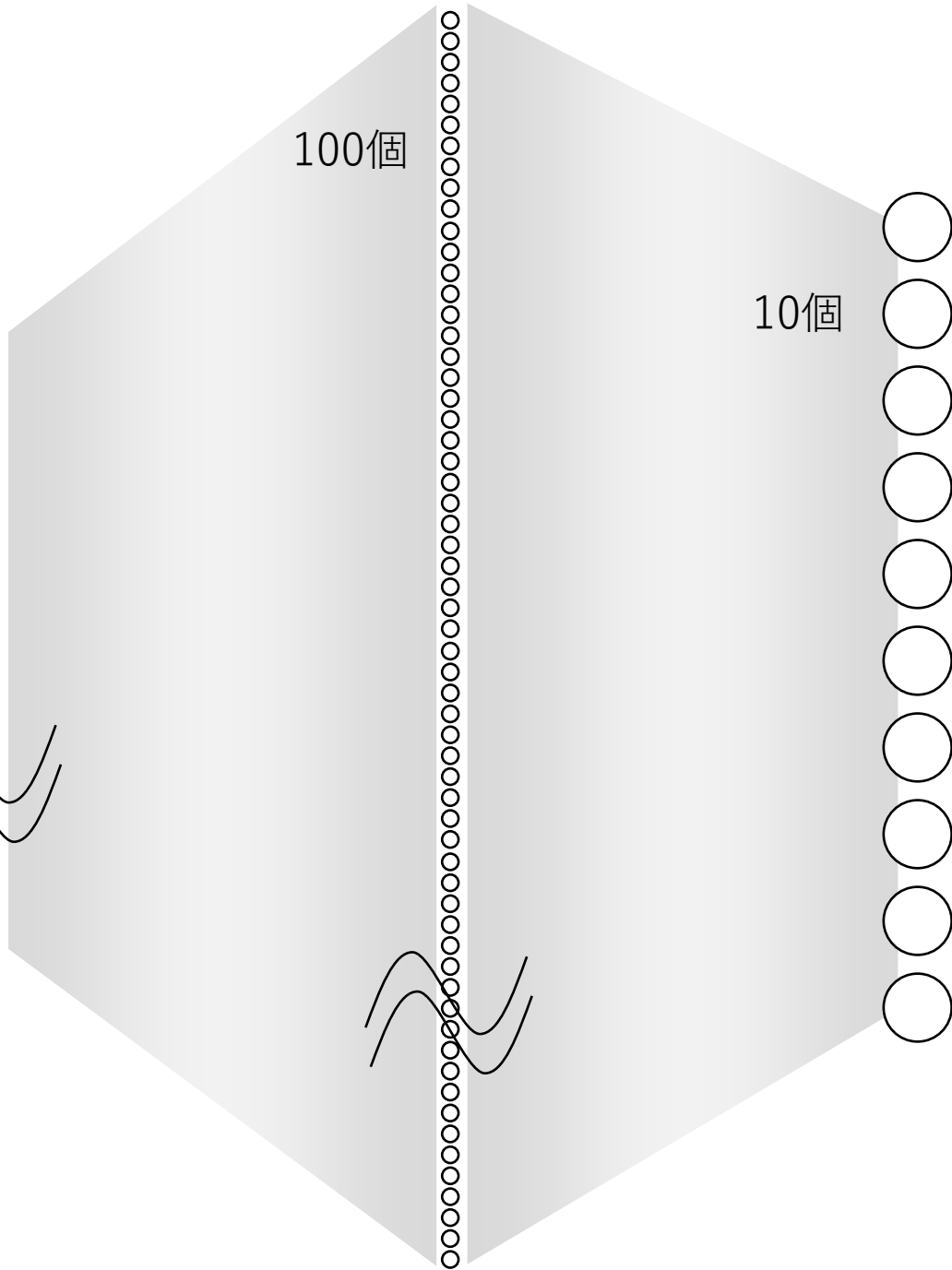
InputOutput

784個



50個

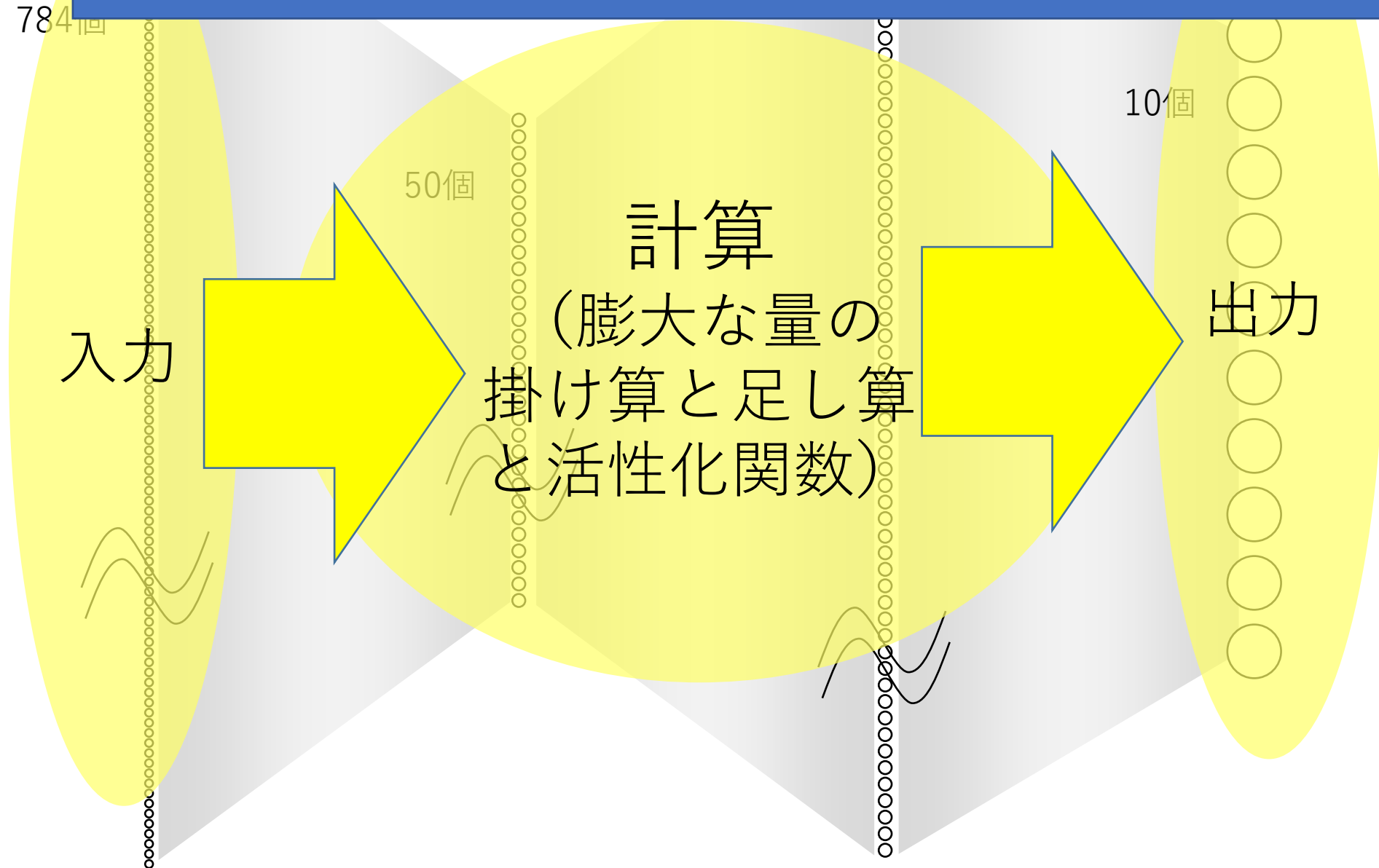
100個



10個



このくらいの数のニューラルネットワークを使うと、  
手書きの文字認識ができちゃいます。



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！



# 今日のアジェンダ

- 今日の立ち位置、言葉の整理
- ディープラーニングの
  - 計算のやり方
  - 計算の道具
- ディープラーニングを用いた手書き文字(数字)認識の実例

ふたつの道具を手に入れよう

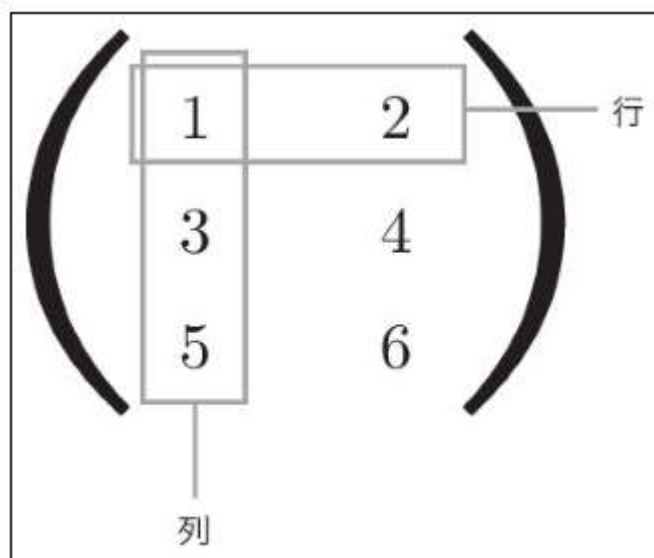
- 数学の道具 – 行列の計算
- プログラミング – Python  
(なぜPythonかは後ほど補足)



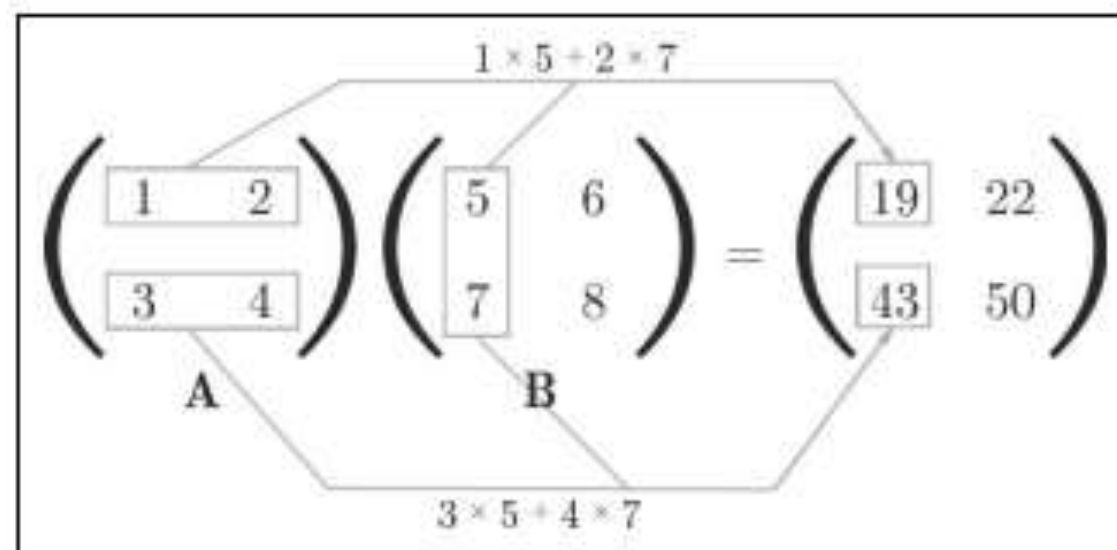
# 数学の便利な道具－行列と行列の内積

なぜ便利かは後ほどわかる

3 × 2の行列の例



行列の内積の例



行列計算に慣れ親しむ

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

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



# 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

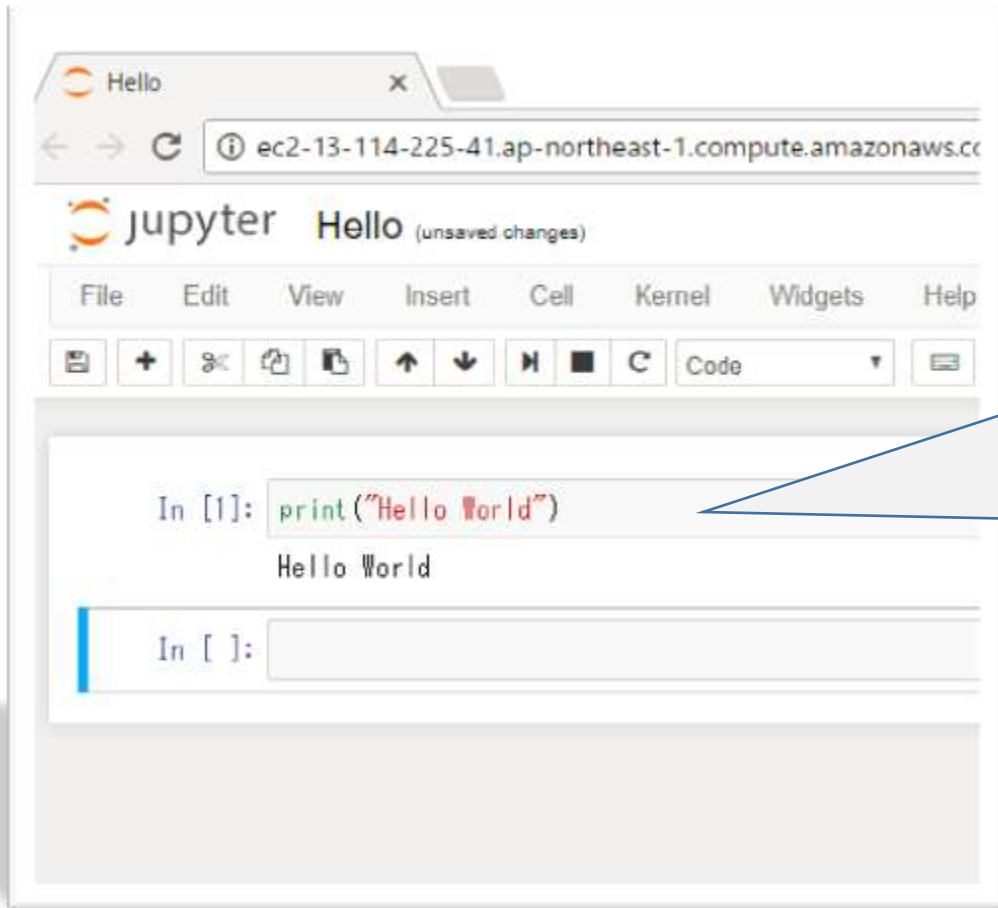
• ベクトル = 配列 = リスト  
数学っぽい                      プログラム..                      特にPython

## 表記方法

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



# いよいよPythonプログラミングです



Jupyterというブラウザベースの開発環境を使います。



+



で、セル内のコマンド(プログラム)を実行します。

# 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

# その他の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？

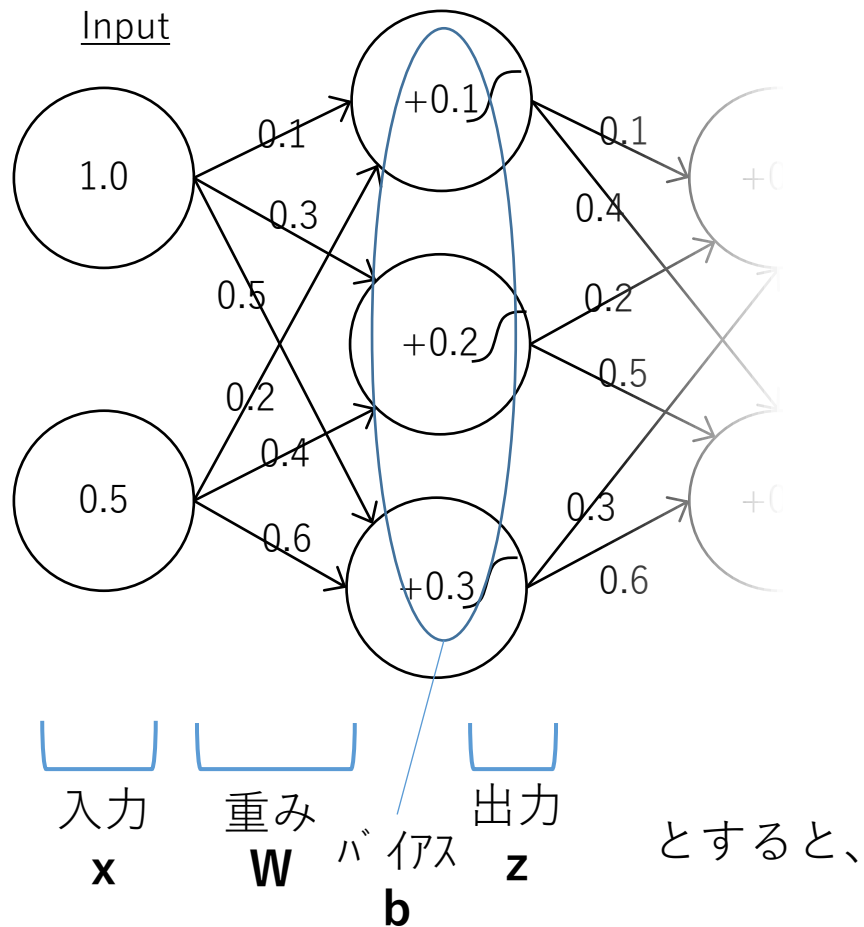
なぜ、行列（内積）なんていう道具を使うのか？

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

$\text{sig}()$  …シグモイド関数

先ほどの模型の前半部分



$$\begin{aligned} z &= \text{sig}(W \cdot x + b) \\ &= \text{sig}\left(\begin{pmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \\ 0.5 & 0.6 \end{pmatrix} \cdot \begin{pmatrix} 1.0 \\ 0.5 \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0.2 \\ 0.3 \end{pmatrix}\right) \\ &= \text{sig}\left(\begin{pmatrix} 0.2 \\ 0.5 \\ 0.8 \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0.2 \\ 0.3 \end{pmatrix}\right) \\ &= \text{sig}\left(\begin{pmatrix} 0.3 \\ 0.7 \\ 1.1 \end{pmatrix}\right) = \begin{pmatrix} 0.574 \\ 0.668 \\ 0.750 \end{pmatrix} \end{aligned}$$

# 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

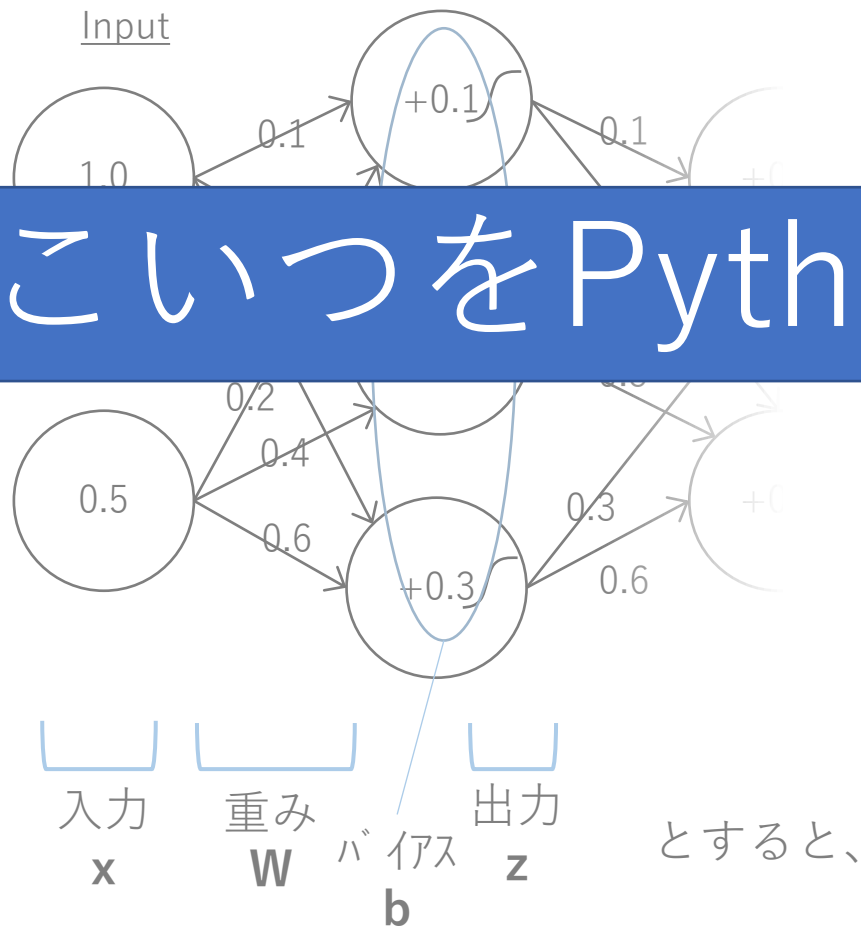
sig() …シグモイド関数

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

こいつをPythonで実装してみよう！

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

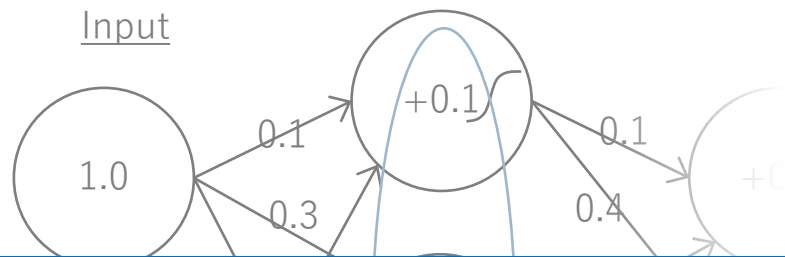
$$= \text{sig}\left(\begin{pmatrix} 0.3 \\ 0.7 \\ 1.1 \end{pmatrix}\right) = \begin{pmatrix} 0.574 \\ 0.668 \\ 0.750 \end{pmatrix}$$





# 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

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

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

$$\begin{pmatrix} 0.2 \\ 0.5 \\ 0.8 \end{pmatrix} + \begin{pmatrix} 0.1 \\ 0.2 \\ 0.3 \end{pmatrix}$$

$$\begin{pmatrix} 0.3 \\ 0.7 \\ 1.1 \end{pmatrix} = \begin{pmatrix} 0.574 \\ 0.668 \\ 0.750 \end{pmatrix}$$

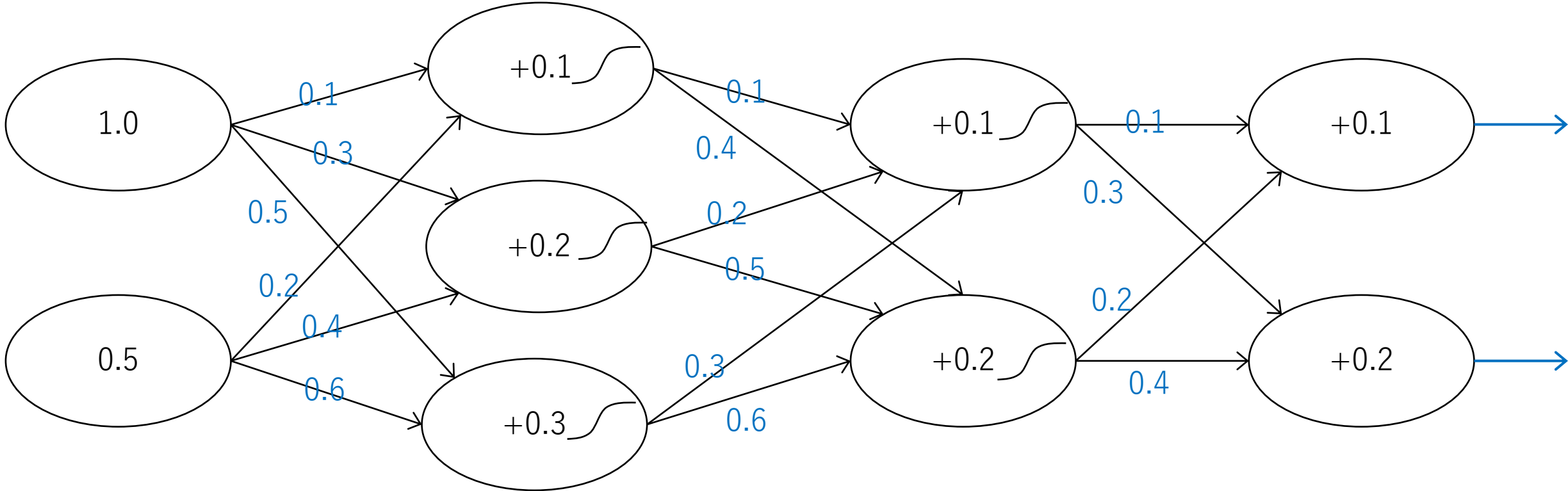
```
import numpy as np

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



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

$z1 = \text{sig}(W1 \cdot x + b1)$

$z2 = \text{sig}(W2 \cdot z1 + b2)$

$z3 = W3 \cdot z2 + b3$

sig() …シグモイド関数

```
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 )

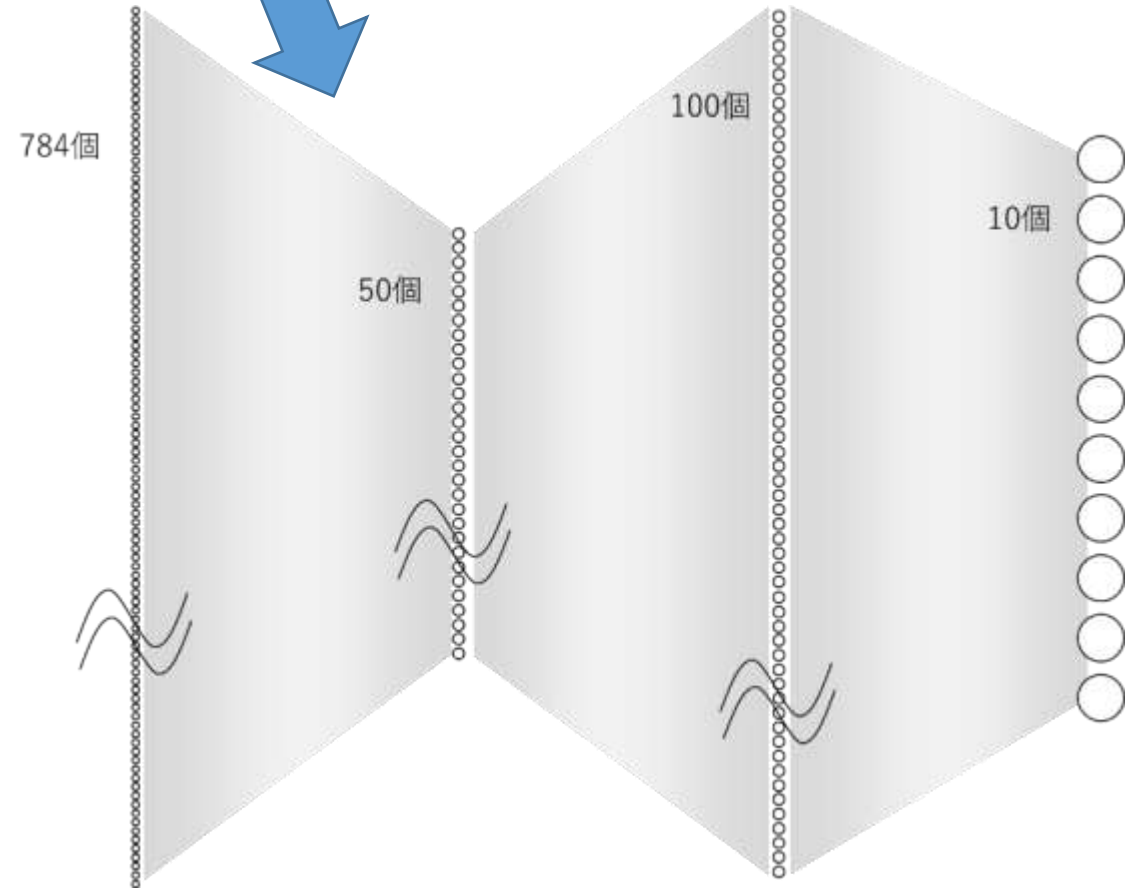
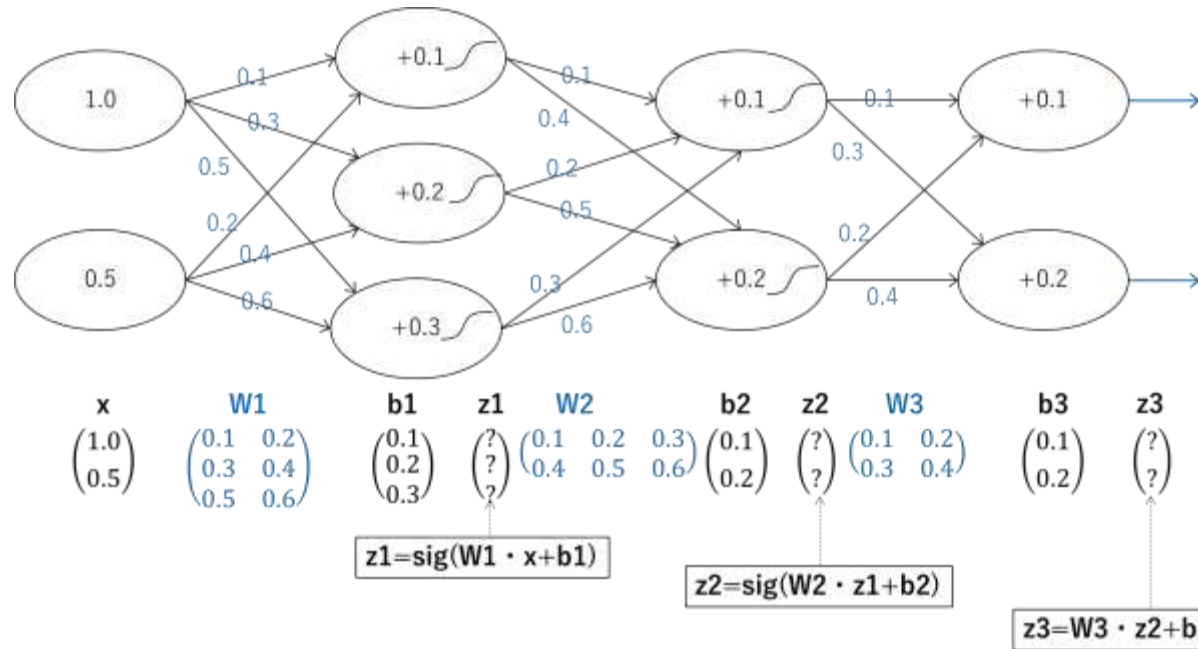
W2 = np.array([[0.1,0.2,0.3], [0.4,0.5,0.6]])
b2 = np.array([0.1, 0.2])
z2 = sig( np.dot(W2, z1) + b2 )

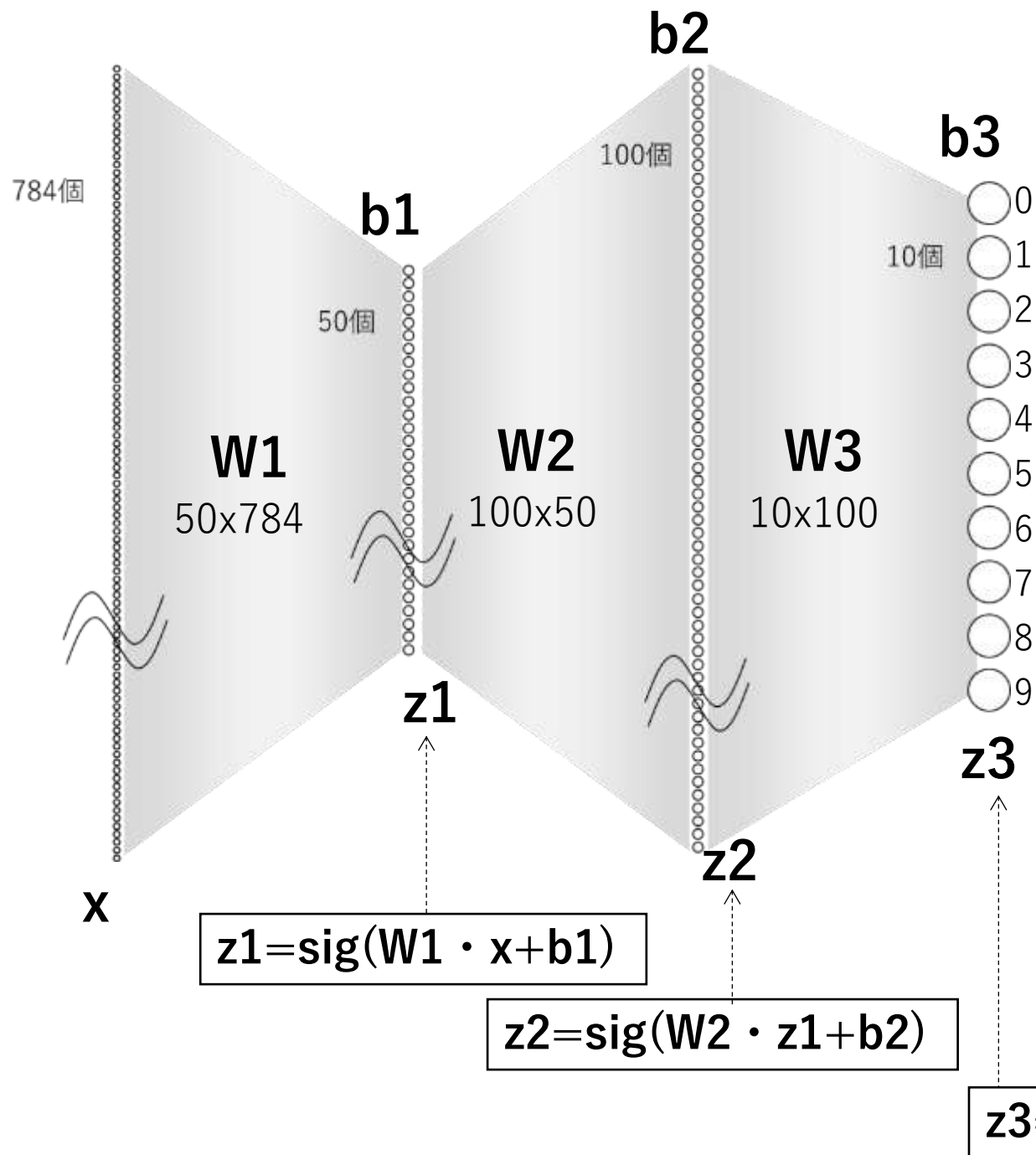
W3 = np.array([[0.1,0.2], [0.3,0.4]])
b3 = np.array([0.1, 0.2])
z3 = np.dot(W3, z2) + b3

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 もいけるんじゃない？





ここの理解、  
今日の踏ん張りどころ

脳みそに汗をかけ

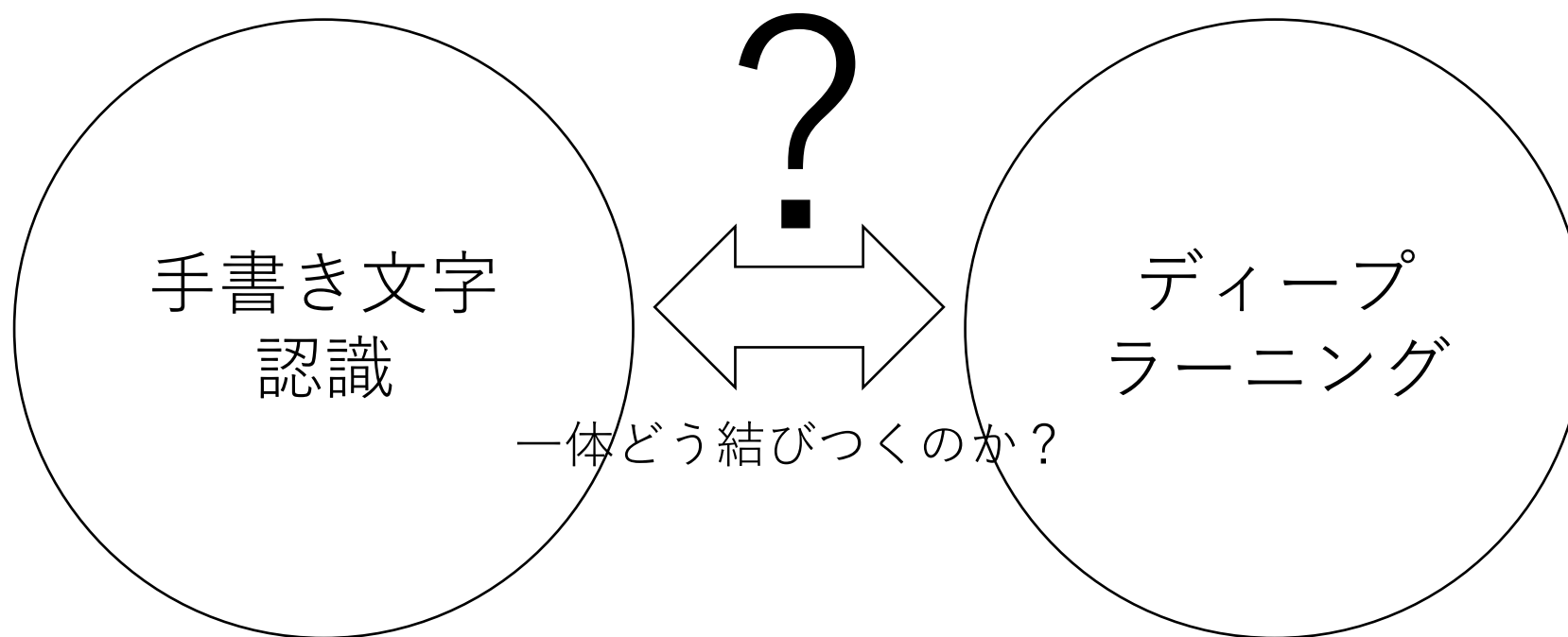


層やノードの数が増えても、  
行列を使うとシンプルに表現  
できる！  
そしてPythonプログラムで  
記述できる。

# 今日のアジェンダ

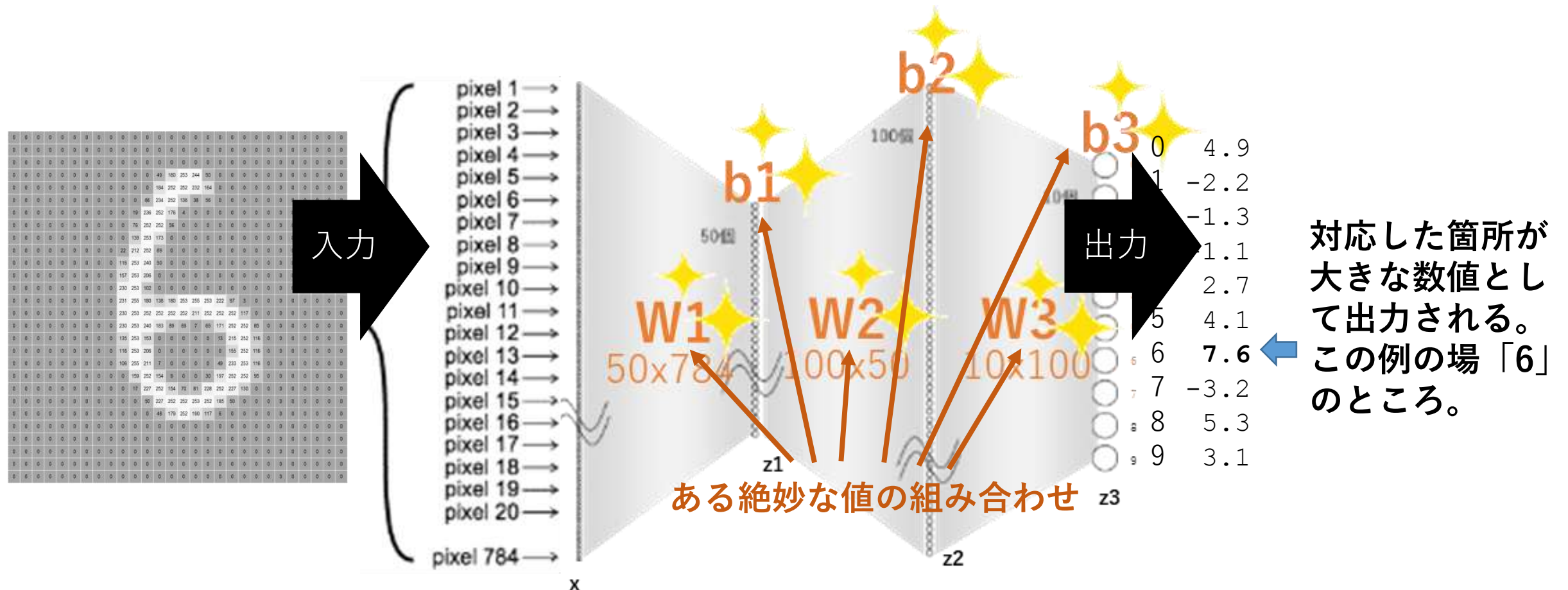
- 今日の立ち位置、言葉の整理
- ディープラーニングの
  - 計算のやり方
  - 計算の道具
- ディープラーニングを用いた手書き文字(数字)認識の実例

# 手書き文字(数字)認識をさせてみる



人間が文字認識する、をホワイトボードでやってみる

目指すべき状態—ニューラルネットワークが文字(数字)を認識できる—とはどういうことか？





# ディープラーニングの全体の流れ

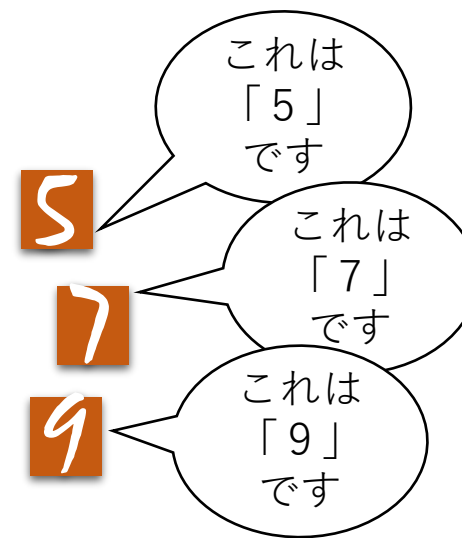
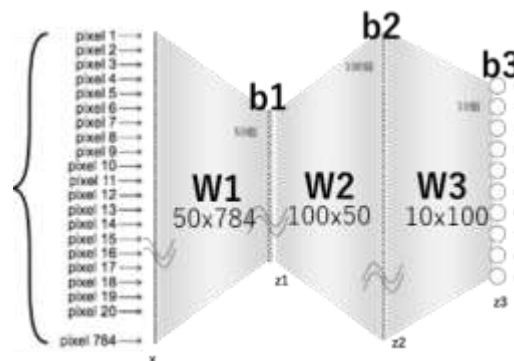


- トレーニング用データの入手・整理

- トレーニング用データによる学習

- 未知のものを分類

今回の文字  
(数字)認識  
の場合



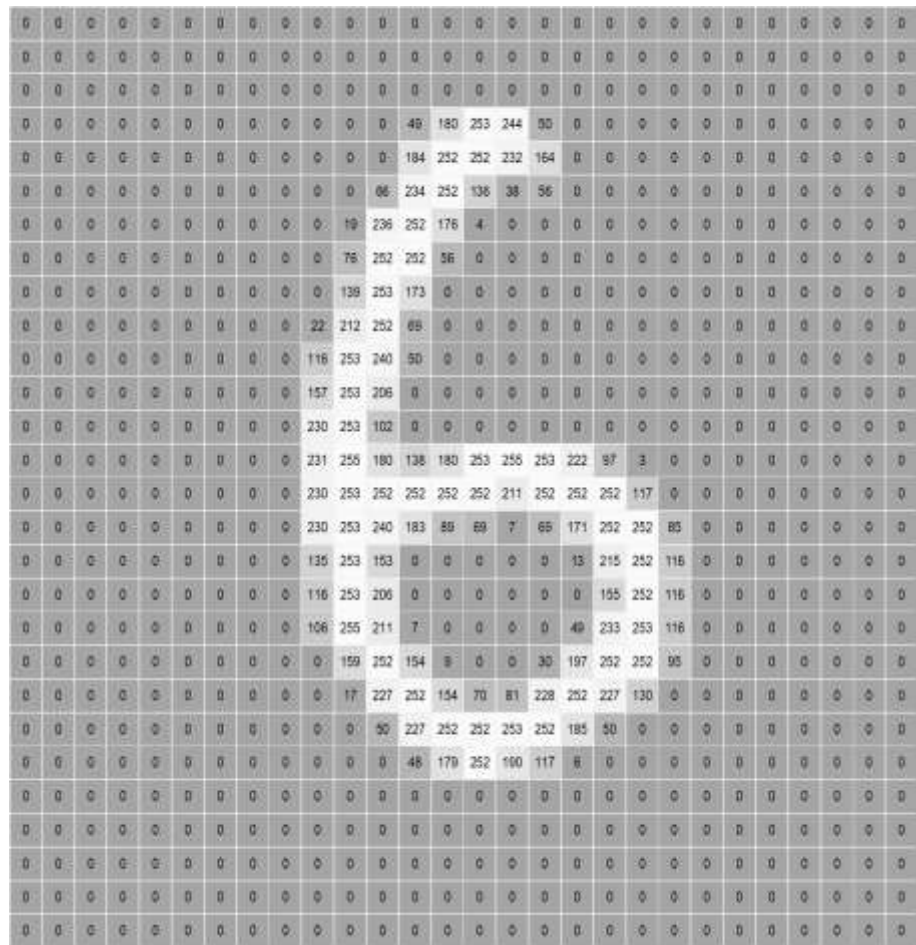
MNIST手書き文字データ 6万文字を  
トレーニング用データとして使用  
<http://yann.lecun.com/exdb/mnist/>

(トレーニングデータとは別の)  
テスト用データ 1万文字を使用

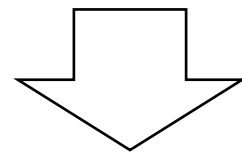
# ディープラーニングにおける学習とは、 手書き文字を数値化し..

28

28



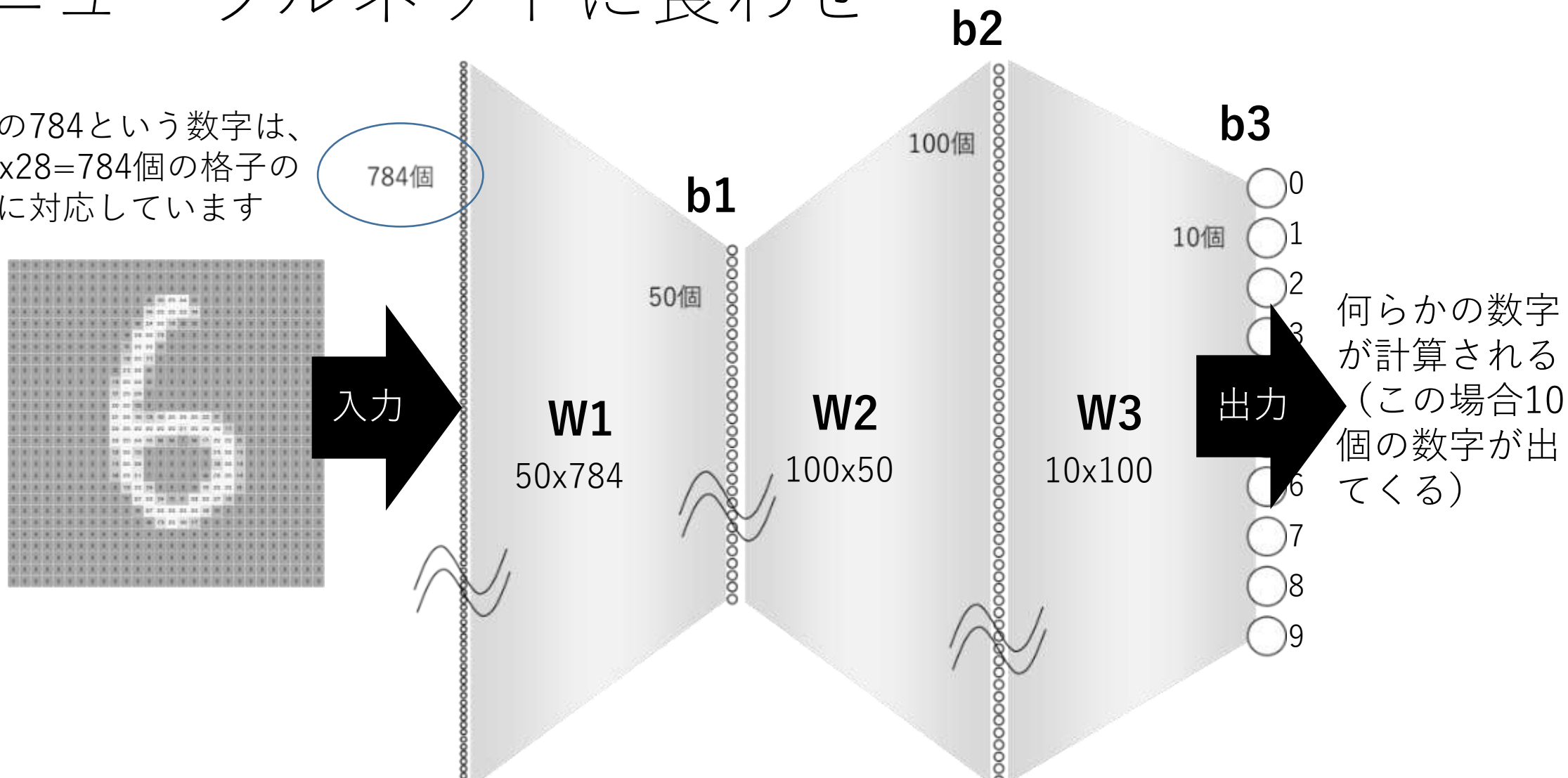
28x28=784個の格子(ピクセル)ごとに、0~255の値で明るさを表現



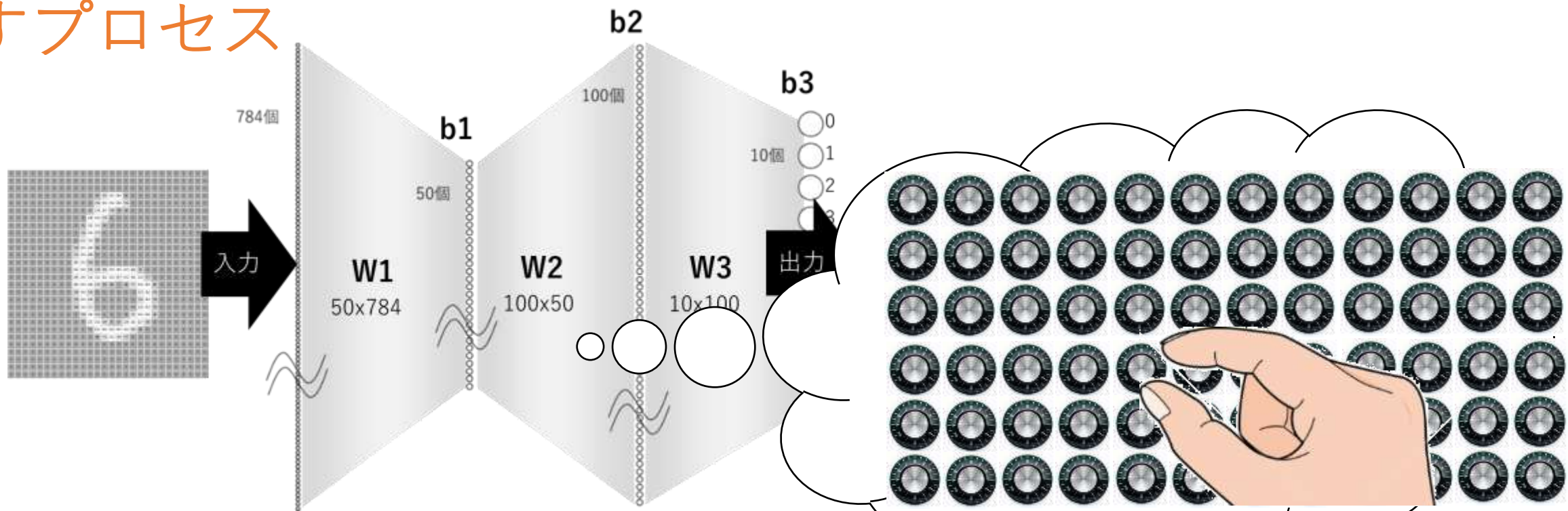
784個の数字の配列で、一つの手書き文字を表現できる

ディープラーニングにおける学習とは、  
手書き文字を数値化し..  
ニューラルネットに食わせ..

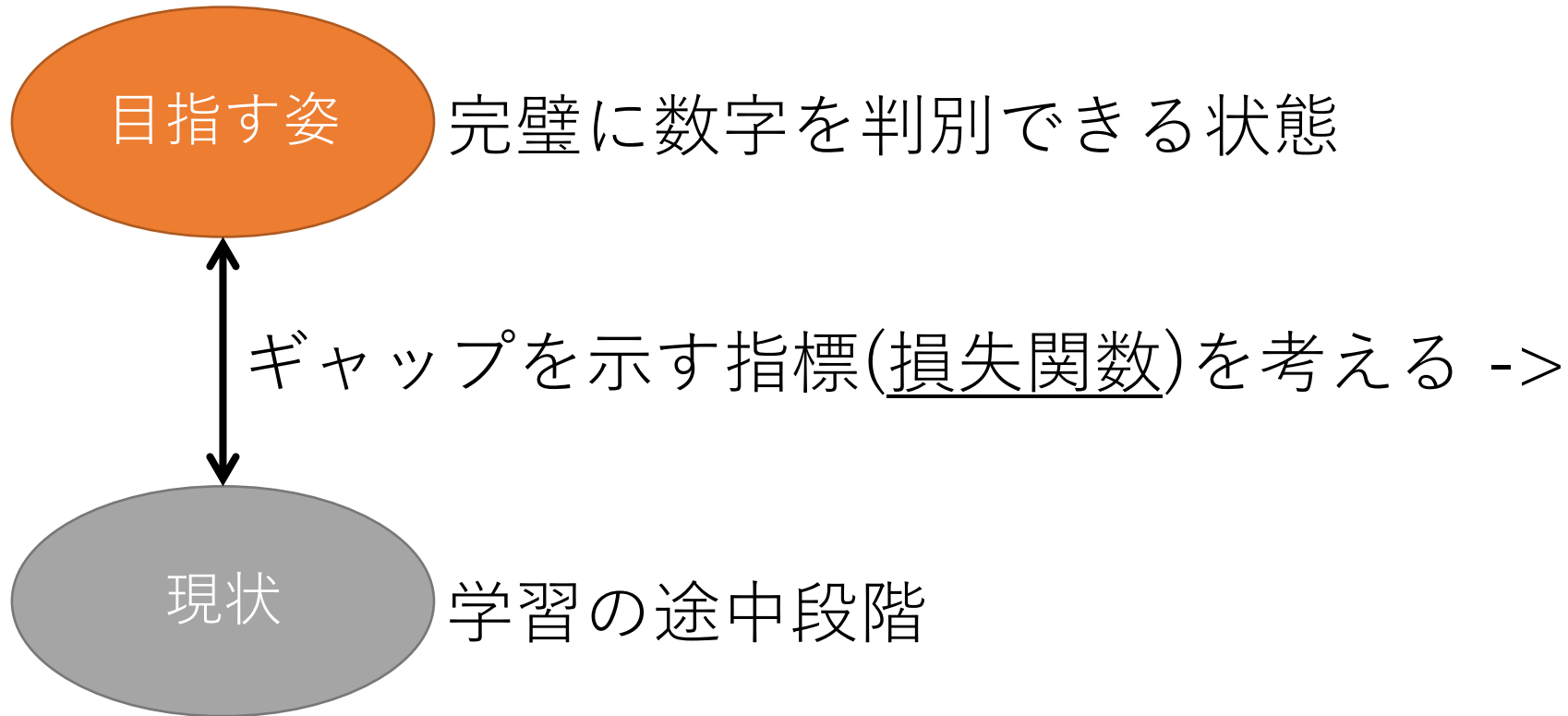
この784という数字は、  
 $28 \times 28 = 784$ 個の格子の  
数に対応しています



ディープラーニングにおける学習とは、  
手書き文字を数値化し..  
ニューラルネットに食わせ..  
ちよつとずつパラメータ $W1, W2, W3, b1, b2, b3$  (全  
部で45,350個の数字)を変えながら目指すべき状態  
になるように**絶妙なパラメータの組み合わせを探  
すプロセス**



# どうアプローチするか



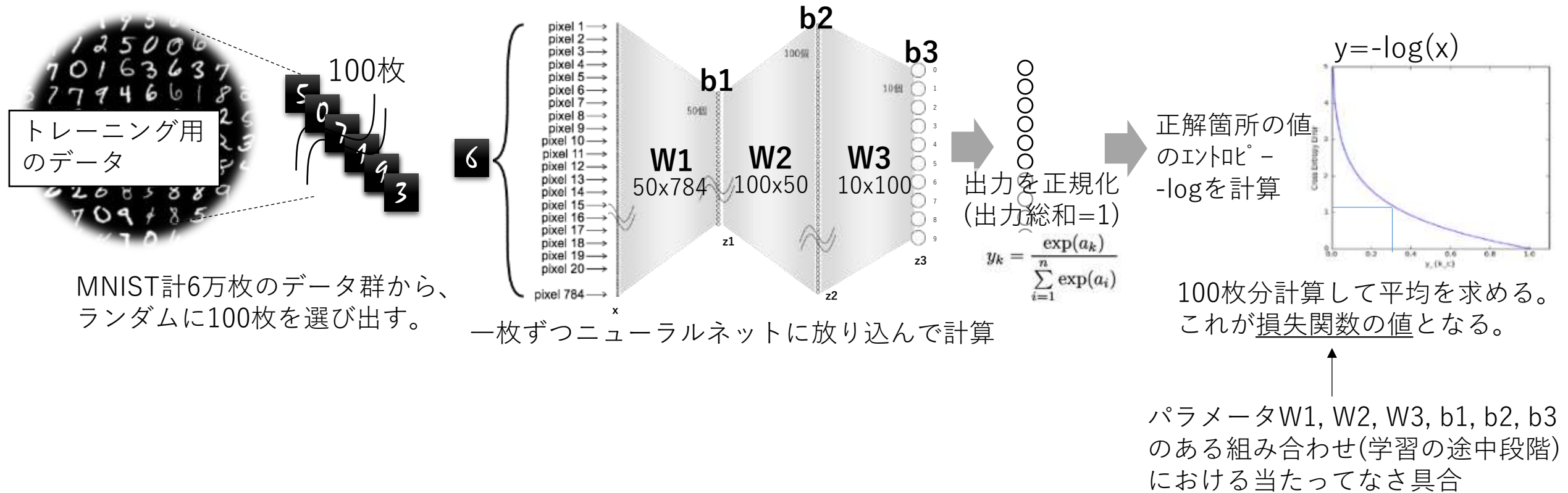
完璧に数字を判別できる状態

この損失関数が最小となるパラメータ $W1$ ,  $W2$ ,  $W3$ ,  $b1$ ,  $b2$ ,  $b3$  の組み合わせを探す

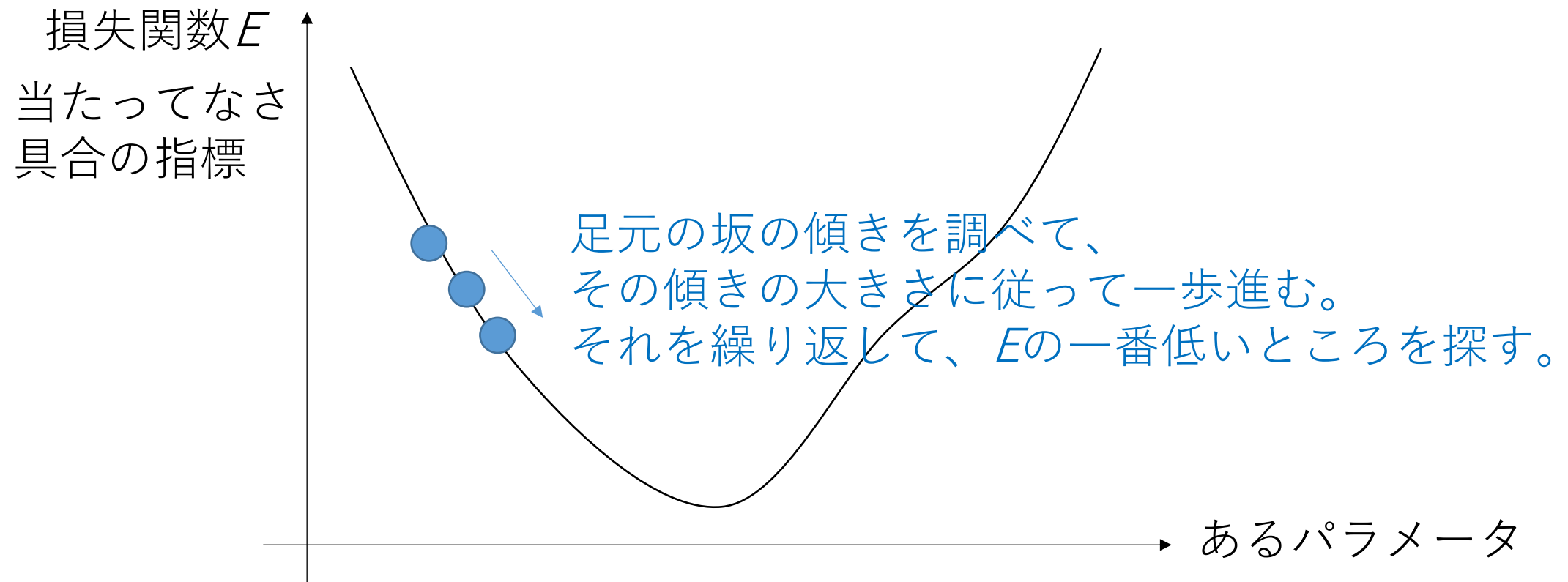
学習の途中段階



# 損失関数～当たってなさ具合の指標

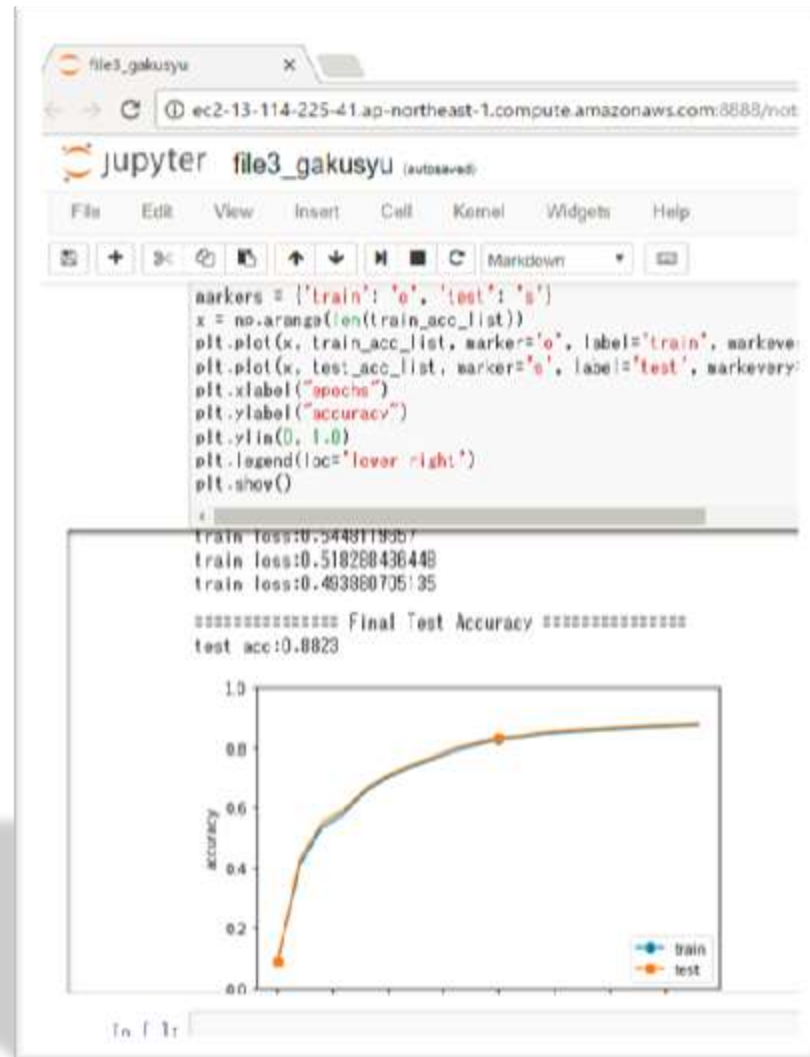


# 損失関数が最小となるパラメータを探す





# 実際の学習の過程を見してみる





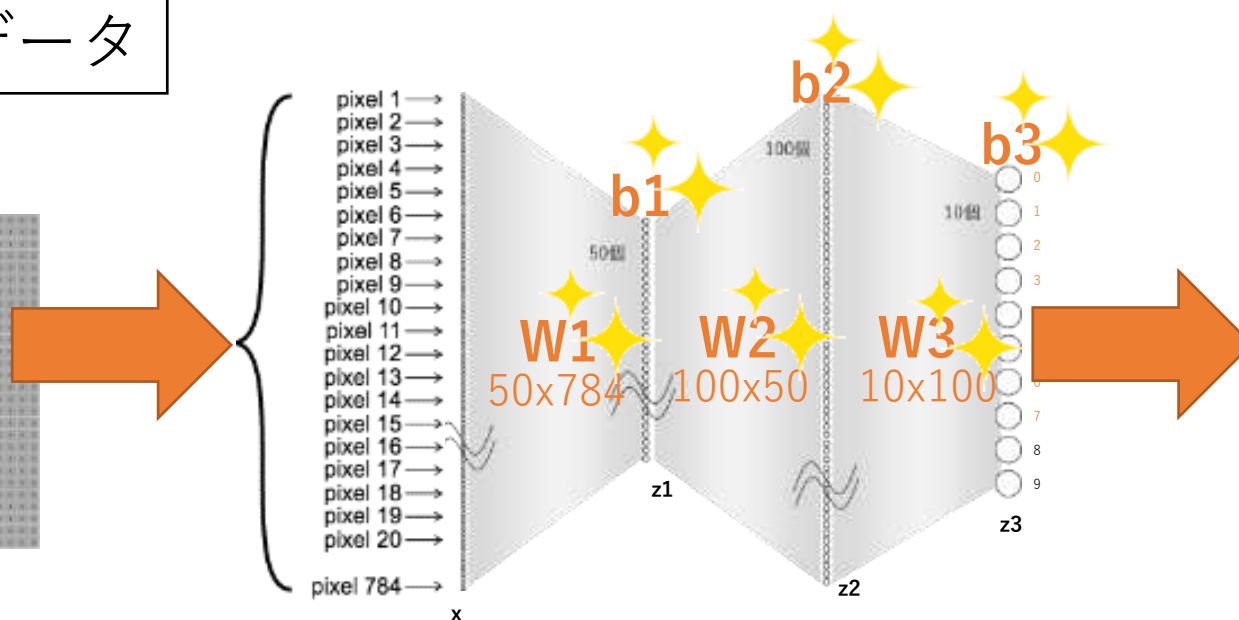
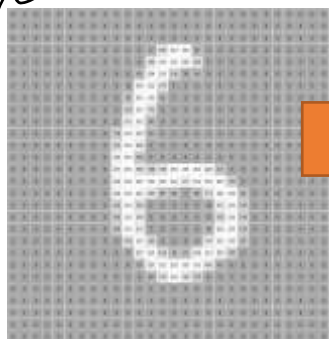
# ディープラーニングの学習 ～ 奥深い世界

- 学習をいかに効率よく行うかが、実際にディープラーニングを使う上で大きな課題
- 学習(コンピューターの数値計算)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

# 学習済のパラメータを使って、文字認識が正しく行われていることを確かめる

テスト用のデータ

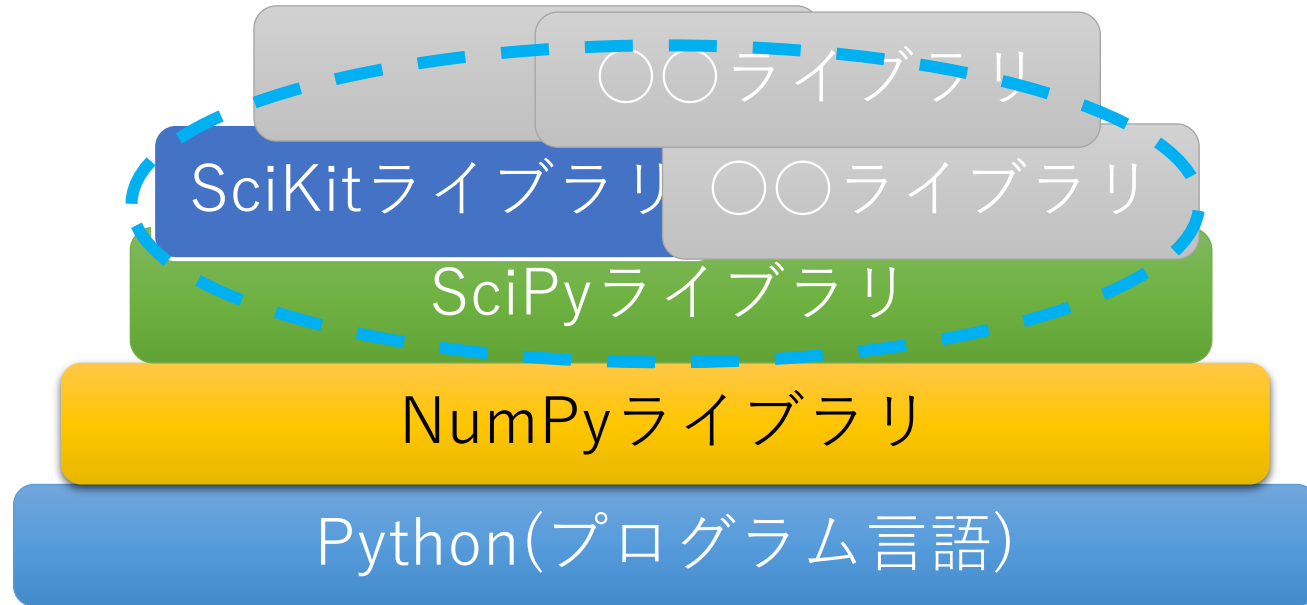
例えば「6」



「6」と認識できるか？  
実際にPythonで動かして  
試してみよう！

ということで、一通りのおさらい

# (ちょっと脱線) 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

# パラメータとハイパーパラメータ

パラメータ

$W1, W2, W3, b1, b2, b3$  ← 計算で求める  
(全部で45,350個の数字)

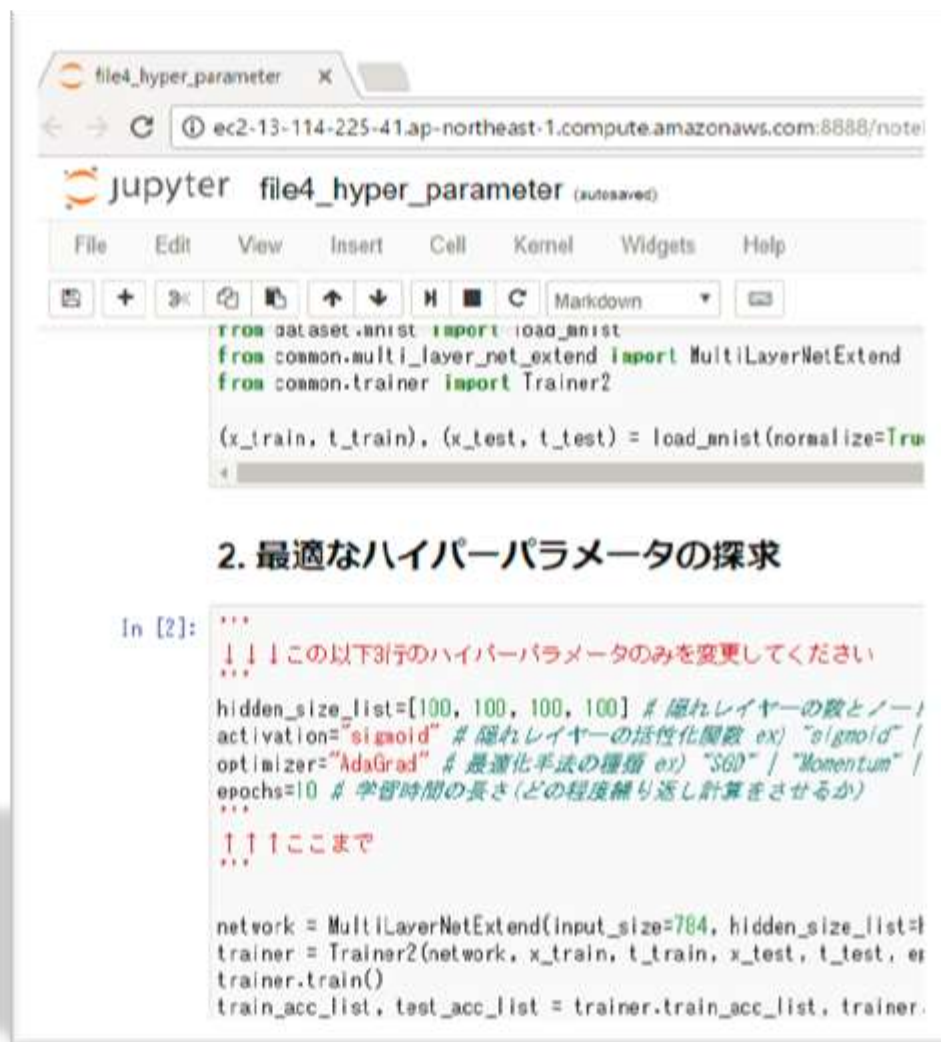
ハイパーパラメータ

- 何層にするか
- 各層のノードの数
- 活性化関数の種類

← 人による設計  
何度も計算させながら試  
行錯誤を繰り返す

より高いコン  
ピュータの計算能  
力が求められる

# ハイパーパラメータ探求を体感



The screenshot shows a Jupyter Notebook window titled "file4\_hyper\_parameter" with a browser address bar showing "ec2-13-114-225-41.ap-northeast-1.compute.amazonaws.com:8888/notebook". The notebook has a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar. The code in the notebook is as follows:

```
from dataset.anist import load_anist
from common.multi_layer_net_extend import MultiLayerNetExtend
from common.trainer import Trainer2

(x_train, t_train), (x_test, t_test) = load_anist(normalize=True)
```

## 2. 最適なハイパーパラメータの探求

In [2]:

```
'''
↓↓↓この以下3行のハイパーパラメータのみを変更してください
'''
hidden_size_list=[100, 100, 100, 100] # 隠れレイヤーの数とノード数
activation="sigmoid" # 隠れレイヤーの活性化関数 ex) "sigmoid" / "tanh" / "ReLU"
optimizer="AdaGrad" # 最適化手法の種類 ex) "SGD" / "Adam" / "Momentum"
epochs=10 # 学習時間の長さ(どの程度繰り返し計算をさせるか)
'''

↑↑↑ここまで

network = MultiLayerNetExtend(input_size=784, hidden_size_list=hidden_size_list,
                               activation=activation, optimizer=optimizer)
trainer = Trainer2(network, x_train, t_train, x_test, t_test, epochs)
trainer.train()
train_acc_list, test_acc_list = trainer.train_acc_list, trainer.test_acc_list
```

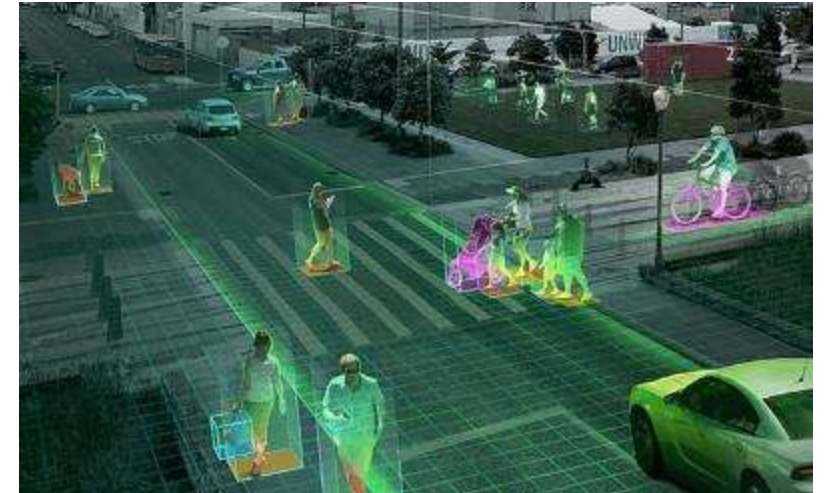
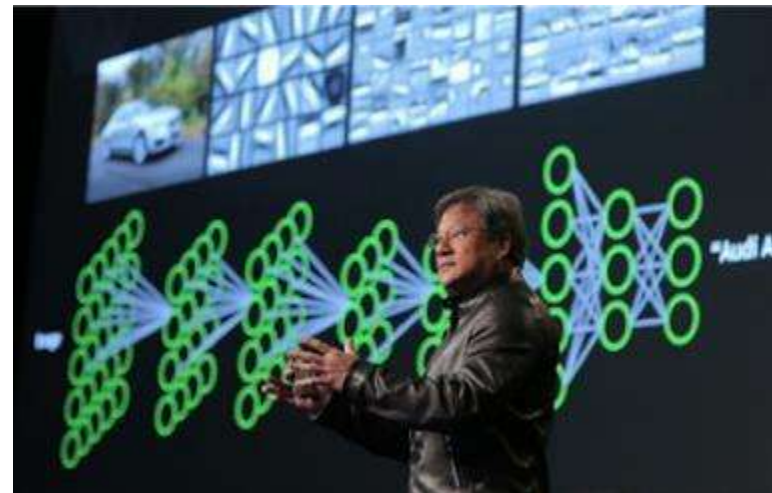
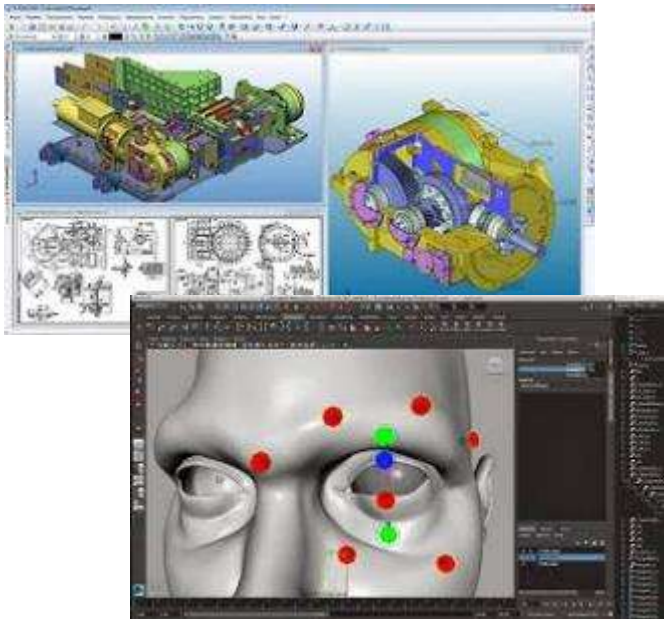
“謎のAI半導体メーカー”☺

# 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。

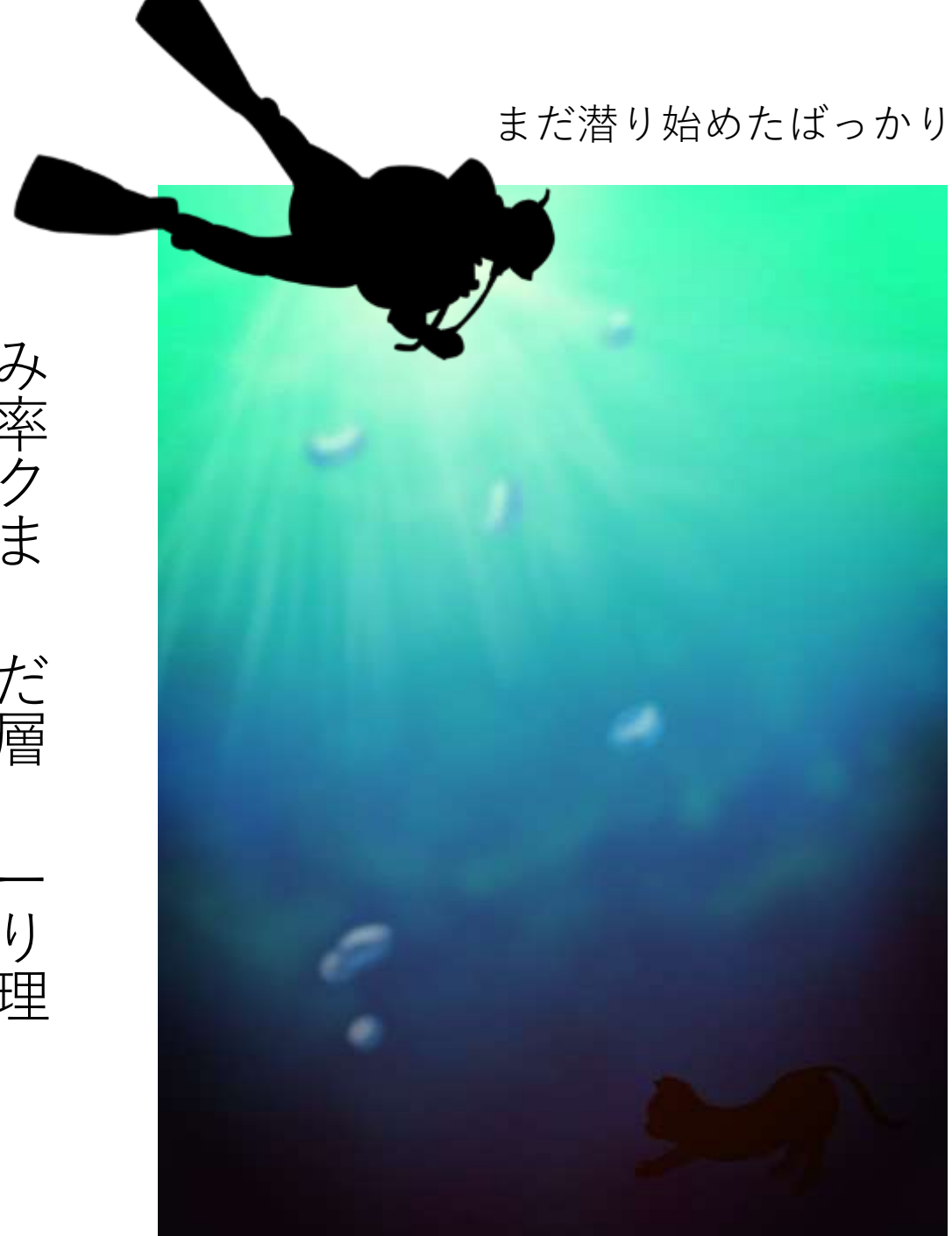




まだ潜り始めたばかり

# 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





# 手書き文字認識

[http://rodrigob.github.io/are we there yet/build/classification datasets results.html](http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html)