

# ニューラルネットワークの 仕組み

金子邦彦





1. 機械学習とは
2. ニューラルネットワーク
3. Python の配列（アレイ）と画像データセット
4. ニューラルネットを作るプログラム
5. ニューラルネットワークの学習を行うプログラム
6. ニューラルネットワークを使うプログラム



# 1 機械学習とは



- 与えられたデータ（教師データ）を使い、  
未知のデータに対しても当てはまる  
パターンや規則を、コンピュータが抽出  
すること

**ニューラルネットワーク**など、機械学習を  
可能にする、多数の技術がある

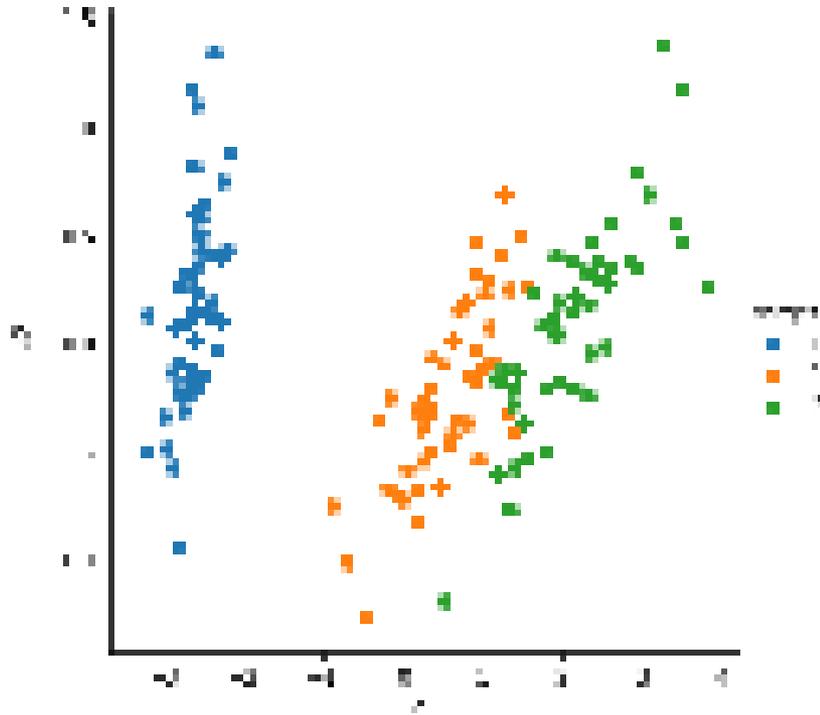
# 機械学習の用途



- **未知のデータ**の分類
- 予測

幅広い応用：画像認識，音声認識，自然言語処理，  
データ分析

# 教師データの例



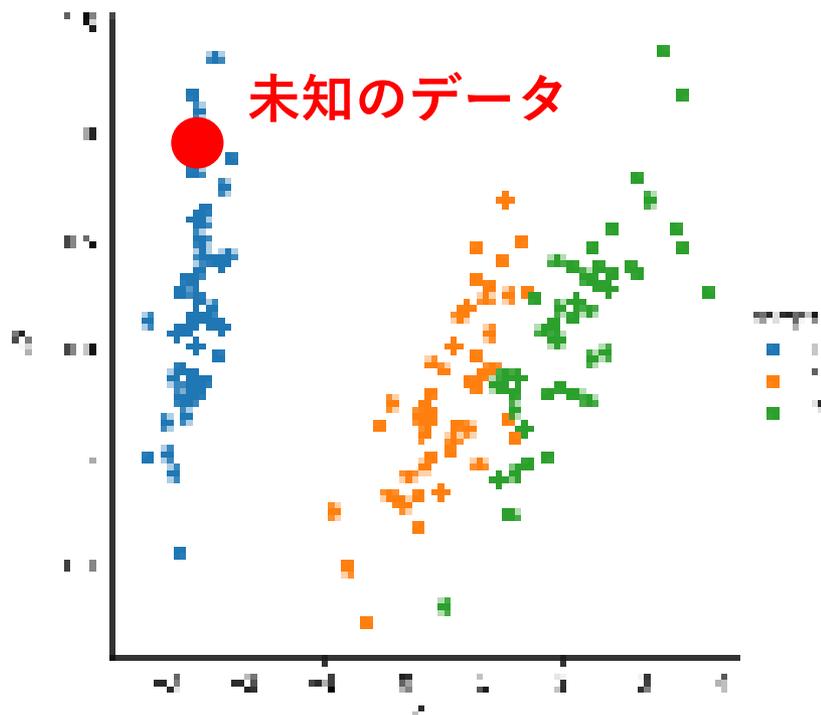
Iris データセット

・ 3種, 150のアヤメの花びらのデータ

※ 右図は, 主成分分析の結果のプロット

- 教師データは, 多数のデータの集まり
- 上の図では, 点1つで, 1つのデータ

# 教師データによる予測



- 新しいデータ (**未知のデータ**) が  
あるとき, 花の種類は何でありそうか  
**教師データ**の利用により, **未知のデータ**についても見通し  
を立てることが可能に



## 2. ニューラルネットワーク

# ニューラルネットワークの進展を助けているもの



- **ニューラルネットワークの技術革新**

基盤技術: Heの初期化, Batch Normalization,

Dropout, CNN, LTSM, GAN系列

モデル: VGG16 など

- **ニューラルネットワークを高速にシミュレーション**で  
きる高性能のコンピュータ

高性能プロセッサ、GPU

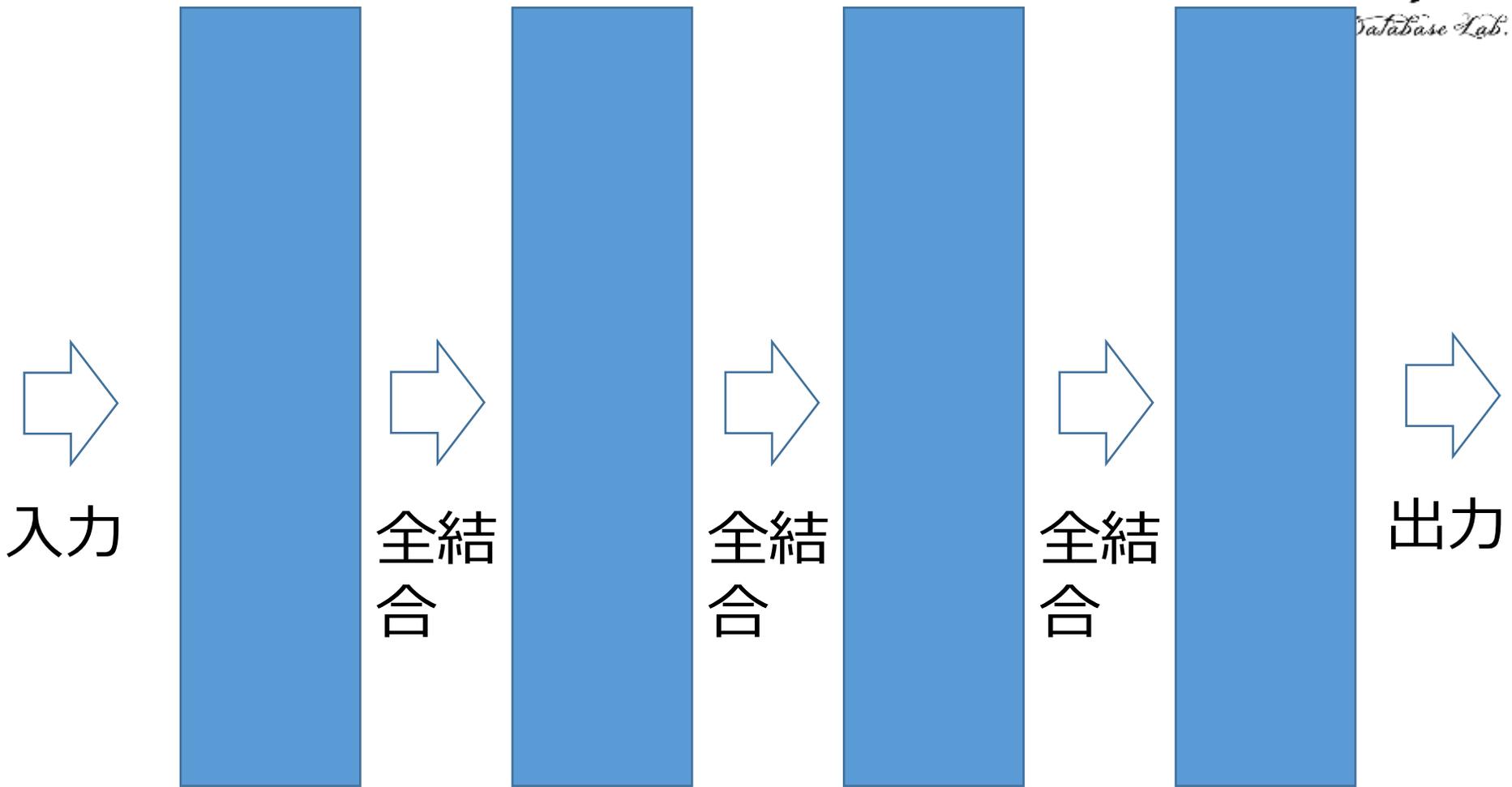
- **ニューラルネットワーク**の学習に役立つ**大量のデータ**

データ計測、データ収集

# 層が直列になっているニューラルネットワーク



Database Lab.

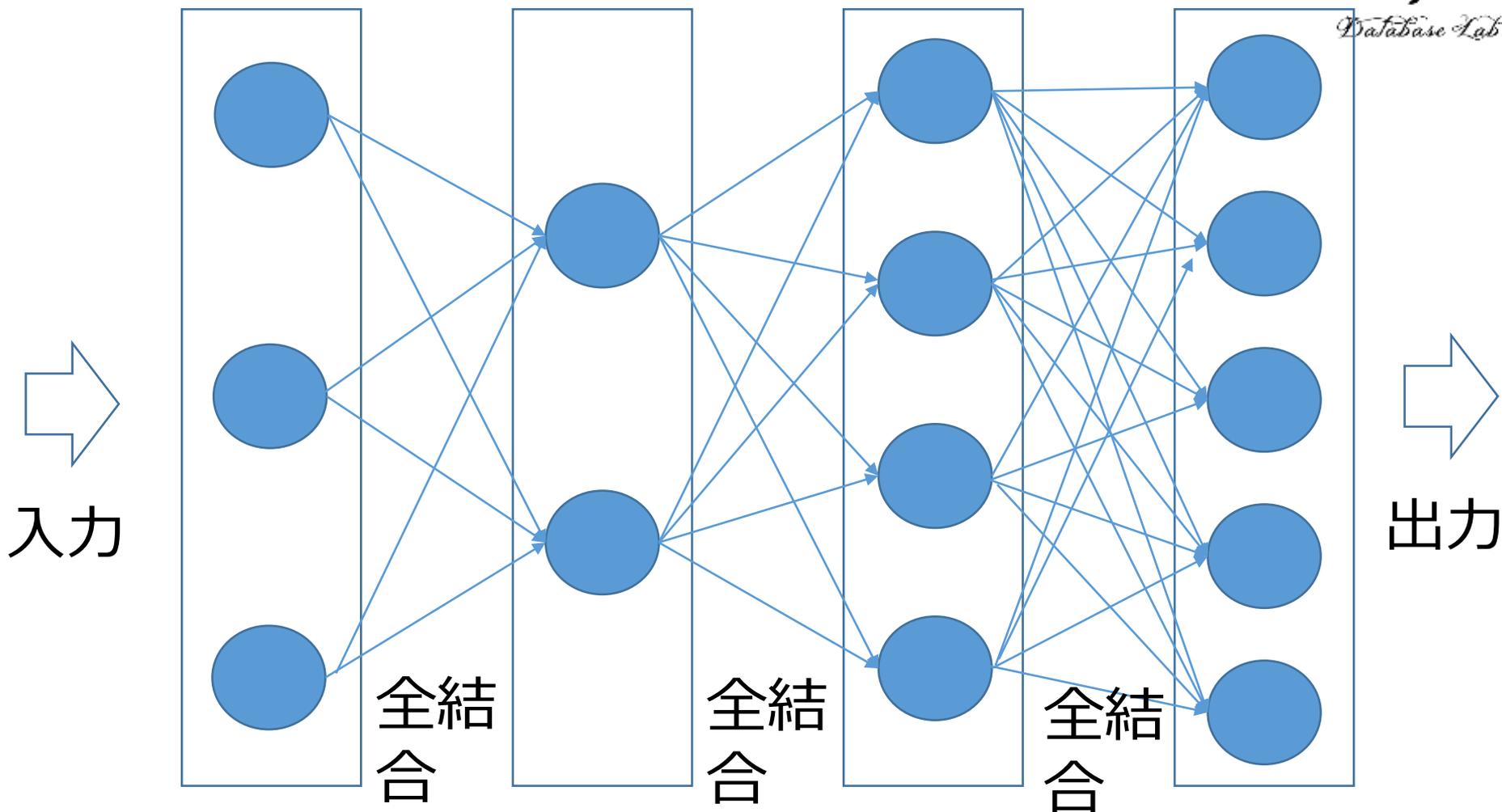


層数が4の場合（総数はいろいろ変わる）

# ユニットと全結合



Database Lab.

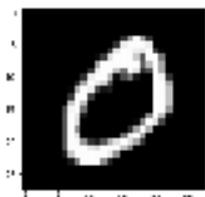


※ 層の中には、ユニットが並ぶ

# ニューラルネットワークによる予測

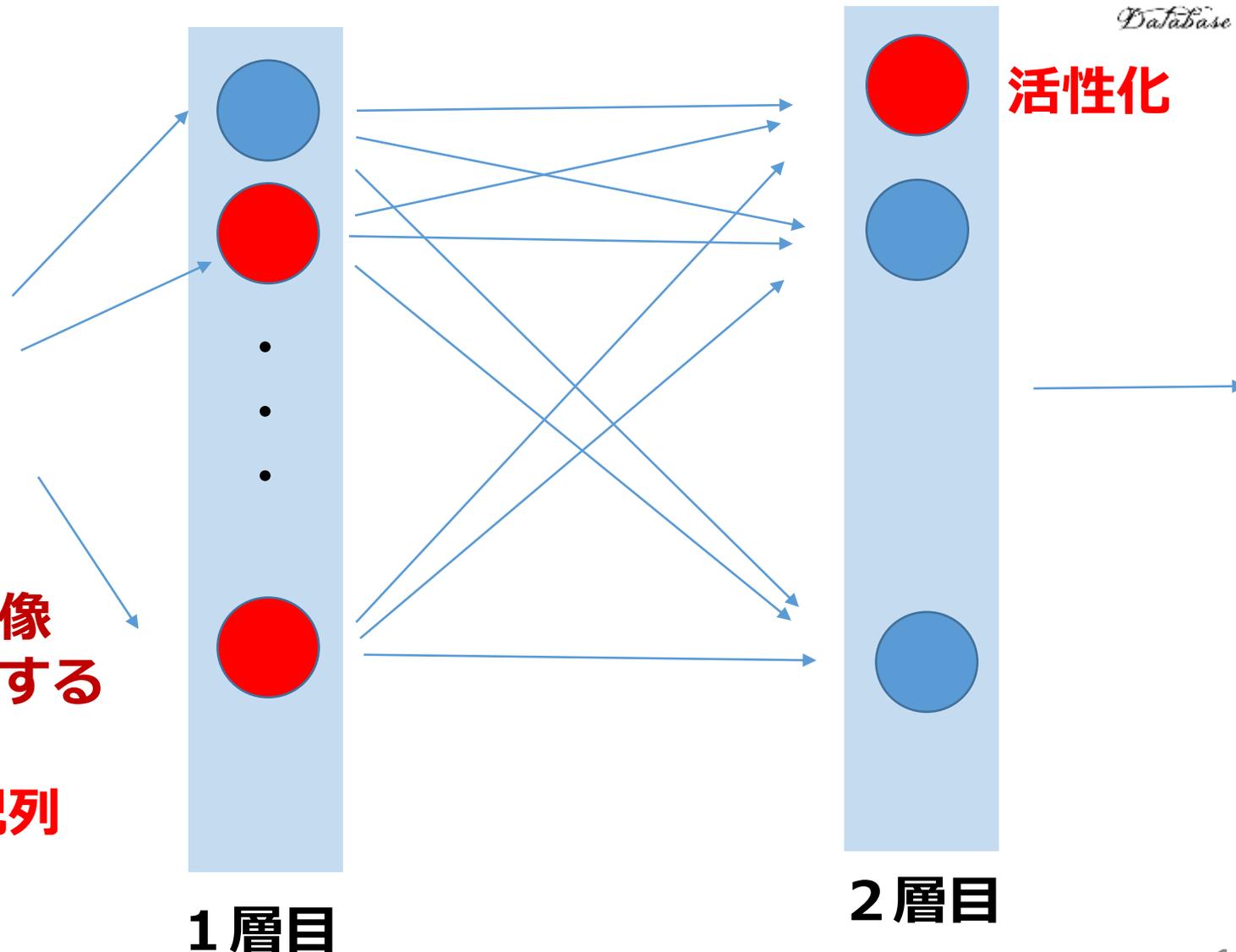


Database Lab.

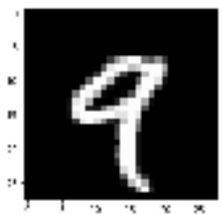


画素数  
784 の  
モノクロ画像  
があったとする

データは配列  
(アレイ)

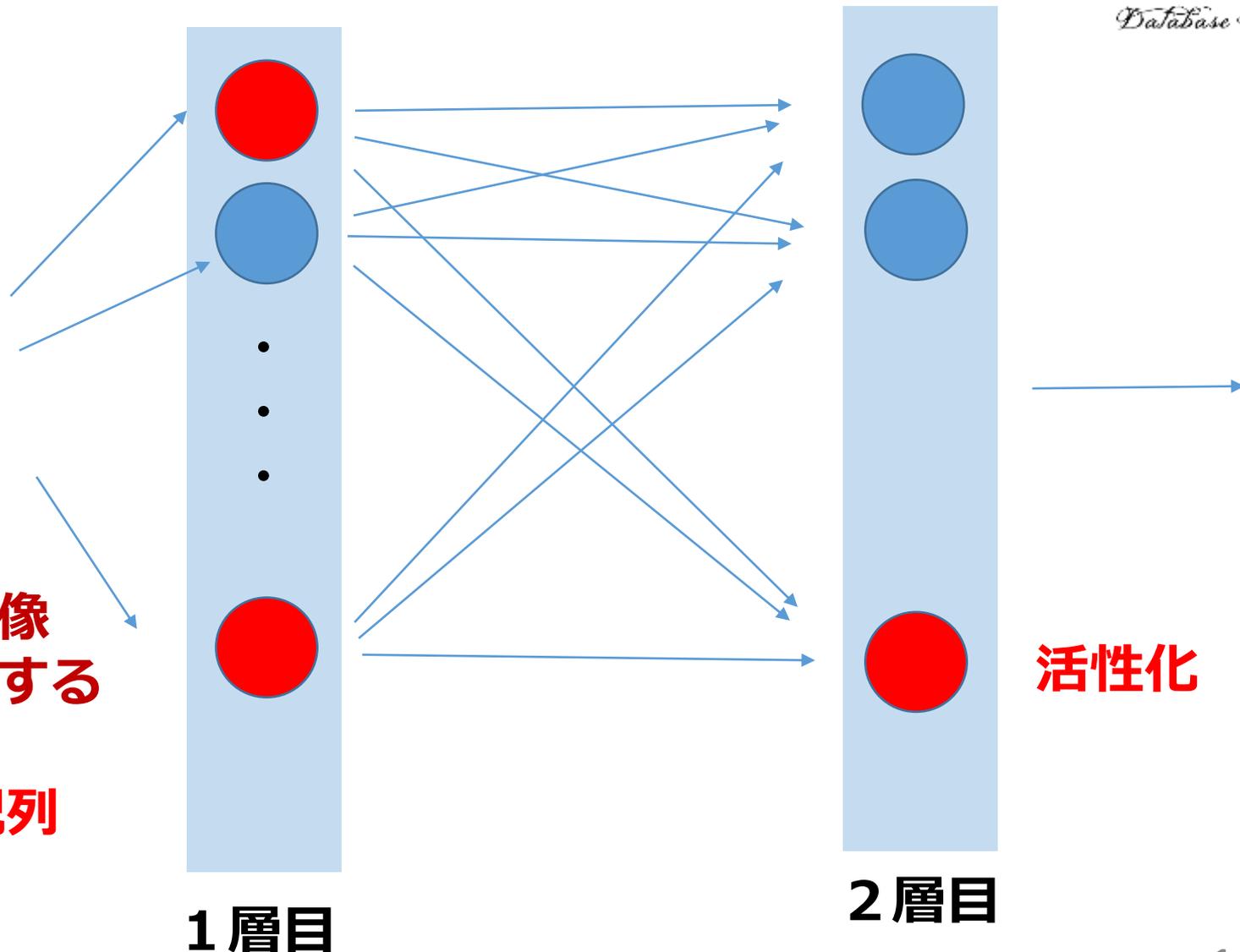


# ニューラルネットワークによる予測

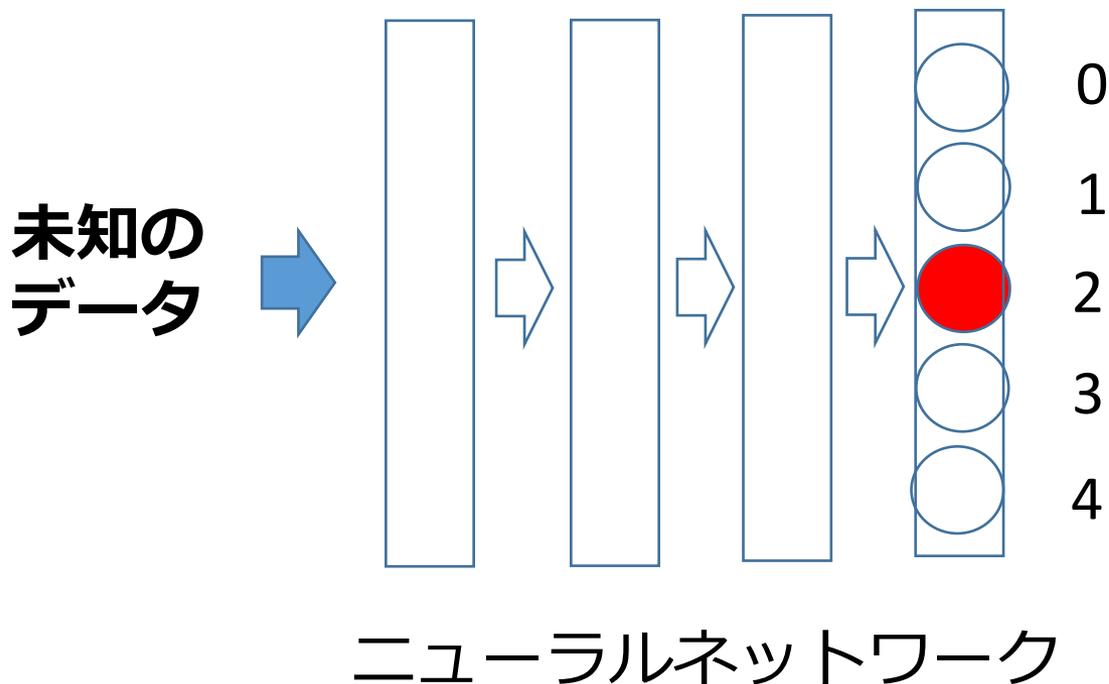


画素数  
784 の  
モノクロ画像  
があったとする

データは配列  
(アレイ)



# ニューラルネットワークによる予測



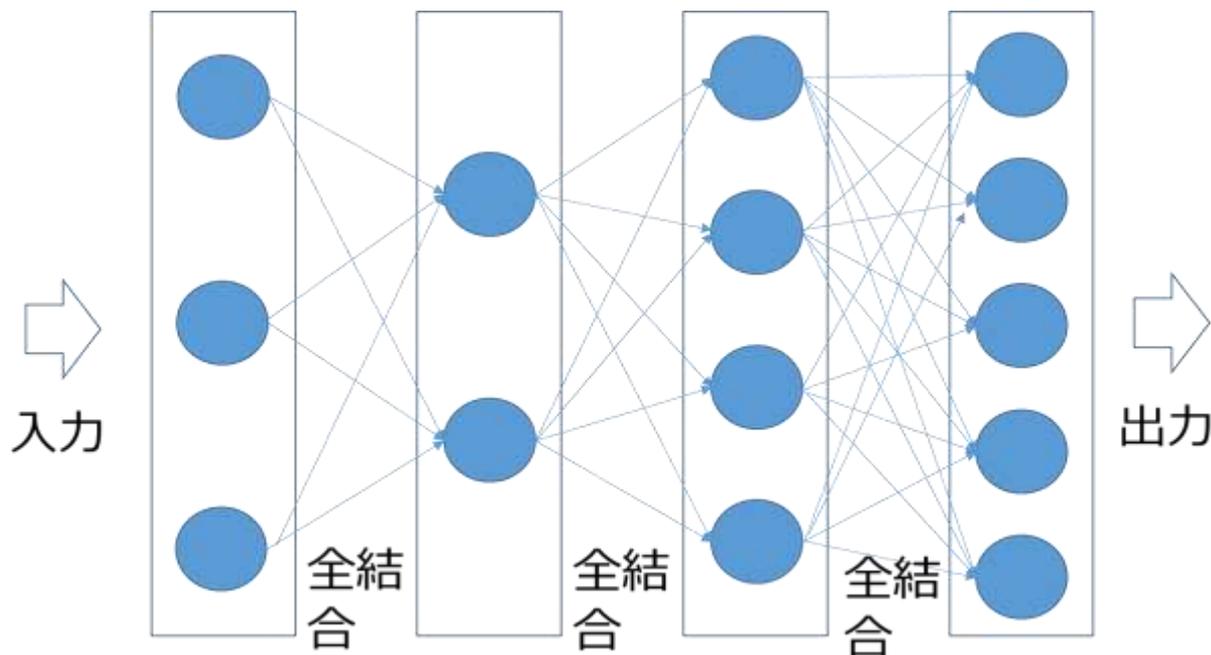
1つが活性化



活性化したユニット  
が「2番」なら、  
「**予測結果は2**」  
の意味

- ・ **ニューラルネットワーク**での予測は、**未知のデータ**を与えて、最終層のユニットを活性化させることで行う

# ニューラルネットワークとは

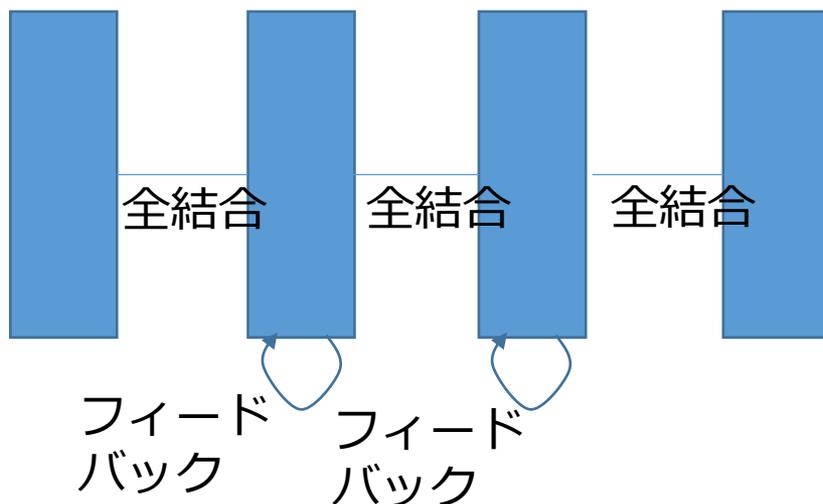


- **ニューラルネットワーク**は、**層**が積み重なっている
- **層**の中には、**ユニット**が並ぶ。
- **ユニット**は、**互いにつながり**、ときには**活性化**する
- **ニューラルネットワーク**での予測は、**未知のデータ**を与えて、最終層の**ユニット**を**活性化**させることで行う

# ニューラルネットワークのバリエーション

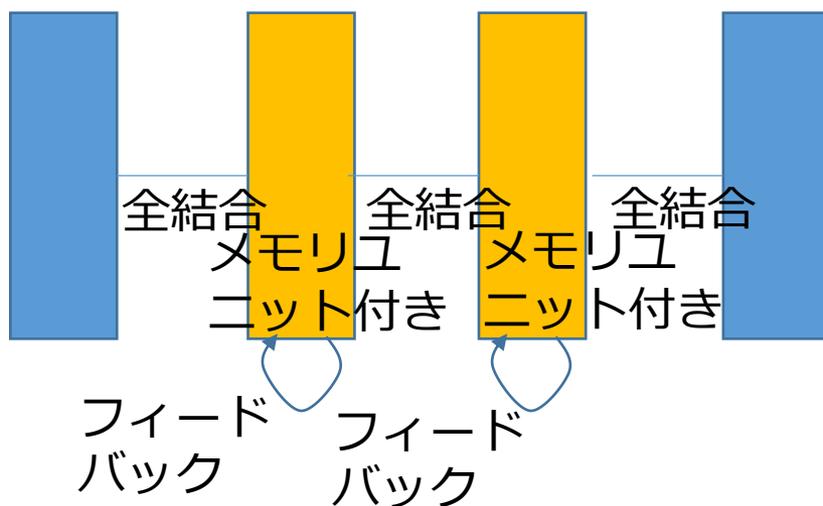


- Recurrent Network



フィード  
バック付き

- LSTM Recurrent Neural Network

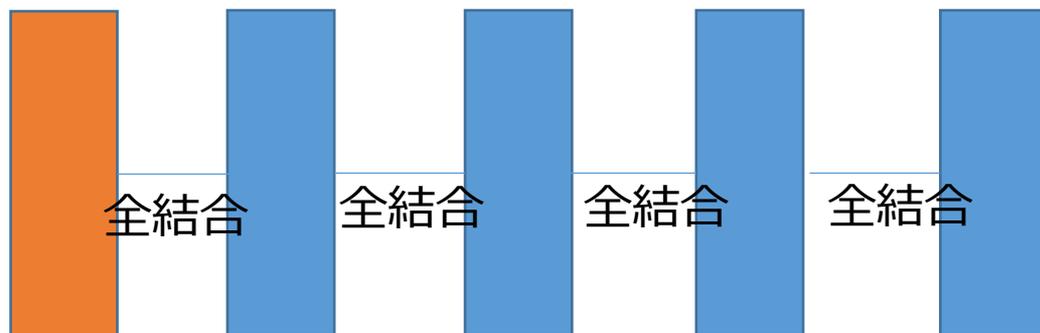


メモリユニット  
付き

# ニューラルネットワークのバリエーション



- CNN



畳みこみ層付き

畳みこみ層



# 3. Python の配列（アレイ）と 画像データセット

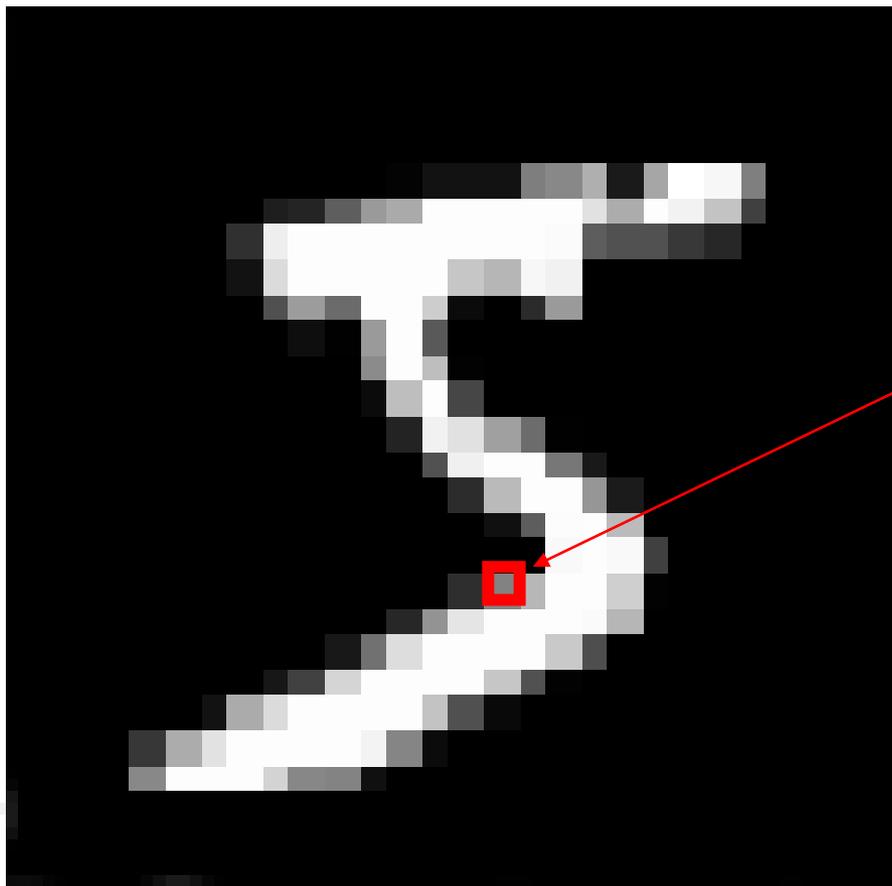
# 配列 (アレイ) とは



0	1	2	3
180	20	250	40

**配列 (アレイ)** とは, データの並びで,  
0 から始まる番号 (添字) が付いている

# 画像と画素



**MNISTデータセット** (手書き文字のデータセットで, 濃淡画像)

画像サイズ: 28 × 28

**画素**



白

**画素値**  
255



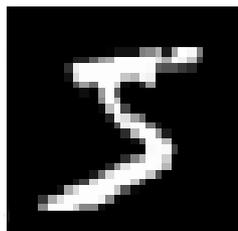
黒

0

**画素値**は, **画素**の明るさに応じた 0 から 255 の数値



# 配列（アレイ）の形と次元



1枚の画像



$28 \times 28$

次元数は **2**

**60000枚の**  
画像データ  
セット



**60000 ×**  
 $28 \times 28$

次元数は **3**

データ

配列（アレイ）の形

次元数

# 画像データセット MNIST



- 濃淡画像 70000枚. 画像サイズは  $28 \times 28$   
うち学習用 60000枚

**配列 (アレイ) の形 :  $60000 \times 28 \times 28$**

テスト用 10000枚

**配列 (アレイ) の形 :  $10000 \times 28 \times 28$**



## 配列（アレイ）の形と次元

サイズ  $28 \times 28$  の濃淡画像は

**配列（アレイ）の形** :  $28 \times 28$

**次元**: 2

サイズ  $28 \times 28$  の濃淡画像 60000 枚の画像  
データセットは

**配列（アレイ）の形** :  $60000 \times 28 \times 28$

**次元**: 3



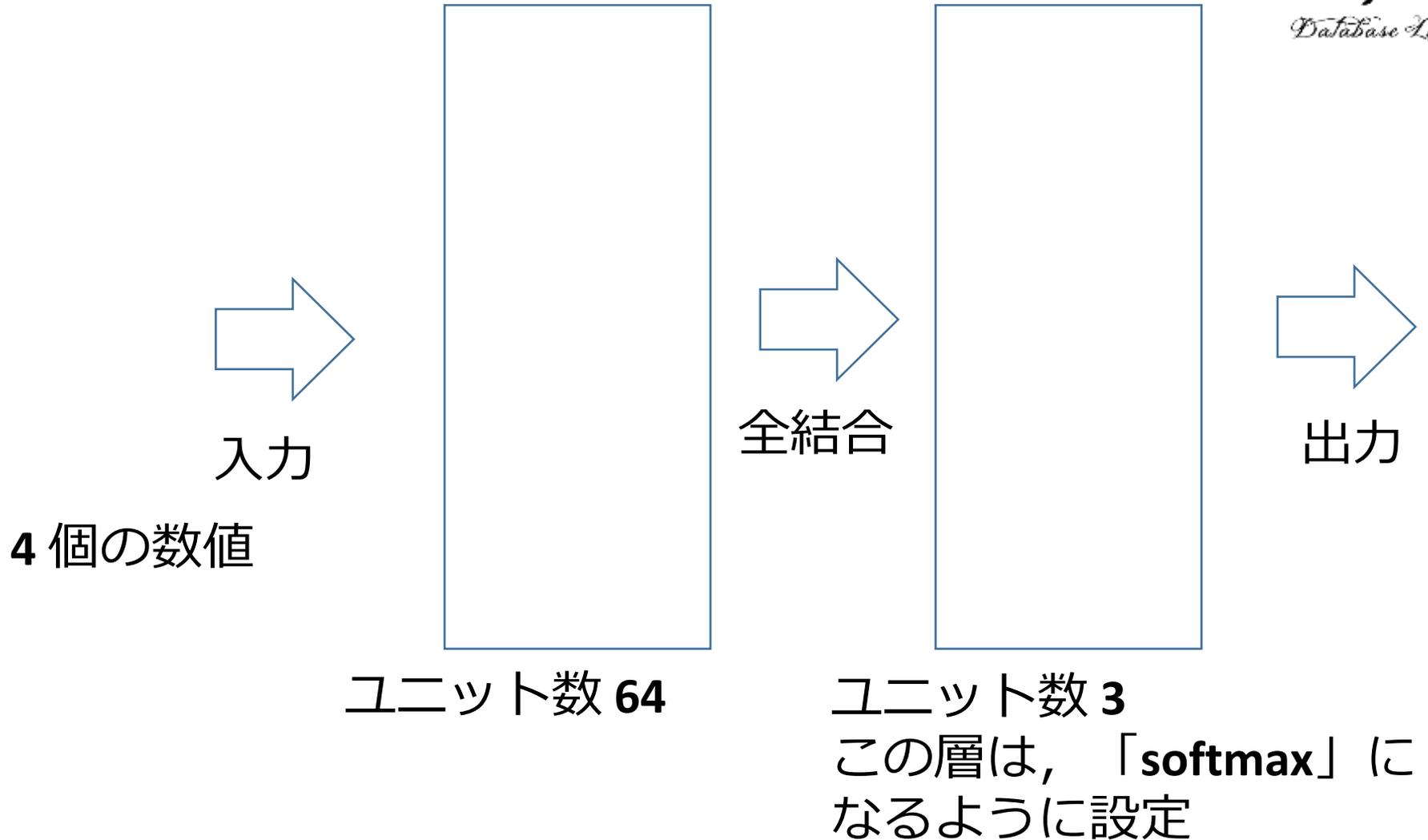
## 4. ニューラルネットワーク を作るプログラム

# Python



- 数多くある**プログラミング言語**の1つ
- さまざまな拡張機能（数百以上）がある
  - numpy 配列計算
  - tensorflow ディープラーニング（深層学習）向けの配列計算
  - Pybrain 機械学習全般
  - Scipy 最適化, 積分, 信号, 画像, 統計, フーリエ変換
  - AIMA
- ※ ディープラーニング（深層学習）は, 多層のニューラルネットワーク, その活用技術のこと
- アイデアをすぐに試したい（数十行以内のプログラム）場合にも向くとされる

# ニューラルネットワークの例

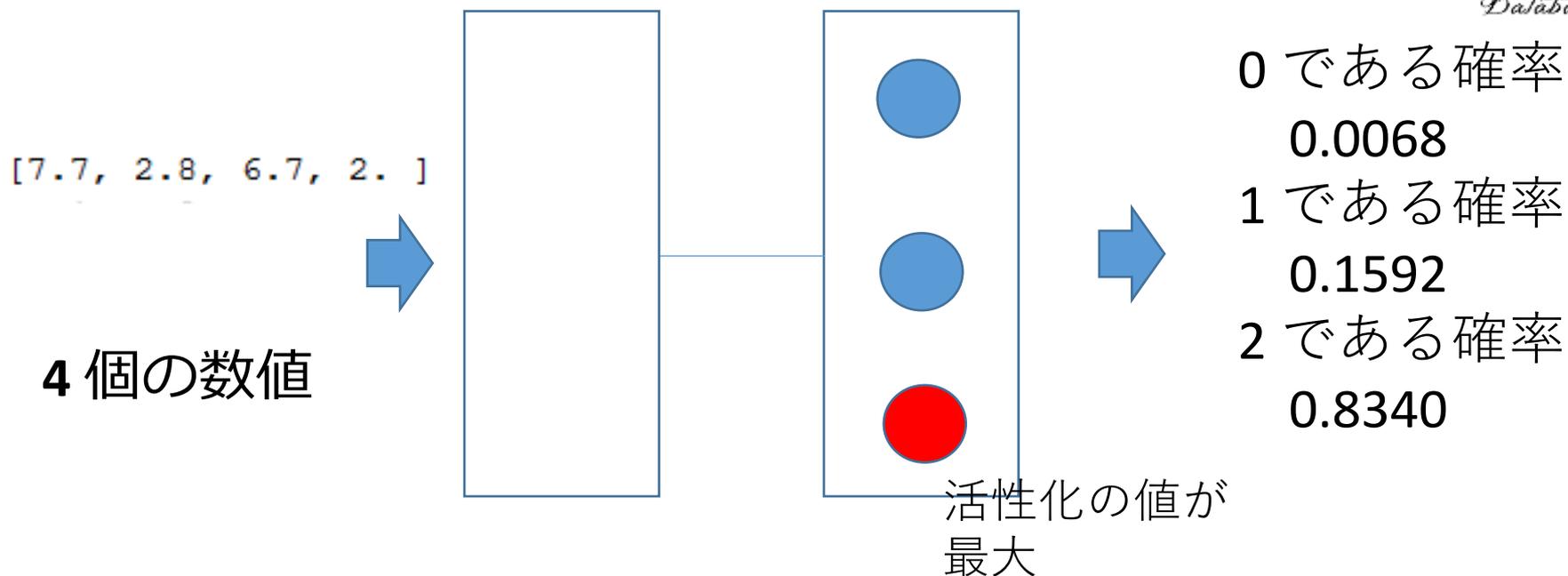


層数 : 2

# ニューラルネットワークの動作イメージ



*Database Lab.*

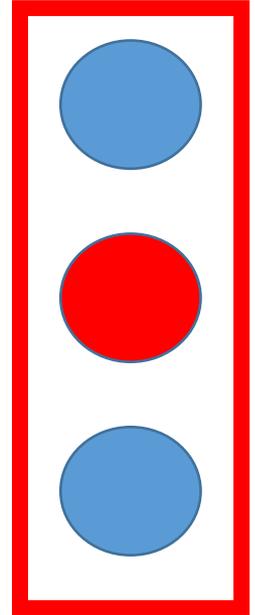


# ソフトマックス (softmax) に設定された層



## ソフトマックス (softmax) に設定したとき

- ・ 各ユニットの活性化の割合は, 0 から 1 の数値
- ・ **最も値が高いものは「活性化している」, それ以外は「活性化してない」と考える**



# ニューラルネットワークを作るプログラム



```
import tensorflow as tf
from tensorflow.keras import layers
```

```
num_classes = 3  ← 3種類に分類
input_dim = 4    ← 入力は4個の数値
```

```
m = tf.keras.Sequential(
    [
        layers.Dense(units=64, input_dim=input_dim, activation='relu'),
        layers.Dense(units=num_classes, activation='softmax'),
    ]
)
```

1層目のユニット数は64

「softmax」に設定

# ニューラルネットワークの作成



プログラムで、次を指定

- 層の数
- 各層のユニットの数
- 各層のユニットの種類



ノートページ

コンピュータ上の作成されるバーチャルなもの



## ■ 資料の Web ページを開く

<https://www.kkaneko.jp/a/ai.html>

## ■ プログラムを、コピー、貼り付け

ニューラルネットワークを作るプログラム

```
import tensorflow as tf
from tensorflow.keras import layers

num_classes = 3
input_dim = 4

m = tf.keras.Sequential([
    layers.Dense(units=64, input_dim=input_dim, activation='relu'),
    layers.Dense(units=num_classes, activation='softmax')])
```

ニューラルネットワークの確認表示

```
print(m.summary())
```

ニューラルネットワークの学習を行うプログラム

```
import numpy as np
```



```
IPython 8.15.0 -- An enhanced Interactive Python.

In [3]: import tensorflow as tf
        from tensorflow.keras import layers

        num_classes = 3
        input_dim = 4

        m = tf.keras.Sequential(
            [
                layers.Dense(units=64, input_dim=input_dim, activation='relu'),
                layers.Dense(units=num_classes, activation='softmax')
            ]
        )

C:\Program Files\Python310\lib\site-packages\numpy\distributor_init.py:30: UserWarning: loaded more than 1 DLL from .libs:
C:\Program Files\Python310\lib\site-packages\numpy\.libs\libopenblas.F85AE2YXYH2IJRDKGGQ3XBKLT43H.gfortran-win_amd64.dll
C:\Program Files\Python310\lib\site-packages\numpy\.libs\libopenblas64_v0.3.21-gcc_10_3_0.dll
warnings.warn("loaded more than 1 DLL from .libs:")

In [2]:
```

コピーしたい範囲をマウスで  
選び、  
マウスの右クリックメニュー  
で「コピー」が便利

spyder の右下の Python コン  
ソールに張り付ける。  
張り付けたあと Enter キー

マウスの右クリックメニュー  
で「Paste」が便利

# ■ ニューラルネットワークの確認表示



```
print(m.summary())
```

```
In [2]: print(m.summary())  
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	320
dense_1 (Dense)	(None, 3)	195

```
=====  
Total params: 515  
Trainable params: 515  
Non-trainable params: 0
```

```
None
```

```
In [3]:
```

ニューラルネットワークのオブジェクト名は **m**

確認表示は `print(m.summary())`

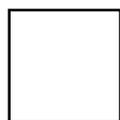


## 5. ニューラルネットワークの学習を行うプログラム

# 画像と画素



画素は、**白と黒の2種類**しかないとする

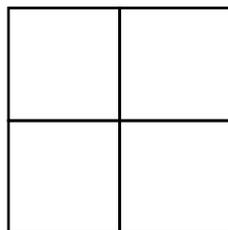


白は 0

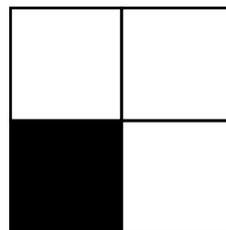


黒は 1, とする

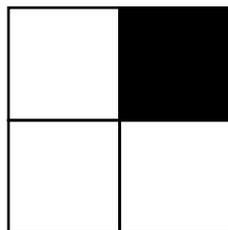
画像のサイズが  $2 \times 2$  のとき



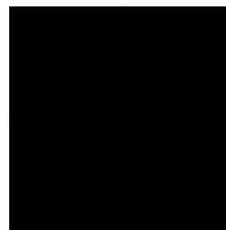
[0, 0, 0, 0]



[0, 0, 1, 0]

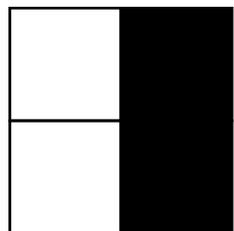


[0, 1, 0, 0]

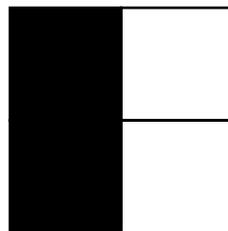


[1, 1, 1, 1]

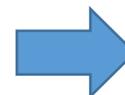
# 学習の例



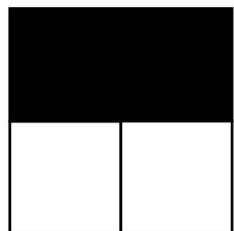
[0, 1, 0, 1]



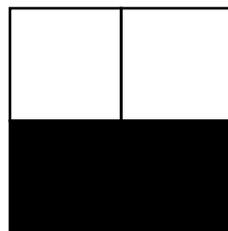
[1, 0, 1, 0]



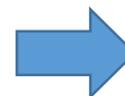
1, を出力させる



[1, 1, 0, 0]

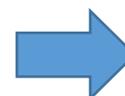


[0, 0, 1, 1]



2 を出力させる

その他



0, を出力させる

# 学習用データセットの例



[[0, 0, 0, 0],	[0,
[0, 0, 0, 1],	0,
[0, 0, 1, 0],	0,
<b>[0, 0, 1, 1],</b>	<b>2,</b>
[0, 1, 0, 0],	0,
<b>[0, 1, 0, 1],</b>	<b>1,</b>
[0, 1, 1, 0],	0,
[0, 1, 1, 1],	0,
[1, 0, 0, 0],	0,
[1, 0, 0, 1],	0,
<b>[1, 0, 1, 0],</b>	<b>1,</b>
[1, 0, 1, 1],	0,
<b>[1, 1, 0, 0],</b>	<b>2,</b>
[1, 1, 0, 1],	0,
[1, 1, 1, 0],	0,
[1, 1, 1, 1]]	0]

どういう**入力**のとき, どういう**出力**が出て欲しいかのデータ

# ニューラルネットワークの学習



学習に使用されるデータセット

- どのような入力するとき, どのような出力が出て欲しいかのデータ
- 形は配列 (アレイ)

# パソコン実習



- さきほど作成したニューラルネットワークは未学習

ニューラルネットワークのオブジェクト名は **m**  
**学習**させるコマンド

```
epochs = 500
```

```
m.compile(loss=tf.keras.losses.categorical_crossentropy,  
          optimizer=tf.keras.optimizers.SGD(lr=0.01,  
          momentum=0.9, nesterov=True))
```

```
m.fit(x, tf.keras.utils.to_categorical(y), epochs=epochs)
```

# ニューラルネットワークの学習

## ■ プログラムを、コピー、貼り付け



ニューラルネットワークの学習を行うプログラム

```
import numpy as np
x = np.array([
    [0, 0, 0, 0],
    [0, 0, 0, 1],
    [0, 0, 1, 0],
    [0, 0, 1, 1],
    [0, 1, 0, 0],
    [0, 1, 0, 1],
    [0, 1, 1, 0],
    [0, 1, 1, 1],
    [1, 0, 0, 0],
    [1, 0, 0, 1],
    [1, 0, 1, 0],
    [1, 0, 1, 1],
    [1, 1, 0, 0],
    [1, 1, 0, 1],
    [1, 1, 1, 0],
    [1, 1, 1, 1]])
y = np.array([
    0,
    0,
    0,
    2,
    0,
    1,
    0,
    0,
    0,
    0,
    0,
    1,
    0,
    0,
    0,
    2,
    0,
    0,
    0,
    0])
epochs = 500

m.compile(loss=tf.keras.losses.categorical_crossentropy,
          optimizer=tf.keras.optimizers.SGD(lr=0.01, momentum=0.9, nesterov=True))
m.fit(x, tf.keras.utils.to_categorical(y), epochs=epochs)
```



```
コンソール 1/A X
[0,
 0,
 0,
 2,
 0,
 1,
 0,
 0,
 0,
 0,
 0,
 1,
 0,
 0,
 0,
 2,
 0,
 0,
 0,
 0])
epochs = 500
m.compile(loss=tf.keras.losses.categorical_crossentropy,
          optimizer=tf.keras.optimizers.SGD(lr=0.01, momentum=0.9, nesterov=True))
m.fit(x, tf.keras.utils.to_categorical(y), epochs=epochs)
```

コピーしたい範囲をマウスで  
選び、  
マウスの右クリックメニュー  
で「コピー」が便利

spyder の右下の Python コン  
ソールに張り付ける。  
張り付けたあと Enter キー

マウスの右クリックメニュー  
で「Paste」が便利



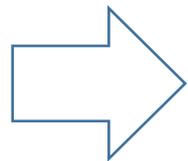
## ■ 学習の終了の確認

```
Epoch 493/500  
1/1 [=====] - 0s 2ms/step - loss: 0.0974  
Epoch 494/500  
1/1 [=====] - 0s 9ms/step - loss: 0.0970  
Epoch 495/500  
1/1 [=====] - 0s 1ms/step - loss: 0.0965  
Epoch 496/500  
1/1 [=====] - 0s 2ms/step - loss: 0.0960  
Epoch 497/500  
1/1 [=====] - 0s 2ms/step - loss: 0.0955  
Epoch 498/500  
1/1 [=====] - 0s 2ms/step - loss: 0.0951  
Epoch 499/500  
1/1 [=====] - 0s 3ms/step - loss: 0.0946  
Epoch 500/500  
1/1 [=====] - 0s 2ms/step - loss: 0.0942  
Out[3]: <keras.callbacks.History at 0x212441db700>  
In [4]:
```

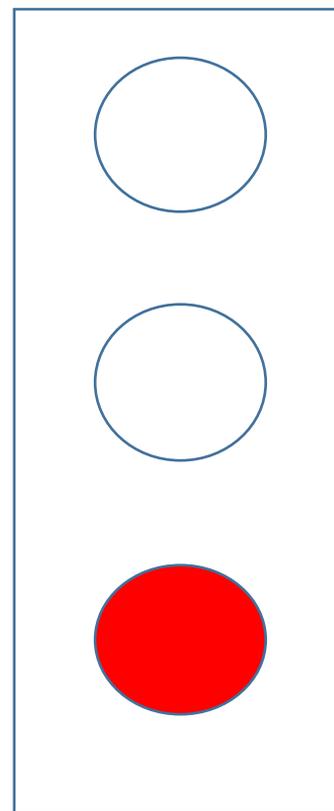
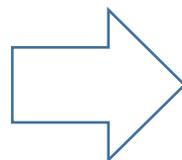


## 6. ニューラルネットワーク を使うプログラム

# ニューラルネットの使用例



入力  
[0, 1, 0, 1]  
4 個の数値



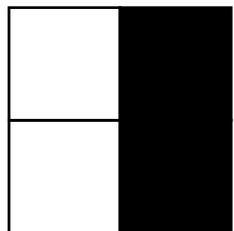
ユニット数 3

0.14816035

0.02995965

0.82188004

出力は、  
各ユニットの  
活性化の度合  
い



# パソコン実習



## ■ プログラムを, コピー, 貼り付け

学習では, 乱数が使用されるので, 各自, 違った値が表示されることに注意.

```
ニューラルネットワークを使ってみる
m.predict( np.array([[0, 1, 0, 1]]) )
第1層と第2層の間の結合の重みを表示
m.get_weights() [2]
```



```
In [4]: m.predict( np.array([[0, 1, 0, 1]]) )
1/1 [=====] - 0s 46ms/step
Out[4]: array([[0.14818741, 0.8310505 , 0.02076209]], dtype=float32)
In [5]:
```

コピーしたい範囲をマウスで  
選び,  
マウスの右クリックメニュー  
で「コピー」が便利

spyder の右下の Python コン  
ソールに張り付ける。  
張り付けたあと **Enter キー**

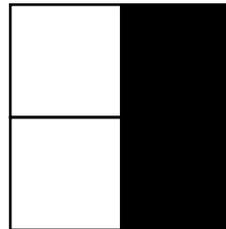
マウスの**右クリックメニュー**  
で「Paste」が便利

# 動作画面の例



```
In [4]: m.predict( np.array([[0, 1, 0, 1]]) )  
1/1 [=====] - 0s 46ms/step  
Out[4]: array([[0.14818741, 0.8310505 , 0.02076209]], dtype=float32)  
  
In [5]:
```

入力  
[0, 1, 0, 1]  
4 個の数値



出力 [0.14818741, 0.8310505, 0.02076209]  
3 個の数値

0 である確率 0.14818741  
1 である確率 0.8310505  
2 である確率 0.02076209



# 結合の重みの表示

## プログラムを、コピー、貼り付け

ニューラルネットワークを使ってみる

```
m.predict( np.array([[0, 1, 0, 1]]) )
```

第1層と第2層の間の結合の重みを表示

```
m.get_weights()[2]
```

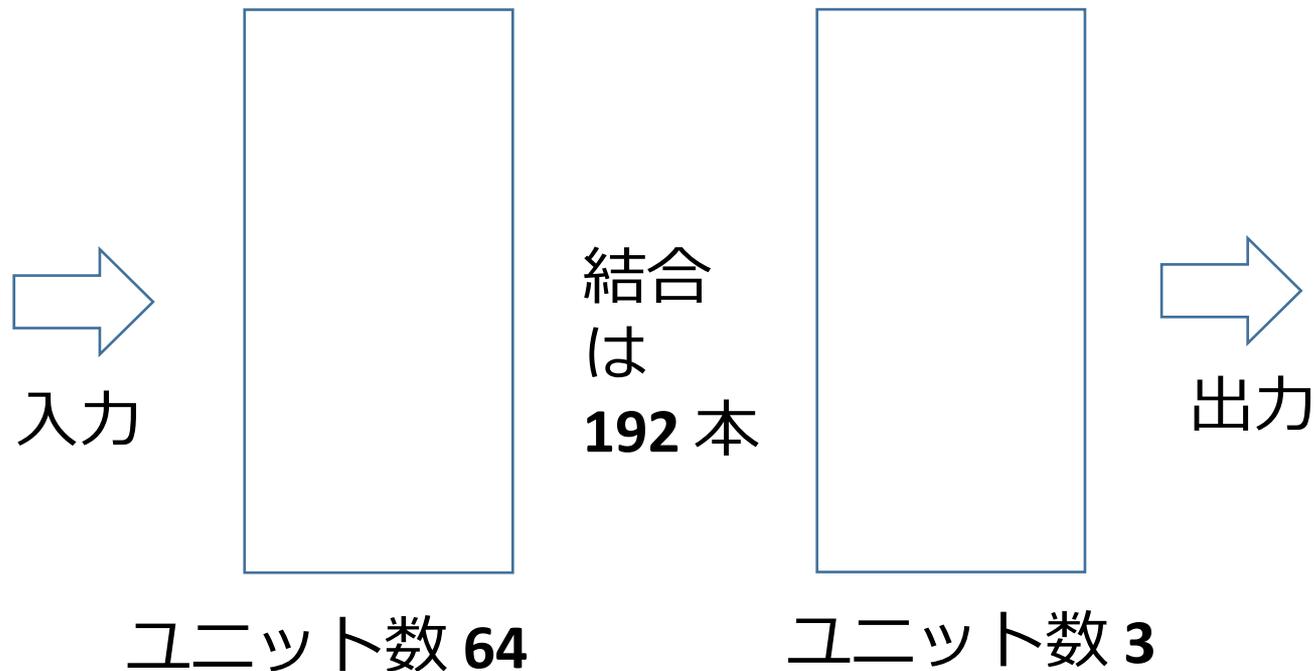


```
[ 0.2632763 , -0.2119094 , -0.16260003 ],  
[ 0.21948093, -0.44927177, -0.113536  ],  
[-0.7334215 , 0.21901026, 0.13347134],  
[ 0.04324561, 0.22607815, -0.21760374],  
[ 0.35473374, 0.09510002, 0.02070638],  
[-0.62371784, 0.08633223, 0.473836  ],  
[-0.39971954, 0.06928192, 0.15764333],  
[ 0.40196273, 0.09825644, -0.5125189  ],  
[-0.05283006, -0.23844175, -0.2908431  ],  
[ 0.1284018 , 0.05880822, -0.27347878],  
[ 0.4538742 , -0.13850422, 0.19888006],  
[ 0.57820505, -0.3302247 , -0.54299176],  
[-0.26995152, 0.00275656, 0.61354136],  
[ 0.01969511, 0.03731089, 0.21472731],  
[-0.05025962, 0.19791228, 0.19244534],  
[ 0.23354772, -0.09778651, -0.21798785],  
[ 0.29553917, 0.14062381, 0.2199938  ],  
[ 0.1205838 , -0.14177398, -0.05534214],  
[ 0.22865112, -0.39954153, -0.3722119  ],  
[-0.00908204, -0.06363981, -0.30074006],  
[ 0.76013106, -0.5798792 , -0.55995315],  
[ 0.07786424, -0.34049904, 0.08216581],  
[ 0.2970195 , -0.39714637, -0.3351546  ],  
[-0.04259956, 0.4394762 , -0.21067457],  
[ 0.35672054, 0.06005513, -0.38829863]], dtype=float32)
```

In [6]:

コピーしたい範囲をマウスで選び、  
マウスの右クリックメニュー  
で「コピー」が便利

# ニューラルネットワークの例



64個のユニットと, 3個のユニットとの結合は 192本

# 結合の重み



- ユニット間の結合の強さを「**結合の重み**」という.
- 学習では, **結合の重み**が変化する

# ニューラルネットワークの使用



- ・学習により, ニューラルネットワークの結合の重みが変化

- ・学習が済んだ後,  
新しい入力を与えて, 出力を見る