dn-1. ニューラルネットワーク

（AI演習の入門）

URL: https://www.kkaneko.jp/ai/dn/index.html

金子邦彦

アウトライン

1. **深層学習**
2. **ニューラルネットワーク**
3. **活性化，伝搬**
4. **ニューラルネットワーク**での**学習**
5. **ニューラルネットワーク**を用いた**分類**
6. **画像**と**画素**
7. **ニューラルネットワーク**の**作成**
8. **学習不足**と**過学習**
9. **ニューラルネットワーク**の**利用の流れ**

- ニューラルネットワークがどういうものか知りたい

- 実際に動かしたい．研究してみたい

（この資料により，実際に動かしてみることも可能）

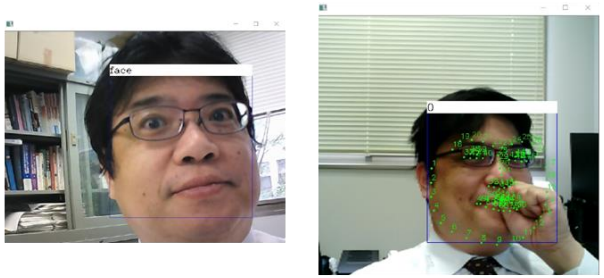
1. 深層学習

人工知能

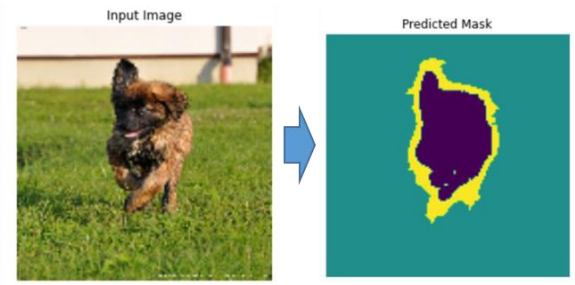
人工知能は，コンピュータが，知的な能力を持つこと

- 知能
- 知識
- 学習

さまざまな応用



顔検知、顔識別



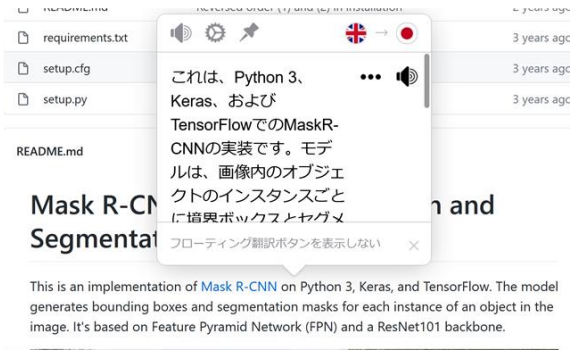
画像のセグメンテーション

金子の顔

有名人の声，表情，語り

金子がその有名人そっくりで語りだす

合成



Webブラウザで翻訳を行う
Mate Translate (Web ブラウザ Firefox のアドオン)

いまの人工知能 (AI) で、できること

- 知識表現，知識の処理
- 人間の言葉に関する処理（自然言語処理）
- 認識や推論
  - 分類
  - 音声認識　（人の声を「文字」化する）
  - 物体検知，物体識別　（画像の中から、「もの」を見つける）
  - 顔認識
- 合成
  - 創作
  - 欠損の補充
  - 翻訳

種々の技術がオンラインで公開されており，
自分のパソコンで試すこともできるように.

人工知能の種類

人工知能

機械学習

学習による上達の能力を
持つ人工知能

知的な IT システム

ルールや知識を**人間が書いた**
人工知能

機械学習のメリット，デメリット

機械学習

学習による上達の能力を
持つ人工知能

メリット

「ルールや知識を，人間がプログラムで書かねばならないことの限界を突破

デメリット

学習不足，過学習，などの注意点がある．完璧に学習が成功するわけではない．

知的な IT システム

ルールや知識を**人間が書いた**
人工知能

深層学習（ディープラーニング）

【できること】

- **学習**による**上達の能力**（**機械学習**）

  → 仕組みはシンプル．最適化（**誤差を自動で最小化**）

- **さまざまな応用**：人間の言葉に関する処理，認識や推論（音声，画像，顔，人体，３次元など），合成，翻訳，欠損の補間など

【できないこと】

- 知識表現，人間が**目に見える形**での知識の獲得

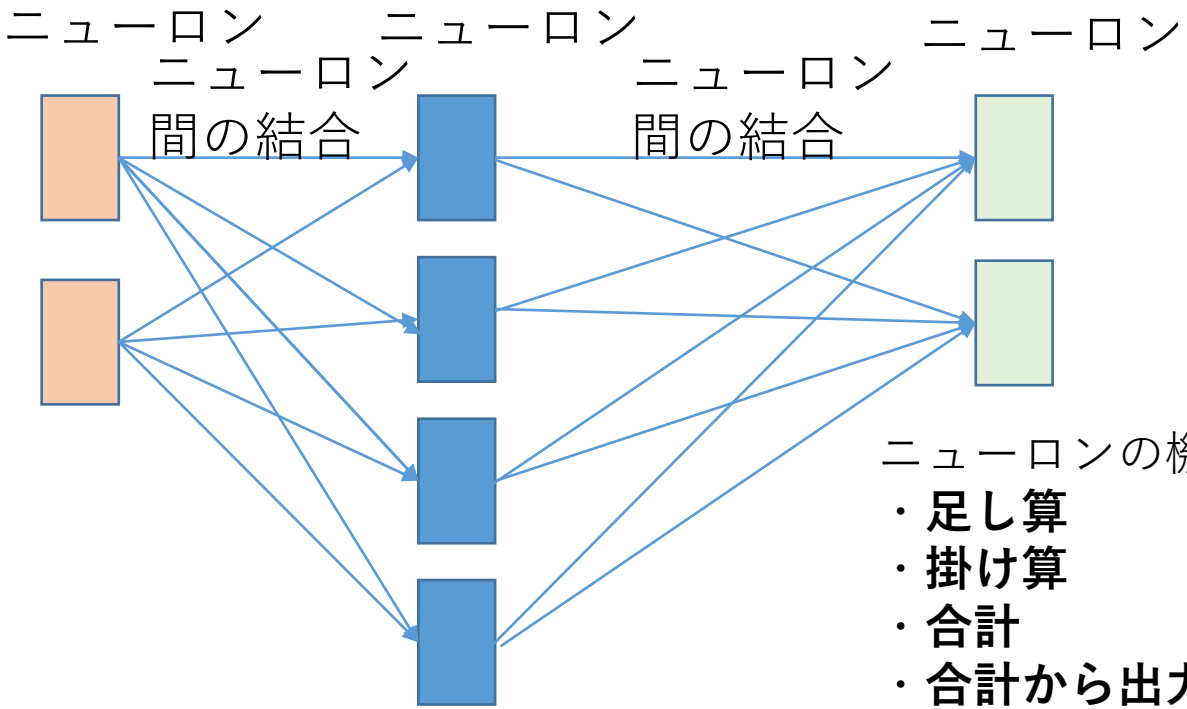
「なぜ，**ディープラーニングが正しく動くのか**」は，

データによる検証が必要

2. ニューラルネットワーク

深層学習とニューラルネットワーク

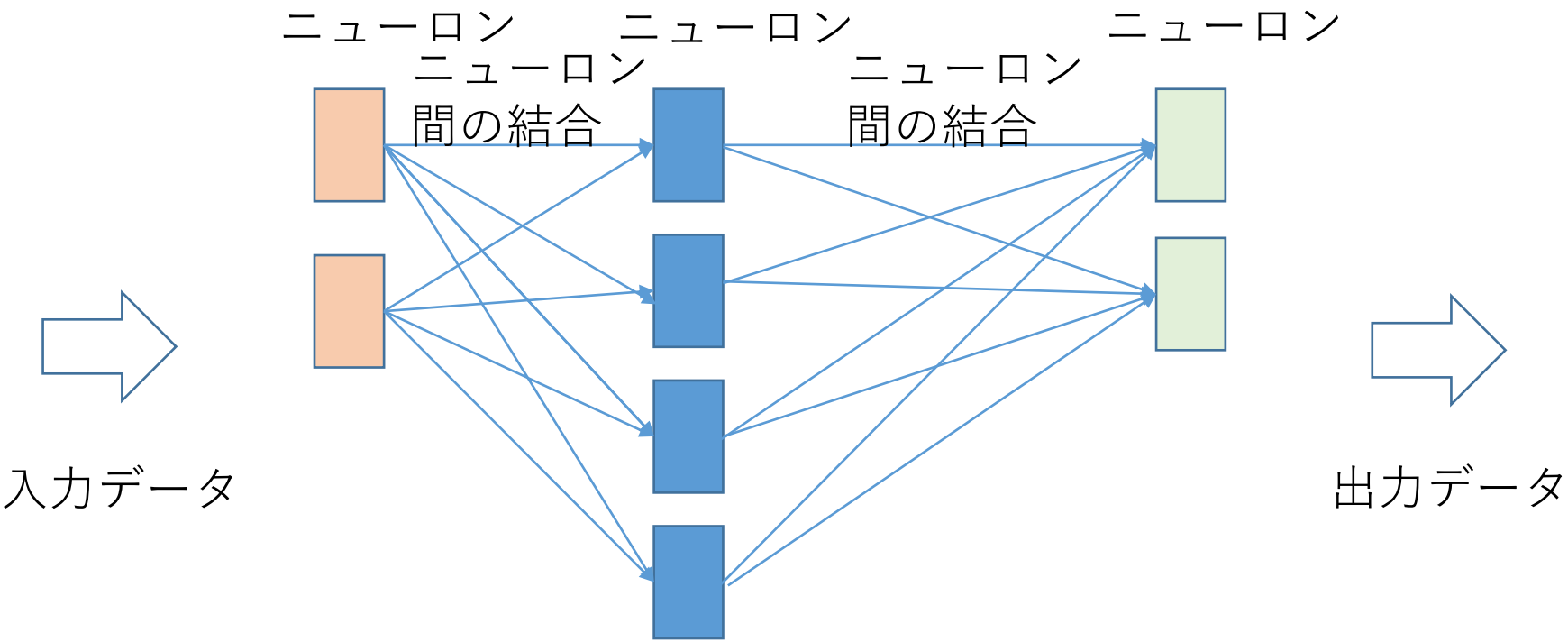
ニューラルネットワーク（ニューロンのネットワーク）は，
深層学習（**ディープラーニング**）の能力を持つ



ニューロンの機能
- **足し算**
- **掛け算**
- **合計**
- **合計から出力値を算出**
  （活性化関数）
- メモリの機能も持つことも

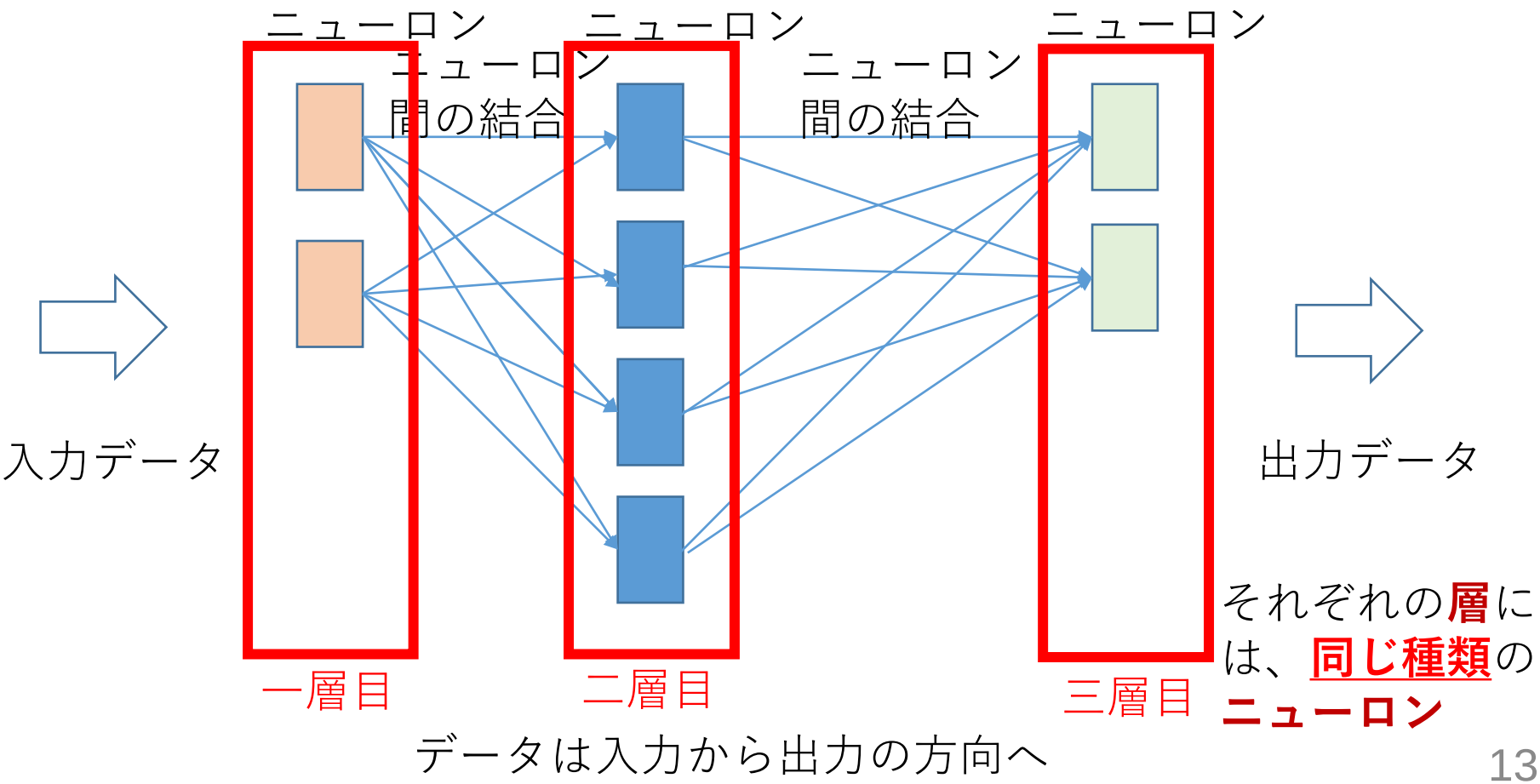
ニューラルネットワークは，
コンピュータでシミュレーション可能

層が直列になっているニューラルネットワーク



データは入力から出力の方向へ

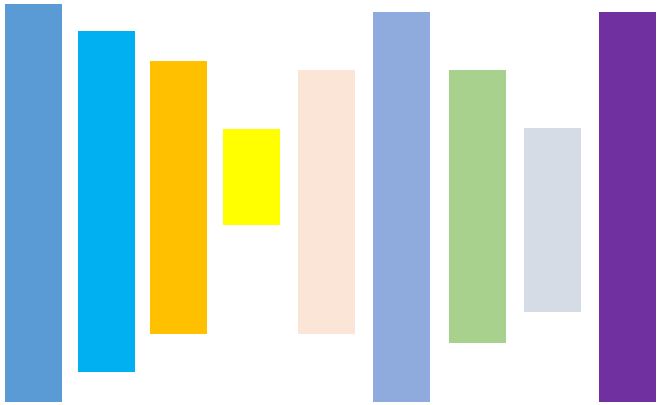
層が直列になっているニューラルネットワーク



深層学習

- **層の数が多い（多層の）ニューラルネットワーク**は，**深層学習**の機能をもつ
- **深層学習**のことを**ディープラーニング**ともいう

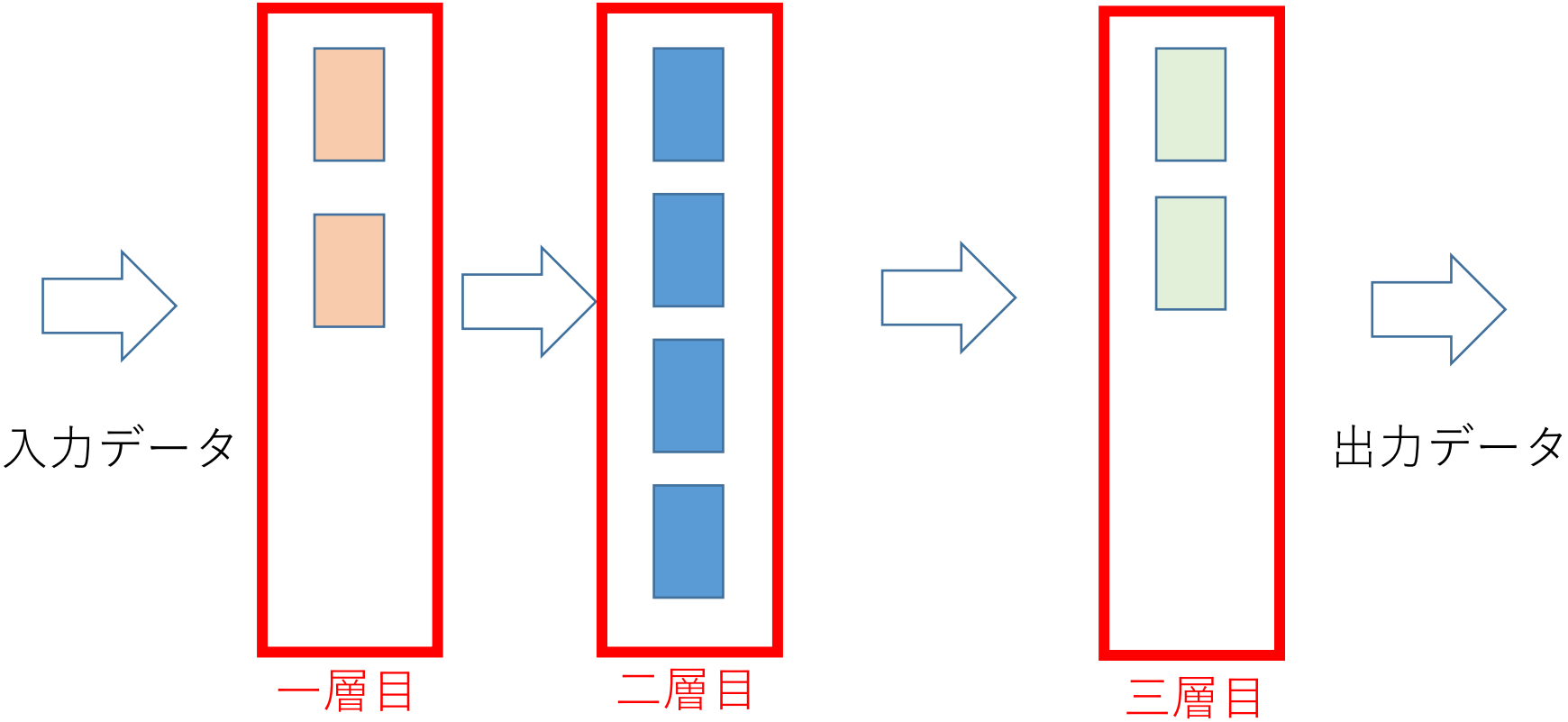
層数が少ない



層数が多い（＝多層）

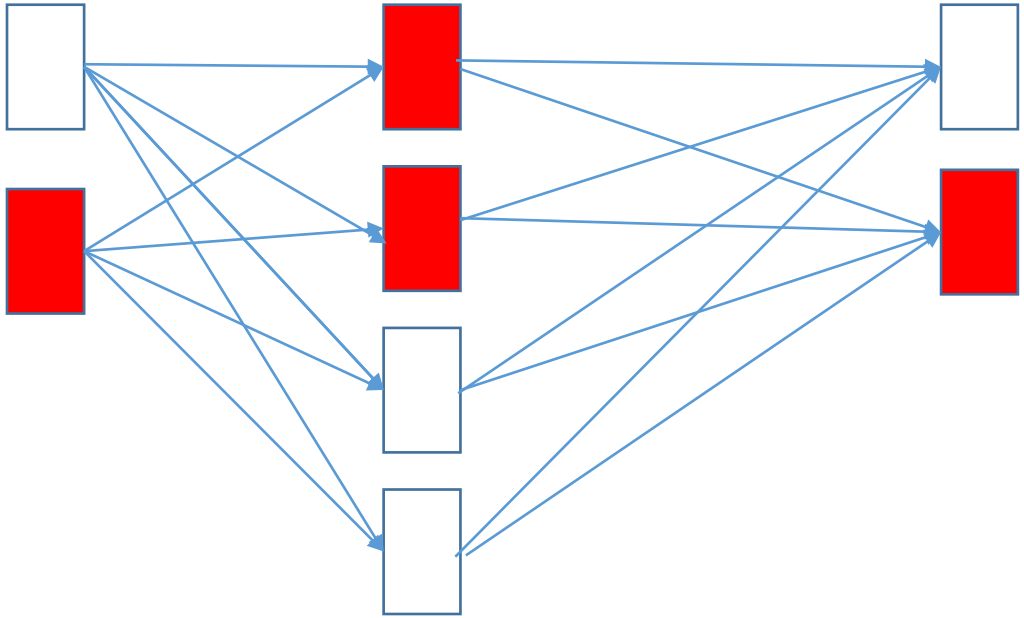
3. 活性化，伝搬

層が直列になっているニューラルネットワーク



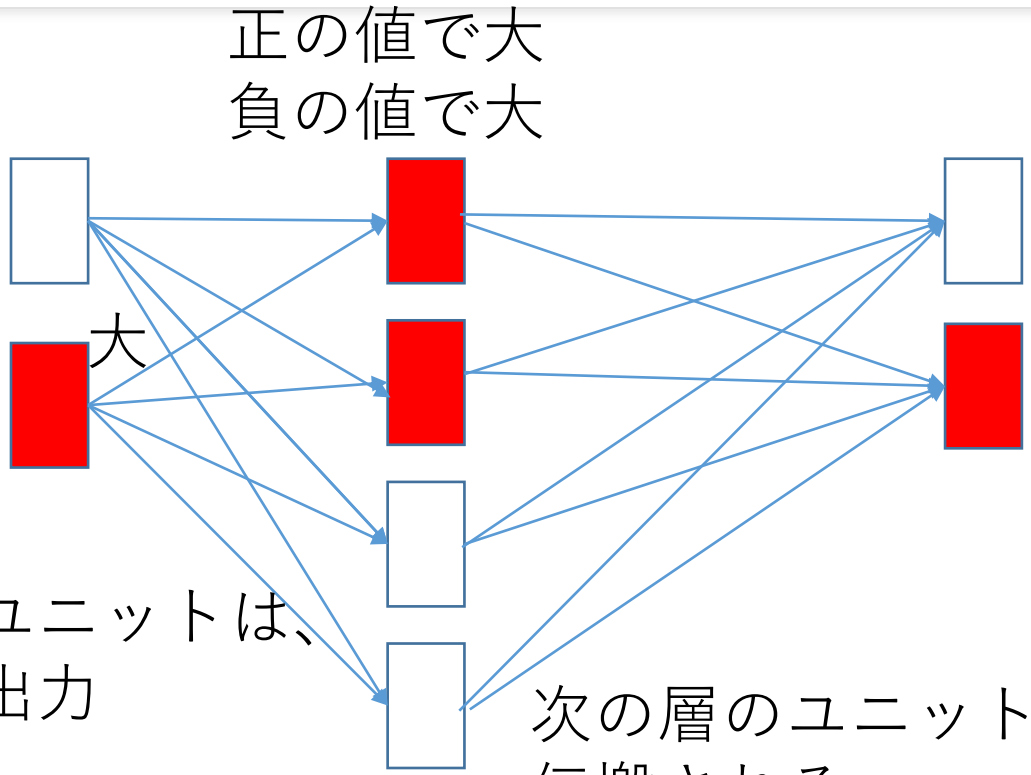
データは入力から出力の方向へ

活性化と非活性化



ニューロンは**活性化**したり、**非活性化**したりする
（入力に応じて**ダイナミック**に変化）

伝搬

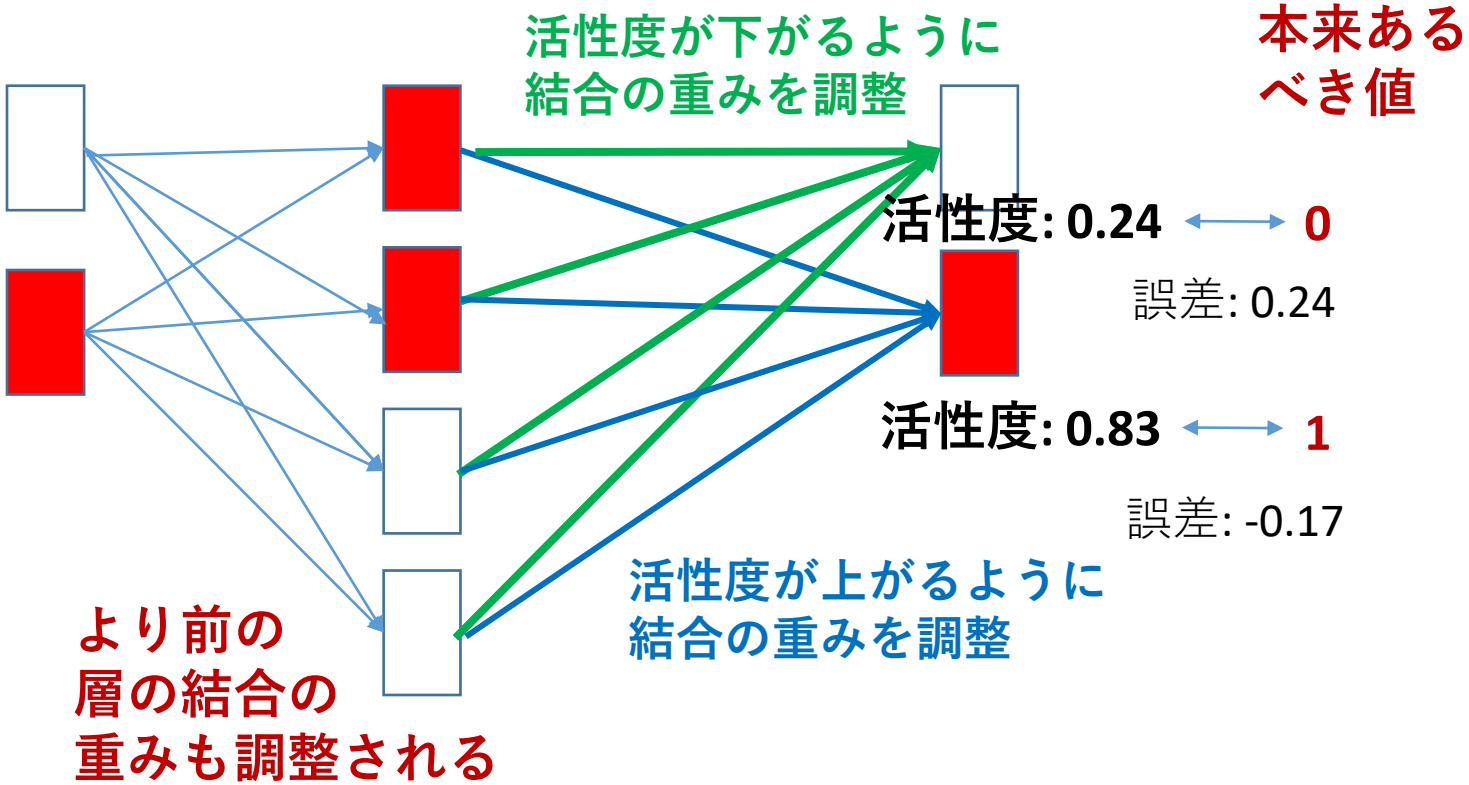


活性化したユニットは、
大きな値を出力

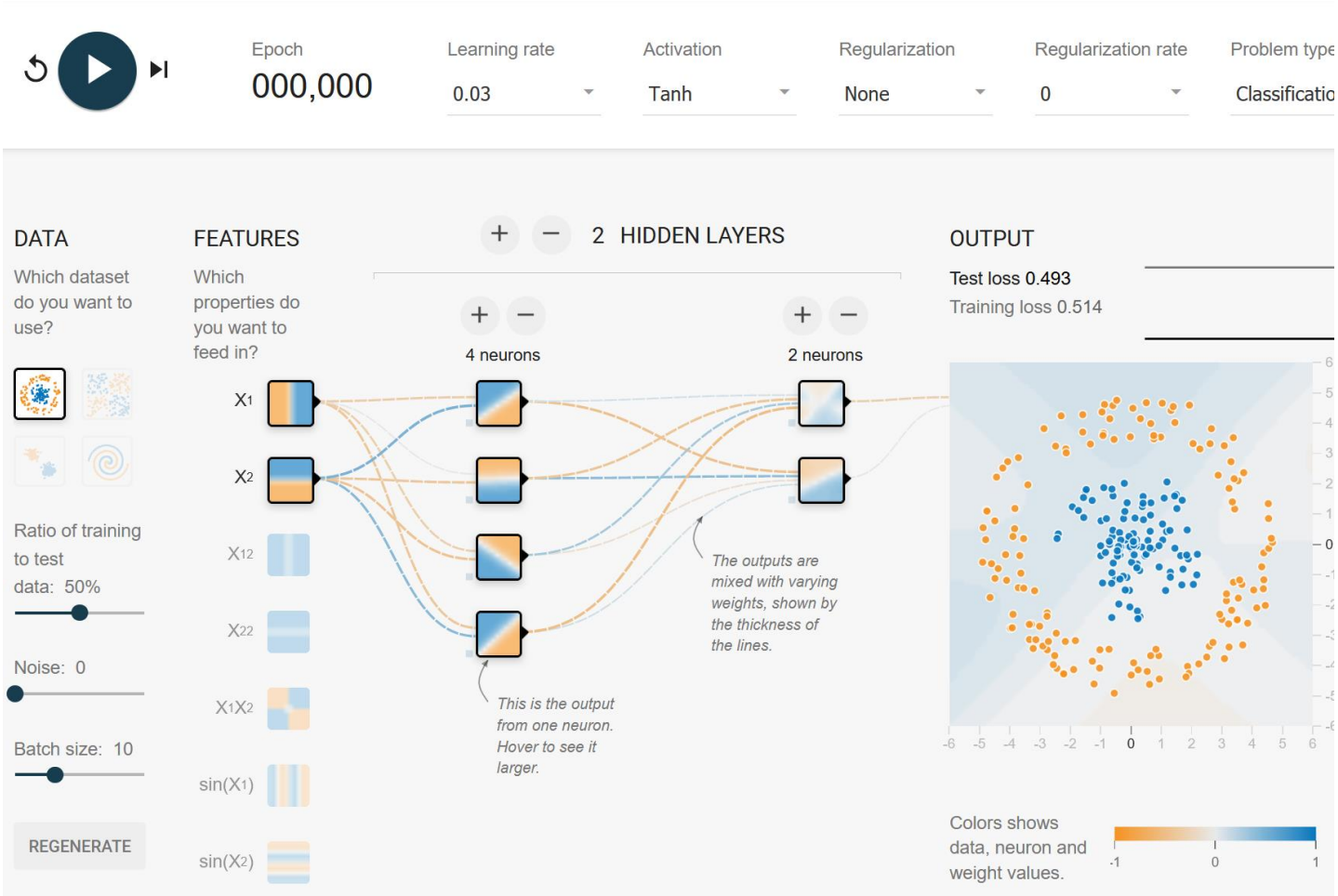
次の層のユニットに、大きな値が
伝搬される
・重みが正なら、正の大きな値
・重みが負なら、負の大きな値

4. ニューラルネットワークでの学習

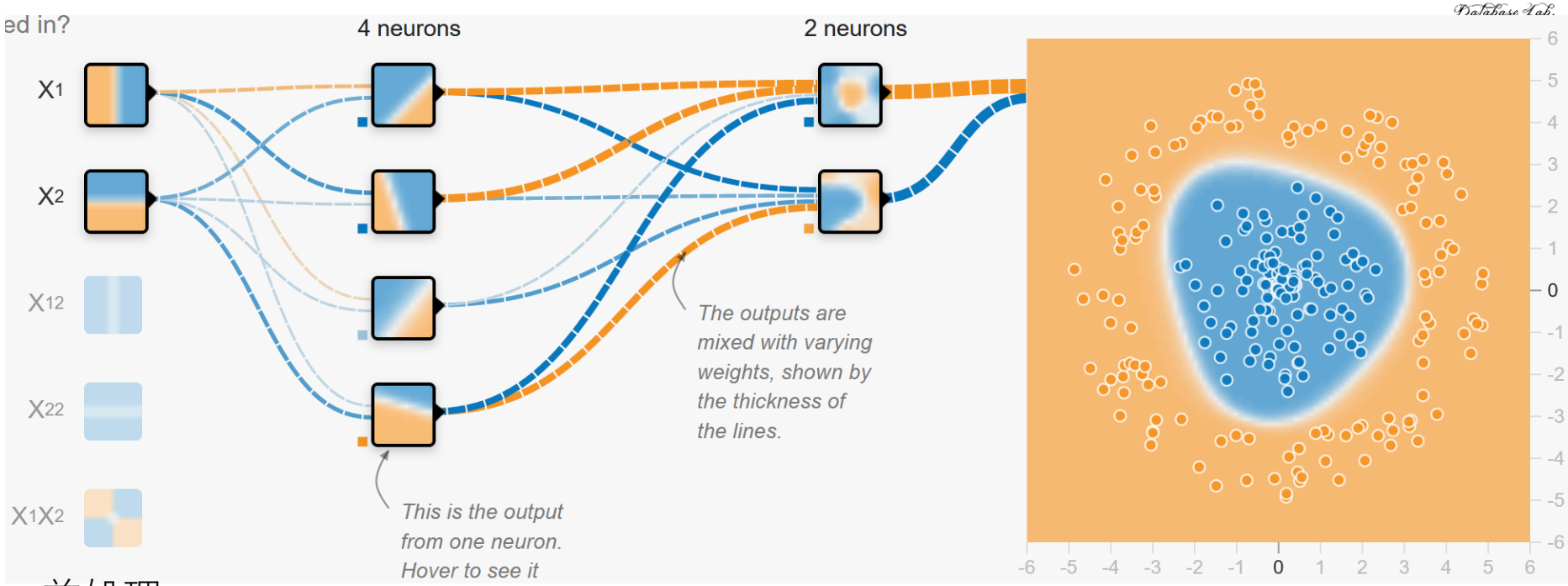
ニューラルネットワークの動作イメージ



学習能力をコンピュータに組み込んでおき，あとでデータを与えて学習させる



http://playground.tensorflow.org



前処理
（データが青い部分にあれば活性化）

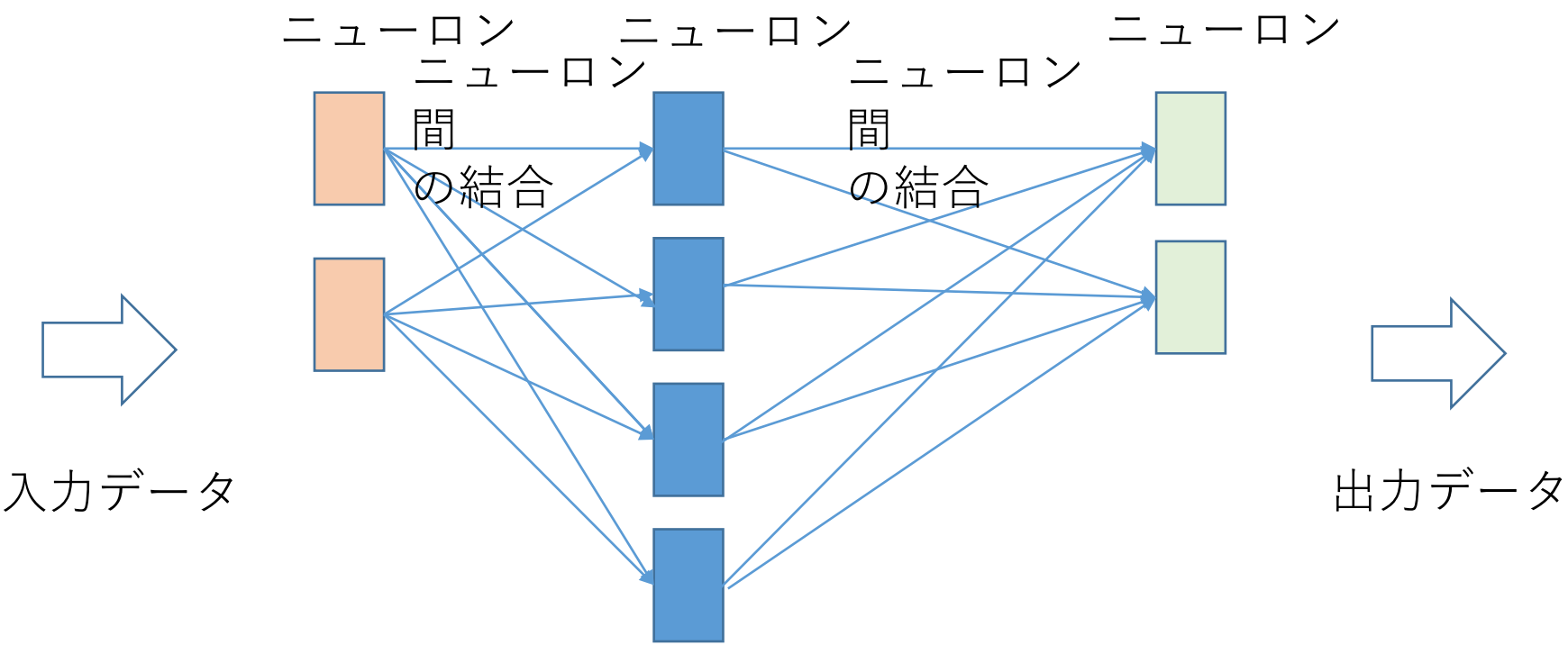
データが中央にあれば活性化

→結合→　1 層目　　→結合→　2 層目

ニューラルネットワーク

URL: https://playground.tensorflow.org/

まとめ

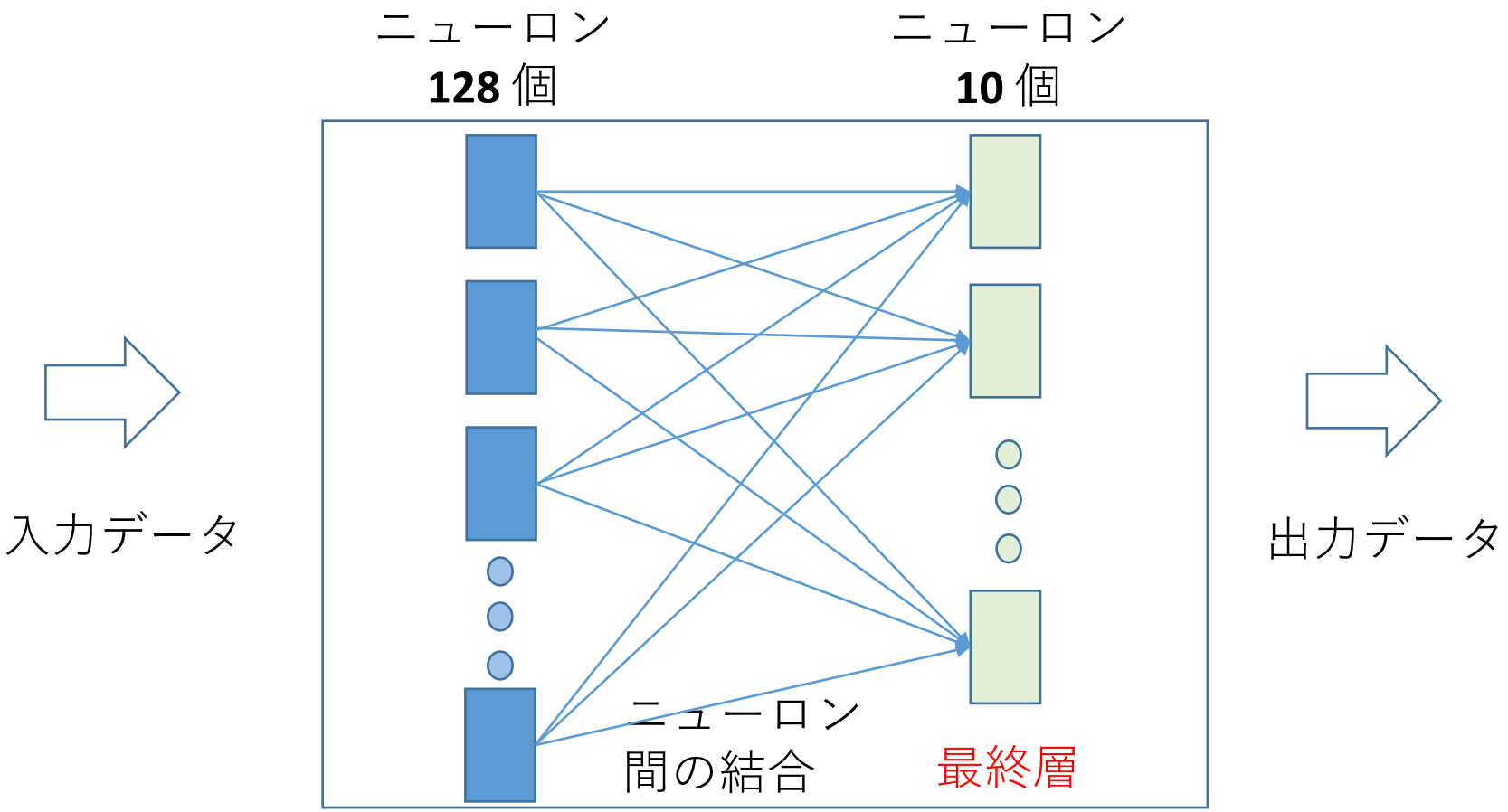


データは入力から出力の方向へ

ニューラルネットワークの**学習**では、**ニューロン間の結合**の**重み**が**変化**

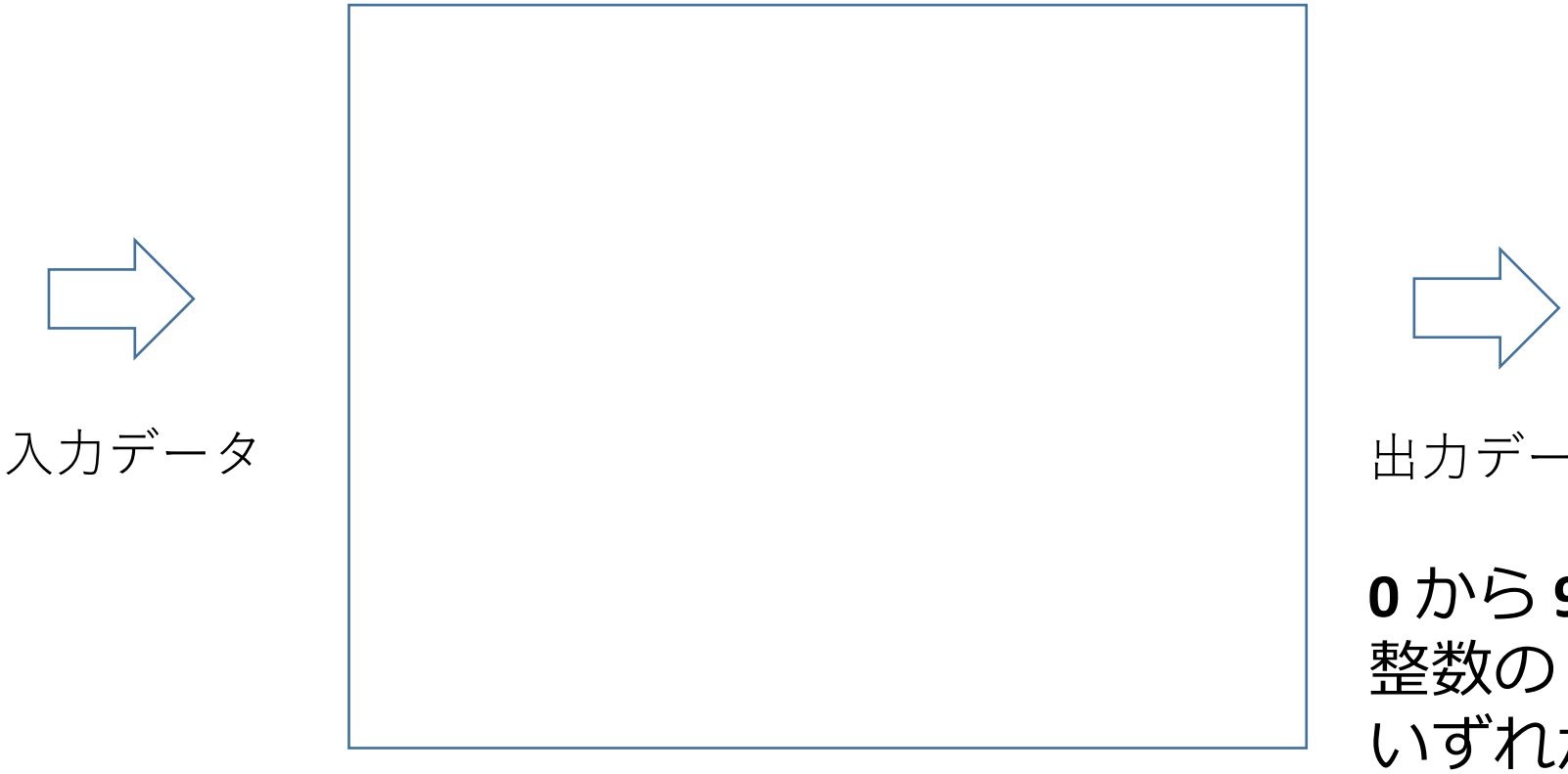
5. ニューラルネットワークを用いた分類

層が直列になっているニューラルネットワーク



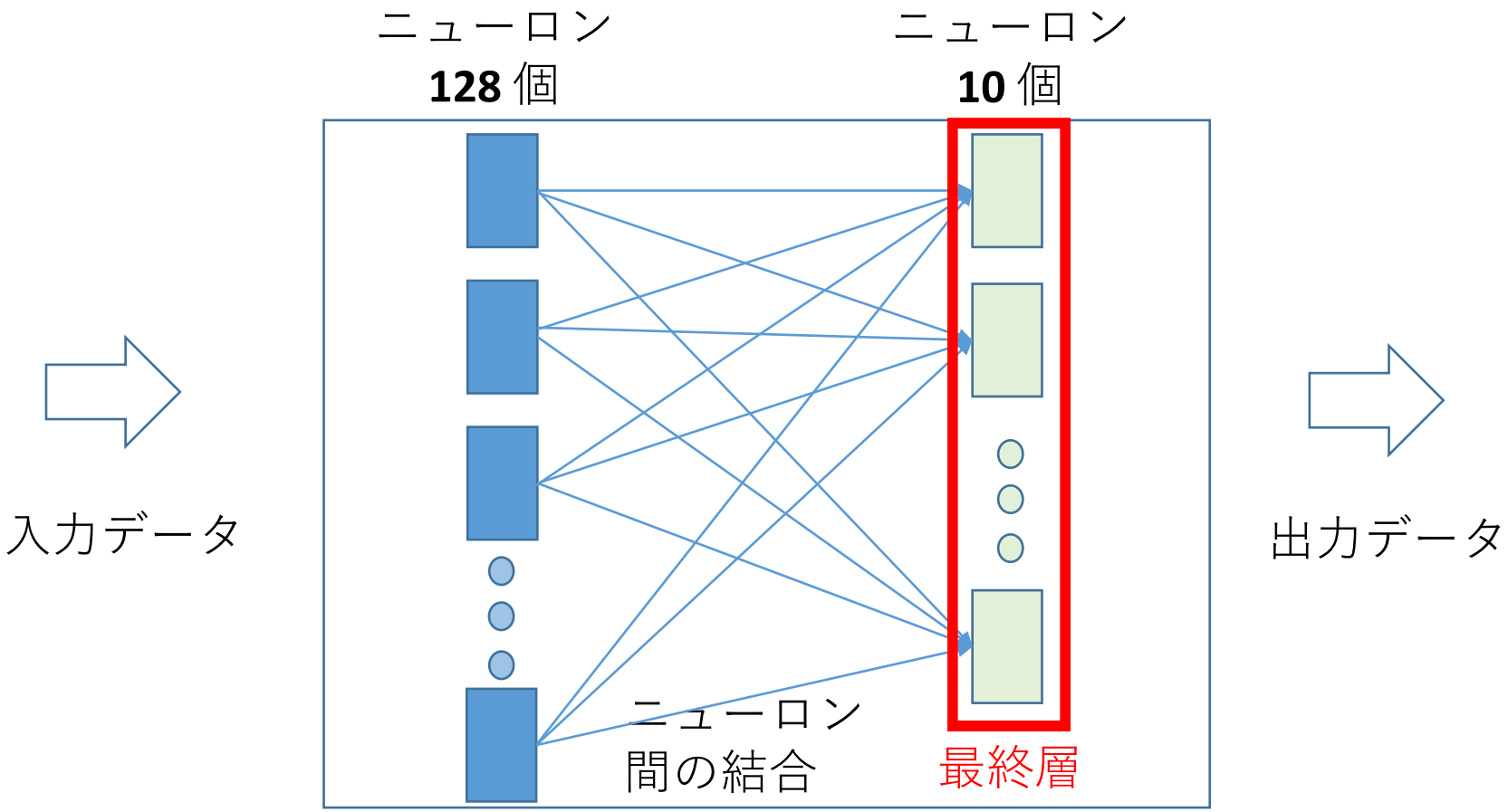
データは入力から出力の方向へ

１０種類の中から１つに分類する場合



１０種類に分類するニューラルネットワーク

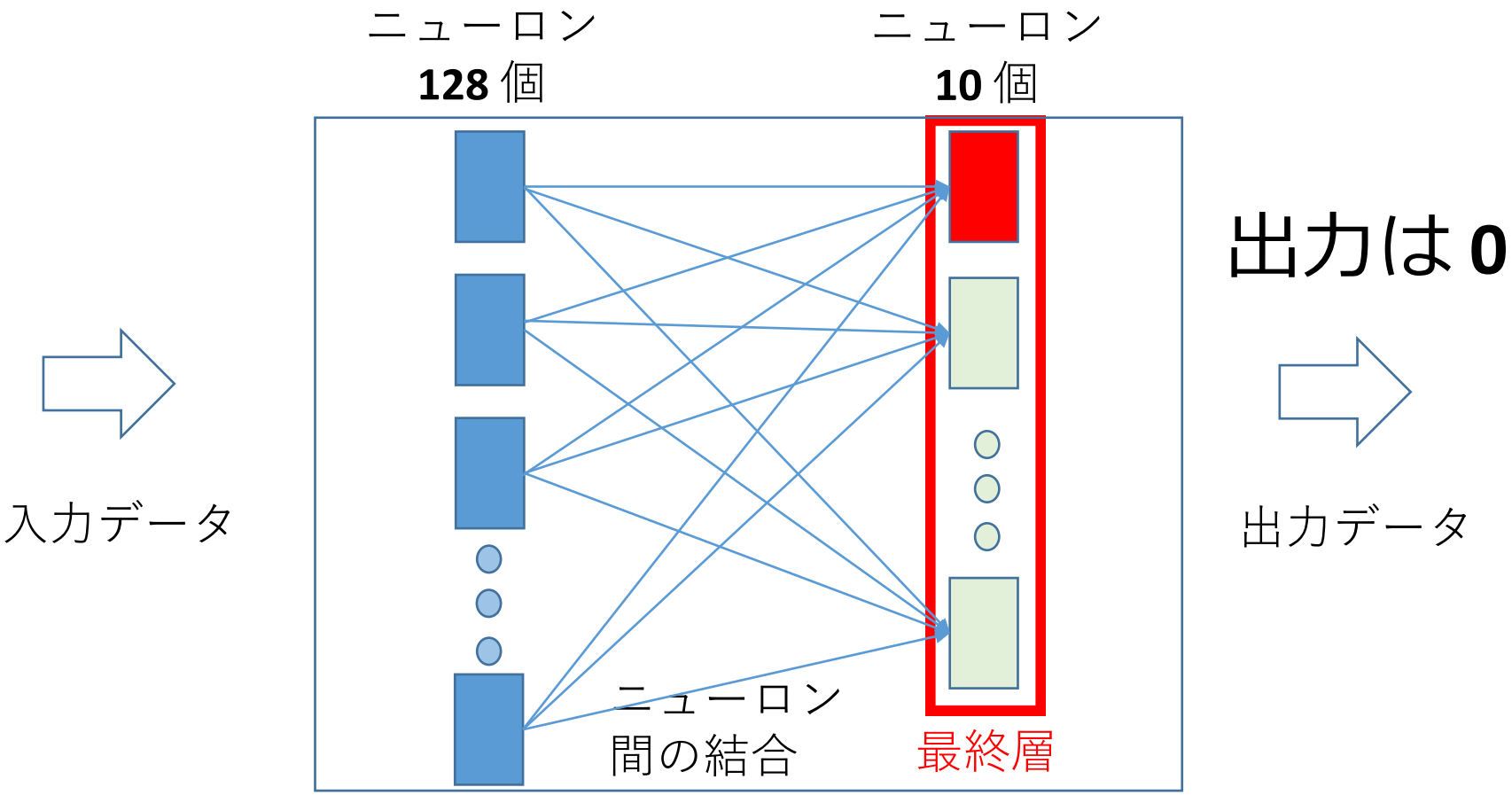
最終層について，１つが強く
活性化するように調整



データは入力から出力の方向へ

１０種類に分類するニューラルネットワーク

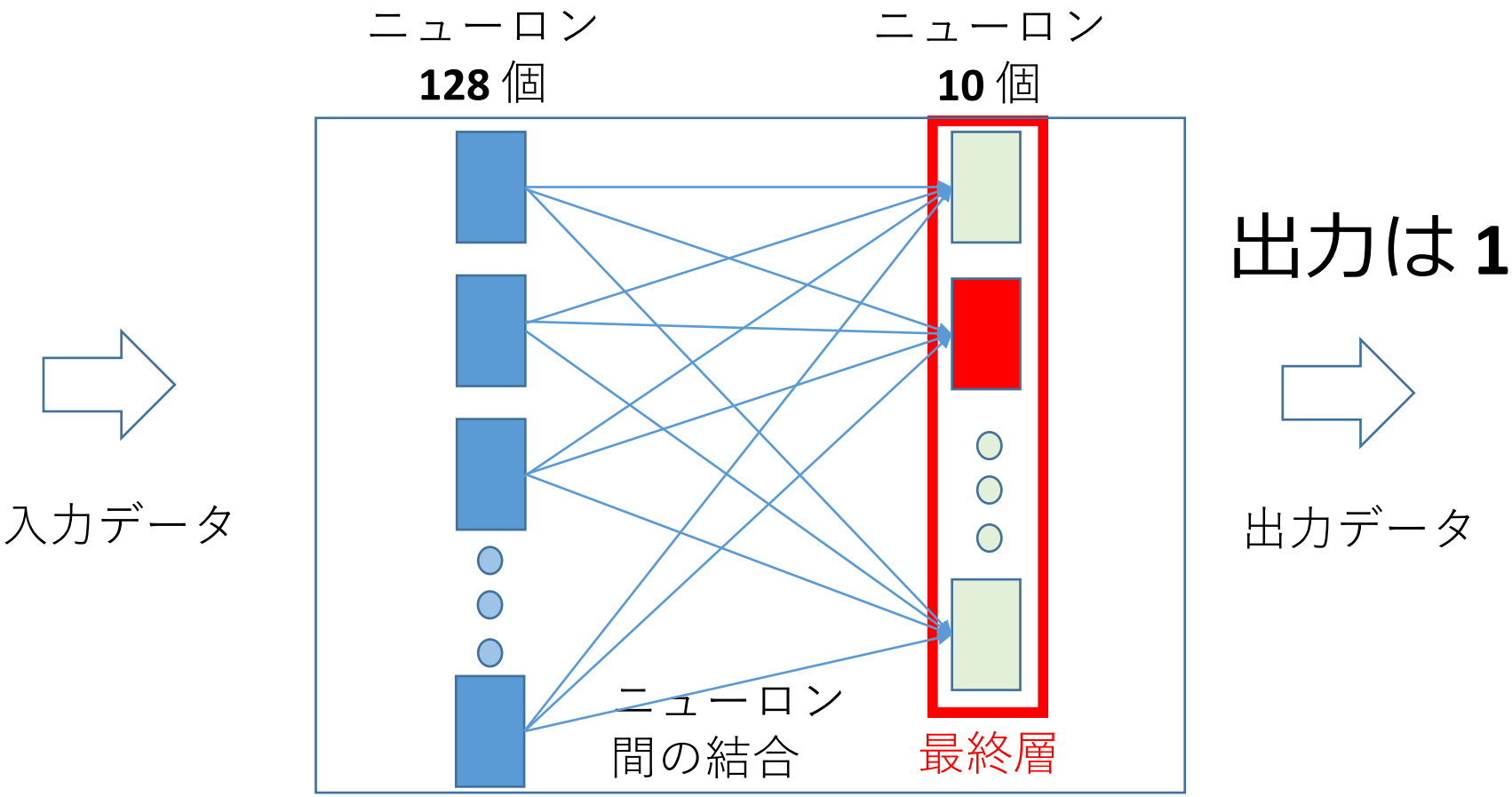
最終層について，１つが強く活性化するように調整



データは入力から出力の方向へ

１０種類に分類するニューラルネットワーク

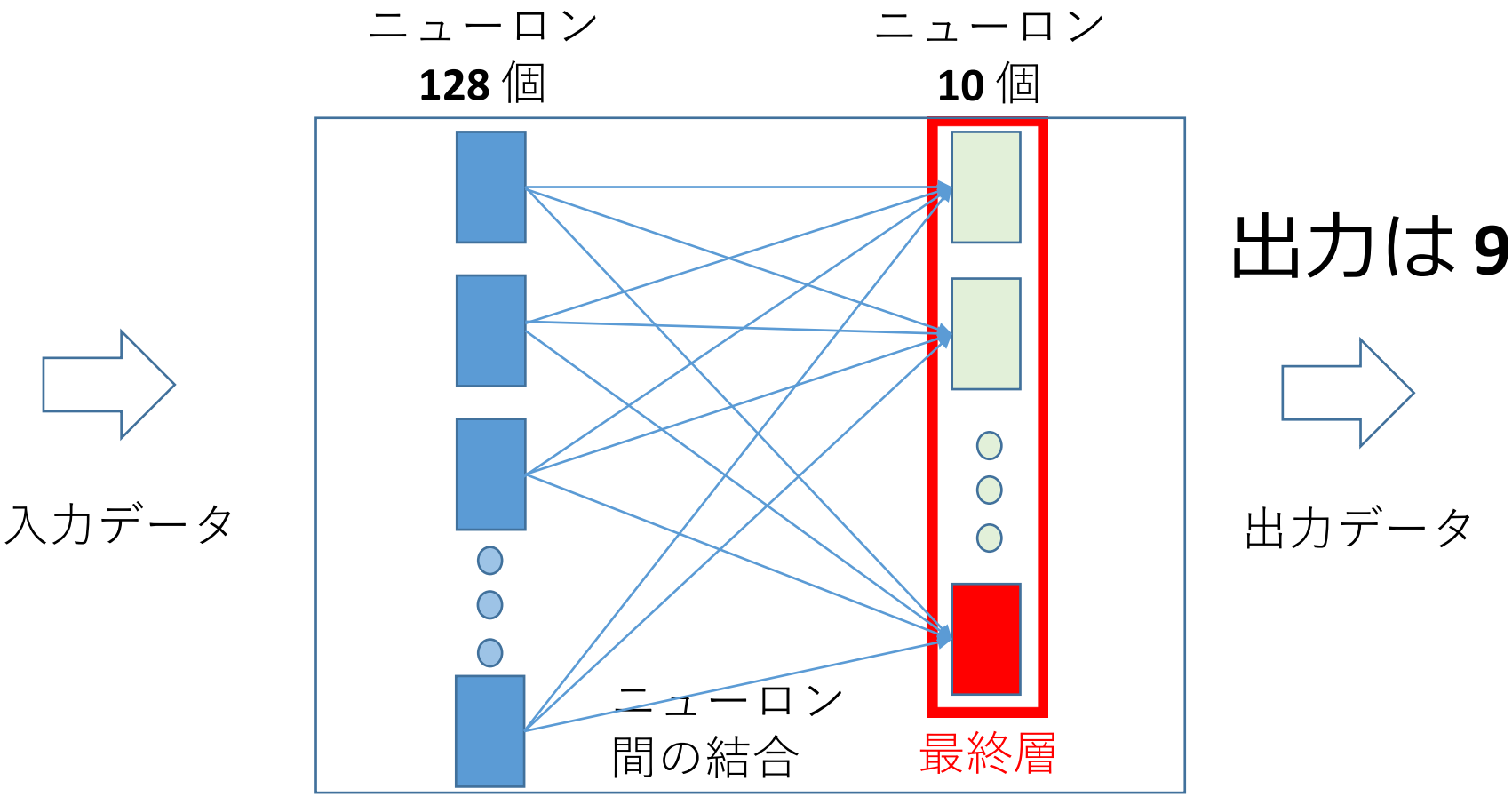
最終層について，１つが強く活性化するように調整



データは入力から出力の方向へ

１０種類に分類するニューラルネットワーク

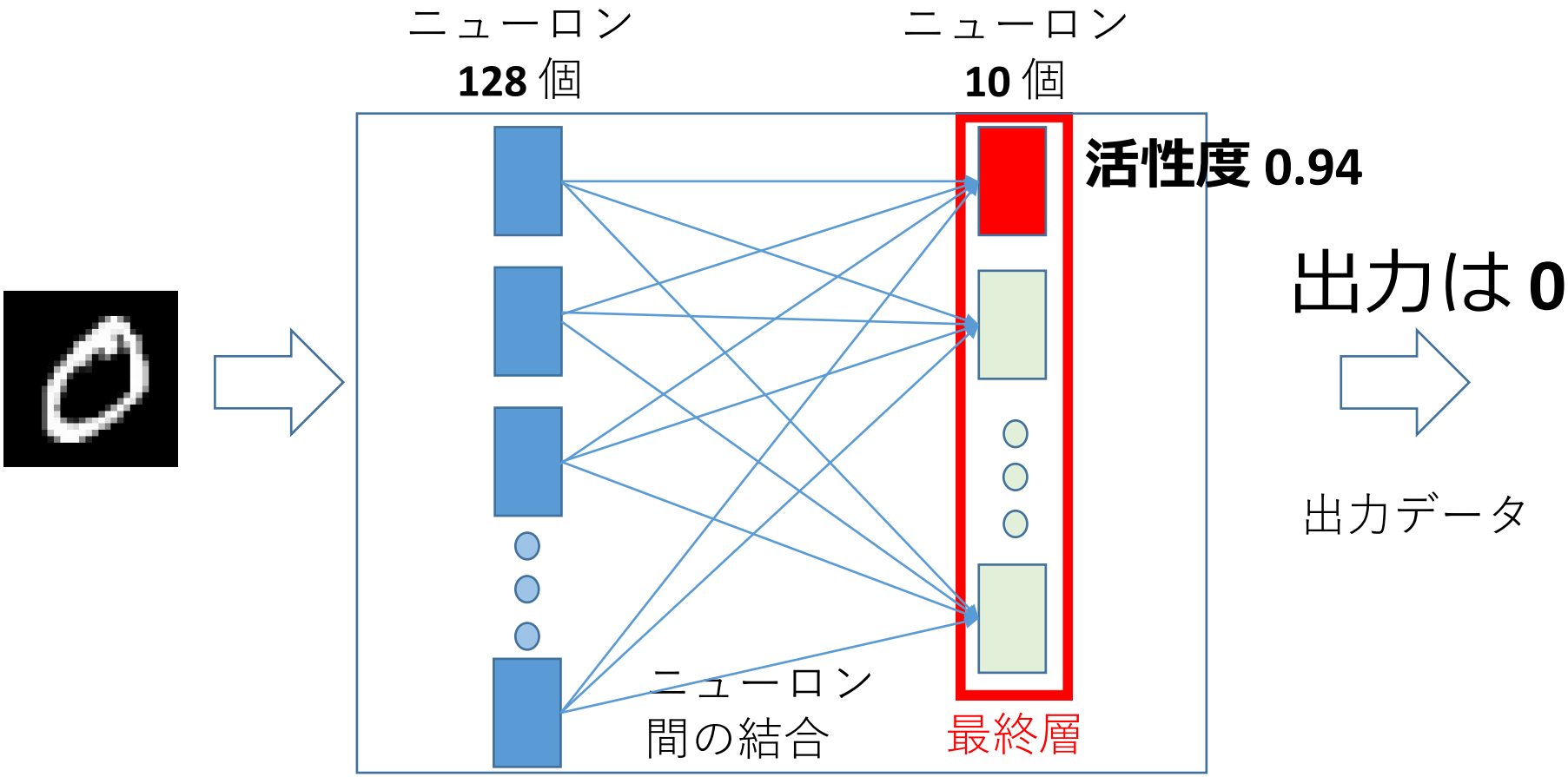
最終層について，１つが強く
活性化するように調整



データは入力から出力の方向へ

１０種類に分類するニューラルネットワーク

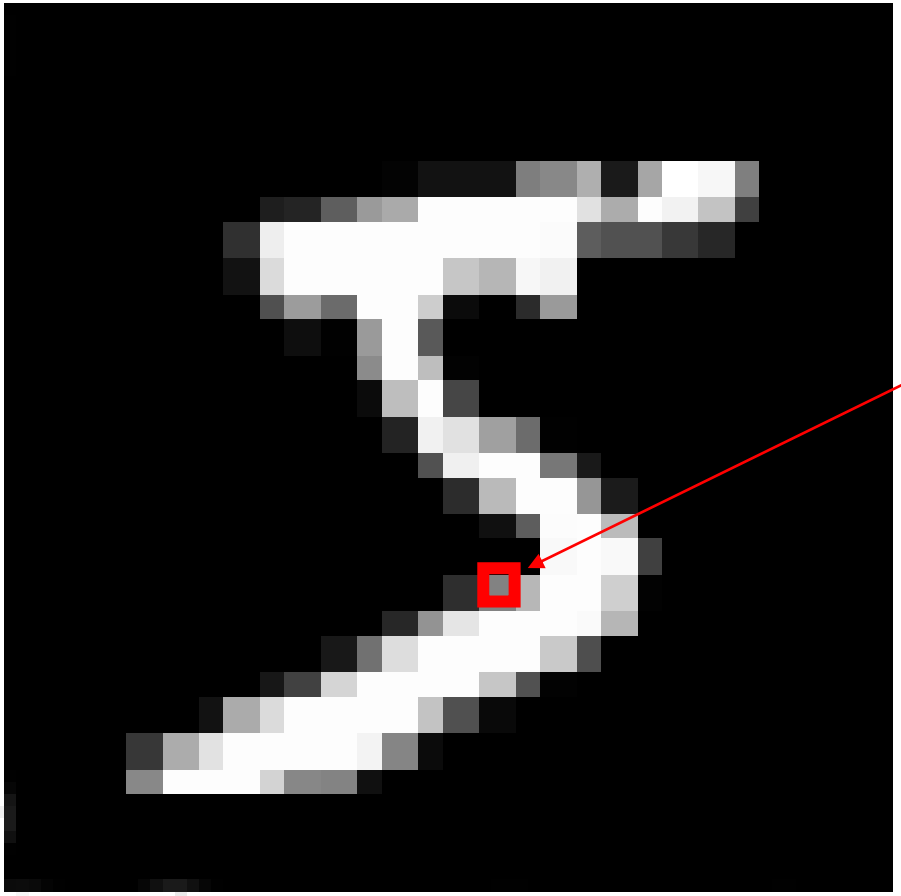
最終層について，１つが強く
活性化するように調整



実際には，活性度は **0 から 1** のような数値である．
最も**活性度**の**値が高いもの**が選ばれて，分類結果となる

6. 画像と画素

画像と画素



画素

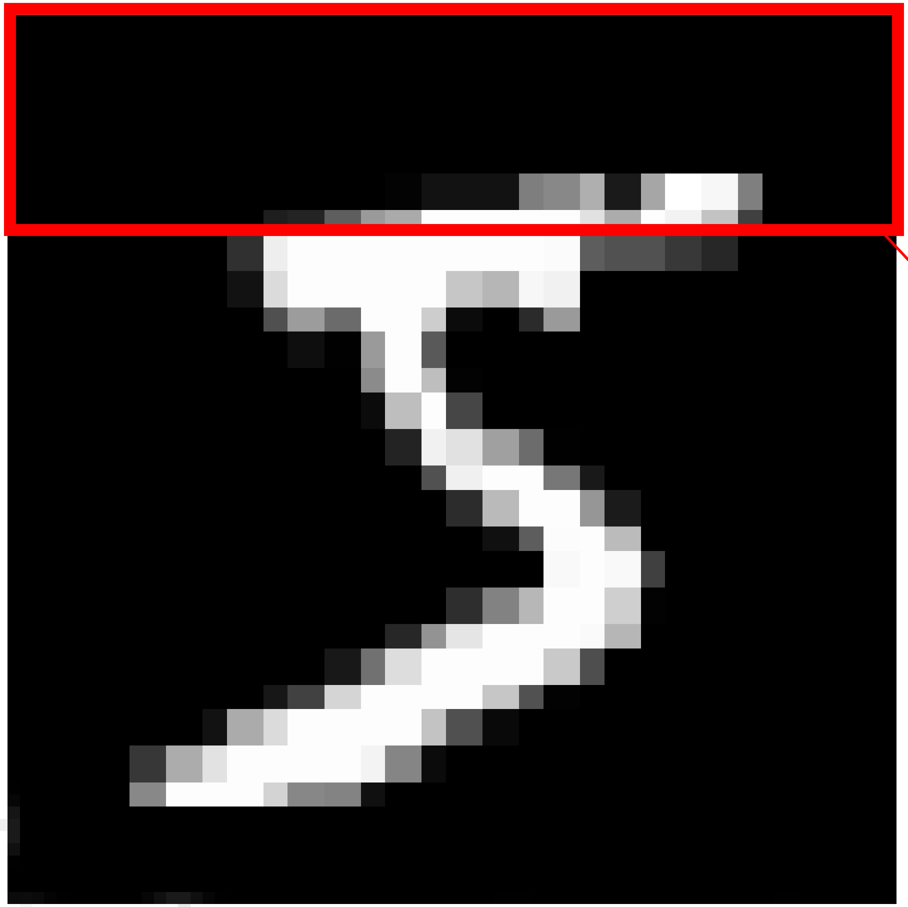
		画素値
	白	255
	黒	0

画素値は，**画素**の明るさに応じた **0 から 255 の数値**

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**

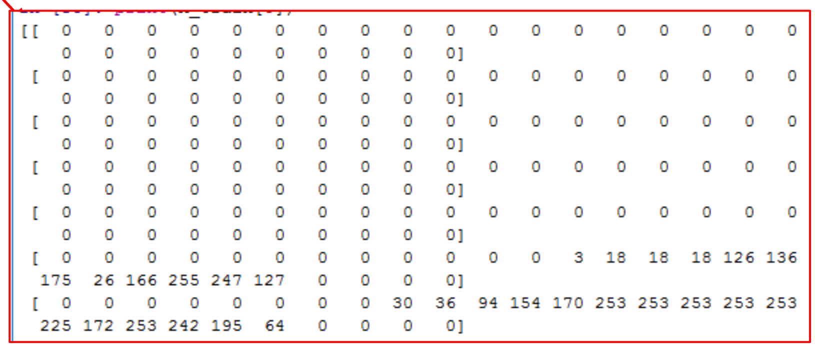
画像と配列（アレイ）



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

画像全体は，サイズ28 ×28 の**２次元**の**配列**



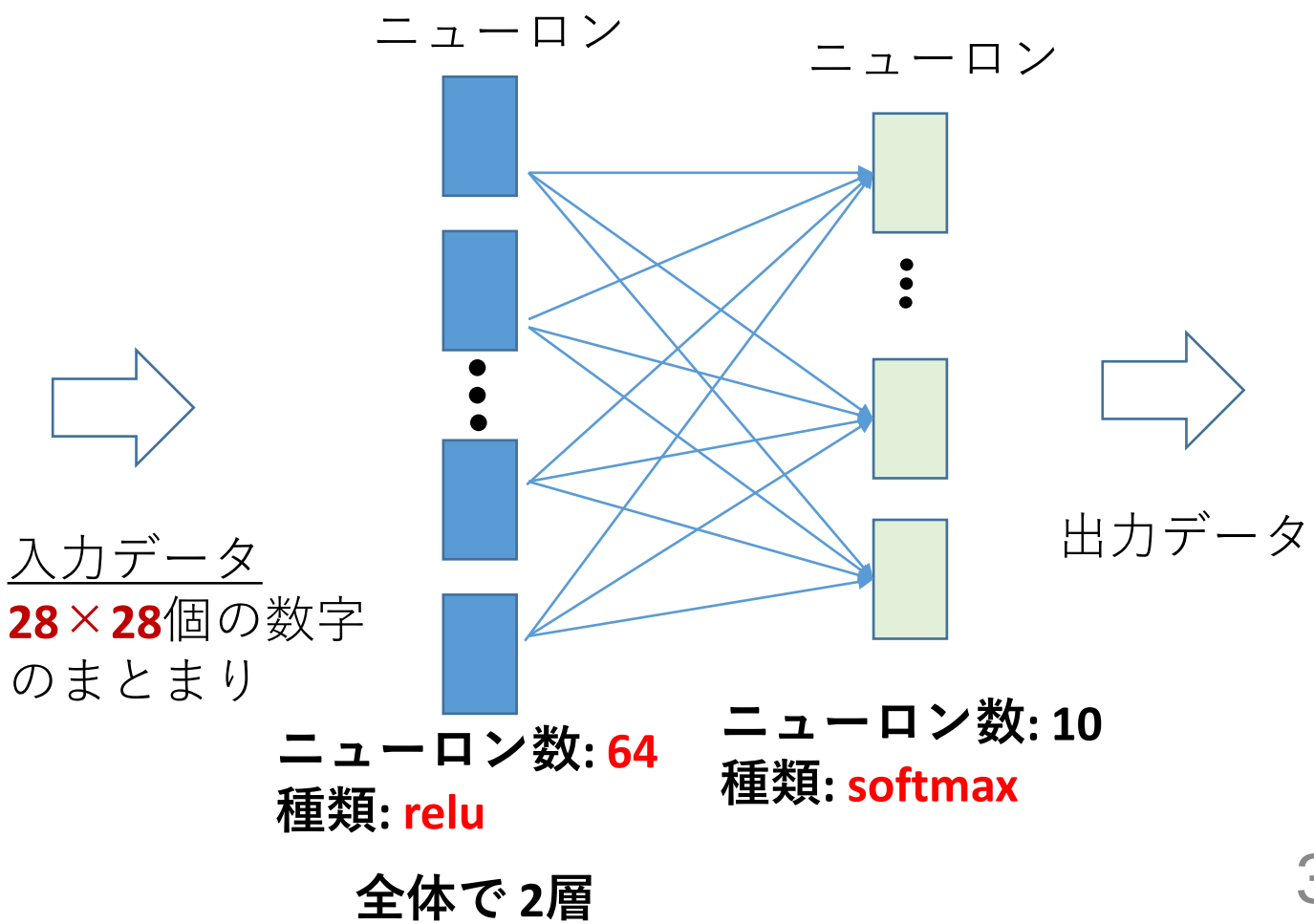
```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136
  175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253
  225 172 253 242 195  64   0   0   0   0]
```

画像の**<u>上 7行分</u>**の画素値を表示したところ（28 ×7分）

7. ニューラルネットワークの作成

ここで作成するニューラルネットワーク

- 入力： **28×28**個の数値のまとまり
- 出力： 「**28×28**個の数字のまとまり」について，**10**種類の中から 1つに**分類**



ニューラルネットワーク作成のプログラム例

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
```

入力データは **28×28**個の数字

１層目のニューロン数は **64**
種類は **relu**

```
m = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28, 1)),
    tf.keras.layers.Dense(units=64, activation=tf.nn.relu),
    tf.keras.layers.Dense(units=10, activation=tf.nn.softmax)
])
```

２層目のニューロン数は **10**
種類は **softmax**

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation

m = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28, 1)),
    tf.keras.layers.Dense(units=64, activation=tf.nn.relu),
    tf.keras.layers.Dense(units=10, activation=tf.nn.softmax)
])

print(m.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_2 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_4 (Dense)              (None, 64)                50240
_________________________________________________________________
dense_5 (Dense)              (None, 10)                650
=================================================================
Total params: 50,890
Trainable params: 50,890
Non-trainable params: 0
_________________________________________________________________
None
```

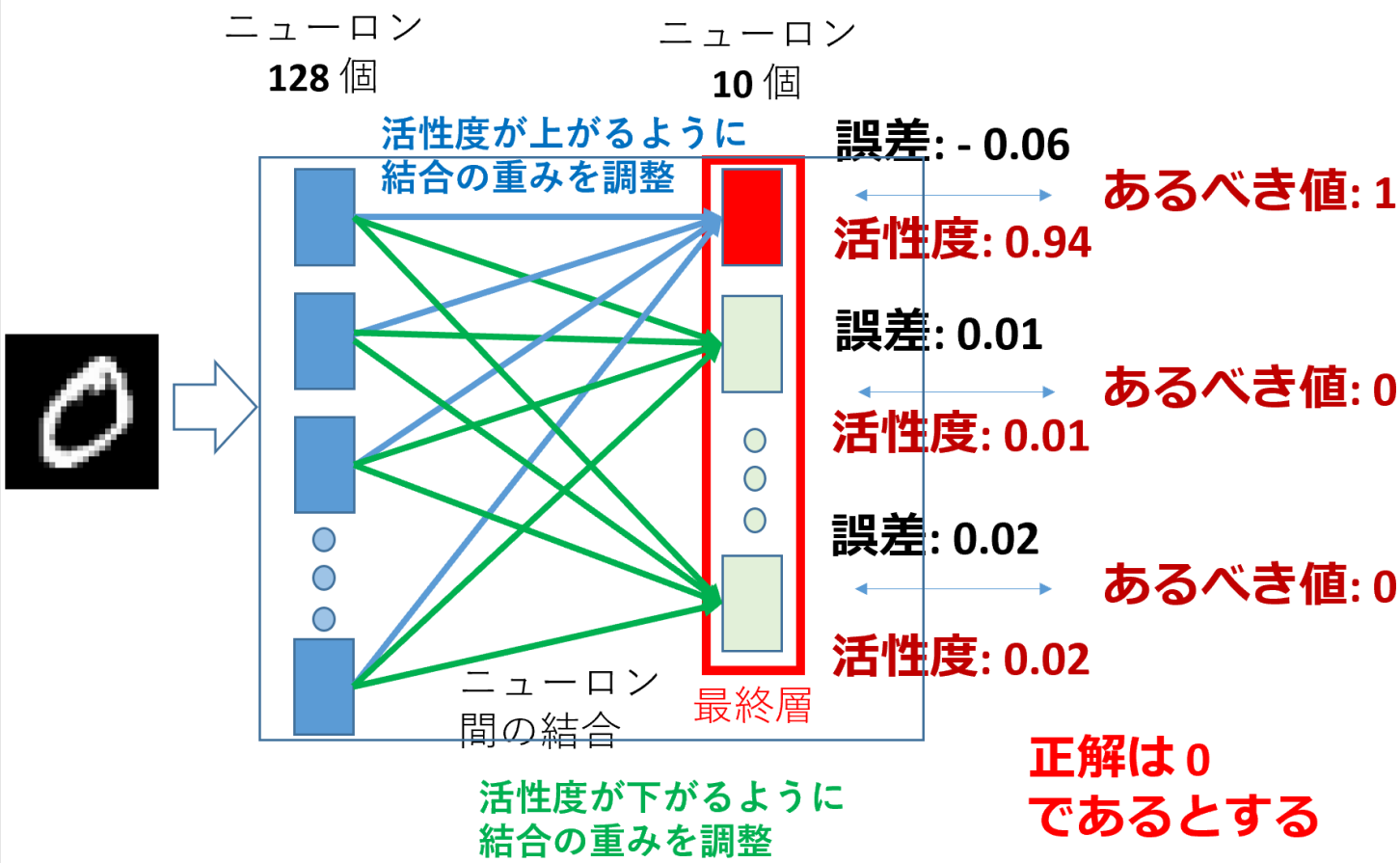
ニューラルネットワークのオブジェクト名は **m**
print(**m**.summary()) は確認表示

8. 学習不足と過学習

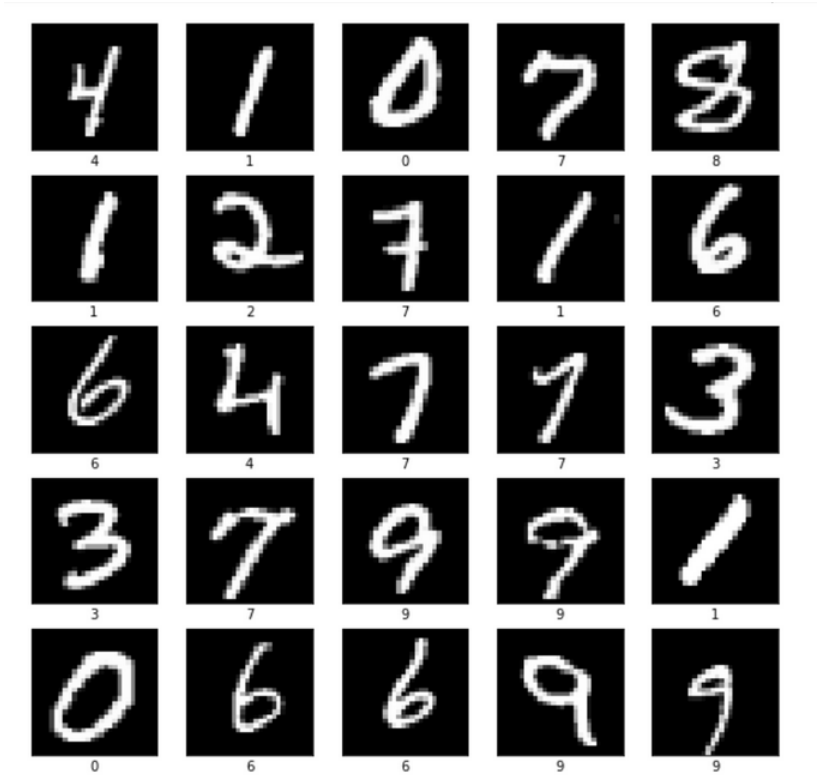
ニューラルネットワークの学習

学習：**訓練データ**を使い，

適切な結果が得られるように，**結合の重み**を調整



訓練データの例



4 1 0 7 8
1 2 7 1 6
6 4 7 7 3
3 7 9 9 1
0 6 6 9 9

画像 60000枚
（うち一部）

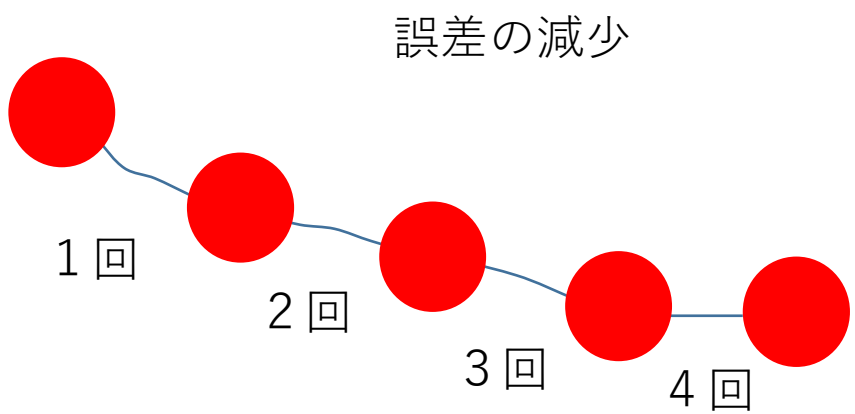
正解 60000個
（うち一部）

学習不足

- **ニューラルネットワーク**の**学習**では，学習のためのデータ（**訓練データ**）を使う

- **訓練データ**を１回使っただけでは，**学習不足**の場合がある

→　同じ**訓練データ**を繰り返し使って学習を行う．
　　繰り返しながら，誤差の減少を確認



```
EPOCHS = 20
history = m.fit(ds_train[0], ds_train[1],
                epochs=EPOCHS,
                validation_data=(ds_test[0], ds_test[1]),
                verbose=1)
```

```
Epoch 1/20
1875/1875 [==============================] - 5s 2ms/step - loss: 0.2997 - sparse_categorical_crossentropy: 0.2997 - accuracy: 0.9150 - val_loss: 0.1632 - val_sparse_categorical_crossentropy: 0.1632 - val_accuracy: 0.9524
Epoch 2/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1437 - sparse_categorical_crossentropy: 0.1437 - accuracy: 0.9576 - val_loss: 0.1315 - val_sparse_categorical_crossentropy: 0.1315 - val_accuracy: 0.9599
Epoch 3/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1049 - sparse_categorical_crossentropy: 0.1049 - accuracy: 0.9686 - val_loss: 0.1018 - val_sparse_categorical_crossentropy: 0.1018 - val_accuracy: 0.9688
Epoch 4/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0844 - sparse_categorical_crossentropy: 0.0844 - accuracy: 0.9747 - val_loss: 0.0945 - val_sparse_categorical_crossentropy: 0.0945 - val_accuracy: 0.9733
Epoch 5/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0695 - sparse_categorical_crossentropy: 0.0695 - accuracy: 0.9794 - val_loss: 0.1018 - val_sparse_categorical_crossentropy: 0.1018 - val_accuracy: 0.9691
Epoch 6/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0589 - sparse_categorical_crossentropy: 0.0589 - accuracy: 0.9827 - val_loss: 0.1003 - val_sparse_categorical_crossentropy: 0.1003 - val_accuracy: 0.9715
Epoch 7/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0500 - sparse_categorical_crossentropy: 0.0500 - accuracy: 0.9847 - val_loss: 0.0918 - val_sparse_categorical_crossentropy: 0.0918 - val_accuracy: 0.9727
Epoch 8/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0433 - sparse_categorical_crossentropy: 0.0433 - accuracy: 0.9867 - val_loss: 0.0979 - val_sparse_categorical_crossentropy: 0.0979 - val_accuracy: 0.9717
Epoch 9/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0373 - sparse_categorical_crossentropy: 0.0373 - accuracy: 0.9889 - val_loss: 0.0904 - val_sparse_categorical_crossentropy: 0.0904 - val_accuracy: 0.9747
Epoch 10/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0329 - sparse_categorical_crossentropy: 0.0329 - accuracy: 0.9901 - val_loss: 0.1027 - val_sparse_categorical_crossentropy: 0.1027 - val_accuracy: 0.9708
Epoch 11/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0294 - sparse_categorical_crossentropy: 0.0294 - accuracy: 0.9908 - val_loss: 0.0924 - val_sparse_categorical_crossentropy: 0.0924 - val_accuracy: 0.9756
Epoch 12/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0251 - sparse_categorical_crossentropy: 0.0251 - accuracy: 0.9921 - val_loss: 0.1053 - val_sparse_categorical_crossentropy: 0.1053 - val_accuracy: 0.9724
Epoch 13/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0223 - sparse_categorical_crossentropy: 0.0223 - accuracy: 0.9931 - val_loss: 0.0979 - val_sparse_categorical_crossentropy: 0.0979 - val_accuracy: 0.9733
Epoch 14/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0199 - sparse_categorical_crossentropy: 0.0199 - accuracy: 0.9942 - val_loss: 0.0968 - val_sparse_categorical_crossentropy: 0.0968 - val_accuracy: 0.9747
Epoch 15/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0178 - sparse_categorical_crossentropy: 0.0178 - accuracy: 0.9944 - val_loss: 0.1180 - val_sparse_categorical_crossentropy: 0.1180 - val_accuracy: 0.9701
Epoch 16/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0157 - sparse_categorical_crossentropy: 0.0157 - accuracy: 0.9952 - val_loss: 0.1062 - val_sparse_categorical_crossentropy: 0.1062 - val_accuracy: 0.9739
Epoch 17/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0152 - sparse_categorical_crossentropy: 0.0152 - accuracy: 0.9953 - val_loss: 0.1077 - val_sparse_categorical_crossentropy: 0.1077 - val_accuracy: 0.9735
Epoch 18/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0120 - sparse_categorical_crossentropy: 0.0120 - accuracy: 0.9965 - val_loss: 0.1193 - val_sparse_categorical_crossentropy: 0.1193 - val_accuracy: 0.9720
Epoch 19/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0120 - sparse_categorical_crossentropy: 0.0120 - accuracy: 0.9963 - val_loss: 0.1201 - val_sparse_categorical_crossentropy: 0.1201 - val_accuracy: 0.9741
Epoch 20/20
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0104 - sparse_categorical_crossentropy: 0.0104 - accuracy: 0.9971 - val_loss: 0.1095 - val_sparse_categorical_crossentropy: 0.1095 - val_accuracy: 0.9748
```

ニューラルネットワークの学習，検証

20 のようにある通り，**学習が 20 回繰り返されている．**

学習のたびに検証を実施．繰り返し学習の間、誤差は減少

ニューラルネットワークの学習のプログラム

訓練データの指定

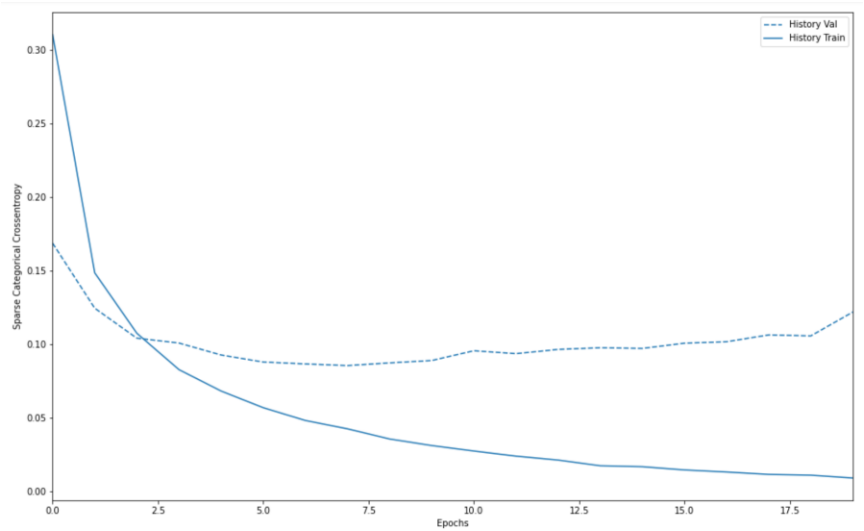
```
EPOCHS = 20
history = m.fit(ds_train[0], ds_train[1],
                epochs=EPOCHS,
                validation_data=(ds_test[0], ds_test[1]),
                verbose=1)
```

学習の実行
学習を 20回繰り返す

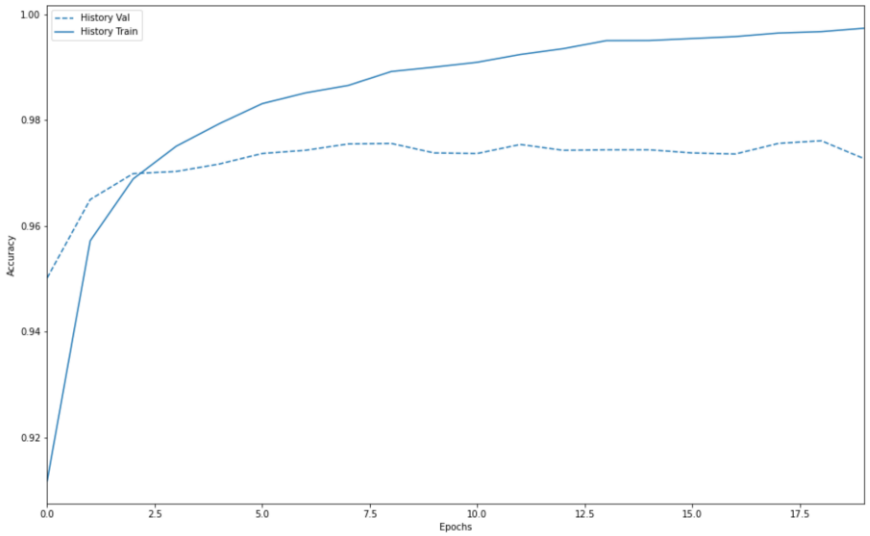
検証データを指定することで，
検証も行われる

学習曲線

- **学習の繰り返しが進む**につれ，**誤差（損失）**は**減少**，**精度**は**向上**



誤差の減少



精度の向上

大量のデータがある場合には、訓練データと検証データに振り分けることができる

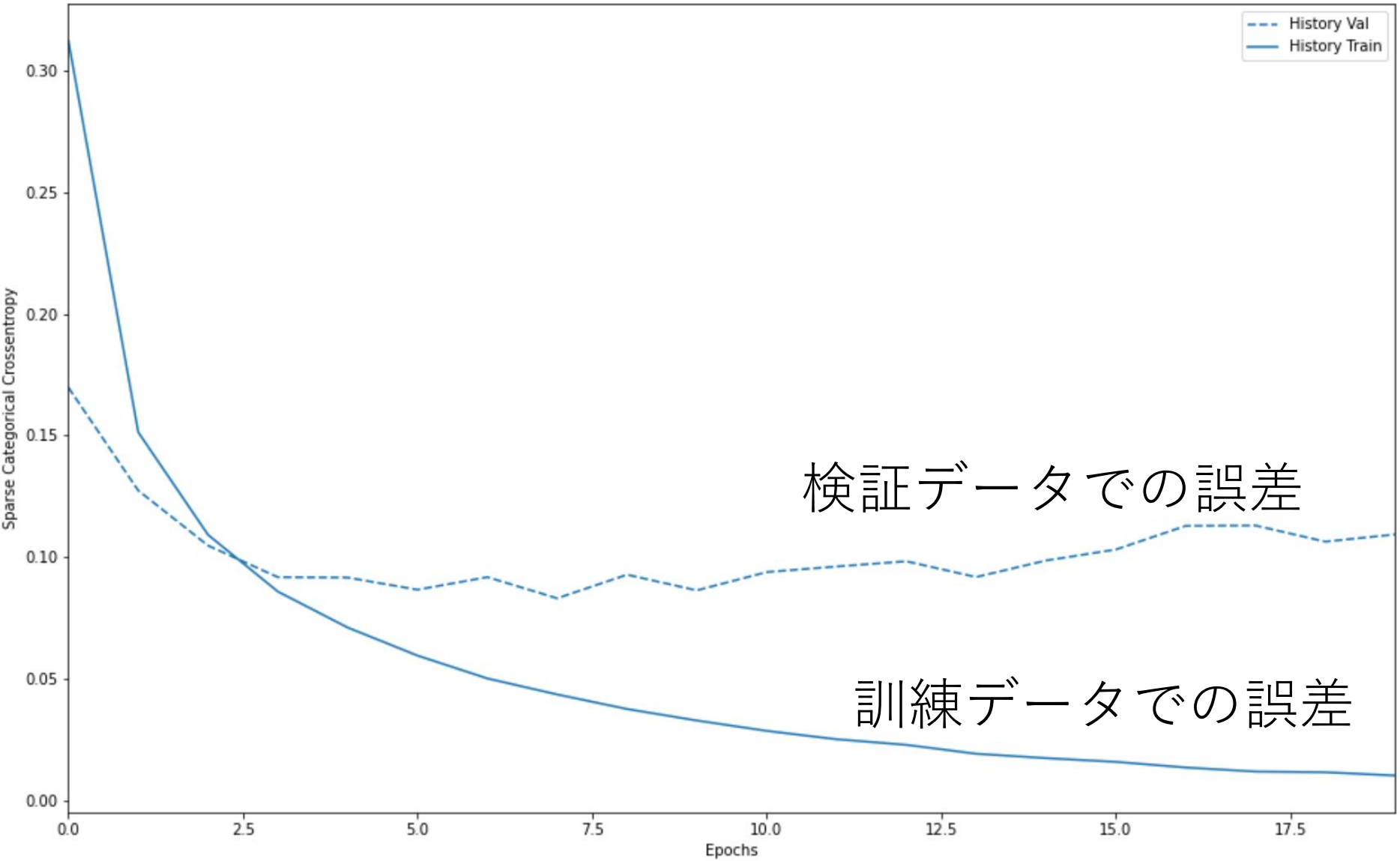
例えば，**70000** 枚の画像データを準備できる場合

うち **60000**枚 → **訓練データ**として使用

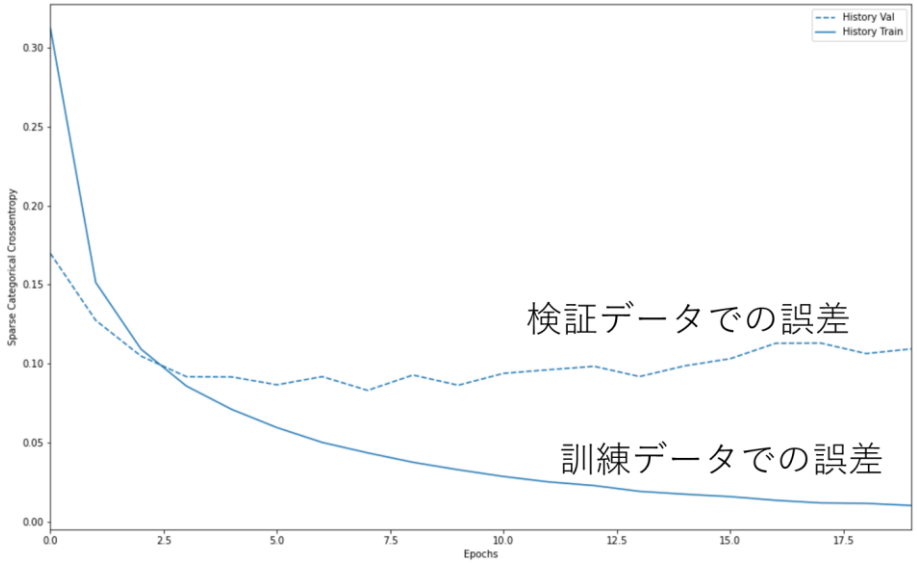
うち **10000**枚 → **<u>検証で使用</u>**

のように考えることができる

<u>※ 訓練データと検証データは別々であることが重要.</u>
60000 や 10000 のような枚数は，自由に決めることができるもの.


History Val
History Train
Sparse Categorical Crossentropy
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epochs
検証データでの誤差
訓練データでの誤差

過学習



訓練データによる**学習の繰り返し**

- **訓練データ**では，精度が向上**する**が，**検証データ**では精度が向上**しない**ことがある　＝　**過学習**
- **過学習**の発見のため，**検証データ**での**検証**が必要
- **過学習**では，**学習の繰り返し**による**精度の悪化**もありえる．学習の**打ち切り**が必要．

結合の重みの表示

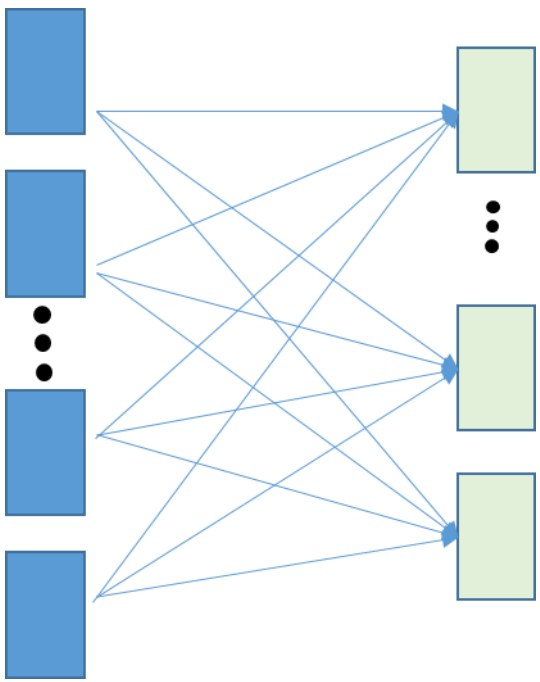
プログラム

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

実行結果

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

```
array([[-1.01239467e+00, -3.47466946e-01, -8.09835494e-02,
         7.45151564e-02,  2.85807520e-01,  5.68768084e-01,
        -1.13867247e+00,  4.49037850e-01, -5.76055467e-01,
         3.08223069e-01],
       [-9.73109961e-01,  3.54716599e-01,  2.76776433e-01,
         2.45247036e-01, -7.31139123e-01,  5.26507080e-01,
        -4.12931621e-01,  8.27608228e-01, -4.87542659e-01,
        -1.92633331e+00],
       [-1.54334641e+00,  5.33413351e-01, -2.16112331e-01,
         2.96245068e-01,  3.39144379e-01,  3.52841973e-01,
        -1.73382103e+00,  3.67599517e-01, -8.94106865e-01,
         3.49798262e-01],
       [ 2.79795349e-01,  3.93170774e-01, -1.23119526e-01,
        -2.24058971e-01,  1.40130982e-01, -6.56837881e-01,
         5.34807503e-01,  7.34667897e-01, -1.04529119e+00,
        -5.81060231e-01],
       [ 2.05441475e-01,  1.93122208e-01,  2.66112298e-01,
        -8.21578577e-02,  1.70343146e-01, -2.91442454e-01,
        -2.61067390e-01, -3.78575660e-02, -1.49935138e+00,
         4.05946940e-01],
       [-2.01313898e-01,  3.22882854e-03,  5.78866839e-01,
        -8.86083782e-01, -5.89539528e-01, -9.65685993e-02,
        -1.41050875e-01,  8.38579014e-02,  2.56438136e-01,
        -6.41472265e-02],
       [ 2.73324281e-01, -4.17244375e-01, -2.91900814e-01,
```



学習により

結合の重みが変化する

9. ニューラルネットワークの利用の流れ

前準備

Google アカウントの取得が必要

- 次のページを使用

https://accounts.google.com/SignUp

- 次の情報を登録する

氏名

自分が希望するメールアドレス

＜ユーザー名＞@gmail.com

パスワード

生年月日，性別

使用するプログラム

① プログラムは、次のページで公開

https://colab.research.google.com/drive/1IfArIvhh-FsvJIE9YTNO8T44Qhpi0rIJ?usp=sharing

② 利用には，**Google アカウント**での**ログイン**が必要

③ 右上の「**ログイン**」を**クリック**して利用開始

④ **コードセル**を**上から順に実行**する.

コードセルの**実行の終了を確認**してから、**次のコードセル**に移ること

ニューラルネットワークの作成，MNIST データセットを用いた学習とデータの分類

1．パッケージのインポートと TensorFlow のバージョン確認

```
from __future__ import absolute_import, division, print_function, unicode_literals
import tensorflow.compat.v2 as tf
import tensorflow_datasets as tfds
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')   # Suppress Matplotlib warnings
tf.enable_v2_behavior()
from tensorflow.keras import backend as K
K.clear_session()

print(tf.__version__)
```

```
2.8.0
```

2．MNIST データセットのロード

tensorflow_datasets の loadで，「batch_size = -1」を指定して，一括読み込みを行っている.

```
[4] mnist, mnist_info = tfds.load('mnist', with_info = True, shuffle_files=True, as_supervised=True, batch_size = -1)
```

3．データセットの中の画像を表示

MatplotLib を用いて，0 番目の画像を表示する

```
[5] import matplotlib.pyplot as plt
```

実行するには Google アカウントによるログインが必要

最後まで続ける

人工知能による分類結果

それぞれの数値の中で、一番大きいものはどれか？

```
[19] m.predict(ds_test[0]).argmax(axis=1)

     array([2, 0, 4, ..., 8, 0, 5])
```

- **分類結果は「2, 0, 4 ... 」のように表示**

学習では、乱数が使用されるので、実行ごとに結果が変わる場合がある

精度の確認

１１．学習（訓練）

学習（訓練）は fit メソッドにより行う． 学習データを投入する．

```
EPOCHS = 20
history = m.fit(ds_train[0], ds_train[1],
                epochs=EPOCHS,
                validation_data=(ds_test[0], ds_test[1]),
                verbose=1)
```

Epoch 1/20

途中省略

```
l_crossentropy: 0.0020 - accuracy: 0.9994 - val
```

④ accuracy のところが精度

1 に近いほど精度が良い

実験１

（仮説）**エポック数**（学習の繰り返し回数）は、

いまは　20.

これを**増やすと**、**精度**は上がるかも。

① まず、50に書き換え

```
EPOCHS = 50
history = m.fit(ds_train[0], ds_train[1],
                epochs=EPOCHS,
                validation_data=(ds_test[0], ds_test[1]),
                verbose=1)
```

② 実行．「９．モデルの作成と確認とコンパイル」，「１０．モデルのビジュアライズ」，「１１．学習（訓練）」のコードセルを順に実行

```
al_crossentropy: 0.0032 - accuracy: 0.9988 - va
```

③「１１．学習（訓練）」の実行結果の accuracy のところで，精度を確認
1 に近いほど精度が良い

実験２

（仮説）**ユニット数**を変えると、**精度**は変化するかも。

② 実行

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation

m = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28, 1)),
    tf.keras.layers.Dense(units=1000, activation=tf.nn.relu),
    tf.keras.layers.Dense(units=10, activation=tf.nn.softmax)
])

print(m.summary
```

① まず、1000に書き換え

```
[1
from tensorflow.keras.utils import plot_model
import pydot
plot_model(m)
```

③ その下のコードセルを「１１．学習（訓練）」のところまで順に実行

```
[14] m.compile(loss=tf.keras.losses.sparse_categorical_crossentropy,
               metrics=['sparse_categorical_crossentropy', 'accuracy'],
               optimizer=tf.keras.optimizers.Adam(learning_rate=0.001))
```

```
from tensorflow.keras.utils import plot_model
import pydot
plot_model(m)
```

```
[32] EPOCHS = 50
     history = m.fit(ds_train[0], ds_train[1],
                     epochs=EPOCHS,
                     validation_data=(ds_test[0], ds_test[1]),
                     verbose=1)
```

④ accuracy のところで，精度を確認
1 に近いほど
精度が良い

```
cal_crossentropy: 0.0016 - accuracy: 0.9997 -
```

考察の例

- **繰り返し回数**は２０回で足りている（２０回でも**学習不足**は**ない**）という場合もある

- ユニット数は，６４で十分に足りているという場合もある．「増やせばよい」というものではない