

13-1 第 1 3 回の内容

(情報システム工学特論)

URL: <https://www.kkaneko.jp/a/cs/index.html>

金子邦彦



第13回の内容

- **ニューラルネットワーク**の作成, **学習**, **検証**,
利用を行う**プログラム**を見る

※ プログラムの詳細な説明は行わない

13-2 ニューラルネットワーク を用いた分類

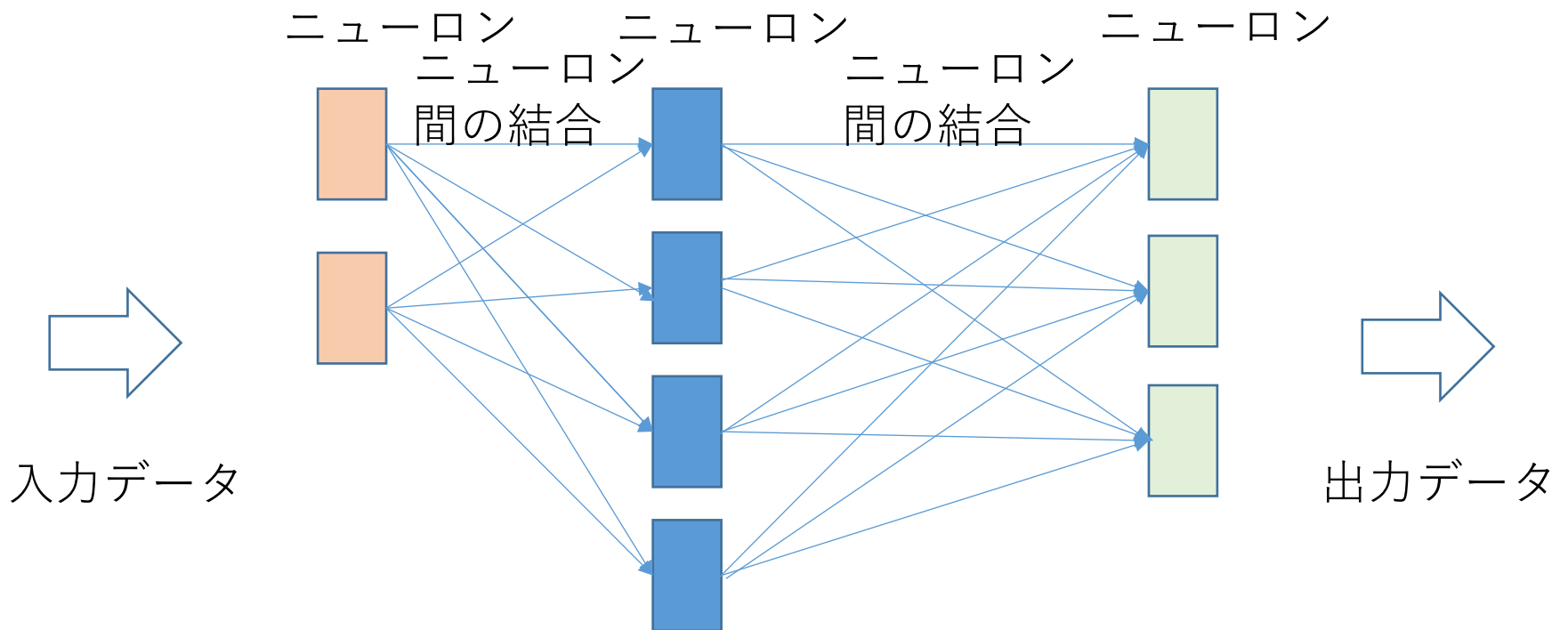
(情報システム工学特論)

URL: <https://www.kkaneko.jp/a/cs/index.html>

金子邦彦

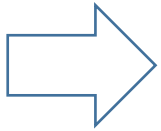


層が直列になっているニューラルネットワーク

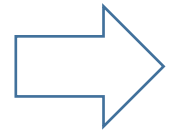
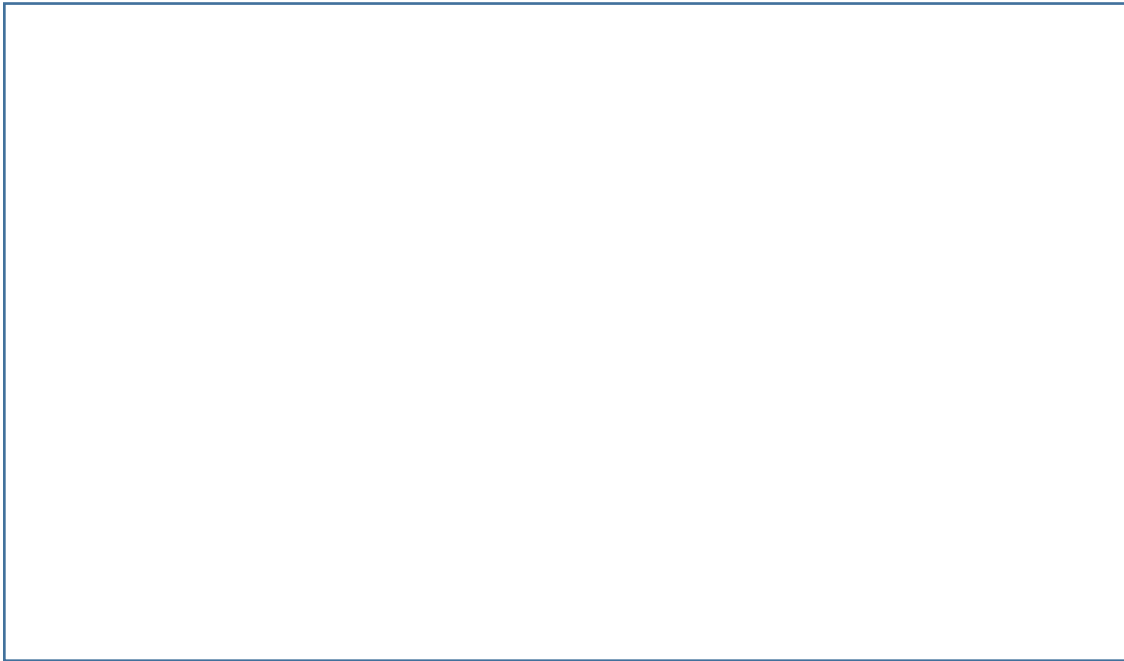


データは入力から出力の方向へ

3種類の中から1つに分類する場合



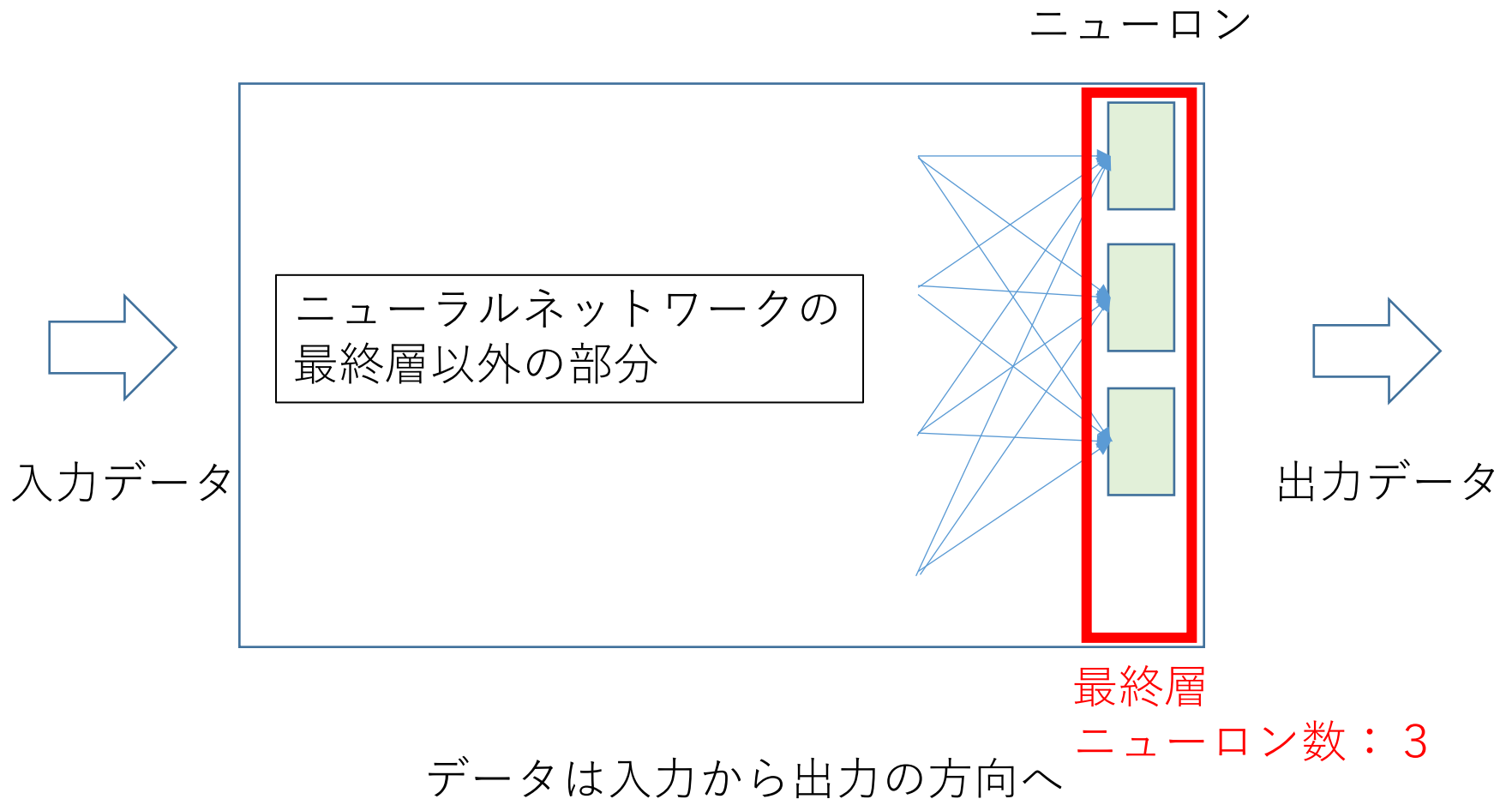
入力データ



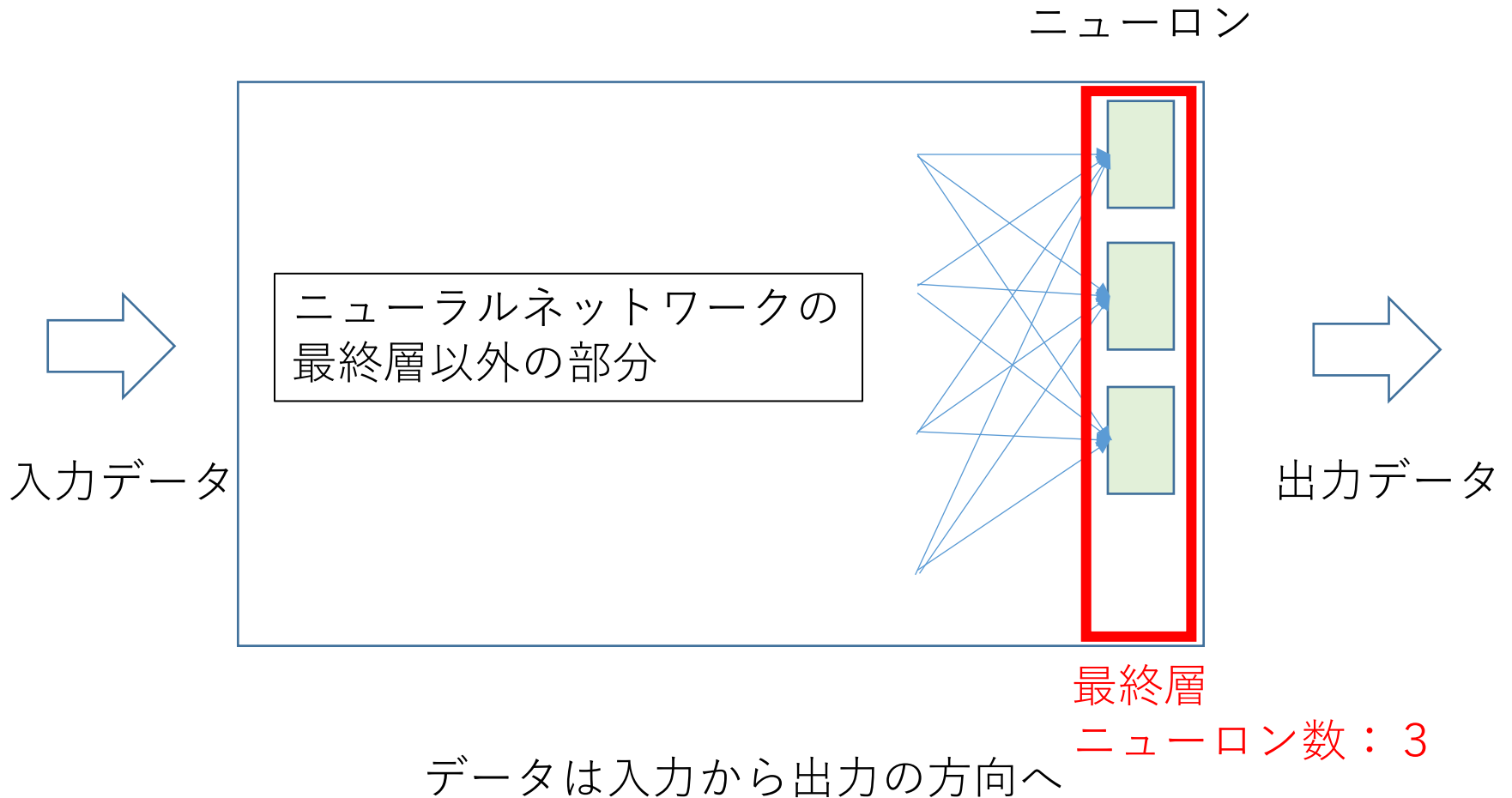
出力データ

0 または **1**
または **2**

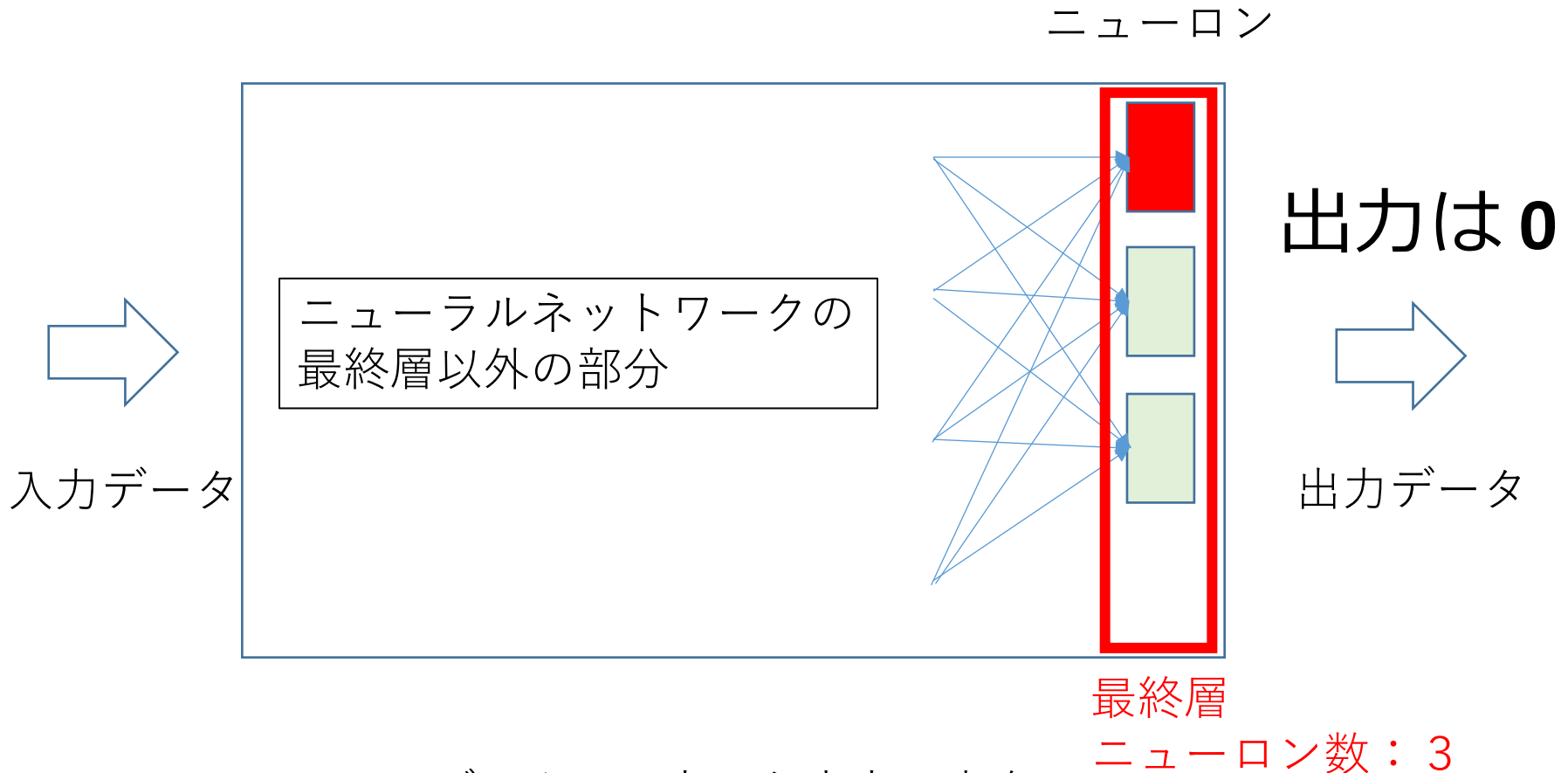
3種類の中から1つに分類する場合



最終層について、1つが強く 活性化するように調整

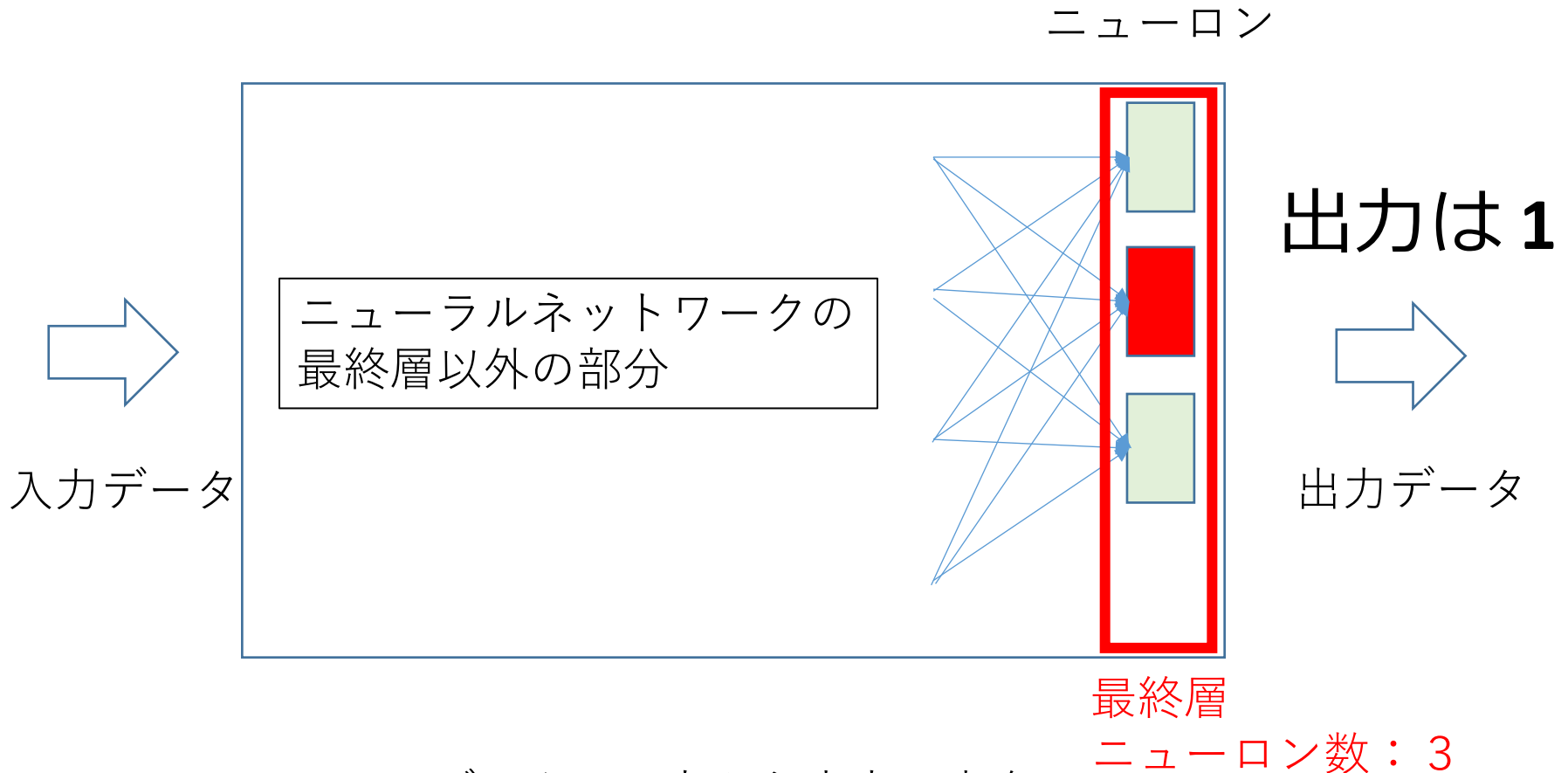


最終層について、1つが強く 活性化するように調整



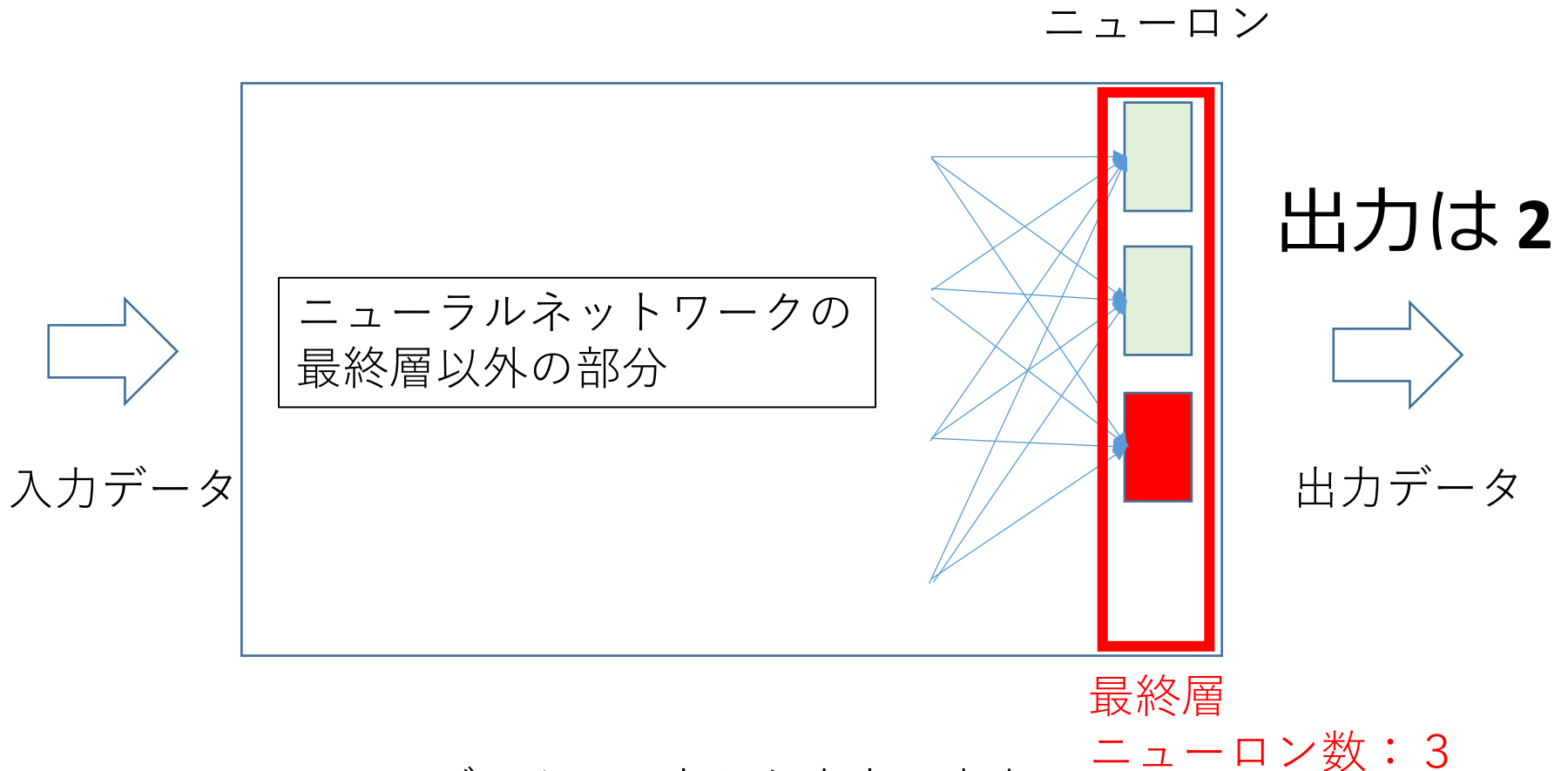
データは入力から出力の方向へ

最終層について、1つが強く 活性化するように調整



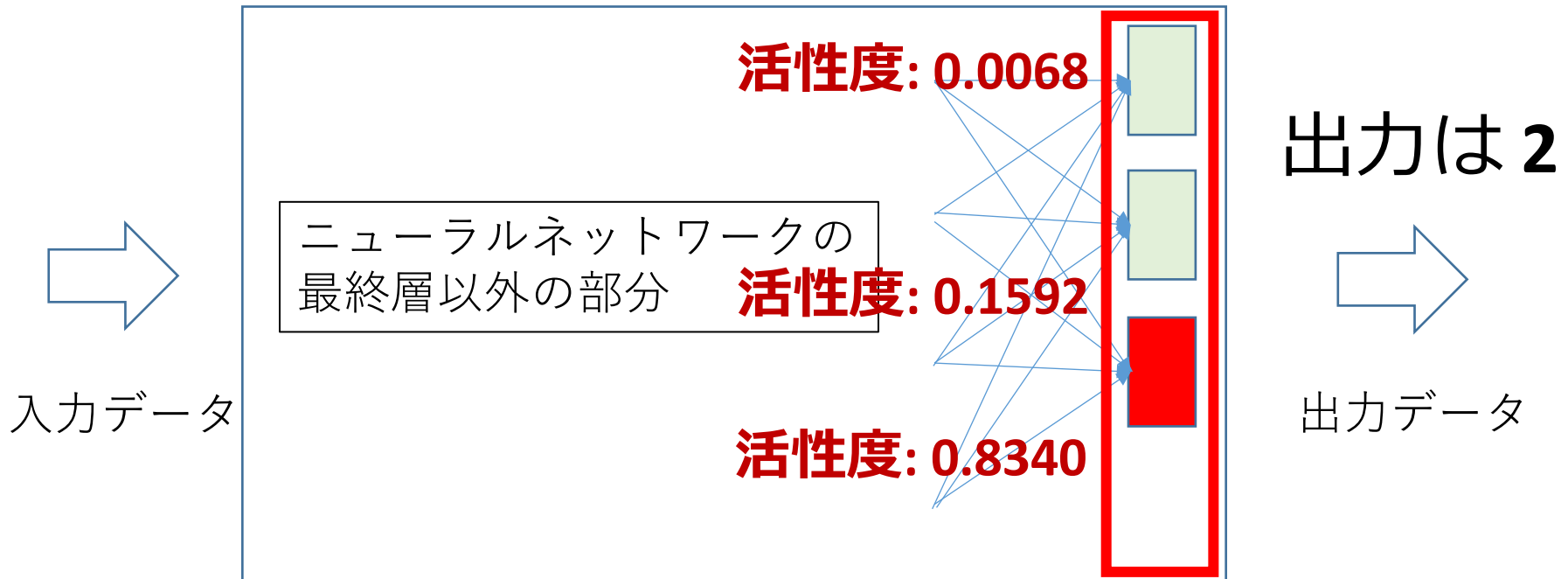
データは入力から出力の方向へ

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

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、**活性化度**は 0 から 1 のような数値である。
最も**活性化度の値が高いもの**が選ばれて、分類結果となる

正解と誤差



正解は 2
であるとする

活性度: 0.0068



誤差: 0.0068



あるべき値: 0

活性度: 0.1592



誤差: 0.1592



あるべき値: 0

活性度: 0.8340



誤差: - 0.1760



あるべき値: 1

誤差をもとに、結合の重みを自動調節

ニューラルネットワークを用いた分類

ニューラルネットワークを分類に使うとき

- **最終層のニューロンで、最も活性度の値の高いもの**が選ばれて、分類結果となる
- そこには**誤差**がある

13-3 画像データと次元

(情報システム工学特論)

URL: <https://www.kkaneko.jp/a/cs/index.html>

金子邦彦



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

1次元と2次元の配列 (アレイ)



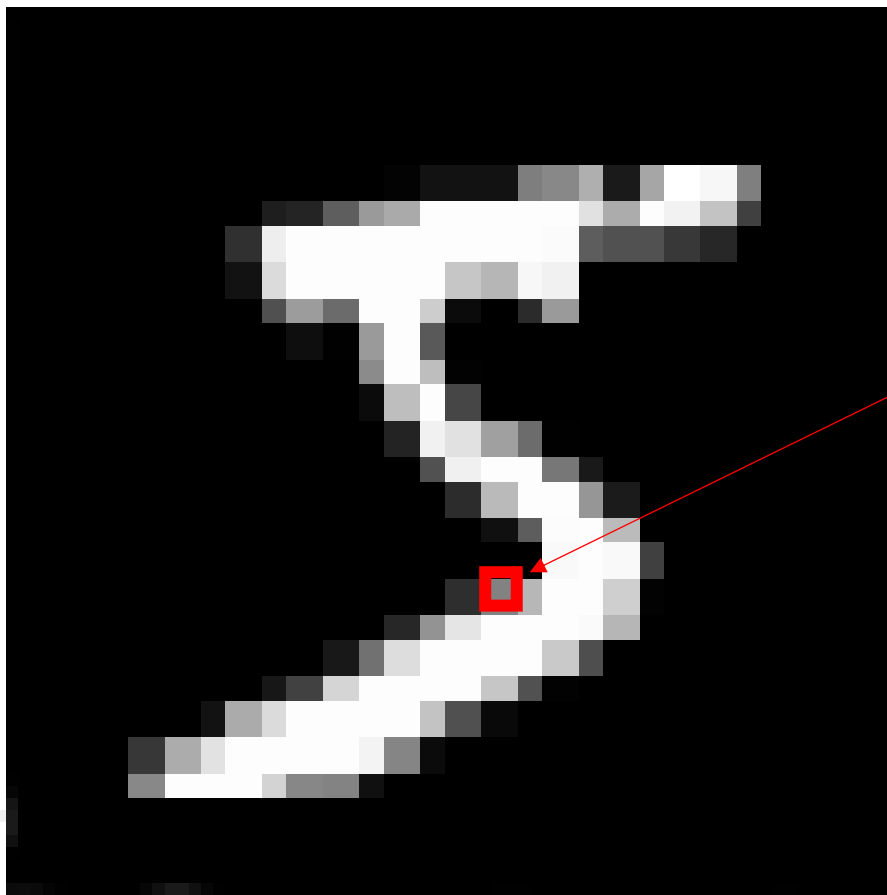
0	1	2	3
180	20	250	40

1次元の配列

0, 0	0, 1	0, 2	0, 3
4	1	2	4
1, 0	1, 1	1, 2	1, 3
5	8	2	8

2次元の配列

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**

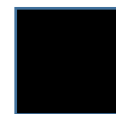
画素

画素値



白

255

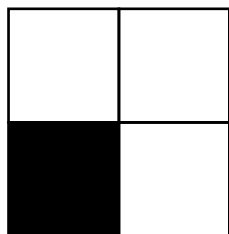


黒

0

画素値は, **画素**の明るさに応じた **0 から 255 の数値**

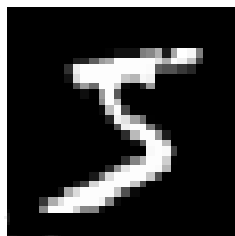
画像のバリエーション



2×2
次元数は 2

数値の個数: 4

画像のサイズ: 2×2
(画素は白または黒)



28×28
次元数は 2

数値の個数: 784

画像のサイズ: 28×28
(画素は濃淡)

13-4 ニューラルネットワーク を作成するプログラム

(情報システム工学特論)

金子邦彦



ニューラルネットワークのプログラムで行うこと



1. ニューラルネットワークの作成

ニューロンの数, ニューロンの種類などの設定

2. 学習に関する設定

使用する最適化手法, 誤差算出法 (損失関数) などの設定

3. 学習の実行

教師データを使用. 結合の重みの変化

4. 検証

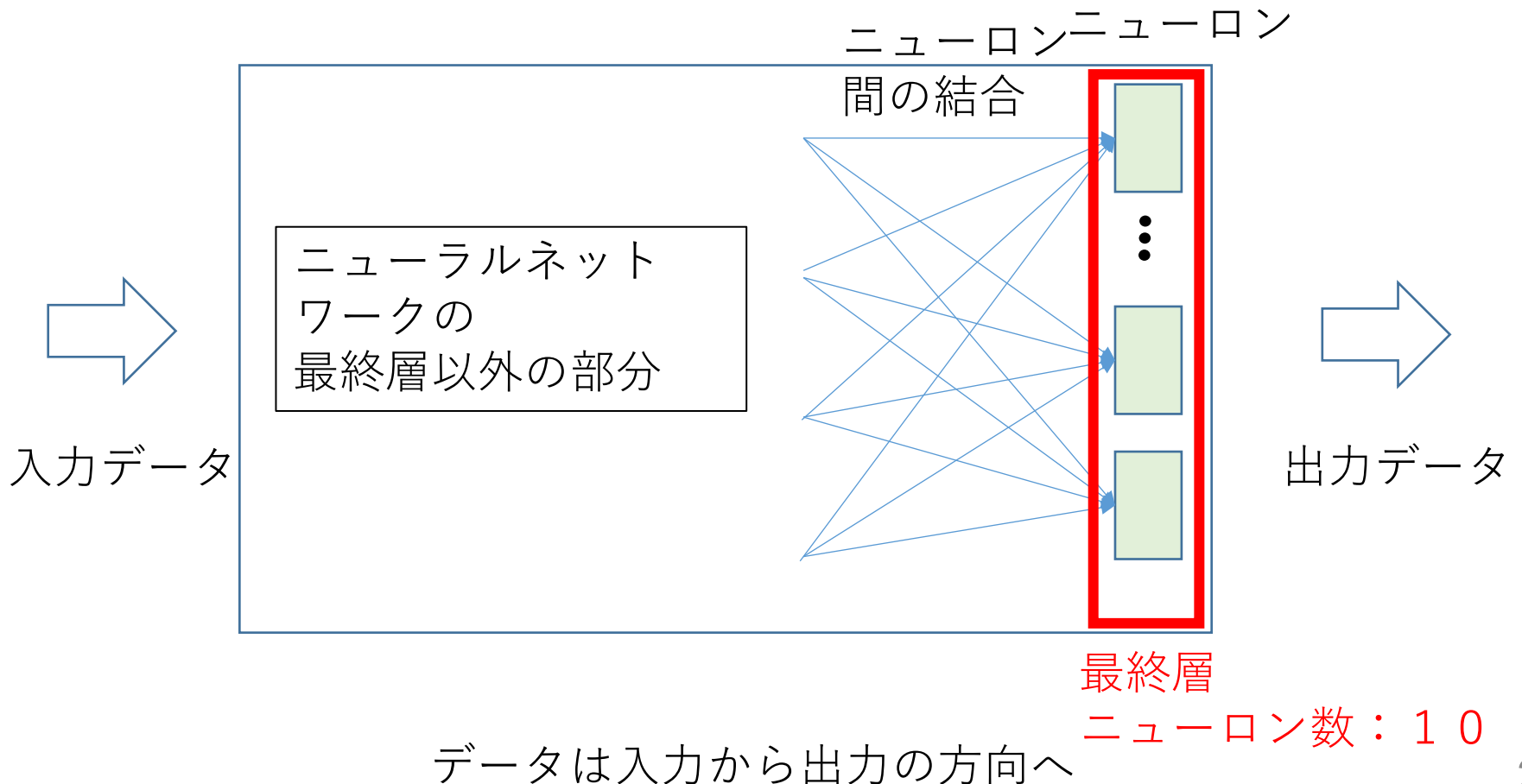
検証用のデータを使用. 学習不足や過学習などの検証

5. 実際の使用

さまざまな用途に使用

今回の授業では、1, 2, 3, 4, 5の一連の流れを説明

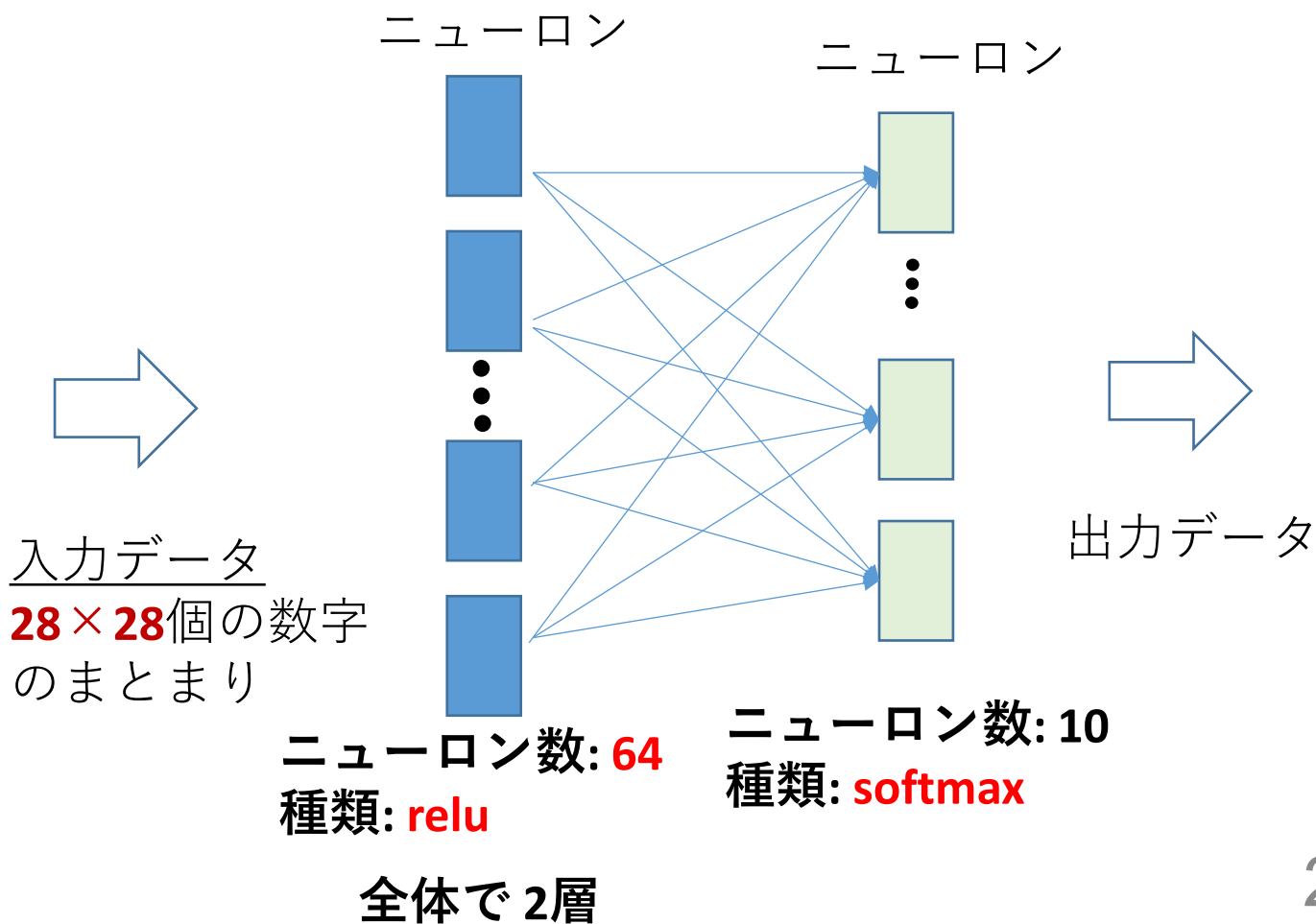
10種類の中の1つに分類を行うニューラルネットワーク



ここで作成するニューラルネットワーク



- 入力： **28×28**個の数値のまとめり
- 出力： 「**28×28**個の数字のまとめり」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の作成

ニューラルネットワークの作成では、次を設定する

- 入力データでの数値の個数
- ニューロンの数（層ごと）
- ニューロンの種類（層ごと）

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**

```
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

13-5 学習に関する設定を行う プログラム

(情報システム工学特論)

金子邦彦



ニューラルネットワークの学習

ニューラルネットワークの出力と、教師データの正解との誤差を最小にするという最適化を行う

- 最適化手法のバリエーション

Adadelta, Adagrad, Adam, RMSprop, SGD など

- 誤差の算出法（損失関数）のバリエーション

binary_crossentropy, categorical_crossentropy, cosine_similarity, kld, kullback_leibler_divergence, mae など

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

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)
])
```

先ほどと
同じプログラムを
参考のため掲載

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

誤差の算出法（損失関数）は **sparse_categorical_crossentropy**,
最適化手法は **Adam**

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

実行により、何か表示されるということはない

13-6 学習の実行, 検証を行う プログラム

(情報システム工学特論)

URL: <https://www.kkaneko.jp/a/cs/index.html>

金子邦彦



大量のデータがある場合には、教師データと検証用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

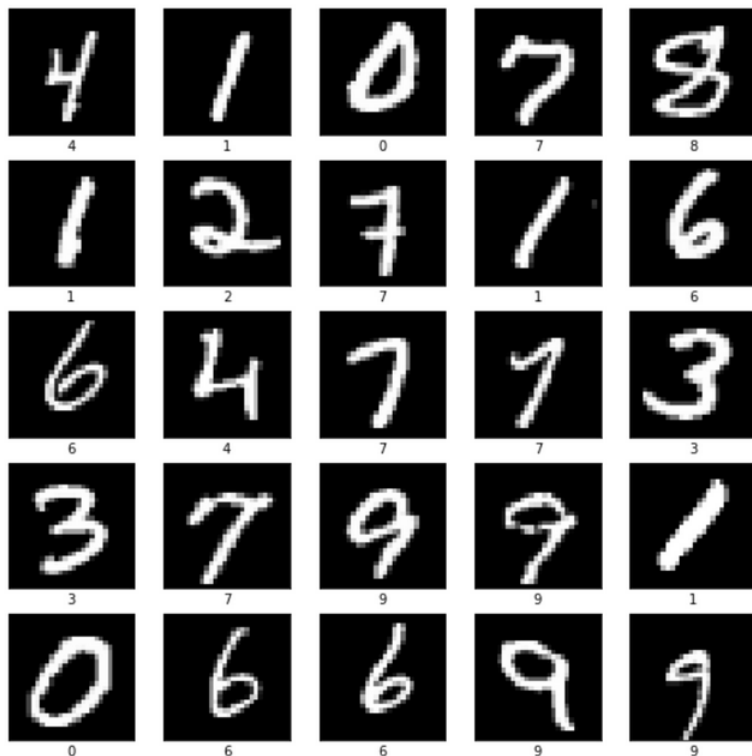
うち **60000** 枚 → **教師データ** として使用

うち **10000** 枚 → 検証で使用

のように考えることができる

※ 教師データと検証用データは別々であることが重要。
60000 や 10000 のような枚数は、自由に決めることができるもの。

教師データの例



画像 60000枚
(うち一部)

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

正解 60000個
(うち一部)

教師データの指定

```
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

検証用データを指定することで、
検証も行われる

```
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
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

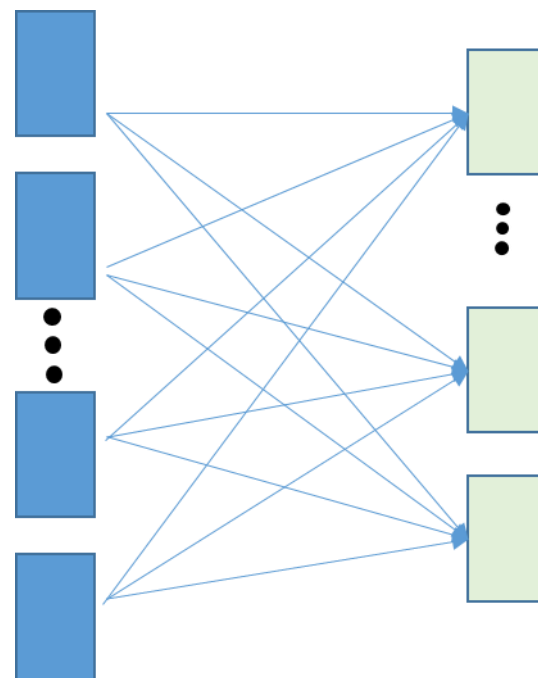
結合の重みの表示

プログラム

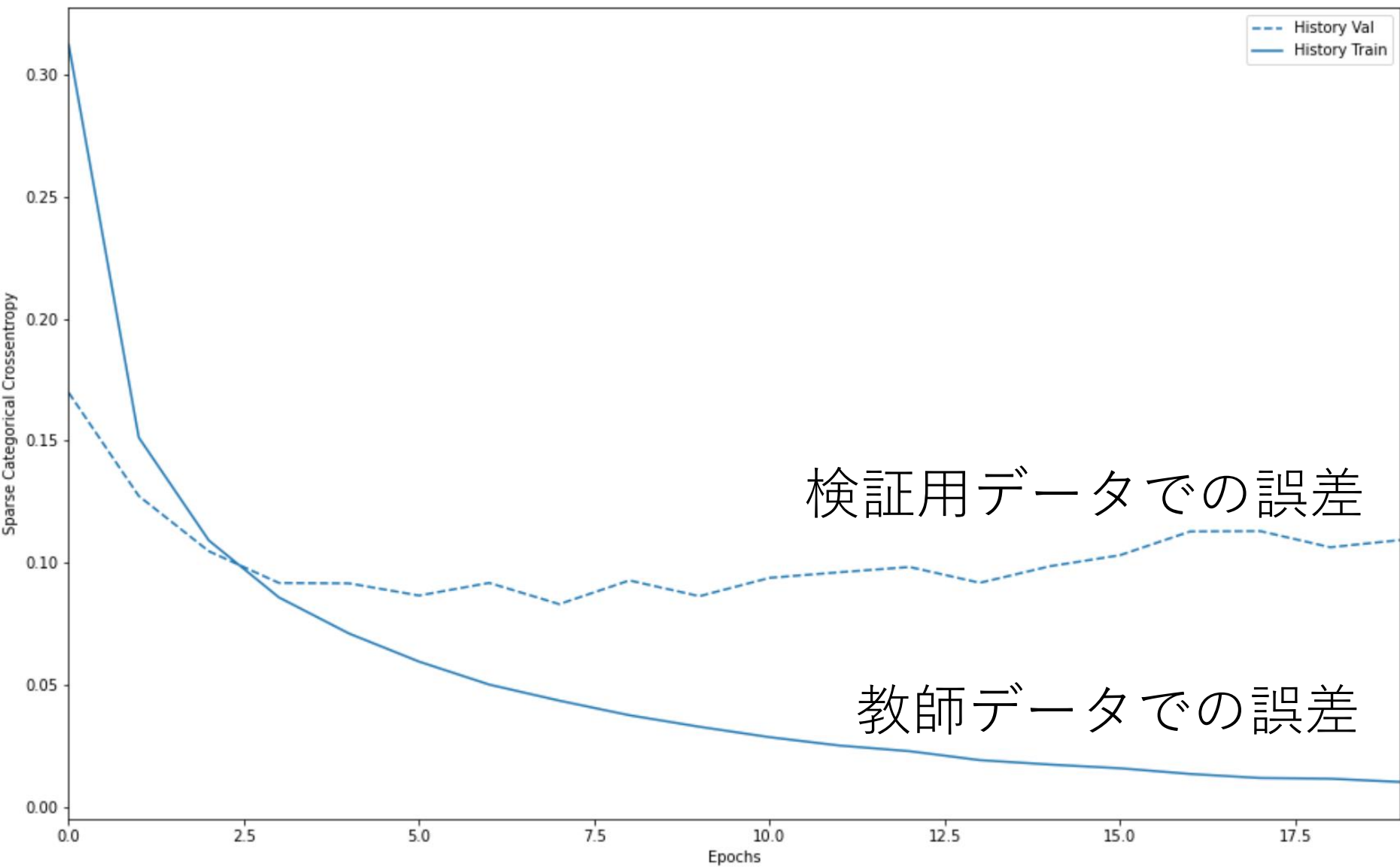
```
m.get_weights()[2]
```

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
結合の重みが変わる



検証用データでの誤差

教師データでの誤差

13-7 ニューラルネットワーク を用いた分類

(情報システム工学特論)

URL: <https://www.kkaneko.jp/a/cs/index.html>

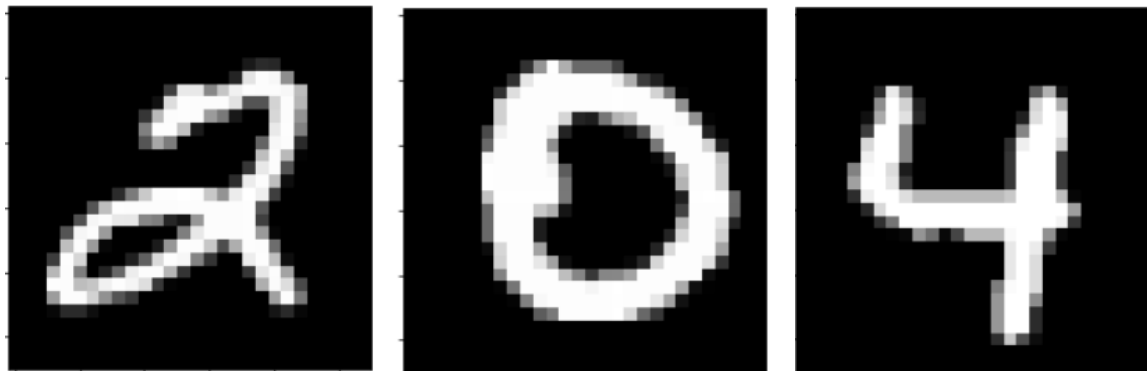
金子邦彦



ニューラルネットワークによる分類



入力
多数の画像



など

上の入力について分類を行うプログラム

```
m.predict( ds_test[0] )
```

実行結果

```
print(m.predict(ds_test[0]))  
[[1.36119690e-16 4.05257840e-22 1.00000000e+00 ... 1.98569462e-18  
 3.22409127e-10 8.19054142e-18]  
 [1.00000000e+00 0.00000000e+00 1.14557860e-23 ... 2.54518175e-19  
 3.64565147e-19 4.89561556e-24]  
 [1.27674932e-20 1.89606327e-19 3.47646780e-16 ... 8.36853416e-12  
 2.97038632e-14 1.05294404e-07]  
 ...  
 [2.34036176e-11 2.65855048e-19 8.73036257e-08 ... 2.96257529e-16  
 9.9999881e-01 5.48949601e-12]  
 [1.00000000e+00 1.06714619e-21 1.21512200e-09 ... 8.56583244e-15  
 7.37025681e-21 3.40695392e-15]  
 [5.88137927e-19 1.16241944e-18 4.45531422e-17 ... 3.85180054e-16  
 2.41860948e-10 3.87167495e-15]]
```

最終層の10個の
ニューロンの活性度
(入力の画像の枚数分)
→ 活性度の最も高い
ニューロンが分類結果

プログラムは、次で公開

<https://www.kkaneko.jp/db/classify/tutorials.html>

Google Colaboratory へのリンク

(利用には Google アカウントが必要)

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