

医療とAI・ビッグデータ応用

③MNISTの読み込みと前処理

本スライドは、自由にお使いください。
使用した場合は、このQRコードからアンケート
に回答をお願いします。



統合教育機構
須藤毅顕

この授業では何をしていたか？

深層学習で画像分類をしたい

深層学習(教師あり機械学習)の復習

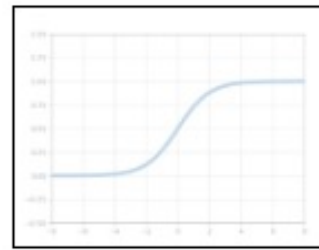
データを用意する

x(特徴量データ) y(正解データ)

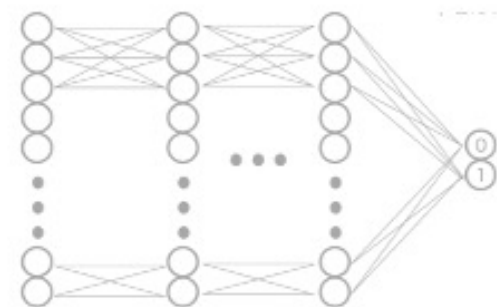


学習させる

ロジスティック回帰分析



ニューラルネットワーク



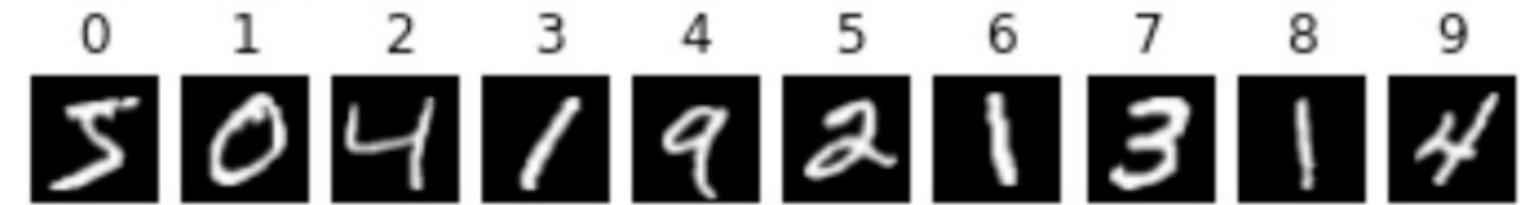
評価する (分類、予測など)

病気が否か
犬か猫か

画像を配列の数値データにして 学習させたい

Pythonには機械学習を実践するために多くの画像セットが用意されている

MNIST : 0~9の文字画像のデータ



FASHION-MNIST : 白黒の洋服の画像データ

0 : T-shirt/top, 1 : Trouser, 2 : Pullover, 3 : Dress, 4 : Coat, 5 : Sandal
6 : Shirt, 7 : Sneaker, 8 : Bag, 9 : Ankle boot



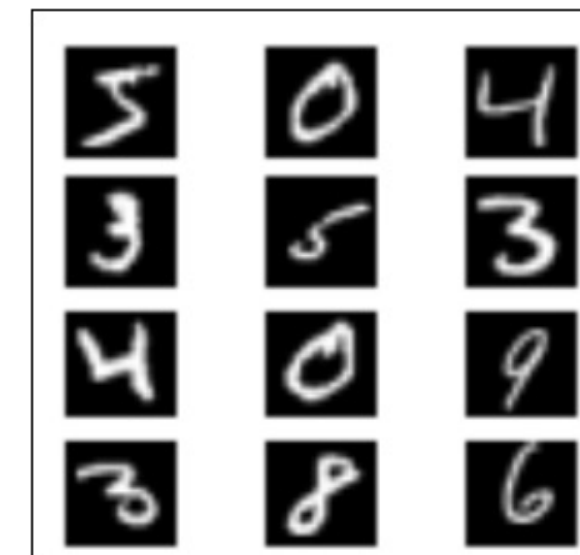
CIFAR10

0 : airplane, 1 : automobile, 2 : bird, 3 : cat, 4 : deer, 5 : dog
6 : frog, 7 : horse, 8 : ship, 9 : truck



画像を読み込んで学習出来るデータ(配列)にする

画像データ



配列データ

```
x_train  y_train
[[150, 70, 60], [[1],
[120, 40, 35],  [0],
[144, 45, 40],  [1],
[162, 56, 50],  [1],
[98, 40, 32],   [0],
[128, 59, 35],  [0],
[155, 77, 45]]  [1]]
```

```
from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

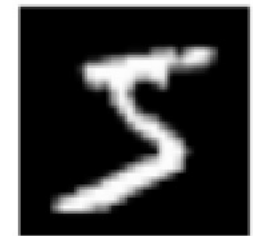
60000個

60000個

10000個

10000個

mnistのdataを読み込む



5



0



4



1

⋮



6



8

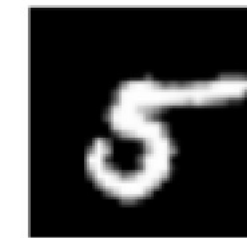


7



2

⋮



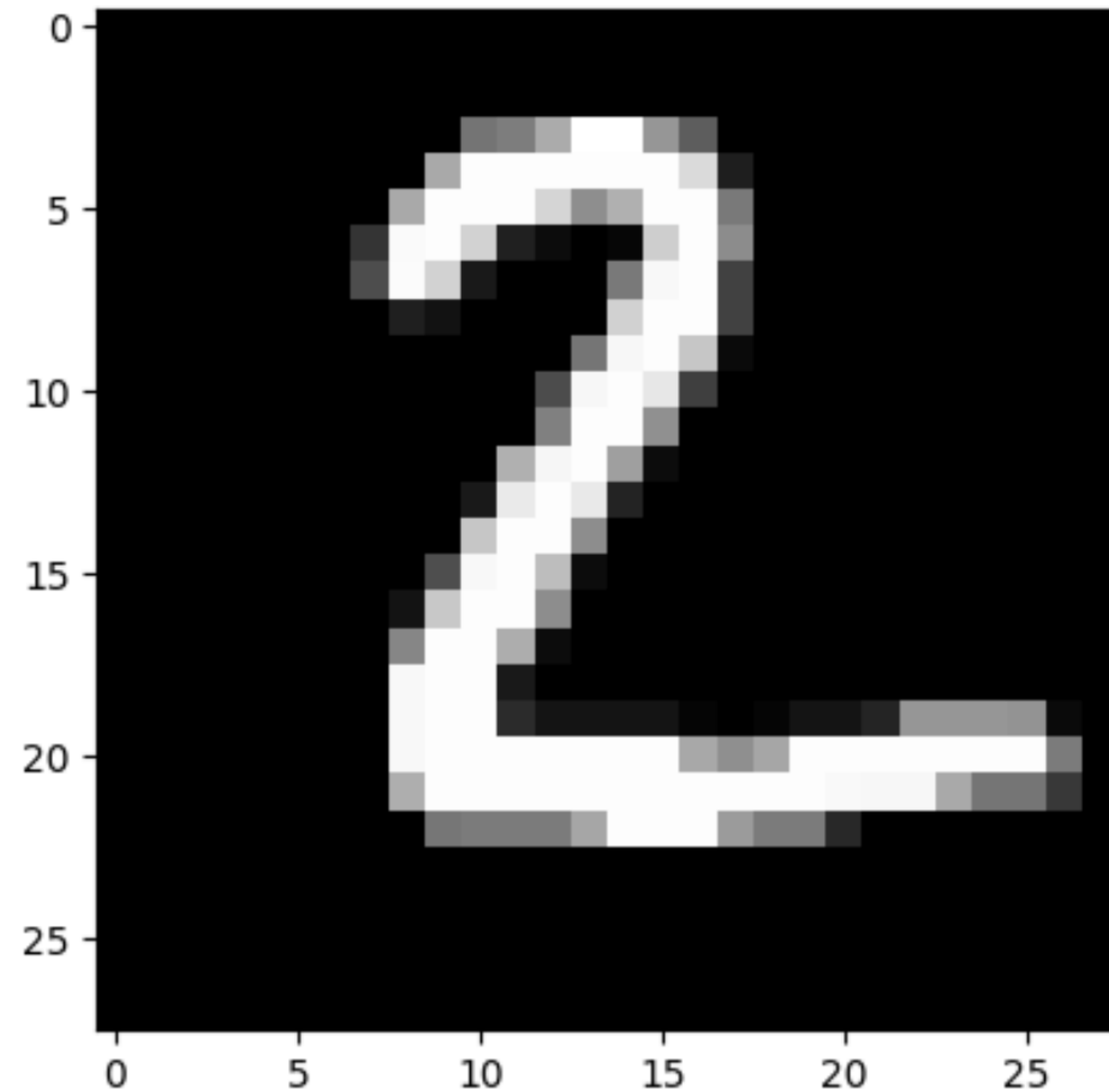
5



6

x_train : 60000枚の画像の配列データ
y_train : 60000枚の正解の数字の配列データ
x_test : 10000枚の画像の配列データ
y_test : 10000枚の正解の数字の配列データ

```
import matplotlib.pyplot as plt
plt.imshow(x_test[1], 'gray')
plt.show()
```



画像を描画する

matplotlib(描画ライブラリ)

‘gray’で白黒を指定

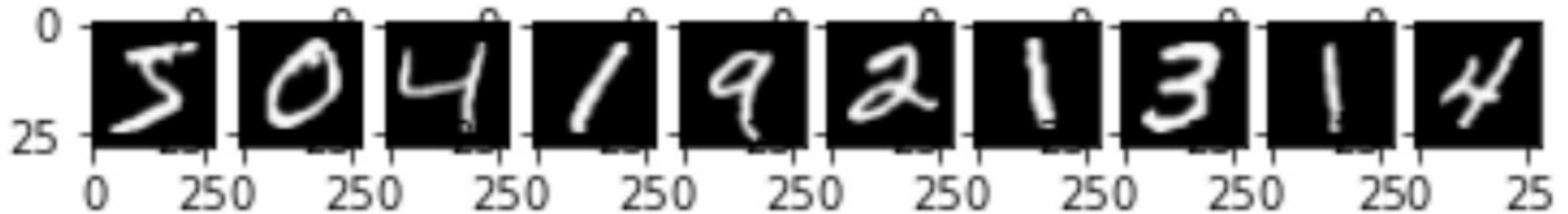
```
print(y_test[1])
```

2

数字の2らしい

10個並べてみる

```
for i in range(10):  
    plt.subplot(1,10,i+1)  
    plt.imshow(x_train[i], 'gray')  
plt.show()
```



for文とrange関数

```
for i in range(1,10,2):  
    print(i)
```

1
3
5
7
9

```
for i in range(5):  
    print(i)
```

0
1
2
3
4

=

```
for i in range(0,5,1):  
    print(i)
```

0
1
2
3
4

for (変数) in range(A,B,C):
 (処理内容)

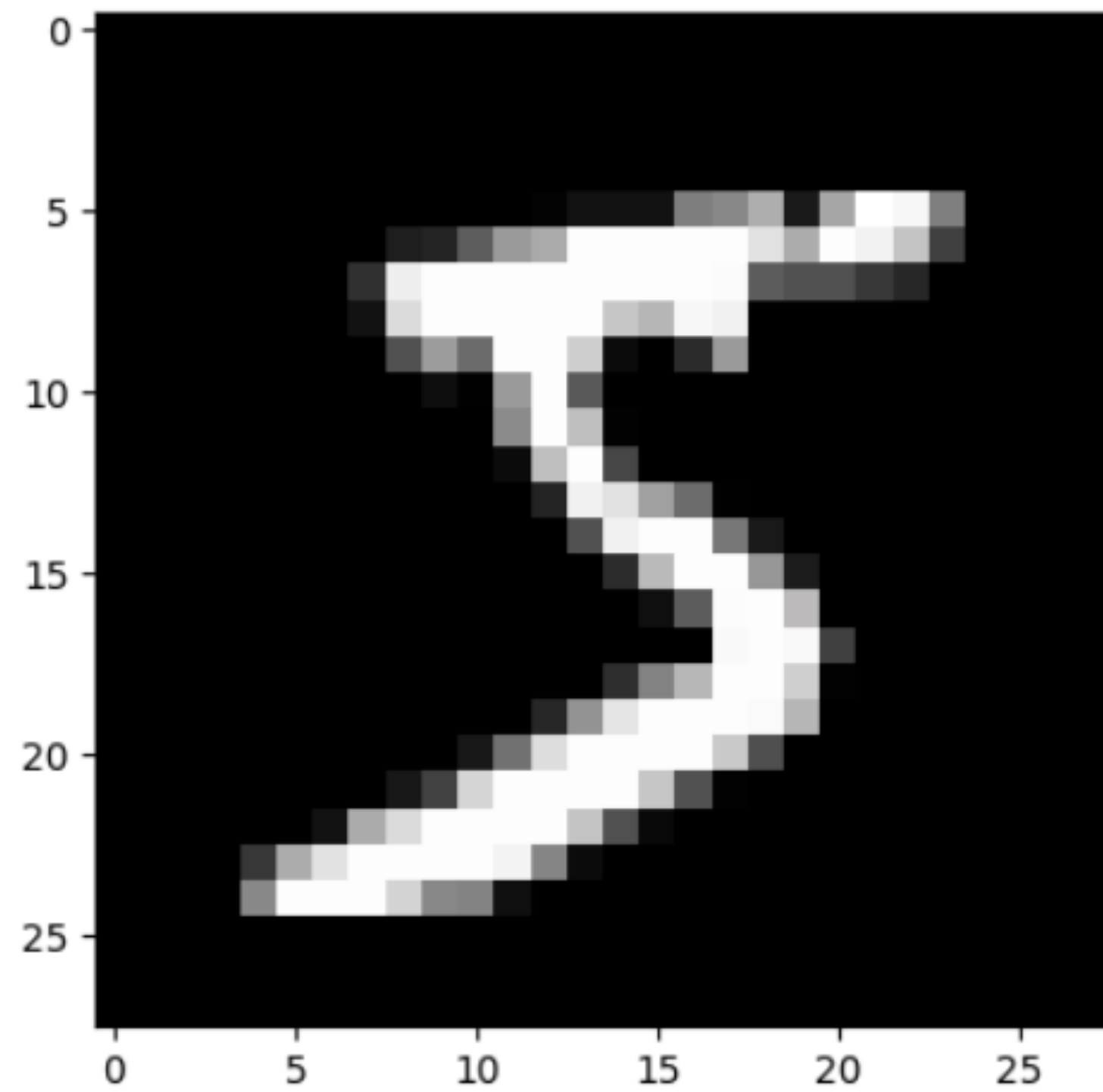
A(以上)からB(未満)でC刻みに変数に代入

この例だと1から1つ飛ばしで9まで
iは順に1,3,5,7,9が代入されて処理

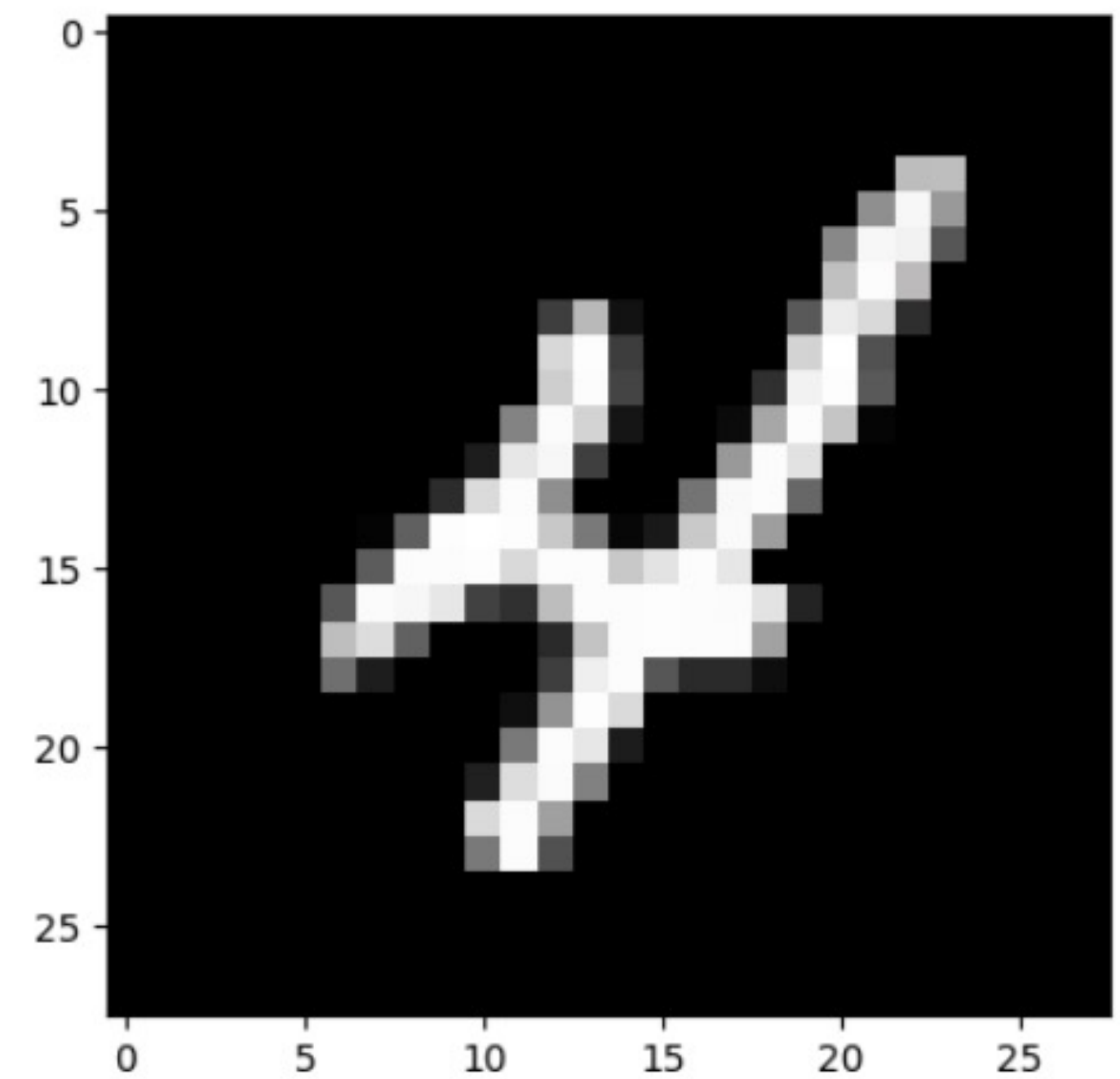
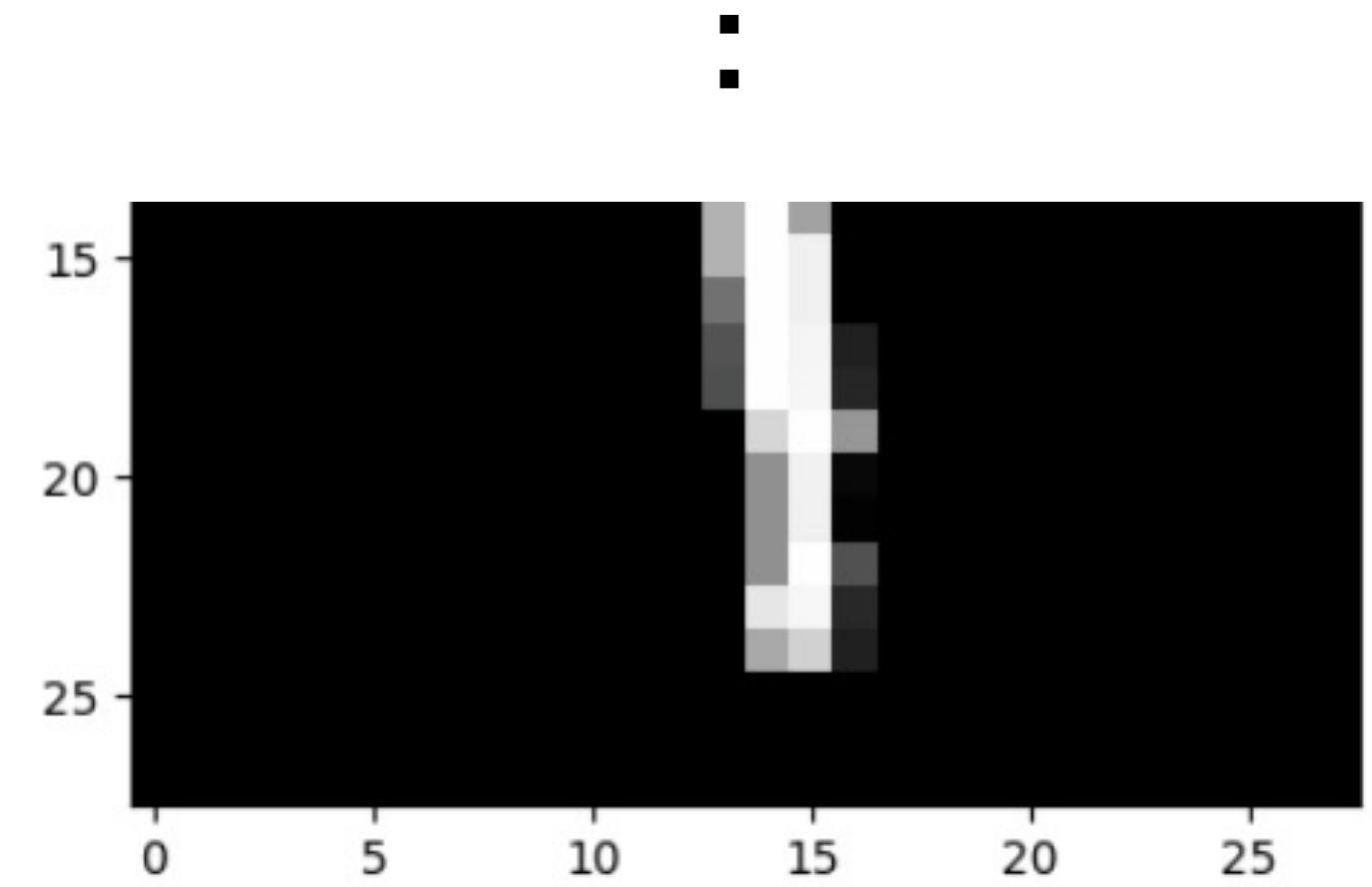
range(B)のように数字1つにすると
開始のAを0、刻み幅Cを1の設定で省略出来る

for文

```
for i in range(10):  
    plt.imshow(x_train[i], 'gray')  
    plt.show()
```



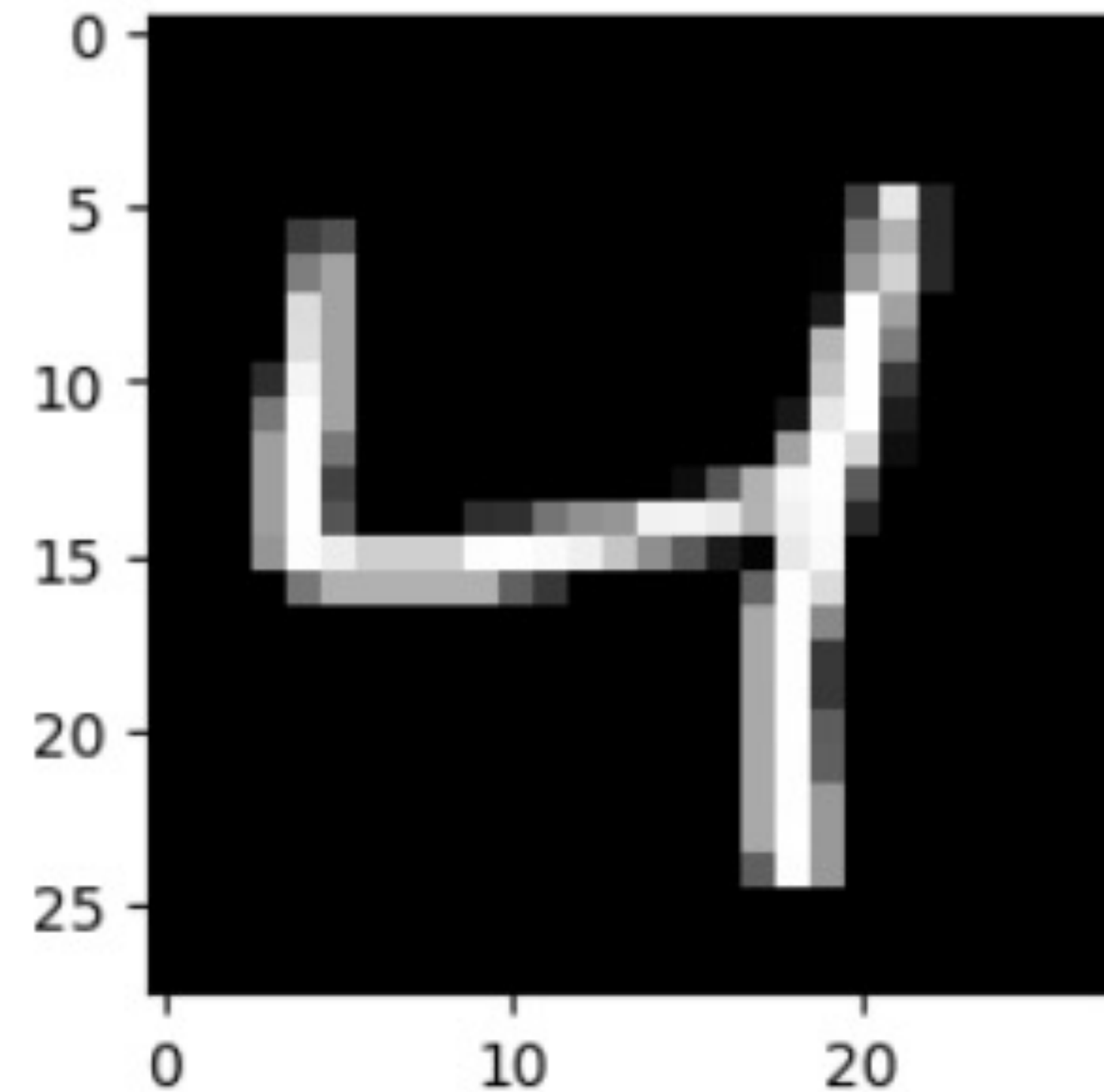
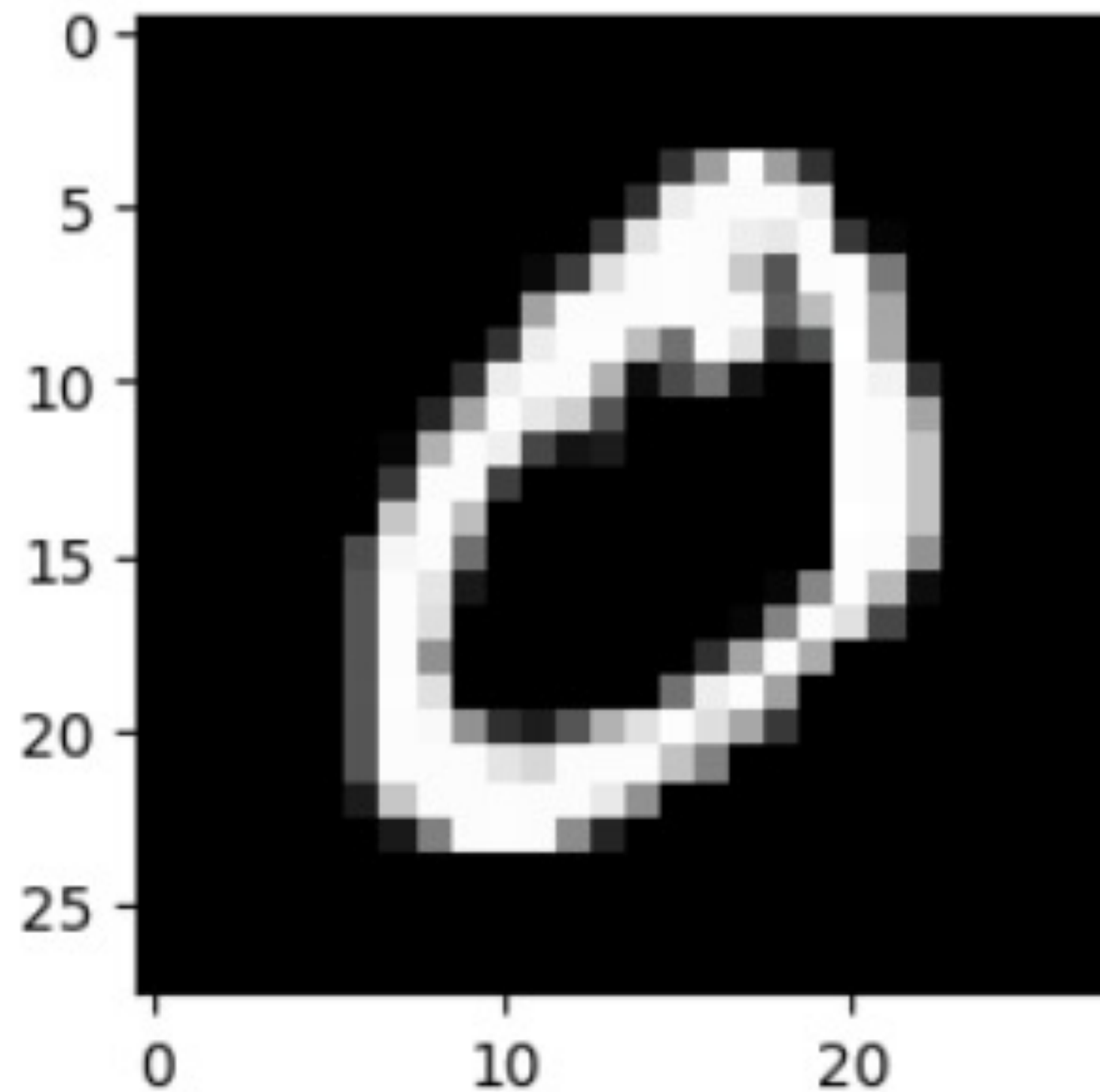
■
■



plt.subplotは図を並べる plt.subplot(縦,横,左上から何番目か)

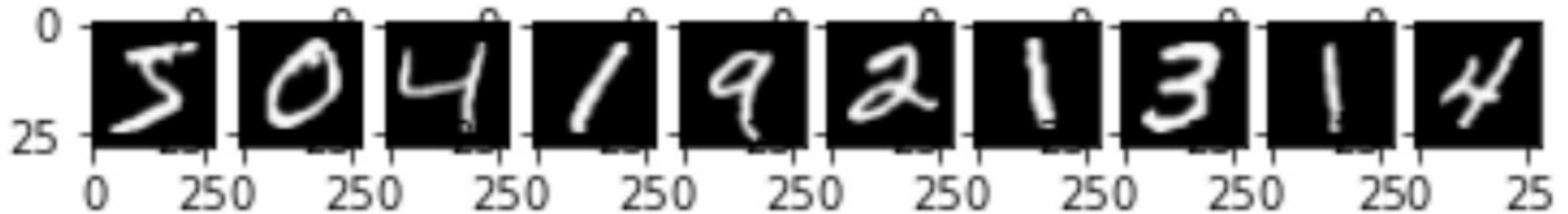
```
[8] plt.subplot(1,2,1)          ←縦1、横2に並べる内の1つ目の宣言
     plt.imshow(x_train[1], 'gray')
     plt.subplot(1,2,2)          ←縦1、横2に並べる内の2つ目の宣言
     plt.imshow(x_train[2], 'gray')

     plt.show()
```



10個並べてみる

```
for i in range(10):  
    plt.subplot(1,10,i+1)  
    plt.imshow(x_train[i], 'gray')  
plt.show()
```



for文

```
for i in range(10):  
    plt.subplot(1,10,i+1)  
    plt.imshow(x_train[i], 'gray')  
plt.show()
```

plt.subplot(1,10,i+1)
plt.imshow(x_train[i], 'gray')

縦に1つ、横に10個、図を書く。
i=0なので(1,10,1)で1番左の図を指定する



x_train[0]

for文

```
for i in range(10):  
    plt.subplot(1,10,i+1)  
    plt.imshow(x_train[i], 'gray')  
plt.show()
```

plt.subplot(1,10,i+1)
plt.imshow(x_train[i], 'gray')

次がi=1
(1,10,2)で左から2つ目の図を指定する
plt.imshow(x_train[1], 'gray')



for文

```
for i in range(10):  
    plt.subplot(1,10,i+1)  
    plt.imshow(x_train[i], 'gray')  
plt.show()
```

plt.subplot(1,10,i+1)
plt.imshow(x_train[i], 'gray')

最後はi=9
(1,10,10)で左から10個目の図を指定する
plt.imshow(x_train[1], 'gray')



for文

```
for i in range(10):  
    plt.subplot(1,10,i+1)  
    plt.imshow(x_train[i], 'gray')  
plt.show()
```

plt.subplot(1,10,i+1)
plt.imshow(x_train[i], 'gray')

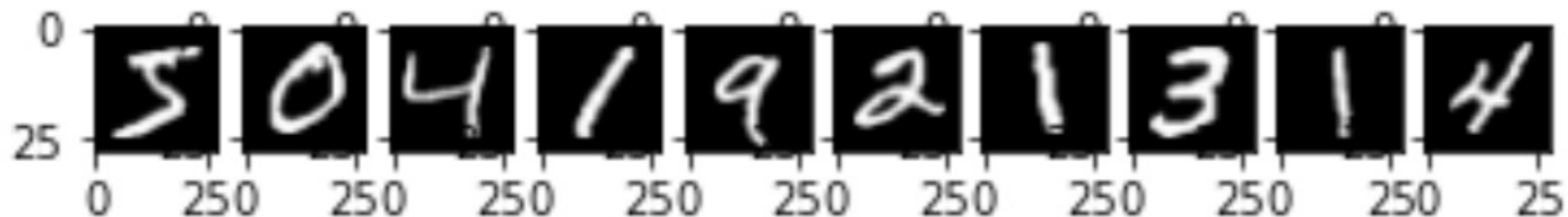
最後はi=9

(1,10,10)で左から10個目の図を指定する

plt.imshow(x_train[1], 'gray')

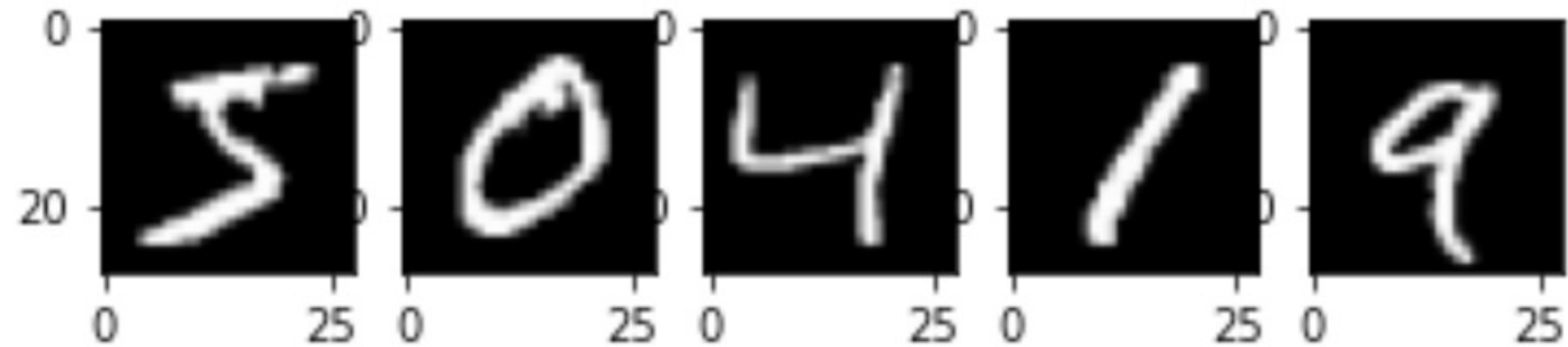


x_train[9]



plt.subplot(2,5,i+1)にすると縦2、横5の図になる

```
for i in range(10):  
    plt.subplot(2,5,i+1)  
    plt.imshow(x_train[i], 'gray')  
plt.show()
```



正解も10個並べてみる

```
print(y_train[0:10])
```

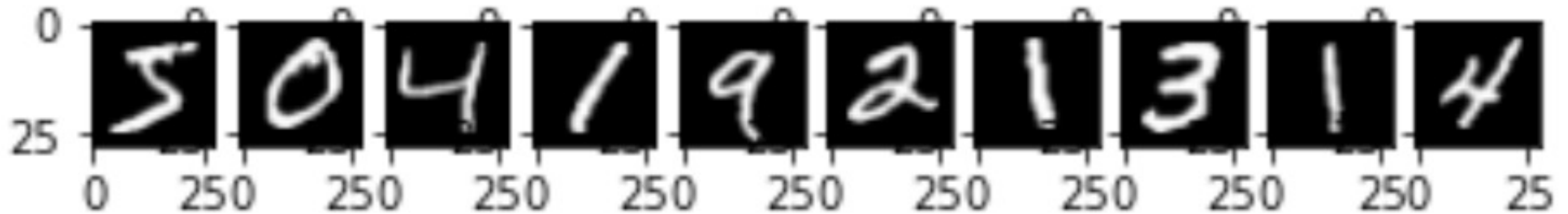
```
[5 0 4 1 9 2 1 3 1 4]
```

配列は[始まりの数字：終わりの数字]で中身(要素)を取り出せる

[0:10]で0から9番目まで！

x_trainとy_trainが特徴量と正解の関係になっている（図でも確認）

```
for i in range(10):  
    plt.subplot(1,10,i+1)  
    plt.imshow(x_train[i], 'gray')  
plt.show()
```



```
print(y_train[0:10])
```

```
[5 0 4 1 9 2 1 3 1 4]
```

深層学習前のデータの整理

x_train (特徴量)

- 画像の2次元の配列を1次元にする
- 正規化する

y_train (正解)

- one-hot encoding

深層学習前のデータの整理

x_train (特徴量)

- ・ 画像の2次元の配列を1次元にする

まだ入力しなくていいです

```
x_train = x_train.reshape((x_train.shape[0], 784))  
x_test = x_test.reshape((x_test.shape[0], 784))  
print(x_train.shape)  
print(x_test.shape)
```

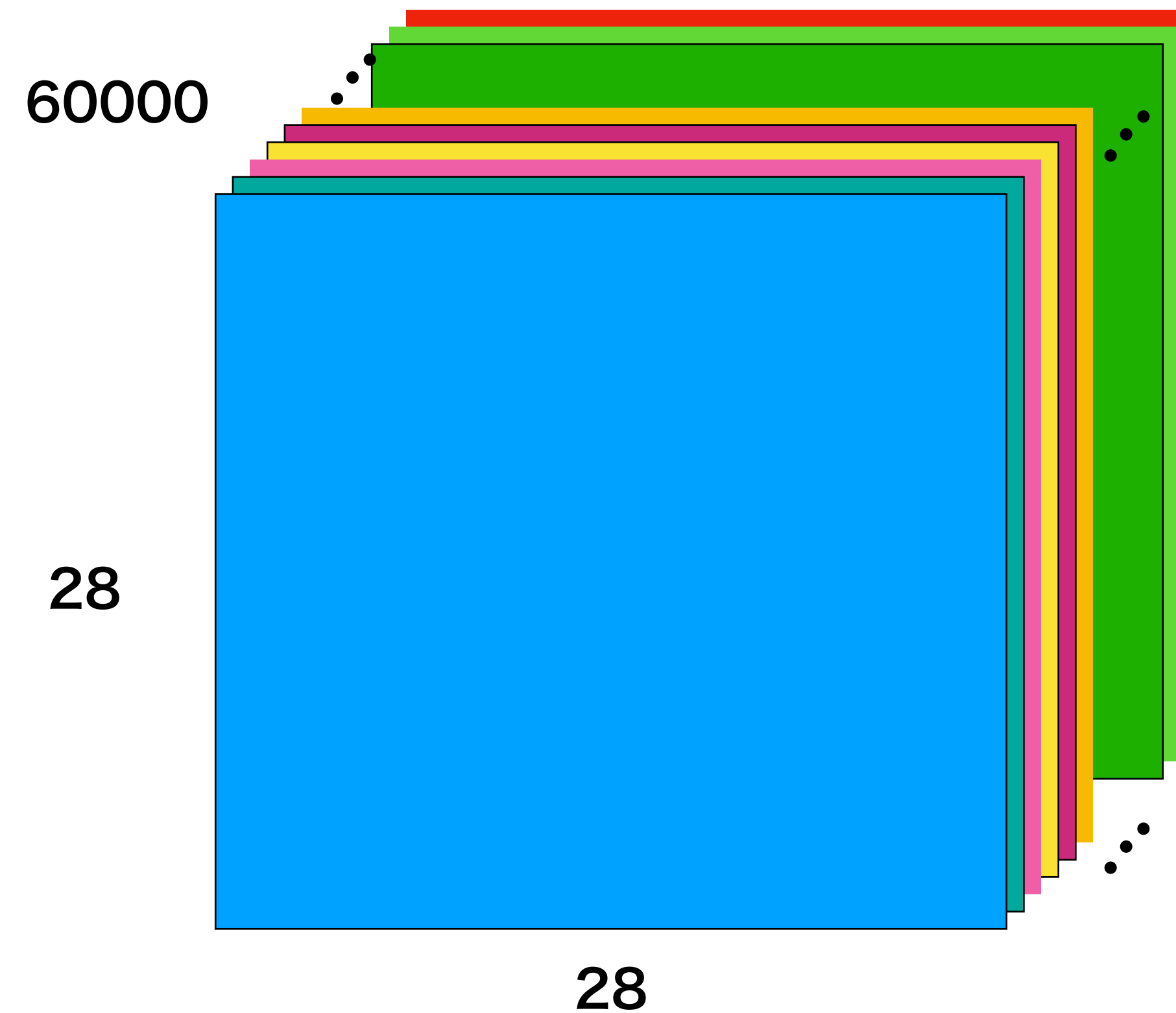
(60000, 784)

(10000, 784)

x_trainのshapeは？

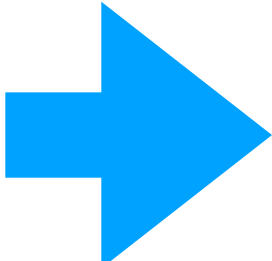
```
print(x_train.shape)  
(60000, 28, 28)
```

x_train[0]のshapeは？



print(x_train[0].shape)は1枚目の画像の配列なので(28,28)となる

画像の2次元配列を1次元配列にしたい

(60000, 28, 28)  (60000, 28 × 28)

$$28 \times 28 = 784$$



画像の2次元配列を1次元配列にしたい

reshape()で配列の形状を変える

```
a = np.array([1,2,3,4,5,6,7,8,])  
print(a)  
[1 2 3 4 5 6 7 8]
```

aを(2,4)に変える

```
a = a.reshape(2,4)  
print(a)  
[[1 2 3 4]  
 [5 6 7 8]]
```

画像の2次元配列を1次元配列にしたい

reshape()で配列の形状を変える

```
a = np.array([1,2,3,4,5,6,7,8,])  
print(a)  
[1 2 3 4 5 6 7 8]
```

aを(2,4)に変える

```
a = a.reshape(2,4)  
print(a)  
  
[[1 2 3 4]  
 [5 6 7 8]]
```

aを(2,2,2)に変える

```
a = a.reshape(2,2,2)  
print(a)
```

```
[[[1 2]  
  [3 4]  
  [5 6]  
  [ 7 8]]]
```

要素の合計が合っていれば
どの形にも変えられる

画像の2次元配列を1次元配列にしたい

reshape()で配列の形状を変える

```
a = np.array([1,2,3,4,5,6,7,8,])  
print(a)  
[1 2 3 4 5 6 7 8]
```

aを(2,4)に変える

```
a = a.reshape(2,4)  
print(a)  
  
[[1 2 3 4]  
 [5 6 7 8]]
```

```
print(x_train.shape)  
(60000, 28, 28)
```

```
x_train = x_train.reshape(60000,784)  
x_test = x_test.reshape(10000,784)
```

```
x_train = x_train.reshape((x_train.shape[0],784))  
x_test = x_test.reshape((x_test.shape[0],784))  
print(x_train.shape)  
print(x_test.shape)
```

```
(60000, 784)  
(10000, 784)
```

x_train.shape[0]は(60000, 28, 28)の1つ目なので60000

深層学習前のデータの整理

x_train (特徴量)

- 正規化する

```
x_train = x_train / 255  
x_test = x_test / 255
```

配列の数字は0~255のいずれか
全てを255で割って0~1の間に変換する

numpy配列は四則演算が
それぞれの要素に行なわれます

```
a = a.reshape(2,2,2)  
print(a)  
print(a.shape)  
  
[[[1 2]  
  [3 4]]  
  
 [[5 6]  
  [7 8]]]  
(2, 2, 2)
```



```
a = a / 10  
print(a)  
  
[[[0.1 0.2]  
  [0.3 0.4]]  
  
 [[0.5 0.6]  
  [0.7 0.8]]]
```

(x_train = x_train / 255 は省略して x_train /= 255 と書くことも出来ます)

深層学習前のデータの整理

y_train (正解)

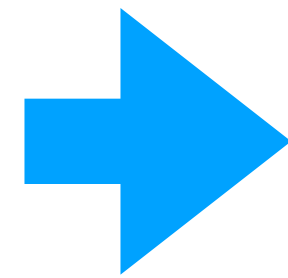
- one-hot encoding

正解は全て0から9のいずれか

これを全て0と1だけで表現するための方法(理由は後述)

one-hot encoding

0
1
2
3
4
5
6
7
8
9



1, 0, 0, 0, 0, 0, 0, 0, 0, 0 , 0
0, 1, 0, 0, 0, 0, 0, 0, 0, 0 , 0
0, 0, 1, 0, 0, 0, 0, 0, 0, 0 , 0
0, 0, 0, 1, 0, 0, 0, 0, 0, 0 , 0
0, 0, 0, 0, 1, 0, 0, 0, 0, 0 , 0
0, 0, 0, 0, 0, 1, 0, 0, 0, 0 , 0
0, 0, 0, 0, 0, 0, 1, 0, 0, 0 , 0
0, 0, 0, 0, 0, 0, 0, 1, 0, 0 , 0
0, 0, 0, 0, 0, 0, 0, 0, 1, 0 , 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0 , 1

0と1だけで0から9の数字を表現する

one-hot encoding

to_categorical()関数を使う

to_categorical(変えたい配列, 正解の数)

```
[17] from keras.utils import to_categorical  
y_train = to_categorical(y_train, 10)  
y_test = to_categorical(y_test, 10)
```

```
[18] print(y_train.shape)  
print(y_test.shape)
```

```
(60000, 10)  
(10000, 10)
```

```
▶ print(y_train[0:10])
```

```
⇒ [[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]  
 [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]  
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]  
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]  
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]  
 [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]  
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]  
 [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]  
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]  
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]]
```

```
print(y_train[0:10])
```

```
[5 0 4 1 9 2 1 3 1 4]
```



深層学習前のデータの整理

x_train (特徴量)

- 画像の2次元の配列を1次元にする
- 正規化する

y_train (正解)

- one-hot encoding

次回もこのファイルを使用します

「Driveにコピーを保存」

```
from keras.datasets import mnist
(x_train,y_train),(x_test,y_test) = mnist.load_data()
```

```
import matplotlib.pyplot as plt
plt.imshow(x_test[1], 'gray')
plt.show()
```

```
print(y_test[1])
```

```
x_train = x_train.reshape(x_train.shape[0],784)
x_test = x_test.reshape(x_test.shape[0],784)
print(x_train.shape)
print(x_test.shape)
```

```
x_train = x_train / 255
x_test = x_test / 255
```

```
from keras.utils import to_categorical
y_train = to_categorical(y_train,10)
y_test = to_categorical(y_test,10)
```

課題

- WebClassにある”kandai3.ipynb”をやってみましょう
- 実行したら”学籍番号_名前_3.ipynb”という名前で保存して提出して下さい。

締め切りは2週間後の11/9の23:59です。

締め切りを過ぎた課題は受け取らないので注意して下さい