Python Learning 3.5~

M1 Kaito Umetsu

Outline

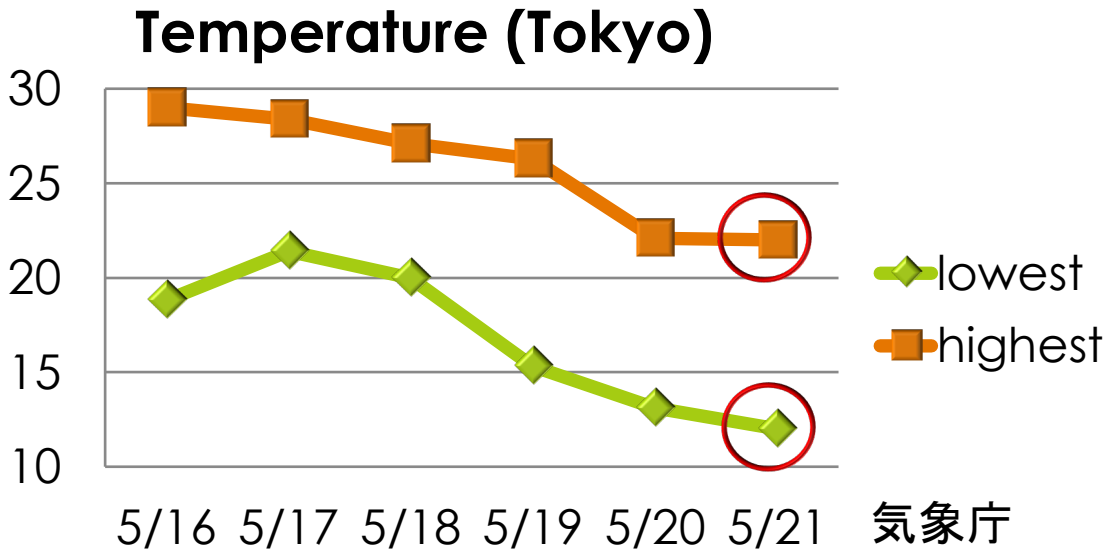
- Output layer design
  - Regression
  - Classification
- Handwriting recognition
  - MNIST dataset
  - Batch processing
- Summary

Output layer design

- Regression
  - What is regression
  - Identity function
- Classification
  - What is classification
  - Softmax function

What is Regression ?

- Regression
  - Prediction of value from Continuous data.
  - Continuous data: stocks price, high temperature, etc.

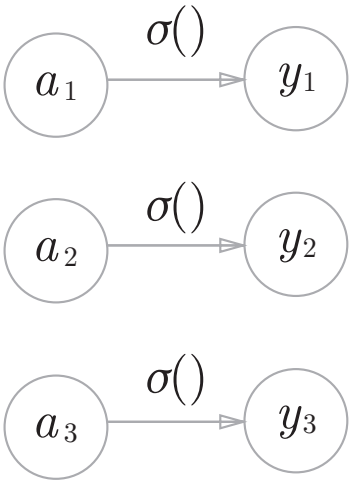


Output: value

Activation function
Identity function

Identity function

- Identity function returns input value as output.
  - H(x) = x



What is classification ?

- Classification
  - Prediction of class from data.
  - “man or woman” , “what number is?”

is 4

1: 0.02% 2: 0.04% 3: 0.01% 4: 96.87% …..

Output : probability of each class

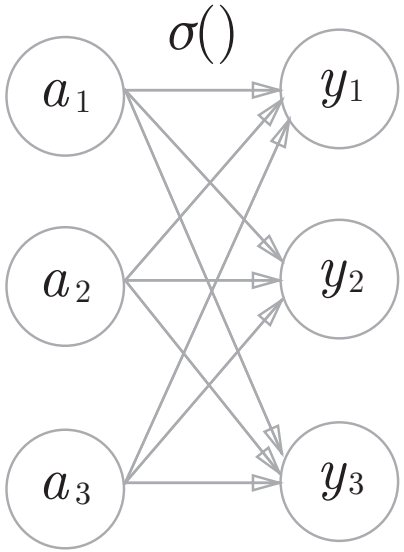
Activation function
Softmax function

Softmax function

- Definition

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

$0 \leqq y_k \leqq 1$



In Python

```
def softmax(a):
    exp_a = np.exp(a)
    sum_exp_a = np.sum(exp_a)
    y = exp_a / sum_exp_a

    return y
```

Warning against the use of softmax

- Exponential function become huge number easily.
  - This is a cause of overflow.
- Solution

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)} = \frac{C \exp(a_k)}{C \sum_{i=1}^{n} \exp(a_i)}$$

$$= \frac{\exp(a_k + \log C)}{\sum_{i=1}^{n} \exp(a_i + \log C)}$$

$$= \frac{\exp(a_k + C')}{\sum_{i=1}^{n} \exp(a_i + C')} \qquad C' = \max(a_k)$$

Feature of softmax function

- Softmax function outputs a value between 0 and 1.
- Summation of all outputs is 1.

Probability

```
>>> a = np.array([0.3, 2.9, 4.0])
>>> y = softmax(a)
>>> print(y)
[ 0.01821127  0.24519181  0.73659691]
>>> np.sum(y)
1.0
```

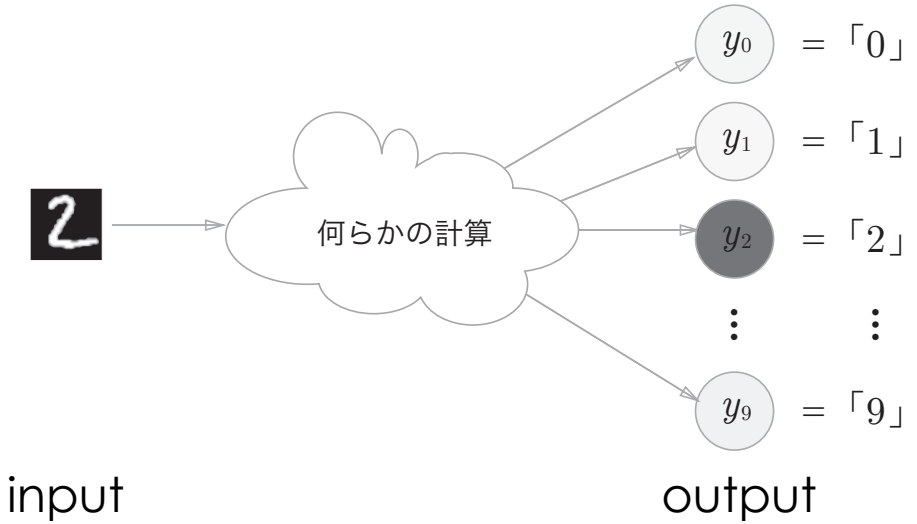
y(0): 1.8% y(1): 24.5% y(2): 73.7%

Highest value

In inference process,
It is good to omit softmax.

Number of nodes in output layer

- Number of nodes depend on how many classes are exist.



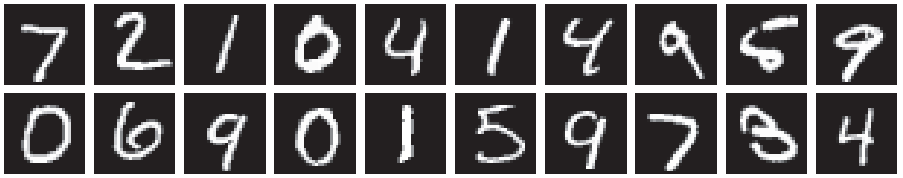
Handwriting recognition

- MNIST dataset
  - Form of data
  - other dataset

- Implementation of neural network
  - Inference process
  - Batch process

Form of MINST dataset

- MNIST dataset consist of handwriting number image from 0 to 9.
- Train image: 60000
- Test image: 10000
- 28×28 gray image
- Each pixel have a value between 0 and 255.



Import MNIST

- dataset/mnist.py
- (x_train, t_train),(x_test,t_test)=load_mnist(flatten=True, nomalize=False, one_hot_label=False)
- (train image, train label),(test image, test label)
- Option
  - Normalize: 0~255 or 0.0~1.0 (preprocessing)
  - Flatten: 1×28×28 or 784
  - One_hot_label: “2” or (0,1,0,0…..)

Dataset

- Image
  - CIFAR-10 (10 types of class: car, bird, etc. Univ. of Toronto)
  - Food 101(101 categories of food, Computer vision laboratory)
  - CelebA Dataset (face image of celeb, 香港中文大学)
- Movie
  - YouTube-BoundingBoxes Dataset (Google)
- AWS public dataset (Amazon AWS)
  - Satellite photo, genomics data, news and website data etc.

Inference process

- Neural network
  - Input layer: 784 nodes (28×28 image size)
  - Output layer: 10 nodes (0~9 classes)
  - 1st hidden layer: 50 nodes (optional value)
  - 2nd hidden layer: 100 nodes (optional value)
- get_data(): import MNIST
- init_network(): import sample_weight.pkl
- predect(): forward prediction

Inference process

```
x, t = get_data()
network = init_network()

accuracy_cnt = 0
for i in range(len(x)):
    y = predict(network, x[i])
    p = np.argmax(y)
    if p == t[i]:
        accuracy_cnt += 1
print("Accuracy:" + str(float(accuracy_cnt) / len(x)))
```

Initialize network

prediction

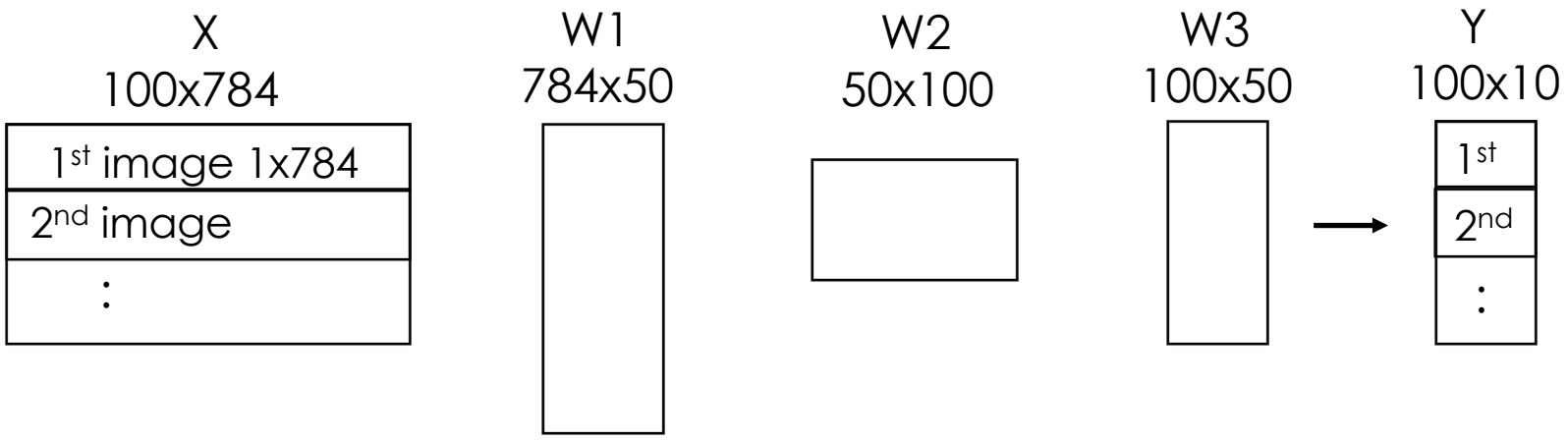
y = [0.1, 0.3, 0.2,.....0.04]

This program predict images one by one.

This work takes time.

Batch process

- Prediction plurality of image at once.



Library can compute large array quickly.

Batch process

```
x, t = get_data()
network = init_network()

batch_size = 100 # バッチの数
accuracy_cnt = 0

for i in range(0, len(x), batch_size):
    x_batch = x[i:i+batch_size]
    y_batch = predict(network, x_batch)
    p = np.argmax(y_batch, axis=1)
    accuracy_cnt += np.sum(p == t[i:i+batch_size])

print("Accuracy:" + str(float(accuracy_cnt) / len(x)))
```

Making batch

Max value each 1st dimension