Python Learning 6/11

4.5 Implementation of learning algorithm

M1　Tsuji Shota

Outline

- Implementation of learning algorithm
  - ✓ the class of two layer neural net
  - ✓ implementation of mini batch learning
  - ✓ evaluate with test data

Implementation of learning algorithm

- SGD (Stochastic Gradient Descent)

STEP1

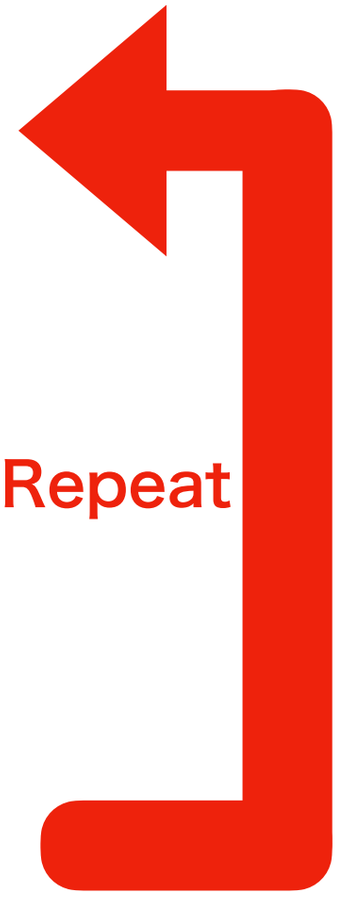
extract some data (mini batch) randomly from the training data

STEP2

calculate the gradient of each weight parameter

STEP3

update the weight parameter in the gradient direction



Two layer neural net

```
class TwoLayerNet:

    def __init__(self, input_size, hidden_size, output_size,
                 weight_init_std=0.01):
        # 重みの初期化
        self.params = {}
        self.params['W1'] = weight_init_std * \
                            np.random.randn(input_size, hidden_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * \
                            np.random.randn(hidden_size, output_size
        self.params['b2'] = np.zeros(output_size)
```

initialize weight and bias

```
    def predict(self, x):
        W1, W2 = self.params['W1'], self.params['W2']
        b1, b2 = self.params['b1'], self.params['b2']

        a1 = np.dot(x, W1) + b1
        z1 = sigmoid(a1)
        a2 = np.dot(z1, W2) + b2
        y = softmax(a2)

        return y
```

recognize from image data

```
    # x:入力データ, t:教師データ
    def loss(self, x, t):
        y = self.predict(x)

        return cross_entropy_error(y, t)
```

calculate the value of loss function

```
    def accuracy(self, x, t):
        y = self.predict(x)
        y = np.argmax(y, axis=1)
        t = np.argmax(t, axis=1)

        accuracy = np.sum(y == t) / float(x.shape[0])
        return accuracy
```

calculate recognition accuracy

```
    # x:入力データ, t:教師データ
    def numerical_gradient(self, x, t):
        loss_W = lambda W: self.loss(x, t)

        grads = {}

        grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
        grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
        grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
        grads['b2'] = numerical_gradient(loss_W, self.params['b2'])

        return grads
```

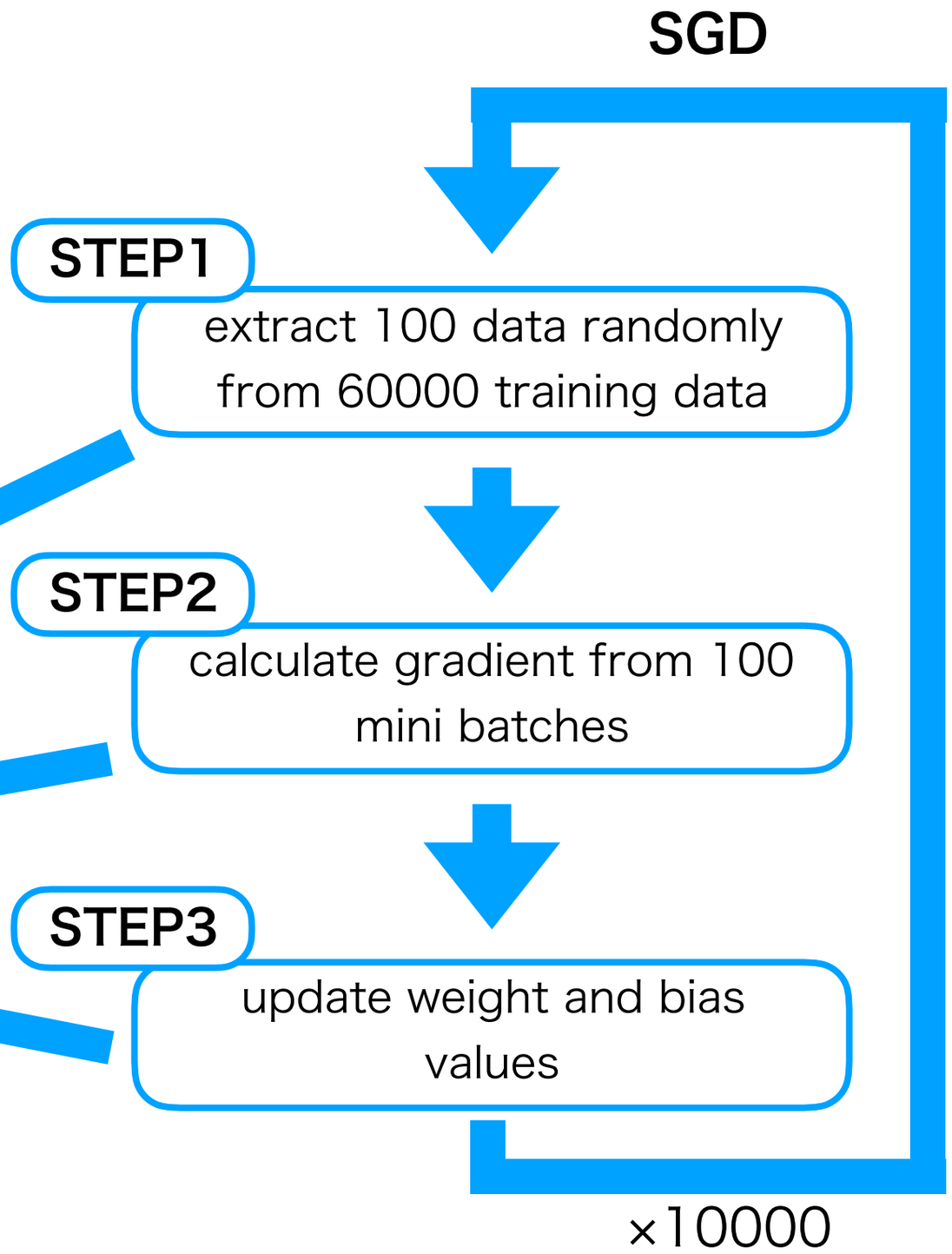
calculate the gradient

Slower than Backpropagation

→Chapter5

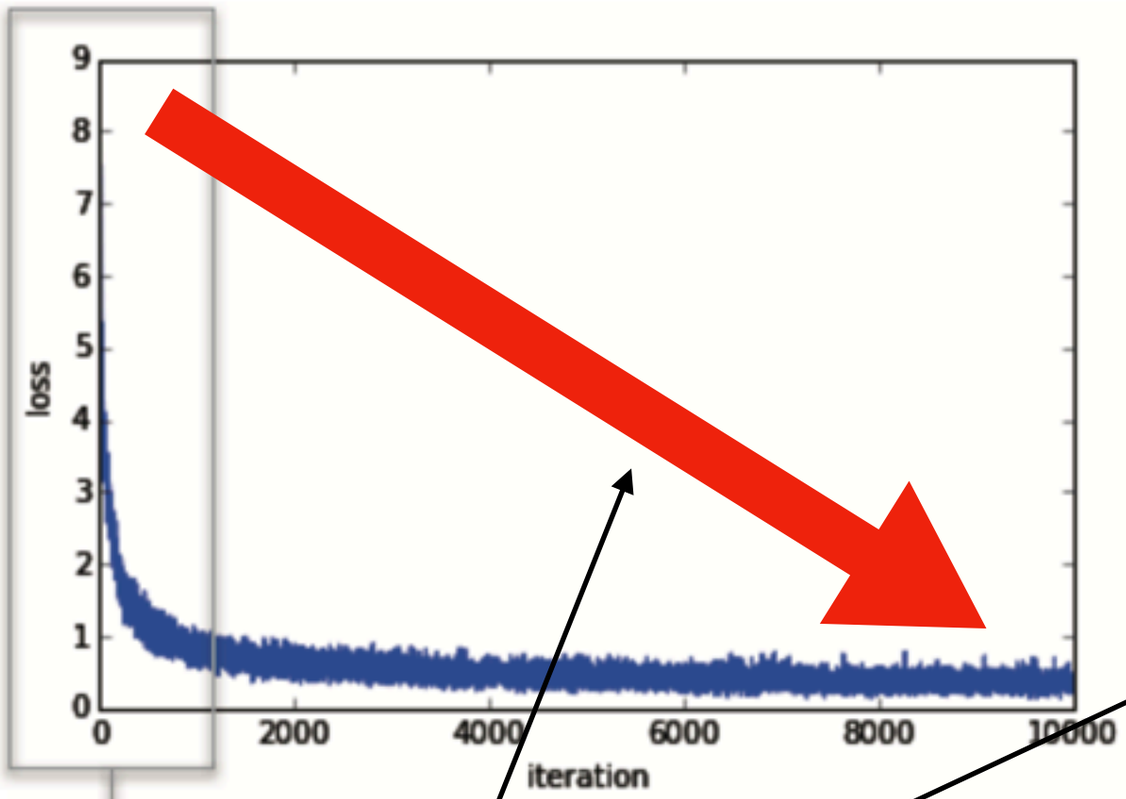
Mini batch learning

```python
import numpy as np
from dataset.mnist import load_mnist
from two_layer_net import TwoLayerNet

(x_train, t_train), (x_test, t_test) = \
    load_mnist(normalize=True, one_hot_label=True)

train_loss_list = []

# ハイパーパラメータ
iters_num = 10000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.1

network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)

for i in range(iters_num):
    # ミニバッチの取得
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    # 勾配の計算
    grad = network.numerical_gradient(x_batch, t_batch)
    # grad = network.gradient(x_batch, t_batch) # 高速版!

    # パラメータの更新
    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * grad[key]

    # 学習経過の記録
    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss)
```

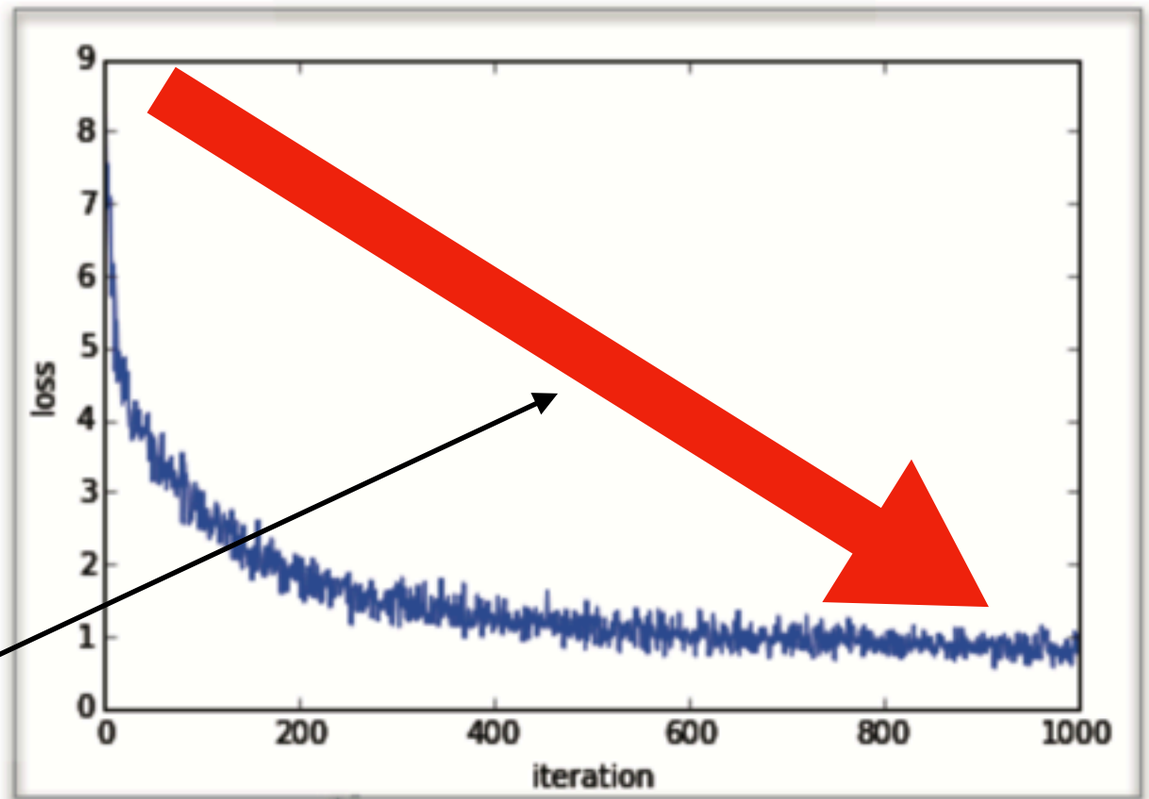


Mini batch learning

the number of iterations:10000



the number of iterations:1000

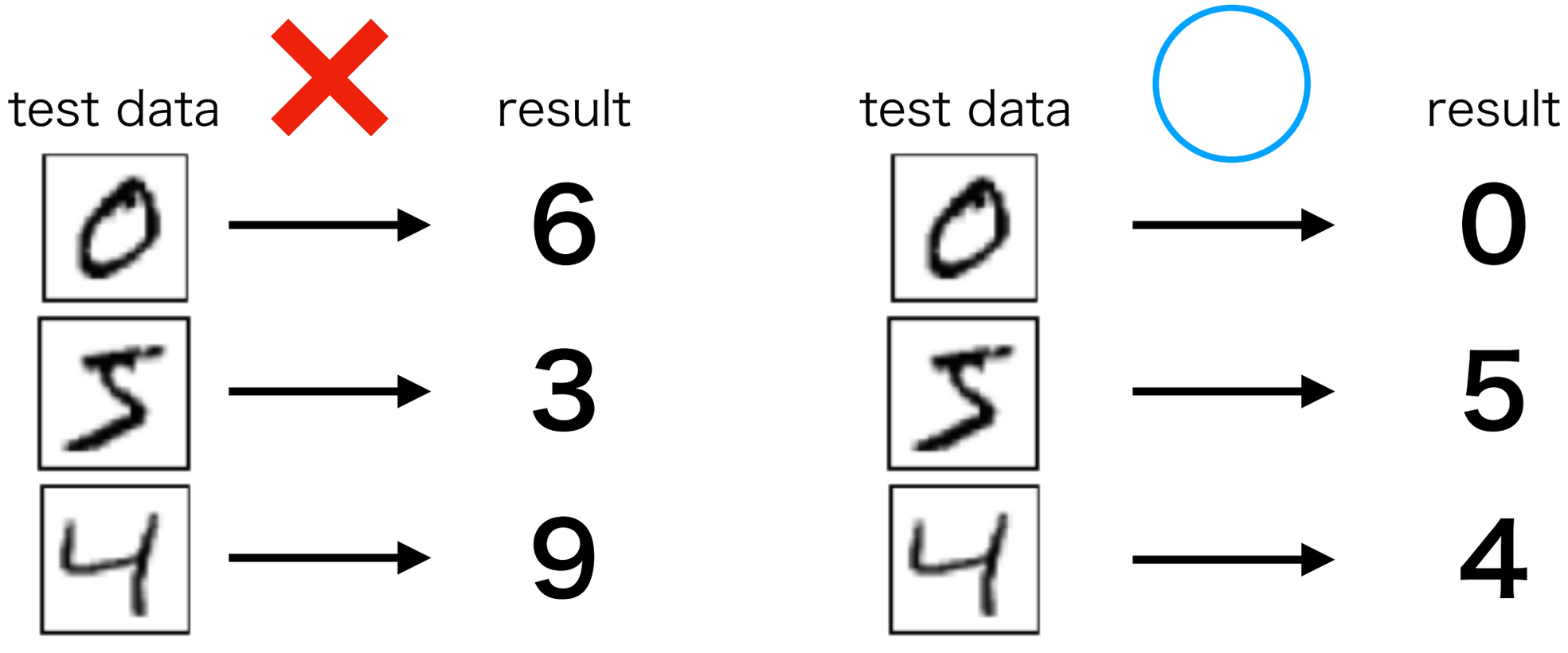


As the parameter update count increases,
the value of the loss function decreases

Learning goes well and weights are approaching optimal values

Evaluate with test data

- From the result of mini batch learning, it was found that the value of the loss function decreases as learning progresses…
- However, we identify the data except the training data with high accuracy



Evaluate with test data

- Therefore, the recognition accuracy of both the training data and the test data is periodically recorded

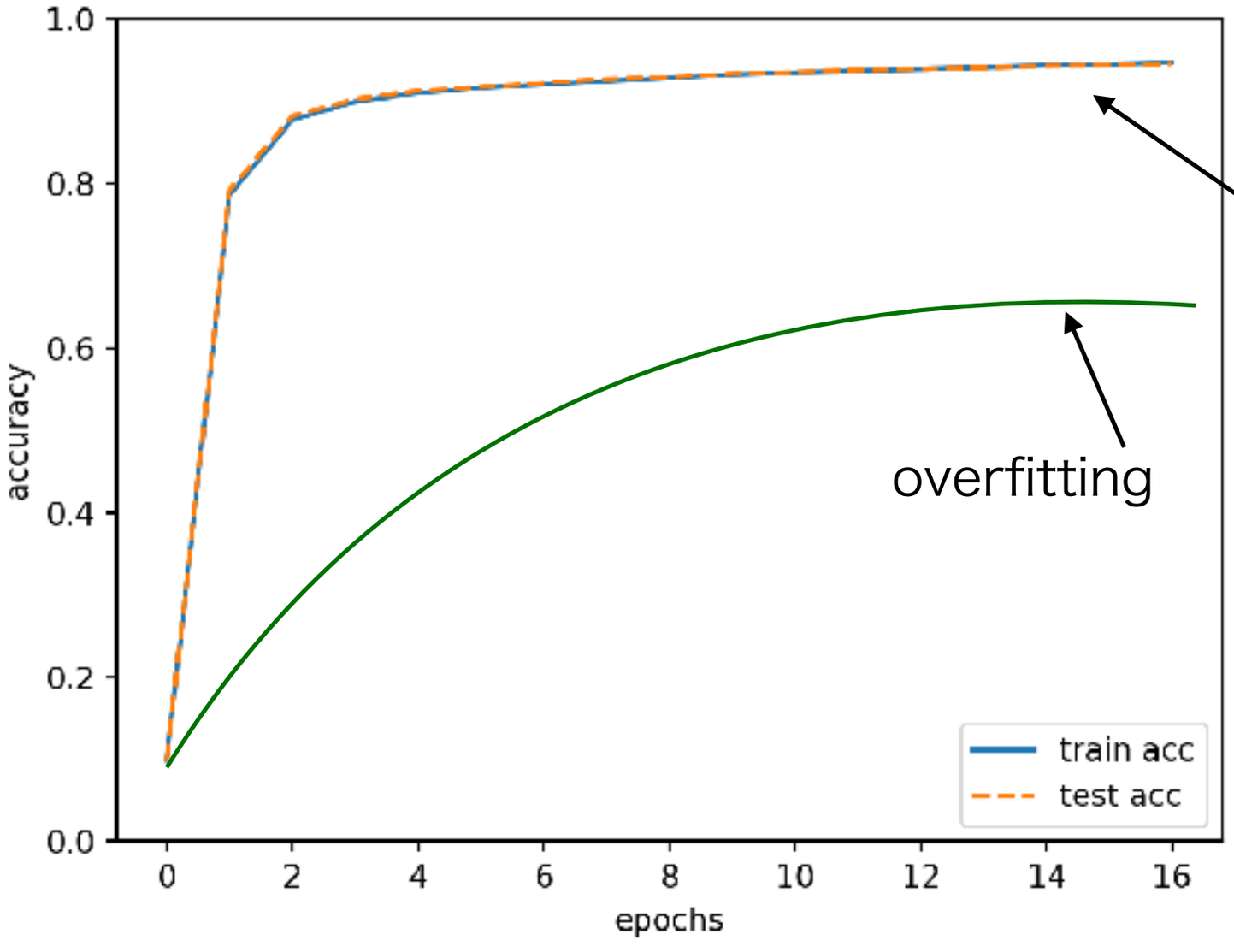
  →record it every epoch

(epoch : the number of times to train one training data)

```
import numpy as np
from dataset.mnist import load_mnist
from two_layer_net import TwoLayerNet

(x_train, t_train), (x_test, t_test) = \
    load_mnist(normalize=True, one_hot_label=True)

train_loss_list = []
train_acc_list = []
test_acc_list = []
# 1エポックあたりの繰り返し数
iter_per_epoch = max(train_size / batch_size, 1)

# ハイパーパラメータ
iters_num = 10000
batch_size = 100
learning_rate = 0.1
```

```
# 1エポックごとに認識精度を計算
if i % iter_per_epoch == 0:
    train_acc = network.accuracy(x_train, t_train)
    test_acc = network.accuracy(x_test, t_test)
    train_acc_list.append(train_acc)
    test_acc_list.append(test_acc)
    print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))
```

Evaluate with test data



recognition accuracy improves for both training data and test data, there is little difference

Summary

- The method of learning using mini batch is called “SGD”
- We can avoid overtraining by recording accuracy for both training data and test data for each epoch