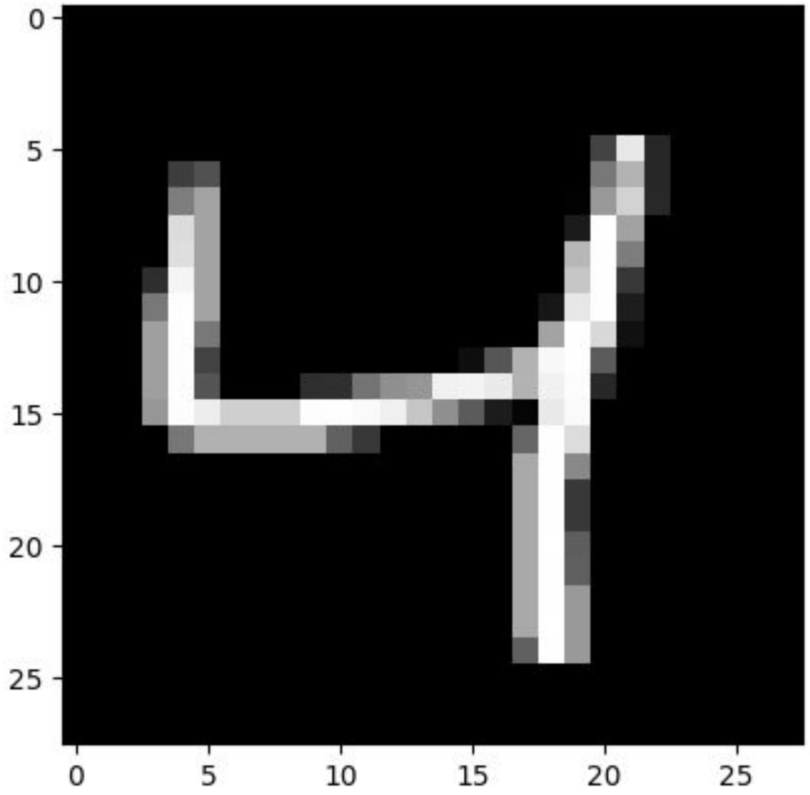
MNIST Data

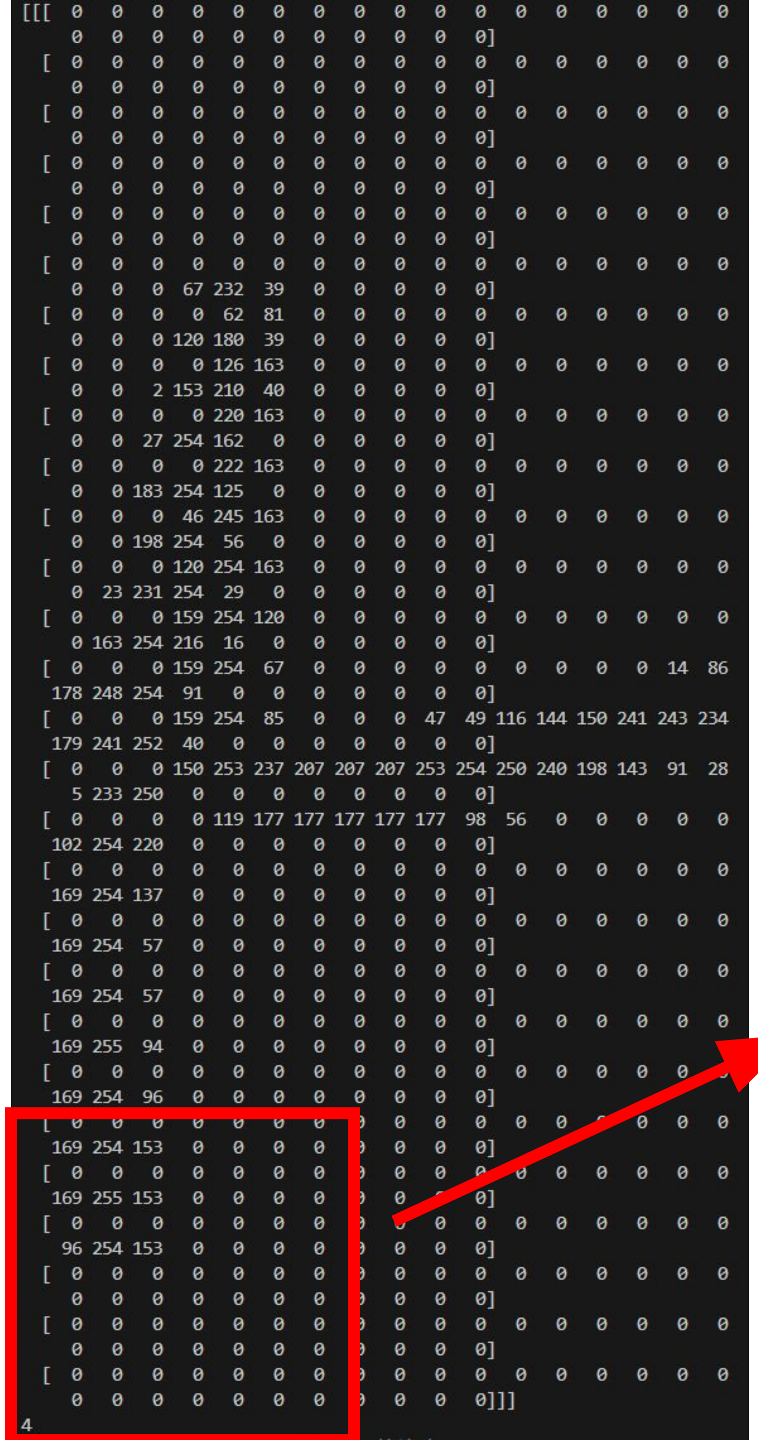
0부터 9까지의 숫자 이미지로 구성

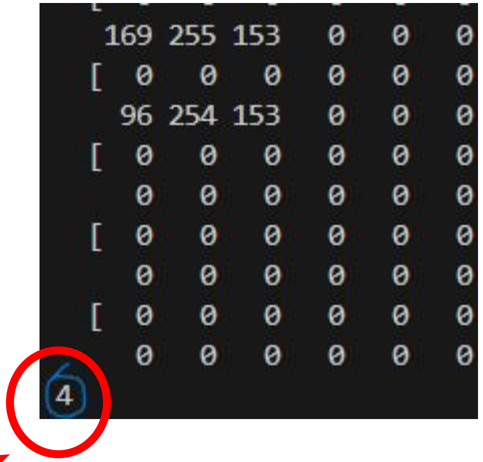
Train data 6만장
Test data 1만장

28 x 28 크기의 회색조 이미지
각 픽셀은 0~255의 값

MNIST Data







이 이미지의 정답
-> 4

ReLU Layer

```python
class Relu:
    def __init__(self):
        self.mask = None

    def forward(self, x):
        self.mask = (x <= 0)
        out = x.copy()
        out[self.mask] = 0

        return out

    def backward(self, dout):
        dout[self.mask] = 0
        dx = dout

        return dx
```

Sigmoid Layer

```python
class Sigmoid:
    def __init__(self):
        self.out = None

    def forward(self, x):
        out = sigmoid(x)
        self.out = out
        return out

    def backward(self, dout):
        dx = dout * self.out * (1-self.out)

        return dx
```

Affine Layer

```python
class Affine:
    def __init__(self, W, b):
        self.W = W
        self.b = b
        self.x = None
        self.original_x_shape = None #for tensor
        self.dW = None
        self.db = None

    def forward(self, x):
        #for tensor
        self.original_x_shape = x.shape
        x = x.reshape(x.shape[0], -1)
        self.x = x

        out = np.dot(self.x, self.W) + self.b

        return out

    def backward(self, dout):
        dx = np.dot(dout, self.W.T)
        self.dW = np.dot(self.x.T, dout)
        self.db = np.sum(dout, axis=0)

        dx = dx.reshape(*self.original_x_shape)

        return dx
```

Softmax With Loss Layer

```python
class SoftmaxWithLoss:
    def __init__(self):
        self.loss = None
        self.y = None
        self.t = None

    def forward(self, x, t):
        self.t = t
        self.y = softmax(x)
        self.loss = cross_entropy_error(self.y, self.t)

        return self.loss

    def backward(self, dout=1):
        batch_size = self.t.shape[0]
        if self.t.size == self.y.size:
            dx = (self.y - self.t) / batch_size
        else:
            dx = self.y.copy()
            dx[np.arange(batch_size), self.t] -= 1
            dx = dx / batch_size

        return dx
```

```python
def softmax(x):
    if x.ndim == 2:
        x = x.T
        x = x - np.max(x, axis=0)
        y = np.exp(x) / np.sum(np.exp(x), axis=0)
        return y.T

    x = x - np.max(x)
    return np.exp(x) / np.sum(np.exp(x))
```

Two Layered Neural Network (1)

```python
class TwoLayerNet:
    def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):
        #가중치 초기화
        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)

        #계층생성(계층을 ordered dict으로 저장)
        self.layers = OrderedDict()
        self.layers['Affine_1'] = Affine(self.params['W1'], self.params['b1'])
        self.layers['Relu'] = Relu()
        self.layers['Affine_2'] = Affine(self.params['W2'], self.params['b2'])
        self.lastlayer = SoftmaxWithLoss()
```

Two Layered Neural Network (2)

```python
def predict(self, x):
    for layer in self.layers.values():
        x = layer.forward(x)
    return x

def loss(self, x, t):
    y = self.predict(x)
    return self.lastlayer.forward(y, t)

def accuracy(self, x, t):
    y = self.predict(x)
    y = np.argmax(y, axis=1)
    if t.ndim != 1: #원-핫-인코딩이 되어 있다면:
        t = np.argmax(t, axis=1)

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

Two Layered Neural Network (3)

```python
def gradient(self, x, t):
    #순전파 계산을 해서 값을 미리 저장(계산)해 놓아야함
    #lastlayer까지 계산을 하기위해 loss호출
    self.loss(x, t)

    dout = 1
    dout = self.lastlayer.backward(dout)

    layers = list(self.layers.values())
    layers.reverse()
    for layer in layers:
        dout = layer.backward(dout)

    grads = {}
    grads['W1'] = self.layers['Affine_1'].dW
    grads['b1'] = self.layers['Affine_1'].db
    grads['W2'] = self.layers['Affine_2'].dW
    grads['b2'] = self.layers['Affine_2'].db

    return grads
```

Train Data (1)

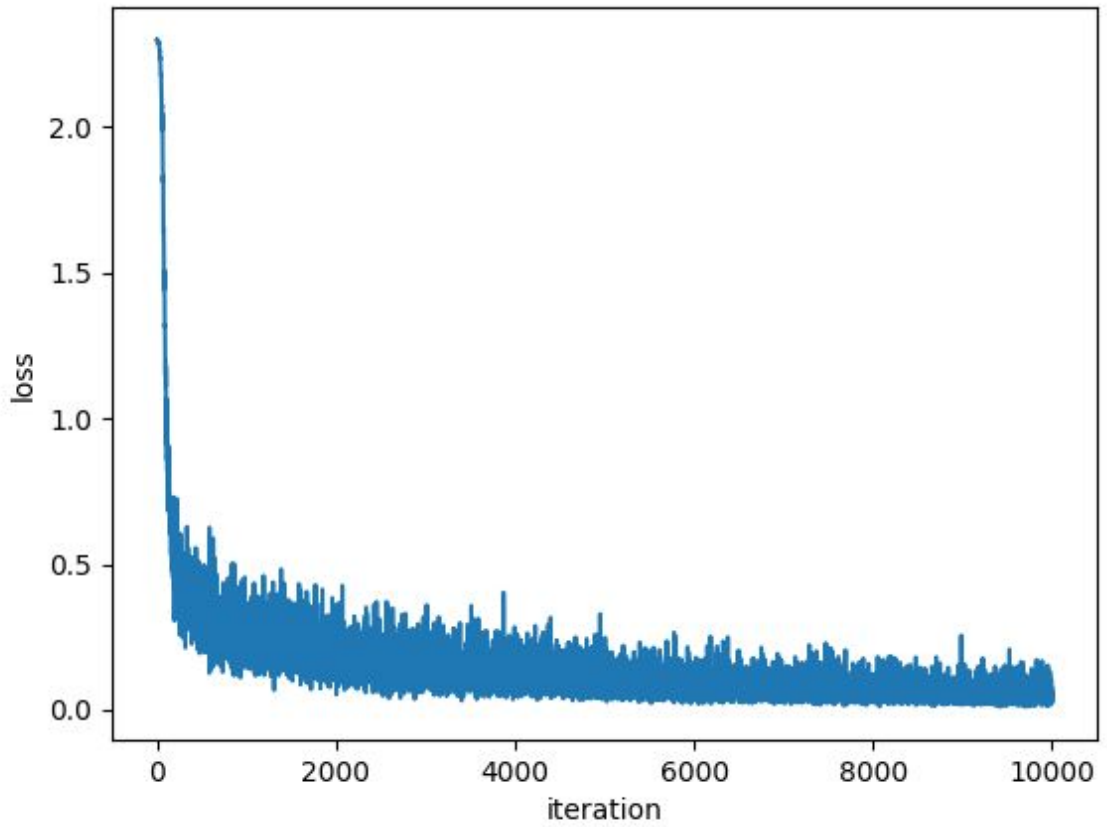
```
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
network = TwoLayerNet(input_size=784, hidden_size=50 ,output_size=10)

iters_num = 10000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.1
ts = datetime.datetime.now()

train_loss_list = []
train_acc_list = []
test_acc_list = []

iter_per_epoch = max(train_size / batch_size, 1)
```

Train Data (2)

```
for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    grad = network.gradient(x_batch, t_batch)

    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * grad[key]

    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss)

    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: ", train_acc, "test_acc :", test_acc)


    #계산하는데 걸린 시간 계산하기
    if i % iter_per_epoch == 0:
        tf = datetime.datetime.now()
        te = tf - ts
        print(i, te)
        ts = datetime.datetime.now()
```

Result

```python
iteration = np.arange(0, iters_num, 1)
plt.plot(iteration, train_loss_list)
plt.xlabel("iteration")
plt.ylabel("loss")
plt.show()
```




```
train_acc:  0.15205 test_acc : 0.1516
0 0:00:00.153337
train_acc:  0.90385 test_acc : 0.9074
600 0:00:00.558744
train_acc:  0.9210833333333334 test_acc : 0.9239
1200 0:00:00.620166
train_acc:  0.9359666666666666 test_acc : 0.9374
1800 0:00:00.518130
train_acc:  0.9437833333333333 test_acc : 0.9435
2400 0:00:00.485219
train_acc:  0.951 test_acc : 0.9512
3000 0:00:00.612764
train_acc:  0.9549166666666666 test_acc : 0.9538
3600 0:00:00.581551
train_acc:  0.9604833333333334 test_acc : 0.9579
4200 0:00:00.579780
train_acc:  0.9623666666666667 test_acc : 0.96
4800 0:00:00.571022
train_acc:  0.9676333333333333 test_acc : 0.9638
5400 0:00:00.565879
train_acc:  0.9693166666666667 test_acc : 0.9662
6000 0:00:00.567349
train_acc:  0.97265 test_acc : 0.9676
6600 0:00:00.574033
train_acc:  0.97385 test_acc : 0.9679
7200 0:00:00.567859
train_acc:  0.9748333333333333 test_acc : 0.968
7800 0:00:00.560277
train_acc:  0.9774333333333334 test_acc : 0.97
8400 0:00:00.572221
train_acc:  0.9787 test_acc : 0.971
9000 0:00:00.569093
train_acc:  0.9795166666666667 test_acc : 0.972
9600 0:00:00.582359
```