第四回レポート課題

In [43]:
```
class SoftmaxWithLoss:
    def __init__(self):#損失
        self.loss = None #softmaxの出力
        self.t = None #教師データ

    def forword(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]
        dx = (self.y - self.t) / batch_size

        return dx
```

In [52]:
```
import numpy as np
def cross_entropy_error(y, t):
    if y.ndim == 1:
        t == t.reshape(1, t.size)
        y == y.reshape(1, y.size)
    if t.size == y.size:
        t =t.argmax(axis=1)
    batch_size = t.argmax(axis=1)
    return -np.sum(np.log(y[np.arrage(batch_size), t])) / batch_siz
e
```

In [53]:
```
import numpy as np
from collections import OrderedDict
def numerical_grad(f, x):
    h = 1e-4 # 0.0001
    grad = np.zeros_like(x)
    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite']
)
    while not it.finished:
        idx = it.multi_index
        tmp_val = x[idx]
        x[idx] = float(tmp_val) + h
        fxh1 = f(x) # f(x+h)
        x[idx] = tmp_val - h
        fxh2 = f(x) # f(x-h)
        grad[idx] = (fxh1 - fxh2) / (2*h)
        x[idx] = tmp_val # 値を元に戻す
        it.iternext()
        return grad
```

```
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(hidde
n_size, output_size)
        self.params['b2'] = np.zeros(output_size)
        # レイヤの生成
        self.layers = OrderedDict()
        self.layers['Affine1'] = Affine(self.params['W1'], self.par
ams['b1'])
        self.layers['Relu1'] = Relu()
        self.layers['Affine2'] = Affine(self.params['W2'], self.par
ams['b2'])
        self.lastLayer = SoftmaxWithLoss()
    def predict(self, x):
        for layer in self.layers.values():
            x = layer.forward(x)
        return x
   # x:入力データ, t:教師データ
    def loss(self, x, t):
        y = self.predict(x)
        return self.lastLayer.forward(y, t)
    def numerical_gradient(self, x, t):
        loss_W = lambda W: self.loss(x, t)
        grads = {}
        grads['W1'] = numerical_grad(loss_W, self.params['W1'])
        grads['b1'] = numerical_grad(loss_W, self.params['b1'])
        grads['W2'] = numerical_grad(loss_W, self.params['W2'])
        grads['b2'] = numerical_grad(loss_W, self.params['b2'])
        return grads
```

In [54]:

```
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
```

In [55]:
```
class Affine:
    def __init__(self, W, b):
        self.W =W
        self.b = b
        self.x = None
        self.original_x_shape = None
        # 重み・バイアスパラメータの微分
        self.dW = None
        self.db = None
    def forward(self, x):
        # テンソル対応
        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)  # 入力データの形状に戻す（テンソル対応）
        return dx
```

In [56]:
```
import sys, os
sys.path.append(os.pardir)
import numpy as np
from dataset.mnist import load_mnist
from two_layer_net import TwoLayerNet

# データの読み込み
(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)

x_batch = x_train[:3]
t_batch = t_train[:3]

grad_numerical = network.numerical_gradient(x_batch, t_batch)
grad_backprop = network.gradient(x_batch, t_batch)

for key in grad_numerical.keys():
    diff = np.average( np.abs(grad_backprop[key] - grad_numerical[key]) )
    print(key + ":" + str(diff))
```

```
W1:4.859015284382053e-10
b1:2.8082409620486856e-09
W2:6.718198485296692e-09
b2:1.395738336962271e-07
```

In [57]:
```
def gradient(network, x, t):
   # 自分で実装したSoftmax with lossクラスを使ってみてください
    lastLayer = SoftmaxWithLoss()
   # forward
   #self.loss(x, t)
    network.loss(x, t)
   # backward
    dout = 1
    dout = lastLayer.backward(dout)
   #layers = list(self.layers.values())
    layers = list(network.layers.values())
    layers.reverse()
    for layer in layers:
        dout = layer.backward(dout)
   # 設定
        grads = {}
   #grads['W1'], grads['b1'] = self.layers['Affine1'].dW, self.laye
rs['Affine1'].db
        grads['W1'], grads['b1'] = network.layers['Affine1'].dW, se
lf.layers['Affine1'].db
   #grads['W2'], grads['b2'] = self.layers['Affine2'].dW, self.laye
rs['Affine2'].db
        grads['W2'], grads['b2'] = network.layers['Affine2'].dW, se
lf.layers['Affine2'].db
    return grads
```

In [63]:
```
import sys, os
sys.path.append(os.pardir)

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

# データの読み込み
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, o
ne_hot_label=True)
network = TwoLayerNet(input_size=784, hidden_size=50, output_size=1
0)

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

train_loss_list = []
train_acc_list = []
test_acc_list = []
iter_per_epoch = max(train_size / batch_size, 1)
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]
    # 勾配の計算
    grad = network.gradient(x_batch, t_batch)

    # パラメーターの更新
    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)

    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)
```

```
0.15121666666666667 0.1499
0.9007166666666667 0.9052
0.9242 0.926
0.9380833333333334 0.9388
0.9469333333333333 0.9467
0.9516166666666667 0.9487
0.9571 0.9551
0.96195 0.9582
0.9657833333333333 0.9614
0.9675166666666667 0.9601
0.9696333333333333 0.9629
0.97205 0.9643
0.9731833333333333 0.9652
0.9756 0.9679
0.9776666666666667 0.9674
0.9787166666666667 0.9691
0.9797666666666667 0.9693
```

感想

実行結果から勾配の誤差は殆どないことがわかった。 誤差逆伝播法の学習結果は学習させることで精度が上がっていることが読み取れた。

参考文献

ゼロから作るDeepLeaning pythonで学ぶディープラーニングの理論と実装