4장 신경망 학습 - 계속

학습 알고리즘 구현

- 손글씨 숫자 인식 MNIST 를 구현하자.
- 신경망은 2층. 은닉층은 1개로 구현
- 구현은 스탠퍼드 CS231n 참고했다함

In [1]:
```
import sys, os
sys.path.append(os.pardir)
from common.functions import *
from common.gradient import numerical_gradient

class TwoLayerNet:
    def __init__(self, input_size, hidden_size, output_size, weight_init_std=0.01):
        """
        input_size: 입력층 뉴런 수
        hidden_size: 은닉층 뉴런 수
        output_size: 출력층 뉴런 수
        """

        # 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_siz
        self.params['b2'] = np.zeros(output_size)

    def predict(self, x):
        """
        x 는 이미지 데이터. 추론해서 y 를 돌려준다.
        """
        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 = sigmoid(a2)

        return y

    def loss(self, x, t):
        """
        손실 계산 : x 는 이미지 t 는 라벨
        """
        y = self.predict(x)
        return cross_entropy_error(y, t)

    def accuracy(self, x, t):
        y = self.predict(x)
        y = np.argmax(t, axis=1)
        t = np.argmax(t, axis=1)
        return np.sum(y==t)/float(x.shape[0])

    def numerical_gradient(self, x, t):
        """
        각 가중치들의 손실함수에 대한 기울기를 구한다.
        """
        loss_W = lambda W: self.loss(x, t)
```

```
        # 기울기값
        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
```

In [2]:
```
# 생성해보면, params 에 이런 것들이 만들어진다.
net = TwoLayerNet(input_size=784, hidden_size=100, output_size=10)
for k,v in net.params.items():
    print(k, v.shape)
```

```
W1 (784, 100)
b1 (100,)
W2 (100, 10)
b2 (10,)
```

In [3]:
```
# 더미데이터로 예측해보기
x = np.random.rand(10, 784)
y = net.predict(x)
#print(y)
```

In [4]:
```
# 이건 실행하면 오래걸린다.
t = np.random.rand(10, 10)  # 샘플 라벨
grads = net.numerical_gradient(x,t)
#print(grads)
for k,v in grads.items():
    print(k, v.shape)
```

```
W1 (784, 100)
b1 (100,)
W2 (100, 10)
b2 (10,)
```

In [ ]:
```
from dataset.mnist import load_mnist
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
train_loss_list = []
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):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]
    grad = network.numerical_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)
```

In [ ]:
```
print(train_loss_list)
```

In [1]:
```
# TODO: 4장 마지막 부분은 진행 못함. 너무 느리고, m1 이 열을 낸다!
```

In [ ]: