2_2_1_vanishing_gradient

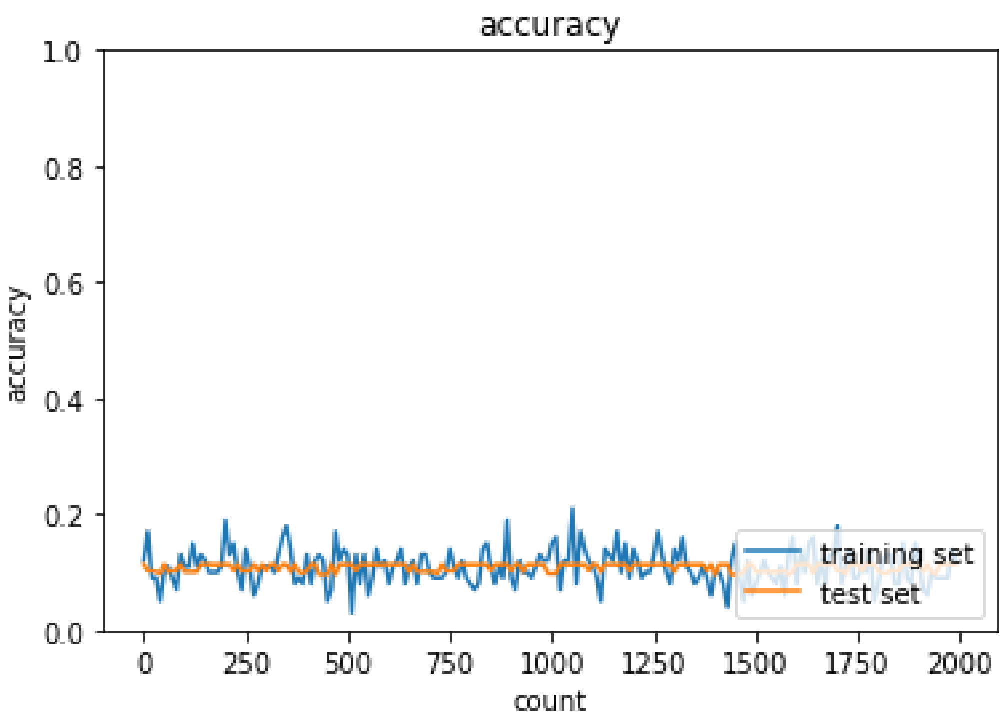
August 16, 2020

1 vanishing gradient

1.1 sigmoid - gauss

[1]:
```
import sys, os
sys.path.append(os.pardir)  #
import numpy as np
from common import layers
from collections import OrderedDict
from common import functions
from data.mnist import load_mnist
import matplotlib.pyplot as plt

# mnist
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True,␣
 ↪one_hot_label=True)
train_size = len(x_train)

print("     ")

#
wieght_init = 0.01
#
input_layer_size = 784
#
hidden_layer_1_size = 40
hidden_layer_2_size = 20

#
output_layer_size = 10
#
iters_num = 2000
#
batch_size = 100
#
learning_rate = 0.1
#
plot_interval=10
```

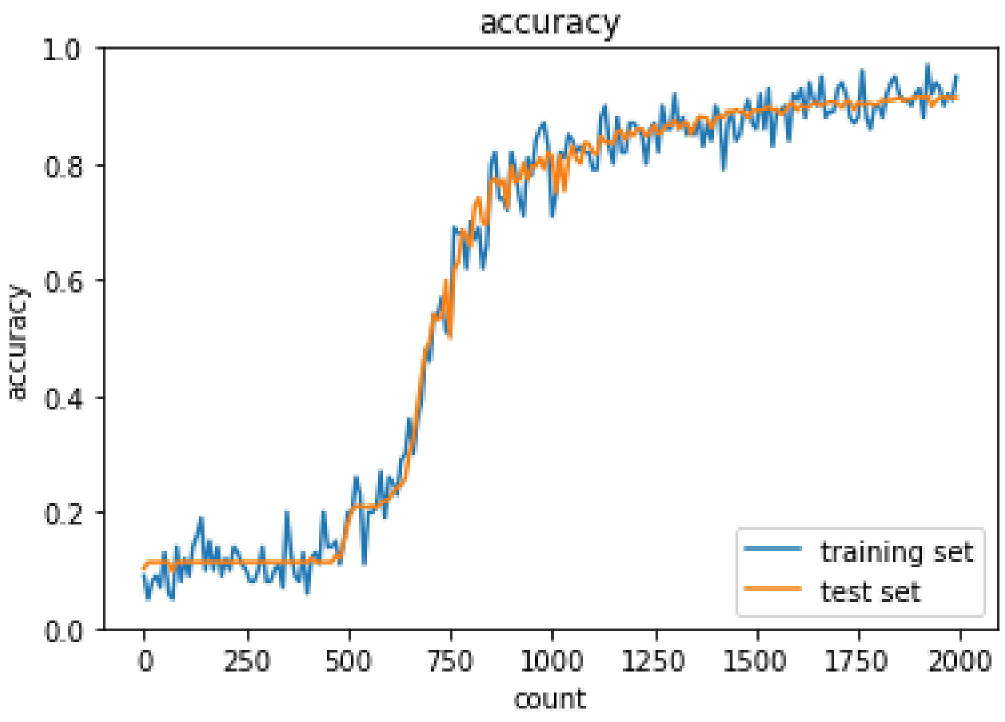
```
#
def init_network():
    network = {}
    network['W1'] = wieght_init * np.random.randn(input_layer_size,␣
 ↪hidden_layer_1_size)
    network['W2'] = wieght_init * np.random.randn(hidden_layer_1_size,␣
 ↪hidden_layer_2_size)
    network['W3'] = wieght_init * np.random.randn(hidden_layer_2_size,␣
 ↪output_layer_size)

    network['b1'] = np.zeros(hidden_layer_1_size)
    network['b2'] = np.zeros(hidden_layer_2_size)
    network['b3'] = np.zeros(output_layer_size)

    return network

#
def forward(network, x):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']
    hidden_f = functions.sigmoid

    u1 =  np.dot(x, W1) + b1
    z1 = hidden_f(u1)
    u2 =  np.dot(z1, W2) + b2
    z2 = hidden_f(u2)
    u3 =  np.dot(z2, W3) + b3
    y = functions.softmax(u3)

    return z1, z2, y

#
def backward(x, d, z1, z2, y):
    grad = {}

    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']
    hidden_d_f = functions.d_sigmoid
    last_d_f = functions.d_softmax_with_loss


    #
    delta3 = last_d_f(d, y)
    # b3
    grad['b3'] = np.sum(delta3, axis=0)
    # W3
    grad['W3'] = np.dot(z2.T, delta3)
    # 2
    delta2 = np.dot(delta3, W3.T) * hidden_d_f(z2)
    # b2
    grad['b2'] = np.sum(delta2, axis=0)
    # W2
    grad['W2'] = np.dot(z1.T, delta2)
    # 1
    delta1 = np.dot(delta2, W2.T) * hidden_d_f(z1)
    # b1
    grad['b1'] = np.sum(delta1, axis=0)
    # W1
    grad['W1'] = np.dot(x.T, delta1)

    return grad

#
network = init_network()

accuracies_train = []
accuracies_test = []

#
def accuracy(x, d):
    z1, z2, y = forward(network, x)
    y = np.argmax(y, axis=1)
    if d.ndim != 1 : d = np.argmax(d, axis=1)
    accuracy = np.sum(y == d) / float(x.shape[0])
    return accuracy

for i in range(iters_num):
    #
    batch_mask = np.random.choice(train_size, batch_size)
    #
    x_batch = x_train[batch_mask]
    #
    d_batch = d_train[batch_mask]



    z1, z2, y = forward(network, x_batch)
    grad = backward(x_batch, d_batch, z1, z2, y)

    if (i+1)%plot_interval==0:
        accr_test = accuracy(x_test, d_test)
        accuracies_test.append(accr_test)
        accr_train = accuracy(x_batch, d_batch)
        accuracies_train.append(accr_train)

#         print('Generation: ' + str(i+1) + '.   (    ) = ' + str(accr_train))
#         print('                    : ' + str(i+1) + '.   (  ) = ' + str(accr_test))

    #
    for key in ('W1', 'W2', 'W3', 'b1', 'b2', 'b3'):
        network[key]  -= learning_rate * grad[key]


lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test,  label="test set")
plt.legend(loc="lower right")
plt.title("accuracy")
plt.xlabel("count")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
#
plt.show()
```



1.2 ReLU - gauss

[2]:
```
import sys, os
sys.path.append(os.pardir)  #
import numpy as np
from data.mnist import load_mnist
from PIL import Image
import pickle
from common import functions
import matplotlib.pyplot as plt

# mnist
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True,␣
 ↪one_hot_label=True)
train_size = len(x_train)

print("     ")

#
wieght_init = 0.01
#
input_layer_size = 784
#
hidden_layer_1_size = 40
hidden_layer_2_size = 20

#
output_layer_size = 10
#
iters_num = 2000
#
batch_size = 100
#
learning_rate = 0.1
#
plot_interval=10

#
def init_network():
    network = {}

    network['W1'] = wieght_init * np.random.randn(input_layer_size,␣
 ↪hidden_layer_1_size)
    network['W2'] = wieght_init * np.random.randn(hidden_layer_1_size,␣
 ↪hidden_layer_2_size)
    network['W3'] = wieght_init * np.random.randn(hidden_layer_2_size,␣
 ↪output_layer_size)
```

```
    network['b1'] = np.zeros(hidden_layer_1_size)
    network['b2'] = np.zeros(hidden_layer_2_size)
    network['b3'] = np.zeros(output_layer_size)

    return network

#
def forward(network, x):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']

    ###########      ##############

    hidden_f = functions.relu

    #################################

    u1 =  np.dot(x, W1) + b1
    z1 = hidden_f(u1)
    u2 =  np.dot(z1, W2) + b2
    z2 = hidden_f(u2)
    u3 =  np.dot(z2, W3) + b3
    y = functions.softmax(u3)

    return z1, z2, y

#
def backward(x, d, z1, z2, y):
    grad = {}

    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']

    ###########      ##############

    hidden_d_f = functions.d_relu

    #################################


    #
    delta3 = functions.d_softmax_with_loss(d, y)
    # b3
    grad['b3'] = np.sum(delta3, axis=0)
    # W3
    grad['W3'] = np.dot(z2.T, delta3)
    # 2
    delta2 = np.dot(delta3, W3.T) * hidden_d_f(z2)
    # b2
    grad['b2'] = np.sum(delta2, axis=0)
    # W2
    grad['W2'] = np.dot(z1.T, delta2)
    # 1
    delta1 = np.dot(delta2, W2.T) * hidden_d_f(z1)
    # b1
    grad['b1'] = np.sum(delta1, axis=0)
    # W1
    grad['W1'] = np.dot(x.T, delta1)

    return grad

#
network = init_network()

accuracies_train = []
accuracies_test = []

#
def accuracy(x, d):
    z1, z2, y = forward(network, x)
    y = np.argmax(y, axis=1)
    if d.ndim != 1 : d = np.argmax(d, axis=1)
    accuracy = np.sum(y == d) / float(x.shape[0])
    return accuracy

for i in range(iters_num):
    #
    batch_mask = np.random.choice(train_size, batch_size)
    #
    x_batch = x_train[batch_mask]
    #
    d_batch = d_train[batch_mask]



    z1, z2, y = forward(network, x_batch)
    grad = backward(x_batch, d_batch, z1, z2, y)

    if (i+1)%plot_interval==0:
        accr_test = accuracy(x_test, d_test)
        accuracies_test.append(accr_test)

        accr_train = accuracy(x_batch, d_batch)
        accuracies_train.append(accr_train)

#         print('Generation: ' + str(i+1) + '.   (    ) = ' + str(accr_train))
#         print('                : ' + str(i+1) + '.   (  ) = ' + str(accr_test))

    #
    for key in ('W1', 'W2', 'W3', 'b1', 'b2', 'b3'):
        network[key]  -= learning_rate * grad[key]


lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test,  label="test set")
plt.legend(loc="lower right")
plt.title("accuracy")
plt.xlabel("count")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
#
plt.show()
```

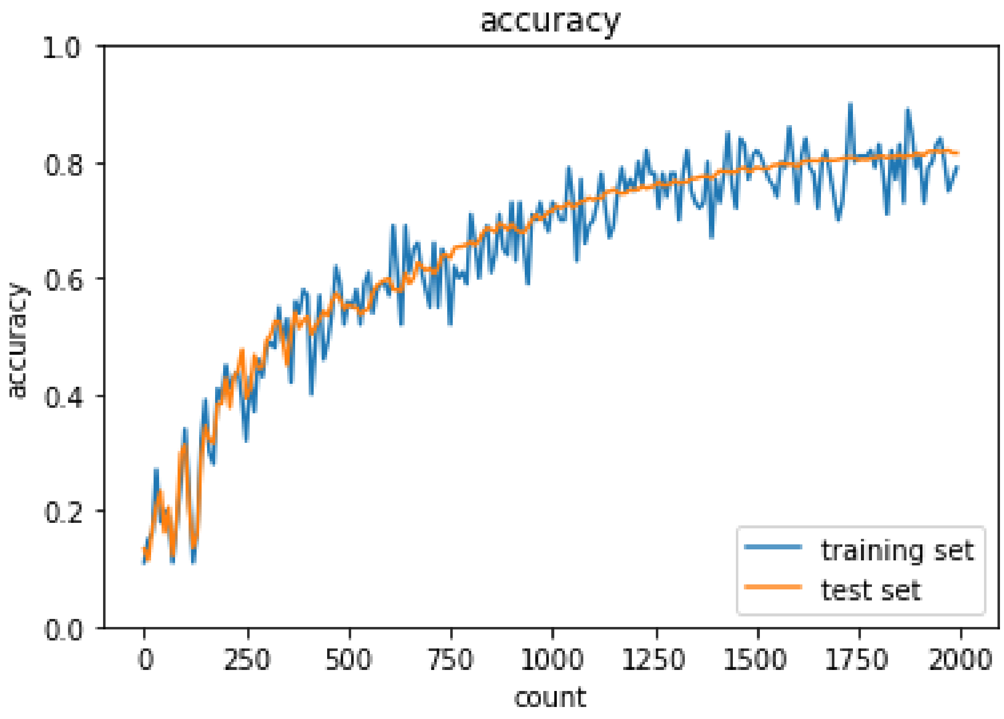


1.3 sigmoid - Xavier

[3]:
```
import sys, os
sys.path.append(os.pardir)  #
import numpy as np
from data.mnist import load_mnist
from PIL import Image
import pickle
from common import functions
import matplotlib.pyplot as plt

# mnist
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True,␣
↪one_hot_label=True)
train_size = len(x_train)

print("     ")

#
input_layer_size = 784
#
hidden_layer_1_size = 40
hidden_layer_2_size = 20
#
output_layer_size = 10
#
iters_num = 2000
#
batch_size = 100
#
learning_rate = 0.1
#
plot_interval=10

#
def init_network():
    network = {}

    ###########      ##############

    # Xavier
    network['W1'] = np.random.randn(input_layer_size, hidden_layer_1_size) /␣
↪(np.sqrt(input_layer_size))
    network['W2'] = np.random.randn(hidden_layer_1_size, hidden_layer_2_size) /␣
↪(np.sqrt(hidden_layer_1_size))
    network['W3'] = np.random.randn(hidden_layer_2_size, output_layer_size) /␣
↪(np.sqrt(hidden_layer_2_size))
```

```
    ##################################

    network['b1'] = np.zeros(hidden_layer_1_size)
    network['b2'] = np.zeros(hidden_layer_2_size)
    network['b3'] = np.zeros(output_layer_size)

    return network

#
def forward(network, x):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']
    hidden_f = functions.sigmoid

    u1 =  np.dot(x, W1) + b1
    z1 = hidden_f(u1)
    u2 =  np.dot(z1, W2) + b2
    z2 = hidden_f(u2)
    u3 =  np.dot(z2, W3) + b3
    y = functions.softmax(u3)

    return z1, z2, y

#
def backward(x, d, z1, z2, y):
    grad = {}

    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']
    hidden_d_f = functions.d_sigmoid

    #
    delta3 = functions.d_softmax_with_loss(d, y)
    # b3
    grad['b3'] = np.sum(delta3, axis=0)
    # W3
    grad['W3'] = np.dot(z2.T, delta3)
    # 2
    delta2 = np.dot(delta3, W3.T) * hidden_d_f(z2)
    # b2
    grad['b2'] = np.sum(delta2, axis=0)
    # W2
    grad['W2'] = np.dot(z1.T, delta2)
    # 1
    delta1 = np.dot(delta2, W2.T) * hidden_d_f(z1)
    # b1
    grad['b1'] = np.sum(delta1, axis=0)
    # W1
    grad['W1'] = np.dot(x.T, delta1)

    return grad

#
network = init_network()

accuracies_train = []
accuracies_test = []

#
def accuracy(x, d):
    z1, z2, y = forward(network, x)
    y = np.argmax(y, axis=1)
    if d.ndim != 1 : d = np.argmax(d, axis=1)
    accuracy = np.sum(y == d) / float(x.shape[0])
    return accuracy

for i in range(iters_num):
    #
    batch_mask = np.random.choice(train_size, batch_size)
    #
    x_batch = x_train[batch_mask]
    #
    d_batch = d_train[batch_mask]



    z1, z2, y = forward(network, x_batch)
    grad = backward(x_batch, d_batch, z1, z2, y)

    if (i+1)%plot_interval==0:
        accr_test = accuracy(x_test, d_test)
        accuracies_test.append(accr_test)

        accr_train = accuracy(x_batch, d_batch)
        accuracies_train.append(accr_train)

#         print('Generation: ' + str(i+1) + '.   (    ) = ' + str(accr_train))
#         print('                : ' + str(i+1) + '.   (  ) = ' + str(accr_test))

    #
    for key in ('W1', 'W2', 'W3', 'b1', 'b2', 'b3'):
        network[key]  -= learning_rate * grad[key]
```

```
lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test,  label="test set")
plt.legend(loc="lower right")
plt.title("accuracy")
plt.xlabel("count")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
#
plt.show()
```

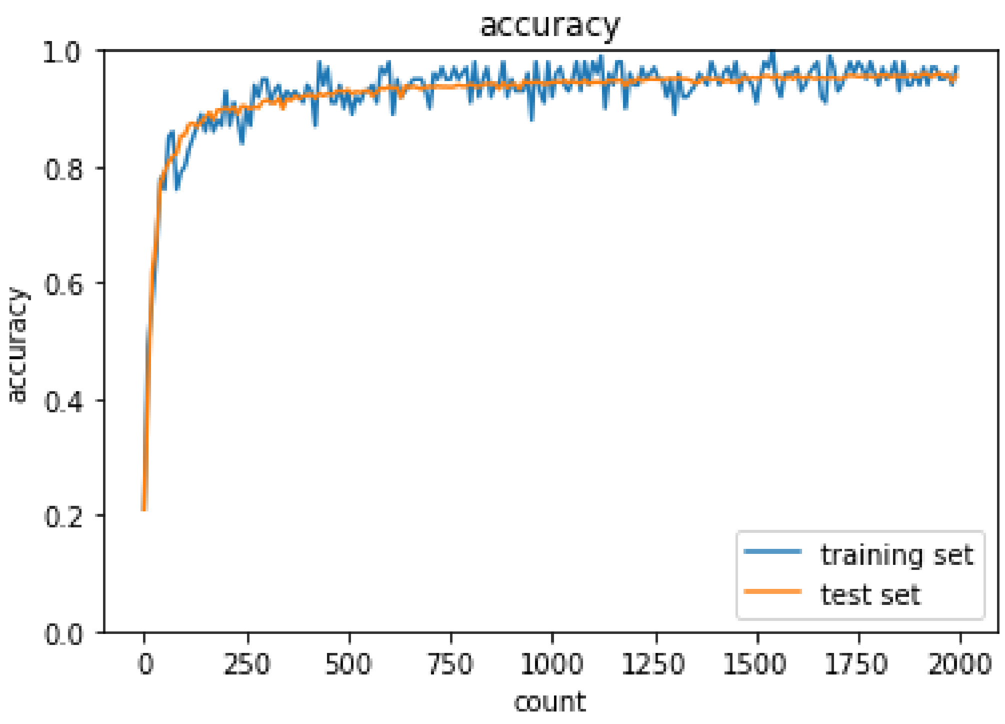


1.4 ReLU - He

[4]:
```
import sys, os
sys.path.append(os.pardir)  #
import numpy as np
from data.mnist import load_mnist
from PIL import Image
import pickle
from common import functions
```

```
import matplotlib.pyplot as plt

# mnist
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True,␣
 ↪one_hot_label=True)
train_size = len(x_train)

print("     ")

#
wieght_init = 0.01
#
input_layer_size = 784
#
hidden_layer_1_size = 40
hidden_layer_2_size = 20

#
output_layer_size = 10
#
iters_num = 2000
#
batch_size = 100
#
learning_rate = 0.1
#
plot_interval=10

#
def init_network():
    network = {}

    ###########      ##############

    # He
    network['W1'] = np.random.randn(input_layer_size, hidden_layer_1_size) / np.
 ↪sqrt(input_layer_size) * np.sqrt(2)
    network['W2'] = np.random.randn(hidden_layer_1_size, hidden_layer_2_size) /␣
 ↪np.sqrt(hidden_layer_1_size) * np.sqrt(2)
    network['W3'] = np.random.randn(hidden_layer_2_size, output_layer_size) /␣
 ↪np.sqrt(hidden_layer_2_size) * np.sqrt(2)

    #################################

    network['b1'] = np.zeros(hidden_layer_1_size)
    network['b2'] = np.zeros(hidden_layer_2_size)
    network['b3'] = np.zeros(output_layer_size)
    return network

#
def forward(network, x):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']

    ###########      ##############

    hidden_f = functions.relu

    #################################

    u1 =  np.dot(x, W1) + b1
    z1 = hidden_f(u1)
    u2 =  np.dot(z1, W2) + b2
    z2 = hidden_f(u2)
    u3 =  np.dot(z2, W3) + b3
    y = functions.softmax(u3)

    return z1, z2, y

#
def backward(x, d, z1, z2, y):
    grad = {}

    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']

    ###########      ##############

    hidden_d_f = functions.d_relu

    #################################

    #
    delta3 = functions.d_softmax_with_loss(d, y)
    # b3
    grad['b3'] = np.sum(delta3, axis=0)
    # W3
    grad['W3'] = np.dot(z2.T, delta3)
    # 2
    delta2 = np.dot(delta3, W3.T) * hidden_d_f(z2)
    # b2
    grad['b2'] = np.sum(delta2, axis=0)
    # W2
    grad['W2'] = np.dot(z1.T, delta2)
    # 1
    delta1 = np.dot(delta2, W2.T) * hidden_d_f(z1)
    # b1
    grad['b1'] = np.sum(delta1, axis=0)
    # W1
    grad['W1'] = np.dot(x.T, delta1)

    return grad

#
network = init_network()

accuracies_train = []
accuracies_test = []

#
def accuracy(x, d):
    z1, z2, y = forward(network, x)
    y = np.argmax(y, axis=1)
    if d.ndim != 1 : d = np.argmax(d, axis=1)
    accuracy = np.sum(y == d) / float(x.shape[0])
    return accuracy

for i in range(iters_num):
    #
    batch_mask = np.random.choice(train_size, batch_size)
    #
    x_batch = x_train[batch_mask]
    #
    d_batch = d_train[batch_mask]



    z1, z2, y = forward(network, x_batch)
    grad = backward(x_batch, d_batch, z1, z2, y)

    if (i+1)%plot_interval==0:
        accr_test = accuracy(x_test, d_test)
        accuracies_test.append(accr_test)

        accr_train = accuracy(x_batch, d_batch)
        accuracies_train.append(accr_train)

#         print('Generation: ' + str(i+1) + '.   (    ) = ' + str(accr_train))
#         print('                : ' + str(i+1) + '.   (  ) = ' + str(accr_test))
```

```
    #
    for key in ('W1', 'W2', 'W3', 'b1', 'b2', 'b3'):
        network[key]  -= learning_rate * grad[key]


lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies_train, label="training set")
plt.plot(lists, accuracies_test,  label="test set")
plt.legend(loc="lower right")
plt.title("accuracy")
plt.xlabel("count")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
#
plt.show()
```


accuracy
accuracy
count
training set
test set