2_4_optimizer

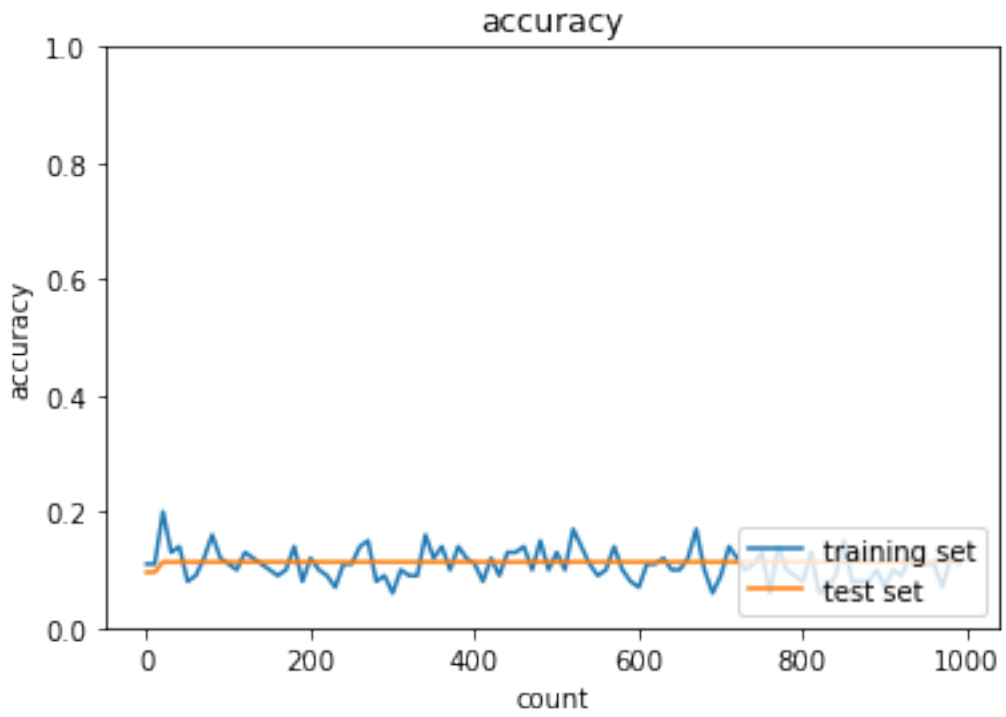
August 16, 2020

1 optimizer

1.1 SGD

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


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

print("     ")

# batch_normalization   ================================
# use_batchnorm = True
use_batchnorm = False
# ======================================================


network = MultiLayerNet(input_size=784, hidden_size_list=[40, 20],␣
↪output_size=10, activation='sigmoid', weight_init_std=0.01,
                        use_batchnorm=use_batchnorm)

iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.01

train_loss_list = []
accuracies_train = []
accuracies_test = []
```

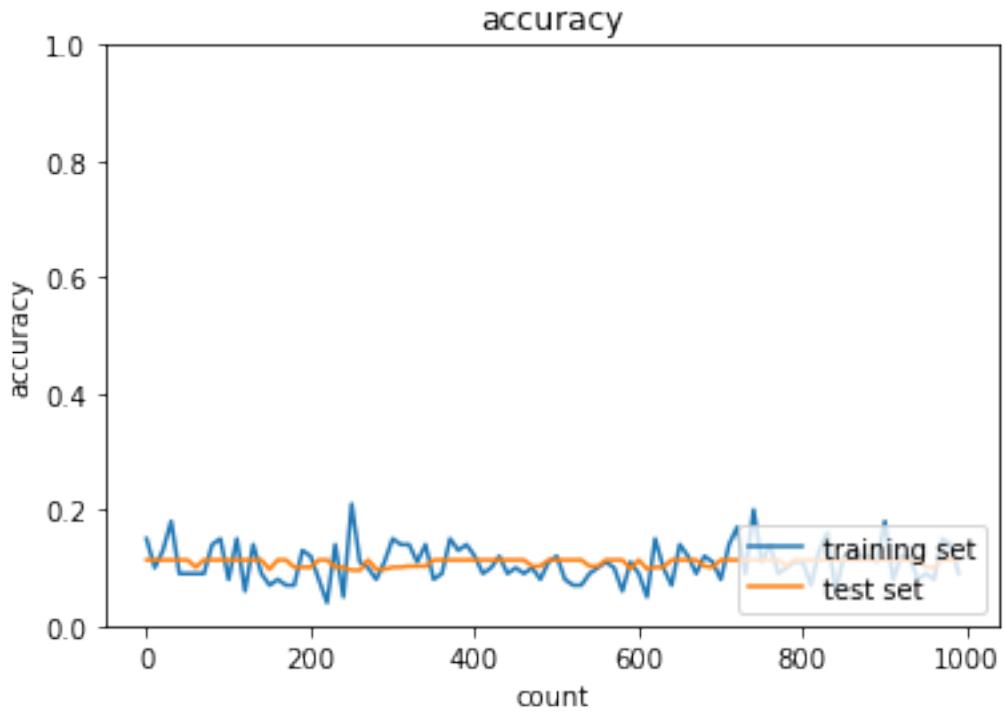
```
plot_interval=10

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]

    #
    grad = network.gradient(x_batch, d_batch)

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

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


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


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 Momentum

[2]:
```
#
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True,␣
 ↪one_hot_label=True)

print("     ")

# batch_normalization   ================================
# use_batchnorm = True
use_batchnorm = False
# ====================================================

network = MultiLayerNet(input_size=784, hidden_size_list=[40, 20],␣
 ↪output_size=10, activation='sigmoid', weight_init_std=0.01,
                        use_batchnorm=use_batchnorm)

iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.01
#
momentum = 0.9
```

```
train_loss_list = []
accuracies_train = []
accuracies_test = []

plot_interval=10

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]

    #
    grad = network.gradient(x_batch, d_batch)
    if i == 0:
        v = {}
    for key in ('W1', 'W2', 'W3', 'b1', 'b2', 'b3'):
        if i == 0:
            v[key] = np.zeros_like(network.params[key])
        v[key] = momentum * v[key] - learning_rate * grad[key]
        network.params[key] += v[key]

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

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


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()
```



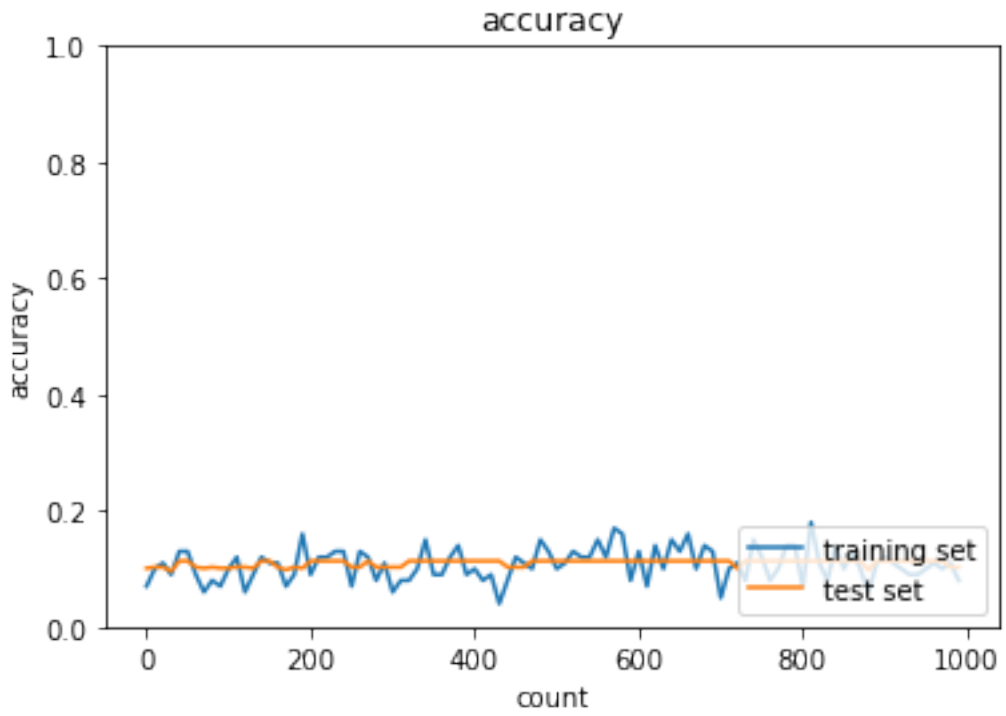
1.3 Momentum AdaGrad

= 1e-4

[3]:
```
# AdaGrad
#
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True,␣
 ↪one_hot_label=True)

print("     ")

# batch_normalization   ================================
# use_batchnorm = True
use_batchnorm = False
# ====================================================

network = MultiLayerNet(input_size=784, hidden_size_list=[40, 20],␣
 ↪output_size=10, activation='sigmoid', weight_init_std=0.01,
                        use_batchnorm=use_batchnorm)

iters_num = 1000
# iters_num = 500 #

train_size = x_train.shape[0]
```

```
batch_size = 100
learning_rate = 0.01

# AdaGrad
# ============================

momentum = 0.9

# ============================

train_loss_list = []
accuracies_train = []
accuracies_test = []

plot_interval=10

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]

    #
    grad = network.gradient(x_batch, d_batch)
    if i == 0:
        h = {}
    for key in ('W1', 'W2', 'W3', 'b1', 'b2', 'b3'):

        #
        # ========================================
        if i == 0:
            h[key] = np.zeros_like(network.params[key])
        h[key] = momentum * h[key] - learning_rate * grad[key]
        network.params[key] += h[key]

        # ========================================

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

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

```
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()
```

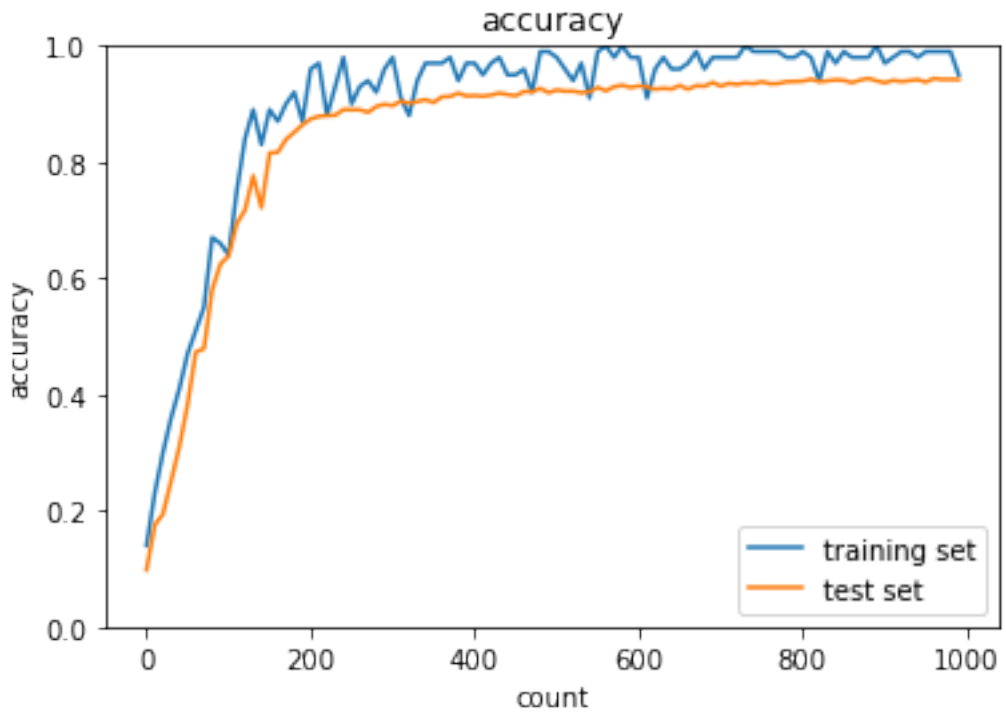


1.4 RSMprop

[4]:
```
#
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True,␣
↪one_hot_label=True)

print("     ")
```

```
# batch_normalization  ================================
# use_batchnorm = True
use_batchnorm = False
# ====================================================

network = MultiLayerNet(input_size=784, hidden_size_list=[40, 20],␣
 ↪output_size=10, activation='sigmoid', weight_init_std=0.01,
                        use_batchnorm=use_batchnorm)

iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.01
decay_rate = 0.99

train_loss_list = []
accuracies_train = []
accuracies_test = []

plot_interval=10

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]

    #
    grad = network.gradient(x_batch, d_batch)
    if i == 0:
        h = {}
    for key in ('W1', 'W2', 'W3', 'b1', 'b2', 'b3'):
        if i == 0:
            h[key] = np.zeros_like(network.params[key])
        h[key] *= decay_rate
        h[key] += (1 - decay_rate) * np.square(grad[key])
        network.params[key] -= learning_rate * grad[key] / (np.sqrt(h[key]) +␣
 ↪1e-7)

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

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


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()
```

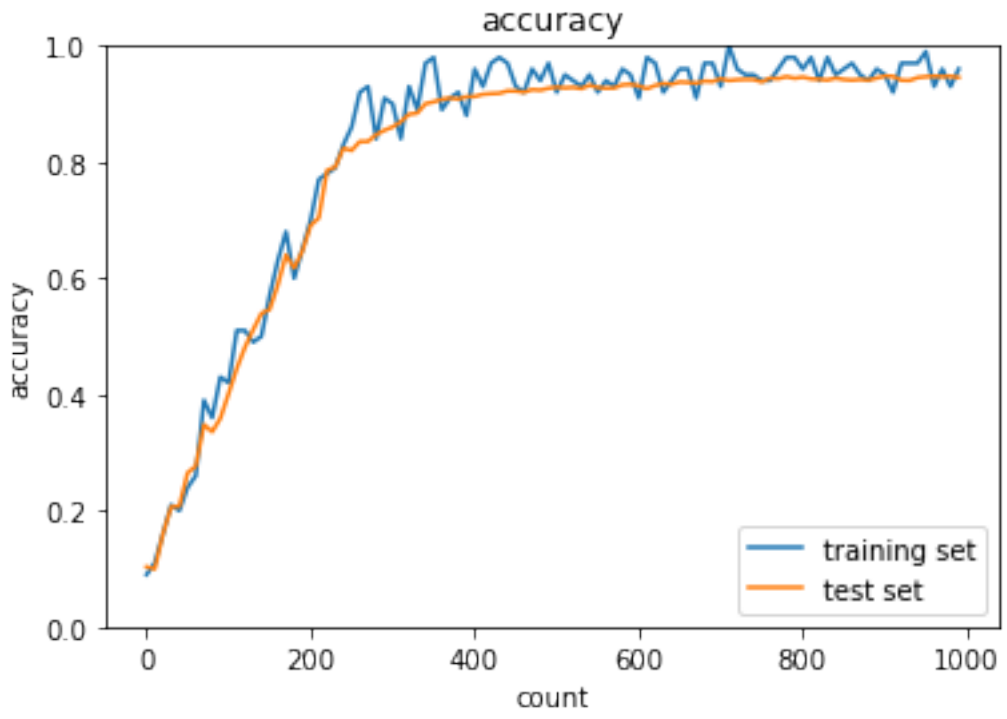


1.5 Adam

[5]:
```
#
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True,␣
↪one_hot_label=True)
```

```
print("     ")

# batch_normalization   ================================
# use_batchnorm = True
use_batchnorm = False
# =====================================================

network = MultiLayerNet(input_size=784, hidden_size_list=[40, 20],␣
 ↪output_size=10, activation='sigmoid', weight_init_std=0.01,
                        use_batchnorm=use_batchnorm)

iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.01
beta1 = 0.9
beta2 = 0.999

train_loss_list = []
accuracies_train = []
accuracies_test = []

plot_interval=10

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]

    #
    grad = network.gradient(x_batch, d_batch)
    if i == 0:
        m = {}
        v = {}
    learning_rate_t  = learning_rate * np.sqrt(1.0 - beta2 ** (i + 1)) / (1.0 -␣
 ↪beta1 ** (i + 1))
    for key in ('W1', 'W2', 'W3', 'b1', 'b2', 'b3'):
        if i == 0:
            m[key] = np.zeros_like(network.params[key])
            v[key] = np.zeros_like(network.params[key])

        m[key] += (1 - beta1) * (grad[key] - m[key])
        v[key] += (1 - beta2) * (grad[key] ** 2 - v[key])
        network.params[key] -= learning_rate_t * m[key] / (np.sqrt(v[key]) +␣
 ↪1e-7)
```

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

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


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()
```



1.6 [try]

1.7 [try]

sigmoid - gauss activation ReLU weight_init_std 'Xavier' 'He'

1.8 [try]

use_batchnorm True
