実装演習 2-2.学習率最適化手法

In [1]:

```
import sys, os
sys.path.append(os.pardir)  # 親ディレクトリのファイルをインポートするための設定
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
plt.style.use('ggplot')
```

In [2]:

```
# MNISTデータの読み込み
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True, one_hot_label=True)
```

In [4]:

```
print(f'x_train.shape: {x_train.shape}')
print(f'd_train.shape: {d_train.shape}')
print(f'x_test.shape: {x_test.shape}')
print(f'd_test.shape: {d_test.shape}')
```

```
x_train.shape: (60000, 784)
d_train.shape: (60000, 10)
x_test.shape: (10000, 784)
d_test.shape: (10000, 10)
```

オプティマイザ：SGD

In [5]:

```
use_batchnorm = False

# MNISTの画像サイズは28*28=784
# 出力は0〜9の分類なので10
# 活性化関数はsigmoid
network = MultiLayerNet(input_size=784, hidden_size_list=[40, 20], output_size=10, activa
tion='sigmoid',
                        weight_init_std=0.01, use_batchnorm=use_batchnorm)
```

In [12]:

```
iters_num = 1000  # 繰り返し回数
train_size = x_train.shape[0]  # 学習データ（MNIST）のサイズ
batch_size = 100  # ミニバッチのサイズ
learning_rate = 0.01  # 学習率

plot_interval=10  # グラフX軸の間隔
```

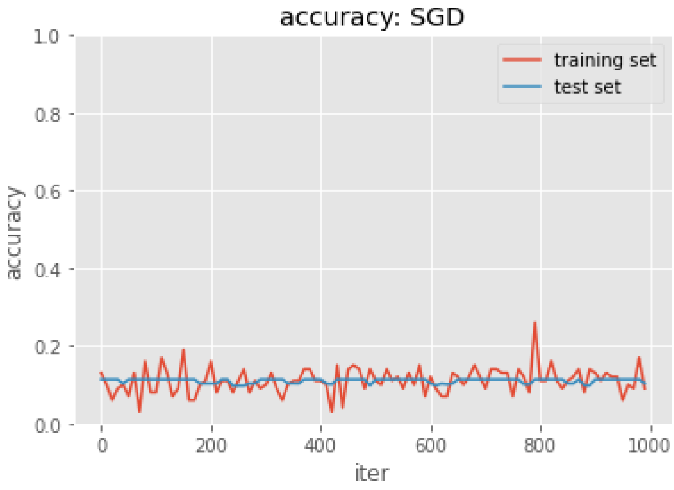
In [15]:

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

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]

    # 勾配
    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)

        if (i + 1) % (plot_interval * 10) == 0:
            print(f'Generation: {i+1}. 正答率(train)={accr_train}')
            print(f'          : {i+1}. 正答率(test)={accr_test}')
```

```
Generation: 100. 正答率(train)=0.12
          : 100. 正答率(test)=0.1135
Generation: 200. 正答率(train)=0.12
          : 200. 正答率(test)=0.1135
Generation: 300. 正答率(train)=0.13
          : 300. 正答率(test)=0.1135
Generation: 400. 正答率(train)=0.14
          : 400. 正答率(test)=0.1135
Generation: 500. 正答率(train)=0.1
          : 500. 正答率(test)=0.1135
Generation: 600. 正答率(train)=0.13
          : 600. 正答率(test)=0.1135
Generation: 700. 正答率(train)=0.13
          : 700. 正答率(test)=0.1135
Generation: 800. 正答率(train)=0.12
          : 800. 正答率(test)=0.1135
Generation: 900. 正答率(train)=0.08
          : 900. 正答率(test)=0.1135
Generation: 1000. 正答率(train)=0.12
          : 1000. 正答率(test)=0.1135
```

In [26]:

```
# グラフ表示
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()
plt.title("accuracy: SGD")
plt.xlabel("iter")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.show()
```



オプティマイザ：MOMENTUM

In [20]:

```
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)
```

In [21]:

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

plot_interval=10
```

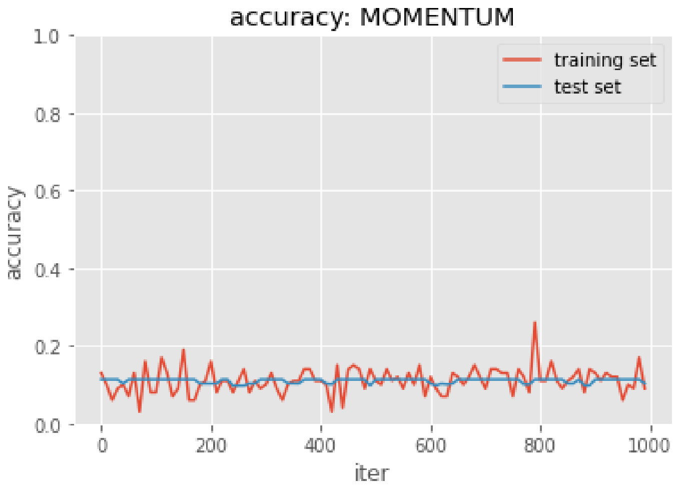
In [22]:

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

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:
            # イテレーションの最初に、vを初期化する
            v[key] = np.zeros_like(network.params[key])

        # momentumの更新量を計算
        v[key] = momentum * v[key] - learning_rate * grad[key]
        # パラメータの更新
        network.params[key] += v[key]

        # 誤差の計算
        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)

        if (i + 1) % (plot_interval * 10) == 0:
            print(f'Generation: {i+1}. 正答率(train)={accr_train}')
            print(f'          : {i+1}. 正答率(test)={accr_test}')
```

```
Generation: 100. 正答率(train)=0.08
          : 100. 正答率(test)=0.1135
Generation: 200. 正答率(train)=0.11
          : 200. 正答率(test)=0.1028
Generation: 300. 正答率(train)=0.09
          : 300. 正答率(test)=0.1135
Generation: 400. 正答率(train)=0.11
          : 400. 正答率(test)=0.1135
Generation: 500. 正答率(train)=0.14
          : 500. 正答率(test)=0.098
Generation: 600. 正答率(train)=0.07
          : 600. 正答率(test)=0.1135
Generation: 700. 正答率(train)=0.12
          : 700. 正答率(test)=0.1135
Generation: 800. 正答率(train)=0.26
          : 800. 正答率(test)=0.1135
Generation: 900. 正答率(train)=0.14
          : 900. 正答率(test)=0.098
Generation: 1000. 正答率(train)=0.09
          : 1000. 正答率(test)=0.1032
```

In [25]:

```
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()
plt.title("accuracy: MOMENTUM")
plt.xlabel("iter")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.show()
```



MomentumをもとにAdaGradを作ってみよう

θ = 1e-4 とする

In [48]:

```
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)
```

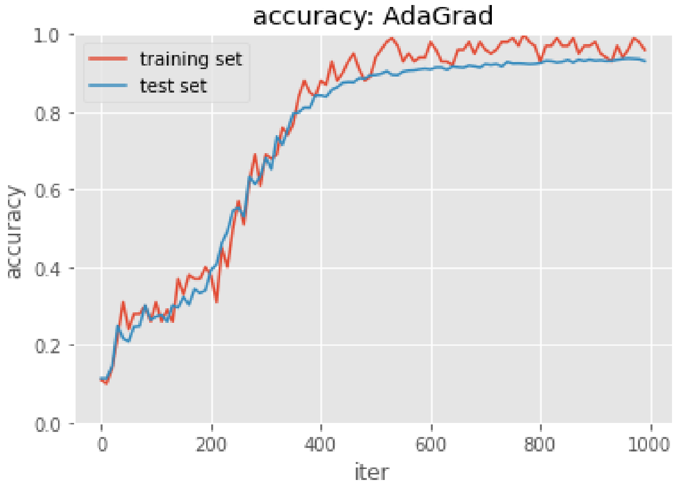
In [49]:

```
iters_num = 1000

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

# momentumは使用しない
theta = 1e-4

plot_interval=10
```

In [50]:
```
train_loss_list = []
accuracies_train = []
accuracies_test = []

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'):
        # 変更しよう =======================================
        if i == 0:
            h[key] = np.zeros_like(theta)  # hの初期値はθ

        h[key] = grad[key] * grad[key]
        network.params[key] -= learning_rate * grad[key] / (np.sqrt(h[key]) + theta)
        # =============================================

        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)

        if (i + 1) % (plot_interval * 10) == 0:
            print(f'Generation: {i+1}. 正答率(train)={accr_train}')
            print(f'          : {i+1}. 正答率(test)={accr_test}')
```

```
Generation: 100. 正答率(train)=0.26
          : 100. 正答率(test)=0.2645
Generation: 200. 正答率(train)=0.4
          : 200. 正答率(test)=0.3406
Generation: 300. 正答率(train)=0.61
          : 300. 正答率(test)=0.6339
Generation: 400. 正答率(train)=0.84
          : 400. 正答率(test)=0.8421
Generation: 500. 正答率(train)=0.89
          : 500. 正答率(test)=0.894
Generation: 600. 正答率(train)=0.94
          : 600. 正答率(test)=0.9115
Generation: 700. 正答率(train)=0.98
          : 700. 正答率(test)=0.9141
Generation: 800. 正答率(train)=0.97
          : 800. 正答率(test)=0.9236
Generation: 900. 正答率(train)=0.97
          : 900. 正答率(test)=0.9348
Generation: 1000. 正答率(train)=0.96
          : 1000. 正答率(test)=0.9309
```

In [51]:
```
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()
plt.title("accuracy: AdaGrad")
plt.xlabel("iter")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.show()
```



オプティマイザ：RMSProp

In [31]:
```
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)
```

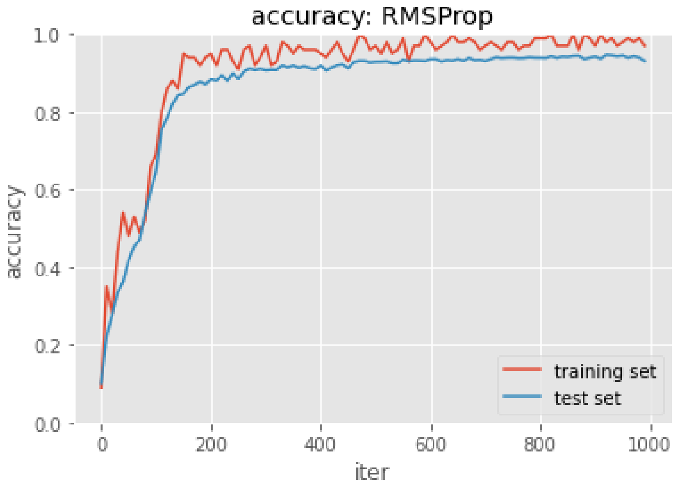
In [32]:
```
iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.01
decay_rate = 0.99

plot_interval=10
```

In [33]:

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

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)

        if (i + 1) % (plot_interval * 10) == 0:
            print(f'Generation: {i+1}. 正答率(train)={accr_train}')
            print(f'          : {i+1}. 正答率(test)={accr_test}')
```

```
Generation: 100. 正答率(train)=0.66
          : 100. 正答率(test)=0.595
Generation: 200. 正答率(train)=0.94
          : 200. 正答率(test)=0.8714
Generation: 300. 正答率(train)=0.94
          : 300. 正答率(test)=0.9111
Generation: 400. 正答率(train)=0.96
          : 400. 正答率(test)=0.9102
Generation: 500. 正答率(train)=0.96
          : 500. 正答率(test)=0.9276
Generation: 600. 正答率(train)=1.0
          : 600. 正答率(test)=0.931
Generation: 700. 正答率(train)=0.96
          : 700. 正答率(test)=0.9338
Generation: 800. 正答率(train)=0.99
          : 800. 正答率(test)=0.9396
Generation: 900. 正答率(train)=0.99
          : 900. 正答率(test)=0.9395
Generation: 1000. 正答率(train)=0.97
          : 1000. 正答率(test)=0.931
```

In [34]:
```
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()
plt.title("accuracy: RMSProp")
plt.xlabel("iter")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.show()
```



オプティマイザ：Adam

In [35]:
```
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)
```

In [36]:
```
iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.01
beta1 = 0.9
beta2 = 0.999

plot_interval=10
```

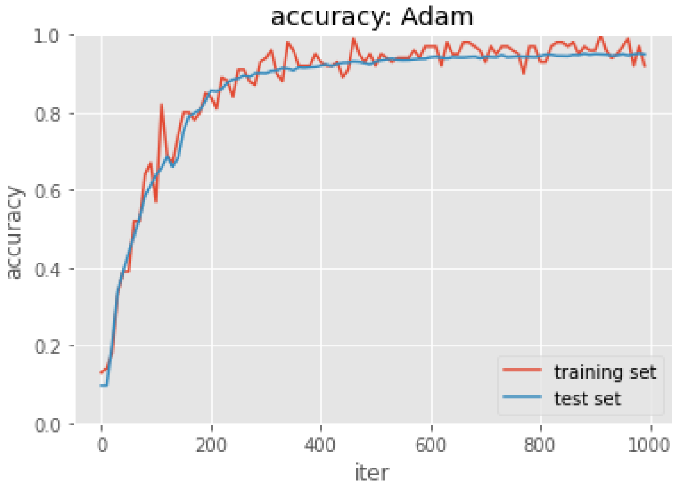
In [37]:

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

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)

        if (i + 1) % (plot_interval * 10) == 0:
            print(f'Generation: {i+1}. 正答率(train)={accr_train}')
            print(f'          : {i+1}. 正答率(test)={accr_test}')
```

```
Generation: 100. 正答率(train)=0.67
          : 100. 正答率(test)=0.6119
Generation: 200. 正答率(train)=0.85
          : 200. 正答率(test)=0.829
Generation: 300. 正答率(train)=0.93
          : 300. 正答率(test)=0.9013
Generation: 400. 正答率(train)=0.95
          : 400. 正答率(test)=0.9172
Generation: 500. 正答率(train)=0.95
          : 500. 正答率(test)=0.9237
Generation: 600. 正答率(train)=0.97
          : 600. 正答率(test)=0.9381
Generation: 700. 正答率(train)=0.96
          : 700. 正答率(test)=0.9395
Generation: 800. 正答率(train)=0.97
          : 800. 正答率(test)=0.9423
Generation: 900. 正答率(train)=0.96
          : 900. 正答率(test)=0.948
Generation: 1000. 正答率(train)=0.92
          : 1000. 正答率(test)=0.9491
```

In [38]:

```
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()
plt.title("accuracy: Adam")
plt.xlabel("iter")
plt.ylabel("accuracy")
plt.ylim(0, 1.0)
plt.show()
```



考察：

SGDとMomentumは、イテレーションを進めても学習が進まない。
AdaGradは、iter=200くらいまでは精度の上昇が緩やかだが、それ以降は順調に学習が進んでいった。
RMSPropとAdamは、早い段階から学習が進んでいった。

In [ ]: