実装演習：2-3.過学習

In [24]:
```
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
from common import optimizer
plt.style.use('ggplot')
```

In [2]:
```
# MNISTデータ読み込み
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)
```

In [3]:
```
# 過学習を再現するために、学習データを削減
x_train = x_train[:300]
d_train = d_train[:300]
```

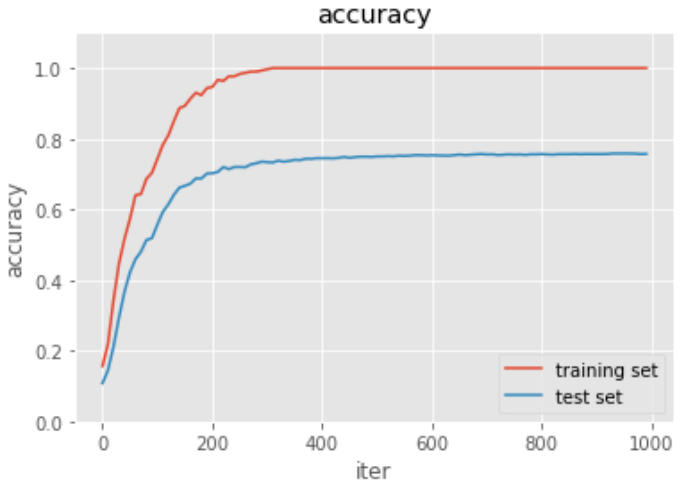
In [4]:
```
# モデル構築
network = MultiLayerNet(input_size=784,
                        hidden_size_list=[100, 100, 100, 100, 100, 100],
                        output_size=10)
# オプティマイザ
optimizer = optimizer.SGD(learning_rate=0.01)
```

In [5]:
```
iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100

plot_interval=10
```

In [6]:
```
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)
    optimizer.update(network.params, grad)

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

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

        if (i+1) % (plot_interval * 10) == 0:
            print(f'iter: {i+1}. accr_train={accr_train}, accr_test={accr_test}')
```

```
iter: 100. accr_train=0.7033333333333334, accr_test=0.5186
iter: 200. accr_train=0.9433333333333334, accr_test=0.7014
iter: 300. accr_train=0.9933333333333333, accr_test=0.7352
iter: 400. accr_train=1.0, accr_test=0.745
iter: 500. accr_train=1.0, accr_test=0.7486
iter: 600. accr_train=1.0, accr_test=0.7526
iter: 700. accr_train=1.0, accr_test=0.7567
iter: 800. accr_train=1.0, accr_test=0.7558
iter: 900. accr_train=1.0, accr_test=0.7566
iter: 1000. accr_train=1.0, accr_test=0.7571
```

In [7]:
```
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")
plt.xlabel("iter")
plt.ylabel("accuracy")
plt.ylim(0, 1.1)
plt.show()
```



weight decay (重み減衰) による過学習の改善

L2正則化

In [8]:
```
network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100, 100],
output_size=10)
```

In [9]:
```
iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
learning_rate=0.01

plot_interval=10
hidden_layer_num = network.hidden_layer_num
print(f'hidden_layer_num: {hidden_layer_num}')

weight_decay_lambda = 0.1
print(f'正則化強度設定: {weight_decay_lambda}')
```

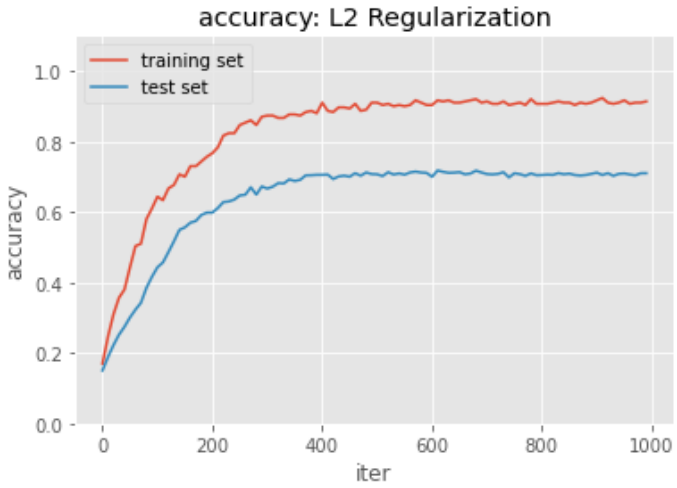
```
hidden_layer_num: 6
正則化強度設定: 0.1
```

In [10]:

```
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)
    weight_decay = 0

    for idx in range(1, hidden_layer_num+1):
        grad['W' + str(idx)] = network.layers['Affine' + str(idx)].dW + weight_decay_lambda *
network.params['W' + str(idx)]
        grad['b' + str(idx)] = network.layers['Affine' + str(idx)].db

        network.params['W' + str(idx)] -= learning_rate * grad['W' + str(idx)]
        network.params['b' + str(idx)] -= learning_rate * grad['b' + str(idx)]

        weight_decay += 0.5 * weight_decay_lambda * np.sqrt(np.sum(network.params['W'
+ str(idx)] ** 2))

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

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

        if (i+1) % (plot_interval * 10) == 0:
            print(f'iter: {i+1}. accr_train={accr_train}, accr_test={accr_test}')
```

```
iter: 100. accr_train=0.61, accr_test=0.4161
iter: 200. accr_train=0.7566666666666667, accr_test=0.5982
iter: 300. accr_train=0.87, accr_test=0.6722
iter: 400. accr_train=0.88, accr_test=0.7053
iter: 500. accr_train=0.91, accr_test=0.7075
iter: 600. accr_train=0.9033333333333333, accr_test=0.7109
iter: 700. accr_train=0.91, accr_test=0.7119
iter: 800. accr_train=0.9066666666666666, accr_test=0.7043
iter: 900. accr_train=0.91, accr_test=0.7081
iter: 1000. accr_train=0.9133333333333333, accr_test=0.7099
```

In [11]:
```
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: L2 Regularization")
plt.xlabel("iter")
plt.ylabel("accuracy")
plt.ylim(0, 1.1)
plt.show()
```



L1正則化

In [12]:
```
network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100, 100],
output_size=10)
```

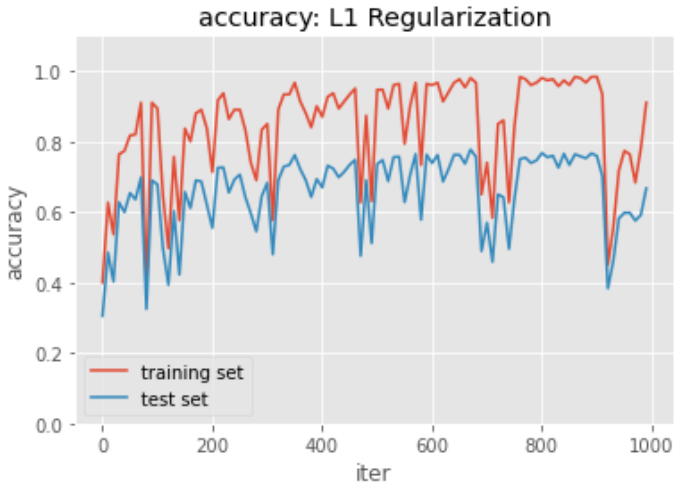
In [13]:
```
iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
learning_rate=0.1

plot_interval=10
hidden_layer_num = network.hidden_layer_num
print(f'hidden_layer_num: {hidden_layer_num}')

weight_decay_lambda = 0.005
print(f'正則化強度設定: {weight_decay_lambda}')
```

```
hidden_layer_num: 6
正則化強度設定: 0.005
```

In [14]:
```
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)
    weight_decay = 0

    for idx in range(1, hidden_layer_num+1):
        grad['W' + str(idx)] = network.layers['Affine' + str(idx)].dW + weight_decay_lambda *
np.sign(network.params['W' + str(idx)])
        grad['b' + str(idx)] = network.layers['Affine' + str(idx)].db

        network.params['W' + str(idx)] -= learning_rate * grad['W' + str(idx)]
        network.params['b' + str(idx)] -= learning_rate * grad['b' + str(idx)]

        weight_decay += weight_decay_lambda * np.sum(np.abs(network.params['W' + str(i
dx)]))

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

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

        if (i+1) % (plot_interval * 10) == 0:
            print(f'iter: {i+1}. accr_train={accr_train}, accr_test={accr_test}')
```

```
iter: 100. accr_train=0.91, accr_test=0.6895
iter: 200. accr_train=0.8366666666666667, accr_test=0.6201
iter: 300. accr_train=0.8333333333333334, accr_test=0.6451
iter: 400. accr_train=0.9, accr_test=0.6933
iter: 500. accr_train=0.63, accr_test=0.5113
iter: 600. accr_train=0.9633333333333334, accr_test=0.7625
iter: 700. accr_train=0.65, accr_test=0.4889
iter: 800. accr_train=0.9666666666666667, accr_test=0.7462
iter: 900. accr_train=0.9833333333333333, accr_test=0.7662
iter: 1000. accr_train=0.91, accr_test=0.6668
```

In [15]:
```
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: L1 Regularization")
plt.xlabel("iter")
plt.ylabel("accuracy")
plt.ylim(0, 1.1)
plt.show()
```



Dropout

In [19]:
```
class Dropout:
    def __init__(self, dropout_ratio=0.5):
        self.dropout_ratio = dropout_ratio
        self.mask = None

    def forward(self, x, train_flg=True):
        if train_flg:
            self.mask = np.random.rand(*x.shape) > self.dropout_ratio
            return x * self.mask
        else:
            return x * (1.0 - self.dropout_ratio)

    def backward(self, dout):
        return dout * self.mask
```

In [20]:
```
# ドロップアウト設定
use_dropout = True
dropout_ratio = 0.15
```

In [21]:
```
network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100, 100],
output_size=10,
                        weight_decay_lambda=weight_decay_lambda, use_dropout=use_dropout,
dropout_ratio=dropout_ratio)
```

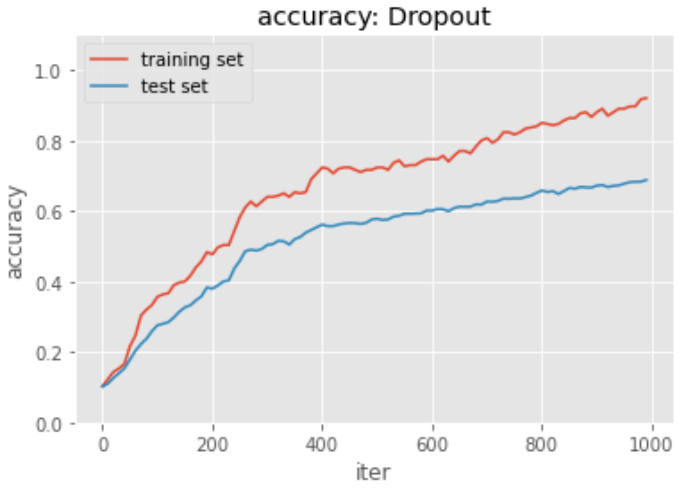
In [25]:
```
sgd = optimizer.SGD(learning_rate=0.01)
```

In [26]:
```
iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
plot_interval=10
```

In [27]:
```
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)
    sgd.update(network.params, grad)

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

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

        if (i+1) % (plot_interval * 10) == 0:
            print(f'iter: {i+1}. accr_train={accr_train}, accr_test={accr_test}')
```

```
iter: 100. accr_train=0.3333333333333333, accr_test=0.2591
iter: 200. accr_train=0.48333333333333334, accr_test=0.383
iter: 300. accr_train=0.6266666666666667, accr_test=0.4921
iter: 400. accr_train=0.7066666666666667, accr_test=0.554
iter: 500. accr_train=0.7166666666666667, accr_test=0.5759
iter: 600. accr_train=0.7466666666666667, accr_test=0.6011
iter: 700. accr_train=0.8, accr_test=0.618
iter: 800. accr_train=0.84, accr_test=0.6512
iter: 900. accr_train=0.8666666666666667, accr_test=0.6662
iter: 1000. accr_train=0.92, accr_test=0.6877
```

In [28]:
```
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: Dropout")
plt.xlabel("iter")
plt.ylabel("accuracy")
plt.ylim(0, 1.1)
plt.show()
```



Dropout + L1正則化

In [29]:
```
# ドロップアウト設定
use_dropout = True
dropout_ratio = 0.08
```

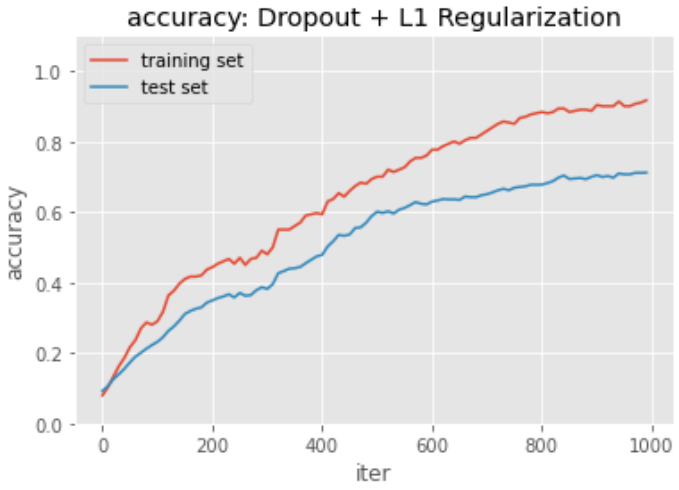
In [30]:
```
network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100, 100],
output_size=10,
                        use_dropout = use_dropout, dropout_ratio = dropout_ratio)
```

In [31]:
```
iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
learning_rate=0.01

hidden_layer_num = network.hidden_layer_num
plot_interval=10

# 正則化強度設定
weight_decay_lambda=0.004
```

In [32]:
```
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)
    weight_decay = 0

    for idx in range(1, hidden_layer_num+1):
        grad['W' + str(idx)] = network.layers['Affine' + str(idx)].dW + weight_decay_lambda *
np.sign(network.params['W' + str(idx)])
        grad['b' + str(idx)] = network.layers['Affine' + str(idx)].db
        network.params['W' + str(idx)] -= learning_rate * grad['W' + str(idx)]
        network.params['b' + str(idx)] -= learning_rate * grad['b' + str(idx)]
        weight_decay += weight_decay_lambda * np.sum(np.abs(network.params['W' + str(i
dx)]))

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

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

        if (i+1) % (plot_interval * 10) == 0:
            print(f'iter: {i+1}. accr_train={accr_train}, accr_test={accr_test}')
```

```
iter: 100. accr_train=0.28, accr_test=0.2225
iter: 200. accr_train=0.43666666666666665, accr_test=0.3428
iter: 300. accr_train=0.49, accr_test=0.3866
iter: 400. accr_train=0.5966666666666667, accr_test=0.4741
iter: 500. accr_train=0.6933333333333334, accr_test=0.5883
iter: 600. accr_train=0.76, accr_test=0.6215
iter: 700. accr_train=0.82, accr_test=0.6475
iter: 800. accr_train=0.88, accr_test=0.6772
iter: 900. accr_train=0.8866666666666667, accr_test=0.6995
iter: 1000. accr_train=0.9166666666666666, accr_test=0.7115
```

In [33]:
```
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: Dropout + L1 Regularization")
plt.xlabel("iter")
plt.ylabel("accuracy")
plt.ylim(0, 1.1)
plt.show()
```



考察：

最初の何もしないデータでは、学習を進めると訓練セットに対しては精度100%になるのに対し、テストセットに対して75%程度となった。
これは訓練セット＞テストセットで過学習と言える。また、精度が100%になる時点で、過学習を疑うべきである。

L2正則化では、過学習自体は改善していないものの、訓練セットの精度が91%程度で収束している。
何もしないときに比べ、過学習は多少改善されているように見える。

L1正則化では、グラフがガクガクに変化している。これは、特徴量を一部完全に捨てていることに起因する。

ドロップアウトをすると、学習が収束するまで時間がかかるようになった。
ドロップアウト＋L1正則化で、ドロップアウトしただけに比べ、テストセットの精度が68%から71%まで改善した。

In [ ]:
```
```