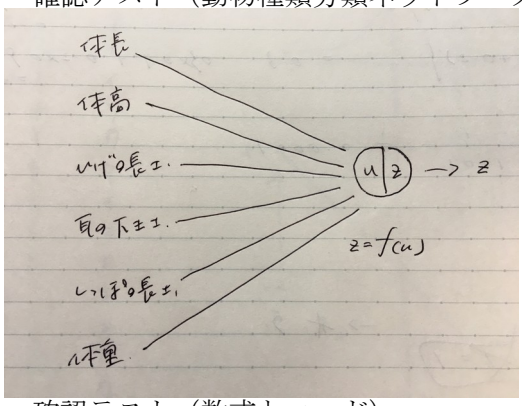
深層学習（前編）

Ｓｅｃｔｉｏｎ１）入力層～中間層
・確認テスト（動物種類分類ネットワーク）

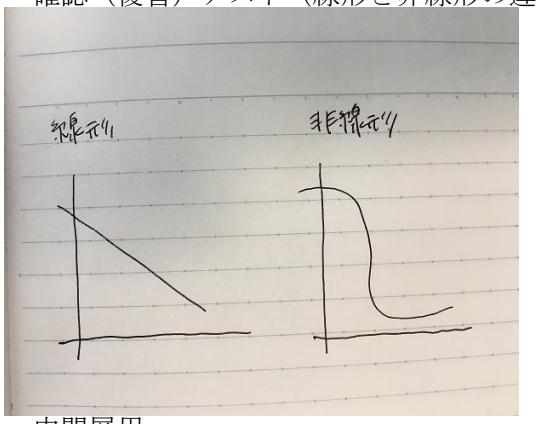


・確認テスト（数式とコード）
　ｕ１＝ｎｐ．ｄｏｔ（Ｗ１、ｘ）＋ｂ１
　※内積は行列の次元を合わせること。合わせない場合エラーとなる
・深層学習１－６～演習問題

In [13]:
```
#順伝播（単層・単ユニット）

#重み
#W = np.array([[0.1], [0.2], [0.3]])

##試してみよう_配列の初期化
#W = np.zeros(2)
#W = np.ones(2)
#W = np.random.rand(2)
W = np.random.randint(5, size=(3))

print_vec("重み", W)


#バイアス
#b = 0.5

##試してみよう_数値の初期化
#b = np.random.rand() # 0~1のランダム数値
b = np.random.rand() * 10 -5  # -5~5のランダム数値

print_vec("バイアス", b)

#入力値
x = np.array([2, 3,3])
print_vec("入力", x)


#総入力
u = np.dot(x, W) + b
print_vec("総入力", u)

#中間層出力
z = functions.relu(u)
print_vec("中間層出力", z)
```

```
*** 重み ***
[0 4 4]

*** バイアス ***
-4.066864334617727

*** 入力 ***
[2 3 3]

*** 総入力 ***
19.933135665382274

*** 中間層出力 ***
19.933135665382274
```

Ｓｅｃｔｉｏｎ２）活性化関数
・活性化関数
　ニューラルネットワークにおいて次の層への出力の強弱を決める非線形の関数
・確認（復習）テスト（線形と非線形の違い）



・中間層用
　－ステップ関数（現在ディープラーニングでは使われていない）
　　閾値越えで発火
　　出力は０ｏｒ１→課題は０～１の間を表現できないこと
　－シグモイド（ロジスティック）関数
　　０～１の間を緩やかに変化→信号の強弱を表現できるように
　　大きな値では出力の変化が微小→勾配消失問題
　－ＲｅＬＵ関数
　　今もっとも使われている活性化関数
　　勾配消失問題の回避とスパース化に貢献
　※シグモイドかＲｅＬＵか→問題の構造で異なる
・出力層用
　－ソフトマックス関数
　－恒等写像
　－シグモイド（ロジスティック）関数
・深層学習１－８～実装演習

In [15]:
```
#順伝播（単層・複数ユニット）

#重み
#W = np.array([
#    [0.1, 0.2, 0.3],
#    [0.2, 0.3, 0.4],
#    [0.3, 0.4, 0.5],
#    [0.4, 0.5, 0.6]
#])

## 試してみよう_配列の初期化
#W = np.zeros((4,3))
#W = np.ones((4,3))
#W = np.random.rand(4,3)
W = np.random.randint(5, size=(4,3))

print_vec("重み", W)

#バイアス
b = np.array([0.1, 0.2, 0.3])
print_vec("バイアス", b)

#入力値
x = np.array([1.0, 5.0, 2.0, -1.0])
print_vec("入力", x)


# 総入力
u = np.dot(x, W) + b
print_vec("総入力", u)
```

```
# 中間層出力
#z = functions.sigmoid(u)
z = functions.relu(u)
print_vec("中間層出力", z)
```

```
*** 重み ***
[[1 1 1]
 [1 4 2]
 [0 1 1]
 [2 0 4]]

*** バイアス ***
[0.1 0.2 0.3]

*** 入力 ***
[ 1.  5.  2. -1.]

*** 総入力 ***
[ 4.1 23.2  9.3]

*** 中間層出力 ***
[ 4.1 23.2  9.3]
```

Ｓｅｃｔｉｏｎ３）出力層

３－１　誤差関数

・事前に用意するデータ
　入力データ
　訓練データ（正解値）
・誤差関数
　出力と訓練データの差を求める

３－２　出力層の活性化関数

・中間層との違い→活性化関数が異なる
　値の強弱：中間層…しきい値の前後で信号の強弱を調整
　　出力層…信号の大きさ（比率）をそのまま変換
　確率出力：分類問題の場合、出力層の出力は０～１の範囲とし、総和を１とする必要がある
・出力層用の活性化関数
　－ソフトマックス関数
　－恒等写像
　－シグモイド（ロジスティック）関数
・出力層の種類

	回帰	二値分類	多クラス分類
活性化関数	恒等写像 $f(u)=u$	シグモイド関数 $f(u)=\frac{1}{1+e^{-u}}$	ソフトマックス関数 $f(i,u)=\frac{e^{-u_i}}{\sum_{k=1}^{k} e^{-u_k}}$
誤差関数	二乗誤差	交差エントロピー	

※活性化関数と誤差関数の組み合わせは計算の相性が良いため上記となる

Ｓｅｃｔｉｏｎ４）勾配降下法

・勾配降下法
　深層学習の目的
　→学習を通して誤差を最小にするネットワークを作成すること
　→誤差Ｅ（ｗ）を最小にするパラメータを発見すること
　学習率の値によって学習の効率が大きく異なる
　→大きくしすぎた場合：最小値にいつまでもたどり着かず発散してしまう
　→小さくしすぎた場合：発散することはないが、収束するまでに時間がかかってしまう
　　大域的極小解にとらわれてしまう
　学習率の決定、収束率向上のためのアルゴリズム
　－Ｍｏｍｅｍｔｕｍ
　－ＡｄａＧｒａｄ
　－Ａｄａｄｅｌｔａ
　－Ａｄａｍ
・確率的勾配降下法（ＳＧＤ）
　ランダムに抽出したサンプルの誤差⇔勾配降下法：全サンプルの平均誤差
　メリット
　－データが冗長な場合の計算コストの軽減
　－望まない局所極小解に収束するリスクの低減
　－オンライン学習が可能（誤差を１訓練データ毎に更新できる）

・ミニバッチ勾配降下法
ランダムに分割したデータの集合（ミニバッチ）D_tに属するサンプルの平均誤差
→確率的勾配降下法のメリットを損なわず、計算機の計算資源を有効活用できる
　ＣＰＵを利用したスレッド並列化、ＧＰＵを利用したＳＩＭＤ並列化
・誤差勾配の計算
数値微分：プログラムで微小な数値を生成し疑似的に微分を計算する一般的な手法

$$\frac{\partial E}{\partial w} \approx \frac{E(W_m + h) - E(W_m - h)}{2h}$$

デメリット：計算負荷が大きい→誤差逆伝播法を利用する

Ｓｅｃｔｉｏｎ５）誤差逆伝播法
・誤差逆伝播法
算出された誤差を出力層側から順に微分し前の層へ伝播。
必要最小限の計算で各パラメータでの微分値を解析的に計算。
・誤差勾配の計算

$$E(y) = \frac{1}{2}\sum_{j=1}^{J}(y_j - d_j)^2 = \frac{1}{2}\|y - d\|^2 \text{：誤差関数}$$

$y = u^{(L)}$：出力層の活性関数（恒等写像）
$u^{(l)} = w^{(l)} z^{(l-1)} + b^{(l)}$：総入力の計算

$$\frac{\partial E}{\partial w_{ji}^{(2)}} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial u}\frac{\partial u}{\partial w_{ji}^{(2)}}$$

$$\frac{\partial E}{\partial y} = \frac{\partial}{\partial y}\frac{1}{2}\|y - d\|^2 = y - d$$

$$\frac{\partial y}{\partial u} = \frac{\partial u}{\partial u} = 1$$

$$\frac{\partial u(w)}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}}\left(w^{(l)} z^{(l-1)} + b^{(l)}\right)$$

$$= \frac{\partial}{\partial w_{ji}}\left(\begin{bmatrix} w_{11}z_1 + \ldots + w_{1i}z_i + \ldots + w_{1I}z_I \\ \vdots \\ w_{ji}z_1 + \ldots + w_{ji}z_i + \ldots + w_{jI}z_I \\ \vdots \\ w_{J1}z_1 + \ldots + w_{Ji}z_i + \ldots + w_{JI}z_I \end{bmatrix} + \begin{bmatrix} b_1 \\ \vdots \\ b_i \\ \vdots \\ b_I \end{bmatrix}\right) = \begin{bmatrix} 0 \\ \vdots \\ z_i \\ \vdots \\ 0 \end{bmatrix}$$

$$\frac{\partial E}{\partial y}\frac{\partial y}{\partial u}\frac{\partial u}{\partial w_{ji}^{(2)}} = (y - d)\cdot\begin{bmatrix} 0 \\ \vdots \\ z_i \\ \vdots \\ 0 \end{bmatrix} = (y_j - d_j) z_i$$

・深層学習１－１５～実装演習

In [15]:

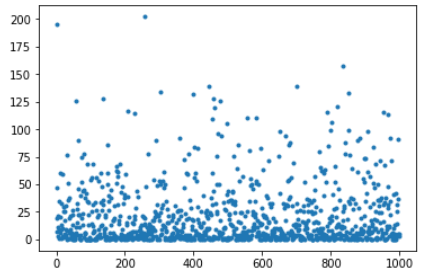
```
#サンプルとする関数
#yの値を予想するAI

def f(x):
    y = 3 * x[0] + 2 * x[1]
    return y

#初期設定
def init_network():
    # print("##### ネットワークの初期化 #####")
    network = {}
    nodesNum = 10
    network['W1'] = np.random.randn(2, nodesNum)
    network['W2'] = np.random.randn(nodesNum)
    network['b1'] = np.random.randn(nodesNum)
    network['b2'] = np.random.randn()

    # print_vec("重み1", network['W1'])
    # print_vec("重み2", network['W2'])
    # print_vec("バイアス1", network['b1'])
    # print_vec("バイアス2", network['b2'])

    return network

#順伝播
def forward(network, x):
    # print("##### 順伝播開始 #####")

    W1, W2 = network['W1'], network['W2']
    b1, b2 = network['b1'], network['b2']
    u1 = np.dot(x, W1) + b1
    #z1 = functions.relu(u1)

    ## 試してみよう
    z1 = functions.sigmoid(u1)
```

```
    u2 = np.dot(z1, W2) + b2
    y = u2

    # print_vec("総入力1", u1)
    # print_vec("中間層出力1", z1)
    # print_vec("総入力2", u2)
    # print_vec("出力1", y)
    # print("出力合計: " + str(np.sum(y)))

    return z1, y

# 誤差逆伝播
def backward(x, d, z1, y):
    # print("\n##### 誤差逆伝播開始 #####")

    grad = {}

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

    # 出力層でのデルタ
    delta2 = functions.d_mean_squared_error(d, y)
    # b2の勾配
    grad['b2'] = np.sum(delta2, axis=0)
    # W2の勾配
    grad['W2'] = np.dot(z1.T, delta2)
    # 中間層でのデルタ
    #delta1 = np.dot(delta2, W2.T) * functions.d_relu(z1)

    ## 試してみよう
    delta1 = np.dot(delta2, W2.T) * functions.d_sigmoid(z1)

    delta1 = delta1[np.newaxis, :]
    # b1の勾配
    grad['b1'] = np.sum(delta1, axis=0)
    x = x[np.newaxis, :]
    # W1の勾配
    grad['W1'] = np.dot(x.T, delta1)
```

```

    # print_vec("偏微分_重み1", grad["W1"])
    # print_vec("偏微分_重み2", grad["W2"])
    # print_vec("偏微分_バイアス1", grad["b1"])
    # print_vec("偏微分_バイアス2", grad["b2"])

    return grad

# サンプルデータを作成
data_sets_size = 100000
data_sets = [0 for i in range(data_sets_size)]

for i in range(data_sets_size):
    data_sets[i] = {}
    # ランダムな値を設定
    data_sets[i]['x'] = np.random.rand(2)

    ## 試してみよう_入力値の設定
    data_sets[i]['x'] = np.random.rand(2) * 10 -5 # -5～5のランダム数値

    # 目標出力を設定
    data_sets[i]['d'] = f(data_sets[i]['x'])

losses = []
# 学習率
learning_rate = 0.07

# 抽出数
epoch = 1000

# パラメータの初期化
network = init_network()
# データのランダム抽出
random_datasets = np.random.choice(data_sets, epoch)

```

```python
#勾配降下の繰り返し
for dataset in random_datasets:
    x, d = dataset['x'], dataset['d']
    z1, y = forward(network, x)
    grad = backward(x, d, z1, y)
    #パラメータに勾配適用
    for key in ('W1', 'W2', 'b1', 'b2'):
        network[key]  -= learning_rate * grad[key]

    #誤差
    loss = functions.mean_squared_error(d, y)
    losses.append(loss)

print("##### 結果表示 #####")
lists = range(epoch)


plt.plot(lists, losses, '.')
#グラフの表示
plt.show()
```

```
##### 結果表示 #####
```



・修了課題について
Ｑ１．課題の目的とは？どのような工夫ができそうか
　ｉｒｉｓデータを使って回帰または分類を行う。
　以下の事項について、工夫する余地がある。
　－入力層のノード数
　－中間層の層数・ノード数
　－回帰または分類に応じた出力層の活性化関数・誤差関数
Ｑ２．課題を分類タスクで解く場合の意味は何か
　該当のデータがどのクラスに属するかを解くことが、課題の意味・目的となる。
Ｑ３．ｉｒｉｓデータとは何か２行で述べよ
　アヤメの花の特徴量を記録したデータ。品種としてセトナ、ヴァージカラー、ヴァージニカの
　３品種、特徴量としてがく片と花びらそれぞれの長さと幅が記録されている。

深層学習（後編）

Ｓｅｃｔｉｏｎ１）勾配消失問題
１－１　活性化関数
・勾配消失問題
　誤差逆伝播法を使用していると、下位層に進んでいくにつれて勾配が緩やかになっていく
　→勾配降下法による更新で誤差項の値が０に近づいて行ってしまう
　シグモイド関数は大きな値では出力の変化が微小なため、勾配消失問題を引き起こすことが多い
　→ＲｅＬＵ関数を使用することで回避する

１－２　初期値の設定方法
・重みの初期値設定（Ｘａｖｉｅｒ）
　ＲｅＬＵ関数、シグモイド関数、双曲線正接関数と共に使用する
　重みの要素を前の層のノード数の平方根で除算
・重みの初期値設定（Ｈｅ）
　ＲｅＬＵ関数と共に使用する（勾配消失問題を回避するためにＲｅＬＵ関数に特化）
　重みの要素を前の層のノード数の平方根で除算後に√２を乗算
・重みの初期値に０を設定する
　誤差逆伝播法ですべての重みの値が同じように更新されてしまう
　→パラメータのチューニングが行われなくなってしまう
　たくさんの重みをもつ意味がなくなってしまうためディープラーニングの手法上ナンセンス
　ランダムな初期値が必要

１－３　バッチ正規化
・バッチ正規化とは？
　ミニバッチ単位で入力値のデータの偏りを抑制
・バッチ正規化の使いどころとは？
　活性化関数に値を渡す前後にバッチ正規化の処理を孕んだ層を加える
・一般的に考えられるバッチ正規化の効果
　→正規化によってデータのばらつきが抑えられることにより計算の高速化が期待できる
　→勾配消失問題が起きづらくなる
・深層学習２－５～実装演習
　Ｓｉｇｍｏｉｄｅ－Ｈｅ

In [3]:
```python
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 data.mnist import load_mnist
from PIL import Image
import pickle
from common import functions
import matplotlib.pyplot as plt

# mnistをロード
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True, one_hot_label=True)
train_size = len(x_train)

print("データ読み込み完了")

#入力層サイズ
input_layer_size = 784
#中間層サイズ
hidden_layer_1_size = 40
hidden_layer_2_size = 20
#出力層サイズ
output_layer_size = 10
#繰り返し数
iters_num = 2000
#ミニバッチサイズ
batch_size = 100
#学習率
learning_rate = 0.1
#描写頻度
plot_interval=10
```

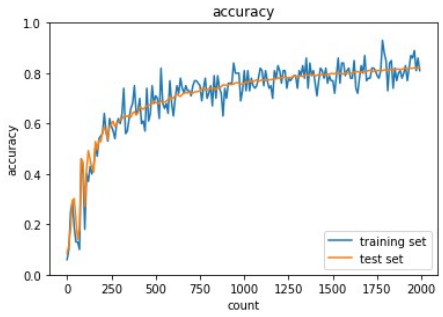
```
31
32  # 初期設定
33  def init_network():
34      network = {}
35
36      ########## 変更箇所 ##############
37
38      # Xavierの初期値
39      #network['W1'] = np.random.randn(input_layer_size, hidden_layer_1_size) / (np.sqrt(input_layer_size))
40      #network['W2'] = np.random.randn(hidden_layer_1_size, hidden_layer_2_size) / (np.sqrt(hidden_layer_1_size))
41      #network['W3'] = np.random.randn(hidden_layer_2_size, output_layer_size) / (np.sqrt(hidden_layer_2_size))
42      # Heの初期値
43      network['W1'] = np.random.randn(input_layer_size, hidden_layer_1_size) / np.sqrt(input_layer_size) * np.sqrt(2)
44      network['W2'] = np.random.randn(hidden_layer_1_size, hidden_layer_2_size) / np.sqrt(hidden_layer_1_size) * np.sqrt(2)
45      network['W3'] = np.random.randn(hidden_layer_2_size, output_layer_size) / np.sqrt(hidden_layer_2_size) * np.sqrt(2)
46
47      ###################################
48
49      network['b1'] = np.zeros(hidden_layer_1_size)
50      network['b2'] = np.zeros(hidden_layer_2_size)
51      network['b3'] = np.zeros(output_layer_size)
52
53      return network
54
55  # 順伝播
56  def forward(network, x):
57      W1, W2, W3 = network['W1'], network['W2'], network['W3']
58      b1, b2, b3 = network['b1'], network['b2'], network['b3']
59      hidden_f = functions.sigmoid
60
61      u1 =  np.dot(x, W1) + b1
62      z1 = hidden_f(u1)
```

```
63      u2 =  np.dot(z1, W2) + b2
64      z2 = hidden_f(u2)
65      u3 =  np.dot(z2, W3) + b3
66      y = functions.softmax(u3)
67
68      return z1, z2, y
69
70  # 誤差逆伝播
71  def backward(x, d, z1, z2, y):
72      grad = {}
73
74      W1, W2, W3 = network['W1'], network['W2'], network['W3']
75      b1, b2, b3 = network['b1'], network['b2'], network['b3']
76      hidden_d_f = functions.d_sigmoid
77
78      # 出力層でのデルタ
79      delta3 = functions.d_softmax_with_loss(d, y)
80      # b3の勾配
81      grad['b3'] = np.sum(delta3, axis=0)
82      # W3の勾配
83      grad['W3'] = np.dot(z2.T, delta3)
84      # 2層でのデルタ
85      delta2 = np.dot(delta3, W3.T) * hidden_d_f(z2)
86      # b2の勾配
87      grad['b2'] = np.sum(delta2, axis=0)
88      # W2の勾配
89      grad['W2'] = np.dot(z1.T, delta2)
90      # 1層でのデルタ
91      delta1 = np.dot(delta2, W2.T) * hidden_d_f(z1)
92      # b1の勾配
93      grad['b1'] = np.sum(delta1, axis=0)
```

```
94      # W1の勾配
95      grad['W1'] = np.dot(x.T, delta1)
96
97      return grad
98
99  # パラメータの初期化
100 network = init_network()
101
102 accuracies_train = []
103 accuracies_test = []
104
105 # 正答率
106 def accuracy(x, d):
107     z1, z2, y = forward(network, x)
108     y = np.argmax(y, axis=1)
109     if d.ndim != 1 : d = np.argmax(d, axis=1)
110     accuracy = np.sum(y == d) / float(x.shape[0])
111     return accuracy
112
113 for i in range(iters_num):
114     # ランダムにバッチを取得
115     batch_mask = np.random.choice(train_size, batch_size)
116     # ミニバッチに対応する教師訓練画像データを取得
117     x_batch = x_train[batch_mask]
118     # ミニバッチに対応する訓練正解ラベルデータを取得する
119     d_batch = d_train[batch_mask]
120
121
122
```

```
    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) + '. 正答率(トレーニング) = ' + str(accr_train))
        print('                : ' + str(i+1) + '. 正答率(テスト) = ' + str(accr_test))

    #パラメータに勾配適用
    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)
#グラフの表示
plt.show()
```

```
Generation: 2000. 正答率(トレーニング) = 0.81
                : 2000. 正答率(テスト) = 0.8214
```



ＲｅＬＵ－Ｘａｖｉｅｒ

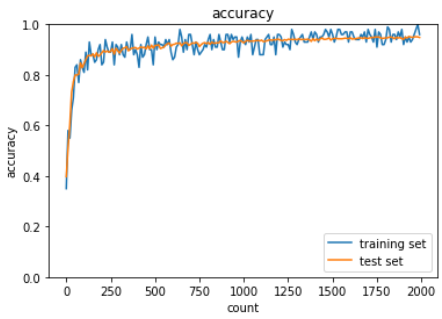
In [4]:

```
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 data.mnist import load_mnist
from PIL import Image
import pickle
from common import functions
import matplotlib.pyplot as plt

# mnistをロード
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True, one_hot_label=True)
train_size = len(x_train)

print("データ読み込み完了")

# 重み初期値補正係数
wieght_init = 0.01
#入力層サイズ
input_layer_size = 784
#中間層サイズ
hidden_layer_1_size = 40
hidden_layer_2_size = 20

#出力層サイズ
output_layer_size = 10
#繰り返し数
iters_num = 2000
#ミニバッチサイズ
batch_size = 100
```

```
# 学習率
learning_rate = 0.1
# 描写頻度
plot_interval=10

# 初期設定
def init_network():
    network = {}

    ########## 変更箇所 ##############

    # Xavierの初期値
    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))
    # Heの初期値
    #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

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

    ########## 変更箇所 ##############

    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

# 誤差逆伝播
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']

    ########## 変更箇所 ##############

    hidden_d_f = functions.d_relu

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


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

    return grad

# パラメータの初期化
network = init_network()

accuracies_train = []
accuracies_test = []

# 正答率
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):
    #ランダムにバッチを取得
    batch_mask = np.random.choice(train_size, batch_size)
    #ミニバッチに対応する教師訓練画像データを取得
    x_batch = x_train[batch_mask]
    #ミニバッチに対応する訓練正解ラベルデータを取得する
    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) + '. 正答率(トレーニング) = ' + str(accr_train))
        print('                : ' + str(i+1) + '. 正答率(テスト) = ' + str(accr_test))

    #パラメータに勾配適用
    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)
#グラフの表示
plt.show()
```

```
Generation: 2000. 正答率(トレーニング) = 0.96
                : 2000. 正答率(テスト) = 0.9481
```



※実務上では、Ｓｉｇｍｏｉｄｅ－ＸａｖｉｅｒよりもＳｉｇｍｏｉｄｅ－Ｈｅ、ＲｅＬＵ－Ｈｅよりも ＲｅＬＵ－Ｘａｖｉｅｒが効果を発揮することがある

Ｓｅｃｔｉｏｎ２）学習率最適化手法

２－１　モメンタム

・モメンタムの特徴

誤差をパラメータで微分したものと学習率の積を減算した後、現在の重みに前回の重みを減算した値と慣性の積を加算。慣性はハイパーパラメータ。０．５～０．９の間で設定

$$V_t = \mu V_{t-1} - \varepsilon \nabla E$$

$$W^{t+1} = W^t + V_t$$

μ：慣性

・モメンタムのメリット

大域的局所解を得やすい

谷間についてから最も低い位置（最適解）に行くまでの時間が早い

２－２　ＡｄａＧｒａｄ

・ＡｄａＧｒａｄの特徴

学習が進むにつれて学習係数を小さくする。最初は大きく学習し、次第に小さく学習する

誤差をパラメータで微分したものと再定義した学習率の積を減算する

$$h_0 = \theta$$

$$h_t = h_{t-1} + (\nabla E)^2$$

$$W^{t+1} = W^t - \varepsilon \frac{1}{\sqrt{h_t + \theta}} \nabla E$$

・ＡｄａＧｒａｄのメリット

緩やかな斜面に入った場合にも最適解に行くまでの時間が早い

・ＡｄａＧｒａｄの課題

学習率が徐々に小さくなるため鞍点問題を引き起こす

（次元が大きくなった場合に、極小値と思っていた点が別の視点から見ると極大値だった）

２－３　ＲＭＳＰｒｏｐ

・ＲＭＳＰｒｏｐの特徴

過去の勾配を徐々に忘れて新しい勾配の情報が大きく反映されるように加算

誤差をパラメータで微分したものと再定義した学習率の積を減算する

$$h_t = \alpha h_{t-1} + (1 - \alpha)(\nabla E)^2$$

$$W^{t+1} = W^t - \varepsilon \frac{1}{\sqrt{h_t + \theta}} \nabla E$$

・ＲＭＳＰｒｏｐのメリット

大域的局所解を得やすい

ハイパーパラメータの調整が必要な場合が少ない

２－４　Ａｄａｍ

・Ａｄａｍの特徴

以下の２つを孕んだ最適化アルゴリズム

－モメンタムの、過去の勾配の指数関数的減衰平均

－ＲＭＳＰｒｏｐの、過去の勾配の２乗の指数関数的減衰平均

・深層学習２－８～実装演習

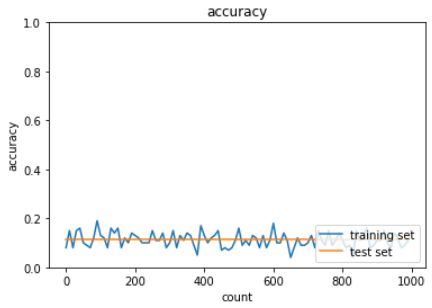
In [6]:

```
# AdaGradを作ってみよう
# データの読み込み
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True, one_hot_label=True)

print("データ読み込み完了")

# batch_normalizationの設定 ================================
# 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 # 処理を短縮

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

# AdaGradでは不必要
# ==============================

#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]
```

```
    #勾配
    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(network.params[key])
        #h[key] = momentum * h[key] - learning_rate * grad[key]
        #network.params[key] += h[key]
        if i == 0:
            h[key] = np.full_like(network.params[key],1e-4)
        else:
            h[key] += np.square(grad[key])
        network.params[key] -= learning_rate * grad[key] / (np.sqrt(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) + '. 正答率(トレーニング) = ' + str(accr_train))
        print('                : ' + str(i+1) + '. 正答率(テスト) = ' + 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)
#グラフの表示
plt.show()
```

```
Generation: 1000. 正答率(トレーニング) = 0.11
                : 1000. 正答率(テスト) = 0.1135
```



Ｓｅｃｔｉｏｎ３）過学習について

３－１　Ｌ１正則化、Ｌ２正則化

・正則化とは
　ネットワークの自由度（層数、ノード数、パラメータの値など）を制約すること

・正則化手法
　－Ｌ１正則化、Ｌ２正則化
　－ドロップアウト

・Ｗｅｉｇｈｔ　ｄｅｃａｙ（荷重減衰）
　－過学習の原因
　　重みが大きい値を取ることで過学習が発生することがある
　－過学習の解決策
　　誤差に対して正則化項を加算することで重みを抑制する
　　過学習が起こりそうな重みの大きさ以下で重みをコントロールし、かつ重みの大きさにばらつきを出す必要がある

・Ｌ１、Ｌ２正則化

$$E_n(w) + \frac{1}{p}\lambda \|x\|_p \quad \cdots \text{誤差関数にp ノルムを加える}$$

$$\|x\|_p = \left(|x_1|^p + \cdots + |x_n|^p \right) \quad \cdots \text{p ノルムの計算}$$

　p = 1 ：Ｌ１正則化 、p = 2 ：Ｌ２正則化

Ｌａｓｓｏ推定：Ｌ１正則化

Ｒｉｄｇｅ推定：Ｌ２正則化・・・計算リソースを食うが精度向上に寄与する

３－２　ドロップアウト
・過学習の課題
　ノード数が多い
　→ランダムにノードを削除して学習させる（ドロップアウト）
・ドロップアウトのメリット
　データ量を変化させずに異なるモデルを学習させていると解釈できる
・深層学習２－１３～実装演習
　ｗｅｉｇｈｔ＿ｄｅｃａｙ＿ｌａｍｄａの値
　→大きすぎても学習が進まない。小さすぎても過学習を抑制できない。
　　どれくらいの値とするのかはデータセットによっても異なる。

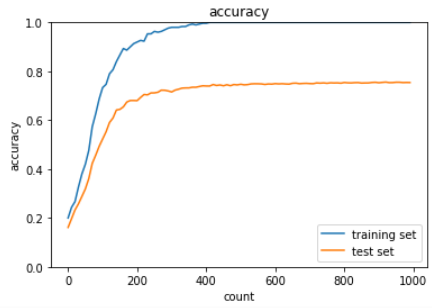
overfiting

In [2]:

```
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
from common import optimizer


(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)

print("データ読み込み完了")

#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]

network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100, 100], output_size=10)
optimizer = optimizer.SGD(learning_rate=0.01)

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

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)
    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)

        print('Generation: ' + str(i+1) + '. 正答率(トレーニング) = ' + str(accr_train))
        print('                : ' + str(i+1) + '. 正答率(テスト) = ' + 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)
#グラフの表示
plt.show()
```

```
Generation: 1000. 正答率(トレーニング) = 1.0
                : 1000. 正答率(テスト) = 0.7541
```



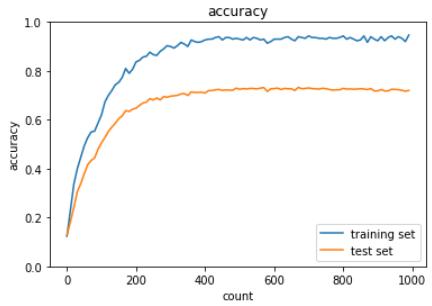
weight decay

L2

In [3]:

```
from common import optimizer

(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)

print("データ読み込み完了")

#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]


network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100, 100], output_size=10)


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
hidden_layer_num = network.hidden_layer_num

# 正則化強度設定 ======================================
weight_decay_lambda = 0.1
# =====================================================

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)

        print('Generation: ' + str(i+1) + '. 正答率(トレーニング) = ' + str(accr_train))
        print('                : ' + str(i+1) + '. 正答率(テスト) = ' + 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)
#グラフの表示
plt.show()
```

```
Generation: 1000. 正答率(トレーニング) = 0.9466666666666667
            : 1000. 正答率(テスト) = 0.7199
```



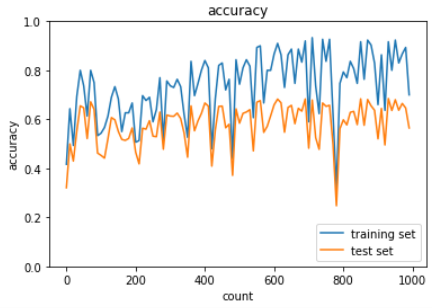
L1

In [7]:

```python
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)

print("データ読み込み完了")

#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]

network = MultiLayerNet(input_size=784, hidden_size_list=[100, 100, 100, 100, 100, 100], output_size=10)


iters_num = 1000
train_size = x_train.shape[0]
batch_size = 100
learning_rate=0.1

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

plot_interval=10
hidden_layer_num = network.hidden_layer_num

# 正則化強度設定 ======================================
#weight_decay_lambda = 0.005
weight_decay_lambda = 0.008
# =================================================

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]

```

```python
    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(idx)]))

    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)

        print('Generation: ' + str(i+1) + '. 正答率(トレーニング) = ' + str(accr_train))
        print('                : ' + str(i+1) + '. 正答率(テスト) = ' + 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)
#グラフの表示
plt.show()
```

```
Generation: 1000. 正答率(トレーニング) = 0.7
            : 1000. 正答率(テスト) = 0.5648
```



[try] weigth_decay_lambdaの値を変更して正則化の強さを確認しよう

・深層学習２－１４～実装演習
実務においてはスピードが求められるので理論をしっかり理解し仮説を立てて検証することが必要

Dropout

In [8]:

```python
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
```

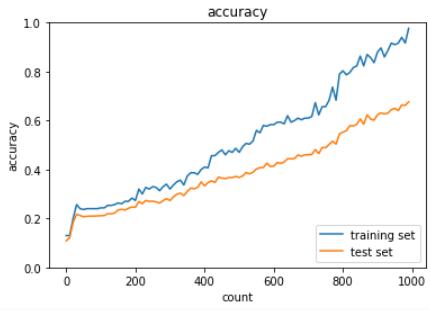
In [10]:

```python
from common import optimizer
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)

print("データ読み込み完了")

# 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]

# ドロップアウト設定 ======================================
use_dropout = True
#dropout_ratio = 0.15
dropout_ratio = 0.3
# ====================================================

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)
#optimizer = optimizer.SGD(learning_rate=0.01)
optimizer = optimizer.Momentum(learning_rate=0.01, momentum=0.9)
```

```
# optimizer = optimizer.AdaGrad(learning_rate=0.01)
# optimizer = optimizer.Adam()

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

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)
    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)

        print('Generation: ' + str(i+1) + '. 正答率(トレーニング) = ' + str(accr_train))
        print('                : ' + str(i+1) + '. 正答率(テスト) = ' + 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)
#グラフの表示
plt.show()
```

```
Generation: 1000. 正答率(トレーニング) = 0.9766666666666667
                : 1000. 正答率(テスト) = 0.6764
```



[try] dropout_ratioの値を変更してみよう

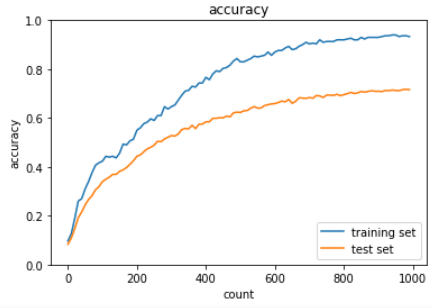
[try] optimizerとdropout_ratioの値を変更してみよう

Dropout + L1

In [11]:

```
from common import optimizer
(x_train, d_train), (x_test, d_test) = load_mnist(normalize=True)

print("データ読み込み完了")

#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]

# ドロップアウト設定 ======================================
use_dropout = True
dropout_ratio = 0.08
# ====================================================

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)

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

train_loss_list = []
accuracies_train = []
accuracies_test = []
hidden_layer_num = network.hidden_layer_num

plot_interval=10

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

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(idx)]))

    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)

        print('Generation: ' + str(i+1) + '. 正答率(トレーニング) = ' + str(accr_train))
        print('                : ' + str(i+1) + '. 正答率(テスト) = ' + 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)
#グラフの表示
plt.show()
```

```
Generation: 1000. 正答率(トレーニング) = 0.9333333333333333
                : 1000. 正答率(テスト) = 0.716
```



Ｓｅｃｔｉｏｎ４）畳み込みニューラルネットワークの概念
４－１　畳み込み層
・画像の場合、縦・横・チャンネルの３次元のデータをそのまま学習し、次に伝える
・画像より小さいサイズのフィルタを用意し、画像とフィルタの値を掛け合わせたものの総和を取る
４－１－１　バイアス
・総和を取った後、ニューラルネットワークの学習同様にバイアスを加算する
４－１－２　パディング
・入力画像と出力画像の大きさを合わせるため、入力画像の周囲にデータを付け足す
　ゼロを付け足すゼロパディングが一般的（ゼロである必要性はない）
４－１－３　ストライド
・畳み込み演算の際にフィルタを移動させる刻み
４－１－４　チャンネル
・画像の奥行き
・チャンネル数分だけフィルタが用意される
・全結合で画像を学習した場合の課題
　画像（縦、横、チャンネルの３次元）が１次元のデータとして処理される
　→ＲＧＢの各チャンネルの関連性が学習に反映されない
　→畳み込み層の登場
・深層学習２－１７～実装演習

simple convolution network

image to column

In [45]:

```python
import sys, os
sys.path.append(os.pardir)
import pickle
import numpy as np
from collections import OrderedDict
from common import layers
from common import optimizer
from data.mnist import load_mnist
import matplotlib.pyplot as plt

# 画像データを２次元配列に変換
'''
input_data: 入力値
filter_h: フィルターの高さ
filter_w: フィルターの横幅
stride: ストライド
pad: パディング
'''
def im2col(input_data, filter_h, filter_w, stride=1, pad=0):
    # N: number, C: channel, H: height, W: width
    N, C, H, W = input_data.shape
    out_h = (H + 2 * pad - filter_h)//stride + 1
    out_w = (W + 2 * pad - filter_w)//stride + 1

    img = np.pad(input_data, [(0,0), (0,0), (pad, pad), (pad, pad)], 'constant')
    col = np.zeros((N, C, filter_h, filter_w, out_h, out_w))

    for y in range(filter_h):
        y_max = y + stride * out_h
        for x in range(filter_w):
            x_max = x + stride * out_w
            col[:, :, y, x, :, :] = img[:, :, y:y_max:stride, x:x_max:stride]

    col = col.transpose(0, 4, 5, 1, 2, 3) # (N, C, filter_h, filter_w, out_h, out_w) -> (N, filter_w, out_h, out_w, C, filter_h)

    col = col.reshape(N * out_h * out_w, -1)
    return col
```

[try] im2colの処理を確認しよう

・関数内でtransposeの処理をしている行をコメントアウトして下のコードを実行してみよう
・input_dataの各次元のサイズやフィルターサイズ・ストライド・パディングを変えてみよう

In [53]:

```python
# im2colの処理確認
input_data = np.random.rand(3, 2, 4, 4)*100//1 # number, channel, height, widthを表す
print('========== input_data ==========\n', input_data)
print('================================')
filter_h = 3
filter_w = 3
stride = 1
pad = 0
col = im2col(input_data, filter_h=filter_h, filter_w=filter_w, stride=stride, pad=pad)
print('============= col ==============\n', col)
print('================================')
```

```
========== input_data ===========
 [[[[23. 88. 10. 61.]
   [70. 86. 54. 46.]
   [ 1. 66. 34. 78.]
   [70. 66. 81. 39.]]

  [[23. 51. 22. 38.]
   [26. 93. 25. 91.]
   [23. 31. 54. 82.]
   [72. 44. 38. 71.]]]


 [[[43. 73. 52. 21.]
   [50. 55.  0. 48.]
   [ 7. 99. 84. 91.]
   [57. 12. 39. 94.]]

  [[27. 27. 22. 77.]
   [17. 71. 37. 74.]
   [47. 72. 43. 28.]
   [34. 84. 23. 60.]]]


 [[[93. 25. 91. 41.]
   [41. 64. 67.  5.]
   [69.  7. 91. 92.]
   [94. 86. 50.  9.]]

  [[93. 15. 77. 54.]
   [ 1. 21. 44. 96.]
   [97. 58. 97. 63.]
   [80. 13. 89. 43.]]]]
================================
```

```
============= col ==============
[[23. 88. 70. 86. 88. 10. 86. 54. 10. 61. 54. 46. 70. 86.  1. 66. 86. 54.]
[66. 34. 54. 46. 34. 78.  1. 66. 70. 66. 66. 34. 66. 81. 34. 78. 81. 39.]
[23. 51. 26. 93. 51. 22. 93. 25. 22. 38. 25. 91. 26. 93. 23. 31. 93. 25.]
[31. 54. 25. 91. 54. 82. 23. 31. 72. 44. 31. 54. 44. 38. 54. 82. 38. 71.]
[43. 73. 50. 55. 73. 52. 55.  0. 52. 21.  0. 48. 50. 55.  7. 99. 55.  0.]
[99. 84.  0. 48. 84. 91.  7. 99. 57. 12. 99. 84. 12. 39. 84. 91. 39. 94.]
[27. 27. 17. 71. 27. 22. 71. 37. 22. 77. 37. 74. 17. 71. 47. 72. 71. 37.]
[72. 43. 37. 74. 43. 28. 47. 72. 34. 84. 72. 43. 84. 23. 43. 28. 23. 60.]
[93. 25. 41. 64. 25. 91. 64. 67. 91. 41. 67.  5. 41. 64. 69.  7. 64. 67.]
[ 7. 91. 67.  5. 91. 92. 69.  7. 94. 86.  7. 91. 86. 50. 91. 92. 50.  9.]
[93. 15.  1. 21. 15. 77. 21. 44. 77. 54. 44. 96.  1. 21. 97. 58. 21. 44.]
[58. 97. 44. 96. 97. 63. 97. 58. 80. 13. 58. 97. 13. 89. 97. 63. 89. 43.]]
================================
```

column to image

In [54]:

```python
# 2次元配列を画像データに変換
def col2im(col, input_shape, filter_h, filter_w, stride=1, pad=0):
    # N: number, C: channel, H: height, W: width
    N, C, H, W = input_shape
    #切り捨て除算
    out_h = (H + 2 * pad - filter_h)//stride + 1
    out_w = (W + 2 * pad - filter_w)//stride + 1
    col = col.reshape(N, out_h, out_w, C, filter_h, filter_w).transpose(0, 3, 4, 5, 1, 2) # (N, filter_h, filter_w, out_h, out_w, C)

    img = np.zeros((N, C, H + 2 * pad + stride - 1, W + 2 * pad + stride - 1))
    for y in range(filter_h):
        y_max = y + stride * out_h
        for x in range(filter_w):
            x_max = x + stride * out_w
            img[:, :, y:y_max:stride, x:x_max:stride] += col[:, :, y, x, :, :]

    return img[:, :, pad:H + pad, pad:W + pad]
```

col2imの処理を確認しよう

・im2colの確認で出力したcolをimageに変換して確認しよう

In [55]:

```python
img = col2im(col, input_shape=input_data.shape, filter_h=filter_h, filter_w=filter_w, stride=stride, pad=pad)
print(img)
```

```
[[[[ 23. 154. 104.  54.]
   [109. 216. 124. 103.]
   [179. 197. 152. 152.]
   [ 93.  48.  53.  72.]]

  [[ 61. 120. 112.  34.]
   [108. 221. 204.  88.]
   [ 92. 301. 196.  93.]
   [ 31. 175.  63.  71.]]]
```

```
[[[ 43. 172. 134.   0.]
  [ 82. 220. 196. 128.]
  [126. 108. 216.  85.]
  [ 71.  84.  94.  34.]]

 [[ 21.  12. 147.  84.]
  [127. 188. 192. 127.]
  [116. 301. 109. 137.]
  [ 72.  99.  60.  60.]]]


[[[ 93.  32. 132.  67.]
  [157. 103. 280. 136.]
  [ 85. 247. 272. 157.]
  [ 21. 141. 135.  80.]]

 [[ 41. 153.  12.  91.]
  [ 95. 207. 273. 188.]
  [  8. 190. 303. 106.]
  [ 58.  84. 133.  43.]]]]
```

４－２　プーリング層
・ＭＡＸプーリング、ＡＶＧプーリングの２種類がある
・対象領域のＭＡＸ値または平均値を取得
・深層学習２－１９～実装演習
　実行速度の改善はＧＰＵで実行／ＣＰＵをグレードアップなど検討すること

convolution class

In [10]:

```python
class Convolution:
    #W: フィルター, b: バイアス
    def __init__(self, W, b, stride=1, pad=0):
        self.W = W
        self.b = b
        self.stride = stride
        self.pad = pad

        #中間データ（backward時に使用）
        self.x = None
        self.col = None
        self.col_W = None

        #フィルター・バイアスパラメータの勾配
        self.dW = None
        self.db = None

    def forward(self, x):
        # FN: filter_number, C: channel, FH: filter_height, FW: filter_width
        FN, C, FH, FW = self.W.shape
        N, C, H, W = x.shape
        #出力値のheight, width
        out_h = 1 + int((H + 2 * self.pad - FH) / self.stride)
        out_w = 1 + int((W + 2 * self.pad - FW) / self.stride)

        #xを行列に変換
        col = im2col(x, FH, FW, self.stride, self.pad)
        #フィルターをxに合わせた行列に変換
        col_W = self.W.reshape(FN, -1).T

        out = np.dot(col, col_W) + self.b
        #計算のために変えた形式を戻す
        out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)

        self.x = x
        self.col = col
        self.col_W = col_W

        return out

    def backward(self, dout):
        FN, C, FH, FW = self.W.shape
        dout = dout.transpose(0, 2, 3, 1).reshape(-1, FN)

        self.db = np.sum(dout, axis=0)
        self.dW = np.dot(self.col.T, dout)
        self.dW = self.dW.transpose(1, 0).reshape(FN, C, FH, FW)

        dcol = np.dot(dout, self.col_W.T)
        #dcolを画像データに変換
        dx = col2im(dcol, self.x.shape, FH, FW, self.stride, self.pad)

        return dx
```

pooling class

In [11]:

```
class Pooling:
    def __init__(self, pool_h, pool_w, stride=1, pad=0):
        self.pool_h = pool_h
        self.pool_w = pool_w
        self.stride = stride
        self.pad = pad

        self.x = None
        self.arg_max = None

    def forward(self, x):
        N, C, H, W = x.shape
        out_h = int(1 + (H - self.pool_h) / self.stride)
        out_w = int(1 + (W - self.pool_w) / self.stride)

        # xを行列に変換
        col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
        # プーリングのサイズに合わせてリサイズ
        col = col.reshape(-1, self.pool_h*self.pool_w)

        # 行ごとに最大値を求める
        arg_max = np.argmax(col, axis=1)
        out = np.max(col, axis=1)
        # 整形
        out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)

        self.x = x
        self.arg_max = arg_max

        return out

    def backward(self, dout):
        dout = dout.transpose(0, 2, 3, 1)

        pool_size = self.pool_h * self.pool_w
        dmax = np.zeros((dout.size, pool_size))
        dmax[np.arange(self.arg_max.size), self.arg_max.flatten()] = dout.flatten()
        dmax = dmax.reshape(dout.shape + (pool_size,))

        dcol = dmax.reshape(dmax.shape[0] * dmax.shape[1] * dmax.shape[2], -1)
        dx = col2im(dcol, self.x.shape, self.pool_h, self.pool_w, self.stride, self.pad)

        return dx

```

sinple convolution network class

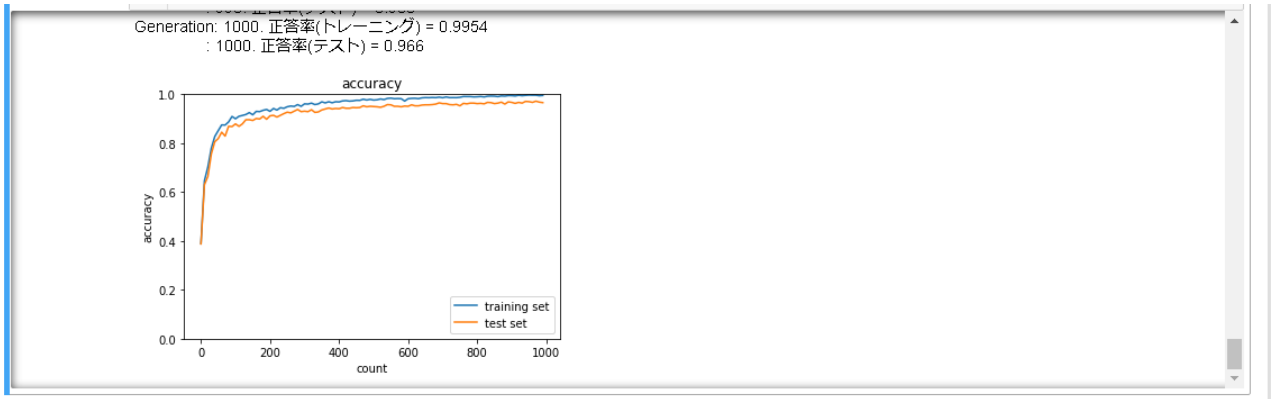
In [12]:

```
class SimpleConvNet:
    # conv - relu - pool - affine - relu - affine - softmax
    def __init__(self, input_dim=(1, 28, 28), conv_param={'filter_num':30, 'filter_size':5, 'pad':0, 'stride':1},
                 hidden_size=100, output_size=10, weight_init_std=0.01):
        filter_num = conv_param['filter_num']
        filter_size = conv_param['filter_size']
        filter_pad = conv_param['pad']
        filter_stride = conv_param['stride']
        input_size = input_dim[1]
        conv_output_size = (input_size - filter_size + 2 * filter_pad) / filter_stride + 1
        pool_output_size = int(filter_num * (conv_output_size / 2) * (conv_output_size / 2))

        # 重みの初期化
        self.params = {}
        self.params['W1'] = weight_init_std * np.random.randn(filter_num, input_dim[0], filter_size, filter_size)
        self.params['b1'] = np.zeros(filter_num)
        self.params['W2'] = weight_init_std * np.random.randn(pool_output_size, hidden_size)
        self.params['b2'] = np.zeros(hidden_size)
        self.params['W3'] = weight_init_std * np.random.randn(hidden_size, output_size)
        self.params['b3'] = np.zeros(output_size)

        # レイヤの生成
        self.layers = OrderedDict()
        self.layers['Conv1'] = layers.Convolution(self.params['W1'], self.params['b1'], conv_param['stride'], conv_param['pad'])
        self.layers['Relu1'] = layers.Relu()
        self.layers['Pool1'] = layers.Pooling(pool_h=2, pool_w=2, stride=2)
        self.layers['Affine1'] = layers.Affine(self.params['W2'], self.params['b2'])
        self.layers['Relu2'] = layers.Relu()
        self.layers['Affine2'] = layers.Affine(self.params['W3'], self.params['b3'])

        self.last_layer = layers.SoftmaxWithLoss()

    def predict(self, x):
        for key in self.layers.keys():
            x = self.layers[key].forward(x)
        return x

    def loss(self, x, d):
        y = self.predict(x)
        return self.last_layer.forward(y, d)

```

```
    def accuracy(self, x, d, batch_size=100):
        if d.ndim != 1 : d = np.argmax(d, axis=1)

        acc = 0.0

        for i in range(int(x.shape[0] / batch_size)):
            tx = x[i*batch_size:(i+1)*batch_size]
            td = d[i*batch_size:(i+1)*batch_size]
            y = self.predict(tx)
            y = np.argmax(y, axis=1)
            acc += np.sum(y == td)

        return acc / x.shape[0]

    def gradient(self, x, d):
        # forward
        self.loss(x, d)

        # backward
        dout = 1
        dout = self.last_layer.backward(dout)
        layers = list(self.layers.values())

        layers.reverse()
        for layer in layers:
            dout = layer.backward(dout)

        # 設定
        grad = {}
        grad['W1'], grad['b1'] = self.layers['Conv1'].dW, self.layers['Conv1'].db
        grad['W2'], grad['b2'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
        grad['W3'], grad['b3'] = self.layers['Affine2'].dW, self.layers['Affine2'].db

        return grad
```

In [13]:

```
from common import optimizer

# データの読み込み
(x_train, d_train), (x_test, d_test) = load_mnist(flatten=False)

print("データ読み込み完了")

# 処理に時間のかかる場合はデータを削減
x_train, d_train = x_train[:5000], d_train[:5000]
x_test, d_test = x_test[:1000], d_test[:1000]


network = SimpleConvNet(input_dim=(1,28,28), conv_param = {'filter_num': 30, 'filter_size': 5, 'pad': 0, 'stride': 1},
                        hidden_size=100, output_size=10, weight_init_std=0.01)

optimizer = optimizer.Adam()

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

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)
    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)

        print('Generation: ' + str(i+1) + '. 正答率(トレーニング) = ' + str(accr_train))
        print('                : ' + str(i+1) + '. 正答率(テスト) = ' + 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)
#グラフの表示
plt.show()
```



Ｓｅｃｔｉｏｎ５）最新のＣＮＮ

５－１　ＡｌｅｘＮｅｔ

・２０１２年開催の画像認識コンペティションで２位に大差をつけて優勝しディープラーニングが大きく注目を集めるきっかけととなったモデル
・５層の畳み込み層・プーリング層およびそれに続く３層の全結合層から構成される
・過学習を抑制する施策としてドロップアウトを使用している