TensolFlow

base

constant

Tensorという形式でデータを保持する。

In [1]:

```
import tensorflow as tf
import numpy as np

# それぞれ定数を定義
a = tf.constant(1)
b = tf.constant(2, dtype=tf.float32, shape=[3,2])
c = tf.constant(np.arange(4), dtype=tf.float32, shape=[2,2])

print('a:', a)
print('b:', b)
print('c:', c)
```

```
C:¥ProgramData¥Anaconda3¥lib¥site-packages¥h5py¥__init__.py:36: FutureWarning: Con
version of the second argument of issubdtype from `float` to `np.floating` is depr
ecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

```
a: Tensor("Const:0", shape=(), dtype=int32)
b: Tensor("Const_1:0", shape=(3, 2), dtype=float32)
c: Tensor("Const_2:0", shape=(2, 2), dtype=float32)
```

Session.run(a)すると、runしたノードaに関連するノードをすべて計算し、出力される。

In [2]:

```
sess = tf.Session()

print('a:', sess.run(a))
print('b:', sess.run(b))
print('c:', sess.run(c))

z=sess.run(a)
print(z*2)
```

```
a: 1
b: [[2. 2.]
 [2. 2.]
 [2. 2.]]
c: [[0. 1.]
 [2. 3.]]
2
```

placeholder

In [3]:

```
# プレースホルダーを定義
x = tf.placeholder(dtype=tf.float32, shape=[None,3])

print('x:', x)

sess = tf.Session()

X = np.random.rand(2,3)
print('X(numpy):', X)
```

```
x: Tensor("Placeholder:0", shape=(?, 3), dtype=float32)
X(numpy): [[0.83370985 0.4860043  0.98081105]
 [0.06851552 0.4632204  0.93428418]]
```

In [4]:

```
# プレースホルダにX[0]を入力
# shapeを(3,)から(1,3)にするためreshape
print('x:', sess.run(x, feed_dict={x:X[0].reshape(1,-1)}))
```

```
x: [[0.83370984 0.4860043  0.98081106]]
```

In [5]:

```
# プレースホルダにX[1]を入力
print('x:', sess.run(x, feed_dict={x:X[1].reshape(1,-1)}))
```

```
x: [[0.06851552 0.4632204  0.9342842 ]]
```

variables

In [6]:

```
# 定数を定義
a = tf.constant(10)
print('a:', a)
# 変数を定義
x = tf.Variable(1)
print('x:', x)

calc_op = x * a
print(calc_op)

# xの値を更新
update_x = tf.assign(x, calc_op)
print(update_x)
```

```
a: Tensor("Const_3:0", shape=(), dtype=int32)
WARNING:tensorflow:From C:¥ProgramData¥Anaconda3¥lib¥site-packages¥tensorflow¥pyth
on¥framework¥op_def_library.py:263: colocate_with (from tensorflow.python.framewor
k.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
x: <tf.Variable 'Variable:0' shape=() dtype=int32_ref>
Tensor("mul:0", shape=(), dtype=int32)
Tensor("Assign:0", shape=(), dtype=int32_ref)
```

In [7]:

```
sess = tf.Session()

# 変数の初期化
init = tf.global_variables_initializer()
sess.run(init)

print(sess.run(x))

sess.run(update_x)
print(sess.run(x))

sess.run(update_x)
print(sess.run(x))
```

```
1
10
100
```

線形回帰

In [32]:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

iters_num = 300
plot_interval = 10

# データを生成
n = 100
x = np.random.randn(n)#小数の乱数
d = 3 * x + 2

# ノイズを加える
noise = 0.3
d = d + noise * np.random.randn(n)

print(x)
```

```
[-1.54694948 -1.51921087  1.57037751  0.44242968  0.93837436 -1.12738744
 -1.30564086 -1.15317279  0.64632078  0.28184709  0.16976725 -0.04971216
  1.94835533 -0.06157433  0.21539213 -1.25684201  0.4468912   1.70219375
  0.21418637 -0.55039244 -1.18742887  1.16456899 -0.61855652  0.243373
  1.7951167  -1.18503724 -0.43109097  1.1094946   0.45238567  0.9242078
  0.12603835  1.26608628 -0.52972964  0.94286854  0.72474552 -0.01722734
  0.71382974  1.13459924 -0.70918891 -0.48249614 -1.26960039  0.13389292
  0.29250417 -0.11003267 -0.5317991   1.94502461  0.46864993  2.49999654
  1.19651059  0.69394048 -0.39563418  0.50650395  0.244284    1.91082568
  0.53858643 -1.32792827 -0.74594445  0.98002203  0.24095125 -1.07468647
  1.42321709  0.9552136   1.53745317  0.60445825  0.82919331 -1.40468116
  1.76418517  2.0902926   1.36217312 -0.67305506  0.92075485  1.55413754
 -1.05193381 -0.35954757 -0.84891241  0.91933351 -1.14127111  0.64555559
  0.05119954  1.64323538  0.04646646  1.15375947 -0.10931806 -0.82747958
 -2.10189543  1.0401296   0.05235331 -0.16250072 -1.20193001 -1.37029845
 -0.19845659  1.0793981  -1.7268737  -0.9387706  -1.147529   -1.42429152
 -0.96081795 -0.49691695 -0.76544498 -0.18550394]
```

In [33]:

```
# 入力値（プレースホルダー）
xt = tf.placeholder(tf.float32)
dt = tf.placeholder(tf.float32)

# 最適化の対象の変数を初期化
W = tf.Variable(tf.zeros([1]))
b = tf.Variable(tf.zeros([1]))

y = W * xt + b

print(xt)
print(dt)
```

```
Tensor("Placeholder_48:0", dtype=float32)
Tensor("Placeholder_49:0", dtype=float32)
```

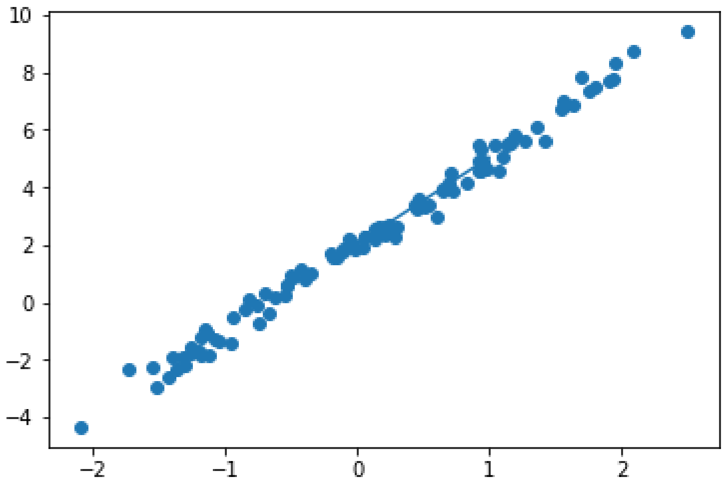
In [34]:

```
# 誤差関数 平均2乗誤差
loss = tf.reduce_mean(tf.square(y - dt))#reduce_meanは平均処理
optimizer = tf.train.GradientDescentOptimizer(0.1)#勾配降下法、学習率0.1、optimizer最適化アルゴ
リズム
train = optimizer.minimize(loss)#損失関数の最小化

# 初期化
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
```

In [35]:

```
# 作成したデータをトレーニングデータとして準備
x_train = x.reshape(-1,1)
d_train = d.reshape(-1,1)#訓練用ラベル

# トレーニング
for i in range(iters_num):#反復回数
    sess.run(train, feed_dict={xt:x_train,dt:d_train})#訓練実行
    if (i+1) % plot_interval == 0:
        loss_val = sess.run(loss, feed_dict={xt:x_train,dt:d_train})
        W_val = sess.run(W)
        b_val = sess.run(b)
        print('Generation: ' + str(i+1) + '. 誤差 = ' + str(loss_val))

print(W_val)
print(b_val)

#  予測関数
def predict(x):
    return W_val * x + b_val

fig = plt.figure()
subplot = fig.add_subplot(1, 1, 1)
plt.scatter(x, d)
linex = np.linspace(0, 1, 2)
liney = predict(linex)
subplot.plot(linex,liney)
plt.show()
```

```
Generation: 10. 誤差 = 0.16891193
Generation: 20. 誤差 = 0.10064526
Generation: 30. 誤差 = 0.10034347
Generation: 40. 誤差 = 0.10034214
Generation: 50. 誤差 = 0.100342125
Generation: 60. 誤差 = 0.10034214
Generation: 70. 誤差 = 0.10034214
Generation: 80. 誤差 = 0.10034214
Generation: 90. 誤差 = 0.10034214
Generation: 100. 誤差 = 0.10034214
Generation: 110. 誤差 = 0.10034214
Generation: 120. 誤差 = 0.10034214
Generation: 130. 誤差 = 0.10034214
Generation: 140. 誤差 = 0.10034214
Generation: 150. 誤差 = 0.10034214
Generation: 160. 誤差 = 0.10034214
Generation: 170. 誤差 = 0.10034214
Generation: 180. 誤差 = 0.10034214
Generation: 190. 誤差 = 0.10034214
Generation: 200. 誤差 = 0.10034214
Generation: 210. 誤差 = 0.10034214
Generation: 220. 誤差 = 0.10034214
Generation: 230. 誤差 = 0.10034214
Generation: 240. 誤差 = 0.10034214
Generation: 250. 誤差 = 0.10034214
Generation: 260. 誤差 = 0.10034214
Generation: 270. 誤差 = 0.10034214
Generation: 280. 誤差 = 0.10034214
Generation: 290. 誤差 = 0.10034214
Generation: 300. 誤差 = 0.10034214
[2.9889605]
[2.041801]
```



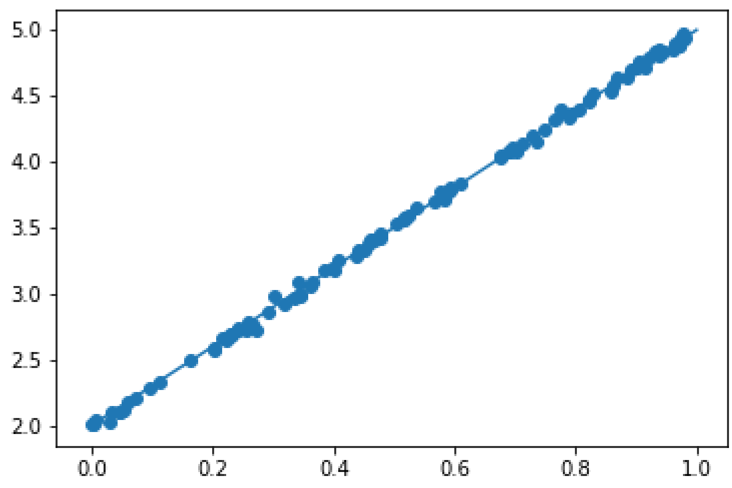
[try]

- noiseの値を変更しよう(noise 0.3を0.03に変更)

In [36]:

```
iters_num = 300
plot_interval = 10

# データを生成
n = 100
x = np.random.rand(n)
d = 3 * x + 2

# ノイズを加える
noise = 0.03
d = d + noise * np.random.randn(n)

# 入力値
xt = tf.placeholder(tf.float32)
dt = tf.placeholder(tf.float32)

# 最適化の対象の変数を初期化
W = tf.Variable(tf.zeros([1]))
b = tf.Variable(tf.zeros([1]))

y = W * xt + b

# 誤差関数 平均2乗誤差
loss = tf.reduce_mean(tf.square(y - dt))
optimizer = tf.train.GradientDescentOptimizer(0.1)
train = optimizer.minimize(loss)

# 初期化
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

# 作成したデータをトレーニングデータとして準備
x_train = x.reshape(-1,1)
d_train = d.reshape(-1,1)

# トレーニング
for i in range(iters_num):
    sess.run(train, feed_dict={xt:x_train,dt:d_train})
    if (i+1) % plot_interval == 0:
        loss_val = sess.run(loss, feed_dict={xt:x_train,dt:d_train})
        W_val = sess.run(W)
        b_val = sess.run(b)
        print('Generation: ' + str(i+1) + '. 誤差 = ' + str(loss_val))

print(W_val)
print(b_val)

#  予測関数
def predict(x):
    return W_val * x + b_val

fig = plt.figure()
subplot = fig.add_subplot(1, 1, 1)
plt.scatter(x, d)
linex = np.linspace(0, 1, 2)
liney = predict(linex)
subplot.plot(linex,liney)
plt.show()
```

```
Generation: 10. 誤差 = 0.17103556
Generation: 20. 誤差 = 0.10534405
Generation: 30. 誤差 = 0.07941533
Generation: 40. 誤差 = 0.05995369
Generation: 50. 誤差 = 0.045305546
Generation: 60. 誤差 = 0.034280248
Generation: 70. 誤差 = 0.025981836
Generation: 80. 誤差 = 0.019735824
Generation: 90. 誤差 = 0.01503463
Generation: 100. 誤差 = 0.011496145
Generation: 110. 誤差 = 0.008832831
Generation: 120. 誤差 = 0.0068282373
Generation: 130. 誤差 = 0.005319431
Generation: 140. 誤差 = 0.0041837976
Generation: 150. 誤差 = 0.003329029
Generation: 160. 誤差 = 0.0026856798
Generation: 170. 誤差 = 0.002201438
Generation: 180. 誤差 = 0.0018369699
Generation: 190. 誤差 = 0.0015626396
Generation: 200. 誤差 = 0.001356159
Generation: 210. 誤差 = 0.0012007478
Generation: 220. 誤差 = 0.0010837722
Generation: 230. 誤差 = 0.0009957276
Generation: 240. 誤差 = 0.0009294575
Generation: 250. 誤差 = 0.00087957934
Generation: 260. 誤差 = 0.0008420385
Generation: 270. 誤差 = 0.0008137801
Generation: 280. 誤差 = 0.000792513
Generation: 290. 誤差 = 0.00077650486
Generation: 300. 誤差 = 0.0007644552
[2.9944782]
[1.9983674]
```



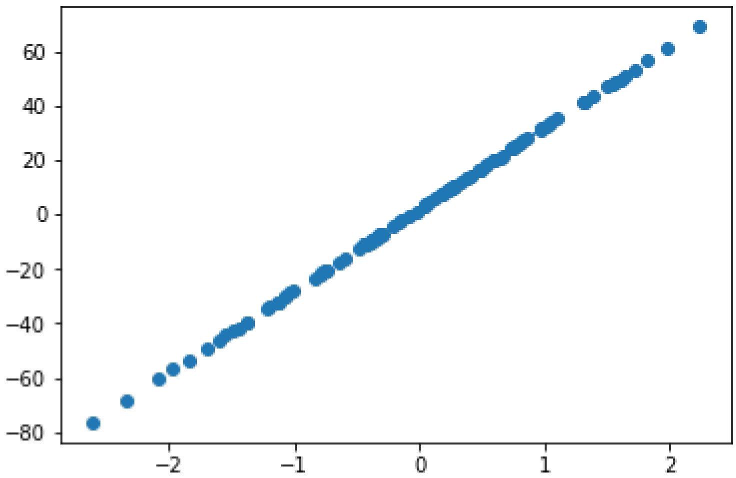
[try]

- dの数値を変更しよう　(d=3x+2をd=30x+2に変更)

In [13]:

```
iters_num = 300
plot_interval = 10

# データを生成
n = 100
x = np.random.randn(n)
d = 30 * x + 2

# ノイズを加える
noise = 0.3
d = d + noise * np.random.randn(n)

# 入力値
xt = tf.placeholder(tf.float32)
dt = tf.placeholder(tf.float32)

# 最適化の対象の変数を初期化
W = tf.Variable(tf.zeros([1]))
b = tf.Variable(tf.zeros([1]))

y = W * xt + b

# 誤差関数 平均2乗誤差
loss = tf.reduce_mean(tf.square(y - dt))
optimizer = tf.train.GradientDescentOptimizer(0.1)
train = optimizer.minimize(loss)

# 初期化
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

# 作成したデータをトレーニングデータとして準備
x_train = x.reshape(-1,1)
d_train = d.reshape(-1,1)

# トレーニング
for i in range(iters_num):
    sess.run(train, feed_dict={xt:x_train,dt:d_train})
    if (i+1) % plot_interval == 0:
        loss_val = sess.run(loss, feed_dict={xt:x_train,dt:d_train})
        W_val = sess.run(W)
        b_val = sess.run(b)
        print('Generation: ' + str(i+1) + '. 誤差 = ' + str(loss_val))

print(W_val)
print(b_val)

#  予測関数
def predict(x):
    return W_val * x + b_val

fig = plt.figure()
subplot = fig.add_subplot(1, 1, 1)
plt.scatter(x, d)
linex = np.linspace(0, 1, 2)
liney = predict(linex)
subplot.plot(linex,liney)
plt.show()
```

```
Generation: 10. 誤差 = 8.645359
Generation: 20. 誤差 = 0.1549129
Generation: 30. 誤差 = 0.0781575
Generation: 40. 誤差 = 0.07746361
Generation: 50. 誤差 = 0.077457316
Generation: 60. 誤差 = 0.077457145
Generation: 70. 誤差 = 0.07745706
Generation: 80. 誤差 = 0.07745706
Generation: 90. 誤差 = 0.07745706
Generation: 100. 誤差 = 0.07745706
Generation: 110. 誤差 = 0.07745706
Generation: 120. 誤差 = 0.07745706
Generation: 130. 誤差 = 0.07745706
Generation: 140. 誤差 = 0.07745706
Generation: 150. 誤差 = 0.07745706
Generation: 160. 誤差 = 0.07745706
Generation: 170. 誤差 = 0.07745706
Generation: 180. 誤差 = 0.07745706
Generation: 190. 誤差 = 0.07745706
Generation: 200. 誤差 = 0.07745706
Generation: 210. 誤差 = 0.07745706
Generation: 220. 誤差 = 0.07745706
Generation: 230. 誤差 = 0.07745706
Generation: 240. 誤差 = 0.07745706
Generation: 250. 誤差 = 0.07745706
Generation: 260. 誤差 = 0.07745706
Generation: 270. 誤差 = 0.07745706
Generation: 280. 誤差 = 0.07745706
Generation: 290. 誤差 = 0.07745706
Generation: 300. 誤差 = 0.07745706
[30.003994]
[1.9790144]
```



非線形回帰

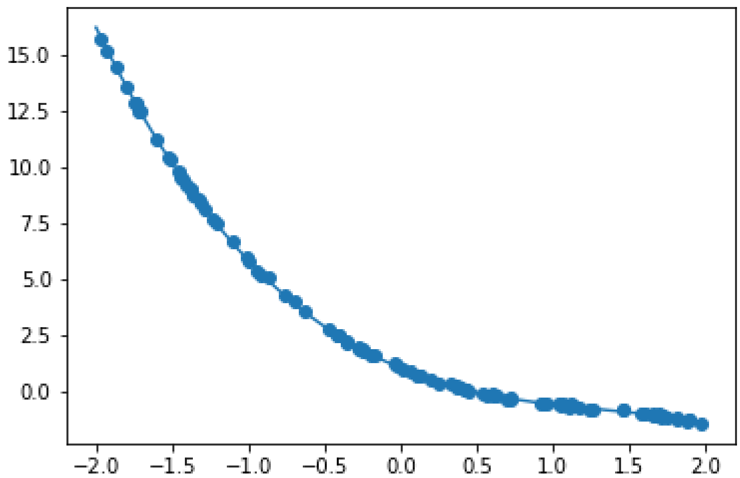
In [14]:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

iters_num = 10000
plot_interval = 100

# データを生成
n=100
x = np.random.rand(n).astype(np.float32) * 4 - 2
d =  - 0.4 * x ** 3 + 1.6 * x ** 2 - 2.8 * x + 1

#  ノイズを加える
noise = 0.05
d = d + noise * np.random.randn(n)

# モデル
# bを使っていないことに注意.
xt = tf.placeholder(tf.float32, [None, 4])
dt = tf.placeholder(tf.float32, [None, 1])
W = tf.Variable(tf.random_normal([4, 1], stddev=0.01))
y = tf.matmul(xt,W)

# 誤差関数 平均２乗誤差
loss = tf.reduce_mean(tf.square(y - dt))
optimizer = tf.train.AdamOptimizer(0.001)
train = optimizer.minimize(loss)

# 初期化
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

# 作成したデータをトレーニングデータとして準備
d_train = d.reshape(-1,1)
x_train = np.zeros([n, 4])
for i in range(n):
    for j in range(4):
        x_train[i, j] = x[i]**j

#  トレーニング
for i in range(iters_num):
    if (i+1) % plot_interval == 0:
        loss_val = sess.run(loss, feed_dict={xt:x_train, dt:d_train})
        W_val = sess.run(W)
        print('Generation: ' + str(i+1) + '. 誤差 = ' + str(loss_val))
    sess.run(train, feed_dict={xt:x_train,dt:d_train})

print(W_val[::-1])

# 予測関数
def predict(x):
    result = 0.
    for i in range(0,4):
        result += W_val[i,0] * x ** i
    return result

fig = plt.figure()
subplot = fig.add_subplot(1,1,1)
```

```
plt.scatter(x ,d)
linex = np.linspace(-2,2,100)
liney = predict(linex)
subplot.plot(linex,liney)
plt.show()
```

```
Generation: 100. 誤差 = 26.545837
Generation: 200. 誤差 = 22.251637
Generation: 300. 誤差 = 18.533943
Generation: 400. 誤差 = 15.338687
Generation: 500. 誤差 = 12.613826
Generation: 600. 誤差 = 10.309445
Generation: 700. 誤差 = 8.377865
Generation: 800. 誤差 = 6.7738466
Generation: 900. 誤差 = 5.4547877
Generation: 1000. 誤差 = 4.380972
Generation: 1100. 誤差 = 3.515791
Generation: 1200. 誤差 = 2.8259377
Generation: 1300. 誤差 = 2.2815566
Generation: 1400. 誤差 = 1.8563008
Generation: 1500. 誤差 = 1.527305
Generation: 1600. 誤差 = 1.2750434
Generation: 1700. 誤差 = 1.083127
Generation: 1800. 誤差 = 0.9380164
Generation: 1900. 誤差 = 0.8286844
Generation: 2000. 誤差 = 0.7463011
Generation: 2100. 誤差 = 0.6838835
Generation: 2200. 誤差 = 0.6359978
Generation: 2300. 誤差 = 0.59847856
Generation: 2400. 誤差 = 0.5681932
Generation: 2500. 誤差 = 0.54283065
Generation: 2600. 誤差 = 0.52072835
Generation: 2700. 誤差 = 0.5007215
Generation: 2800. 誤差 = 0.4820201
Generation: 2900. 誤差 = 0.46410888
Generation: 3000. 誤差 = 0.4466671
Generation: 3100. 誤差 = 0.42950803
Generation: 3200. 誤差 = 0.41253242
Generation: 3300. 誤差 = 0.3956974
Generation: 3400. 誤差 = 0.3789939
Generation: 3500. 誤差 = 0.3624318
Generation: 3600. 誤差 = 0.3460315
Generation: 3700. 誤差 = 0.32981813
Generation: 3800. 誤差 = 0.3138184
Generation: 3900. 誤差 = 0.29805923
Generation: 4000. 誤差 = 0.2825663
Generation: 4100. 誤差 = 0.26736426
Generation: 4200. 誤差 = 0.2524762
Generation: 4300. 誤差 = 0.2379235
Generation: 4400. 誤差 = 0.223726
Generation: 4500. 誤差 = 0.20990208
Generation: 4600. 誤差 = 0.19646865
Generation: 4700. 誤差 = 0.18344124
Generation: 4800. 誤差 = 0.17083448
Generation: 4900. 誤差 = 0.15866151
Generation: 5000. 誤差 = 0.14693469
Generation: 5100. 誤差 = 0.13566533
Generation: 5200. 誤差 = 0.12486374
Generation: 5300. 誤差 = 0.11453949
Generation: 5400. 誤差 = 0.104701005
Generation: 5500. 誤差 = 0.09535494
Generation: 5600. 誤差 = 0.08650741
Generation: 5700. 誤差 = 0.07816276
Generation: 5800. 誤差 = 0.07032368
Generation: 5900. 誤差 = 0.0629908
Generation: 6000. 誤差 = 0.05616277
Generation: 6100. 誤差 = 0.049835563
```

```
Generation: 6200. 誤差 = 0.044003095
Generation: 6300. 誤差 = 0.03865607
Generation: 6400. 誤差 = 0.033782884
Generation: 6500. 誤差 = 0.029369364
Generation: 6600. 誤差 = 0.025398478
Generation: 6700. 誤差 = 0.021850767
Generation: 6800. 誤差 = 0.018704606
Generation: 6900. 誤差 = 0.015936418
Generation: 7000. 誤差 = 0.013520882
Generation: 7100. 誤差 = 0.011431473
Generation: 7200. 誤差 = 0.009641048
Generation: 7300. 誤差 = 0.00812167
Generation: 7400. 誤差 = 0.006845699
Generation: 7500. 誤差 = 0.0057857935
Generation: 7600. 誤差 = 0.0049155597
Generation: 7700. 誤差 = 0.004209771
Generation: 7800. 誤差 = 0.0036447167
Generation: 7900. 誤差 = 0.0031985224
Generation: 8000. 誤差 = 0.002851249
Generation: 8100. 誤差 = 0.0025850318
Generation: 8200. 誤差 = 0.0023842803
Generation: 8300. 誤差 = 0.0022354496
Generation: 8400. 誤差 = 0.0021270707
Generation: 8500. 誤差 = 0.002049628
Generation: 8600. 誤差 = 0.0019954022
Generation: 8700. 誤差 = 0.0019582256
Generation: 8800. 誤差 = 0.0019332838
Generation: 8900. 誤差 = 0.0019169616
Generation: 9000. 誤差 = 0.0019065297
Generation: 9100. 誤差 = 0.0019000337
Generation: 9200. 誤差 = 0.001896099
Generation: 9300. 誤差 = 0.0018937829
Generation: 9400. 誤差 = 0.0018924671
Generation: 9500. 誤差 = 0.001891742
Generation: 9600. 誤差 = 0.001891348
Generation: 9700. 誤差 = 0.0018911499
Generation: 9800. 誤差 = 0.001891051
Generation: 9900. 誤差 = 0.0018910011
Generation: 10000. 誤差 = 0.0018909833
[[-0.3999309]
 [ 1.5972086]
 [-2.80164  ]
 [ 1.0043371]]
```



[try]

- noiseの値を変更しよう(0.05を1に変更)

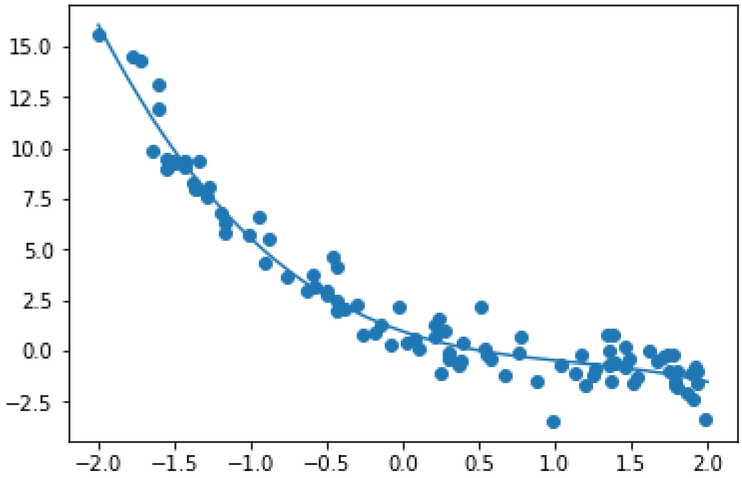
In [15]:

```
iters_num = 10000
plot_interval = 100

    # データを生成
n=100
x = np.random.rand(n).astype(np.float32) * 4 - 2
d =  - 0.4 * x ** 3 + 1.6 * x ** 2 - 2.8 * x + 1

#  ノイズを加える
noise = 1
d = d + noise * np.random.randn(n)

# モデル
# bを使っていないことに注意.
xt = tf.placeholder(tf.float32, [None, 4])
dt = tf.placeholder(tf.float32, [None, 1])
W = tf.Variable(tf.random_normal([4, 1], stddev=0.01))
y = tf.matmul(xt,W)

# 誤差関数 平均２乗誤差
loss = tf.reduce_mean(tf.square(y - dt))
optimizer = tf.train.AdamOptimizer(0.001)
train = optimizer.minimize(loss)

# 初期化
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

# 作成したデータをトレーニングデータとして準備
d_train = d.reshape(-1,1)
x_train = np.zeros([n, 4])
for i in range(n):
    for j in range(4):
        x_train[i, j] = x[i]**j

#  トレーニング
for i in range(iters_num):
    if (i+1) % plot_interval == 0:
        loss_val = sess.run(loss, feed_dict={xt:x_train, dt:d_train})
        W_val = sess.run(W)
        print('Generation: ' + str(i+1) + '. 誤差 = ' + str(loss_val))
    sess.run(train, feed_dict={xt:x_train,dt:d_train})

print(W_val[::-1])

# 予測関数
def predict(x):
    result = 0.
    for i in range(0,4):
        result += W_val[i,0] * x ** i
    return result

fig = plt.figure()
subplot = fig.add_subplot(1,1,1)
plt.scatter(x ,d)
linex = np.linspace(-2,2,100)
liney = predict(linex)
```

```
subplot.plot(linex,liney)
plt.show()
```

```
Generation: 100. 誤差 = 21.640308
Generation: 200. 誤差 = 18.205376
Generation: 300. 誤差 = 15.24938
Generation: 400. 誤差 = 12.723288
Generation: 500. 誤差 = 10.580398
Generation: 600. 誤差 = 8.776509
Generation: 700. 誤差 = 7.270082
Generation: 800. 誤差 = 6.0224476
Generation: 900. 誤差 = 4.997961
Generation: 1000. 誤差 = 4.1641345
Generation: 1100. 誤差 = 3.4916868
Generation: 1200. 誤差 = 2.9545393
Generation: 1300. 誤差 = 2.5297062
Generation: 1400. 誤差 = 2.1971385
Generation: 1500. 誤差 = 1.939532
Generation: 1600. 誤差 = 1.7420802
Generation: 1700. 誤差 = 1.5922426
Generation: 1800. 誤差 = 1.4795123
Generation: 1900. 誤差 = 1.3951956
Generation: 2000. 誤差 = 1.3322101
Generation: 2100. 誤差 = 1.2848932
Generation: 2200. 誤差 = 1.2488123
Generation: 2300. 誤差 = 1.2205856
Generation: 2400. 誤差 = 1.1976997
Generation: 2500. 誤差 = 1.1783452
Generation: 2600. 誤差 = 1.1612641
Generation: 2700. 誤差 = 1.1456128
Generation: 2800. 誤差 = 1.1308494
Generation: 2900. 誤差 = 1.1166435
Generation: 3000. 誤差 = 1.1028062
Generation: 3100. 誤差 = 1.0892389
Generation: 3200. 誤差 = 1.0758988
Generation: 3300. 誤差 = 1.0627748
Generation: 3400. 誤差 = 1.0498741
Generation: 3500. 誤差 = 1.0372111
Generation: 3600. 誤差 = 1.0248044
Generation: 3700. 誤差 = 1.0126725
Generation: 3800. 誤差 = 1.0008332
Generation: 3900. 誤差 = 0.9893027
Generation: 4000. 誤差 = 0.97809494
Generation: 4100. 誤差 = 0.9672225
Generation: 4200. 誤差 = 0.9566959
Generation: 4300. 誤差 = 0.946524
Generation: 4400. 誤差 = 0.936714
Generation: 4500. 誤差 = 0.9272717
Generation: 4600. 誤差 = 0.9182016
Generation: 4700. 誤差 = 0.9095078
Generation: 4800. 誤差 = 0.9011929
Generation: 4900. 誤差 = 0.89325935
Generation: 5000. 誤差 = 0.8857092
Generation: 5100. 誤差 = 0.87854266
Generation: 5200. 誤差 = 0.8717607
Generation: 5300. 誤差 = 0.8653635
Generation: 5400. 誤差 = 0.85935014
Generation: 5500. 誤差 = 0.85371894
Generation: 5600. 誤差 = 0.8484676
Generation: 5700. 誤差 = 0.84359246
Generation: 5800. 誤差 = 0.8390884
Generation: 5900. 誤差 = 0.83494896
Generation: 6000. 誤差 = 0.8311658
Generation: 6100. 誤差 = 0.8277293
```

```
Generation: 6200. 誤差 = 0.8246274
Generation: 6300. 誤差 = 0.8218471
Generation: 6400. 誤差 = 0.8193729
Generation: 6500. 誤差 = 0.8171881
Generation: 6600. 誤差 = 0.81527495
Generation: 6700. 誤差 = 0.8136139
Generation: 6800. 誤差 = 0.81218505
Generation: 6900. 誤差 = 0.8109681
Generation: 7000. 誤差 = 0.809942
Generation: 7100. 誤差 = 0.8090862
Generation: 7200. 誤差 = 0.8083809
Generation: 7300. 誤差 = 0.8078063
Generation: 7400. 誤差 = 0.8073444
Generation: 7500. 誤差 = 0.806978
Generation: 7600. 誤差 = 0.8066915
Generation: 7700. 誤差 = 0.80647093
Generation: 7800. 誤差 = 0.8063038
Generation: 7900. 誤差 = 0.80617905
Generation: 8000. 誤差 = 0.80608785
Generation: 8100. 誤差 = 0.80602235
Generation: 8200. 誤差 = 0.8059763
Generation: 8300. 誤差 = 0.8059442
Generation: 8400. 誤差 = 0.80592275
Generation: 8500. 誤差 = 0.8059087
Generation: 8600. 誤差 = 0.8058995
Generation: 8700. 誤差 = 0.8058937
Generation: 8800. 誤差 = 0.8058903
Generation: 8900. 誤差 = 0.80588835
Generation: 9000. 誤差 = 0.805887
Generation: 9100. 誤差 = 0.8058862
Generation: 9200. 誤差 = 0.80588603
Generation: 9300. 誤差 = 0.8058857
Generation: 9400. 誤差 = 0.8058856
Generation: 9500. 誤差 = 0.8058856
Generation: 9600. 誤差 = 0.80588555
Generation: 9700. 誤差 = 0.8058856
Generation: 9800. 誤差 = 0.8058855
Generation: 9900. 誤差 = 0.80588555
Generation: 10000. 誤差 = 0.8058856
[[-0.4597927]
 [ 1.578666 ]
 [-2.5477927]
 [ 0.9510553]]
```



[try]

- dの数値を変更しよう

(変更前）d = - 0.4 *x ** 3 + 1.6* x **2 - 2.8 *x + 1 (変更後）d = 0.4* x** 3 + 1.6 *x ** 2 - 2.8* x + 1

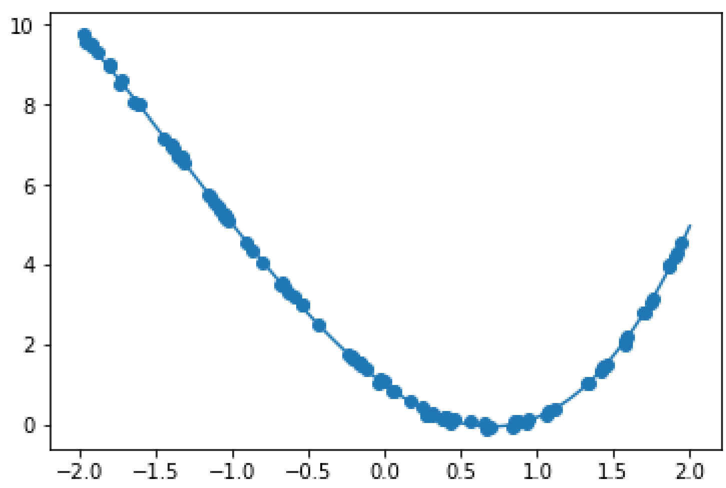
In [16]:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

iters_num = 10000
plot_interval = 100

# データを生成
n=100
x = np.random.rand(n).astype(np.float32) * 4 - 2
d =  0.4 * x ** 3 + 1.6 * x ** 2 - 2.8 * x + 1

#  ノイズを加える
noise = 0.05
d = d + noise * np.random.randn(n)

# モデル
# bを使っていないことに注意.
xt = tf.placeholder(tf.float32, [None, 4])
dt = tf.placeholder(tf.float32, [None, 1])
W = tf.Variable(tf.random_normal([4, 1], stddev=0.01))
y = tf.matmul(xt,W)

# 誤差関数 平均２乗誤差
loss = tf.reduce_mean(tf.square(y - dt))
optimizer = tf.train.AdamOptimizer(0.001)
train = optimizer.minimize(loss)

# 初期化
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

# 作成したデータをトレーニングデータとして準備
d_train = d.reshape(-1,1)
x_train = np.zeros([n, 4])
for i in range(n):
    for j in range(4):
        x_train[i, j] = x[i]**j

#  トレーニング
for i in range(iters_num):
    if (i+1) % plot_interval == 0:
        loss_val = sess.run(loss, feed_dict={xt:x_train, dt:d_train})
        W_val = sess.run(W)
        print('Generation: ' + str(i+1) + '. 誤差 = ' + str(loss_val))
    sess.run(train, feed_dict={xt:x_train,dt:d_train})

print(W_val[::-1])

# 予測関数
def predict(x):
    result = 0.
    for i in range(0,4):
        result += W_val[i,0] * x ** i
    return result

fig = plt.figure()
subplot = fig.add_subplot(1,1,1)
```

```
plt.scatter(x ,d)
linex = np.linspace(-2,2,100)
liney = predict(linex)
subplot.plot(linex,liney)
plt.show()
```

```
Generation: 100. 誤差 = 16.858486
Generation: 200. 誤差 = 13.833863
Generation: 300. 誤差 = 11.420864
Generation: 400. 誤差 = 9.5036955
Generation: 500. 誤差 = 7.976492
Generation: 600. 誤差 = 6.7478223
Generation: 700. 誤差 = 5.7439466
Generation: 800. 誤差 = 4.9096203
Generation: 900. 誤差 = 4.206179
Generation: 1000. 誤差 = 3.6077561
Generation: 1100. 誤差 = 3.0970533
Generation: 1200. 誤差 = 2.6618237
Generation: 1300. 誤差 = 2.292502
Generation: 1400. 誤差 = 1.9808923
Generation: 1500. 誤差 = 1.719563
Generation: 1600. 誤差 = 1.5016239
Generation: 1700. 誤差 = 1.3206865
Generation: 1800. 誤差 = 1.1708791
Generation: 1900. 誤差 = 1.0468887
Generation: 2000. 誤差 = 0.94398165
Generation: 2100. 誤差 = 0.8580217
Generation: 2200. 誤差 = 0.78546834
Generation: 2300. 誤差 = 0.7233559
Generation: 2400. 誤差 = 0.6692601
Generation: 2500. 誤差 = 0.6212512
Generation: 2600. 誤差 = 0.57783854
Generation: 2700. 誤差 = 0.537905
Generation: 2800. 誤差 = 0.50064397
Generation: 2900. 誤差 = 0.4654965
Generation: 3000. 誤差 = 0.4320932
Generation: 3100. 誤差 = 0.40020633
Generation: 3200. 誤差 = 0.36970666
Generation: 3300. 誤差 = 0.340531
Generation: 3400. 誤差 = 0.31265804
Generation: 3500. 誤差 = 0.28609
Generation: 3600. 誤差 = 0.26083988
Generation: 3700. 誤差 = 0.23692356
Generation: 3800. 誤差 = 0.21435456
Generation: 3900. 誤差 = 0.19314188
Generation: 4000. 誤差 = 0.17328703
Generation: 4100. 誤差 = 0.15478437
Generation: 4200. 誤差 = 0.13762054
Generation: 4300. 誤差 = 0.12177444
Generation: 4400. 誤差 = 0.10721764
Generation: 4500. 誤差 = 0.09391472
Generation: 4600. 誤差 = 0.08182403
Generation: 4700. 誤差 = 0.07089843
Generation: 4800. 誤差 = 0.061084695
Generation: 4900. 誤差 = 0.05232543
Generation: 5000. 誤差 = 0.04455975
Generation: 5100. 誤差 = 0.037723497
Generation: 5200. 誤差 = 0.0317501
Generation: 5300. 誤差 = 0.026572032
Generation: 5400. 誤差 = 0.022120733
Generation: 5500. 誤差 = 0.018328058
Generation: 5600. 誤差 = 0.015126827
Generation: 5700. 誤差 = 0.012451649
Generation: 5800. 誤差 = 0.010239606
Generation: 5900. 誤差 = 0.008431014
Generation: 6000. 誤差 = 0.0069697965
Generation: 6100. 誤差 = 0.0058040507
```

```
Generation: 6200. 誤差 = 0.0048864726
Generation: 6300. 誤差 = 0.004174379
Generation: 6400. 誤差 = 0.003630045
Generation: 6500. 誤差 = 0.0032205302
Generation: 6600. 誤差 = 0.0029176122
Generation: 6700. 誤差 = 0.0026974913
Generation: 6800. 誤差 = 0.0025405593
Generation: 6900. 誤差 = 0.0024308737
Generation: 7000. 誤差 = 0.0023558019
Generation: 7100. 誤差 = 0.0023055633
Generation: 7200. 誤差 = 0.0022727223
Generation: 7300. 誤差 = 0.0022517713
Generation: 7400. 誤差 = 0.0022387647
Generation: 7500. 誤差 = 0.0022308992
Generation: 7600. 誤差 = 0.0022262812
Generation: 7700. 誤差 = 0.0022236563
Generation: 7800. 誤差 = 0.002222209
Generation: 7900. 誤差 = 0.0022214376
Generation: 8000. 誤差 = 0.002221041
Generation: 8100. 誤差 = 0.002220846
Generation: 8200. 誤差 = 0.002220753
Generation: 8300. 誤差 = 0.002220709
Generation: 8400. 誤差 = 0.002220692
Generation: 8500. 誤差 = 0.0022206828
Generation: 8600. 誤差 = 0.0022206793
Generation: 8700. 誤差 = 0.0022206819
Generation: 8800. 誤差 = 0.0022206807
Generation: 8900. 誤差 = 0.0022206793
Generation: 9000. 誤差 = 0.0022206774
Generation: 9100. 誤差 = 0.00222068
Generation: 9200. 誤差 = 0.002220675
Generation: 9300. 誤差 = 0.0022206767
Generation: 9400. 誤差 = 0.0022206784
Generation: 9500. 誤差 = 0.0022206828
Generation: 9600. 誤差 = 0.0022206819
Generation: 9700. 誤差 = 0.0022206805
Generation: 9800. 誤差 = 0.002220678
Generation: 9900. 誤差 = 0.002220681
Generation: 10000. 誤差 = 0.0022206795
[[ 0.4005766]
 [ 1.5947251]
 [-2.8117151]
 [ 1.0070072]]
```



[try]

- 次の式をモデルとして回帰を行おう

$$y = 30x^2 + 0.5x + 0.2$$

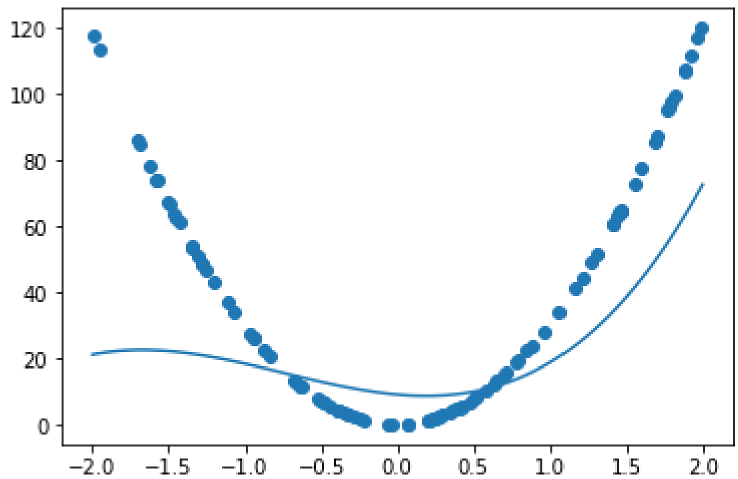
In [17]:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

iters_num = 10000
plot_interval = 100

# データを生成
n=100
x = np.random.rand(n).astype(np.float32) * 4 - 2
d =  30 * x ** 2 + 0.5 * x + 0.2

#  ノイズを加える
noise = 0.05
d = d + noise * np.random.randn(n)

# モデル
# bを使っていないことに注意.
xt = tf.placeholder(tf.float32, [None, 4])
dt = tf.placeholder(tf.float32, [None, 1])
W = tf.Variable(tf.random_normal([4, 1], stddev=0.01))
y = tf.matmul(xt,W)

# 誤差関数 平均２乗誤差
loss = tf.reduce_mean(tf.square(y - dt))
optimizer = tf.train.AdamOptimizer(0.001)
train = optimizer.minimize(loss)

# 初期化
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

# 作成したデータをトレーニングデータとして準備
d_train = d.reshape(-1,1)
x_train = np.zeros([n, 4])
for i in range(n):
    for j in range(4):
        x_train[i, j] = x[i]**j

#  トレーニング
for i in range(iters_num):
    if (i+1) % plot_interval == 0:
        loss_val = sess.run(loss, feed_dict={xt:x_train, dt:d_train})
        W_val = sess.run(W)
        print('Generation: ' + str(i+1) + '. 誤差 = ' + str(loss_val))
    sess.run(train, feed_dict={xt:x_train,dt:d_train})

print(W_val[::-1])

# 予測関数
def predict(x):
    result = 0.
    for i in range(0,4):
        result += W_val[i,0] * x ** i
    return result

fig = plt.figure()
subplot = fig.add_subplot(1,1,1)
```

```
plt.scatter(x ,d)
linex = np.linspace(-2,2,100)
liney = predict(linex)
subplot.plot(linex,liney)
plt.show()
```

```
Generation: 100. 誤差 = 2658.9
Generation: 200. 誤差 = 2624.853
Generation: 300. 誤差 = 2591.5386
Generation: 400. 誤差 = 2558.9316
Generation: 500. 誤差 = 2527.011
Generation: 600. 誤差 = 2495.7532
Generation: 700. 誤差 = 2465.1367
Generation: 800. 誤差 = 2435.1409
Generation: 900. 誤差 = 2405.7466
Generation: 1000. 誤差 = 2376.9329
Generation: 1100. 誤差 = 2348.682
Generation: 1200. 誤差 = 2320.975
Generation: 1300. 誤差 = 2293.794
Generation: 1400. 誤差 = 2267.1218
Generation: 1500. 誤差 = 2240.9407
Generation: 1600. 誤差 = 2215.235
Generation: 1700. 誤差 = 2189.9873
Generation: 1800. 誤差 = 2165.182
Generation: 1900. 誤差 = 2140.8025
Generation: 2000. 誤差 = 2116.8337
Generation: 2100. 誤差 = 2093.2593
Generation: 2200. 誤差 = 2070.064
Generation: 2300. 誤差 = 2047.2323
Generation: 2400. 誤差 = 2024.7489
Generation: 2500. 誤差 = 2002.5984
Generation: 2600. 誤差 = 1980.765
Generation: 2700. 誤差 = 1959.2339
Generation: 2800. 誤差 = 1937.9894
Generation: 2900. 誤差 = 1917.0167
Generation: 3000. 誤差 = 1896.3007
Generation: 3100. 誤差 = 1875.8263
Generation: 3200. 誤差 = 1855.5793
Generation: 3300. 誤差 = 1835.5457
Generation: 3400. 誤差 = 1815.7119
Generation: 3500. 誤差 = 1796.0654
Generation: 3600. 誤差 = 1776.5948
Generation: 3700. 誤差 = 1757.2894
Generation: 3800. 誤差 = 1738.1394
Generation: 3900. 誤差 = 1719.1375
Generation: 4000. 誤差 = 1700.2762
Generation: 4100. 誤差 = 1681.5513
Generation: 4200. 誤差 = 1662.9575
Generation: 4300. 誤差 = 1644.4926
Generation: 4400. 誤差 = 1626.1547
Generation: 4500. 誤差 = 1607.9418
Generation: 4600. 誤差 = 1589.8544
Generation: 4700. 誤差 = 1571.8894
Generation: 4800. 誤差 = 1554.0518
Generation: 4900. 誤差 = 1536.3359
Generation: 5000. 誤差 = 1518.7477
Generation: 5100. 誤差 = 1501.2825
Generation: 5200. 誤差 = 1483.9448
Generation: 5300. 誤差 = 1466.7319
Generation: 5400. 誤差 = 1449.6462
Generation: 5500. 誤差 = 1432.6859
Generation: 5600. 誤差 = 1415.8538
Generation: 5700. 誤差 = 1399.1469
Generation: 5800. 誤差 = 1382.5685
Generation: 5900. 誤差 = 1366.115
Generation: 6000. 誤差 = 1349.7905
Generation: 6100. 誤差 = 1333.5906
```

```
Generation: 6200. 誤差 = 1317.5192
Generation: 6300. 誤差 = 1301.5728
Generation: 6400. 誤差 = 1285.7533
Generation: 6500. 誤差 = 1270.0598
Generation: 6600. 誤差 = 1254.4907
Generation: 6700. 誤差 = 1239.0492
Generation: 6800. 誤差 = 1223.731
Generation: 6900. 誤差 = 1208.5386
Generation: 7000. 誤差 = 1193.471
Generation: 7100. 誤差 = 1178.5264
Generation: 7200. 誤差 = 1163.7075
Generation: 7300. 誤差 = 1149.0118
Generation: 7400. 誤差 = 1134.439
Generation: 7500. 誤差 = 1119.9907
Generation: 7600. 誤差 = 1105.665
Generation: 7700. 誤差 = 1091.4614
Generation: 7800. 誤差 = 1077.3811
Generation: 7900. 誤差 = 1063.4231
Generation: 8000. 誤差 = 1049.5868
Generation: 8100. 誤差 = 1035.8721
Generation: 8200. 誤差 = 1022.28
Generation: 8300. 誤差 = 1008.8091
Generation: 8400. 誤差 = 995.45917
Generation: 8500. 誤差 = 982.23193
Generation: 8600. 誤差 = 969.12335
Generation: 8700. 誤差 = 956.1356
Generation: 8800. 誤差 = 943.27026
Generation: 8900. 誤差 = 930.5217
Generation: 9000. 誤差 = 917.8977
Generation: 9100. 誤差 = 905.38947
Generation: 9200. 誤差 = 893.00476
Generation: 9300. 誤差 = 880.7364
Generation: 9400. 誤差 = 868.5899
Generation: 9500. 誤差 = 856.5608
Generation: 9600. 誤差 = 844.6525
Generation: 9700. 誤差 = 832.86127
Generation: 9800. 誤差 = 821.1902
Generation: 9900. 誤差 = 809.63617
Generation: 10000. 誤差 = 798.2025
[[ 4.237667 ]
 [ 9.442325 ]
 [-4.06645  ]
 [ 9.2579775]]
```



[try]

- 誤差が収束するようlearning_rateを調整しよう(0.001を0.1に変更。)

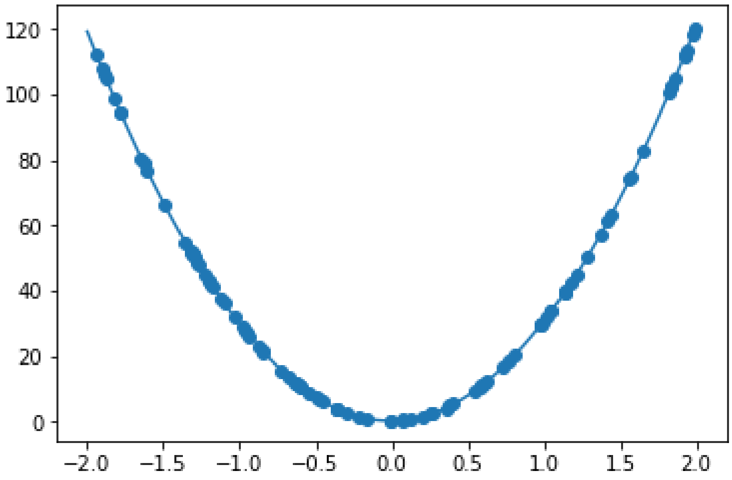
In [18]:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

iters_num = 10000
plot_interval = 100

# データを生成
n=100
x = np.random.rand(n).astype(np.float32) * 4 - 2
d =  30 * x ** 2 + 0.5 * x + 0.2

#  ノイズを加える
noise = 0.05
d = d + noise * np.random.randn(n)

# モデル
# bを使っていないことに注意.
xt = tf.placeholder(tf.float32, [None, 4])
dt = tf.placeholder(tf.float32, [None, 1])
W = tf.Variable(tf.random_normal([4, 1], stddev=0.01))
y = tf.matmul(xt,W)

# 誤差関数 平均２乗誤差
loss = tf.reduce_mean(tf.square(y - dt))
optimizer = tf.train.AdamOptimizer(0.1)
train = optimizer.minimize(loss)

# 初期化
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

# 作成したデータをトレーニングデータとして準備
d_train = d.reshape(-1,1)
x_train = np.zeros([n, 4])
for i in range(n):
    for j in range(4):
        x_train[i, j] = x[i]**j

#  トレーニング
for i in range(iters_num):
    if (i+1) % plot_interval == 0:
        loss_val = sess.run(loss, feed_dict={xt:x_train, dt:d_train})
        W_val = sess.run(W)
        print('Generation: ' + str(i+1) + '. 誤差 = ' + str(loss_val))
    sess.run(train, feed_dict={xt:x_train,dt:d_train})

print(W_val[::-1])

# 予測関数
def predict(x):
    result = 0.
    for i in range(0,4):
        result += W_val[i,0] * x ** i
    return result

fig = plt.figure()
subplot = fig.add_subplot(1,1,1)
```

```
plt.scatter(x ,d)
linex = np.linspace(-2,2,100)
liney = predict(linex)
subplot.plot(linex,liney)
plt.show()
```

```
Generation: 100. 誤差 = 1000.7947
Generation: 200. 誤差 = 338.7539
Generation: 300. 誤差 = 159.625
Generation: 400. 誤差 = 102.543076
Generation: 500. 誤差 = 69.70407
Generation: 600. 誤差 = 45.770912
Generation: 700. 誤差 = 28.660395
Generation: 800. 誤差 = 17.090307
Generation: 900. 誤差 = 9.697431
Generation: 1000. 誤差 = 5.2305775
Generation: 1100. 誤差 = 2.6783893
Generation: 1200. 誤差 = 1.3003355
Generation: 1300. 誤差 = 0.5978639
Generation: 1400. 誤差 = 0.2602947
Generation: 1500. 誤差 = 0.10762539
Generation: 1600. 誤差 = 0.042778827
Generation: 1700. 誤差 = 0.016966928
Generation: 1800. 誤差 = 0.007366726
Generation: 1900. 誤差 = 0.0040382203
Generation: 2000. 誤差 = 0.0029663749
Generation: 2100. 誤差 = 0.0026466073
Generation: 2200. 誤差 = 0.002558667
Generation: 2300. 誤差 = 0.0025364738
Generation: 2400. 誤差 = 0.002531321
Generation: 2500. 誤差 = 0.0025302577
Generation: 2600. 誤差 = 0.0025300742
Generation: 2700. 誤差 = 0.0025300109
Generation: 2800. 誤差 = 0.002530031
Generation: 2900. 誤差 = 0.0025300079
Generation: 3000. 誤差 = 0.002529989
Generation: 3100. 誤差 = 0.0025300074
Generation: 3200. 誤差 = 0.0025300079
Generation: 3300. 誤差 = 0.0025300067
Generation: 3400. 誤差 = 0.0025300083
Generation: 3500. 誤差 = 0.0025300118
Generation: 3600. 誤差 = 0.0025300253
Generation: 3700. 誤差 = 0.0025300323
Generation: 3800. 誤差 = 0.0025300318
Generation: 3900. 誤差 = 0.0025300167
Generation: 4000. 誤差 = 0.0025299883
Generation: 4100. 誤差 = 0.002530022
Generation: 4200. 誤差 = 0.0025300074
Generation: 4300. 誤差 = 0.002529998
Generation: 4400. 誤差 = 0.0025300132
Generation: 4500. 誤差 = 0.0025300127
Generation: 4600. 誤差 = 0.0025300076
Generation: 4700. 誤差 = 0.0025300381
Generation: 4800. 誤差 = 0.0025300256
Generation: 4900. 誤差 = 0.0025300088
Generation: 5000. 誤差 = 0.0025300253
Generation: 5100. 誤差 = 0.0025300246
Generation: 5200. 誤差 = 0.0025300172
Generation: 5300. 誤差 = 0.002530029
Generation: 5400. 誤差 = 0.0025300141
Generation: 5500. 誤差 = 0.002530029
Generation: 5600. 誤差 = 0.002530029
Generation: 5700. 誤差 = 0.0025300365
Generation: 5800. 誤差 = 0.0025300365
Generation: 5900. 誤差 = 0.0025300318
Generation: 6000. 誤差 = 0.0025300353
Generation: 6100. 誤差 = 0.0025300318
```

```
Generation: 6200. 誤差 = 0.002530033
Generation: 6300. 誤差 = 0.0025300265
Generation: 6400. 誤差 = 0.002530027
Generation: 6500. 誤差 = 0.00253003
Generation: 6600. 誤差 = 0.0025300365
Generation: 6700. 誤差 = 0.002530017
Generation: 6800. 誤差 = 0.0025300223
Generation: 6900. 誤差 = 0.0025300207
Generation: 7000. 誤差 = 0.002530033
Generation: 7100. 誤差 = 0.0029303504
Generation: 7200. 誤差 = 0.002530018
Generation: 7300. 誤差 = 0.0025299925
Generation: 7400. 誤差 = 0.0025781246
Generation: 7500. 誤差 = 0.0025300414
Generation: 7600. 誤差 = 0.002529984
Generation: 7700. 誤差 = 0.0027396241
Generation: 7800. 誤差 = 0.0025300141
Generation: 7900. 誤差 = 0.0025347432
Generation: 8000. 誤差 = 0.0025301247
Generation: 8100. 誤差 = 0.0025299871
Generation: 8200. 誤差 = 0.0025302928
Generation: 8300. 誤差 = 0.002531091
Generation: 8400. 誤差 = 0.002530015
Generation: 8500. 誤差 = 0.0025299971
Generation: 8600. 誤差 = 0.0025300093
Generation: 8700. 誤差 = 0.002536235
Generation: 8800. 誤差 = 0.0025299948
Generation: 8900. 誤差 = 0.002529994
Generation: 9000. 誤差 = 0.002533938
Generation: 9100. 誤差 = 0.002530016
Generation: 9200. 誤差 = 0.0025300095
Generation: 9300. 誤差 = 0.0025318987
Generation: 9400. 誤差 = 0.0025299897
Generation: 9500. 誤差 = 0.0066457605
Generation: 9600. 誤差 = 0.0025300877
Generation: 9700. 誤差 = 0.0025299944
Generation: 9800. 誤差 = 0.00280275
Generation: 9900. 誤差 = 0.0025299934
Generation: 10000. 誤差 = 0.0025299964
[[4.5494391e-03]
 [2.9994068e+01]
 [4.8868811e-01]
 [2.0332426e-01]]
```



[try]

- 誤差が収束するようiters_numを調整しよう

In [19]:

```
iters_num = 50000
plot_interval = 100

# データを生成
n=100
x = np.random.rand(n).astype(np.float32) * 4 - 2
d =  30 * x ** 2 + 0.5 * x + 0.2

#  ノイズを加える
noise = 0.05
d = d + noise * np.random.randn(n)

# モデル
# bを使っていないことに注意.
xt = tf.placeholder(tf.float32, [None, 4])
dt = tf.placeholder(tf.float32, [None, 1])
W = tf.Variable(tf.random_normal([4, 1], stddev=0.01))
y = tf.matmul(xt,W)

# 誤差関数 平均２乗誤差
loss = tf.reduce_mean(tf.square(y - dt))
optimizer = tf.train.AdamOptimizer(0.001)
train = optimizer.minimize(loss)

# 初期化
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

# 作成したデータをトレーニングデータとして準備
d_train = d.reshape(-1,1)
x_train = np.zeros([n, 4])
for i in range(n):
    for j in range(4):
        x_train[i, j] = x[i]**j

#  トレーニング
for i in range(iters_num):
    if (i+1) % plot_interval == 0:
        loss_val = sess.run(loss, feed_dict={xt:x_train, dt:d_train})
        W_val = sess.run(W)
        print('Generation: ' + str(i+1) + '. 誤差 = ' + str(loss_val))
    sess.run(train, feed_dict={xt:x_train,dt:d_train})

print(W_val[::-1])

# 予測関数
def predict(x):
    result = 0.
    for i in range(0,4):
        result += W_val[i,0] * x ** i
    return result

fig = plt.figure()
subplot = fig.add_subplot(1,1,1)
plt.scatter(x ,d)
linex = np.linspace(-2,2,100)
liney = predict(linex)
```

```
subplot.plot(linex,liney)
plt.show()
```

```
Generation: 100. 誤差 = 2443.1685
Generation: 200. 誤差 = 2408.7278
Generation: 300. 誤差 = 2375.0154
Generation: 400. 誤差 = 2342.0115
Generation: 500. 誤差 = 2309.6956
Generation: 600. 誤差 = 2278.051
Generation: 700. 誤差 = 2247.0579
Generation: 800. 誤差 = 2216.6995
Generation: 900. 誤差 = 2186.9592
Generation: 1000. 誤差 = 2157.8208
Generation: 1100. 誤差 = 2129.2686
Generation: 1200. 誤差 = 2101.2876
Generation: 1300. 誤差 = 2073.863
Generation: 1400. 誤差 = 2046.9812
Generation: 1500. 誤差 = 2020.6282
Generation: 1600. 誤差 = 1994.7908
Generation: 1700. 誤差 = 1969.4558
Generation: 1800. 誤差 = 1944.6111
Generation: 1900. 誤差 = 1920.2444
Generation: 2000. 誤差 = 1896.3434
Generation: 2100. 誤差 = 1872.8969
Generation: 2200. 誤差 = 1849.8934
Generation: 2300. 誤差 = 1827.3213
Generation: 2400. 誤差 = 1805.17
Generation: 2500. 誤差 = 1783.4285
Generation: 2600. 誤差 = 1762.0853
Generation: 2700. 誤差 = 1741.1306
Generation: 2800. 誤差 = 1720.5535
Generation: 2900. 誤差 = 1700.3433
Generation: 3000. 誤差 = 1680.4893
Generation: 3100. 誤差 = 1660.9811
Generation: 3200. 誤差 = 1641.8081
Generation: 3300. 誤差 = 1622.9594
Generation: 3400. 誤差 = 1604.425
Generation: 3500. 誤差 = 1586.1935
Generation: 3600. 誤差 = 1568.2543
Generation: 3700. 誤差 = 1550.5962
Generation: 3800. 誤差 = 1533.2086
Generation: 3900. 誤差 = 1516.0803
Generation: 4000. 誤差 = 1499.1997
Generation: 4100. 誤差 = 1482.5564
Generation: 4200. 誤差 = 1466.1384
Generation: 4300. 誤差 = 1449.9346
Generation: 4400. 誤差 = 1433.9353
Generation: 4500. 誤差 = 1418.1272
Generation: 4600. 誤差 = 1402.5005
Generation: 4700. 誤差 = 1387.047
Generation: 4800. 誤差 = 1371.7535
Generation: 4900. 誤差 = 1356.611
Generation: 5000. 誤差 = 1341.6106
Generation: 5100. 誤差 = 1326.744
Generation: 5200. 誤差 = 1312.0038
Generation: 5300. 誤差 = 1297.3817
Generation: 5400. 誤差 = 1282.87
Generation: 5500. 誤差 = 1268.4669
Generation: 5600. 誤差 = 1254.1656
Generation: 5700. 誤差 = 1239.9626
Generation: 5800. 誤差 = 1225.8561
Generation: 5900. 誤差 = 1211.8422
Generation: 6000. 誤差 = 1197.9213
Generation: 6100. 誤差 = 1184.0906
```

```
Generation: 6200. 誤差 = 1170.3516
Generation: 6300. 誤差 = 1156.7024
Generation: 6400. 誤差 = 1143.1459
Generation: 6500. 誤差 = 1129.6786
Generation: 6600. 誤差 = 1116.305
Generation: 6700. 誤差 = 1103.0255
Generation: 6800. 誤差 = 1089.8367
Generation: 6900. 誤差 = 1076.7432
Generation: 7000. 誤差 = 1063.7455
Generation: 7100. 誤差 = 1050.8438
Generation: 7200. 誤差 = 1038.0387
Generation: 7300. 誤差 = 1025.3303
Generation: 7400. 誤差 = 1012.71906
Generation: 7500. 誤差 = 1000.20685
Generation: 7600. 誤差 = 987.79443
Generation: 7700. 誤差 = 975.47974
Generation: 7800. 誤差 = 963.2643
Generation: 7900. 誤差 = 951.14905
Generation: 8000. 誤差 = 939.13306
Generation: 8100. 誤差 = 927.21735
Generation: 8200. 誤差 = 915.40186
Generation: 8300. 誤差 = 903.6861
Generation: 8400. 誤差 = 892.072
Generation: 8500. 誤差 = 880.5577
Generation: 8600. 誤差 = 869.1467
Generation: 8700. 誤差 = 857.83417
Generation: 8800. 誤差 = 846.622
Generation: 8900. 誤差 = 835.5128
Generation: 9000. 誤差 = 824.5012
Generation: 9100. 誤差 = 813.59485
Generation: 9200. 誤差 = 802.78674
Generation: 9300. 誤差 = 792.0814
Generation: 9400. 誤差 = 781.47565
Generation: 9500. 誤差 = 770.97186
Generation: 9600. 誤差 = 760.5686
Generation: 9700. 誤差 = 750.2664
Generation: 9800. 誤差 = 740.06506
Generation: 9900. 誤差 = 729.96466
Generation: 10000. 誤差 = 719.96466
Generation: 10100. 誤差 = 710.0661
Generation: 10200. 誤差 = 700.2668
Generation: 10300. 誤差 = 690.5697
Generation: 10400. 誤差 = 680.97064
Generation: 10500. 誤差 = 671.475
Generation: 10600. 誤差 = 662.0767
Generation: 10700. 誤差 = 652.7803
Generation: 10800. 誤差 = 643.58307
Generation: 10900. 誤差 = 634.4849
Generation: 11000. 誤差 = 625.4879
Generation: 11100. 誤差 = 616.58844
Generation: 11200. 誤差 = 607.7898
Generation: 11300. 誤差 = 599.0898
Generation: 11400. 誤差 = 590.4875
Generation: 11500. 誤差 = 581.98615
Generation: 11600. 誤差 = 573.5819
Generation: 11700. 誤差 = 565.27576
Generation: 11800. 誤差 = 557.06915
Generation: 11900. 誤差 = 548.95935
Generation: 12000. 誤差 = 540.9467
Generation: 12100. 誤差 = 533.0332
Generation: 12200. 誤差 = 525.216
```

```
Generation: 12300. 誤差 = 517.4951
Generation: 12400. 誤差 = 509.8721
Generation: 12500. 誤差 = 502.34552
Generation: 12600. 誤差 = 494.9146
Generation: 12700. 誤差 = 487.57907
Generation: 12800. 誤差 = 480.3408
Generation: 12900. 誤差 = 473.19736
Generation: 13000. 誤差 = 466.14874
Generation: 13100. 誤差 = 459.1947
Generation: 13200. 誤差 = 452.33548
Generation: 13300. 誤差 = 445.5711
Generation: 13400. 誤差 = 438.9003
Generation: 13500. 誤差 = 432.32297
Generation: 13600. 誤差 = 425.83878
Generation: 13700. 誤差 = 419.4475
Generation: 13800. 誤差 = 413.14905
Generation: 13900. 誤差 = 406.94333
Generation: 14000. 誤差 = 400.82938
Generation: 14100. 誤差 = 394.8069
Generation: 14200. 誤差 = 388.87558
Generation: 14300. 誤差 = 383.035
Generation: 14400. 誤差 = 377.28485
Generation: 14500. 誤差 = 371.62463
Generation: 14600. 誤差 = 366.05408
Generation: 14700. 誤差 = 360.57266
Generation: 14800. 誤差 = 355.17996
Generation: 14900. 誤差 = 349.87555
Generation: 15000. 誤差 = 344.6589
Generation: 15100. 誤差 = 339.52945
Generation: 15200. 誤差 = 334.48676
Generation: 15300. 誤差 = 329.53024
Generation: 15400. 誤差 = 324.65927
Generation: 15500. 誤差 = 319.87332
Generation: 15600. 誤差 = 315.1716
Generation: 15700. 誤差 = 310.5535
Generation: 15800. 誤差 = 306.01834
Generation: 15900. 誤差 = 301.56534
Generation: 16000. 誤差 = 297.19385
Generation: 16100. 誤差 = 292.90286
Generation: 16200. 誤差 = 288.6916
Generation: 16300. 誤差 = 284.55917
Generation: 16400. 誤差 = 280.50452
Generation: 16500. 誤差 = 276.52673
Generation: 16600. 誤差 = 272.6246
Generation: 16700. 誤差 = 268.79712
Generation: 16800. 誤差 = 265.04294
Generation: 16900. 誤差 = 261.36087
Generation: 17000. 誤差 = 257.74948
Generation: 17100. 誤差 = 254.20726
Generation: 17200. 誤差 = 250.7327
Generation: 17300. 誤差 = 247.32413
Generation: 17400. 誤差 = 243.97993
Generation: 17500. 誤差 = 240.6963
Generation: 17600. 誤差 = 237.47523
Generation: 17700. 誤差 = 234.31157
Generation: 17800. 誤差 = 231.20143
Generation: 17900. 誤差 = 228.14793
Generation: 18000. 誤差 = 225.14484
Generation: 18100. 誤差 = 222.18848
Generation: 18200. 誤差 = 219.27625
Generation: 18300. 誤差 = 216.40564
```

```
Generation: 18400. 誤差 = 213.57399
Generation: 18500. 誤差 = 210.77853
Generation: 18600. 誤差 = 208.01653
Generation: 18700. 誤差 = 205.28517
Generation: 18800. 誤差 = 202.58177
Generation: 18900. 誤差 = 199.90059
Generation: 19000. 誤差 = 197.23967
Generation: 19100. 誤差 = 194.59744
Generation: 19200. 誤差 = 191.97047
Generation: 19300. 誤差 = 189.35664
Generation: 19400. 誤差 = 186.75536
Generation: 19500. 誤差 = 184.16496
Generation: 19600. 誤差 = 181.58414
Generation: 19700. 誤差 = 179.01228
Generation: 19800. 誤差 = 176.44969
Generation: 19900. 誤差 = 173.89644
Generation: 20000. 誤差 = 171.35281
Generation: 20100. 誤差 = 168.81927
Generation: 20200. 誤差 = 166.29646
Generation: 20300. 誤差 = 163.7851
Generation: 20400. 誤差 = 161.28607
Generation: 20500. 誤差 = 158.79922
Generation: 20600. 誤差 = 156.32567
Generation: 20700. 誤差 = 153.86703
Generation: 20800. 誤差 = 151.42311
Generation: 20900. 誤差 = 148.9942
Generation: 21000. 誤差 = 146.58229
Generation: 21100. 誤差 = 144.18611
Generation: 21200. 誤差 = 141.8073
Generation: 21300. 誤差 = 139.44766
Generation: 21400. 誤差 = 137.10352
Generation: 21500. 誤差 = 134.77814
Generation: 21600. 誤差 = 132.47232
Generation: 21700. 誤差 = 130.18622
Generation: 21800. 誤差 = 127.91986
Generation: 21900. 誤差 = 125.673294
Generation: 22000. 誤差 = 123.4466
Generation: 22100. 誤差 = 121.23981
Generation: 22200. 誤差 = 119.05292
Generation: 22300. 誤差 = 116.88598
Generation: 22400. 誤差 = 114.73908
Generation: 22500. 誤差 = 112.61207
Generation: 22600. 誤差 = 110.50637
Generation: 22700. 誤差 = 108.423485
Generation: 22800. 誤差 = 106.36059
Generation: 22900. 誤差 = 104.31766
Generation: 23000. 誤差 = 102.29709
Generation: 23100. 誤差 = 100.297966
Generation: 23200. 誤差 = 98.31873
Generation: 23300. 誤差 = 96.36215
Generation: 23400. 誤差 = 94.426575
Generation: 23500. 誤差 = 92.511696
Generation: 23600. 誤差 = 90.61939
Generation: 23700. 誤差 = 88.74693
Generation: 23800. 誤差 = 86.89777
Generation: 23900. 誤差 = 85.06828
Generation: 24000. 誤差 = 83.26183
Generation: 24100. 誤差 = 81.47521
Generation: 24200. 誤差 = 79.71163
Generation: 24300. 誤差 = 77.96812
Generation: 24400. 誤差 = 76.24706
```

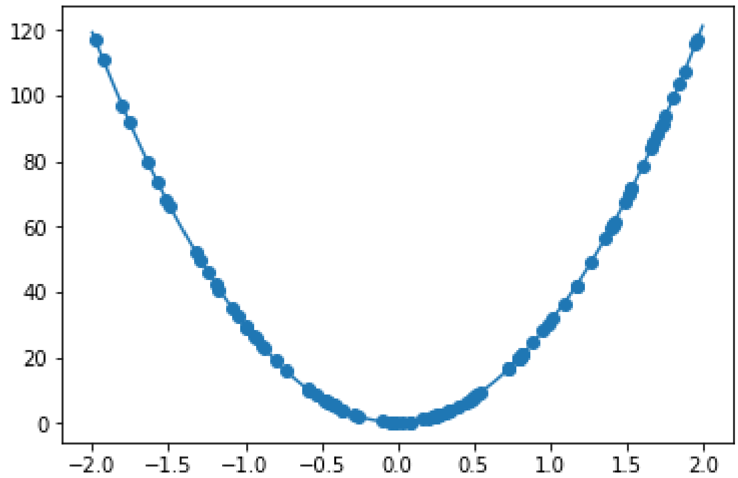
```
Generation: 24500. 誤差 = 74.54671
Generation: 24600. 誤差 = 72.86782
Generation: 24700. 誤差 = 71.210655
Generation: 24800. 誤差 = 69.57411
Generation: 24900. 誤差 = 67.95959
Generation: 25000. 誤差 = 66.36611
Generation: 25100. 誤差 = 64.793396
Generation: 25200. 誤差 = 63.242535
Generation: 25300. 誤差 = 61.712753
Generation: 25400. 誤差 = 60.20375
Generation: 25500. 誤差 = 58.715824
Generation: 25600. 誤差 = 57.249493
Generation: 25700. 誤差 = 55.80395
Generation: 25800. 誤差 = 54.37918
Generation: 25900. 誤差 = 52.975235
Generation: 26000. 誤差 = 51.592354
Generation: 26100. 誤差 = 50.230515
Generation: 26200. 誤差 = 48.88937
Generation: 26300. 誤差 = 47.568897
Generation: 26400. 誤差 = 46.269066
Generation: 26500. 誤差 = 44.989845
Generation: 26600. 誤差 = 43.7312
Generation: 26700. 誤差 = 42.493065
Generation: 26800. 誤差 = 41.275414
Generation: 26900. 誤差 = 40.078175
Generation: 27000. 誤差 = 38.901276
Generation: 27100. 誤差 = 37.74463
Generation: 27200. 誤差 = 36.608173
Generation: 27300. 誤差 = 35.491837
Generation: 27400. 誤差 = 34.39554
Generation: 27500. 誤差 = 33.31921
Generation: 27600. 誤差 = 32.26283
Generation: 27700. 誤差 = 31.22644
Generation: 27800. 誤差 = 30.209742
Generation: 27900. 誤差 = 29.212646
Generation: 28000. 誤差 = 28.235235
Generation: 28100. 誤差 = 27.277424
Generation: 28200. 誤差 = 26.338882
Generation: 28300. 誤差 = 25.419872
Generation: 28400. 誤差 = 24.520031
Generation: 28500. 誤差 = 23.639399
Generation: 28600. 誤差 = 22.77783
Generation: 28700. 誤差 = 21.935253
Generation: 28800. 誤差 = 21.11151
Generation: 28900. 誤差 = 20.306704
Generation: 29000. 誤差 = 19.520472
Generation: 29100. 誤差 = 18.752733
Generation: 29200. 誤差 = 18.003635
Generation: 29300. 誤差 = 17.272812
Generation: 29400. 誤差 = 16.560204
Generation: 29500. 誤差 = 15.865724
Generation: 29600. 誤差 = 15.18923
Generation: 29700. 誤差 = 14.530587
Generation: 29800. 誤差 = 13.889705
Generation: 29900. 誤差 = 13.266402
Generation: 30000. 誤差 = 12.660532
Generation: 30100. 誤差 = 12.072055
Generation: 30200. 誤差 = 11.500737
Generation: 30300. 誤差 = 10.946376
Generation: 30400. 誤差 = 10.408994
Generation: 30500. 誤差 = 9.88823
```

```
Generation: 30600. 誤差 = 9.384121
Generation: 30700. 誤差 = 8.896343
Generation: 30800. 誤差 = 8.42477
Generation: 30900. 誤差 = 7.96923
Generation: 31000. 誤差 = 7.5295577
Generation: 31100. 誤差 = 7.105573
Generation: 31200. 誤差 = 6.6970787
Generation: 31300. 誤差 = 6.303854
Generation: 31400. 誤差 = 5.9256935
Generation: 31500. 誤差 = 5.562395
Generation: 31600. 誤差 = 5.2137513
Generation: 31700. 誤差 = 4.87954
Generation: 31800. 誤差 = 4.559515
Generation: 31900. 誤差 = 4.2534647
Generation: 32000. 誤差 = 3.961162
Generation: 32100. 誤差 = 3.6823394
Generation: 32200. 誤差 = 3.4167526
Generation: 32300. 誤差 = 3.1641228
Generation: 32400. 誤差 = 2.9241996
Generation: 32500. 誤差 = 2.6967065
Generation: 32600. 誤差 = 2.481377
Generation: 32700. 誤差 = 2.2778962
Generation: 32800. 誤差 = 2.0859842
Generation: 32900. 誤差 = 1.9053346
Generation: 33000. 誤差 = 1.7356573
Generation: 33100. 誤差 = 1.5766104
Generation: 33200. 誤差 = 1.4278903
Generation: 33300. 誤差 = 1.2891583
Generation: 33400. 誤差 = 1.1600832
Generation: 33500. 誤差 = 1.040321
Generation: 33600. 誤差 = 0.92952347
Generation: 33700. 誤差 = 0.82733667
Generation: 33800. 誤差 = 0.7334047
Generation: 33900. 誤差 = 0.6473639
Generation: 34000. 誤差 = 0.568844
Generation: 34100. 誤差 = 0.49747458
Generation: 34200. 誤差 = 0.4328746
Generation: 34300. 誤差 = 0.3746727
Generation: 34400. 誤差 = 0.32248497
Generation: 34500. 誤差 = 0.2759327
Generation: 34600. 誤差 = 0.23463911
Generation: 34700. 誤差 = 0.19822556
Generation: 34800. 誤差 = 0.1663171
Generation: 34900. 誤差 = 0.13854635
Generation: 35000. 誤差 = 0.114553906
Generation: 35100. 誤差 = 0.09398413
Generation: 35200. 誤差 = 0.07649599
Generation: 35300. 誤差 = 0.06175964
Generation: 35400. 誤差 = 0.04946148
Generation: 35500. 誤差 = 0.039301928
Generation: 35600. 誤差 = 0.030999407
Generation: 35700. 誤差 = 0.024293266
Generation: 35800. 誤差 = 0.01894249
Generation: 35900. 誤差 = 0.014729746
Generation: 36000. 誤差 = 0.011458495
Generation: 36100. 誤差 = 0.008956441
Generation: 36200. 誤差 = 0.0070726504
Generation: 36300. 誤差 = 0.0056772875
Generation: 36400. 誤差 = 0.0046627778
Generation: 36500. 誤差 = 0.003938858
Generation: 36600. 誤差 = 0.003432084
```

```
Generation: 36700. 誤差 = 0.0030848074
Generation: 36800. 誤差 = 0.0028521293
Generation: 36900. 誤差 = 0.002699988
Generation: 37000. 誤差 = 0.00260277
Generation: 37100. 誤差 = 0.0025423407
Generation: 37200. 誤差 = 0.0025057143
Generation: 37300. 誤差 = 0.002484363
Generation: 37400. 誤差 = 0.0024719175
Generation: 37500. 誤差 = 0.0024651638
Generation: 37600. 誤差 = 0.002461565
Generation: 37700. 誤差 = 0.002459694
Generation: 37800. 誤差 = 0.0024587428
Generation: 37900. 誤差 = 0.0024583607
Generation: 38000. 誤差 = 0.0024580867
Generation: 38100. 誤差 = 0.0024580199
Generation: 38200. 誤差 = 0.0024579843
Generation: 38300. 誤差 = 0.0024579633
Generation: 38400. 誤差 = 0.0024579847
Generation: 38500. 誤差 = 0.0024579437
Generation: 38600. 誤差 = 0.0024579517
Generation: 38700. 誤差 = 0.0024579898
Generation: 38800. 誤差 = 0.0024579486
Generation: 38900. 誤差 = 0.0024579982
Generation: 39000. 誤差 = 0.0024579559
Generation: 39100. 誤差 = 0.0024579654
Generation: 39200. 誤差 = 0.0024579626
Generation: 39300. 誤差 = 0.0024579617
Generation: 39400. 誤差 = 0.0024579328
Generation: 39500. 誤差 = 0.002457936
Generation: 39600. 誤差 = 0.0024579624
Generation: 39700. 誤差 = 0.0024579621
Generation: 39800. 誤差 = 0.0024579572
Generation: 39900. 誤差 = 0.0024579554
Generation: 40000. 誤差 = 0.0024579538
Generation: 40100. 誤差 = 0.0024579538
Generation: 40200. 誤差 = 0.0024579377
Generation: 40300. 誤差 = 0.0024579442
Generation: 40400. 誤差 = 0.0024579389
Generation: 40500. 誤差 = 0.002457958
Generation: 40600. 誤差 = 0.002457952
Generation: 40700. 誤差 = 0.0024579398
Generation: 40800. 誤差 = 0.0024579628
Generation: 40900. 誤差 = 0.0024579493
Generation: 41000. 誤差 = 0.0024579612
Generation: 41100. 誤差 = 0.0024580075
Generation: 41200. 誤差 = 0.002458107
Generation: 41300. 誤差 = 0.0024579808
Generation: 41400. 誤差 = 0.0024579598
Generation: 41500. 誤差 = 0.0024579451
Generation: 41600. 誤差 = 0.0024579547
Generation: 41700. 誤差 = 0.002457933
Generation: 41800. 誤差 = 0.0024579526
Generation: 41900. 誤差 = 0.0024579512
Generation: 42000. 誤差 = 0.0024579612
Generation: 42100. 誤差 = 0.0024579437
Generation: 42200. 誤差 = 0.002457954
Generation: 42300. 誤差 = 0.002457948
Generation: 42400. 誤差 = 0.0024579582
Generation: 42500. 誤差 = 0.0024579335
Generation: 42600. 誤差 = 0.0024579319
Generation: 42700. 誤差 = 0.00245794
```

```
Generation: 42800. 誤差 = 0.0024579703
Generation: 42900. 誤差 = 0.0024580285
Generation: 43000. 誤差 = 0.0024579596
Generation: 43100. 誤差 = 0.0024580148
Generation: 43200. 誤差 = 0.002457944
Generation: 43300. 誤差 = 0.0024580196
Generation: 43400. 誤差 = 0.002457973
Generation: 43500. 誤差 = 0.0024579638
Generation: 43600. 誤差 = 0.0024579796
Generation: 43700. 誤差 = 0.0024579584
Generation: 43800. 誤差 = 0.0024579742
Generation: 43900. 誤差 = 0.0024579822
Generation: 44000. 誤差 = 0.0024579614
Generation: 44100. 誤差 = 0.0024579496
Generation: 44200. 誤差 = 0.0024579642
Generation: 44300. 誤差 = 0.0024579847
Generation: 44400. 誤差 = 0.0024579626
Generation: 44500. 誤差 = 0.002457951
Generation: 44600. 誤差 = 0.0024579756
Generation: 44700. 誤差 = 0.002457956
Generation: 44800. 誤差 = 0.002457976
Generation: 44900. 誤差 = 0.00245796
Generation: 45000. 誤差 = 0.0024579586
Generation: 45100. 誤差 = 0.002457968
Generation: 45200. 誤差 = 0.0024579586
Generation: 45300. 誤差 = 0.002457959
Generation: 45400. 誤差 = 0.0024579605
Generation: 45500. 誤差 = 0.002457961
Generation: 45600. 誤差 = 0.0024579624
Generation: 45700. 誤差 = 0.0024579575
Generation: 45800. 誤差 = 0.002457974
Generation: 45900. 誤差 = 0.0024581521
Generation: 46000. 誤差 = 0.002457973
Generation: 46100. 誤差 = 0.0024579512
Generation: 46200. 誤差 = 0.002457959
Generation: 46300. 誤差 = 0.0024579891
Generation: 46400. 誤差 = 0.0024579533
Generation: 46500. 誤差 = 0.0024579572
Generation: 46600. 誤差 = 0.0024579745
Generation: 46700. 誤差 = 0.0024579552
Generation: 46800. 誤差 = 0.002457975
Generation: 46900. 誤差 = 0.0024579566
Generation: 47000. 誤差 = 0.0024579605
Generation: 47100. 誤差 = 0.0024579784
Generation: 47200. 誤差 = 0.0024579582
Generation: 47300. 誤差 = 0.0024579489
Generation: 47400. 誤差 = 0.002457965
Generation: 47500. 誤差 = 0.002457955
Generation: 47600. 誤差 = 0.0024579782
Generation: 47700. 誤差 = 0.0024579596
Generation: 47800. 誤差 = 0.0024579559
Generation: 47900. 誤差 = 0.0024579593
Generation: 48000. 誤差 = 0.0024579593
Generation: 48100. 誤差 = 0.0024579563
Generation: 48200. 誤差 = 0.0024579687
Generation: 48300. 誤差 = 0.0024579628
Generation: 48400. 誤差 = 0.002458017
Generation: 48500. 誤差 = 0.0024579645
Generation: 48600. 誤差 = 0.0024579398
Generation: 48700. 誤差 = 0.0024579705
Generation: 48800. 誤差 = 0.002457972
```

```
Generation: 48900. 誤差 = 0.0024579815
Generation: 49000. 誤差 = 0.00245797
Generation: 49100. 誤差 = 0.0024579754
Generation: 49200. 誤差 = 0.0024579468
Generation: 49300. 誤差 = 0.0024579745
Generation: 49400. 誤差 = 0.0024579493
Generation: 49500. 誤差 = 0.0024579572
Generation: 49600. 誤差 = 0.002457943
Generation: 49700. 誤差 = 0.0024579575
Generation: 49800. 誤差 = 0.0024580092
Generation: 49900. 誤差 = 0.0024579528
Generation: 50000. 誤差 = 0.0024580394
[[2.3939405e-03]
 [2.9999142e+01]
 [4.9005434e-01]
 [2.0139773e-01]]
```



分類1層 (mnist) (入力層→出力層)

[try]

- x：入力値, d：教師データ, W：重み, b：バイアス をそれぞれ定義しよう

In [20]:

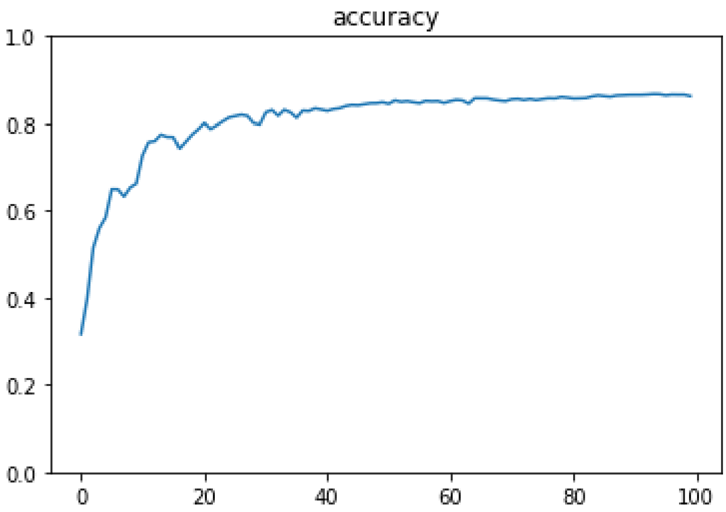
```
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

iters_num = 100
batch_size = 100
plot_interval = 1

# -------------- ここを補填 ------------------------
x = tf.placeholder(tf.float32, [None, 784])
d = tf.placeholder(tf.float32, [None, 10])
W = tf.Variable(tf.random_normal([784, 10], stddev=0.01))
b = tf.Variable(tf.zeros([10]))
# ------------------------------------------------------

y = tf.nn.softmax(tf.matmul(x, W) + b)

# 交差エントロピー
cross_entropy = -tf.reduce_sum(d * tf.log(y), reduction_indices=[1])
loss = tf.reduce_mean(cross_entropy)
train = tf.train.GradientDescentOptimizer(0.1).minimize(loss)

# 正誤を保存
correct = tf.equal(tf.argmax(y, 1), tf.argmax(d, 1))
# 正解率
accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

accuracies = []
for i in range(iters_num):
    x_batch, d_batch = mnist.train.next_batch(batch_size)
    sess.run(train, feed_dict={x: x_batch, d: d_batch})
    if (i+1) % plot_interval == 0:
        print(sess.run(correct, feed_dict={x: mnist.test.images, d: mnist.test.labels}))
        accuracy_val = sess.run(accuracy, feed_dict={x: mnist.test.images, d: mnist.test.labels
})
        accuracies.append(accuracy_val)
        print('Generation: ' + str(i+1) + '. 正解率 = ' + str(accuracy_val))


lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies)
plt.title("accuracy")
plt.ylim(0, 1.0)
plt.show()
```

```
WARNING:tensorflow:From <ipython-input-20-77bf5b52a1ed>:4: read_data_sets (from te
nsorflow.contrib.learn.python.learn.datasets.mnist) is deprecated and will be remo
ved in a future version.
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/models.
WARNING:tensorflow:From C:¥ProgramData¥Anaconda3¥lib¥site-packages¥tensorflow¥cont
rib¥learn¥python¥learn¥datasets¥mnist.py:260: maybe_download (from tensorflow.cont
rib.learn.python.learn.datasets.base) is deprecated and will be removed in a futur
e version.
Instructions for updating:
Please write your own downloading logic.
WARNING:tensorflow:From C:¥ProgramData¥Anaconda3¥lib¥site-packages¥tensorflow¥cont
rib¥learn¥python¥learn¥datasets¥mnist.py:262: extract_images (from tensorflow.cont
rib.learn.python.learn.datasets.mnist) is deprecated and will be removed in a futu
re version.
Instructions for updating:
Please use tf.data to implement this functionality.
Extracting MNIST_data/train-images-idx3-ubyte.gz
WARNING:tensorflow:From C:¥ProgramData¥Anaconda3¥lib¥site-packages¥tensorflow¥cont
rib¥learn¥python¥learn¥datasets¥mnist.py:267: extract_labels (from tensorflow.cont
rib.learn.python.learn.datasets.mnist) is deprecated and will be removed in a futu
re version.
Instructions for updating:
Please use tf.data to implement this functionality.
Extracting MNIST_data/train-labels-idx1-ubyte.gz
WARNING:tensorflow:From C:¥ProgramData¥Anaconda3¥lib¥site-packages¥tensorflow¥cont
rib¥learn¥python¥learn¥datasets¥mnist.py:110: dense_to_one_hot (from tensorflow.co
ntrib.learn.python.learn.datasets.mnist) is deprecated and will be removed in a fu
ture version.
Instructions for updating:
Please use tf.one_hot on tensors.
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
WARNING:tensorflow:From C:¥ProgramData¥Anaconda3¥lib¥site-packages¥tensorflow¥cont
rib¥learn¥python¥learn¥datasets¥mnist.py:290: DataSet.__init__ (from tensorflow.co
ntrib.learn.python.learn.datasets.mnist) is deprecated and will be removed in a fu
ture version.
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/models.
[False False False ... False False False]
Generation: 1. 正解率 = 0.3164
[ True False  True ... False False  True]
Generation: 2. 正解率 = 0.3979
[ True False  True ... False  True  True]
Generation: 3. 正解率 = 0.5153
[ True False  True ... False  True  True]
Generation: 4. 正解率 = 0.5601
[ True False  True ... False False  True]
Generation: 5. 正解率 = 0.5844
[ True False  True ... False  True  True]
Generation: 6. 正解率 = 0.6493
[ True  True  True ... False False  True]
Generation: 7. 正解率 = 0.649
[ True False  True ... False False  True]
Generation: 8. 正解率 = 0.6325
[ True  True  True ... False False  True]
Generation: 9. 正解率 = 0.653
[ True  True  True ... False False  True]
Generation: 10. 正解率 = 0.6624
[ True  True  True ... False False  True]
Generation: 11. 正解率 = 0.7268
```

```
[ True  True  True ... False False  True]
Generation: 12. 正解率 = 0.7573
[ True False  True ... False False  True]
Generation: 13. 正解率 = 0.7597
[ True False  True ... False False  True]
Generation: 14. 正解率 = 0.7744
[ True  True  True ... False False  True]
Generation: 15. 正解率 = 0.7692
[ True False  True ...  True False  True]
Generation: 16. 正解率 = 0.7687
[ True False  True ...  True False  True]
Generation: 17. 正解率 = 0.7425
[ True False  True ...  True False  True]
Generation: 18. 正解率 = 0.7586
[ True False  True ... False False  True]
Generation: 19. 正解率 = 0.7744
[ True  True  True ...  True False  True]
Generation: 20. 正解率 = 0.7867
[ True  True  True ...  True False  True]
Generation: 21. 正解率 = 0.801
[ True  True  True ...  True False  True]
Generation: 22. 正解率 = 0.787
[ True  True  True ...  True False  True]
Generation: 23. 正解率 = 0.7957
[ True  True  True ...  True False  True]
Generation: 24. 正解率 = 0.805
[ True  True  True ...  True False  True]
Generation: 25. 正解率 = 0.8135
[ True  True  True ...  True False  True]
Generation: 26. 正解率 = 0.8167
[ True  True  True ...  True False  True]
Generation: 27. 正解率 = 0.82
[ True  True  True ...  True False  True]
Generation: 28. 正解率 = 0.8172
[ True  True  True ...  True False  True]
Generation: 29. 正解率 = 0.8008
[ True  True  True ...  True False  True]
Generation: 30. 正解率 = 0.7973
[ True  True  True ...  True False  True]
Generation: 31. 正解率 = 0.8253
[ True  True  True ...  True False  True]
Generation: 32. 正解率 = 0.8305
[ True  True  True ... False False  True]
Generation: 33. 正解率 = 0.8175
[ True  True  True ...  True False  True]
Generation: 34. 正解率 = 0.831
[ True  True  True ... False False  True]
Generation: 35. 正解率 = 0.8253
[ True  True  True ... False False  True]
Generation: 36. 正解率 = 0.8129
[ True  True  True ...  True False  True]
Generation: 37. 正解率 = 0.8293
[ True  True  True ...  True False  True]
Generation: 38. 正解率 = 0.8282
[ True  True  True ...  True False  True]
Generation: 39. 正解率 = 0.8344
[ True  True  True ...  True False  True]
Generation: 40. 正解率 = 0.8317
[ True  True  True ... False False  True]
Generation: 41. 正解率 = 0.8286
[ True  True  True ...  True False  True]
```

```
Generation: 42. 正解率 = 0.8328
[ True  True  True ...  True False  True]
Generation: 43. 正解率 = 0.8346
[ True  True  True ...  True False  True]
Generation: 44. 正解率 = 0.8396
[ True  True  True ...  True False  True]
Generation: 45. 正解率 = 0.8422
[ True  True  True ...  True False  True]
Generation: 46. 正解率 = 0.8414
[ True  True  True ...  True False  True]
Generation: 47. 正解率 = 0.8437
[ True  True  True ...  True False  True]
Generation: 48. 正解率 = 0.846
[ True  True  True ...  True False  True]
Generation: 49. 正解率 = 0.8462
[ True  True  True ... False  True  True]
Generation: 50. 正解率 = 0.8483
[ True  True  True ... False False  True]
Generation: 51. 正解率 = 0.8447
[ True  True  True ...  True False  True]
Generation: 52. 正解率 = 0.8526
[ True  True  True ... False False  True]
Generation: 53. 正解率 = 0.8493
[ True  True  True ...  True False  True]
Generation: 54. 正解率 = 0.8508
[ True  True  True ...  True False  True]
Generation: 55. 正解率 = 0.8483
[ True  True  True ...  True  True  True]
Generation: 56. 正解率 = 0.8461
[ True  True  True ...  True False  True]
Generation: 57. 正解率 = 0.8514
[ True  True  True ...  True False  True]
Generation: 58. 正解率 = 0.8504
[ True  True  True ...  True  True  True]
Generation: 59. 正解率 = 0.8509
[ True  True  True ...  True  True  True]
Generation: 60. 正解率 = 0.8467
[ True  True  True ...  True  True  True]
Generation: 61. 正解率 = 0.8508
[ True  True  True ...  True  True  True]
Generation: 62. 正解率 = 0.8539
[ True  True  True ...  True  True  True]
Generation: 63. 正解率 = 0.8525
[ True  True  True ...  True  True  True]
Generation: 64. 正解率 = 0.8452
[ True  True  True ...  True  True  True]
Generation: 65. 正解率 = 0.8576
[ True  True  True ...  True  True  True]
Generation: 66. 正解率 = 0.8574
[ True  True  True ...  True  True  True]
Generation: 67. 正解率 = 0.8573
[ True  True  True ...  True  True  True]
Generation: 68. 正解率 = 0.8543
[ True  True  True ...  True  True  True]
Generation: 69. 正解率 = 0.8523
[ True  True  True ...  True False  True]
Generation: 70. 正解率 = 0.8508
[ True  True  True ...  True  True  True]
Generation: 71. 正解率 = 0.8551
[ True  True  True ...  True False  True]
Generation: 72. 正解率 = 0.8561
```

```
[ True  True  True ...  True False  True]
Generation: 73. 正解率 = 0.8535
[ True  True  True ...  True False  True]
Generation: 74. 正解率 = 0.8558
[ True  True  True ...  True False  True]
Generation: 75. 正解率 = 0.8533
[ True  True  True ...  True False  True]
Generation: 76. 正解率 = 0.8556
[ True  True  True ...  True False  True]
Generation: 77. 正解率 = 0.8576
[ True  True  True ...  True False  True]
Generation: 78. 正解率 = 0.8571
[ True  True  True ...  True False  True]
Generation: 79. 正解率 = 0.8604
[ True  True  True ...  True  True  True]
Generation: 80. 正解率 = 0.8589
[ True  True  True ...  True  True  True]
Generation: 81. 正解率 = 0.8569
[ True  True  True ...  True False  True]
Generation: 82. 正解率 = 0.8573
[ True  True  True ...  True False  True]
Generation: 83. 正解率 = 0.8579
[ True  True  True ...  True False  True]
Generation: 84. 正解率 = 0.8616
[ True  True  True ...  True  True  True]
Generation: 85. 正解率 = 0.864
[ True  True  True ...  True False  True]
Generation: 86. 正解率 = 0.8623
[ True  True  True ...  True False  True]
Generation: 87. 正解率 = 0.8609
[ True  True  True ...  True False  True]
Generation: 88. 正解率 = 0.8638
[ True  True  True ...  True False  True]
Generation: 89. 正解率 = 0.8644
[ True  True  True ...  True False  True]
Generation: 90. 正解率 = 0.8652
[ True  True  True ...  True False  True]
Generation: 91. 正解率 = 0.8653
[ True  True  True ...  True False  True]
Generation: 92. 正解率 = 0.8653
[ True  True  True ...  True False  True]
Generation: 93. 正解率 = 0.866
[ True  True  True ...  True False  True]
Generation: 94. 正解率 = 0.8671
[ True  True  True ...  True False  True]
Generation: 95. 正解率 = 0.8668
[ True  True  True ...  True False  True]
Generation: 96. 正解率 = 0.8643
[ True  True  True ...  True False  True]
Generation: 97. 正解率 = 0.8659
[ True  True  True ...  True False  True]
Generation: 98. 正解率 = 0.8656
[ True  True  True ...  True False  True]
Generation: 99. 正解率 = 0.8657
[ True  True  True ...  True False  True]
Generation: 100. 正解率 = 0.8624
```


accuracy
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100

分類3層 (mnist)　（入力層→中間層→中間層→出力層）

tf.train.GradientDescentOptimizer

```
__init__(
    learning_rate,
    use_locking=False,
    name='GradientDescent'
)
```

tf.train.MomentumOptimizer

```
__init__(
    learning_rate,
    momentum,
    use_locking=False,
    name='Momentum',
    use_nesterov=False
)
```

tf.train.AdagradOptimizer

```
__init__(
    learning_rate,
    initial_accumulator_value=0.1,
    use_locking=False,
    name='Adagrad'
)
```

tf.train.RMSPropOptimizer

```
__init__(
    learning_rate,
    decay=0.9,
    momentum=0.0,
    epsilon=1e-10,
    use_locking=False,
    centered=False,
    name='RMSProp'
)
```

tf.train.AdamOptimizer

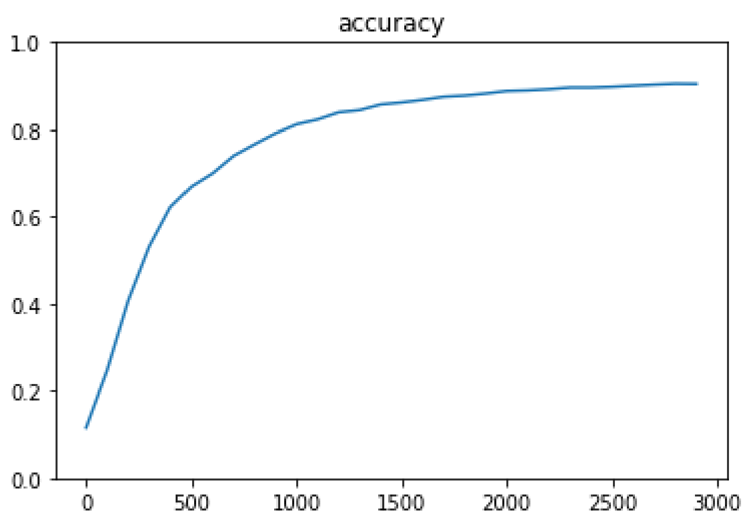
```
__init__(
    learning_rate=0.001,
    beta1=0.9,
    beta2=0.999,
    epsilon=1e-08,
    use_locking=False,
    name='Adam'
)
```

In [21]:

```
import tensorflow as tf
import numpy as np
from tensorflow.examples.tutorials.mnist import input_data
import matplotlib.pyplot as plt

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

iters_num = 3000
batch_size = 100
plot_interval = 100

hidden_layer_size_1 = 600
hidden_layer_size_2 = 300

dropout_rate = 0.5

x = tf.placeholder(tf.float32, [None, 784])
d = tf.placeholder(tf.float32, [None, 10])
W1 = tf.Variable(tf.random_normal([784, hidden_layer_size_1], stddev=0.01))
W2 = tf.Variable(tf.random_normal([hidden_layer_size_1, hidden_layer_size_2], stddev=0.01))
W3 = tf.Variable(tf.random_normal([hidden_layer_size_2, 10], stddev=0.01))

b1 = tf.Variable(tf.zeros([hidden_layer_size_1]))
b2 = tf.Variable(tf.zeros([hidden_layer_size_2]))
b3 = tf.Variable(tf.zeros([10]))

z1 = tf.sigmoid(tf.matmul(x, W1) + b1)
z2 = tf.sigmoid(tf.matmul(z1, W2) + b2)

keep_prob = tf.placeholder(tf.float32)
drop = tf.nn.dropout(z2, keep_prob)

y = tf.nn.softmax(tf.matmul(drop, W3) + b3)
loss = tf.reduce_mean(-tf.reduce_sum(d * tf.log(y), reduction_indices=[1]))


optimizer = tf.train.AdamOptimizer(1e-4)

train = optimizer.minimize(loss)
correct = tf.equal(tf.argmax(y, 1), tf.argmax(d, 1))
accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

accuracies = []
for i in range(iters_num):
    x_batch, d_batch = mnist.train.next_batch(batch_size)
    sess.run(train, feed_dict={x:x_batch, d:d_batch, keep_prob:(1 - dropout_rate)})
    if (i+1) % plot_interval == 0:
        accuracy_val = sess.run(accuracy, feed_dict={x:mnist.test.images, d:mnist.test.labels, k
eep_prob:1.0})
        accuracies.append(accuracy_val)
        print('Generation: ' + str(i+1) + '. 正解率 = ' + str(accuracy_val))

lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies)
plt.title("accuracy")
```

```
plt.ylim(0, 1.0)
plt.show()
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
WARNING:tensorflow:From <ipython-input-21-7595c106b6f4>:31: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed in
a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_p
rob`.
Generation: 100. 正解率 = 0.116
Generation: 200. 正解率 = 0.2471
Generation: 300. 正解率 = 0.4066
Generation: 400. 正解率 = 0.5313
Generation: 500. 正解率 = 0.6221
Generation: 600. 正解率 = 0.6691
Generation: 700. 正解率 = 0.6994
Generation: 800. 正解率 = 0.7396
Generation: 900. 正解率 = 0.7661
Generation: 1000. 正解率 = 0.7913
Generation: 1100. 正解率 = 0.8118
Generation: 1200. 正解率 = 0.8228
Generation: 1300. 正解率 = 0.839
Generation: 1400. 正解率 = 0.8441
Generation: 1500. 正解率 = 0.857
Generation: 1600. 正解率 = 0.8615
Generation: 1700. 正解率 = 0.8675
Generation: 1800. 正解率 = 0.8746
Generation: 1900. 正解率 = 0.8773
Generation: 2000. 正解率 = 0.882
Generation: 2100. 正解率 = 0.8878
Generation: 2200. 正解率 = 0.8892
Generation: 2300. 正解率 = 0.8919
Generation: 2400. 正解率 = 0.8958
Generation: 2500. 正解率 = 0.8958
Generation: 2600. 正解率 = 0.8975
Generation: 2700. 正解率 = 0.9001
Generation: 2800. 正解率 = 0.9025
Generation: 2900. 正解率 = 0.9045
Generation: 3000. 正解率 = 0.9041
```



[try]

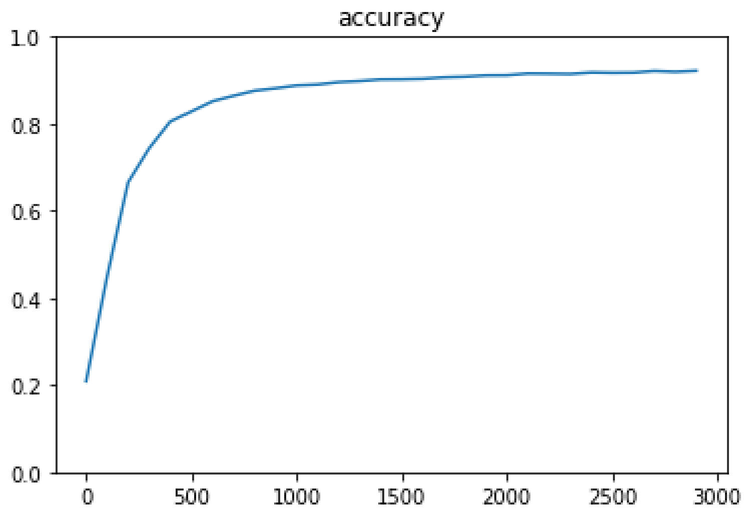
- 隠れ層のサイズを変更してみよう

In [22]:

```
import tensorflow as tf
import numpy as np
from tensorflow.examples.tutorials.mnist import input_data
import matplotlib.pyplot as plt

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

iters_num = 3000
batch_size = 100
plot_interval = 100

hidden_layer_size_1 = 1500
hidden_layer_size_2 = 1000

dropout_rate = 0.5

x = tf.placeholder(tf.float32, [None, 784])
d = tf.placeholder(tf.float32, [None, 10])
W1 = tf.Variable(tf.random_normal([784, hidden_layer_size_1], stddev=0.01))
W2 = tf.Variable(tf.random_normal([hidden_layer_size_1, hidden_layer_size_2], stddev=0.01))
W3 = tf.Variable(tf.random_normal([hidden_layer_size_2, 10], stddev=0.01))

b1 = tf.Variable(tf.zeros([hidden_layer_size_1]))
b2 = tf.Variable(tf.zeros([hidden_layer_size_2]))
b3 = tf.Variable(tf.zeros([10]))

z1 = tf.sigmoid(tf.matmul(x, W1) + b1)
z2 = tf.sigmoid(tf.matmul(z1, W2) + b2)

keep_prob = tf.placeholder(tf.float32)
drop = tf.nn.dropout(z2, keep_prob)

y = tf.nn.softmax(tf.matmul(drop, W3) + b3)
loss = tf.reduce_mean(-tf.reduce_sum(d * tf.log(y), reduction_indices=[1]))


optimizer = tf.train.AdamOptimizer(1e-4)

train = optimizer.minimize(loss)
correct = tf.equal(tf.argmax(y, 1), tf.argmax(d, 1))
accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

accuracies = []
for i in range(iters_num):
    x_batch, d_batch = mnist.train.next_batch(batch_size)
    sess.run(train, feed_dict={x:x_batch, d:d_batch, keep_prob:(1 - dropout_rate)})
    if (i+1) % plot_interval == 0:
        accuracy_val = sess.run(accuracy, feed_dict={x:mnist.test.images, d:mnist.test.labels, k
eep_prob:1.0})
        accuracies.append(accuracy_val)
        print('Generation: ' + str(i+1) + '. 正解率 = ' + str(accuracy_val))

lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies)
plt.title("accuracy")
```

```
plt.ylim(0, 1.0)
plt.show()
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
Generation: 100. 正解率 = 0.209
Generation: 200. 正解率 = 0.4502
Generation: 300. 正解率 = 0.6653
Generation: 400. 正解率 = 0.7434
Generation: 500. 正解率 = 0.8048
Generation: 600. 正解率 = 0.8284
Generation: 700. 正解率 = 0.8517
Generation: 800. 正解率 = 0.8642
Generation: 900. 正解率 = 0.8759
Generation: 1000. 正解率 = 0.8814
Generation: 1100. 正解率 = 0.888
Generation: 1200. 正解率 = 0.8901
Generation: 1300. 正解率 = 0.8956
Generation: 1400. 正解率 = 0.8981
Generation: 1500. 正解率 = 0.9018
Generation: 1600. 正解率 = 0.9021
Generation: 1700. 正解率 = 0.9032
Generation: 1800. 正解率 = 0.9066
Generation: 1900. 正解率 = 0.9081
Generation: 2000. 正解率 = 0.9111
Generation: 2100. 正解率 = 0.9114
Generation: 2200. 正解率 = 0.9154
Generation: 2300. 正解率 = 0.915
Generation: 2400. 正解率 = 0.9143
Generation: 2500. 正解率 = 0.9178
Generation: 2600. 正解率 = 0.917
Generation: 2700. 正解率 = 0.9174
Generation: 2800. 正解率 = 0.9213
Generation: 2900. 正解率 = 0.9193
Generation: 3000. 正解率 = 0.9218
```



[try]

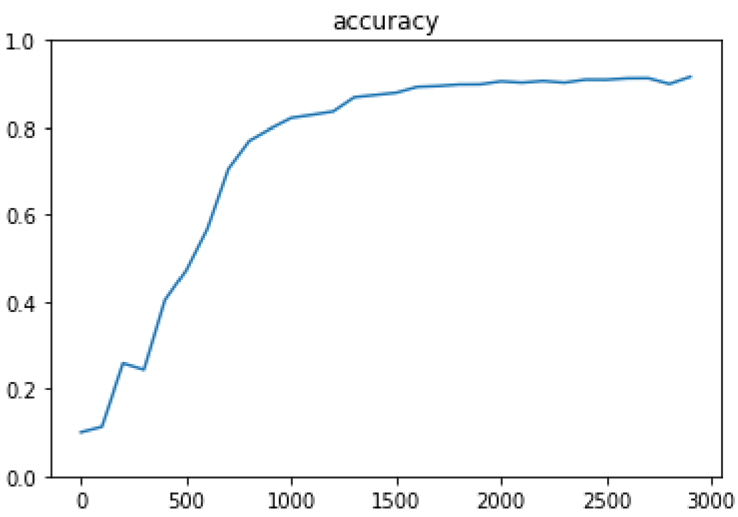
- optimizerを変更しよう
  (AdamをGDに変更)

In [23]:

```
import tensorflow as tf
import numpy as np
from tensorflow.examples.tutorials.mnist import input_data
import matplotlib.pyplot as plt

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

iters_num = 3000
batch_size = 100
plot_interval = 100

hidden_layer_size_1 = 600
hidden_layer_size_2 = 300

dropout_rate = 0.5

x = tf.placeholder(tf.float32, [None, 784])
d = tf.placeholder(tf.float32, [None, 10])
W1 = tf.Variable(tf.random_normal([784, hidden_layer_size_1], stddev=0.01))
W2 = tf.Variable(tf.random_normal([hidden_layer_size_1, hidden_layer_size_2], stddev=0.01))
W3 = tf.Variable(tf.random_normal([hidden_layer_size_2, 10], stddev=0.01))

b1 = tf.Variable(tf.zeros([hidden_layer_size_1]))
b2 = tf.Variable(tf.zeros([hidden_layer_size_2]))
b3 = tf.Variable(tf.zeros([10]))

z1 = tf.sigmoid(tf.matmul(x, W1) + b1)
z2 = tf.sigmoid(tf.matmul(z1, W2) + b2)

keep_prob = tf.placeholder(tf.float32)
drop = tf.nn.dropout(z2, keep_prob)

y = tf.nn.softmax(tf.matmul(drop, W3) + b3)
loss = tf.reduce_mean(-tf.reduce_sum(d * tf.log(y), reduction_indices=[1]))


optimizer = tf.train.GradientDescentOptimizer(0.5)


train = optimizer.minimize(loss)
correct = tf.equal(tf.argmax(y, 1), tf.argmax(d, 1))
accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

accuracies = []
for i in range(iters_num):
    x_batch, d_batch = mnist.train.next_batch(batch_size)
    sess.run(train, feed_dict={x:x_batch, d:d_batch, keep_prob:(1 - dropout_rate)})
    if (i+1) % plot_interval == 0:
        accuracy_val = sess.run(accuracy, feed_dict={x:mnist.test.images, d:mnist.test.labels, k
eep_prob:1.0})
        accuracies.append(accuracy_val)
        print('Generation: ' + str(i+1) + '. 正解率 = ' + str(accuracy_val))

lists = range(0, iters_num, plot_interval)
```

```
plt.plot(lists, accuracies)
plt.title("accuracy")
plt.ylim(0, 1.0)
plt.show()
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
Generation: 100. 正解率 = 0.1009
Generation: 200. 正解率 = 0.1135
Generation: 300. 正解率 = 0.2584
Generation: 400. 正解率 = 0.2434
Generation: 500. 正解率 = 0.4038
Generation: 600. 正解率 = 0.4722
Generation: 700. 正解率 = 0.5675
Generation: 800. 正解率 = 0.705
Generation: 900. 正解率 = 0.7695
Generation: 1000. 正解率 = 0.797
Generation: 1100. 正解率 = 0.8216
Generation: 1200. 正解率 = 0.8287
Generation: 1300. 正解率 = 0.8364
Generation: 1400. 正解率 = 0.8689
Generation: 1500. 正解率 = 0.8741
Generation: 1600. 正解率 = 0.8792
Generation: 1700. 正解率 = 0.8927
Generation: 1800. 正解率 = 0.8947
Generation: 1900. 正解率 = 0.8982
Generation: 2000. 正解率 = 0.8985
Generation: 2100. 正解率 = 0.9055
Generation: 2200. 正解率 = 0.9025
Generation: 2300. 正解率 = 0.9063
Generation: 2400. 正解率 = 0.9025
Generation: 2500. 正解率 = 0.9094
Generation: 2600. 正解率 = 0.9092
Generation: 2700. 正解率 = 0.9124
Generation: 2800. 正解率 = 0.9127
Generation: 2900. 正解率 = 0.8995
Generation: 3000. 正解率 = 0.9159
```



[try]

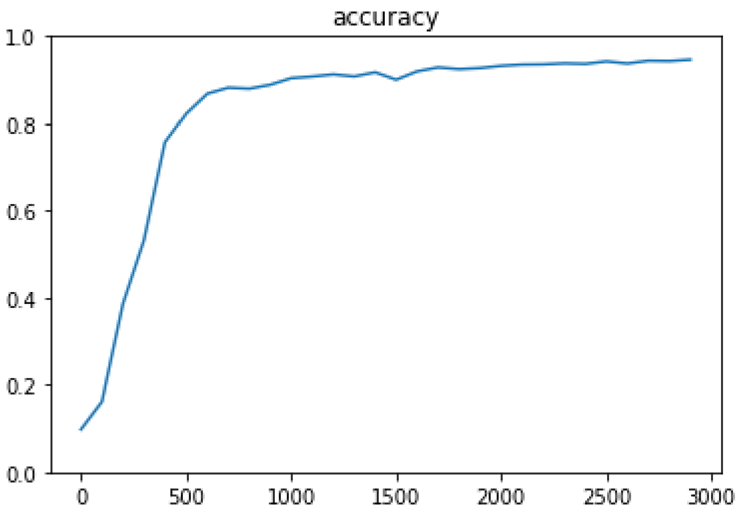
- optimizerを変更しよう
  (AdamをMomentumに変更)

In [24]:

```
import tensorflow as tf
import numpy as np
from tensorflow.examples.tutorials.mnist import input_data
import matplotlib.pyplot as plt

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

iters_num = 3000
batch_size = 100
plot_interval = 100

hidden_layer_size_1 = 600
hidden_layer_size_2 = 300

dropout_rate = 0.5

x = tf.placeholder(tf.float32, [None, 784])
d = tf.placeholder(tf.float32, [None, 10])
W1 = tf.Variable(tf.random_normal([784, hidden_layer_size_1], stddev=0.01))
W2 = tf.Variable(tf.random_normal([hidden_layer_size_1, hidden_layer_size_2], stddev=0.01))
W3 = tf.Variable(tf.random_normal([hidden_layer_size_2, 10], stddev=0.01))

b1 = tf.Variable(tf.zeros([hidden_layer_size_1]))
b2 = tf.Variable(tf.zeros([hidden_layer_size_2]))
b3 = tf.Variable(tf.zeros([10]))

z1 = tf.sigmoid(tf.matmul(x, W1) + b1)
z2 = tf.sigmoid(tf.matmul(z1, W2) + b2)

keep_prob = tf.placeholder(tf.float32)
drop = tf.nn.dropout(z2, keep_prob)

y = tf.nn.softmax(tf.matmul(drop, W3) + b3)
loss = tf.reduce_mean(-tf.reduce_sum(d * tf.log(y), reduction_indices=[1]))


optimizer = tf.train.MomentumOptimizer(0.1, 0.9)



train = optimizer.minimize(loss)
correct = tf.equal(tf.argmax(y, 1), tf.argmax(d, 1))
accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

accuracies = []
for i in range(iters_num):
    x_batch, d_batch = mnist.train.next_batch(batch_size)
    sess.run(train, feed_dict={x:x_batch, d:d_batch, keep_prob:(1 - dropout_rate)})
    if (i+1) % plot_interval == 0:
        accuracy_val = sess.run(accuracy, feed_dict={x:mnist.test.images, d:mnist.test.labels, k
eep_prob:1.0})
        accuracies.append(accuracy_val)
        print('Generation: ' + str(i+1) + '. 正解率 = ' + str(accuracy_val))

lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies)
```

```
plt.title("accuracy")
plt.ylim(0, 1.0)
plt.show()
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
Generation: 100. 正解率 = 0.0982
Generation: 200. 正解率 = 0.1619
Generation: 300. 正解率 = 0.3859
Generation: 400. 正解率 = 0.532
Generation: 500. 正解率 = 0.7573
Generation: 600. 正解率 = 0.8224
Generation: 700. 正解率 = 0.868
Generation: 800. 正解率 = 0.8817
Generation: 900. 正解率 = 0.8794
Generation: 1000. 正解率 = 0.8884
Generation: 1100. 正解率 = 0.9036
Generation: 1200. 正解率 = 0.9073
Generation: 1300. 正解率 = 0.9122
Generation: 1400. 正解率 = 0.9075
Generation: 1500. 正解率 = 0.917
Generation: 1600. 正解率 = 0.9
Generation: 1700. 正解率 = 0.9192
Generation: 1800. 正解率 = 0.9283
Generation: 1900. 正解率 = 0.9243
Generation: 2000. 正解率 = 0.9268
Generation: 2100. 正解率 = 0.9319
Generation: 2200. 正解率 = 0.9348
Generation: 2300. 正解率 = 0.9353
Generation: 2400. 正解率 = 0.9376
Generation: 2500. 正解率 = 0.9365
Generation: 2600. 正解率 = 0.9421
Generation: 2700. 正解率 = 0.9373
Generation: 2800. 正解率 = 0.9436
Generation: 2900. 正解率 = 0.9429
Generation: 3000. 正解率 = 0.9461
```



[try]

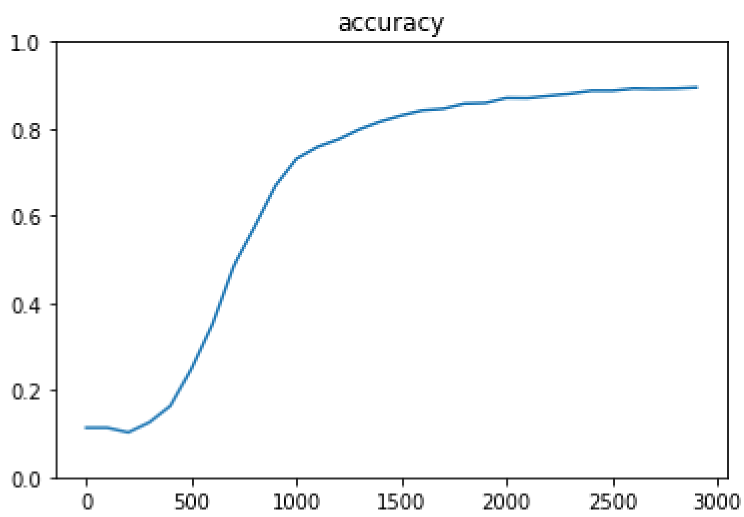
- optimizerを変更しよう
  (AdamをAdagradに変更)

In [25]:

```
import tensorflow as tf
import numpy as np
from tensorflow.examples.tutorials.mnist import input_data
import matplotlib.pyplot as plt

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

iters_num = 3000
batch_size = 100
plot_interval = 100

hidden_layer_size_1 = 600
hidden_layer_size_2 = 300

dropout_rate = 0.5

x = tf.placeholder(tf.float32, [None, 784])
d = tf.placeholder(tf.float32, [None, 10])
W1 = tf.Variable(tf.random_normal([784, hidden_layer_size_1], stddev=0.01))
W2 = tf.Variable(tf.random_normal([hidden_layer_size_1, hidden_layer_size_2], stddev=0.01))
W3 = tf.Variable(tf.random_normal([hidden_layer_size_2, 10], stddev=0.01))

b1 = tf.Variable(tf.zeros([hidden_layer_size_1]))
b2 = tf.Variable(tf.zeros([hidden_layer_size_2]))
b3 = tf.Variable(tf.zeros([10]))

z1 = tf.sigmoid(tf.matmul(x, W1) + b1)
z2 = tf.sigmoid(tf.matmul(z1, W2) + b2)

keep_prob = tf.placeholder(tf.float32)
drop = tf.nn.dropout(z2, keep_prob)

y = tf.nn.softmax(tf.matmul(drop, W3) + b3)
loss = tf.reduce_mean(-tf.reduce_sum(d * tf.log(y), reduction_indices=[1]))


optimizer = tf.train.AdagradOptimizer(0.1)



train = optimizer.minimize(loss)
correct = tf.equal(tf.argmax(y, 1), tf.argmax(d, 1))
accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

accuracies = []
for i in range(iters_num):
    x_batch, d_batch = mnist.train.next_batch(batch_size)
    sess.run(train, feed_dict={x:x_batch, d:d_batch, keep_prob:(1 - dropout_rate)})
    if (i+1) % plot_interval == 0:
        accuracy_val = sess.run(accuracy, feed_dict={x:mnist.test.images, d:mnist.test.labels, k
eep_prob:1.0})
        accuracies.append(accuracy_val)
        print('Generation: ' + str(i+1) + '. 正解率 = ' + str(accuracy_val))

lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies)
```

```
plt.title("accuracy")
plt.ylim(0, 1.0)
plt.show()
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
Generation: 100. 正解率 = 0.1135
Generation: 200. 正解率 = 0.1135
Generation: 300. 正解率 = 0.1028
Generation: 400. 正解率 = 0.1259
Generation: 500. 正解率 = 0.1637
Generation: 600. 正解率 = 0.2493
Generation: 700. 正解率 = 0.3515
Generation: 800. 正解率 = 0.4838
Generation: 900. 正解率 = 0.5739
Generation: 1000. 正解率 = 0.669
Generation: 1100. 正解率 = 0.731
Generation: 1200. 正解率 = 0.7581
Generation: 1300. 正解率 = 0.7756
Generation: 1400. 正解率 = 0.7986
Generation: 1500. 正解率 = 0.8166
Generation: 1600. 正解率 = 0.8304
Generation: 1700. 正解率 = 0.842
Generation: 1800. 正解率 = 0.8459
Generation: 1900. 正解率 = 0.8577
Generation: 2000. 正解率 = 0.8592
Generation: 2100. 正解率 = 0.8709
Generation: 2200. 正解率 = 0.8705
Generation: 2300. 正解率 = 0.8755
Generation: 2400. 正解率 = 0.8805
Generation: 2500. 正解率 = 0.8874
Generation: 2600. 正解率 = 0.8874
Generation: 2700. 正解率 = 0.8925
Generation: 2800. 正解率 = 0.8916
Generation: 2900. 正解率 = 0.8925
Generation: 3000. 正解率 = 0.8949
```



[try]

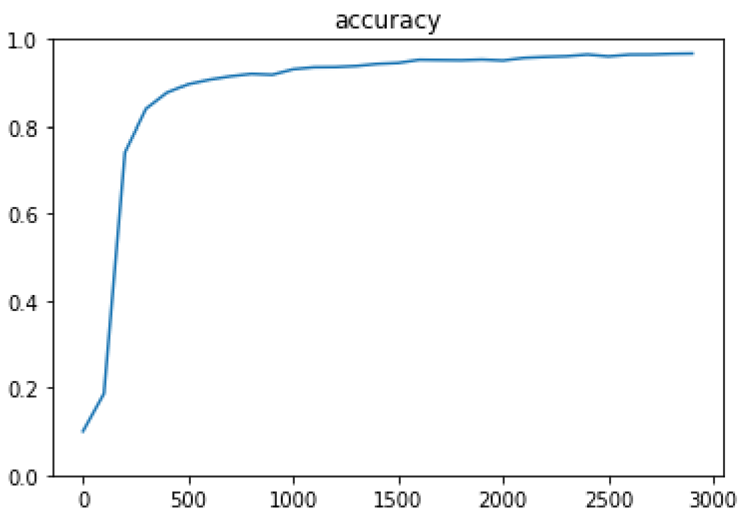
- optimizerを変更しよう
  (AdamをRMSPropに変更)

In [26]:

```
import tensorflow as tf
import numpy as np
from tensorflow.examples.tutorials.mnist import input_data
import matplotlib.pyplot as plt

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

iters_num = 3000
batch_size = 100
plot_interval = 100

hidden_layer_size_1 = 600
hidden_layer_size_2 = 300

dropout_rate = 0.5

x = tf.placeholder(tf.float32, [None, 784])
d = tf.placeholder(tf.float32, [None, 10])
W1 = tf.Variable(tf.random_normal([784, hidden_layer_size_1], stddev=0.01))
W2 = tf.Variable(tf.random_normal([hidden_layer_size_1, hidden_layer_size_2], stddev=0.01))
W3 = tf.Variable(tf.random_normal([hidden_layer_size_2, 10], stddev=0.01))

b1 = tf.Variable(tf.zeros([hidden_layer_size_1]))
b2 = tf.Variable(tf.zeros([hidden_layer_size_2]))
b3 = tf.Variable(tf.zeros([10]))

z1 = tf.sigmoid(tf.matmul(x, W1) + b1)
z2 = tf.sigmoid(tf.matmul(z1, W2) + b2)

keep_prob = tf.placeholder(tf.float32)
drop = tf.nn.dropout(z2, keep_prob)

y = tf.nn.softmax(tf.matmul(drop, W3) + b3)
loss = tf.reduce_mean(-tf.reduce_sum(d * tf.log(y), reduction_indices=[1]))


optimizer = tf.train.RMSPropOptimizer(0.001)


train = optimizer.minimize(loss)
correct = tf.equal(tf.argmax(y, 1), tf.argmax(d, 1))
accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

accuracies = []
for i in range(iters_num):
    x_batch, d_batch = mnist.train.next_batch(batch_size)
    sess.run(train, feed_dict={x:x_batch, d:d_batch, keep_prob:(1 - dropout_rate)})
    if (i+1) % plot_interval == 0:
        accuracy_val = sess.run(accuracy, feed_dict={x:mnist.test.images, d:mnist.test.labels, k
eep_prob:1.0})
        accuracies.append(accuracy_val)
        print('Generation: ' + str(i+1) + '. 正解率 = ' + str(accuracy_val))

lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies)
```

```
plt.title("accuracy")
plt.ylim(0, 1.0)
plt.show()
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
Generation: 100. 正解率 = 0.1009
Generation: 200. 正解率 = 0.1876
Generation: 300. 正解率 = 0.7408
Generation: 400. 正解率 = 0.8401
Generation: 500. 正解率 = 0.8773
Generation: 600. 正解率 = 0.8965
Generation: 700. 正解率 = 0.9069
Generation: 800. 正解率 = 0.9147
Generation: 900. 正解率 = 0.9203
Generation: 1000. 正解率 = 0.9183
Generation: 1100. 正解率 = 0.931
Generation: 1200. 正解率 = 0.9356
Generation: 1300. 正解率 = 0.936
Generation: 1400. 正解率 = 0.9384
Generation: 1500. 正解率 = 0.9432
Generation: 1600. 正解率 = 0.9454
Generation: 1700. 正解率 = 0.9527
Generation: 1800. 正解率 = 0.9521
Generation: 1900. 正解率 = 0.9516
Generation: 2000. 正解率 = 0.9532
Generation: 2100. 正解率 = 0.951
Generation: 2200. 正解率 = 0.9571
Generation: 2300. 正解率 = 0.9593
Generation: 2400. 正解率 = 0.9609
Generation: 2500. 正解率 = 0.9648
Generation: 2600. 正解率 = 0.9605
Generation: 2700. 正解率 = 0.9646
Generation: 2800. 正解率 = 0.9648
Generation: 2900. 正解率 = 0.9664
Generation: 3000. 正解率 = 0.9671
```



[try]

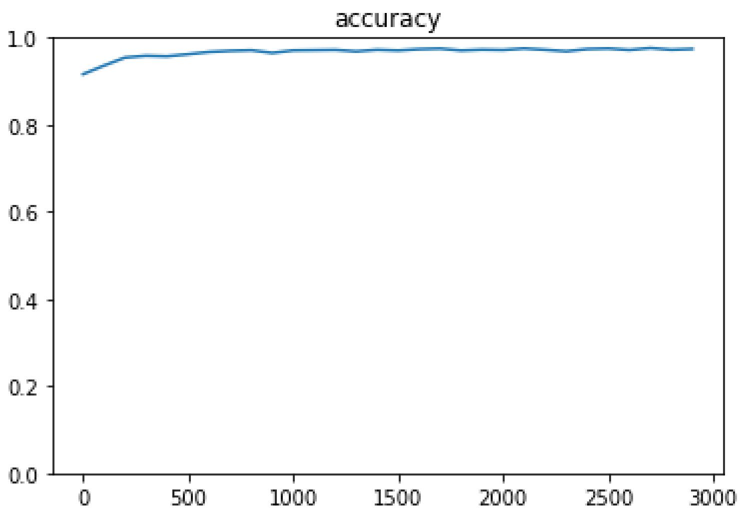
- optimizerを変更しよう
  (Adamをに変更)

In [28]:

```
import tensorflow as tf
import numpy as np
from tensorflow.examples.tutorials.mnist import input_data
import matplotlib.pyplot as plt

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

iters_num = 3000
batch_size = 100
plot_interval = 100

hidden_layer_size_1 = 600
hidden_layer_size_2 = 300

dropout_rate = 0.5

x = tf.placeholder(tf.float32, [None, 784])
d = tf.placeholder(tf.float32, [None, 10])
W1 = tf.Variable(tf.random_normal([784, hidden_layer_size_1], stddev=0.01))
W2 = tf.Variable(tf.random_normal([hidden_layer_size_1, hidden_layer_size_2], stddev=0.01))
W3 = tf.Variable(tf.random_normal([hidden_layer_size_2, 10], stddev=0.01))

b1 = tf.Variable(tf.zeros([hidden_layer_size_1]))
b2 = tf.Variable(tf.zeros([hidden_layer_size_2]))
b3 = tf.Variable(tf.zeros([10]))

z1 = tf.sigmoid(tf.matmul(x, W1) + b1)
z2 = tf.sigmoid(tf.matmul(z1, W2) + b2)

keep_prob = tf.placeholder(tf.float32)
drop = tf.nn.dropout(z2, keep_prob)

y = tf.nn.softmax(tf.matmul(drop, W3) + b3)
loss = tf.reduce_mean(-tf.reduce_sum(d * tf.log(y), reduction_indices=[1]))


optimizer = tf.train.AdamOptimizer(1e-2)

train = optimizer.minimize(loss)
correct = tf.equal(tf.argmax(y, 1), tf.argmax(d, 1))
accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

accuracies = []
for i in range(iters_num):
    x_batch, d_batch = mnist.train.next_batch(batch_size)
    sess.run(train, feed_dict={x:x_batch, d:d_batch, keep_prob:(1 - dropout_rate)})
    if (i+1) % plot_interval == 0:
        accuracy_val = sess.run(accuracy, feed_dict={x:mnist.test.images, d:mnist.test.labels, k
eep_prob:1.0})
        accuracies.append(accuracy_val)
        print('Generation: ' + str(i+1) + '. 正解率 = ' + str(accuracy_val))

lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies)
plt.title("accuracy")
```

```
plt.ylim(0, 1.0)
plt.show()
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
Generation: 100. 正解率 = 0.9161
Generation: 200. 正解率 = 0.9359
Generation: 300. 正解率 = 0.9544
Generation: 400. 正解率 = 0.9585
Generation: 500. 正解率 = 0.9572
Generation: 600. 正解率 = 0.9621
Generation: 700. 正解率 = 0.9673
Generation: 800. 正解率 = 0.9698
Generation: 900. 正解率 = 0.9712
Generation: 1000. 正解率 = 0.965
Generation: 1100. 正解率 = 0.9708
Generation: 1200. 正解率 = 0.9714
Generation: 1300. 正解率 = 0.972
Generation: 1400. 正解率 = 0.9686
Generation: 1500. 正解率 = 0.9723
Generation: 1600. 正解率 = 0.9706
Generation: 1700. 正解率 = 0.9737
Generation: 1800. 正解率 = 0.975
Generation: 1900. 正解率 = 0.9706
Generation: 2000. 正解率 = 0.9726
Generation: 2100. 正解率 = 0.9718
Generation: 2200. 正解率 = 0.9753
Generation: 2300. 正解率 = 0.9723
Generation: 2400. 正解率 = 0.9688
Generation: 2500. 正解率 = 0.974
Generation: 2600. 正解率 = 0.9752
Generation: 2700. 正解率 = 0.9718
Generation: 2800. 正解率 = 0.9763
Generation: 2900. 正解率 = 0.9723
Generation: 3000. 正解率 = 0.9742
```



分類CNN (mnist)

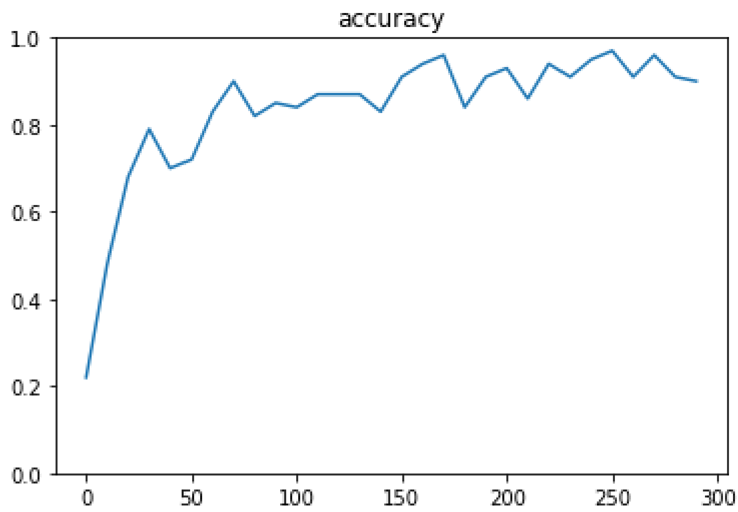
conv - relu - pool - conv - relu - pool -
affin - relu - dropout - affin - softmax

In [29]:

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
import matplotlib.pyplot as plt

iters_num = 300
batch_size = 100
plot_interval = 10

dropout_rate = 0.5

# placeholder
x = tf.placeholder(tf.float32, shape=[None, 784])
d = tf.placeholder(tf.float32, shape=[None, 10])

# 画像を784の一次元から28x28の二次元に変換する
# 画像を28x28にreshape
x_image = tf.reshape(x, [-1,28,28,1])#画像n個、28行、28列、1チャンネル

# 第一層のweightsとbiasのvariable
W_conv1 = tf.Variable(tf.truncated_normal([5, 5, 1, 32], stddev=0.1))#5行5列のフィルタ、1チャン
ネルを32チャンネルに拡張、標準偏差0.01
b_conv1 = tf.Variable(tf.constant(0.1, shape=[32]))

# 第一層のconvolutionalとpool
# strides[0] = strides[3] = 1固定
h_conv1 = tf.nn.relu(tf.nn.conv2d(x_image, W_conv1, strides=[1, 1, 1, 1], padding='SAME') + b_co
nv1)
# プーリングサイズ n*n にしたい場合 ksize=[1, n, n, 1]
h_pool1 = tf.nn.max_pool(h_conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

# 第二層
W_conv2 = tf.Variable(tf.truncated_normal([5, 5, 32, 64], stddev=0.1))
b_conv2 = tf.Variable(tf.constant(0.1, shape=[64]))
h_conv2 = tf.nn.relu(tf.nn.conv2d(h_pool1, W_conv2, strides=[1, 1, 1, 1], padding='SAME') + b_co
nv2)
h_pool2 = tf.nn.max_pool(h_conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

# 第一層と第二層でreduceされてできた特徴に対してrelu
W_fc1 = tf.Variable(tf.truncated_normal([7 * 7 * 64, 1024], stddev=0.1))
b_fc1 = tf.Variable(tf.constant(0.1, shape=[1024]))
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

# Dropout
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

# 出来上がったものに対してSoftmax
W_fc2 = tf.Variable(tf.truncated_normal([1024, 10], stddev=0.1))
b_fc2 = tf.Variable(tf.constant(0.1, shape=[10]))
y_conv=tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)

# 交差エントロピー
loss = -tf.reduce_sum(d * tf.log(y_conv))

train = tf.train.AdamOptimizer(1e-4).minimize(loss)

correct = tf.equal(tf.argmax(y_conv,1), tf.argmax(d,1))
```

```
accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)


accuracies = []
for i in range(iters_num):
    x_batch, d_batch = mnist.train.next_batch(batch_size)
    sess.run(train, feed_dict={x: x_batch, d: d_batch, keep_prob: 1-dropout_rate})
    if (i+1) % plot_interval == 0:
        accuracy_val = sess.run(accuracy, feed_dict={x:x_batch, d: d_batch, keep_prob: 1.0})
        accuracies.append(accuracy_val)
        print('Generation: ' + str(i+1) + '. 正解率 = ' + str(accuracy_val))


lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies)
plt.title("accuracy")
plt.ylim(0, 1.0)
plt.show()
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
Generation: 10. 正解率 = 0.22
Generation: 20. 正解率 = 0.48
Generation: 30. 正解率 = 0.68
Generation: 40. 正解率 = 0.79
Generation: 50. 正解率 = 0.7
Generation: 60. 正解率 = 0.72
Generation: 70. 正解率 = 0.83
Generation: 80. 正解率 = 0.9
Generation: 90. 正解率 = 0.82
Generation: 100. 正解率 = 0.85
Generation: 110. 正解率 = 0.84
Generation: 120. 正解率 = 0.87
Generation: 130. 正解率 = 0.87
Generation: 140. 正解率 = 0.87
Generation: 150. 正解率 = 0.83
Generation: 160. 正解率 = 0.91
Generation: 170. 正解率 = 0.94
Generation: 180. 正解率 = 0.96
Generation: 190. 正解率 = 0.84
Generation: 200. 正解率 = 0.91
Generation: 210. 正解率 = 0.93
Generation: 220. 正解率 = 0.86
Generation: 230. 正解率 = 0.94
Generation: 240. 正解率 = 0.91
Generation: 250. 正解率 = 0.95
Generation: 260. 正解率 = 0.97
Generation: 270. 正解率 = 0.91
Generation: 280. 正解率 = 0.96
Generation: 290. 正解率 = 0.91
Generation: 300. 正解率 = 0.9
```



[try]

- ドロップアウト率を0に変更しよう

In [30]:

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
import matplotlib.pyplot as plt

iters_num = 300
batch_size = 100
plot_interval = 10

dropout_rate = 0

# placeholder
x = tf.placeholder(tf.float32, shape=[None, 784])
d = tf.placeholder(tf.float32, shape=[None, 10])

# 画像を784の一次元から28x28の二次元に変換する
# 画像を28x28にreshape
x_image = tf.reshape(x, [-1,28,28,1])

# 第一層のweightsとbiasのvariable
W_conv1 = tf.Variable(tf.truncated_normal([5, 5, 1, 32], stddev=0.1))
b_conv1 = tf.Variable(tf.constant(0.1, shape=[32]))

# 第一層のconvolutionalとpool
# strides[0] = strides[3] = 1固定
h_conv1 = tf.nn.relu(tf.nn.conv2d(x_image, W_conv1, strides=[1, 1, 1, 1], padding='SAME') + b_co
nv1)
# プーリングサイズ n*n にしたい場合 ksize=[1, n, n, 1]
h_pool1 = tf.nn.max_pool(h_conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

# 第二層
W_conv2 = tf.Variable(tf.truncated_normal([5, 5, 32, 64], stddev=0.1))
b_conv2 = tf.Variable(tf.constant(0.1, shape=[64]))
h_conv2 = tf.nn.relu(tf.nn.conv2d(h_pool1, W_conv2, strides=[1, 1, 1, 1], padding='SAME') + b_co
nv2)
h_pool2 = tf.nn.max_pool(h_conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

# 第一層と第二層でreduceされてできた特徴に対してrelu
W_fc1 = tf.Variable(tf.truncated_normal([7 * 7 * 64, 1024], stddev=0.1))
b_fc1 = tf.Variable(tf.constant(0.1, shape=[1024]))
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

# Dropout
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

# 出来上がったものに対してSoftmax
W_fc2 = tf.Variable(tf.truncated_normal([1024, 10], stddev=0.1))
b_fc2 = tf.Variable(tf.constant(0.1, shape=[10]))
y_conv=tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)

# 交差エントロピー
loss = -tf.reduce_sum(d * tf.log(y_conv))

train = tf.train.AdamOptimizer(1e-4).minimize(loss)

correct = tf.equal(tf.argmax(y_conv,1), tf.argmax(d,1))
accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
```

```
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)


accuracies = []
for i in range(iters_num):
    x_batch, d_batch = mnist.train.next_batch(batch_size)
    sess.run(train, feed_dict={x: x_batch, d: d_batch, keep_prob: 1-dropout_rate})
    if (i+1) % plot_interval == 0:
        accuracy_val = sess.run(accuracy, feed_dict={x:x_batch, d: d_batch, keep_prob: 1.0})
        accuracies.append(accuracy_val)
        print('Generation: ' + str(i+1) + '. 正解率 = ' + str(accuracy_val))


lists = range(0, iters_num, plot_interval)
plt.plot(lists, accuracies)
plt.title("accuracy")
plt.ylim(0, 1.0)
plt.show()
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
Generation: 10. 正解率 = 0.21
Generation: 20. 正解率 = 0.47
Generation: 30. 正解率 = 0.65
Generation: 40. 正解率 = 0.75
Generation: 50. 正解率 = 0.73
Generation: 60. 正解率 = 0.78
Generation: 70. 正解率 = 0.91
Generation: 80. 正解率 = 0.83
Generation: 90. 正解率 = 0.85
Generation: 100. 正解率 = 0.91
Generation: 110. 正解率 = 0.9
Generation: 120. 正解率 = 0.89
Generation: 130. 正解率 = 0.84
Generation: 140. 正解率 = 0.87
Generation: 150. 正解率 = 0.87
Generation: 160. 正解率 = 0.9
Generation: 170. 正解率 = 0.92
Generation: 180. 正解率 = 0.93
Generation: 190. 正解率 = 0.94
Generation: 200. 正解率 = 0.92
Generation: 210. 正解率 = 0.96
Generation: 220. 正解率 = 0.92
Generation: 230. 正解率 = 0.94
Generation: 240. 正解率 = 0.93
Generation: 250. 正解率 = 0.97
Generation: 260. 正解率 = 0.93
Generation: 270. 正解率 = 0.95
Generation: 280. 正解率 = 0.93
Generation: 290. 正解率 = 0.94
Generation: 300. 正解率 = 0.94
```