keras

線形回帰

In [1]:

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

iters_num = 1000#反復回数
plot_interval = 10#プロット間隔

x = np.linspace(-1, 1, 200)#-1～1から200分割。
np.random.shuffle(x)
d = 0.5 * x + 2 + np.random.normal(0, 0.05, (200,))

print(np.random.normal(0, 0.05, (200,)))#平均０、分散0.05、1次元200個。
print(d)

from keras.models import Sequential
from keras.layers import Dense

# モデルを作成
model = Sequential()
model.add(Dense(input_dim=1, output_dim=1))

# モデルを表示
model.summary()

# モデルのコンパイル
model.compile(loss='mse', optimizer='sgd')

# train
for i in range(iters_num):
    loss = model.train_on_batch(x, d)#
    if (i+1) % plot_interval == 0:
        print('Generation: ' + str(i+1) + '. 誤差 = ' + str(loss))

W, b = model.layers[0].get_weights()
print('W:', W)
print('b:', b)

y = model.predict(x)
plt.scatter(x, d)
plt.plot(x, y)
plt.show()
```

```
[ 2.74582959e-02 -2.59902196e-02  8.03121476e-03  1.16871107e-02
  2.61793303e-02  2.40209717e-02 -2.92994679e-02  1.39201791e-02
 -4.06900567e-02 -4.23980158e-02 -5.59945535e-02  2.91311878e-02
 -6.57230159e-03  5.39182158e-02  5.40927172e-02  9.11772426e-02
 -1.37467356e-02 -1.14806884e-02 -2.04841173e-03  1.08382942e-01
 -1.65313202e-03  4.13683227e-02  4.03971896e-02  7.94713241e-02
  4.76520736e-02 -2.45205910e-02  4.00503645e-02  2.91786464e-02
  5.33477307e-04 -7.05164740e-02 -7.86384280e-03  1.83292092e-02
  1.06183716e-01 -3.90834942e-02  6.53641075e-02  7.19245268e-02
 -8.66352481e-03  1.95581181e-02 -6.84497329e-02  2.02163469e-02
  4.92708332e-02 -1.30728401e-02 -5.51643461e-02 -4.09490126e-02
  1.81201074e-02 -1.66512208e-02  6.46269243e-02  4.37734796e-03
  7.07448205e-03 -8.66609883e-03  6.12995233e-03  2.42403871e-02
 -9.03065661e-02  2.86492069e-03 -6.79688598e-02  2.69220108e-02
  3.57107206e-02  3.53044915e-02 -1.04071405e-02 -2.63118621e-02
 -4.60258495e-02  3.04718846e-02  1.11400328e-02  6.80411942e-03
 -5.88019870e-02  3.42633617e-02 -1.15251464e-02 -2.58210426e-02
  1.29440244e-01  5.88807870e-02 -6.30432451e-03  2.61059230e-02
  8.73797448e-02  3.61887811e-04 -7.05487716e-02 -3.16342808e-02
 -1.50080296e-02  1.63841515e-02 -6.56267980e-02  1.15837917e-03
 -2.76806876e-02 -3.10524001e-03 -9.70242568e-03  4.29986636e-02
  8.29811494e-03 -4.83503458e-02 -3.64957379e-02 -3.00321876e-03
  5.35412800e-02 -1.20688359e-01 -6.66251817e-02 -6.53711945e-02
  1.92513462e-02  6.66284918e-02 -3.95403001e-03  3.64051424e-02
  4.02826011e-02 -3.41909795e-03 -1.06738669e-01 -2.36150044e-02
 -1.79734443e-02  3.79141750e-02  8.85510227e-02 -1.51708169e-03
  6.20991903e-02  6.36762849e-02  5.90828086e-05 -1.74274897e-02
 -1.87922510e-02  6.68996314e-02  1.12520719e-03  8.86648765e-03
 -5.57743751e-02 -2.51558557e-02 -3.44356265e-02  3.70439762e-02
  4.67184839e-02  2.20196418e-02 -1.41232563e-02  2.30765697e-02
  5.88344326e-02  3.29383080e-03  3.31250921e-02 -1.47070972e-03
  1.83119019e-02 -4.21161797e-02 -6.53591448e-02  6.25903282e-02
 -8.51434517e-03  2.88409872e-02  6.02836019e-02  8.65146453e-02
  2.60153442e-02  6.90323006e-02  6.67500758e-02  2.51127532e-02
  2.72762365e-02  7.11635557e-02  8.32599495e-02 -1.93890006e-02
 -3.95515719e-04 -1.60099493e-02 -7.59224431e-02  8.54285802e-02
 -6.31692244e-02 -4.80328272e-02  3.46274509e-02  1.12639228e-01
 -6.54194775e-03  3.27542160e-02  7.75049582e-02  4.47308985e-02
  5.03580052e-02  2.22009740e-02  8.33502884e-02 -4.88934254e-02
 -3.36472711e-02  2.10786829e-03 -7.29651871e-02  4.51914047e-02
 -4.89439557e-03  1.36283212e-01  6.81675128e-02  6.26660025e-02
 -8.92424069e-02  3.16130570e-02  8.91553899e-02 -1.50062103e-02
  7.51009683e-02 -5.27354384e-02  2.32224949e-02 -4.46301161e-02
 -1.09249241e-03 -9.74192679e-04  1.99795266e-03  4.75534696e-03
 -6.15072309e-02  1.26778903e-02 -4.60284318e-02 -3.54910844e-02
 -8.33929829e-03  7.49576401e-02  2.20555500e-02  7.54870104e-02
 -3.44552272e-02  6.03033019e-02  7.23057291e-02  4.82969468e-02
  5.75928611e-02 -2.84443072e-02  5.96645818e-02  4.49886774e-02
  4.89499419e-02 -6.79031008e-02  1.73997752e-02  1.05359362e-04
  1.77036394e-02  7.03695011e-02  3.00076223e-02  6.29943592e-02]
[1.84698029 1.653061   1.91934707 2.51566554 1.56701867 2.45003466
 2.37414452 2.38078078 1.77219702 2.21047988 2.29306696 1.63228752
 1.59187923 2.00599606 1.85310005 2.44896867 1.79596913 2.37979021
 1.93762172 1.98076057 1.65992764 2.21780334 1.55328872 1.76795833
 2.37757644 1.57339687 2.34953927 1.63449054 2.22336754 1.84194785
 1.42991122 2.09588994 1.70582866 1.71792063 2.08604287 2.31489486
 2.47883792 1.92536756 2.33366923 1.84913544 1.81104634 1.75617137
 1.90942348 2.23236308 2.32696857 2.18175242 2.21953108 1.69203416
 2.03693985 2.14198389 2.10466762 2.14185957 1.85432803 1.65842552
 2.18709437 2.28336114 1.83579146 1.71097885 2.15903092 1.92236089
 1.84821878 2.15275321 2.26281351 1.55702578 1.68491853 1.7922993
```

```
 1.89429196 2.24339772 1.70542861 2.35242181 2.10148813 1.86864602
 2.46561293 1.904909   1.82087535 1.94821016 1.49911267 1.72247035
 1.60802099 2.48362672 2.5176658  2.00720803 1.69239627 2.38752466
 1.66822582 1.88467408 1.55023022 1.89739755 1.71597765 2.22076262
 1.526569   2.44034959 2.0283354  2.39345441 1.63600829 2.36910278
 2.20689589 1.93628121 1.46484208 2.06541603 2.06227165 2.25550262
 1.90462492 2.00105092 1.9335377  2.36331072 1.89442383 1.63106961
 2.19447047 2.20752221 1.86630999 1.69978921 2.02614409 1.82563999
 2.2655599  2.23348887 1.94244205 2.22931325 2.07127407 2.29068628
 2.46080837 2.14722618 1.92160295 1.72717305 1.52460948 2.18385061
 2.15032191 1.82270166 1.83009271 1.58292025 1.75773536 2.09302143
 2.07082687 2.49953119 2.02417148 2.0330701  1.90097931 2.34269692
 2.05709443 1.87142149 2.52569183 2.14208368 1.91627397 2.55116002
 1.57263444 1.68944886 1.98589008 1.56484611 1.98271422 2.11493514
 2.23846232 1.64926241 1.74737737 2.35869172 1.99308498 1.70265516
 1.83435847 2.37170884 1.52809954 1.80884694 2.31056468 1.71431977
 2.05854354 1.8427576  1.74605056 2.55299693 2.31733259 2.44925887
 2.49293919 1.96803049 2.12038746 1.46805904 1.70320275 2.2510622
 2.44466259 1.49760184 2.10003174 2.29100013 1.62094249 2.29332088
 1.62208692 1.62509794 2.27161021 2.14698138 2.15652861 2.25319203
 1.99610561 1.80724813 2.38944873 1.63906826 2.34057848 1.738482
 2.48834418 2.09094113 1.59341701 1.5488722  2.01832974 2.05020165
 1.70758475 1.7128832 ]
```

```
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
Using TensorFlow backend.
```

```
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.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1)                 2
=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
WARNING:tensorflow:From C:¥ProgramData¥Anaconda3¥lib¥site-packages¥tensorflow¥pyth
on¥ops¥math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is depreca
ted and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
```

```
C:¥ProgramData¥Anaconda3¥lib¥site-packages¥ipykernel_launcher.py:19: UserWarning:
Update your `Dense` call to the Keras 2 API: `Dense(input_dim=1, units=1)`
```

```
Generation: 10. 誤差 = 3.2394366
Generation: 20. 誤差 = 2.2602508
Generation: 30. 誤差 = 1.5943204
Generation: 40. 誤差 = 1.1390663
Generation: 50. 誤差 = 0.8258107
Generation: 60. 誤差 = 0.6085329
Generation: 70. 誤差 = 0.4563601
Generation: 80. 誤差 = 0.34855172
Generation: 90. 誤差 = 0.271147
Generation: 100. 誤差 = 0.21472636
Generation: 110. 誤差 = 0.17291468
Generation: 120. 誤差 = 0.1413799
Generation: 130. 誤差 = 0.11716375
Generation: 140. 誤差 = 0.09823335
Generation: 150. 誤差 = 0.08318112
Generation: 160. 誤差 = 0.07102316
Generation: 170. 誤差 = 0.061063938
Generation: 180. 誤差 = 0.05280554
Generation: 190. 誤差 = 0.045886025
Generation: 200. 誤差 = 0.040038113
Generation: 210. 誤差 = 0.03506087
Generation: 220. 誤差 = 0.030800536
Generation: 230. 誤差 = 0.027137313
Generation: 240. 誤差 = 0.02397624
Generation: 250. 誤差 = 0.021240836
Generation: 260. 誤差 = 0.018868623
Generation: 270. 誤差 = 0.016807888
Generation: 280. 誤差 = 0.015015402
Generation: 290. 誤差 = 0.013454668
Generation: 300. 誤差 = 0.012094669
Generation: 310. 誤差 = 0.010908886
Generation: 320. 誤差 = 0.009874528
Generation: 330. 誤差 = 0.008971933
Generation: 340. 誤差 = 0.008184114
Generation: 350. 誤差 = 0.007496334
Generation: 360. 誤差 = 0.0068957927
Generation: 370. 誤差 = 0.0063713635
Generation: 380. 誤差 = 0.0059133563
Generation: 390. 誤差 = 0.005513331
Generation: 400. 誤差 = 0.005163927
Generation: 410. 誤差 = 0.00485873
Generation: 420. 誤差 = 0.0045921346
Generation: 430. 誤差 = 0.0043592546
Generation: 440. 誤差 = 0.004155821
Generation: 450. 誤差 = 0.0039781076
Generation: 460. 誤差 = 0.0038228633
Generation: 470. 誤差 = 0.0036872465
Generation: 480. 誤差 = 0.0035687725
Generation: 490. 誤差 = 0.0034652739
Generation: 500. 誤差 = 0.0033748618
Generation: 510. 誤差 = 0.003295877
Generation: 520. 誤差 = 0.003226874
Generation: 530. 誤差 = 0.0031665934
Generation: 540. 誤差 = 0.0031139327
Generation: 550. 誤差 = 0.003067927
Generation: 560. 誤差 = 0.0030277378
Generation: 570. 誤差 = 0.0029926277
Generation: 580. 誤差 = 0.0029619571
Generation: 590. 誤差 = 0.0029351625
Generation: 600. 誤差 = 0.0029117537
Generation: 610. 誤差 = 0.0028913051
```

```
Generation: 620. 誤差 = 0.002873441
Generation: 630. 誤差 = 0.0028578339
Generation: 640. 誤差 = 0.0028442005
Generation: 650. 誤差 = 0.0028322893
Generation: 660. 誤差 = 0.0028218837
Generation: 670. 誤差 = 0.002812794
Generation: 680. 誤差 = 0.0028048537
Generation: 690. 誤差 = 0.0027979163
Generation: 700. 誤差 = 0.002791855
Generation: 710. 誤差 = 0.002786561
Generation: 720. 誤差 = 0.0027819353
Generation: 730. 誤差 = 0.002777894
Generation: 740. 誤差 = 0.0027743643
Generation: 750. 誤差 = 0.0027712805
Generation: 760. 誤差 = 0.0027685866
Generation: 770. 誤差 = 0.0027662325
Generation: 780. 誤差 = 0.002764177
Generation: 790. 誤差 = 0.0027623817
Generation: 800. 誤差 = 0.0027608126
Generation: 810. 誤差 = 0.0027594424
Generation: 820. 誤差 = 0.0027582436
Generation: 830. 誤差 = 0.0027571982
Generation: 840. 誤差 = 0.0027562839
Generation: 850. 誤差 = 0.0027554855
Generation: 860. 誤差 = 0.0027547872
Generation: 870. 誤差 = 0.0027541781
Generation: 880. 誤差 = 0.0027536466
Generation: 890. 誤差 = 0.0027531805
Generation: 900. 誤差 = 0.0027527746
Generation: 910. 誤差 = 0.0027524198
Generation: 920. 誤差 = 0.0027521094
Generation: 930. 誤差 = 0.002751839
Generation: 940. 誤差 = 0.0027516028
Generation: 950. 誤差 = 0.0027513953
Generation: 960. 誤差 = 0.0027512156
Generation: 970. 誤差 = 0.0027510584
Generation: 980. 誤差 = 0.0027509194
Generation: 990. 誤差 = 0.002750799
Generation: 1000. 誤差 = 0.002750694
W: [[0.4935466]]
b: [1.9952797]

<Figure size 640x480 with 1 Axes>
```

単純パーセプトロン

OR回路

In [2]:

```
# モジュール読み込み
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD

# 乱数を固定値で初期化
np.random.seed(0)

# シグモイドの単純パーセプトロン作成
model = Sequential()
model.add(Dense(input_dim=2, units=1))
model.add(Activation('sigmoid'))
model.summary()

model.compile(loss='binary_crossentropy', optimizer=SGD(lr=0.1))

# トレーニング用入力 X と正解データ T
X = np.array( [[0,0], [0,1], [1,0], [1,1]] )
T = np.array( [[0], [1], [1], [1]] )

# トレーニング
model.fit(X, T, epochs=30, batch_size=1)#バッチサイズが1のSGD。

# トレーニングの入力を流用して実際に分類
Y = model.predict_classes(X, batch_size=1)

print("TEST")
print(Y == T)

#重み2個、バイアス1個。トータルパラメータ3個。
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 1)                 3
_________________________________________________________________
activation_1 (Activation)    (None, 1)                 0
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
Epoch 1/30
4/4 [==============================] - 0s 50ms/step - loss: 0.4352
Epoch 2/30
4/4 [==============================] - 0s 2ms/step - loss: 0.4204
Epoch 3/30
4/4 [==============================] - 0s 2ms/step - loss: 0.4079
Epoch 4/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3971
Epoch 5/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3876
Epoch 6/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3790
Epoch 7/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3717
Epoch 8/30
4/4 [==============================] - 0s 4ms/step - loss: 0.3650
Epoch 9/30
4/4 [==============================] - 0s 3ms/step - loss: 0.3586
Epoch 10/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3528
Epoch 11/30
4/4 [==============================] - 0s 1ms/step - loss: 0.3476
Epoch 12/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3425
Epoch 13/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3378
Epoch 14/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3333
Epoch 15/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3291
Epoch 16/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3250
Epoch 17/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3210
Epoch 18/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3172
Epoch 19/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3136
Epoch 20/30
4/4 [==============================] - 0s 3ms/step - loss: 0.3100
Epoch 21/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3067
Epoch 22/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3032
Epoch 23/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3000
Epoch 24/30
4/4 [==============================] - ETA: 0s - loss: 0.129 - 0s 2ms/step - loss:
0.2968
Epoch 25/30
```

```
4/4 [==============================] - 0s 2ms/step - loss: 0.2938
Epoch 26/30
4/4 [==============================] - 0s 2ms/step - loss: 0.2908
Epoch 27/30
4/4 [==============================] - ETA: 0s - loss: 0.267 - 0s 2ms/step - loss:
0.2878
Epoch 28/30
4/4 [==============================] - 0s 2ms/step - loss: 0.2850
Epoch 29/30
4/4 [==============================] - 0s 2ms/step - loss: 0.2821
Epoch 30/30
4/4 [==============================] - 0s 3ms/step - loss: 0.2794
TEST
[[ True]
 [ True]
 [ True]
 [ True]]
```

[try]

- np.random.seed(0)をnp.random.seed(1)に変更

In [3]:

```
# モジュール読み込み
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD

# 乱数を固定値で初期化
np.random.seed(1)

# シグモイドの単純パーセプトロン作成
model = Sequential()
model.add(Dense(input_dim=2, units=1))
model.add(Activation('sigmoid'))
model.summary()

model.compile(loss='binary_crossentropy', optimizer=SGD(lr=0.1))

# トレーニング用入力 X と正解データ T
X = np.array( [[0,0], [0,1], [1,0], [1,1]] )
T = np.array( [[0], [1], [1], [1]] )

# トレーニング
model.fit(X, T, epochs=30, batch_size=1)

# トレーニングの入力を流用して実際に分類
Y = model.predict_classes(X, batch_size=1)

print("TEST")
print(Y == T)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_3 (Dense)              (None, 1)                 3         
_________________________________________________________________
activation_2 (Activation)    (None, 1)                 0         
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
Epoch 1/30
4/4 [==============================] - 0s 46ms/step - loss: 0.4976
Epoch 2/30
4/4 [==============================] - 0s 2ms/step - loss: 0.4734
Epoch 3/30
4/4 [==============================] - 0s 2ms/step - loss: 0.4528
Epoch 4/30
4/4 [==============================] - 0s 2ms/step - loss: 0.4364
Epoch 5/30
4/4 [==============================] - 0s 2ms/step - loss: 0.4231
Epoch 6/30
4/4 [==============================] - 0s 2ms/step - loss: 0.4115
Epoch 7/30
4/4 [==============================] - 0s 3ms/step - loss: 0.4016
Epoch 8/30
4/4 [==============================] - 0s 3ms/step - loss: 0.3928
Epoch 9/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3852
Epoch 10/30
4/4 [==============================] - 0s 1ms/step - loss: 0.3781
Epoch 11/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3719
Epoch 12/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3661
Epoch 13/30
4/4 [==============================] - 0s 1ms/step - loss: 0.3605
Epoch 14/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3555
Epoch 15/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3506
Epoch 16/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3460
Epoch 17/30
4/4 [==============================] - 0s 1ms/step - loss: 0.3417
Epoch 18/30
4/4 [==============================] - 0s 3ms/step - loss: 0.3375
Epoch 19/30
4/4 [==============================] - 0s 3ms/step - loss: 0.3334
Epoch 20/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3294
Epoch 21/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3257
Epoch 22/30
4/4 [==============================] - 0s 3ms/step - loss: 0.3220
Epoch 23/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3186
Epoch 24/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3151
Epoch 25/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3117
```

```
Epoch 26/30
4/4 [==============================] - 0s 1ms/step - loss: 0.3085
Epoch 27/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3053
Epoch 28/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3022
Epoch 29/30
4/4 [==============================] - 0s 2ms/step - loss: 0.2991
Epoch 30/30
4/4 [==============================] - 0s 2ms/step - loss: 0.2961
TEST
[[False]
 [ True]
 [ True]
 [ True]]
```

[try]

- エポック数を100に変更

In [4]:

```
# モジュール読み込み
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD

# 乱数を固定値で初期化
np.random.seed(0)

# シグモイドの単純パーセプトロン作成
model = Sequential()
model.add(Dense(input_dim=2, units=1))
model.add(Activation('sigmoid'))
model.summary()

model.compile(loss='binary_crossentropy', optimizer=SGD(lr=0.1))

# トレーニング用入力 X と正解データ T
X = np.array( [[0,0], [0,1], [1,0], [1,1]] )
T = np.array( [[0], [1], [1], [1]] )

# トレーニング
model.fit(X, T, epochs=100, batch_size=1)

# トレーニングの入力を流用して実際に分類
Y = model.predict_classes(X, batch_size=1)

print("TEST")
print(Y == T)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_4 (Dense)              (None, 1)                 3         
_________________________________________________________________
activation_3 (Activation)    (None, 1)                 0         
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
4/4 [==============================] - 0s 51ms/step - loss: 0.4352
Epoch 2/100
4/4 [==============================] - 0s 2ms/step - loss: 0.4204
Epoch 3/100
4/4 [==============================] - 0s 2ms/step - loss: 0.4079
Epoch 4/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3971
Epoch 5/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3876
Epoch 6/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3790
Epoch 7/100
4/4 [==============================] - 0s 3ms/step - loss: 0.3717
Epoch 8/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3650
Epoch 9/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3586
Epoch 10/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3528
Epoch 11/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3476
Epoch 12/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3425
Epoch 13/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3378
Epoch 14/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3333
Epoch 15/100
4/4 [==============================] - 0s 1ms/step - loss: 0.3291
Epoch 16/100
4/4 [==============================] - 0s 3ms/step - loss: 0.3250
Epoch 17/100
4/4 [==============================] - 0s 5ms/step - loss: 0.3210
Epoch 18/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3172
Epoch 19/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3136
Epoch 20/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3100
Epoch 21/100
4/4 [==============================] - 0s 2ms/step - loss: 0.3067
Epoch 22/100
4/4 [==============================] - 0s 3ms/step - loss: 0.3032
Epoch 23/100
4/4 [==============================] - 0s 3ms/step - loss: 0.3000
Epoch 24/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2968
Epoch 25/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2938
```

```
Epoch 26/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2908
Epoch 27/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2878
Epoch 28/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2850
Epoch 29/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2821
Epoch 30/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2794
Epoch 31/100
4/4 [==============================] - 0s 1ms/step - loss: 0.2767
Epoch 32/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2740
Epoch 33/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2714
Epoch 34/100
4/4 [==============================] - 0s 4ms/step - loss: 0.2688
Epoch 35/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2663
Epoch 36/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2638
Epoch 37/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2614
Epoch 38/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2590
Epoch 39/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2567
Epoch 40/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2544
Epoch 41/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2522
Epoch 42/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2499
Epoch 43/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2477
Epoch 44/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2456
Epoch 45/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2435
Epoch 46/100
4/4 [==============================] - 0s 4ms/step - loss: 0.2414
Epoch 47/100
4/4 [==============================] - 0s 3ms/step - loss: 0.2393
Epoch 48/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2373
Epoch 49/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2353
Epoch 50/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2334
Epoch 51/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2315
Epoch 52/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2296
Epoch 53/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2277
Epoch 54/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2259
Epoch 55/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2241
Epoch 56/100
```

```
4/4 [==============================] - 0s 2ms/step - loss: 0.2223
Epoch 57/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2206
Epoch 58/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2188
Epoch 59/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2171
Epoch 60/100
4/4 [==============================] - 0s 4ms/step - loss: 0.2155
Epoch 61/100
4/4 [==============================] - 0s 3ms/step - loss: 0.2138
Epoch 62/100
4/4 [==============================] - 0s 3ms/step - loss: 0.2122
Epoch 63/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2106
Epoch 64/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2090
Epoch 65/100
4/4 [==============================] - 0s 3ms/step - loss: 0.2075
Epoch 66/100
4/4 [==============================] - 0s 1ms/step - loss: 0.2059
Epoch 67/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2044
Epoch 68/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2029
Epoch 69/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2014
Epoch 70/100
4/4 [==============================] - 0s 2ms/step - loss: 0.2000
Epoch 71/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1985
Epoch 72/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1971
Epoch 73/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1957
Epoch 74/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1943
Epoch 75/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1930
Epoch 76/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1917
Epoch 77/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1903
Epoch 78/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1890
Epoch 79/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1877
Epoch 80/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1865
Epoch 81/100
4/4 [==============================] - 0s 3ms/step - loss: 0.1852
Epoch 82/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1839
Epoch 83/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1827
Epoch 84/100
4/4 [==============================] - 0s 1ms/step - loss: 0.1815
Epoch 85/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1803
Epoch 86/100
4/4 [==============================] - 0s 5ms/step - loss: 0.1791
```

```
Epoch 87/100
4/4 [==============================] - 0s 3ms/step - loss: 0.1780
Epoch 88/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1768
Epoch 89/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1757
Epoch 90/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1746
Epoch 91/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1735
Epoch 92/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1724
Epoch 93/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1713
Epoch 94/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1702
Epoch 95/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1692
Epoch 96/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1681
Epoch 97/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1671
Epoch 98/100
4/4 [==============================] - 0s 5ms/step - loss: 0.1660
Epoch 99/100
4/4 [==============================] - 0s 3ms/step - loss: 0.1650
Epoch 100/100
4/4 [==============================] - 0s 2ms/step - loss: 0.1640
TEST
[[ True]
 [ True]
 [ True]
 [ True]]
```

[try]

- AND回路に変更

In [5]:

```
# モジュール読み込み
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD

# 乱数を固定値で初期化
np.random.seed(0)

# シグモイドの単純パーセプトロン作成
model = Sequential()
model.add(Dense(input_dim=2, units=1))
model.add(Activation('sigmoid'))
model.summary()

model.compile(loss='binary_crossentropy', optimizer=SGD(lr=0.1))

# トレーニング用入力 X と正解データ T
X = np.array( [[0,0], [0,1], [1,0], [1,1]] )
T = np.array( [[0], [0], [0], [1]] )

# トレーニング
model.fit(X, T, epochs=30, batch_size=1)

# トレーニングの入力を流用して実際に分類
Y = model.predict_classes(X, batch_size=1)

print("TEST")
print(Y == T)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 1)                 3
_________________________________________________________________
activation_4 (Activation)    (None, 1)                 0
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
Epoch 1/30
4/4 [==============================] - 0s 57ms/step - loss: 0.8084
Epoch 2/30
4/4 [==============================] - 0s 2ms/step - loss: 0.7406
Epoch 3/30
4/4 [==============================] - 0s 3ms/step - loss: 0.6849
Epoch 4/30
4/4 [==============================] - 0s 3ms/step - loss: 0.6409
Epoch 5/30
4/4 [==============================] - 0s 3ms/step - loss: 0.6056
Epoch 6/30
4/4 [==============================] - 0s 2ms/step - loss: 0.5781
Epoch 7/30
4/4 [==============================] - 0s 2ms/step - loss: 0.5573
Epoch 8/30
4/4 [==============================] - 0s 2ms/step - loss: 0.5409
Epoch 9/30
4/4 [==============================] - 0s 2ms/step - loss: 0.5270
Epoch 10/30
4/4 [==============================] - 0s 2ms/step - loss: 0.5151
Epoch 11/30
4/4 [==============================] - 0s 3ms/step - loss: 0.5051
Epoch 12/30
4/4 [==============================] - 0s 3ms/step - loss: 0.4965
Epoch 13/30
4/4 [==============================] - 0s 1ms/step - loss: 0.4889
Epoch 14/30
4/4 [==============================] - 0s 2ms/step - loss: 0.4818
Epoch 15/30
4/4 [==============================] - 0s 2ms/step - loss: 0.4751
Epoch 16/30
4/4 [==============================] - 0s 2ms/step - loss: 0.4682
Epoch 17/30
4/4 [==============================] - 0s 2ms/step - loss: 0.4623
Epoch 18/30
4/4 [==============================] - 0s 2ms/step - loss: 0.4565
Epoch 19/30
4/4 [==============================] - 0s 2ms/step - loss: 0.4512
Epoch 20/30
4/4 [==============================] - 0s 2ms/step - loss: 0.4459
Epoch 21/30
4/4 [==============================] - 0s 2ms/step - loss: 0.4403
Epoch 22/30
4/4 [==============================] - 0s 2ms/step - loss: 0.4354
Epoch 23/30
4/4 [==============================] - 0s 4ms/step - loss: 0.4309
Epoch 24/30
4/4 [==============================] - 0s 3ms/step - loss: 0.4264
Epoch 25/30
4/4 [==============================] - 0s 2ms/step - loss: 0.4219
```

```
Epoch 26/30
4/4 [==============================] - 0s 2ms/step - loss: 0.4176
Epoch 27/30
4/4 [==============================] - 0s 2ms/step - loss: 0.4132
Epoch 28/30
4/4 [==============================] - 0s 2ms/step - loss: 0.4090
Epoch 29/30
4/4 [==============================] - 0s 2ms/step - loss: 0.4046
Epoch 30/30
4/4 [==============================] - 0s 2ms/step - loss: 0.4011
TEST
[[ True]
 [ True]
 [ True]
 [ True]]
```

[try]

- XOR回路に変更

In [6]:

```
# モジュール読み込み
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD

# 乱数を固定値で初期化
np.random.seed(0)

# シグモイドの単純パーセプトロン作成
model = Sequential()
model.add(Dense(input_dim=2, units=1))
model.add(Activation('sigmoid'))
model.summary()

model.compile(loss='binary_crossentropy', optimizer=SGD(lr=0.1))

# トレーニング用入力 X と正解データ T
X = np.array( [[0,0], [0,1], [1,0], [1,1]] )
T = np.array( [[0], [1], [1], [0]] )

# トレーニング
model.fit(X, T, epochs=30, batch_size=1)

# トレーニングの入力を流用して実際に分類
Y = model.predict_classes(X, batch_size=1)

print("TEST")
print(Y == T)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_6 (Dense)              (None, 1)                 3         
_________________________________________________________________
activation_5 (Activation)    (None, 1)                 0         
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
Epoch 1/30
4/4 [==============================] - 0s 64ms/step - loss: 0.8427
Epoch 2/30
4/4 [==============================] - 0s 2ms/step - loss: 0.8205
Epoch 3/30
4/4 [==============================] - 0s 2ms/step - loss: 0.8049
Epoch 4/30
4/4 [==============================] - 0s 2ms/step - loss: 0.7924
Epoch 5/30
4/4 [==============================] - 0s 1ms/step - loss: 0.7822
Epoch 6/30
4/4 [==============================] - 0s 2ms/step - loss: 0.7754
Epoch 7/30
4/4 [==============================] - 0s 3ms/step - loss: 0.7688
Epoch 8/30
4/4 [==============================] - 0s 2ms/step - loss: 0.7622
Epoch 9/30
4/4 [==============================] - 0s 1ms/step - loss: 0.7578
Epoch 10/30
4/4 [==============================] - 0s 3ms/step - loss: 0.7555
Epoch 11/30
4/4 [==============================] - 0s 1ms/step - loss: 0.7520
Epoch 12/30
4/4 [==============================] - 0s 2ms/step - loss: 0.7497
Epoch 13/30
4/4 [==============================] - 0s 1ms/step - loss: 0.7468
Epoch 14/30
4/4 [==============================] - 0s 2ms/step - loss: 0.7454
Epoch 15/30
4/4 [==============================] - 0s 2ms/step - loss: 0.7439
Epoch 16/30
4/4 [==============================] - 0s 2ms/step - loss: 0.7425
Epoch 17/30
4/4 [==============================] - 0s 3ms/step - loss: 0.7422
Epoch 18/30
4/4 [==============================] - 0s 4ms/step - loss: 0.7404
Epoch 19/30
4/4 [==============================] - 0s 2ms/step - loss: 0.7394
Epoch 20/30
4/4 [==============================] - 0s 3ms/step - loss: 0.7393
Epoch 21/30
4/4 [==============================] - 0s 2ms/step - loss: 0.7380
Epoch 22/30
4/4 [==============================] - 0s 2ms/step - loss: 0.7368
Epoch 23/30
4/4 [==============================] - 0s 3ms/step - loss: 0.7371
Epoch 24/30
4/4 [==============================] - 0s 2ms/step - loss: 0.7363
Epoch 25/30
4/4 [==============================] - 0s 1ms/step - loss: 0.7356
```

```
Epoch 26/30
4/4 [==============================] - 0s 2ms/step - loss: 0.7348
Epoch 27/30
4/4 [==============================] - 0s 2ms/step - loss: 0.7344
Epoch 28/30
4/4 [==============================] - 0s 2ms/step - loss: 0.7338
Epoch 29/30
4/4 [==============================] - 0s 2ms/step - loss: 0.7328
Epoch 30/30
4/4 [==============================] - 0s 3ms/step - loss: 0.7326
TEST
[[ True]
 [ True]
 [False]
 [False]]
```

[try]

- XOR回路に変更して、epochは100に変更

In [7]:

```
# モジュール読み込み
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD

# 乱数を固定値で初期化
np.random.seed(0)

# シグモイドの単純パーセプトロン作成
model = Sequential()
model.add(Dense(input_dim=2, units=1))
model.add(Activation('sigmoid'))
model.summary()

model.compile(loss='binary_crossentropy', optimizer=SGD(lr=0.1))

# トレーニング用入力 X と正解データ T
X = np.array( [[0,0], [0,1], [1,0], [1,1]] )
T = np.array( [[0], [1], [1], [0]] )

# トレーニング
model.fit(X, T, epochs=100, batch_size=1)

# トレーニングの入力を流用して実際に分類
Y = model.predict_classes(X, batch_size=1)

print("TEST")
print(Y == T)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 1)                 3
_________________________________________________________________
activation_6 (Activation)    (None, 1)                 0
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
Epoch 1/100
4/4 [==============================] - 0s 99ms/step - loss: 0.8427
Epoch 2/100
4/4 [==============================] - 0s 3ms/step - loss: 0.8205
Epoch 3/100
4/4 [==============================] - 0s 2ms/step - loss: 0.8049
Epoch 4/100
4/4 [==============================] - 0s 3ms/step - loss: 0.7924
Epoch 5/100
4/4 [==============================] - 0s 1ms/step - loss: 0.7822
Epoch 6/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7754
Epoch 7/100
4/4 [==============================] - 0s 4ms/step - loss: 0.7688
Epoch 8/100
4/4 [==============================] - 0s 3ms/step - loss: 0.7622
Epoch 9/100
4/4 [==============================] - 0s 3ms/step - loss: 0.7578
Epoch 10/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7555
Epoch 11/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7520
Epoch 12/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7497
Epoch 13/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7468
Epoch 14/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7454
Epoch 15/100
4/4 [==============================] - 0s 3ms/step - loss: 0.7439
Epoch 16/100
4/4 [==============================] - ETA: 0s - loss: 0.887 - 0s 2ms/step - loss:
0.7425
Epoch 17/100
4/4 [==============================] - 0s 3ms/step - loss: 0.7422
Epoch 18/100
4/4 [==============================] - 0s 3ms/step - loss: 0.7404
Epoch 19/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7394
Epoch 20/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7393
Epoch 21/100
4/4 [==============================] - 0s 3ms/step - loss: 0.7380
Epoch 22/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7368
Epoch 23/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7371
Epoch 24/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7363
Epoch 25/100
```

```
4/4 [==============================] - 0s 2ms/step - loss: 0.7356
Epoch 26/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7348
Epoch 27/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7344
Epoch 28/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7338
Epoch 29/100
4/4 [==============================] - 0s 1ms/step - loss: 0.7328
Epoch 30/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7326
Epoch 31/100
4/4 [==============================] - 0s 5ms/step - loss: 0.7319
Epoch 32/100
4/4 [==============================] - 0s 3ms/step - loss: 0.7317
Epoch 33/100
4/4 [==============================] - 0s 3ms/step - loss: 0.7304
Epoch 34/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7306
Epoch 35/100
4/4 [==============================] - 0s 1ms/step - loss: 0.7304
Epoch 36/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7293
Epoch 37/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7295
Epoch 38/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7281
Epoch 39/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7280
Epoch 40/100
4/4 [==============================] - 0s 3ms/step - loss: 0.7282
Epoch 41/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7276
Epoch 42/100
4/4 [==============================] - 0s 3ms/step - loss: 0.7277
Epoch 43/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7274
Epoch 44/100
4/4 [==============================] - 0s 4ms/step - loss: 0.7270
Epoch 45/100
4/4 [==============================] - 0s 4ms/step - loss: 0.7267
Epoch 46/100
4/4 [==============================] - 0s 4ms/step - loss: 0.7262
Epoch 47/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7251
Epoch 48/100
4/4 [==============================] - 0s 3ms/step - loss: 0.7249
Epoch 49/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7247
Epoch 50/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7244
Epoch 51/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7246
Epoch 52/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7250
Epoch 53/100
4/4 [==============================] - 0s 3ms/step - loss: 0.7246
Epoch 54/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7242
Epoch 55/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7241
```

```
Epoch 56/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7242
Epoch 57/100
4/4 [==============================] - 0s 5ms/step - loss: 0.7233
Epoch 58/100
4/4 [==============================] - 0s 3ms/step - loss: 0.7237
Epoch 59/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7235
Epoch 60/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7228
Epoch 61/100
4/4 [==============================] - 0s 3ms/step - loss: 0.7232
Epoch 62/100
4/4 [==============================] - 0s 3ms/step - loss: 0.7231
Epoch 63/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7229
Epoch 64/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7224
Epoch 65/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7225
Epoch 66/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7215
Epoch 67/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7224
Epoch 68/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7219
Epoch 69/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7218
Epoch 70/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7221
Epoch 71/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7220
Epoch 72/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7219
Epoch 73/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7211
Epoch 74/100
4/4 [==============================] - 0s 4ms/step - loss: 0.7210
Epoch 75/100
4/4 [==============================] - 0s 4ms/step - loss: 0.7213
Epoch 76/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7216
Epoch 77/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7205
Epoch 78/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7213
Epoch 79/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7214
Epoch 80/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7213
Epoch 81/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7205
Epoch 82/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7204
Epoch 83/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7210
Epoch 84/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7206
Epoch 85/100
4/4 [==============================] - 0s 3ms/step - loss: 0.7209
Epoch 86/100
```

```
4/4 [==============================] - 0s 2ms/step - loss: 0.7208
Epoch 87/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7206
Epoch 88/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7207
Epoch 89/100
4/4 [==============================] - 0s 1ms/step - loss: 0.7196
Epoch 90/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7206
Epoch 91/100
4/4 [==============================] - 0s 5ms/step - loss: 0.7198
Epoch 92/100
4/4 [==============================] - 0s 3ms/step - loss: 0.7202
Epoch 93/100
4/4 [==============================] - 0s 3ms/step - loss: 0.7203
Epoch 94/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7197
Epoch 95/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7200
Epoch 96/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7196
Epoch 97/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7200
Epoch 98/100
4/4 [==============================] - 0s 3ms/step - loss: 0.7203
Epoch 99/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7203
Epoch 100/100
4/4 [==============================] - 0s 2ms/step - loss: 0.7195
TEST
[[ True]
 [ True]
 [False]
 [False]]
```

[try]

- OR回路にしてバッチサイズを10に変更

In [8]:

```
# モジュール読み込み
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD

# 乱数を固定値で初期化
np.random.seed(0)

# シグモイドの単純パーセプトロン作成
model = Sequential()
model.add(Dense(input_dim=2, units=1))
model.add(Activation('sigmoid'))
model.summary()

model.compile(loss='binary_crossentropy', optimizer=SGD(lr=0.1))

# トレーニング用入力 X と正解データ T
X = np.array( [[0,0], [0,1], [1,0], [1,1]] )
T = np.array( [[0], [1], [1], [1]] )

# トレーニング
model.fit(X, T, epochs=30, batch_size=10)#バッチサイズが最大で10個。

# トレーニングの入力を流用して実際に分類
Y = model.predict_classes(X, batch_size=1)

print("TEST")
print(Y == T)

#paramは重みが２つとバイアスが１つの計３つ。
#Outputは？行1列。バッチかミニバッチかオンラインで行が決まるはず。
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_8 (Dense)              (None, 1)                 3         
_________________________________________________________________
activation_7 (Activation)    (None, 1)                 0         
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
Epoch 1/30
4/4 [==============================] - 0s 76ms/step - loss: 0.4326
Epoch 2/30
4/4 [==============================] - 0s 749us/step - loss: 0.4285
Epoch 3/30
4/4 [==============================] - 0s 496us/step - loss: 0.4246
Epoch 4/30
4/4 [==============================] - 0s 250us/step - loss: 0.4208
Epoch 5/30
4/4 [==============================] - 0s 500us/step - loss: 0.4172
Epoch 6/30
4/4 [==============================] - 0s 250us/step - loss: 0.4138
Epoch 7/30
4/4 [==============================] - 0s 500us/step - loss: 0.4105
Epoch 8/30
4/4 [==============================] - 0s 250us/step - loss: 0.4074
Epoch 9/30
4/4 [==============================] - 0s 500us/step - loss: 0.4044
Epoch 10/30
4/4 [==============================] - 0s 500us/step - loss: 0.4014
Epoch 11/30
4/4 [==============================] - 0s 499us/step - loss: 0.3986
Epoch 12/30
4/4 [==============================] - 0s 324us/step - loss: 0.3959
Epoch 13/30
4/4 [==============================] - 0s 500us/step - loss: 0.3933
Epoch 14/30
4/4 [==============================] - 0s 250us/step - loss: 0.3908
Epoch 15/30
4/4 [==============================] - 0s 500us/step - loss: 0.3884
Epoch 16/30
4/4 [==============================] - 0s 500us/step - loss: 0.3860
Epoch 17/30
4/4 [==============================] - 0s 500us/step - loss: 0.3837
Epoch 18/30
4/4 [==============================] - 0s 250us/step - loss: 0.3815
Epoch 19/30
4/4 [==============================] - 0s 500us/step - loss: 0.3794
Epoch 20/30
4/4 [==============================] - 0s 749us/step - loss: 0.3773
Epoch 21/30
4/4 [==============================] - 0s 752us/step - loss: 0.3753
Epoch 22/30
4/4 [==============================] - 0s 2ms/step - loss: 0.3733
Epoch 23/30
4/4 [==============================] - 0s 750us/step - loss: 0.3714
Epoch 24/30
4/4 [==============================] - 0s 499us/step - loss: 0.3696
Epoch 25/30
4/4 [==============================] - 0s 499us/step - loss: 0.3678
```

```
Epoch 26/30
4/4 [==============================] - 0s 499us/step - loss: 0.3660
Epoch 27/30
4/4 [==============================] - 0s 1ms/step - loss: 0.3643
Epoch 28/30
4/4 [==============================] - 0s 250us/step - loss: 0.3626
Epoch 29/30
4/4 [==============================] - 0s 499us/step - loss: 0.3610
Epoch 30/30
4/4 [==============================] - 0s 499us/step - loss: 0.3594
TEST
[[False]
 [ True]
 [ True]
 [ True]]
```

[try]

- エポック数を300に変更しよう

In [9]:

```
# モジュール読み込み
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD

# 乱数を固定値で初期化
np.random.seed(0)

# シグモイドの単純パーセプトロン作成
model = Sequential()
model.add(Dense(input_dim=2, units=1))
model.add(Activation('sigmoid'))
model.summary()

model.compile(loss='binary_crossentropy', optimizer=SGD(lr=0.1))

# トレーニング用入力 X と正解データ T
X = np.array( [[0,0], [0,1], [1,0], [1,1]] )
T = np.array( [[0], [1], [1], [1]] )

# トレーニング
model.fit(X, T, epochs=300, batch_size=1)

# トレーニングの入力を流用して実際に分類
Y = model.predict_classes(X, batch_size=1)

print("TEST")
print(Y == T)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_9 (Dense)              (None, 1)                 3
_________________________________________________________________
activation_8 (Activation)    (None, 1)                 0
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
Epoch 1/300
4/4 [==============================] - 0s 86ms/step - loss: 0.4352
Epoch 2/300
4/4 [==============================] - 0s 2ms/step - loss: 0.4204
Epoch 3/300
4/4 [==============================] - 0s 2ms/step - loss: 0.4079
Epoch 4/300
4/4 [==============================] - 0s 3ms/step - loss: 0.3971
Epoch 5/300
4/4 [==============================] - 0s 2ms/step - loss: 0.3876
Epoch 6/300
4/4 [==============================] - 0s 2ms/step - loss: 0.3790
Epoch 7/300
4/4 [==============================] - 0s 2ms/step - loss: 0.3717
Epoch 8/300
4/4 [==============================] - 0s 4ms/step - loss: 0.3650
Epoch 9/300
4/4 [==============================] - 0s 3ms/step - loss: 0.3586
Epoch 10/300
4/4 [==============================] - 0s 2ms/step - loss: 0.3528
Epoch 11/300
4/4 [==============================] - 0s 2ms/step - loss: 0.3476
Epoch 12/300
4/4 [==============================] - 0s 2ms/step - loss: 0.3425
Epoch 13/300
4/4 [==============================] - 0s 2ms/step - loss: 0.3378
Epoch 14/300
4/4 [==============================] - 0s 1ms/step - loss: 0.3333
Epoch 15/300
4/4 [==============================] - 0s 2ms/step - loss: 0.3291
Epoch 16/300
4/4 [==============================] - 0s 2ms/step - loss: 0.3250
Epoch 17/300
4/4 [==============================] - 0s 4ms/step - loss: 0.3210
Epoch 18/300
4/4 [==============================] - 0s 2ms/step - loss: 0.3172
Epoch 19/300
4/4 [==============================] - 0s 1ms/step - loss: 0.3136
Epoch 20/300
4/4 [==============================] - 0s 2ms/step - loss: 0.3100
Epoch 21/300
4/4 [==============================] - 0s 1ms/step - loss: 0.3067
Epoch 22/300
4/4 [==============================] - 0s 2ms/step - loss: 0.3032
Epoch 23/300
4/4 [==============================] - 0s 2ms/step - loss: 0.3000
Epoch 24/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2968
Epoch 25/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2938
```

```
Epoch 26/300
4/4 [==============================] - 0s 5ms/step - loss: 0.2908
Epoch 27/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2878
Epoch 28/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2850
Epoch 29/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2821
Epoch 30/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2794
Epoch 31/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2767
Epoch 32/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2740
Epoch 33/300
4/4 [==============================] - ETA: 0s - loss: 0.034 - 0s 1ms/step - loss:
0.2714
Epoch 34/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2688
Epoch 35/300
4/4 [==============================] - 0s 4ms/step - loss: 0.2663
Epoch 36/300
4/4 [==============================] - 0s 3ms/step - loss: 0.2638
Epoch 37/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2614
Epoch 38/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2590
Epoch 39/300
4/4 [==============================] - 0s 3ms/step - loss: 0.2567
Epoch 40/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2544
Epoch 41/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2522
Epoch 42/300
4/4 [==============================] - 0s 3ms/step - loss: 0.2499
Epoch 43/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2477
Epoch 44/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2456
Epoch 45/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2435
Epoch 46/300
4/4 [==============================] - 0s 4ms/step - loss: 0.2414
Epoch 47/300
4/4 [==============================] - 0s 4ms/step - loss: 0.2393
Epoch 48/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2373
Epoch 49/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2353
Epoch 50/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2334
Epoch 51/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2315
Epoch 52/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2296
Epoch 53/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2277
Epoch 54/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2259
Epoch 55/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2241
```

```
Epoch 56/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2223
Epoch 57/300
4/4 [==============================] - 0s 3ms/step - loss: 0.2206
Epoch 58/300
4/4 [==============================] - 0s 5ms/step - loss: 0.2188
Epoch 59/300
4/4 [==============================] - 0s 3ms/step - loss: 0.2171
Epoch 60/300
4/4 [==============================] - 0s 3ms/step - loss: 0.2155
Epoch 61/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2138
Epoch 62/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2122
Epoch 63/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2106
Epoch 64/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2090
Epoch 65/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2075
Epoch 66/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2059
Epoch 67/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2044
Epoch 68/300
4/4 [==============================] - 0s 2ms/step - loss: 0.2029
Epoch 69/300
4/4 [==============================] - 0s 3ms/step - loss: 0.2014
Epoch 70/300
4/4 [==============================] - 0s 3ms/step - loss: 0.2000
Epoch 71/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1985
Epoch 72/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1971
Epoch 73/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1957
Epoch 74/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1943
Epoch 75/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1930
Epoch 76/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1917
Epoch 77/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1903
Epoch 78/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1890
Epoch 79/300
4/4 [==============================] - 0s 4ms/step - loss: 0.1877
Epoch 80/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1865
Epoch 81/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1852
Epoch 82/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1839
Epoch 83/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1827
Epoch 84/300
4/4 [==============================] - 0s 7ms/step - loss: 0.1815
Epoch 85/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1803
Epoch 86/300
```

```
4/4 [==============================] - 0s 2ms/step - loss: 0.1791
Epoch 87/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1780
Epoch 88/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1768
Epoch 89/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1757
Epoch 90/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1746
Epoch 91/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1735
Epoch 92/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1724
Epoch 93/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1713
Epoch 94/300
4/4 [==============================] - 0s 1ms/step - loss: 0.1702
Epoch 95/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1692
Epoch 96/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1681
Epoch 97/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1671
Epoch 98/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1660
Epoch 99/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1650
Epoch 100/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1640
Epoch 101/300
4/4 [==============================] - 0s 4ms/step - loss: 0.1630
Epoch 102/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1621
Epoch 103/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1611
Epoch 104/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1602
Epoch 105/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1592
Epoch 106/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1583
Epoch 107/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1573
Epoch 108/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1564
Epoch 109/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1555
Epoch 110/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1546
Epoch 111/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1538
Epoch 112/300
4/4 [==============================] - 0s 5ms/step - loss: 0.1529
Epoch 113/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1520
Epoch 114/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1512
Epoch 115/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1503
Epoch 116/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1495
```

```
Epoch 117/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1487
Epoch 118/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1478
Epoch 119/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1470
Epoch 120/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1462
Epoch 121/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1454
Epoch 122/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1446
Epoch 123/300
4/4 [==============================] - 0s 4ms/step - loss: 0.1439
Epoch 124/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1431
Epoch 125/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1423
Epoch 126/300
4/4 [==============================] - ETA: 0s - loss: 0.125 - 0s 2ms/step - loss:
0.1416
Epoch 127/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1408
Epoch 128/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1401
Epoch 129/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1393
Epoch 130/300
4/4 [==============================] - 0s 4ms/step - loss: 0.1386
Epoch 131/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1379
Epoch 132/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1372
Epoch 133/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1365
Epoch 134/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1358
Epoch 135/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1351
Epoch 136/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1344
Epoch 137/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1337
Epoch 138/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1331
Epoch 139/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1324
Epoch 140/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1317
Epoch 141/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1311
Epoch 142/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1304
Epoch 143/300
4/4 [==============================] - ETA: 0s - loss: 0.004 - 0s 3ms/step - loss:
0.1298
Epoch 144/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1292
Epoch 145/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1285
Epoch 146/300
```

```
4/4 [==============================] - 0s 4ms/step - loss: 0.1279
Epoch 147/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1273
Epoch 148/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1267
Epoch 149/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1261
Epoch 150/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1255
Epoch 151/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1249
Epoch 152/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1243
Epoch 153/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1237
Epoch 154/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1231
Epoch 155/300
4/4 [==============================] - 0s 4ms/step - loss: 0.1225
Epoch 156/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1220
Epoch 157/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1214
Epoch 158/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1209
Epoch 159/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1203
Epoch 160/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1197
Epoch 161/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1192
Epoch 162/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1187
Epoch 163/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1181
Epoch 164/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1176
Epoch 165/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1171
Epoch 166/300
4/4 [==============================] - 0s 4ms/step - loss: 0.1166
Epoch 167/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1160
Epoch 168/300
4/4 [==============================] - 0s 7ms/step - loss: 0.1155
Epoch 169/300
4/4 [==============================] - 0s 4ms/step - loss: 0.1150
Epoch 170/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1145
Epoch 171/300
4/4 [==============================] - 0s 4ms/step - loss: 0.1140
Epoch 172/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1135
Epoch 173/300
4/4 [==============================] - 0s 4ms/step - loss: 0.1130
Epoch 174/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1125
Epoch 175/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1120
Epoch 176/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1116
```

```
Epoch 177/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1111
Epoch 178/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1106
Epoch 179/300
4/4 [==============================] - 0s 4ms/step - loss: 0.1101
Epoch 180/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1097
Epoch 181/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1092
Epoch 182/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1087
Epoch 183/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1083
Epoch 184/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1078
Epoch 185/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1074
Epoch 186/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1070
Epoch 187/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1065
Epoch 188/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1061
Epoch 189/300
4/4 [==============================] - 0s 5ms/step - loss: 0.1056
Epoch 190/300
4/4 [==============================] - 0s 4ms/step - loss: 0.1052
Epoch 191/300
4/4 [==============================] - 0s 4ms/step - loss: 0.1048
Epoch 192/300
4/4 [==============================] - 0s 4ms/step - loss: 0.1044
Epoch 193/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1039
Epoch 194/300
4/4 [==============================] - 0s 1ms/step - loss: 0.1035
Epoch 195/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1031
Epoch 196/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1027
Epoch 197/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1023
Epoch 198/300
4/4 [==============================] - 0s 3ms/step - loss: 0.1019
Epoch 199/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1015
Epoch 200/300
4/4 [==============================] - 0s 2ms/step - loss: 0.1011
Epoch 201/300
4/4 [==============================] - 0s 4ms/step - loss: 0.1007
Epoch 202/300
4/4 [==============================] - 0s 5ms/step - loss: 0.1003
Epoch 203/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0999
Epoch 204/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0995
Epoch 205/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0991
Epoch 206/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0987
Epoch 207/300
```

```
4/4 [==============================] - ETA: 0s - loss: 0.001 - 0s 4ms/step - loss:
0.0984
Epoch 208/300
4/4 [==============================] - ETA: 0s - loss: 0.001 - 0s 3ms/step - loss:
0.0980
Epoch 209/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0976
Epoch 210/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0973
Epoch 211/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0969
Epoch 212/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0965
Epoch 213/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0962
Epoch 214/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0958
Epoch 215/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0954
Epoch 216/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0951
Epoch 217/300
4/4 [==============================] - 0s 5ms/step - loss: 0.0947
Epoch 218/300
4/4 [==============================] - 0s 2ms/step - loss: 0.0944
Epoch 219/300
4/4 [==============================] - 0s 5ms/step - loss: 0.0940
Epoch 220/300
4/4 [==============================] - 0s 5ms/step - loss: 0.0937
Epoch 221/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0933
Epoch 222/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0930
Epoch 223/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0927
Epoch 224/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0923
Epoch 225/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0920
Epoch 226/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0917
Epoch 227/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0913
Epoch 228/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0910
Epoch 229/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0907
Epoch 230/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0904
Epoch 231/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0900
Epoch 232/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0897
Epoch 233/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0894
Epoch 234/300
4/4 [==============================] - 0s 6ms/step - loss: 0.0891
Epoch 235/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0888
Epoch 236/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0885
```

```
Epoch 237/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0882
Epoch 238/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0879
Epoch 239/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0876
Epoch 240/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0873
Epoch 241/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0870
Epoch 242/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0867
Epoch 243/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0864
Epoch 244/300
4/4 [==============================] - 0s 2ms/step - loss: 0.0861
Epoch 245/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0858
Epoch 246/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0855
Epoch 247/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0852
Epoch 248/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0849
Epoch 249/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0846
Epoch 250/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0843
Epoch 251/300
4/4 [==============================] - 0s 7ms/step - loss: 0.0841
Epoch 252/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0838
Epoch 253/300
4/4 [==============================] - 0s 5ms/step - loss: 0.0835
Epoch 254/300
4/4 [==============================] - 0s 5ms/step - loss: 0.0832
Epoch 255/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0830
Epoch 256/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0827
Epoch 257/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0824
Epoch 258/300
4/4 [==============================] - 0s 5ms/step - loss: 0.0821
Epoch 259/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0819
Epoch 260/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0816
Epoch 261/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0813
Epoch 262/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0811
Epoch 263/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0808
Epoch 264/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0806
Epoch 265/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0803
Epoch 266/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0801
Epoch 267/300
```

```
4/4 [==============================] - 0s 2ms/step - loss: 0.0798
Epoch 268/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0795
Epoch 269/300
4/4 [==============================] - 0s 5ms/step - loss: 0.0793
Epoch 270/300
4/4 [==============================] - 0s 5ms/step - loss: 0.0790
Epoch 271/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0788
Epoch 272/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0785
Epoch 273/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0783
Epoch 274/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0781
Epoch 275/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0778
Epoch 276/300
4/4 [==============================] - 0s 2ms/step - loss: 0.0776
Epoch 277/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0773
Epoch 278/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0771
Epoch 279/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0769
Epoch 280/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0766
Epoch 281/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0764
Epoch 282/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0762
Epoch 283/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0759
Epoch 284/300
4/4 [==============================] - 0s 4ms/step - loss: 0.0757
Epoch 285/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0755
Epoch 286/300
4/4 [==============================] - 0s 2ms/step - loss: 0.0752
Epoch 287/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0750
Epoch 288/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0748
Epoch 289/300
4/4 [==============================] - 0s 2ms/step - loss: 0.0746
Epoch 290/300
4/4 [==============================] - 0s 2ms/step - loss: 0.0743
Epoch 291/300
4/4 [==============================] - 0s 2ms/step - loss: 0.0741
Epoch 292/300
4/4 [==============================] - 0s 2ms/step - loss: 0.0739
Epoch 293/300
4/4 [==============================] - 0s 2ms/step - loss: 0.0737
Epoch 294/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0735
Epoch 295/300
4/4 [==============================] - 0s 2ms/step - loss: 0.0733
Epoch 296/300
4/4 [==============================] - 0s 3ms/step - loss: 0.0730
Epoch 297/300
4/4 [==============================] - 0s 2ms/step - loss: 0.0728
```

```
Epoch 298/300
4/4 [==============================] - 0s 1ms/step - loss: 0.0726
Epoch 299/300
4/4 [==============================] - 0s 2ms/step - loss: 0.0724
Epoch 300/300
4/4 [==============================] - 0s 2ms/step - loss: 0.0722
TEST
[[ True]
 [ True]
 [ True]
 [ True]]
```

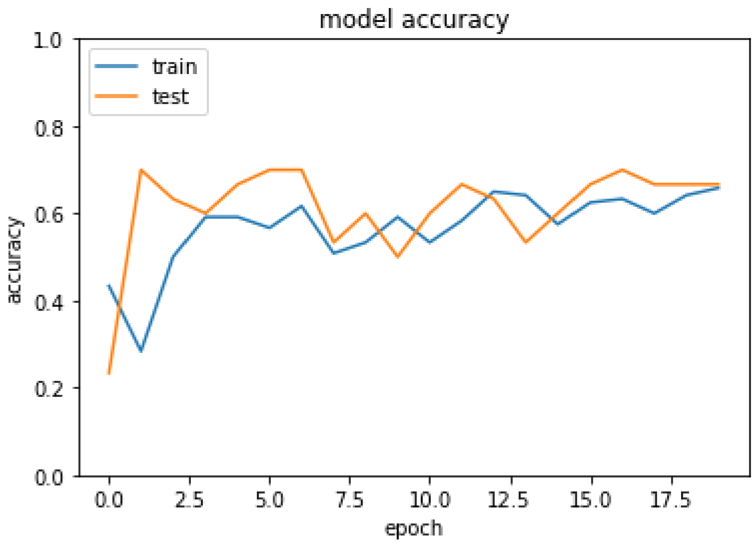
分類 (iris)

In [11]:

```
import matplotlib.pyplot as plt
from sklearn import datasets
iris = datasets.load_iris()
x = iris.data
d = iris.target

from sklearn.model_selection import train_test_split
x_train, x_test, d_train, d_test = train_test_split(x, d, test_size=0.2)

from keras.models import Sequential
from keras.layers import Dense, Activation
# from keras.optimizers import SGD

#モデルの設定
model = Sequential()
model.add(Dense(12, input_dim=4))
model.add(Activation('relu'))
# model.add(Activation('sigmoid'))
model.add(Dense(3, input_dim=12))
model.add(Activation('softmax'))
model.summary()

model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = model.fit(x_train, d_train, batch_size=5, epochs=20, verbose=1, validation_data=(x_tes
t, d_test))
loss = model.evaluate(x_test, d_test, verbose=0)

#Accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.ylim(0, 1.0)
plt.show()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 12)                60
_________________________________________________________________
activation_11 (Activation)   (None, 12)                0
_________________________________________________________________
dense_13 (Dense)             (None, 3)                 39
_________________________________________________________________
activation_12 (Activation)   (None, 3)                 0
=================================================================
Total params: 99
Trainable params: 99
Non-trainable params: 0
_________________________________________________________________
Train on 120 samples, validate on 30 samples
Epoch 1/20
120/120 [==============================] - 0s 4ms/step - loss: 1.5292 - acc: 0.433
3 - val_loss: 1.0707 - val_acc: 0.2333
Epoch 2/20
120/120 [==============================] - 0s 300us/step - loss: 1.0405 - acc: 0.2
833 - val_loss: 0.9770 - val_acc: 0.7000
Epoch 3/20
120/120 [==============================] - 0s 242us/step - loss: 0.9692 - acc: 0.5
000 - val_loss: 0.9803 - val_acc: 0.6333
Epoch 4/20
120/120 [==============================] - 0s 416us/step - loss: 0.9543 - acc: 0.5
917 - val_loss: 0.8821 - val_acc: 0.6000
Epoch 5/20
120/120 [==============================] - 0s 258us/step - loss: 0.8629 - acc: 0.5
917 - val_loss: 0.9066 - val_acc: 0.6667
Epoch 6/20
120/120 [==============================] - 0s 391us/step - loss: 0.8289 - acc: 0.5
667 - val_loss: 0.8026 - val_acc: 0.7000
Epoch 7/20
120/120 [==============================] - 0s 275us/step - loss: 0.7922 - acc: 0.6
167 - val_loss: 0.7643 - val_acc: 0.7000
Epoch 8/20
120/120 [==============================] - 0s 450us/step - loss: 0.7597 - acc: 0.5
083 - val_loss: 0.7011 - val_acc: 0.5333
Epoch 9/20
120/120 [==============================] - 0s 316us/step - loss: 0.7190 - acc: 0.5
333 - val_loss: 0.6758 - val_acc: 0.6000
Epoch 10/20
120/120 [==============================] - 0s 466us/step - loss: 0.6861 - acc: 0.5
917 - val_loss: 0.6500 - val_acc: 0.5000
Epoch 11/20
120/120 [==============================] - 0s 316us/step - loss: 0.6549 - acc: 0.5
333 - val_loss: 0.6194 - val_acc: 0.6000
Epoch 12/20
120/120 [==============================] - 0s 441us/step - loss: 0.6320 - acc: 0.5
833 - val_loss: 0.6318 - val_acc: 0.6667
Epoch 13/20
120/120 [==============================] - 0s 258us/step - loss: 0.6272 - acc: 0.6
500 - val_loss: 0.5781 - val_acc: 0.6333
Epoch 14/20
120/120 [==============================] - 0s 333us/step - loss: 0.5940 - acc: 0.6
417 - val_loss: 0.5578 - val_acc: 0.5333
Epoch 15/20
120/120 [==============================] - 0s 358us/step - loss: 0.5793 - acc: 0.5
750 - val_loss: 0.5419 - val_acc: 0.6000
```

```
Epoch 16/20
120/120 [==============================] - 0s 283us/step - loss: 0.5669 - acc: 0.6
250 - val_loss: 0.5305 - val_acc: 0.6667
Epoch 17/20
120/120 [==============================] - 0s 350us/step - loss: 0.5545 - acc: 0.6
333 - val_loss: 0.5177 - val_acc: 0.7000
Epoch 18/20
120/120 [==============================] - 0s 267us/step - loss: 0.5418 - acc: 0.6
000 - val_loss: 0.5033 - val_acc: 0.6667
Epoch 19/20
120/120 [==============================] - 0s 350us/step - loss: 0.5225 - acc: 0.6
417 - val_loss: 0.5106 - val_acc: 0.6667
Epoch 20/20
120/120 [==============================] - 0s 267us/step - loss: 0.5178 - acc: 0.6
583 - val_loss: 0.4915 - val_acc: 0.6667
```



[try]

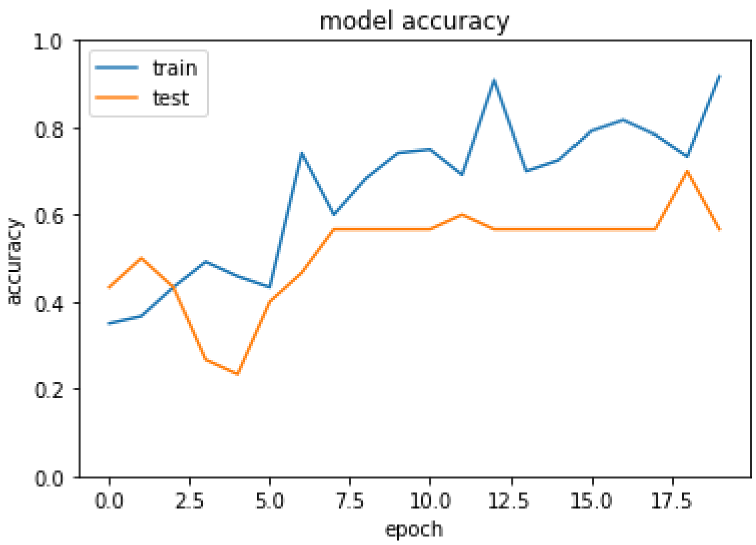
- 中間層の活性関数をsigmoidに変更しよう

In [12]:

```
import matplotlib.pyplot as plt
from sklearn import datasets
iris = datasets.load_iris()
x = iris.data
d = iris.target

from sklearn.model_selection import train_test_split
x_train, x_test, d_train, d_test = train_test_split(x, d, test_size=0.2)

from keras.models import Sequential
from keras.layers import Dense, Activation
# from keras.optimizers import SGD

#モデルの設定
model = Sequential()
model.add(Dense(12, input_dim=4))
#model.add(Activation('relu'))
model.add(Activation('sigmoid'))
model.add(Dense(3, input_dim=12))
model.add(Activation('softmax'))
model.summary()

model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = model.fit(x_train, d_train, batch_size=5, epochs=20, verbose=1, validation_data=(x_tes
t, d_test))
loss = model.evaluate(x_test, d_test, verbose=0)

#Accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.ylim(0, 1.0)
plt.show()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_14 (Dense)             (None, 12)                60
_________________________________________________________________
activation_13 (Activation)   (None, 12)                0
_________________________________________________________________
dense_15 (Dense)             (None, 3)                 39
_________________________________________________________________
activation_14 (Activation)   (None, 3)                 0
=================================================================
Total params: 99
Trainable params: 99
Non-trainable params: 0
_________________________________________________________________
Train on 120 samples, validate on 30 samples
Epoch 1/20
120/120 [==============================] - 1s 4ms/step - loss: 1.1024 - acc: 0.350
0 - val_loss: 1.1125 - val_acc: 0.4333
Epoch 2/20
120/120 [==============================] - 0s 375us/step - loss: 1.0687 - acc: 0.3
667 - val_loss: 1.0705 - val_acc: 0.5000
Epoch 3/20
120/120 [==============================] - 0s 309us/step - loss: 1.0462 - acc: 0.4
333 - val_loss: 1.0436 - val_acc: 0.4333
Epoch 4/20
120/120 [==============================] - 0s 358us/step - loss: 1.0250 - acc: 0.4
917 - val_loss: 1.0216 - val_acc: 0.2667
Epoch 5/20
120/120 [==============================] - 0s 326us/step - loss: 1.0093 - acc: 0.4
583 - val_loss: 0.9995 - val_acc: 0.2333
Epoch 6/20
120/120 [==============================] - 0s 366us/step - loss: 0.9956 - acc: 0.4
333 - val_loss: 0.9816 - val_acc: 0.4000
Epoch 7/20
120/120 [==============================] - 0s 383us/step - loss: 0.9805 - acc: 0.7
417 - val_loss: 0.9723 - val_acc: 0.4667
Epoch 8/20
120/120 [==============================] - 0s 466us/step - loss: 0.9677 - acc: 0.6
000 - val_loss: 0.9527 - val_acc: 0.5667
Epoch 9/20
120/120 [==============================] - 0s 416us/step - loss: 0.9543 - acc: 0.6
833 - val_loss: 0.9347 - val_acc: 0.5667
Epoch 10/20
120/120 [==============================] - 0s 341us/step - loss: 0.9424 - acc: 0.7
417 - val_loss: 0.9235 - val_acc: 0.5667
Epoch 11/20
120/120 [==============================] - 0s 383us/step - loss: 0.9295 - acc: 0.7
500 - val_loss: 0.9110 - val_acc: 0.5667
Epoch 12/20
120/120 [==============================] - 0s 383us/step - loss: 0.9159 - acc: 0.6
917 - val_loss: 0.8874 - val_acc: 0.6000
Epoch 13/20
120/120 [==============================] - 0s 567us/step - loss: 0.9048 - acc: 0.9
083 - val_loss: 0.8842 - val_acc: 0.5667
Epoch 14/20
120/120 [==============================] - 0s 267us/step - loss: 0.8925 - acc: 0.7
000 - val_loss: 0.8643 - val_acc: 0.5667
Epoch 15/20
120/120 [==============================] - 0s 368us/step - loss: 0.8798 - acc: 0.7
250 - val_loss: 0.8487 - val_acc: 0.5667
```

```
Epoch 16/20
120/120 [==============================] - 0s 283us/step - loss: 0.8681 - acc: 0.7
917 - val_loss: 0.8364 - val_acc: 0.5667
Epoch 17/20
120/120 [==============================] - 0s 366us/step - loss: 0.8577 - acc: 0.8
167 - val_loss: 0.8272 - val_acc: 0.5667
Epoch 18/20
120/120 [==============================] - 0s 316us/step - loss: 0.8443 - acc: 0.7
833 - val_loss: 0.8172 - val_acc: 0.5667
Epoch 19/20
120/120 [==============================] - 0s 366us/step - loss: 0.8320 - acc: 0.7
333 - val_loss: 0.7970 - val_acc: 0.7000
Epoch 20/20
120/120 [==============================] - 0s 275us/step - loss: 0.8213 - acc: 0.9
167 - val_loss: 0.7911 - val_acc: 0.5667
```



[try]

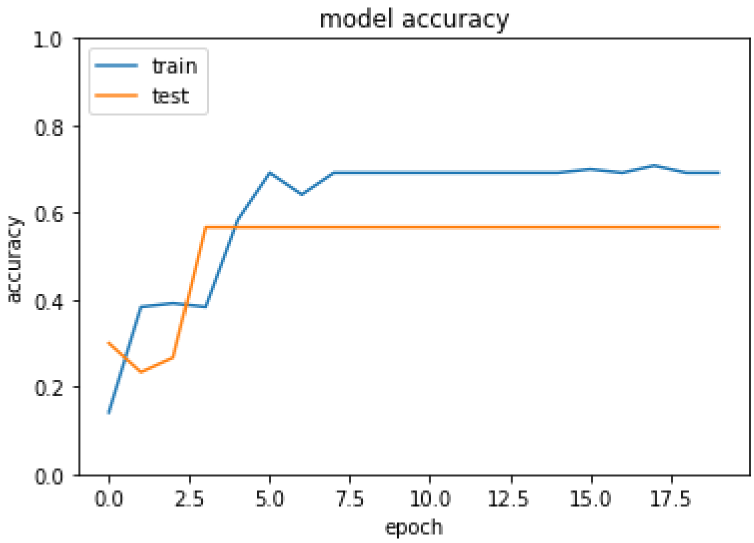
- 中間層の活性関数をsigmoidに変更しよう
- SGDをimportしoptimizerをSGD(lr=0.1)に変更しよう

In [13]:

```
import matplotlib.pyplot as plt
from sklearn import datasets
iris = datasets.load_iris()
x = iris.data
d = iris.target

from sklearn.model_selection import train_test_split
x_train, x_test, d_train, d_test = train_test_split(x, d, test_size=0.2)

from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD

#モデルの設定
model = Sequential()
model.add(Dense(12, input_dim=4))
#model.add(Activation('relu'))
model.add(Activation('sigmoid'))
model.add(Dense(3, input_dim=12))
model.add(Activation('softmax'))
model.summary()

sgd=SGD(lr=0.1)

model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = model.fit(x_train, d_train, batch_size=5, epochs=20, verbose=1, validation_data=(x_tes
t, d_test))
loss = model.evaluate(x_test, d_test, verbose=0)

#Accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.ylim(0, 1.0)
plt.show()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_16 (Dense)             (None, 12)                60
_________________________________________________________________
activation_15 (Activation)   (None, 12)                0
_________________________________________________________________
dense_17 (Dense)             (None, 3)                 39
_________________________________________________________________
activation_16 (Activation)   (None, 3)                 0
=================================================================
Total params: 99
Trainable params: 99
Non-trainable params: 0
_________________________________________________________________
Train on 120 samples, validate on 30 samples
Epoch 1/20
120/120 [==============================] - 1s 4ms/step - loss: 1.1119 - acc: 0.141
7 - val_loss: 1.1253 - val_acc: 0.3000
Epoch 2/20
120/120 [==============================] - 0s 383us/step - loss: 1.0917 - acc: 0.3
833 - val_loss: 1.1125 - val_acc: 0.2333
Epoch 3/20
120/120 [==============================] - 0s 250us/step - loss: 1.0801 - acc: 0.3
917 - val_loss: 1.1084 - val_acc: 0.2667
Epoch 4/20
120/120 [==============================] - 0s 441us/step - loss: 1.0639 - acc: 0.3
833 - val_loss: 1.0910 - val_acc: 0.5667
Epoch 5/20
120/120 [==============================] - 0s 267us/step - loss: 1.0515 - acc: 0.5
833 - val_loss: 1.0773 - val_acc: 0.5667
Epoch 6/20
120/120 [==============================] - 0s 475us/step - loss: 1.0396 - acc: 0.6
917 - val_loss: 1.0654 - val_acc: 0.5667
Epoch 7/20
120/120 [==============================] - 0s 283us/step - loss: 1.0249 - acc: 0.6
417 - val_loss: 1.0559 - val_acc: 0.5667
Epoch 8/20
120/120 [==============================] - 0s 400us/step - loss: 1.0129 - acc: 0.6
917 - val_loss: 1.0474 - val_acc: 0.5667
Epoch 9/20
120/120 [==============================] - 0s 283us/step - loss: 0.9992 - acc: 0.6
917 - val_loss: 1.0309 - val_acc: 0.5667
Epoch 10/20
120/120 [==============================] - 0s 350us/step - loss: 0.9854 - acc: 0.6
917 - val_loss: 1.0147 - val_acc: 0.5667
Epoch 11/20
120/120 [==============================] - 0s 400us/step - loss: 0.9734 - acc: 0.6
917 - val_loss: 1.0061 - val_acc: 0.5667
Epoch 12/20
120/120 [==============================] - 0s 350us/step - loss: 0.9602 - acc: 0.6
917 - val_loss: 0.9938 - val_acc: 0.5667
Epoch 13/20
120/120 [==============================] - 0s 400us/step - loss: 0.9485 - acc: 0.6
917 - val_loss: 0.9867 - val_acc: 0.5667
Epoch 14/20
120/120 [==============================] - 0s 275us/step - loss: 0.9356 - acc: 0.6
917 - val_loss: 0.9736 - val_acc: 0.5667
Epoch 15/20
120/120 [==============================] - 0s 366us/step - loss: 0.9204 - acc: 0.6
917 - val_loss: 0.9612 - val_acc: 0.5667
```

```
Epoch 16/20
120/120 [==============================] - 0s 283us/step - loss: 0.9104 - acc: 0.7
000 - val_loss: 0.9507 - val_acc: 0.5667
Epoch 17/20
120/120 [==============================] - 0s 391us/step - loss: 0.8943 - acc: 0.6
917 - val_loss: 0.9365 - val_acc: 0.5667
Epoch 18/20
120/120 [==============================] - 0s 291us/step - loss: 0.8827 - acc: 0.7
083 - val_loss: 0.9259 - val_acc: 0.5667
Epoch 19/20
120/120 [==============================] - 0s 391us/step - loss: 0.8692 - acc: 0.6
917 - val_loss: 0.9115 - val_acc: 0.5667
Epoch 20/20
120/120 [==============================] - 0s 283us/step - loss: 0.8562 - acc: 0.6
917 - val_loss: 0.8987 - val_acc: 0.5667
```



[try]

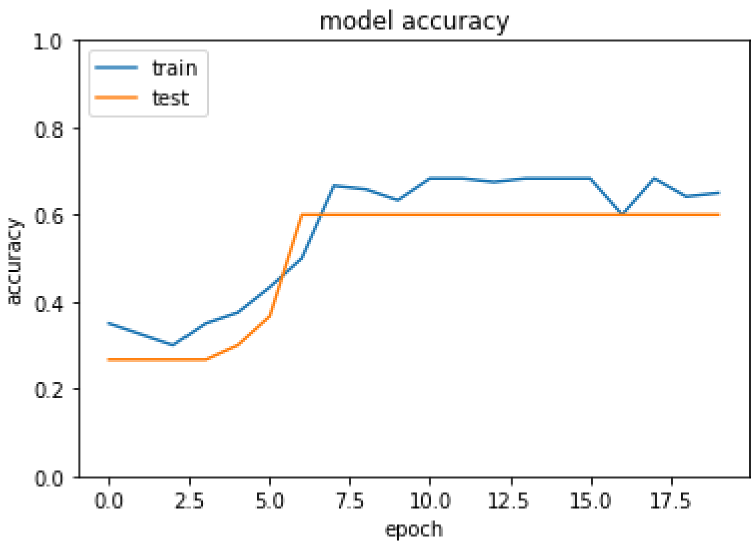
- 中間層の活性関数をsigmoidに変更しよう
- SGDをimportしoptimizerをSGD(lr=1)に変更しよう

In [14]:

```
import matplotlib.pyplot as plt
from sklearn import datasets
iris = datasets.load_iris()
x = iris.data
d = iris.target

from sklearn.model_selection import train_test_split
x_train, x_test, d_train, d_test = train_test_split(x, d, test_size=0.2)

from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD

#モデルの設定
model = Sequential()
model.add(Dense(12, input_dim=4))
#model.add(Activation('relu'))
model.add(Activation('sigmoid'))
model.add(Dense(3, input_dim=12))
model.add(Activation('softmax'))
model.summary()

sgd=SGD(lr=1)

model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = model.fit(x_train, d_train, batch_size=5, epochs=20, verbose=1, validation_data=(x_tes
t, d_test))
loss = model.evaluate(x_test, d_test, verbose=0)

#Accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.ylim(0, 1.0)
plt.show()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_18 (Dense)             (None, 12)                60
_________________________________________________________________
activation_17 (Activation)   (None, 12)                0
_________________________________________________________________
dense_19 (Dense)             (None, 3)                 39
_________________________________________________________________
activation_18 (Activation)   (None, 3)                 0
=================================================================
Total params: 99
Trainable params: 99
Non-trainable params: 0
_________________________________________________________________
Train on 120 samples, validate on 30 samples
Epoch 1/20
120/120 [==============================] - 1s 5ms/step - loss: 1.1511 - acc: 0.350
0 - val_loss: 1.1524 - val_acc: 0.2667
Epoch 2/20
120/120 [==============================] - 0s 375us/step - loss: 1.1348 - acc: 0.3
250 - val_loss: 1.1345 - val_acc: 0.2667
Epoch 3/20
120/120 [==============================] - 0s 341us/step - loss: 1.1203 - acc: 0.3
000 - val_loss: 1.1218 - val_acc: 0.2667
Epoch 4/20
120/120 [==============================] - 0s 333us/step - loss: 1.1043 - acc: 0.3
500 - val_loss: 1.1057 - val_acc: 0.2667
Epoch 5/20
120/120 [==============================] - 0s 292us/step - loss: 1.0914 - acc: 0.3
750 - val_loss: 1.0877 - val_acc: 0.3000
Epoch 6/20
120/120 [==============================] - 0s 441us/step - loss: 1.0778 - acc: 0.4
333 - val_loss: 1.0768 - val_acc: 0.3667
Epoch 7/20
120/120 [==============================] - 0s 275us/step - loss: 1.0658 - acc: 0.5
000 - val_loss: 1.0639 - val_acc: 0.6000
Epoch 8/20
120/120 [==============================] - 0s 441us/step - loss: 1.0508 - acc: 0.6
667 - val_loss: 1.0469 - val_acc: 0.6000
Epoch 9/20
120/120 [==============================] - 0s 275us/step - loss: 1.0367 - acc: 0.6
583 - val_loss: 1.0343 - val_acc: 0.6000
Epoch 10/20
120/120 [==============================] - 0s 433us/step - loss: 1.0205 - acc: 0.6
333 - val_loss: 1.0220 - val_acc: 0.6000
Epoch 11/20
120/120 [==============================] - 0s 275us/step - loss: 1.0052 - acc: 0.6
833 - val_loss: 1.0073 - val_acc: 0.6000
Epoch 12/20
120/120 [==============================] - 0s 400us/step - loss: 0.9885 - acc: 0.6
833 - val_loss: 0.9874 - val_acc: 0.6000
Epoch 13/20
120/120 [==============================] - 0s 375us/step - loss: 0.9688 - acc: 0.6
750 - val_loss: 0.9710 - val_acc: 0.6000
Epoch 14/20
120/120 [==============================] - 0s 392us/step - loss: 0.9527 - acc: 0.6
833 - val_loss: 0.9536 - val_acc: 0.6000
Epoch 15/20
120/120 [==============================] - 0s 316us/step - loss: 0.9340 - acc: 0.6
833 - val_loss: 0.9345 - val_acc: 0.6000
```

```
Epoch 16/20
120/120 [==============================] - 0s 325us/step - loss: 0.9144 - acc: 0.6
833 - val_loss: 0.9113 - val_acc: 0.6000
Epoch 17/20
120/120 [==============================] - 0s 416us/step - loss: 0.8987 - acc: 0.6
000 - val_loss: 0.8975 - val_acc: 0.6000
Epoch 18/20
120/120 [==============================] - 0s 350us/step - loss: 0.8830 - acc: 0.6
833 - val_loss: 0.8771 - val_acc: 0.6000
Epoch 19/20
120/120 [==============================] - 0s 400us/step - loss: 0.8657 - acc: 0.6
417 - val_loss: 0.8619 - val_acc: 0.6000
Epoch 20/20
120/120 [==============================] - 0s 291us/step - loss: 0.8512 - acc: 0.6
500 - val_loss: 0.8446 - val_acc: 0.6000
```



[try]

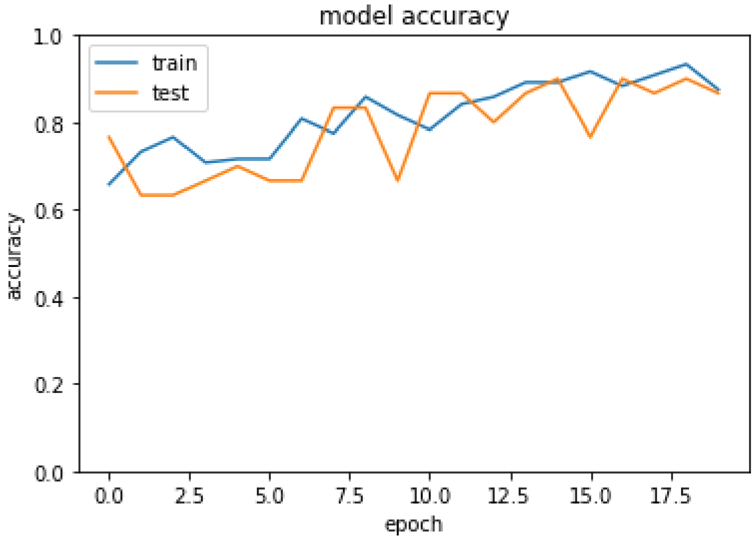
- 中間層の活性関数をReLUに戻す
- SGDをimportしoptimizerをSGD(lr=1)に変更しよう

In [22]:

```
import matplotlib.pyplot as plt
from sklearn import datasets
iris = datasets.load_iris()
x = iris.data
d = iris.target

from sklearn.model_selection import train_test_split
x_train, x_test, d_train, d_test = train_test_split(x, d, test_size=0.2)

from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD

#モデルの設定
model = Sequential()
model.add(Dense(12, input_dim=4))
model.add(Activation('relu'))
#model.add(Activation('sigmoid'))
model.add(Dense(3, input_dim=12))
model.add(Activation('softmax'))
model.summary()

sgd=SGD(lr=1)

model.compile(optimizer='sgd', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

history = model.fit(x_train, d_train, batch_size=5, epochs=20, verbose=1, validation_data=(x_tes
t, d_test))
loss = model.evaluate(x_test, d_test, verbose=0)

#Accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.ylim(0, 1.0)
plt.show()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_41 (Dense)             (None, 12)                60
_________________________________________________________________
activation_19 (Activation)   (None, 12)                0
_________________________________________________________________
dense_42 (Dense)             (None, 3)                 39
_________________________________________________________________
activation_20 (Activation)   (None, 3)                 0
=================================================================
Total params: 99
Trainable params: 99
Non-trainable params: 0
_________________________________________________________________
Train on 120 samples, validate on 30 samples
Epoch 1/20
120/120 [==============================] - 1s 9ms/step - loss: 1.0122 - acc: 0.658
3 - val_loss: 0.7689 - val_acc: 0.7667
Epoch 2/20
120/120 [==============================] - 0s 300us/step - loss: 0.7268 - acc: 0.7
333 - val_loss: 0.7083 - val_acc: 0.6333
Epoch 3/20
120/120 [==============================] - 0s 267us/step - loss: 0.6392 - acc: 0.7
667 - val_loss: 0.6956 - val_acc: 0.6333
Epoch 4/20
120/120 [==============================] - 0s 400us/step - loss: 0.6099 - acc: 0.7
083 - val_loss: 0.5898 - val_acc: 0.6667
Epoch 5/20
120/120 [==============================] - 0s 366us/step - loss: 0.5663 - acc: 0.7
167 - val_loss: 0.5526 - val_acc: 0.7000
Epoch 6/20
120/120 [==============================] - 0s 316us/step - loss: 0.5594 - acc: 0.7
167 - val_loss: 0.5335 - val_acc: 0.6667
Epoch 7/20
120/120 [==============================] - 0s 383us/step - loss: 0.5079 - acc: 0.8
083 - val_loss: 0.5521 - val_acc: 0.6667
Epoch 8/20
120/120 [==============================] - 0s 283us/step - loss: 0.4874 - acc: 0.7
750 - val_loss: 0.4724 - val_acc: 0.8333
Epoch 9/20
120/120 [==============================] - 0s 275us/step - loss: 0.4748 - acc: 0.8
583 - val_loss: 0.4669 - val_acc: 0.8333
Epoch 10/20
120/120 [==============================] - 0s 391us/step - loss: 0.4527 - acc: 0.8
167 - val_loss: 0.4950 - val_acc: 0.6667
Epoch 11/20
120/120 [==============================] - 0s 250us/step - loss: 0.4599 - acc: 0.7
833 - val_loss: 0.4353 - val_acc: 0.8667
Epoch 12/20
120/120 [==============================] - 0s 458us/step - loss: 0.4365 - acc: 0.8
417 - val_loss: 0.4177 - val_acc: 0.8667
Epoch 13/20
120/120 [==============================] - 0s 358us/step - loss: 0.4114 - acc: 0.8
583 - val_loss: 0.4121 - val_acc: 0.8000
Epoch 14/20
120/120 [==============================] - 0s 550us/step - loss: 0.4160 - acc: 0.8
917 - val_loss: 0.4113 - val_acc: 0.8667
Epoch 15/20
120/120 [==============================] - 0s 358us/step - loss: 0.3956 - acc: 0.8
917 - val_loss: 0.3916 - val_acc: 0.9000
```

```
Epoch 16/20
120/120 [==============================] - 0s 516us/step - loss: 0.3832 - acc: 0.9
167 - val_loss: 0.4146 - val_acc: 0.7667
Epoch 17/20
120/120 [==============================] - 0s 316us/step - loss: 0.3755 - acc: 0.8
833 - val_loss: 0.3705 - val_acc: 0.9000
Epoch 18/20
120/120 [==============================] - 0s 450us/step - loss: 0.3708 - acc: 0.9
083 - val_loss: 0.3822 - val_acc: 0.8667
Epoch 19/20
120/120 [==============================] - 0s 300us/step - loss: 0.3521 - acc: 0.9
333 - val_loss: 0.3553 - val_acc: 0.9000
Epoch 20/20
120/120 [==============================] - 0s 475us/step - loss: 0.3607 - acc: 0.8
750 - val_loss: 0.3611 - val_acc: 0.8667
```



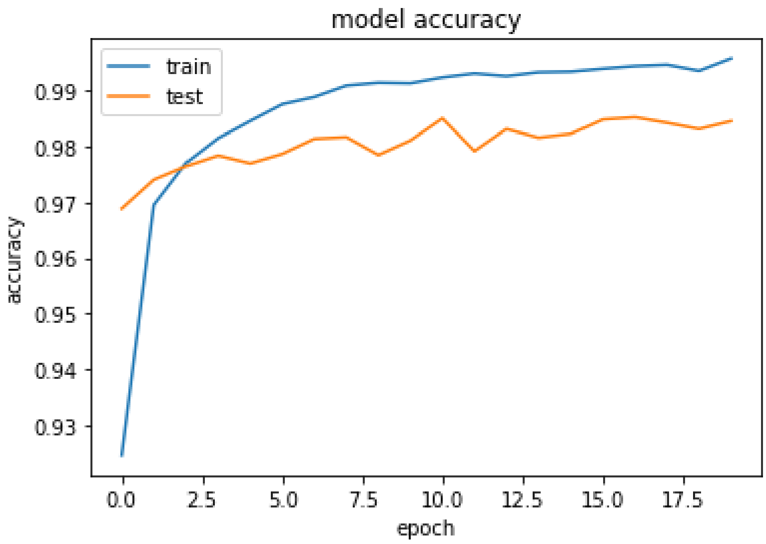
分類 (mnist)

In [15]:

```
# 必要なライブラリのインポート
import sys, os
sys.path.append(os.pardir)  # 親ディレクトリのファイルをインポートするための設定
import keras
import matplotlib.pyplot as plt
from data.mnist import load_mnist

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

# 必要なライブラリのインポート、最適化手法はAdamを使う
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam

# モデル作成
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))#入力層のニューロン数784個、中間層のニューロン数512個。
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))
model.summary()

# バッチサイズ、エポック数
batch_size = 128
epochs = 20

model.compile(loss='categorical_crossentropy',
              optimizer=Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False),
              metrics=['accuracy'])

history = model.fit(x_train, d_train, batch_size=batch_size, epochs=epochs, verbose=1, validation_data=(x_test, d_test))
loss = model.evaluate(x_test, d_test, verbose=0)
print('Test loss:', loss[0])
print('Test accuracy:', loss[1])
#Accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
# plt.ylim(0, 1.0)
plt.show()
```

```
WARNING:tensorflow:From C:¥ProgramData¥Anaconda3¥lib¥site-packages¥keras¥backend¥t
ensorflow_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_ops) wit
h 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`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_20 (Dense)             (None, 512)               401920
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_21 (Dense)             (None, 512)               262656
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_22 (Dense)             (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 22s 371us/step - loss: 0.2519 - ac
c: 0.9246 - val_loss: 0.1057 - val_acc: 0.9689
Epoch 2/20
60000/60000 [==============================] - 23s 376us/step - loss: 0.1011 - ac
c: 0.9696 - val_loss: 0.0839 - val_acc: 0.9741
Epoch 3/20
60000/60000 [==============================] - 21s 347us/step - loss: 0.0728 - ac
c: 0.9771 - val_loss: 0.0800 - val_acc: 0.9765
Epoch 4/20
60000/60000 [==============================] - 20s 333us/step - loss: 0.0563 - ac
c: 0.9815 - val_loss: 0.0676 - val_acc: 0.9784
Epoch 5/20
60000/60000 [==============================] - 20s 338us/step - loss: 0.0467 - ac
c: 0.9846 - val_loss: 0.0754 - val_acc: 0.9770
Epoch 6/20
60000/60000 [==============================] - 20s 335us/step - loss: 0.0377 - ac
c: 0.9876 - val_loss: 0.0746 - val_acc: 0.9787
Epoch 7/20
60000/60000 [==============================] - 22s 371us/step - loss: 0.0352 - ac
c: 0.9889 - val_loss: 0.0688 - val_acc: 0.9814
Epoch 8/20
60000/60000 [==============================] - 21s 344us/step - loss: 0.0281 - ac
c: 0.9909 - val_loss: 0.0680 - val_acc: 0.9817
Epoch 9/20
60000/60000 [==============================] - 20s 338us/step - loss: 0.0268 - ac
c: 0.9915 - val_loss: 0.0788 - val_acc: 0.9785
Epoch 10/20
60000/60000 [==============================] - 21s 342us/step - loss: 0.0256 - ac
c: 0.9914 - val_loss: 0.0713 - val_acc: 0.9811
Epoch 11/20
60000/60000 [==============================] - 22s 359us/step - loss: 0.0221 - ac
c: 0.9924 - val_loss: 0.0653 - val_acc: 0.9851
Epoch 12/20
60000/60000 [==============================] - 23s 376us/step - loss: 0.0199 - ac
c: 0.9931 - val_loss: 0.0840 - val_acc: 0.9792
Epoch 13/20
```

```
60000/60000 [==============================] - 23s 377us/step - loss: 0.0216 - ac
c: 0.9926 - val_loss: 0.0696 - val_acc: 0.9832
Epoch 14/20
60000/60000 [==============================] - 24s 396us/step - loss: 0.0210 - ac
c: 0.9933 - val_loss: 0.0758 - val_acc: 0.9816
Epoch 15/20
60000/60000 [==============================] - 25s 420us/step - loss: 0.0193 - ac
c: 0.9934 - val_loss: 0.0812 - val_acc: 0.9823
Epoch 16/20
60000/60000 [==============================] - 24s 398us/step - loss: 0.0179 - ac
c: 0.9940 - val_loss: 0.0697 - val_acc: 0.9849
Epoch 17/20
60000/60000 [==============================] - 24s 398us/step - loss: 0.0166 - ac
c: 0.9944 - val_loss: 0.0668 - val_acc: 0.9853
Epoch 18/20
60000/60000 [==============================] - 24s 393us/step - loss: 0.0160 - ac
c: 0.9947 - val_loss: 0.0713 - val_acc: 0.9843
Epoch 19/20
60000/60000 [==============================] - 23s 383us/step - loss: 0.0188 - ac
c: 0.9936 - val_loss: 0.0761 - val_acc: 0.9832
Epoch 20/20
60000/60000 [==============================] - 22s 366us/step - loss: 0.0126 - ac
c: 0.9958 - val_loss: 0.0781 - val_acc: 0.9846
Test loss: 0.07812294893332487
Test accuracy: 0.9846
```



[try]

- load_mnistのone_hot_labelをFalseに変更しよう (error)

In [17]:

```
# 必要なライブラリのインポート
import sys, os
sys.path.append(os.pardir)  # 親ディレクトリのファイルをインポートするための設定
import keras
import matplotlib.pyplot as plt
from data.mnist import load_mnist

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

# 必要なライブラリのインポート、最適化手法はAdamを使う
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam

# モデル作成
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))#入力層のニューロン数784個、中間層のニューロン数512個。
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))
model.summary()

# バッチサイズ、エポック数
batch_size = 128
epochs = 20

model.compile(loss='categorical_crossentropy',
              optimizer=Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False),
              metrics=['accuracy'])

history = model.fit(x_train, d_train, batch_size=batch_size, epochs=epochs, verbose=1, validation_data=(x_test, d_test))
loss = model.evaluate(x_test, d_test, verbose=0)
print('Test loss:', loss[0])
print('Test accuracy:', loss[1])
#Accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
# plt.ylim(0, 1.0)
plt.show()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_26 (Dense)             (None, 512)               401920    
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_27 (Dense)             (None, 512)               262656    
_________________________________________________________________
dropout_6 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_28 (Dense)             (None, 10)                5130      
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-17-6438ebfa5ff4> in <module>()
     30               metrics=['accuracy'])
     31 
---> 32 history = model.fit(x_train, d_train, batch_size=batch_size, epochs=epoch
s, verbose=1, validation_data=(x_test, d_test))
     33 loss = model.evaluate(x_test, d_test, verbose=0)
     34 print('Test loss:', loss[0])

C:¥ProgramData¥Anaconda3¥lib¥site-packages¥keras¥engine¥training.py in fit(sel
f, x, y, batch_size, epochs, verbose, callbacks, validation_split, validation_d
ata, shuffle, class_weight, sample_weight, initial_epoch, steps_per_epoch, vali
dation_steps, **kwargs)
    950             sample_weight=sample_weight,
    951             class_weight=class_weight,
--> 952             batch_size=batch_size)
    953         # Prepare validation data.
    954         do_validation = False

C:¥ProgramData¥Anaconda3¥lib¥site-packages¥keras¥engine¥training.py in _standar
dize_user_data(self, x, y, sample_weight, class_weight, check_array_lengths, ba
tch_size)
    787                 feed_output_shapes,
    788                 check_batch_axis=False,  # Don't enforce the batch size.
--> 789                 exception_prefix='target')
    790 
    791             # Generate sample-wise weight values given the `sample_weigh
t` and

C:¥ProgramData¥Anaconda3¥lib¥site-packages¥keras¥engine¥training_utils.py in st
andardize_input_data(data, names, shapes, check_batch_axis, exception_prefix)
    136                             ': expected ' + names[i] + ' to have shape '
+
    137                             str(shape) + ' but got array with shape ' +
--> 138                             str(data_shape))
    139     return data
    140 

ValueError: Error when checking target: expected dense_28 to have shape (10,) but
got array with shape (1,)
```

[try]

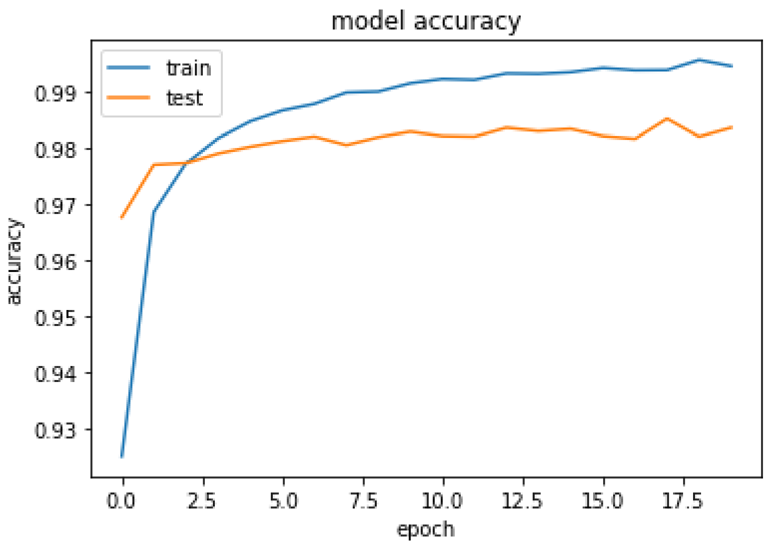
- load_mnistのone_hot_labelをFalseに変更しよう
- 誤差関数をsparse_categorical_crossentropyに変更しよう
- ※　one_hot_vector化する場合は、誤差関数をcategorical_crossentropy
- ※　one_hot_vector化しない場合は、誤差関数をsparse_categorical_crossentropy

In [23]:

```
# 必要なライブラリのインポート
import sys, os
sys.path.append(os.pardir)  # 親ディレクトリのファイルをインポートするための設定
import keras
import matplotlib.pyplot as plt
from data.mnist import load_mnist

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

# 必要なライブラリのインポート、最適化手法はAdamを使う
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam

# モデル作成
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))#入力層のニューロン数784個、中間層のニューロン数512個。
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))
model.summary()

# バッチサイズ、エポック数
batch_size = 128
epochs = 20

model.compile(loss='sparse_categorical_crossentropy',
              optimizer=Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False),
              metrics=['accuracy'])

print(d_test.shape)
history = model.fit(x_train, d_train, batch_size=batch_size, epochs=epochs, verbose=1, validation_data=(x_test, d_test))
loss = model.evaluate(x_test, d_test, verbose=0)
print('Test loss:', loss[0])
print('Test accuracy:', loss[1])
#Accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
# plt.ylim(0, 1.0)
plt.show()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_43 (Dense)             (None, 512)               401920
_________________________________________________________________
dropout_15 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_44 (Dense)             (None, 512)               262656
_________________________________________________________________
dropout_16 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_45 (Dense)             (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
(10000,)
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 25s 414us/step - loss: 0.2525 - ac
c: 0.9250 - val_loss: 0.0989 - val_acc: 0.9677
Epoch 2/20
60000/60000 [==============================] - 22s 373us/step - loss: 0.1026 - ac
c: 0.9686 - val_loss: 0.0749 - val_acc: 0.9771
Epoch 3/20
60000/60000 [==============================] - 22s 365us/step - loss: 0.0714 - ac
c: 0.9773 - val_loss: 0.0705 - val_acc: 0.9774
Epoch 4/20
60000/60000 [==============================] - 23s 384us/step - loss: 0.0572 - ac
c: 0.9818 - val_loss: 0.0670 - val_acc: 0.9791
Epoch 5/20
60000/60000 [==============================] - 21s 356us/step - loss: 0.0452 - ac
c: 0.9848 - val_loss: 0.0653 - val_acc: 0.9803
Epoch 6/20
60000/60000 [==============================] - 21s 349us/step - loss: 0.0396 - ac
c: 0.9868 - val_loss: 0.0655 - val_acc: 0.9813
Epoch 7/20
60000/60000 [==============================] - 21s 357us/step - loss: 0.0355 - ac
c: 0.9879 - val_loss: 0.0637 - val_acc: 0.9821
Epoch 8/20
60000/60000 [==============================] - 21s 351us/step - loss: 0.0299 - ac
c: 0.9900 - val_loss: 0.0677 - val_acc: 0.9806
Epoch 9/20
60000/60000 [==============================] - 22s 375us/step - loss: 0.0295 - ac
c: 0.9901 - val_loss: 0.0621 - val_acc: 0.9820
Epoch 10/20
60000/60000 [==============================] - 23s 390us/step - loss: 0.0251 - ac
c: 0.9916 - val_loss: 0.0638 - val_acc: 0.9830
Epoch 11/20
60000/60000 [==============================] - 22s 372us/step - loss: 0.0228 - ac
c: 0.9924 - val_loss: 0.0676 - val_acc: 0.9822ss: 0.0228 - acc: 0.99
Epoch 12/20
60000/60000 [==============================] - 23s 380us/step - loss: 0.0226 - ac
c: 0.9922 - val_loss: 0.0773 - val_acc: 0.9821 acc: 0
Epoch 13/20
60000/60000 [==============================] - 21s 355us/step - loss: 0.0200 - ac
c: 0.9934 - val_loss: 0.0634 - val_acc: 0.9837
Epoch 14/20
60000/60000 [==============================] - 22s 365us/step - loss: 0.0205 - ac
c: 0.9933 - val_loss: 0.0661 - val_acc: 0.9831
```

```
Epoch 15/20
60000/60000 [==============================] - 21s 357us/step - loss: 0.0192 - ac
c: 0.9936 - val_loss: 0.0650 - val_acc: 0.9835
Epoch 16/20
60000/60000 [==============================] - 22s 373us/step - loss: 0.0172 - ac
c: 0.9944 - val_loss: 0.0760 - val_acc: 0.9822
Epoch 17/20
60000/60000 [==============================] - 22s 364us/step - loss: 0.0184 - ac
c: 0.9939 - val_loss: 0.0832 - val_acc: 0.9817
Epoch 18/20
60000/60000 [==============================] - 21s 357us/step - loss: 0.0173 - ac
c: 0.9940 - val_loss: 0.0677 - val_acc: 0.9853
Epoch 19/20
60000/60000 [==============================] - 22s 360us/step - loss: 0.0118 - ac
c: 0.9958 - val_loss: 0.0859 - val_acc: 0.9821
Epoch 20/20
60000/60000 [==============================] - 21s 358us/step - loss: 0.0158 - ac
c: 0.9947 - val_loss: 0.0801 - val_acc: 0.9837
Test loss: 0.0800618705158955
Test accuracy: 0.9837
```



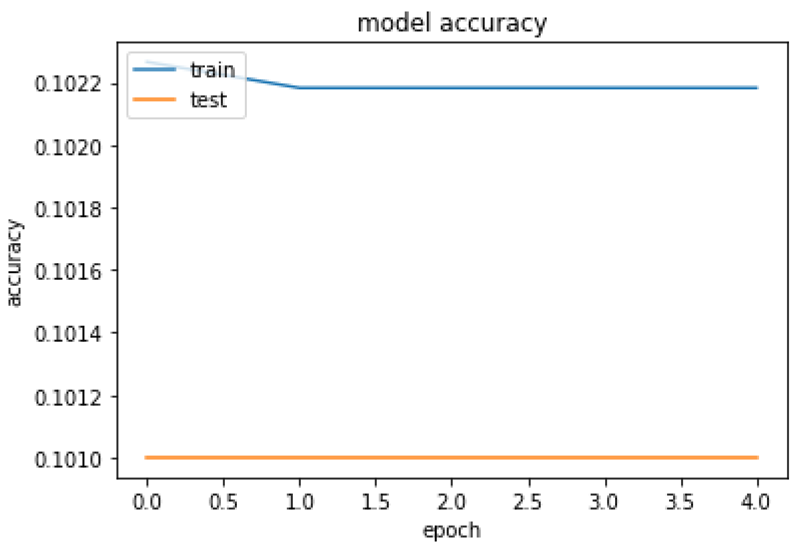
[try]

- 誤差関数をsparse_categorical_crossentropyに変更しよう
- Adamの引数の値を変更しよう(0.001→0.1)

In [26]:

```
# 必要なライブラリのインポート
import sys, os
sys.path.append(os.pardir)  # 親ディレクトリのファイルをインポートするための設定
import keras
import matplotlib.pyplot as plt
from data.mnist import load_mnist

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

# 必要なライブラリのインポート、最適化手法はAdamを使う
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam

# モデル作成
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))#入力層のニューロン数784個、中間層の
ニューロン数512個。
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))
model.summary()

# バッチサイズ、エポック数
batch_size = 128
epochs = 5

model.compile(loss='sparse_categorical_crossentropy',
              optimizer=Adam(lr=0.1, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=
False),
              metrics=['accuracy'])

print(d_test.shape)
history = model.fit(x_train, d_train, batch_size=batch_size, epochs=epochs, verbose=1, validatio
n_data=(x_test, d_test))
loss = model.evaluate(x_test, d_test, verbose=0)
print('Test loss:', loss[0])
print('Test accuracy:', loss[1])
#Accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
# plt.ylim(0, 1.0)
plt.show()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_52 (Dense)             (None, 512)               401920    
_________________________________________________________________
dropout_21 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_53 (Dense)             (None, 512)               262656    
_________________________________________________________________
dropout_22 (Dropout)         (None, 512)               0         
_________________________________________________________________
dense_54 (Dense)             (None, 10)                5130      
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
(10000,)
Train on 60000 samples, validate on 10000 samples
Epoch 1/5
60000/60000 [==============================] - 21s 343us/step - loss: 14.4438 - ac
c: 0.1023 - val_loss: 14.4902 - val_acc: 0.1010
Epoch 2/5
60000/60000 [==============================] - 18s 299us/step - loss: 14.4711 - ac
c: 0.1022 - val_loss: 14.4902 - val_acc: 0.1010
Epoch 3/5
60000/60000 [==============================] - 18s 300us/step - loss: 14.4711 - ac
c: 0.1022 - val_loss: 14.4902 - val_acc: 0.1010
Epoch 4/5
60000/60000 [==============================] - 18s 304us/step - loss: 14.4711 - ac
c: 0.1022 - val_loss: 14.4902 - val_acc: 0.1010
Epoch 5/5
60000/60000 [==============================] - 19s 311us/step - loss: 14.4711 - ac
c: 0.1022 - val_loss: 14.4902 - val_acc: 0.1010
Test loss: 14.490168928527831
Test accuracy: 0.101
```



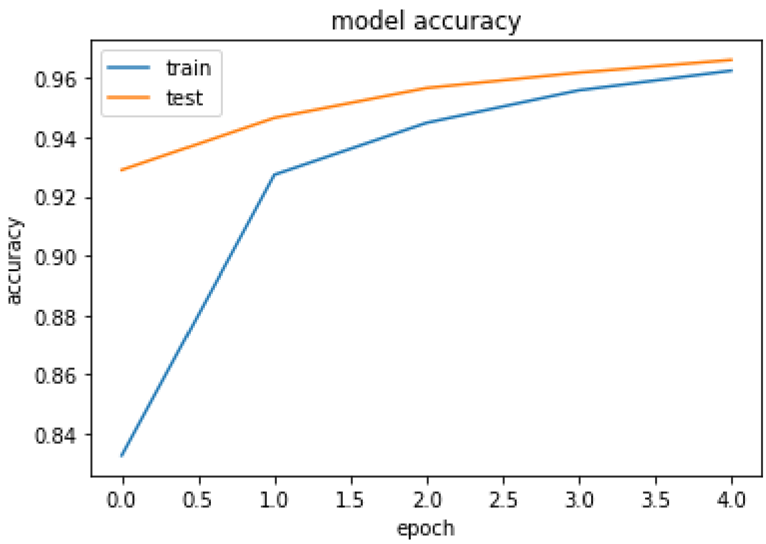
[try]

- 誤差関数をsparse_categorical_crossentropyに変更しよう
- Adamの引数の値を変更しよう(0.001→0.0001)

In [27]:

```
# 必要なライブラリのインポート
import sys, os
sys.path.append(os.pardir)  # 親ディレクトリのファイルをインポートするための設定
import keras
import matplotlib.pyplot as plt
from data.mnist import load_mnist

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

# 必要なライブラリのインポート、最適化手法はAdamを使う
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam

# モデル作成
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))#入力層のニューロン数784個、中間層の
ニューロン数512個。
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))
model.summary()

# バッチサイズ、エポック数
batch_size = 128
epochs = 5

model.compile(loss='sparse_categorical_crossentropy',
              optimizer=Adam(lr=0.0001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgr
ad=False),
              metrics=['accuracy'])

print(d_test.shape)
history = model.fit(x_train, d_train, batch_size=batch_size, epochs=epochs, verbose=1, validatio
n_data=(x_test, d_test))
loss = model.evaluate(x_test, d_test, verbose=0)
print('Test loss:', loss[0])
print('Test accuracy:', loss[1])
#Accuracy
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
# plt.ylim(0, 1.0)
plt.show()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_55 (Dense)             (None, 512)               401920
_________________________________________________________________
dropout_23 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_56 (Dense)             (None, 512)               262656
_________________________________________________________________
dropout_24 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_57 (Dense)             (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
(10000,)
Train on 60000 samples, validate on 10000 samples
Epoch 1/5
60000/60000 [==============================] - 20s 332us/step - loss: 0.6166 - ac
c: 0.8326 - val_loss: 0.2491 - val_acc: 0.9290
Epoch 2/5
60000/60000 [==============================] - 18s 301us/step - loss: 0.2496 - ac
c: 0.9274 - val_loss: 0.1792 - val_acc: 0.9466
Epoch 3/5
60000/60000 [==============================] - 18s 308us/step - loss: 0.1893 - ac
c: 0.9449 - val_loss: 0.1443 - val_acc: 0.9567
Epoch 4/5
60000/60000 [==============================] - 19s 315us/step - loss: 0.1525 - ac
c: 0.9559 - val_loss: 0.1249 - val_acc: 0.9619
Epoch 5/5
60000/60000 [==============================] - 19s 312us/step - loss: 0.1282 - ac
c: 0.9626 - val_loss: 0.1095 - val_acc: 0.9662
Test loss: 0.10950917197987438
Test accuracy: 0.9662
```



CNN分類 (mnist)

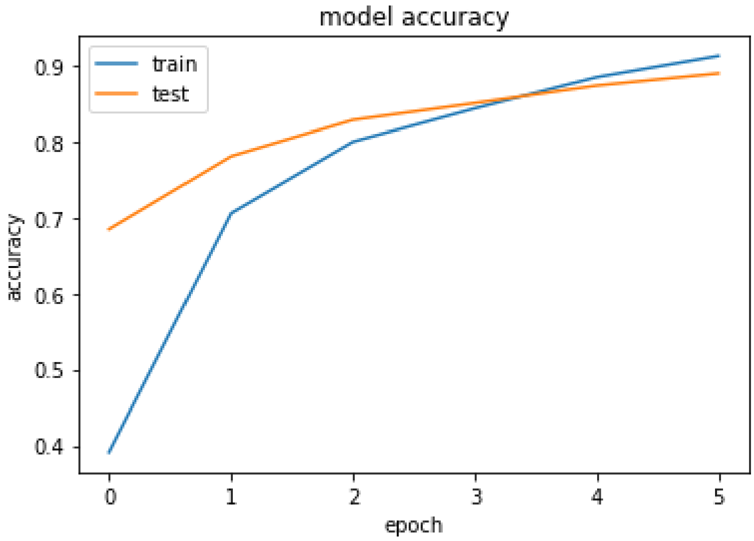
実行に時間がかかるため割愛

In [30]:

```
# 必要なライブラリのインポート
import sys, os
sys.path.append(os.pardir)  # 親ディレクトリのファイルをインポートするための設定
import keras
import matplotlib.pyplot as plt
from data.mnist import load_mnist

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

#実行時間を減らすために、データ数を削減。
x_train=x_train[:1000]
d_train=d_train[:1000]
x_test=x_test[:1000]
d_test=d_test[:1000]


# 行列として入力するための加工
batch_size = 128
num_classes = 10
#epochs = 3#20から３に変更

img_rows, img_cols = 28, 28

x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
input_shape = (img_rows, img_cols, 1)


# 必要なライブラリのインポート、最適化手法はAdamを使う
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.optimizers import Adam

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))#32チャンネル
model.add(Conv2D(64, (3, 3), activation='relu'))#64チャンネル
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.summary()

# バッチサイズ、エポック数
batch_size = 128
epochs = 6#20から3に変更


model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=['accuracy'])
history = model.fit(x_train, d_train, batch_size=batch_size, epochs=epochs, verbose=1, validatio
n_data=(x_test, d_test))

#Accuracy
plt.plot(history.history['acc'])
```

```
plt.plot(history.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
# plt.ylim(0, 1.0)
plt.show()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_3 (Conv2D)            (None, 26, 26, 32)        320       
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 24, 24, 64)        18496     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 64)        0         
_________________________________________________________________
dropout_27 (Dropout)         (None, 12, 12, 64)        0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 9216)              0         
_________________________________________________________________
dense_60 (Dense)             (None, 128)               1179776   
_________________________________________________________________
dropout_28 (Dropout)         (None, 128)               0         
_________________________________________________________________
dense_61 (Dense)             (None, 10)                1290      
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
_________________________________________________________________
Train on 1000 samples, validate on 1000 samples
Epoch 1/6
1000/1000 [==============================] - 7s 7ms/step - loss: 1.8847 - acc: 0.3
910 - val_loss: 1.1695 - val_acc: 0.6850
Epoch 2/6
1000/1000 [==============================] - 5s 5ms/step - loss: 0.9351 - acc: 0.7
060 - val_loss: 0.6664 - val_acc: 0.7810
Epoch 3/6
1000/1000 [==============================] - 5s 5ms/step - loss: 0.6129 - acc: 0.8
000 - val_loss: 0.5399 - val_acc: 0.8290
Epoch 4/6
1000/1000 [==============================] - 5s 5ms/step - loss: 0.4703 - acc: 0.8
440 - val_loss: 0.4293 - val_acc: 0.8510
Epoch 5/6
1000/1000 [==============================] - 5s 5ms/step - loss: 0.3989 - acc: 0.8
850 - val_loss: 0.4045 - val_acc: 0.8740
Epoch 6/6
1000/1000 [==============================] - 5s 5ms/step - loss: 0.2980 - acc: 0.9
130 - val_loss: 0.3398 - val_acc: 0.8900
```



cifar10

実行に時間がかかるため割愛

データセット cifar10
32x32ピクセルのカラー画像データ
10種のラベル「飛行機、自動車、鳥、猫、鹿、犬、蛙、馬、船、トラック」
トレーニングデータ数:50000, テストデータ数:10000
http://www.cs.toronto.edu/~kriz/cifar.html (http://www.cs.toronto.edu/~kriz/cifar.html)

In [38]:

```
#CIFAR-10のデータセットのインポート
from keras.datasets import cifar10
(x_train, d_train), (x_test, d_test) = cifar10.load_data()

#CIFAR-10の正規化
from keras.utils import to_categorical

# 特徴量の正規化
x_train = x_train/255.
x_test = x_test/255.

# クラスラベルの1-hotベクトル化
d_train = to_categorical(d_train, 10)
d_test = to_categorical(d_test, 10)

#実行時間を減らすために、データ数を削減。
x_train=x_train[:1000]
d_train=d_train[:1000]
x_test=x_test[:1000]
d_test=d_test[:1000]

# CNNの構築
import keras
from keras.models import Sequential
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.layers.core import Dense, Dropout, Activation, Flatten
import numpy as np
from keras.layers.normalization import BatchNormalization

model = Sequential()

model.add(Conv2D(32, (3, 3), padding='same',input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))
#model.add(BatchNormalization())#バッチ正規化
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(10))
model.add(Activation('softmax'))

# コンパイル
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])

#訓練
history = model.fit(x_train, d_train, epochs=5,validation_data=(x_test,d_test))
```

```
# モデルの保存
model.save('./CIFAR-10.h5')

#評価 & 評価結果出力
print(model.evaluate(x_test, d_test))
```

```
Train on 1000 samples, validate on 1000 samples
Epoch 1/5
1000/1000 [==============================] - 13s 13ms/step - loss: 2.2894 - acc:
0.1130 - val_loss: 2.2273 - val_acc: 0.1370
Epoch 2/5
1000/1000 [==============================] - 11s 11ms/step - loss: 2.0791 - acc:
0.2210 - val_loss: 1.9908 - val_acc: 0.2480
Epoch 3/5
1000/1000 [==============================] - 11s 11ms/step - loss: 1.9508 - acc:
0.2720 - val_loss: 1.8973 - val_acc: 0.3060
Epoch 4/5
1000/1000 [==============================] - 11s 11ms/step - loss: 1.8424 - acc:
0.3250 - val_loss: 1.8555 - val_acc: 0.3030
Epoch 5/5
1000/1000 [==============================] - 11s 11ms/step - loss: 1.7347 - acc:
0.3380 - val_loss: 1.7427 - val_acc: 0.3830
1000/1000 [==============================] - 3s 3ms/step
[1.742703842163086, 0.383]
```

In [35]:

```
model.load_weights('./CIFAR-10.h5')
model.evaluate(x_test,d_test)
```

```
1000/1000 [==============================] - 3s 3ms/step
```

Out[35]:

```
[1.8431919717788696, 0.299]
```

RNN

2進数足し算の予測

Keras RNNのドキュメント https://keras.io/ja/layers/recurrent/#simplernn (https://keras.io/ja/layers/recurrent/#simplernn)

In [14]:

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

import keras
from keras.models import Sequential
from keras.layers.core import Dense, Dropout,Activation
from keras.layers.wrappers import TimeDistributed
from keras.optimizers import SGD
from keras.layers.recurrent import SimpleRNN, LSTM, GRU


# データを用意
# 2進数の桁数
binary_dim = 8
# 最大値 + 1
largest_number = pow(2, binary_dim)

# largest_numberまで2進数を用意
binary = np.unpackbits(np.array([range(largest_number)], dtype=np.uint8).T,axis=1)[:, ::-1]


# A, B初期化 (a + b = d)
a_int = np.random.randint(largest_number/2, size=20000)
a_bin = binary[a_int] # binary encoding
b_int = np.random.randint(largest_number/2, size=20000)
b_bin = binary[b_int] # binary encoding

x_int = []
x_bin = []
for i in range(10000):
    x_int.append(np.array([a_int[i], b_int[i]]).T)
    x_bin.append(np.array([a_bin[i], b_bin[i]]).T)

x_int_test = []
x_bin_test = []
for i in range(10001, 20000):
    x_int_test.append(np.array([a_int[i], b_int[i]]).T)
    x_bin_test.append(np.array([a_bin[i], b_bin[i]]).T)

x_int = np.array(x_int)
x_bin = np.array(x_bin)
x_int_test = np.array(x_int_test)
x_bin_test = np.array(x_bin_test)


# 正解データ
d_int = a_int + b_int
d_bin = binary[d_int][0:10000]
d_bin_test = binary[d_int][10001:20000]

model = Sequential()

#隠れ層のユニット数16
#入力層の形状が[8,2]  ８タイムステップ、２データ/タイムステップ
#中間層の活性化関数がReLU
model.add(SimpleRNN(units=16,
               return_sequences=True,
```

```
                input_shape=[8, 2],
                go_backwards=False,
                activation='relu',
                # dropout=0.5,
                # recurrent_dropout=0.3,
                # unroll = True,
            ))
#出力層のユニット数が１
#出力層の活性化関数がsigmoid
#input_shape(？、２データ)
model.add(Dense(1, activation='sigmoid', input_shape=(-1,2)))
model.summary()
model.compile(loss='mean_squared_error', optimizer=SGD(lr=0.1), metrics=['accuracy'])
# model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])

history = model.fit(x_bin, d_bin.reshape(-1, 8, 1), epochs=5, batch_size=2)

# テスト結果出力
score = model.evaluate(x_bin_test, d_bin_test.reshape(-1,8,1), verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_7 (SimpleRNN)     (None, 8, 16)             304
_________________________________________________________________
dense_7 (Dense)              (None, 8, 1)              17
=================================================================
Total params: 321
Trainable params: 321
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
10000/10000 [==============================] - 12s 1ms/step - loss: 0.0972 - acc:
0.8954
Epoch 2/5
10000/10000 [==============================] - 15s 1ms/step - loss: 0.0035 - acc:
1.0000
Epoch 3/5
10000/10000 [==============================] - 16s 2ms/step - loss: 8.5375e-04 - a
cc: 1.0000
Epoch 4/5
10000/10000 [==============================] - 15s 2ms/step - loss: 4.4483e-04 - a
cc: 1.0000
Epoch 5/5
10000/10000 [==============================] - 13s 1ms/step - loss: 2.9167e-04 - a
cc: 1.0000
Test loss: 0.00024475360282742703
Test accuracy: 1.0
```

In [ ]:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
simple_rnn_7 (SimpleRNN)     (None, 8, 16)             304       (バッチサイズ不明、8タイムステップ、16隠れ層)
                                                                  入→隠の重み32（2×16）、バイアス16、隠→隠の重み256（16×16）
_________________________________________________________________
dense_7 (Dense)              (None, 8, 1)              17        （バッチサイズ不明、８タイムステップ、1出力層）
                                                                  隠→出の重み16、バイアス1
=================================================================
Total params: 321
Trainable params: 321
Non-trainable params: 0
_________________________________________________________________
```

In [3]:

```
a_int[0], b_int[0], d_int[0]
```

Out[3]:

```
(20, 10, 30)
```

In [4]:

```
a_bin[0], b_bin[0], d_bin[0]
```

Out[4]:

```
(array([0, 0, 1, 0, 1, 0, 0, 0], dtype=uint8),
 array([0, 1, 0, 1, 0, 0, 0, 0], dtype=uint8),
 array([0, 1, 1, 1, 1, 0, 0, 0], dtype=uint8))
```

In [8]:

```
zzz1=np.dot(np.array([0, 0, 1, 0, 1, 0, 0, 0]),np.array([2**0, 2**1, 2**2, 2**3, 2**4, 2**5, 2**6, 2**7]))
zzz2=np.dot(np.array([0, 1, 0, 1, 0, 0, 0, 0]),np.array([2**0, 2**1, 2**2, 2**3, 2**4, 2**5, 2**6, 2**7]))
print(zzz1,zzz2)
```

```
20 10
```

[try]

- RNNの隠れ層のノード数を128に変更

In [10]:

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

import keras
from keras.models import Sequential
from keras.layers.core import Dense, Dropout,Activation
from keras.layers.wrappers import TimeDistributed
from keras.optimizers import SGD
from keras.layers.recurrent import SimpleRNN, LSTM, GRU


# データを用意
# 2進数の桁数
binary_dim = 8
# 最大値 + 1
largest_number = pow(2, binary_dim)

# largest_numberまで2進数を用意
binary = np.unpackbits(np.array([range(largest_number)], dtype=np.uint8).T,axis=1)[:, ::-1]


# A, B初期化 (a + b = d)
a_int = np.random.randint(largest_number/2, size=20000)
a_bin = binary[a_int] # binary encoding
b_int = np.random.randint(largest_number/2, size=20000)
b_bin = binary[b_int] # binary encoding

x_int = []
x_bin = []
for i in range(10000):
    x_int.append(np.array([a_int[i], b_int[i]]).T)
    x_bin.append(np.array([a_bin[i], b_bin[i]]).T)

x_int_test = []
x_bin_test = []
for i in range(10001, 20000):
    x_int_test.append(np.array([a_int[i], b_int[i]]).T)
    x_bin_test.append(np.array([a_bin[i], b_bin[i]]).T)

x_int = np.array(x_int)
x_bin = np.array(x_bin)
x_int_test = np.array(x_int_test)
x_bin_test = np.array(x_bin_test)


# 正解データ
d_int = a_int + b_int
d_bin = binary[d_int][0:10000]
d_bin_test = binary[d_int][10001:20000]

model = Sequential()

model.add(SimpleRNN(units=128,
               return_sequences=True,
               input_shape=[8, 2],
               go_backwards=False,
               activation='relu',
```

```
                # dropout=0.5,
                # recurrent_dropout=0.3,
                # unroll = True,
            ))
# 出力層
model.add(Dense(1, activation='sigmoid', input_shape=(-1,2)))
model.summary()
model.compile(loss='mean_squared_error', optimizer=SGD(lr=0.1), metrics=['accuracy'])
# model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])

history = model.fit(x_bin, d_bin.reshape(-1, 8, 1), epochs=5, batch_size=2)

# テスト結果出力
score = model.evaluate(x_bin_test, d_bin_test.reshape(-1,8,1), verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_3 (SimpleRNN)     (None, 8, 128)            16768
_________________________________________________________________
dense_3 (Dense)              (None, 8, 1)              129
=================================================================
Total params: 16,897
Trainable params: 16,897
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
10000/10000 [==============================] - 23s 2ms/step - loss: 0.0652 - acc:
0.9352
Epoch 2/5
10000/10000 [==============================] - 22s 2ms/step - loss: 0.0017 - acc:
1.0000
Epoch 3/5
10000/10000 [==============================] - 18s 2ms/step - loss: 6.6554e-04 - a
cc: 1.0000
Epoch 4/5
10000/10000 [==============================] - 19s 2ms/step - loss: 3.9610e-04 - a
cc: 1.0000
Epoch 5/5
10000/10000 [==============================] - 19s 2ms/step - loss: 2.7694e-04 - a
cc: 1.0000
Test loss: 0.00023822386813199272
Test accuracy: 1.0
```

In [ ]:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_3 (SimpleRNN)     (None, 8, 128)            16768     (バッチサイズ不明、８タイムス
テップ、128隠れ層)
                                                                  入→隠の重み256（2×128）、
_________________________________________________________________
バイアス128、隠→隠の重み16384（128×128）

dense_3 (Dense)              (None, 8, 1)              129       （バッチサイズ不明、８タイム
ステップ、1出力層)
                                                                  隠→出の重み128、バイアス1

=================================================================
Total params: 16,897                                              16768+129
Trainable params: 16,897
Non-trainable params: 0
_________________________________________________________________
```

[try]

- RNNの隠れ層の活性化関数を sigmoid に変更

In [11]:

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

import keras
from keras.models import Sequential
from keras.layers.core import Dense, Dropout,Activation
from keras.layers.wrappers import TimeDistributed
from keras.optimizers import SGD
from keras.layers.recurrent import SimpleRNN, LSTM, GRU


# データを用意
# 2進数の桁数
binary_dim = 8
# 最大値 + 1
largest_number = pow(2, binary_dim)

# largest_numberまで2進数を用意
binary = np.unpackbits(np.array([range(largest_number)], dtype=np.uint8).T,axis=1)[:, ::-1]


# A, B初期化 (a + b = d)
a_int = np.random.randint(largest_number/2, size=20000)
a_bin = binary[a_int] # binary encoding
b_int = np.random.randint(largest_number/2, size=20000)
b_bin = binary[b_int] # binary encoding

x_int = []
x_bin = []
for i in range(10000):
    x_int.append(np.array([a_int[i], b_int[i]]).T)
    x_bin.append(np.array([a_bin[i], b_bin[i]]).T)

x_int_test = []
x_bin_test = []
for i in range(10001, 20000):
    x_int_test.append(np.array([a_int[i], b_int[i]]).T)
    x_bin_test.append(np.array([a_bin[i], b_bin[i]]).T)

x_int = np.array(x_int)
x_bin = np.array(x_bin)
x_int_test = np.array(x_int_test)
x_bin_test = np.array(x_bin_test)


# 正解データ
d_int = a_int + b_int
d_bin = binary[d_int][0:10000]
d_bin_test = binary[d_int][10001:20000]

model = Sequential()

model.add(SimpleRNN(units=16,
               return_sequences=True,
               input_shape=[8, 2],
               go_backwards=False,
               activation='sigmoid',#隠れ層の活性化関数をsigmoidに変更
```

```
                # dropout=0.5,
                # recurrent_dropout=0.3,
                # unroll = True,
            ))
# 出力層
model.add(Dense(1, activation='sigmoid', input_shape=(-1,2)))
model.summary()
model.compile(loss='mean_squared_error', optimizer=SGD(lr=0.1), metrics=['accuracy'])
# model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])

history = model.fit(x_bin, d_bin.reshape(-1, 8, 1), epochs=5, batch_size=2)

# テスト結果出力
score = model.evaluate(x_bin_test, d_bin_test.reshape(-1,8,1), verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_4 (SimpleRNN)     (None, 8, 16)             304
_________________________________________________________________
dense_4 (Dense)              (None, 8, 1)              17
=================================================================
Total params: 321
Trainable params: 321
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
10000/10000 [==============================] - 12s 1ms/step - loss: 0.2506 - acc:
0.5007
Epoch 2/5
10000/10000 [==============================] - 11s 1ms/step - loss: 0.2497 - acc:
0.5150
Epoch 3/5
10000/10000 [==============================] - 12s 1ms/step - loss: 0.2480 - acc:
0.5456
Epoch 4/5
10000/10000 [==============================] - 10s 1ms/step - loss: 0.2398 - acc:
0.6102
Epoch 5/5
10000/10000 [==============================] - 10s 1ms/step - loss: 0.1788 - acc:
0.7998
Test loss: 0.10639868681654834
Test accuracy: 0.8990649064668048
```

[try]

- RNNの隠れ層の活性化関数を tanh に変更

In [18]:

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

import keras
from keras.models import Sequential
from keras.layers.core import Dense, Dropout,Activation
from keras.layers.wrappers import TimeDistributed
from keras.optimizers import SGD
from keras.layers.recurrent import SimpleRNN, LSTM, GRU


# データを用意
# 2進数の桁数
binary_dim = 8
# 最大値 + 1
largest_number = pow(2, binary_dim)

# largest_numberまで2進数を用意
binary = np.unpackbits(np.array([range(largest_number)], dtype=np.uint8).T,axis=1)[:, ::-1]


# A, B初期化 (a + b = d)
a_int = np.random.randint(largest_number/2, size=20000)
a_bin = binary[a_int] # binary encoding
b_int = np.random.randint(largest_number/2, size=20000)
b_bin = binary[b_int] # binary encoding

x_int = []
x_bin = []
for i in range(10000):
    x_int.append(np.array([a_int[i], b_int[i]]).T)
    x_bin.append(np.array([a_bin[i], b_bin[i]]).T)

x_int_test = []
x_bin_test = []
for i in range(10001, 20000):
    x_int_test.append(np.array([a_int[i], b_int[i]]).T)
    x_bin_test.append(np.array([a_bin[i], b_bin[i]]).T)

x_int = np.array(x_int)
x_bin = np.array(x_bin)
x_int_test = np.array(x_int_test)
x_bin_test = np.array(x_bin_test)


# 正解データ
d_int = a_int + b_int
d_bin = binary[d_int][0:10000]
d_bin_test = binary[d_int][10001:20000]

model = Sequential()

model.add(SimpleRNN(units=16,
               return_sequences=True,
               input_shape=[8, 2],
               go_backwards=False,
               activation='tanh',#隠れ層の活性化関数をtanhに変更
```

```
                # dropout=0.5,
                # recurrent_dropout=0.3,
                # unroll = True,
            ))
# 出力層
model.add(Dense(1, activation='sigmoid', input_shape=(-1,2)))
model.summary()
model.compile(loss='mean_squared_error', optimizer=SGD(lr=0.1), metrics=['accuracy'])
# model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])

history = model.fit(x_bin, d_bin.reshape(-1, 8, 1), epochs=5, batch_size=2)

# テスト結果出力
score = model.evaluate(x_bin_test, d_bin_test.reshape(-1,8,1), verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
simple_rnn_8 (SimpleRNN)     (None, 8, 16)             304       
_________________________________________________________________
dense_8 (Dense)              (None, 8, 1)              17        
=================================================================
Total params: 321
Trainable params: 321
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
10000/10000 [==============================] - 11s 1ms/step - loss: 0.1119 - acc:
0.8373
Epoch 2/5
10000/10000 [==============================] - 11s 1ms/step - loss: 0.0021 - acc:
1.0000
Epoch 3/5
10000/10000 [==============================] - 10s 984us/step - loss: 7.0375e-04 -
acc: 1.0000
Epoch 4/5
10000/10000 [==============================] - 10s 1ms/step - loss: 4.0114e-04 - a
cc: 1.0000
Epoch 5/5
10000/10000 [==============================] - 11s 1ms/step - loss: 2.7321e-04 - a
cc: 1.0000
Test loss: 0.00023464548112196086
Test accuracy: 1.0
```

[try]

- 最適化方法をadamに変更

In [19]:

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

import keras
from keras.models import Sequential
from keras.layers.core import Dense, Dropout,Activation
from keras.layers.wrappers import TimeDistributed
from keras.optimizers import SGD
from keras.layers.recurrent import SimpleRNN, LSTM, GRU


# データを用意
# 2進数の桁数
binary_dim = 8
# 最大値 + 1
largest_number = pow(2, binary_dim)

# largest_numberまで2進数を用意
binary = np.unpackbits(np.array([range(largest_number)], dtype=np.uint8).T,axis=1)[:, ::-1]


# A, B初期化 (a + b = d)
a_int = np.random.randint(largest_number/2, size=20000)
a_bin = binary[a_int] # binary encoding
b_int = np.random.randint(largest_number/2, size=20000)
b_bin = binary[b_int] # binary encoding

x_int = []
x_bin = []
for i in range(10000):
    x_int.append(np.array([a_int[i], b_int[i]]).T)
    x_bin.append(np.array([a_bin[i], b_bin[i]]).T)

x_int_test = []
x_bin_test = []
for i in range(10001, 20000):
    x_int_test.append(np.array([a_int[i], b_int[i]]).T)
    x_bin_test.append(np.array([a_bin[i], b_bin[i]]).T)

x_int = np.array(x_int)
x_bin = np.array(x_bin)
x_int_test = np.array(x_int_test)
x_bin_test = np.array(x_bin_test)


# 正解データ
d_int = a_int + b_int
d_bin = binary[d_int][0:10000]
d_bin_test = binary[d_int][10001:20000]

model = Sequential()

#隠れ層のユニット数16
#入力層の形状が[8,2]  ８タイムステップ、２データ/タイムステップ
#中間層の活性化関数がReLU
model.add(SimpleRNN(units=16,
               return_sequences=True,
```

```
                input_shape=[8, 2],
                go_backwards=False,
                activation='relu',
                # dropout=0.5,
                # recurrent_dropout=0.3,
                # unroll = True,
            ))
#出力層のユニット数が１
#出力層の活性化関数がsigmoid
#input_shape(？、２データ)
model.add(Dense(1, activation='sigmoid', input_shape=(-1,2)))
model.summary()
#model.compile(loss='mean_squared_error', optimizer=SGD(lr=0.1), metrics=['accuracy'])
model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])

history = model.fit(x_bin, d_bin.reshape(-1, 8, 1), epochs=5, batch_size=2)

# テスト結果出力
score = model.evaluate(x_bin_test, d_bin_test.reshape(-1,8,1), verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
simple_rnn_9 (SimpleRNN)     (None, 8, 16)             304       
_________________________________________________________________
dense_9 (Dense)              (None, 8, 1)              17        
=================================================================
Total params: 321
Trainable params: 321
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
10000/10000 [==============================] - 12s 1ms/step - loss: 0.0915 - acc:
0.8986
Epoch 2/5
10000/10000 [==============================] - 10s 1ms/step - loss: 0.0059 - acc:
1.0000
Epoch 3/5
10000/10000 [==============================] - 10s 1ms/step - loss: 1.2051e-04 - a
cc: 1.0000
Epoch 4/5
10000/10000 [==============================] - 11s 1ms/step - loss: 5.9374e-06 - a
cc: 1.0000
Epoch 5/5
10000/10000 [==============================] - 11s 1ms/step - loss: 4.0165e-07 - a
cc: 1.0000
Test loss: 8.514582141717867e-08
Test accuracy: 1.0
```

[try]

- RNNの入力 Dropout を0.5に設定

In [20]:

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

import keras
from keras.models import Sequential
from keras.layers.core import Dense, Dropout,Activation
from keras.layers.wrappers import TimeDistributed
from keras.optimizers import SGD
from keras.layers.recurrent import SimpleRNN, LSTM, GRU


# データを用意
# 2進数の桁数
binary_dim = 8
# 最大値 + 1
largest_number = pow(2, binary_dim)

# largest_numberまで2進数を用意
binary = np.unpackbits(np.array([range(largest_number)], dtype=np.uint8).T,axis=1)[:, ::-1]


# A, B初期化 (a + b = d)
a_int = np.random.randint(largest_number/2, size=20000)
a_bin = binary[a_int] # binary encoding
b_int = np.random.randint(largest_number/2, size=20000)
b_bin = binary[b_int] # binary encoding

x_int = []
x_bin = []
for i in range(10000):
    x_int.append(np.array([a_int[i], b_int[i]]).T)
    x_bin.append(np.array([a_bin[i], b_bin[i]]).T)

x_int_test = []
x_bin_test = []
for i in range(10001, 20000):
    x_int_test.append(np.array([a_int[i], b_int[i]]).T)
    x_bin_test.append(np.array([a_bin[i], b_bin[i]]).T)

x_int = np.array(x_int)
x_bin = np.array(x_bin)
x_int_test = np.array(x_int_test)
x_bin_test = np.array(x_bin_test)


# 正解データ
d_int = a_int + b_int
d_bin = binary[d_int][0:10000]
d_bin_test = binary[d_int][10001:20000]

model = Sequential()

#隠れ層のユニット数16
#入力層の形状が[8,2]  ８タイムステップ、２データ/タイムステップ
#中間層の活性化関数がReLU
model.add(SimpleRNN(units=16,
               return_sequences=True,
```

```
                input_shape=[8, 2],
                go_backwards=False,
                activation='relu',
                dropout=0.5,
                # recurrent_dropout=0.3,
                # unroll = True,
            ))
#出力層のユニット数が1
#出力層の活性化関数がsigmoid
#input_shape(?、2データ)
model.add(Dense(1, activation='sigmoid', input_shape=(-1,2)))
model.summary()
model.compile(loss='mean_squared_error', optimizer=SGD(lr=0.1), metrics=['accuracy'])
#model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])

history = model.fit(x_bin, d_bin.reshape(-1, 8, 1), epochs=5, batch_size=2)

# テスト結果出力
score = model.evaluate(x_bin_test, d_bin_test.reshape(-1,8,1), verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
WARNING:tensorflow:From C:¥ProgramData¥Anaconda3¥lib¥site-packages¥keras¥backend¥t
ensorflow_backend.py:3445: calling dropout (from tensorflow.python.ops.nn_ops) wit
h 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`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_10 (SimpleRNN)    (None, 8, 16)             304
_________________________________________________________________
dense_10 (Dense)             (None, 8, 1)              17
=================================================================
Total params: 321
Trainable params: 321
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
10000/10000 [==============================] - 11s 1ms/step - loss: 0.2349 - acc:
0.5840
Epoch 2/5
10000/10000 [==============================] - 10s 1ms/step - loss: 0.2139 - acc:
0.6199
Epoch 3/5
10000/10000 [==============================] - 10s 1ms/step - loss: 0.2067 - acc:
0.6299
Epoch 4/5
10000/10000 [==============================] - 11s 1ms/step - loss: 0.2059 - acc:
0.6281
Epoch 5/5
10000/10000 [==============================] - 12s 1ms/step - loss: 0.2041 - acc:
0.6261
Test loss: 0.24452789137960493
Test accuracy: 0.5604935493787797
```

[try]

- RNNの再帰 Dropout を0.3に設定

In [21]:

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

import keras
from keras.models import Sequential
from keras.layers.core import Dense, Dropout,Activation
from keras.layers.wrappers import TimeDistributed
from keras.optimizers import SGD
from keras.layers.recurrent import SimpleRNN, LSTM, GRU


# データを用意
# 2進数の桁数
binary_dim = 8
# 最大値 + 1
largest_number = pow(2, binary_dim)

# largest_numberまで2進数を用意
binary = np.unpackbits(np.array([range(largest_number)], dtype=np.uint8).T,axis=1)[:, ::-1]


# A, B初期化 (a + b = d)
a_int = np.random.randint(largest_number/2, size=20000)
a_bin = binary[a_int] # binary encoding
b_int = np.random.randint(largest_number/2, size=20000)
b_bin = binary[b_int] # binary encoding

x_int = []
x_bin = []
for i in range(10000):
    x_int.append(np.array([a_int[i], b_int[i]]).T)
    x_bin.append(np.array([a_bin[i], b_bin[i]]).T)

x_int_test = []
x_bin_test = []
for i in range(10001, 20000):
    x_int_test.append(np.array([a_int[i], b_int[i]]).T)
    x_bin_test.append(np.array([a_bin[i], b_bin[i]]).T)

x_int = np.array(x_int)
x_bin = np.array(x_bin)
x_int_test = np.array(x_int_test)
x_bin_test = np.array(x_bin_test)


# 正解データ
d_int = a_int + b_int
d_bin = binary[d_int][0:10000]
d_bin_test = binary[d_int][10001:20000]

model = Sequential()

#隠れ層のユニット数16
#入力層の形状が[8,2]　８タイムステップ、２データ/タイムステップ
#中間層の活性化関数がReLU
model.add(SimpleRNN(units=16,
               return_sequences=True,
```

```
                input_shape=[8, 2],
                go_backwards=False,
                activation='relu',
                #dropout=0.5,
                recurrent_dropout=0.3,
                # unroll = True,
            ))
#出力層のユニット数が１
#出力層の活性化関数がsigmoid
#input_shape(？、２データ)
model.add(Dense(1, activation='sigmoid', input_shape=(-1,2)))
model.summary()
model.compile(loss='mean_squared_error', optimizer=SGD(lr=0.1), metrics=['accuracy'])
#model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])

history = model.fit(x_bin, d_bin.reshape(-1, 8, 1), epochs=5, batch_size=2)

# テスト結果出力
score = model.evaluate(x_bin_test, d_bin_test.reshape(-1,8,1), verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_11 (SimpleRNN)    (None, 8, 16)             304
_________________________________________________________________
dense_11 (Dense)             (None, 8, 1)              17
=================================================================
Total params: 321
Trainable params: 321
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
10000/10000 [==============================] - 12s 1ms/step - loss: 0.1424 - acc:
0.8147
Epoch 2/5
10000/10000 [==============================] - 11s 1ms/step - loss: 0.0438 - acc:
0.9619
Epoch 3/5
10000/10000 [==============================] - 11s 1ms/step - loss: 0.0281 - acc:
0.9736
Epoch 4/5
10000/10000 [==============================] - 11s 1ms/step - loss: 0.0221 - acc:
0.9788
Epoch 5/5
10000/10000 [==============================] - 11s 1ms/step - loss: 0.0208 - acc:
0.9794
Test loss: 0.008070644402498528
Test accuracy: 0.9912491249244134
```

[try]

- RNNのunrollをTrueに設定

In [22]:

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

import keras
from keras.models import Sequential
from keras.layers.core import Dense, Dropout,Activation
from keras.layers.wrappers import TimeDistributed
from keras.optimizers import SGD
from keras.layers.recurrent import SimpleRNN, LSTM, GRU


# データを用意
# 2進数の桁数
binary_dim = 8
# 最大値 + 1
largest_number = pow(2, binary_dim)

# largest_numberまで2進数を用意
binary = np.unpackbits(np.array([range(largest_number)], dtype=np.uint8).T,axis=1)[:, ::-1]


# A, B初期化 (a + b = d)
a_int = np.random.randint(largest_number/2, size=20000)
a_bin = binary[a_int] # binary encoding
b_int = np.random.randint(largest_number/2, size=20000)
b_bin = binary[b_int] # binary encoding

x_int = []
x_bin = []
for i in range(10000):
    x_int.append(np.array([a_int[i], b_int[i]]).T)
    x_bin.append(np.array([a_bin[i], b_bin[i]]).T)

x_int_test = []
x_bin_test = []
for i in range(10001, 20000):
    x_int_test.append(np.array([a_int[i], b_int[i]]).T)
    x_bin_test.append(np.array([a_bin[i], b_bin[i]]).T)

x_int = np.array(x_int)
x_bin = np.array(x_bin)
x_int_test = np.array(x_int_test)
x_bin_test = np.array(x_bin_test)


# 正解データ
d_int = a_int + b_int
d_bin = binary[d_int][0:10000]
d_bin_test = binary[d_int][10001:20000]

model = Sequential()

#隠れ層のユニット数16
#入力層の形状が[8,2]  ８タイムステップ、２データ/タイムステップ
#中間層の活性化関数がReLU
model.add(SimpleRNN(units=16,
               return_sequences=True,
```

```
                input_shape=[8, 2],
                go_backwards=False,
                activation='relu',
                # dropout=0.5,
                # recurrent_dropout=0.3,
                unroll = True,
            ))
#出力層のユニット数が１
#出力層の活性化関数がsigmoid
#input_shape(?、２データ)
model.add(Dense(1, activation='sigmoid', input_shape=(-1,2)))
model.summary()
model.compile(loss='mean_squared_error', optimizer=SGD(lr=0.1), metrics=['accuracy'])
#model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])

history = model.fit(x_bin, d_bin.reshape(-1, 8, 1), epochs=5, batch_size=2)

# テスト結果出力
score = model.evaluate(x_bin_test, d_bin_test.reshape(-1,8,1), verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_12 (SimpleRNN)    (None, 8, 16)             304
_________________________________________________________________
dense_12 (Dense)             (None, 8, 1)              17
=================================================================
Total params: 321
Trainable params: 321
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
10000/10000 [==============================] - 9s 917us/step - loss: 0.0863 - acc:
0.8928
Epoch 2/5
10000/10000 [==============================] - 7s 740us/step - loss: 0.0017 - acc:
1.0000
Epoch 3/5
10000/10000 [==============================] - 7s 715us/step - loss: 6.4742e-04 -
acc: 1.0000
Epoch 4/5
10000/10000 [==============================] - 7s 706us/step - loss: 3.8425e-04 -
acc: 1.0000
Epoch 5/5
10000/10000 [==============================] - 7s 735us/step - loss: 2.6872e-04 -
acc: 1.0000
Test loss: 0.00023335137011170403
Test accuracy: 1.0
```

In [ ]:

```
-----------------------------------------------
### [try]
-  RNNの出力ノード数を128に変更
-  RNNの出力活性化関数を sigmoid に変更
-  RNNの出力活性化関数を tanh に変更
-  最適化方法をadamに変更
-  RNNの入力 Dropout を0.5に設定
-  RNNの再帰 Dropout を0.3に設定
-  RNNのunrollをTrueに設定

-----------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------
```