医療とAI・ビッグデータ応用

MLP②

本スライドは、自由にお使いください。
使用した場合は、このQRコードからアンケートに回答をお願いします。

統合教育機構
須藤毅顕

```
result = model.fit(x_train, y_train, epochs=50, batch_size=64, verbose=1,    validation_split=0.2, shuffle=True)
```

```
Epoch 1/50 750/750 [==============================] - 2s 2ms/step - loss: 0.6172 - accuracy: 0.7892 - val_loss: 0.4628 - val_accuracy: 0.8407
Epoch 2/50 750/750 [==============================] - 1s 2ms/step - loss: 0.4376 - accuracy: 0.8482 - val_loss: 0.4198 - val_accuracy: 0.8545
Epoch 3/50 750/750 [==============================] - 1s 2ms/step - loss: 0.4035 - accuracy: 0.8605 - val_loss: 0.4151 - val_accuracy: 0.8565
Epoch 4/50 750/750 [==============================] - 1s 2ms/step - loss: 0.3793 - accuracy: 0.8673 - val_loss: 0.3926 - val_accuracy: 0.8601
Epoch 5/50 750/750 [==============================] - 1s 2ms/step - loss: 0.3628 - accuracy: 0.8721 - val_loss: 0.3776 - val_accuracy: 0.8664
                                      .
                                      .
Epoch 47/50 750/750 [==============================] - 1s 2ms/step - loss: 0.1602 - accuracy: 0.9409 - val_loss: 0.4712 - val_accuracy: 0.8773
Epoch 48/50 750/750 [==============================] - 1s 2ms/step - loss: 0.1564 - accuracy: 0.9427 - val_loss: 0.4850 - val_accuracy: 0.8735
Epoch 49/50 750/750 [==============================] - 1s 2ms/step - loss: 0.1570 - accuracy: 0.9425 - val_loss: 0.4780 - val_accuracy: 0.8767
Epoch 50/50 750/750 [==============================] - 1s 2ms/step - loss: 0.1542 - accuracy: 0.9434 - val_loss: 0.5010 - val_accuracy: 0.8724
```

60000枚 → 48000枚 / 12000枚

64枚ずつ取り出して学習
64×750 = 48000
→750回重みを更新

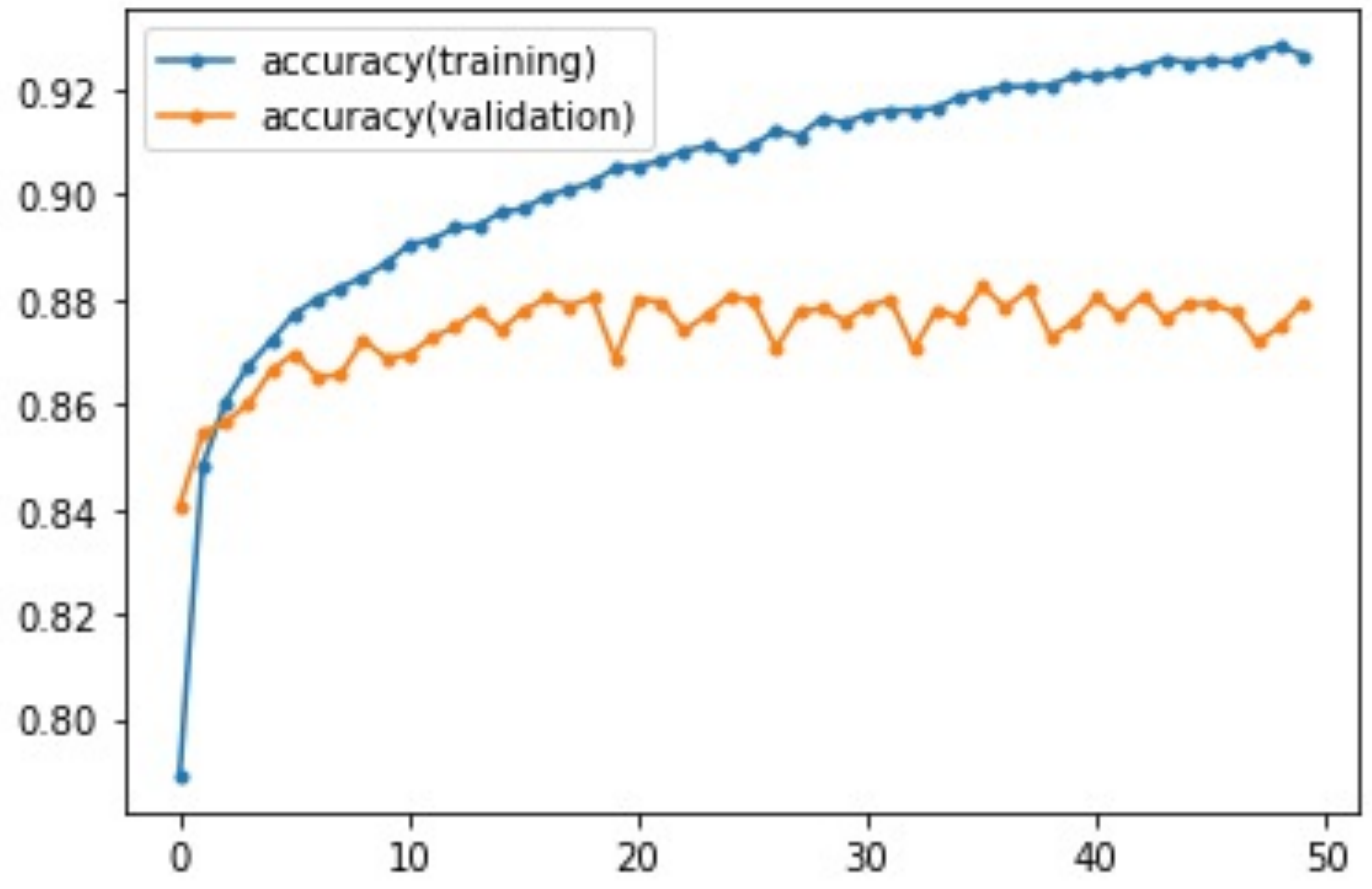
↓

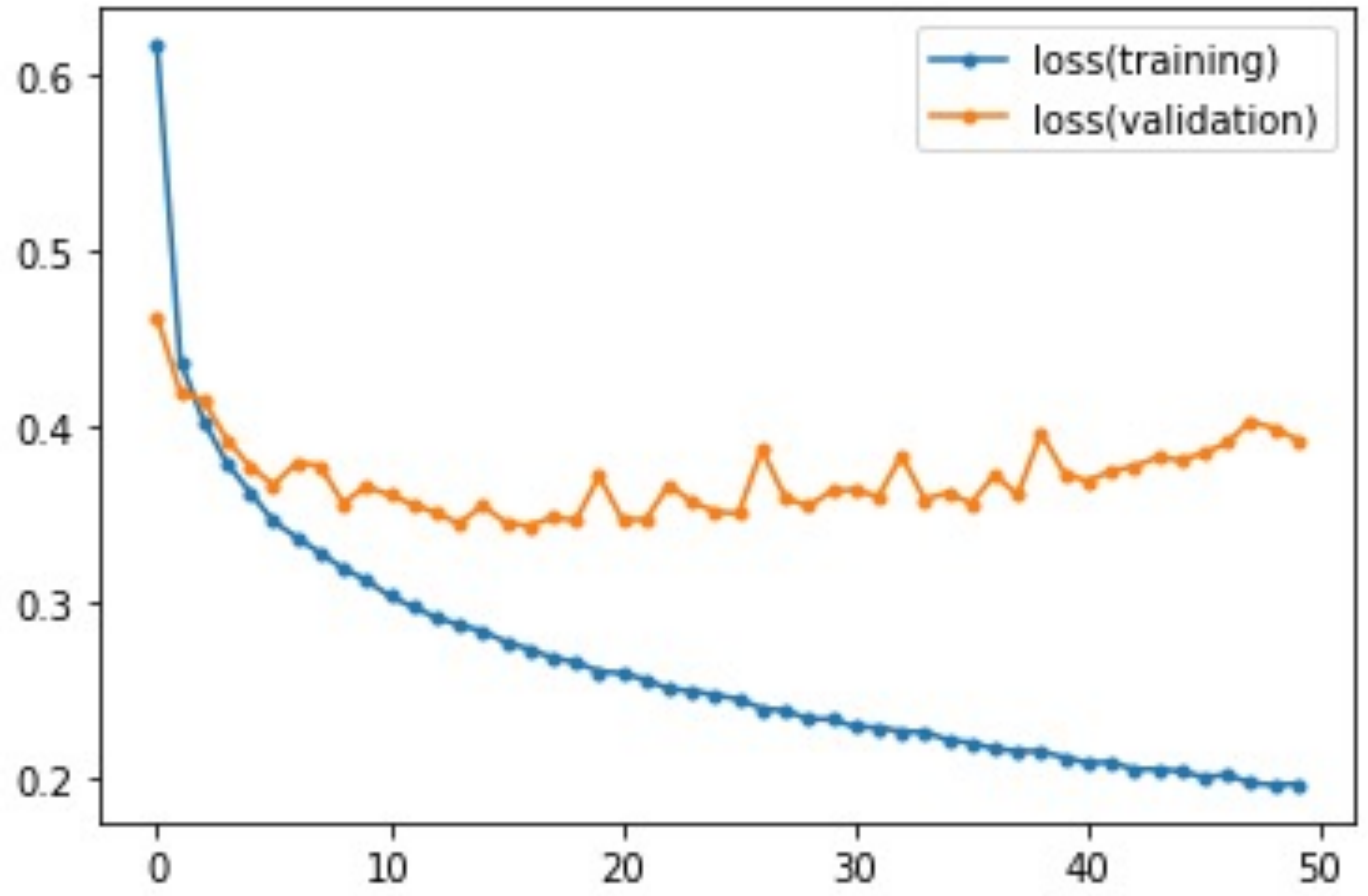
12000枚の画像で検証

この作業を
50回繰り返す

結果の作図

```
import matplotlib.pyplot as plt
plt.plot(result.history['loss'], marker='.', label='loss(training)')
plt.plot(result.history['val_loss'], marker='.', label='loss(validation)')
plt.legend()
plt.show()
plt.plot(result.history['accuracy'], marker='.', label='accuracy(training)')
plt.plot(result.history['val_accuracy'], marker='.', label='accuracy(validation)')
plt.legend()
plt.show()
```





作図について

print(result.history)

```
{'loss': [0.6407680511474609, 0.43205487728118896, 0.3960464298725128, 0.37231454253196716, 0.3531425893306732, 0.3418952226638794, 0.3289151191711426,
0.31962519884109497, 0.30994951725006104, 0.30448371171951294, 0.2966049909591675, 0.2925708293914795, 0.285235196352005, 0.2824288308620453, 0.2758510410785675,
0.2741311490535736, 0.26759836077690125, 0.2619331479072571, 0.25975126028060913, 0.25485366582870483, 0.25159457325935364, 0.2509157359600067, 0.24495647847652435,
0.24495787918567657, 0.23874390125274658, 0.23846033215522766, 0.23423752188682556, 0.23313100636005402, 0.23063215613365173, 0.22683009505271912,
0.22243058681488037, 0.22219112515449524, 0.21908551454544067, 0.21668700873851776, 0.21586231887340546, 0.2144242376089096, 0.20987261831760406,
0.21000073850154877, 0.20688402652740479, 0.20396345853805542, 0.202382892370224, 0.20032840967178345, 0.1998140513896942, 0.19791573286056519, 0.19822660088539124,
0.19370360672473907, 0.1935003399848938, 0.19082102179527283, 0.18894101679325104, 0.1871000975370407],
 'accuracy': [0.7818958163261414, 0.8507708311080933, 0.8607708215713501, 0.8679583072662354, 0.8738541603088379, 0.8764791488647461, 0.8806874752044678,
0.8846874833106995, 0.8880833387374878, 0.8889791369438171, 0.8920208215713501, 0.8934583067893982, 0.8970416784286499, 0.8958125114440918, 0.8988958597183228,
0.8995000123977661, 0.9016249775886536, 0.9033958315849304, 0.9054999947547913, 0.906000018119812, 0.9069583415985107, 0.9070208072662354, 0.909500002861023,
0.9091249704360962, 0.9125208258628845, 0.9121875166893005, 0.9140625, 0.9133124947547913, 0.9148333072662354, 0.9163749814033508, 0.917395830154419,
0.9182708263397217, 0.9196458458900452, 0.9202499985694885, 0.9198750257492065, 0.9193750023841858, 0.921916663646698, 0.9225833415985107, 0.9239583611488342,
0.9244583249092102, 0.925208330154419, 0.925083339214325, 0.925083339214325, 0.9260416626930237, 0.926437497138977, 0.9287291765213013, 0.9288958311080933,
0.9292708039283752, 0.9307083487510681, 0.9314374923706055],
'val_loss': [0.47452786564826965, 0.41108548641204834, 0.4009964168071747, 0.3757151961326599, 0.37599340081214905, 0.36084845662117004, 0.361476331949234,
0.36045193672180176, 0.35319408774375916, 0.35784009099006653, 0.3581872284412384, 0.33637475967407227, 0.34890124201774597, 0.33941298723220825,
0.36372897028923035, 0.3423093259334564, 0.34483370184898376, 0.34261226654052734, 0.3480249047279358, 0.34918493032455444, 0.3549243211746216, 0.3424607217311859,
0.354319304227829, 0.36245837807655334, 0.34716251492500305, 0.3522222936153412, 0.35628339648246765, 0.3469776213169098, 0.35598990321159363, 0.36482152342796326,
0.36312034726142883, 0.3614174425601959, 0.35187089443206787, 0.35411563515663147, 0.3598410189151764, 0.3613741397857666, 0.3786758482456207, 0.361969530582428,
0.37253522872924805, 0.37296152114868164, 0.37992843985557556, 0.38681110739707947, 0.38001614809036255, 0.40297767519950867, 0.3726571798324585,
0.3766944110393524, 0.38012608885765076, 0.3803871273994446, 0.382973313331604, 0.3888922929763794],
'val_accuracy': [0.8349166512489319, 0.8532500267028809, 0.8616666793823242, 0.8682500123977661, 0.8664166927337646, 0.8725000023841858, 0.8692499995231628,
0.8735833168029785, 0.8762500286102295, 0.8736666440963745, 0.8679166436195374, 0.8805833458900452, 0.8770833611488342, 0.8803333044052124, 0.8727499842643738,
0.8790000081062317, 0.8770833611488342, 0.8823333382606506, 0.8795833587646484, 0.8806666731834412, 0.8792499899864197, 0.8837500214576721, 0.8807500004768372,
0.8763333559036255, 0.8841666579246521, 0.8816666603088379, 0.8797500133514404, 0.8841666579246521, 0.8814166784286499, 0.8811666369438171, 0.8786666393280029,
0.8788333535194397, 0.8848333358764648, 0.8842499852180481, 0.8796666860580444, 0.8836666941642761, 0.8801666498184204, 0.8830833435058594, 0.8824999928474426,
0.8813333511352539, 0.8794166445732117, 0.8784166574478149, 0.8809999823570251, 0.8755833506584167, 0.8820833563804626, 0.8839166760444641, 0.8811666369438171,
0.8830833435058594, 0.8820000290870667, 0.8829166889190674]}
```

{'loss':[1回目の学習用データの損失,2回目の学習用データの損失,...,50回目の学習用データの損失],
'accuracy':[1回目の学習用データの正解率,2回目の学習用データの正解率,...,50回目の学習用データの正解率],
'val_loss':[1回目の検証用データの損失,2回目の検証用データの損失,...,50回目の検証用データの損失],
'val_accuracy':[1回目の検証用データの正解率,2回目の検証用データの正解率,...,50回目の検証用データの正解率]}

作図について

print(result.history[‘loss’])

```
[0.6407680511474609, 0.43205487728118896, 0.3960464298725128, 0.37231454253196716, 0.3531425893306732, 0.3418952226638794, 0.328915119171426,
0.31962519884109497, 0.30994951725006104, 0.30448371171951294, 0.2966049909591675, 0.2925708293914795, 0.285235196352005, 0.2824288308620453,
0.2758510410785675, 0.2741311490535736, 0.26759836077690125, 0.2619331479072571, 0.2597512602806913, 0.25485366582870483, 0.25159457325935364,
0.2509157359600067, 0.24495647847652435, 0.24495787918567657, 0.23874390125274658, 0.23846033215522766, 0.23423752188682556, 0.23313100636005402,
0.23063215613365173, 0.22683009505271912, 0.22243058681488037, 0.22219112515449524, 0.21908551454544067, 0.21668700873851776, 0.21586231887340546,
0.2144242376089096, 0.20987261831760406, 0.21000073850154877, 0.20688402652740479, 0.20396345853805542, 0.202382892370224, 0.20032840967178345,
0.1998140513896942, 0.19791573286056519, 0.19822660088539124, 0.19370360672473907, 0.1935003399848938, 0.19082102179527283, 0.18894101679325104,
0.1871000975370407]
```

x = [要素,要素,...,要素]

a = [1,1,3,3,3]
これはリスト型
b = {‘name’:’sudo’,’age’:36}
これは辞書型と言われます。

x = {key:value,key:value,...,key:value}

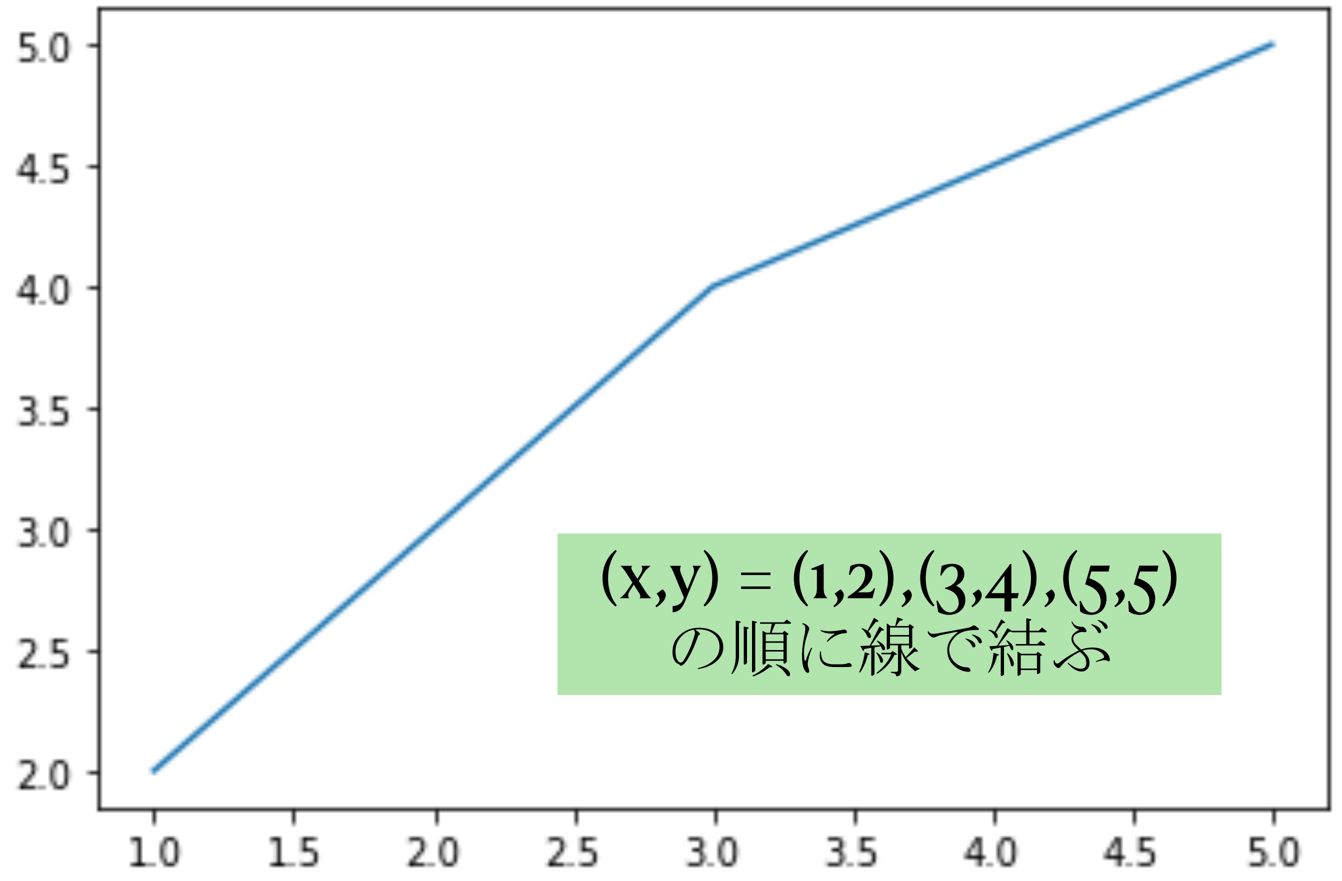
辞書型は変数名[key]でvalueを取り出せる!

```
b = {'name':'sudo','age':36}
print(type(b))
print(b['name'])
print(b['age'])
```

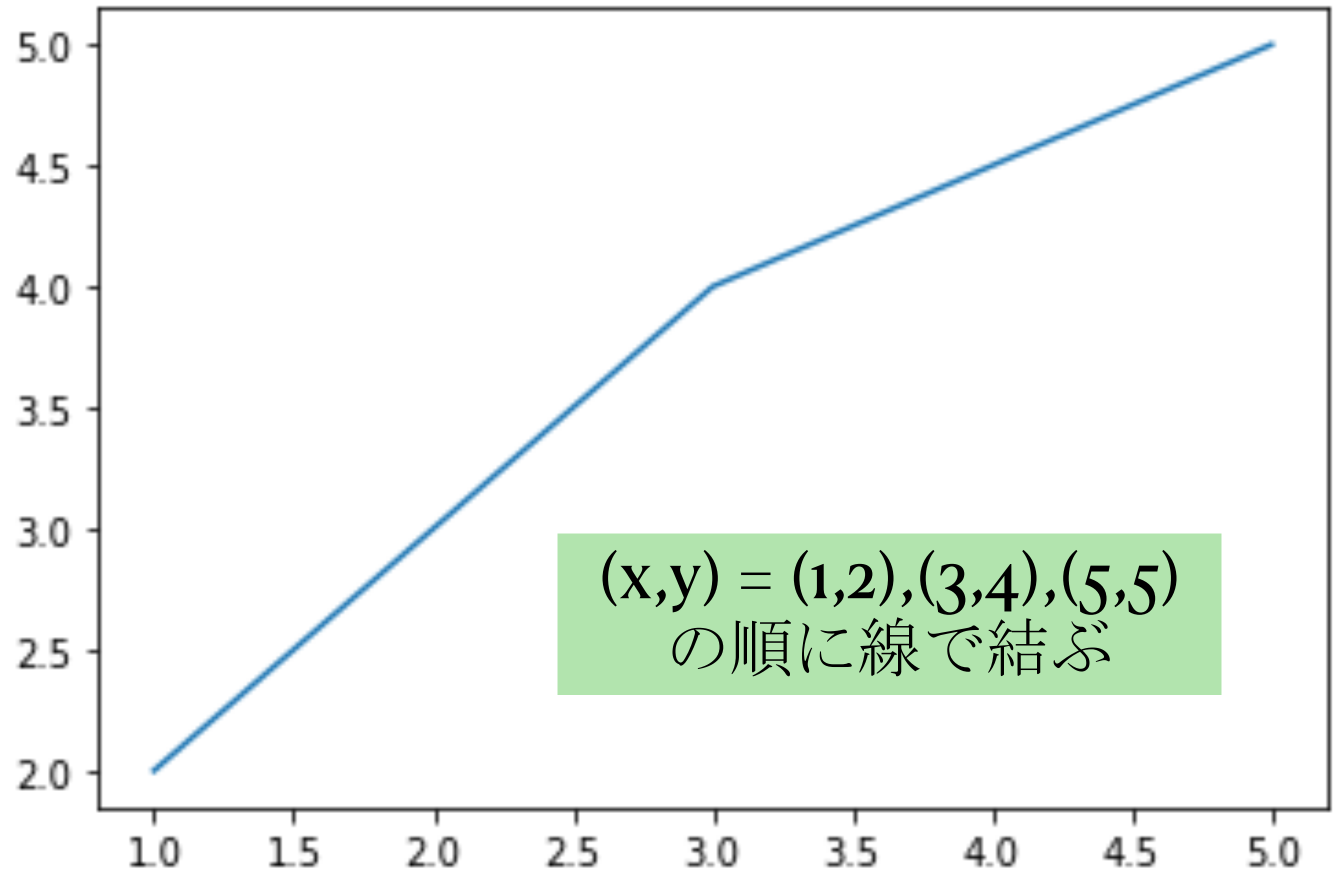
```
<class 'dict'>
sudo
36
```

result.history[‘loss’]で{‘loss’:[〜〜], ...}の[〜〜]を取り出している

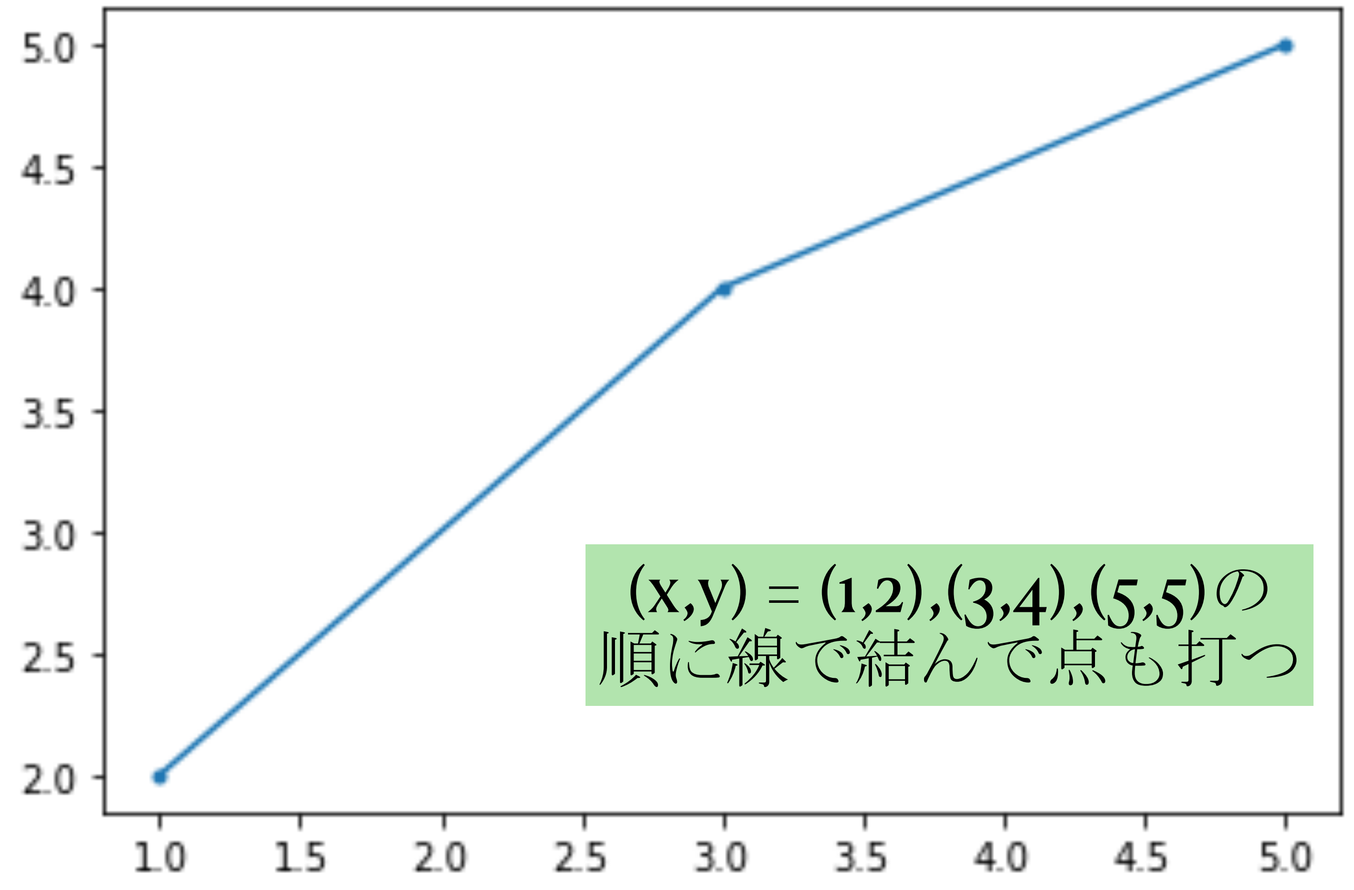
```
x = [1,3,5]
y = [2,4,5]
plt.plot(x,y)
plt.show()
```



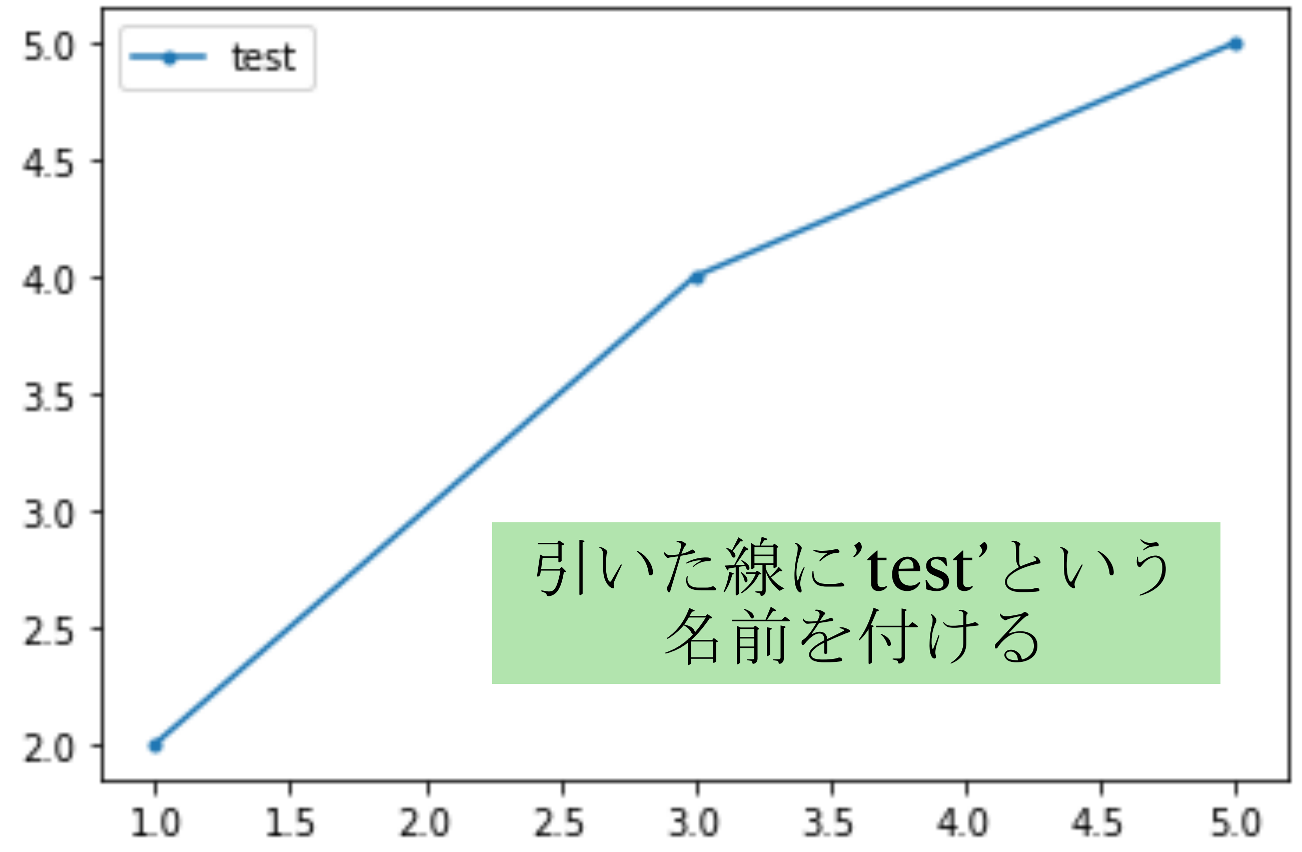
```
x = [1,3,5]
y = [2,4,5]
plt.plot(x,y)
plt.show()
```



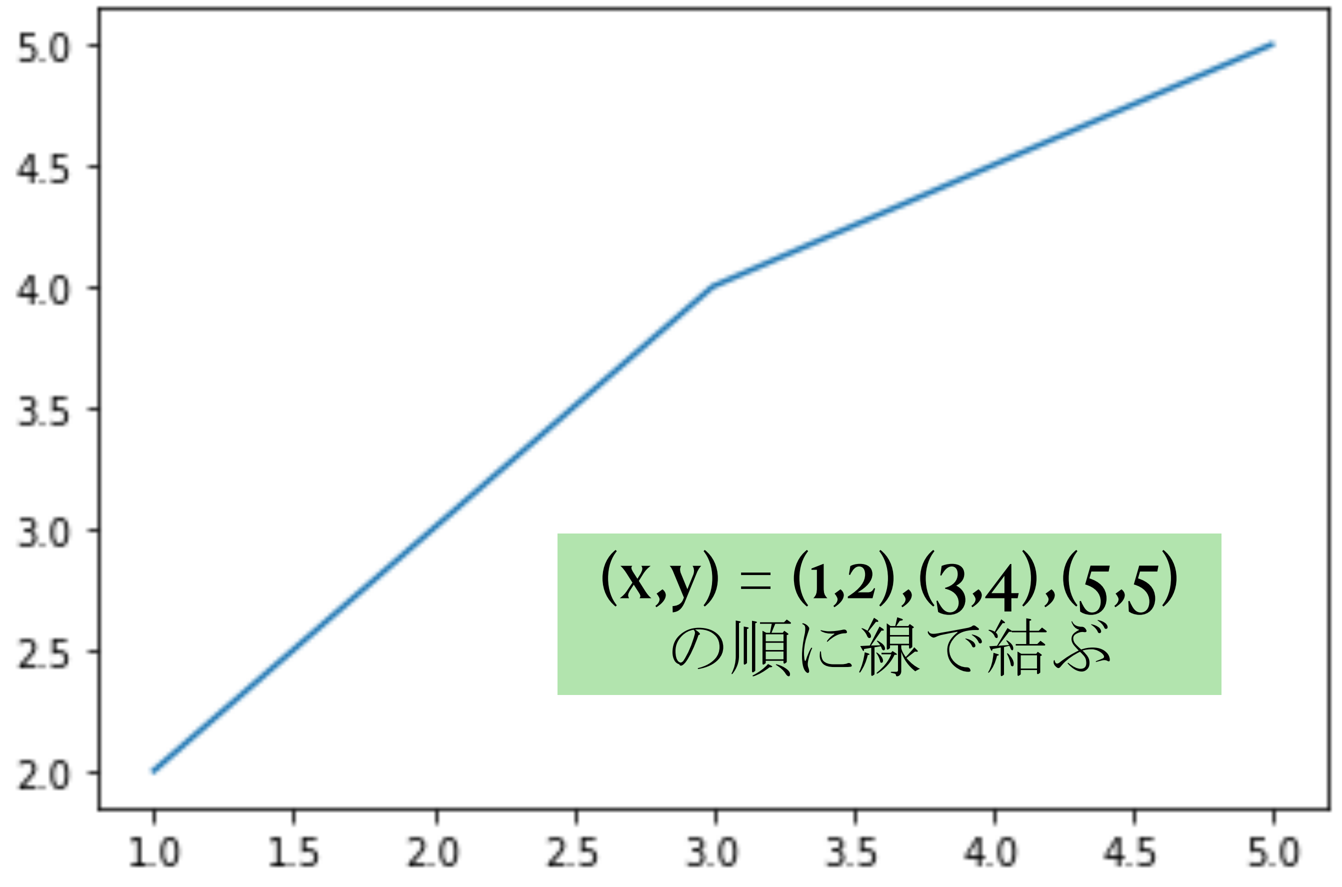

```
x = [1,3,5]
y = [2,4,5]
plt.plot(x,y,marker='.')
plt.show()
```



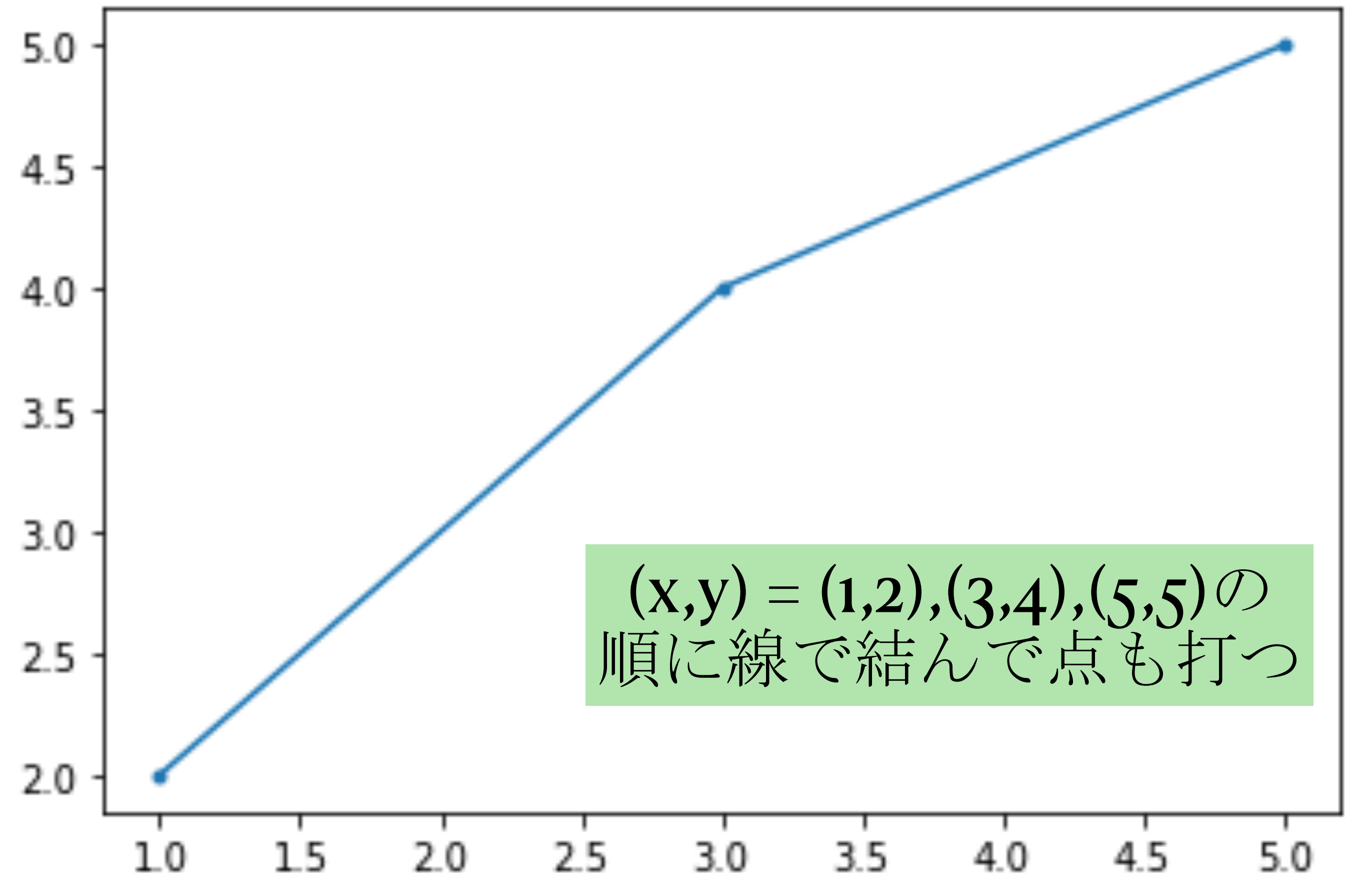

```
x = [1,3,5]
y = [2,4,5]
plt.plot(x,y,marker='.',label='test')
plt.legend()
plt.show()
```



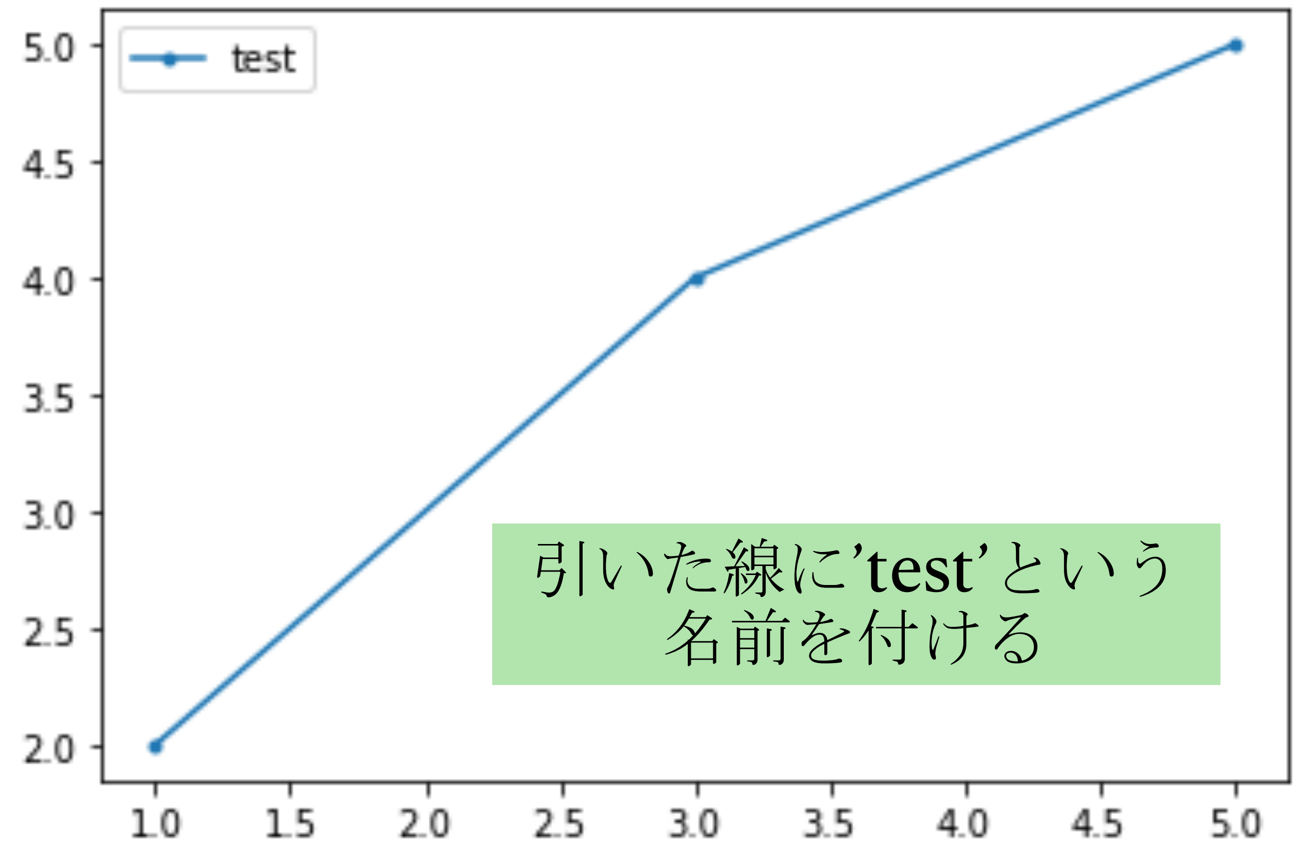
```
x = [1,3,5]
y = [2,4,5]
plt.plot(x,y)
plt.show()
```



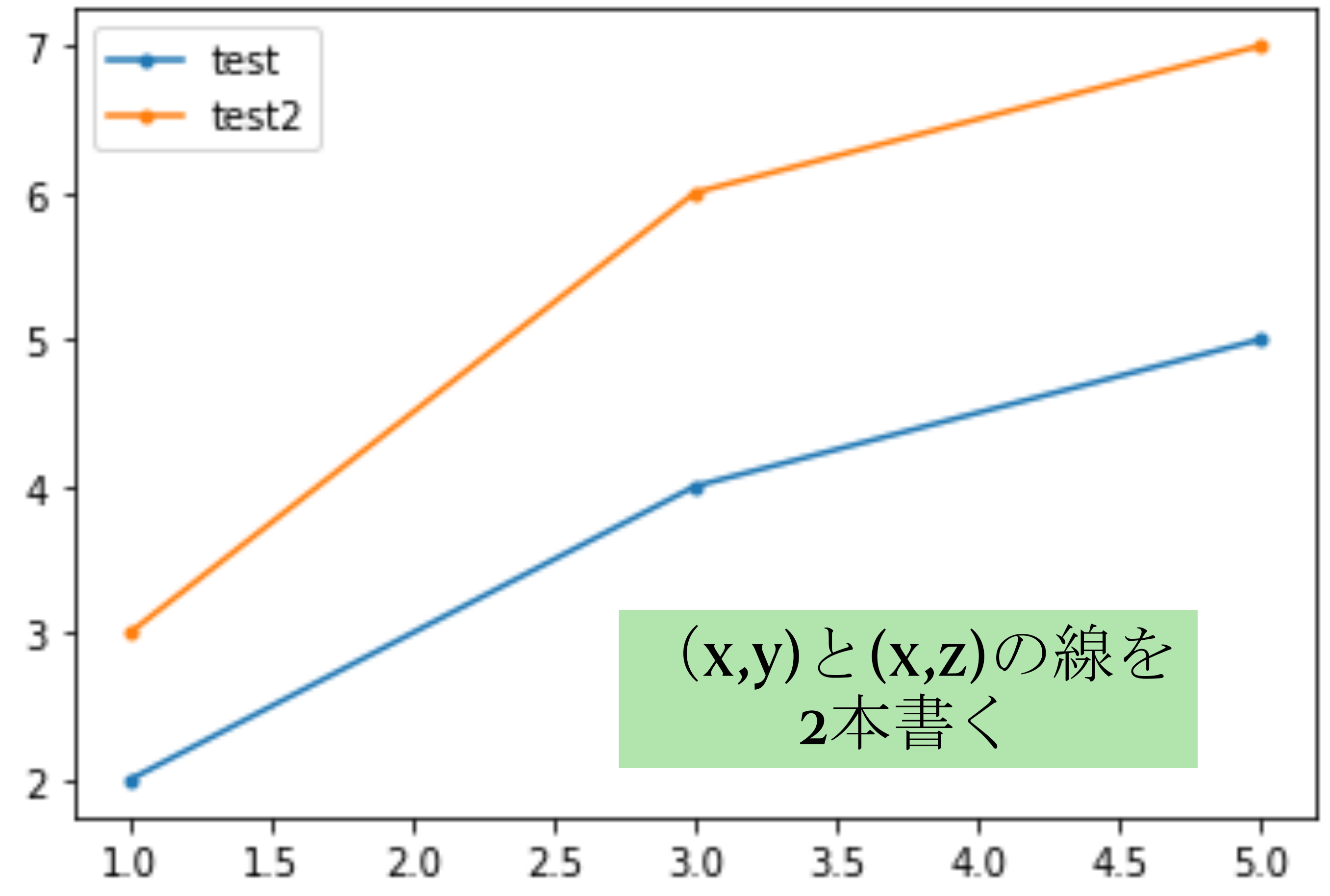

```
x = [1,3,5]
y = [2,4,5]
plt.plot(x,y,marker='.')
plt.show()
```



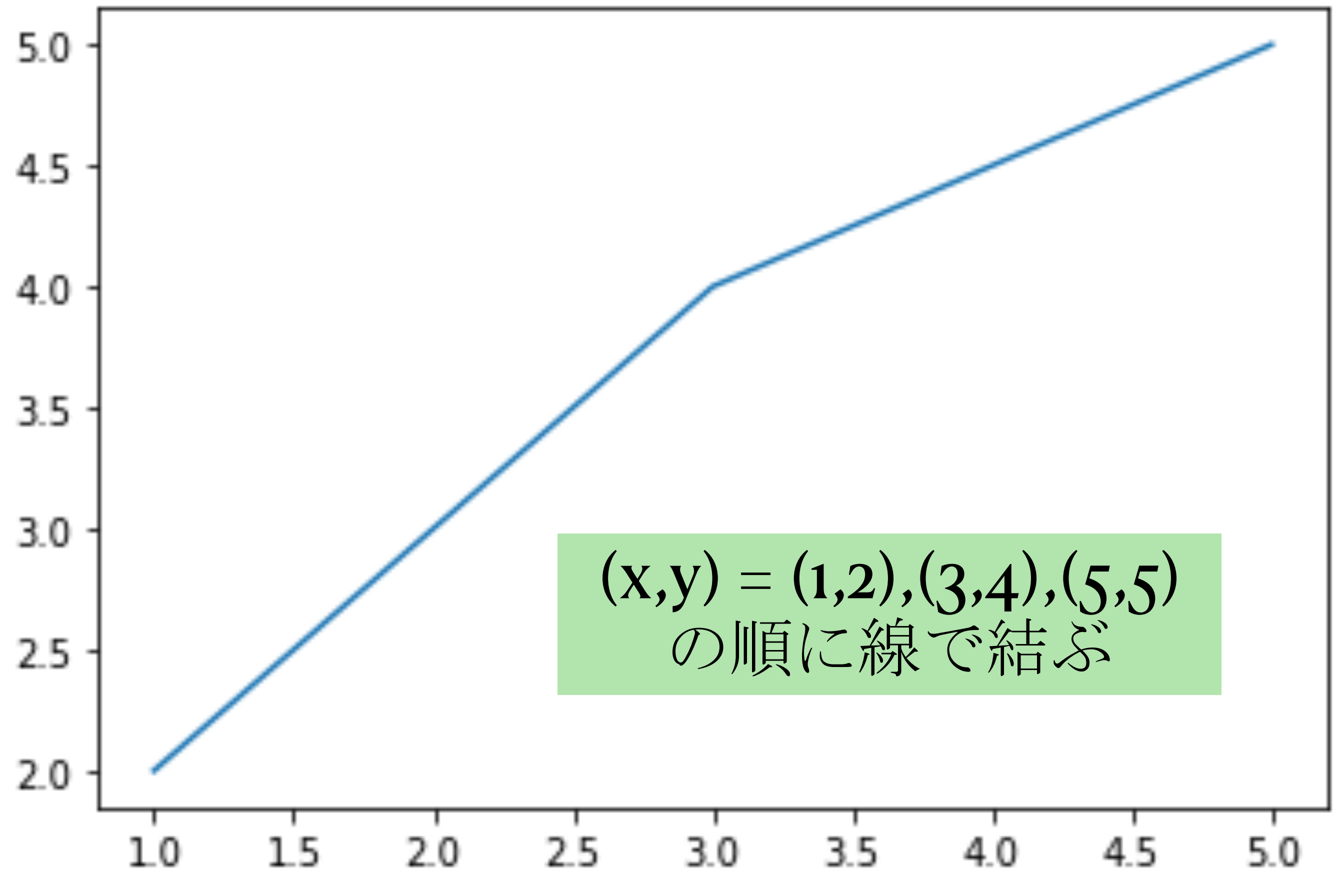

```
x = [1,3,5]
y = [2,4,5]
plt.plot(x,y,marker='.',label='test')
plt.legend()
plt.show()
```



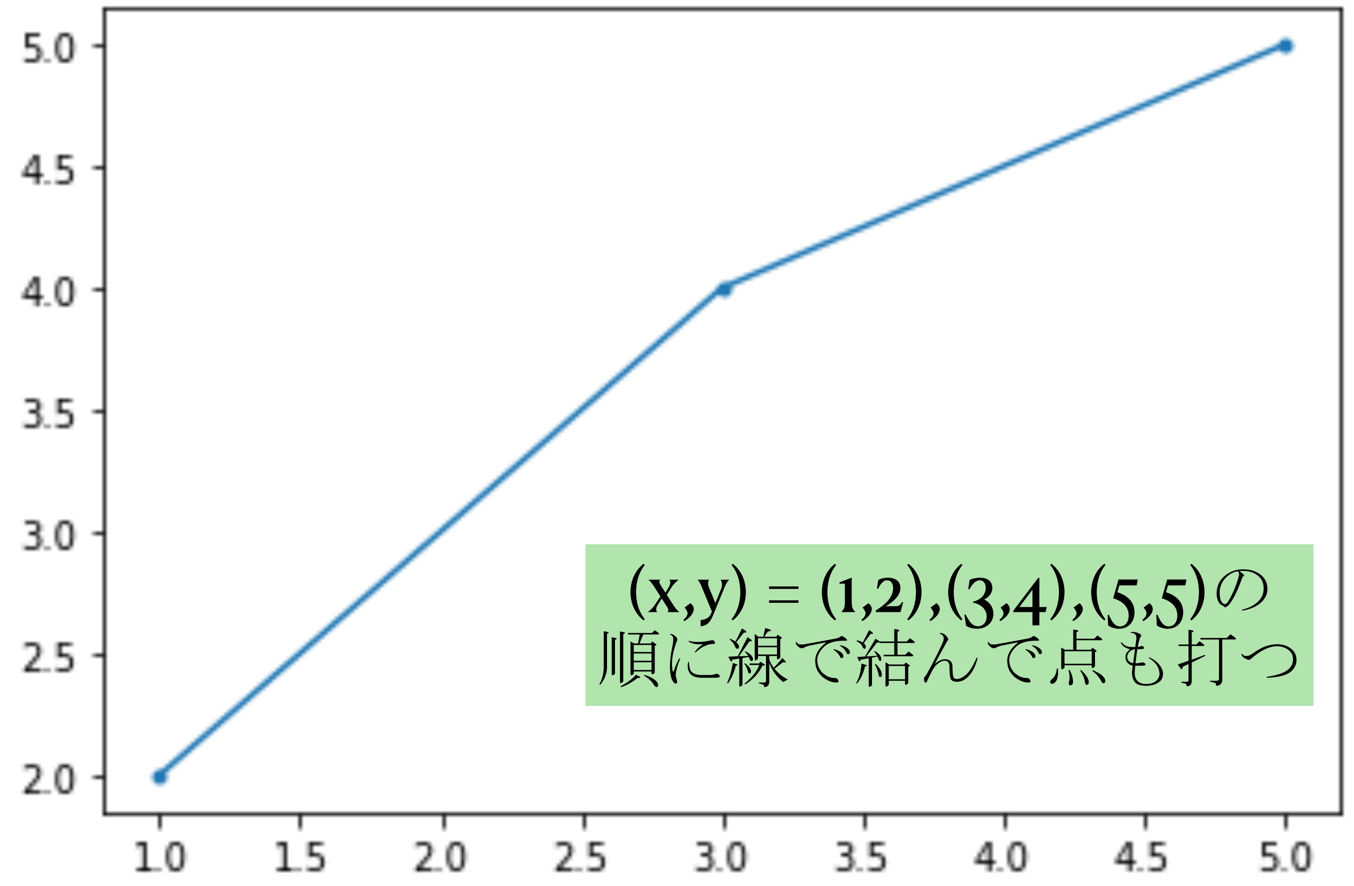

```
x = [1,3,5]
y = [2,4,5]
z = [3,6,7]
plt.plot(x,y,marker='.',label='test')
plt.plot(x,z,marker='.',label='test2')
plt.legend()
plt.show()
```



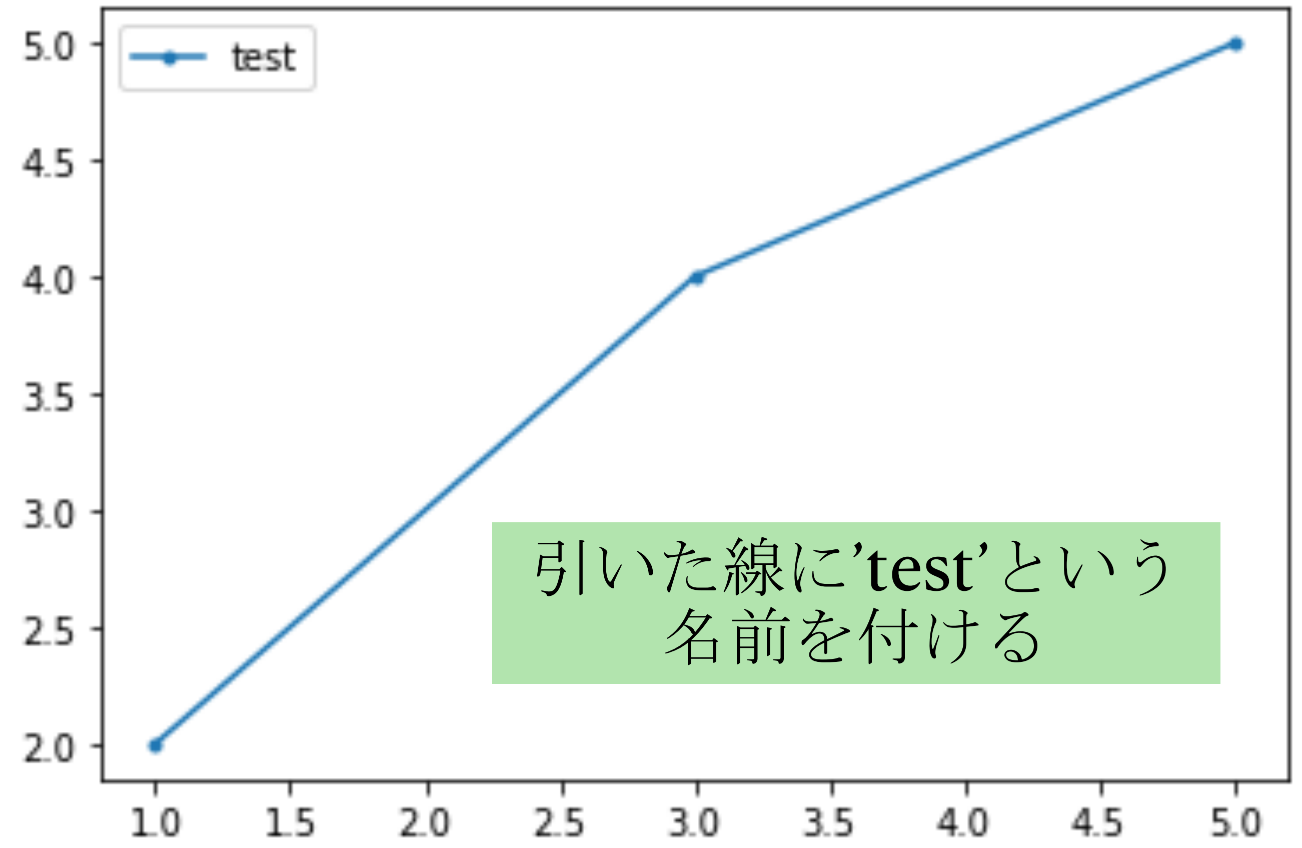
```
x = [1,3,5]
y = [2,4,5]
plt.plot(x,y)
plt.show()
```



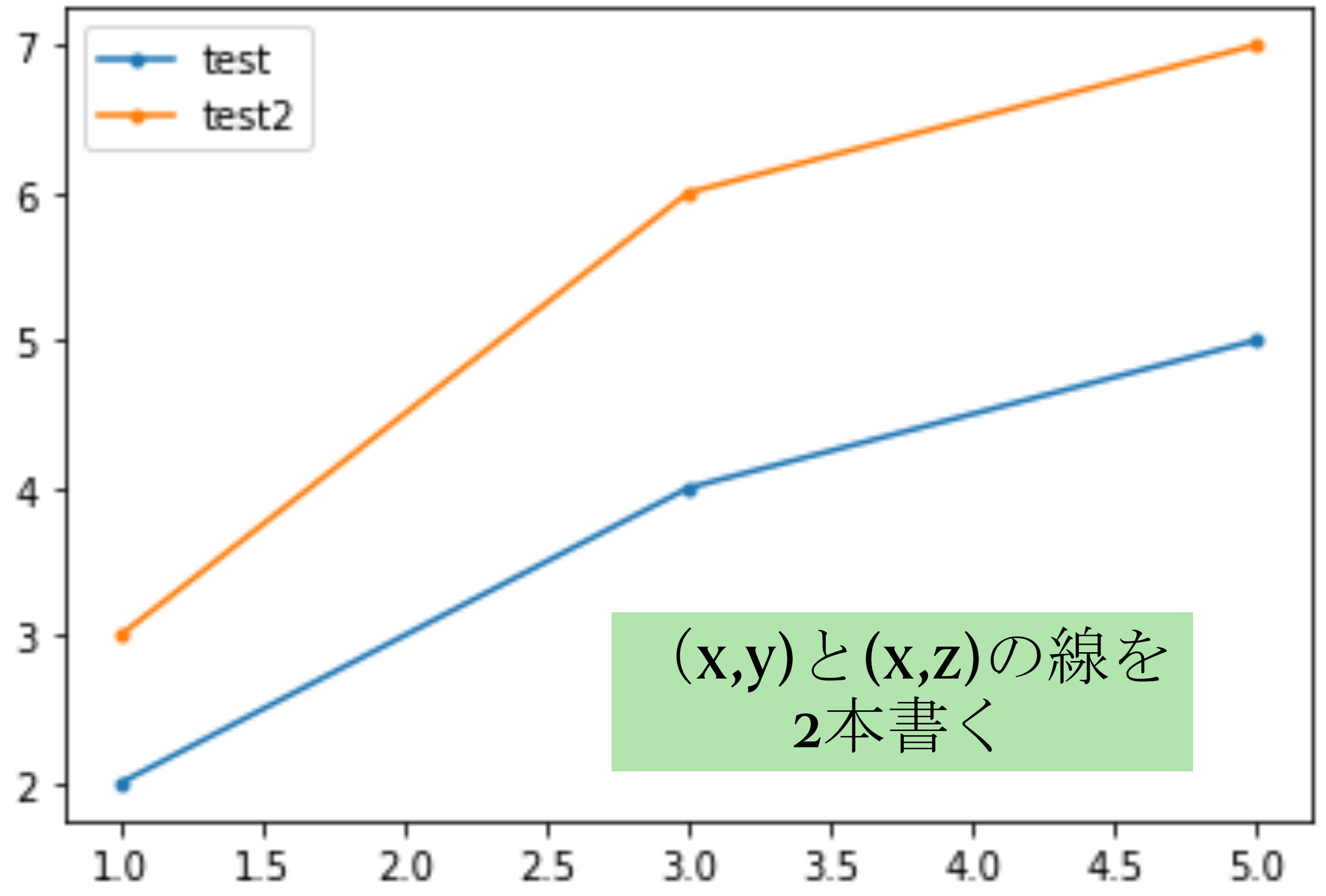

```
x = [1,3,5]
y = [2,4,5]
plt.plot(x,y,marker='.')
plt.show()
```



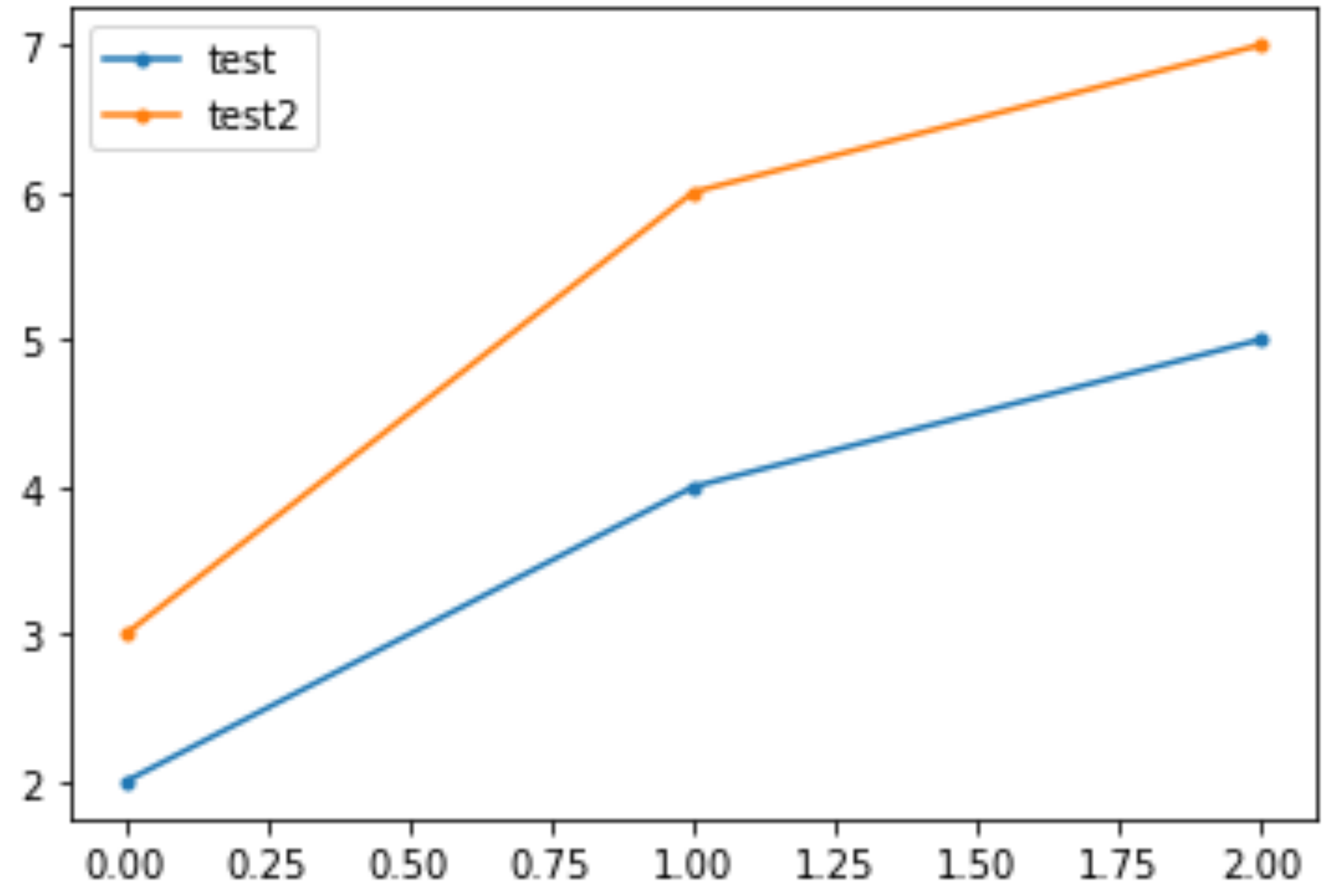

```
x = [1,3,5]
y = [2,4,5]
plt.plot(x,y,marker='.',label='test')
plt.legend()
plt.show()
```




```
x = [1,3,5]
y = [2,4,5]
z = [3,6,7]
plt.plot(x,y,marker='.',label='test')
plt.plot(x,z,marker='.',label='test2')
plt.legend()
plt.show()
```



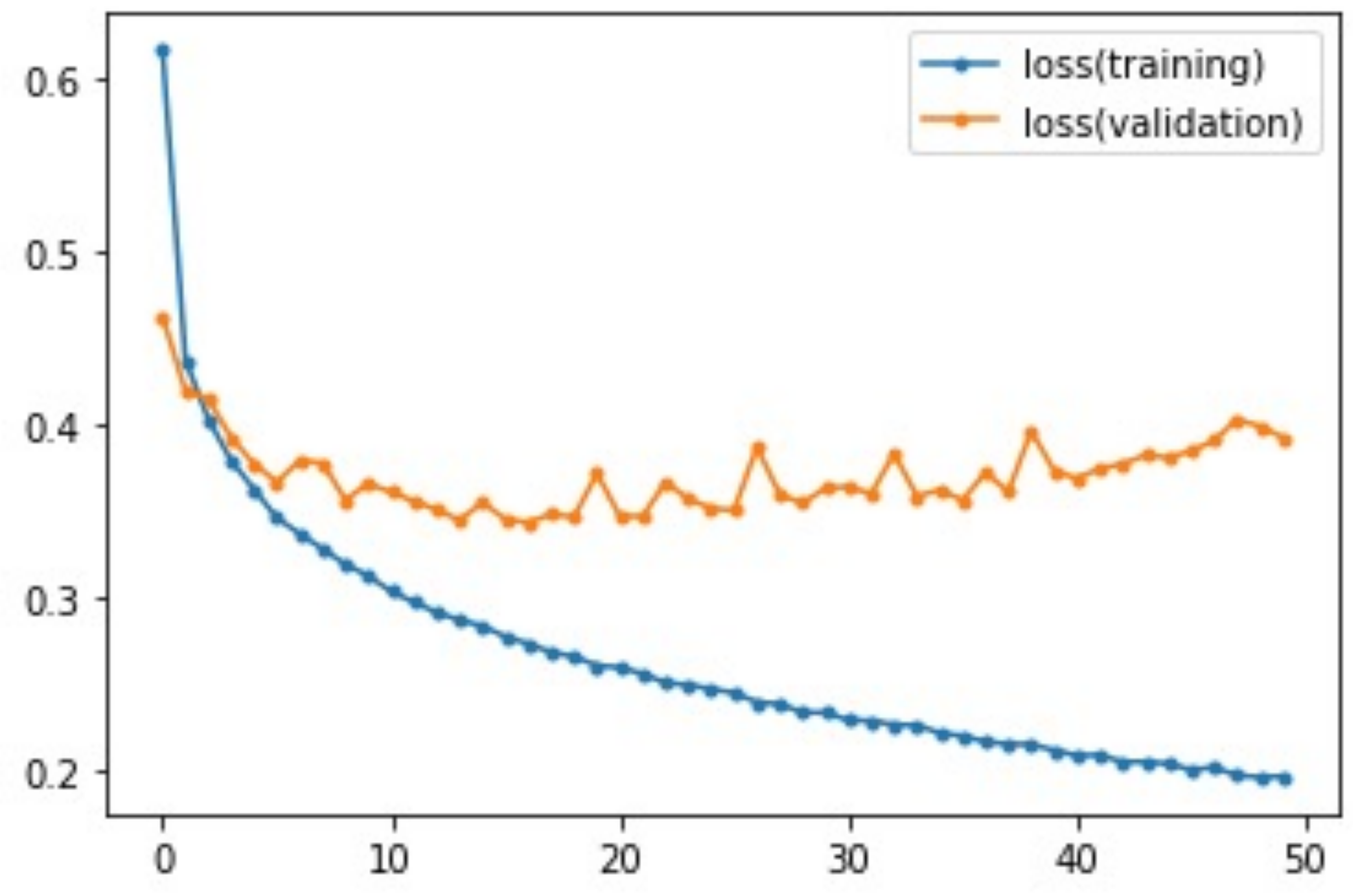

```
y = [2,4,5]
z = [3,6,7]
plt.plot(y,marker='.',label='test')
plt.plot(z,marker='.',label='test2')
plt.legend()
plt.show()
```



x軸の変数が与えられない時は**y**軸の個数だけ順に**1,2,3,..**と与えられる。

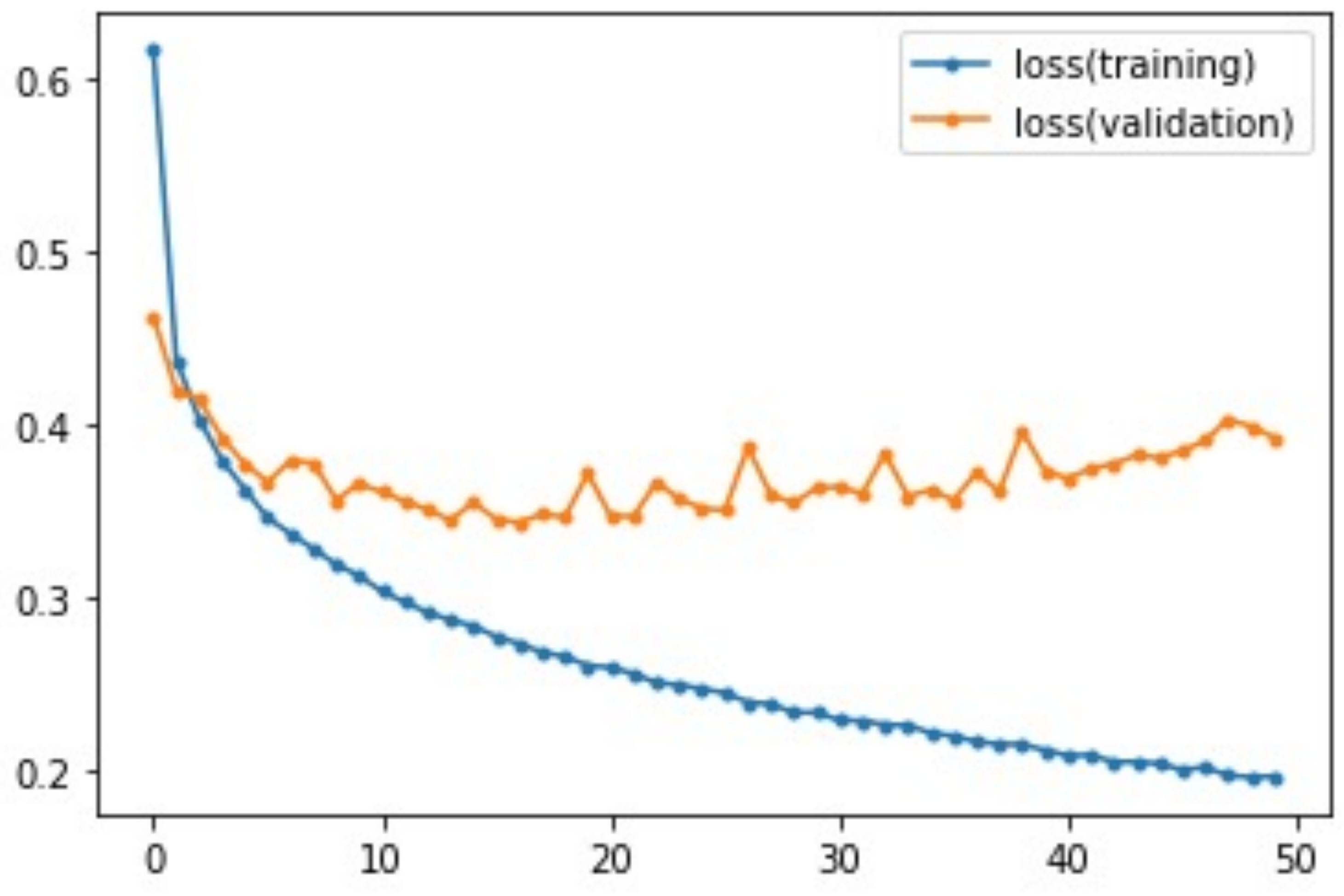
```
①import matplotlib.pyplot as plt
②plt.plot(result.history['loss'], marker='.', label='loss(training)‘)
③plt.plot(result.history['val_loss'], marker='.', label='loss(validation)‘)
④plt.legend()
⑤plt.show()
```

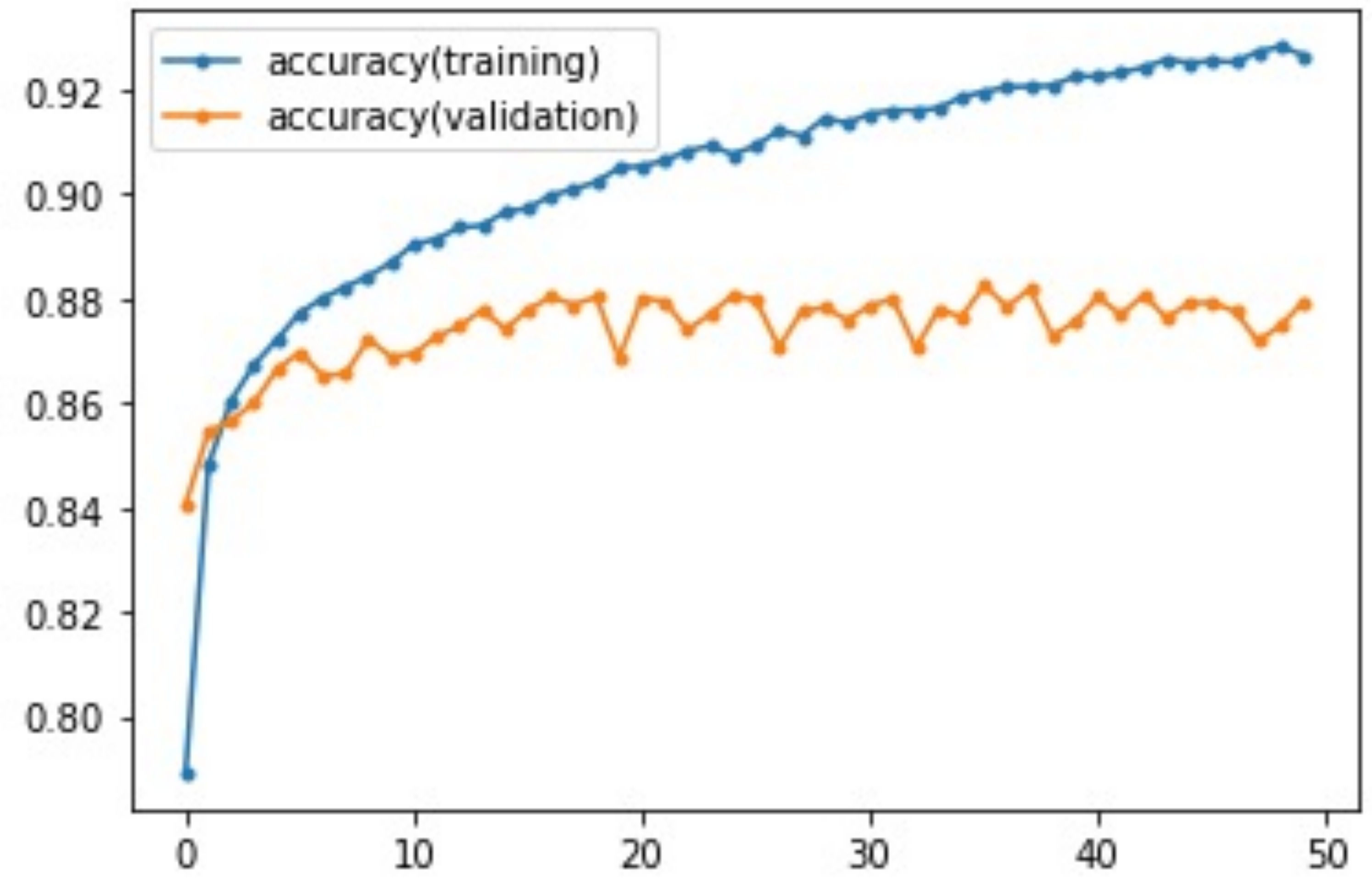
①：matplotlibの読み込み
②～⑤：損失の作図



```
①import matplotlib.pyplot as plt
②plt.plot(result.history['loss'], marker='.', label='loss(training)‘)
③plt.plot(result.history['val_loss'], marker='.', label='loss(validation)‘)
④plt.legend()
⑤plt.show()
⑥plt.plot(result.history['accuracy'], marker='.', label='accuracy(training)‘)
⑦plt.plot(result.history[’val_accuracy‘], marker=’.‘, label=’accuracy(validation)‘)
⑧plt.legend()
⑨plt.show()
```

①：matplotlibの読み込み
②～⑤：損失の作図
⑥～⑨：正解率の作図





```
score = model.evaluate(x_test, y_test)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

model.evaluate()で評価

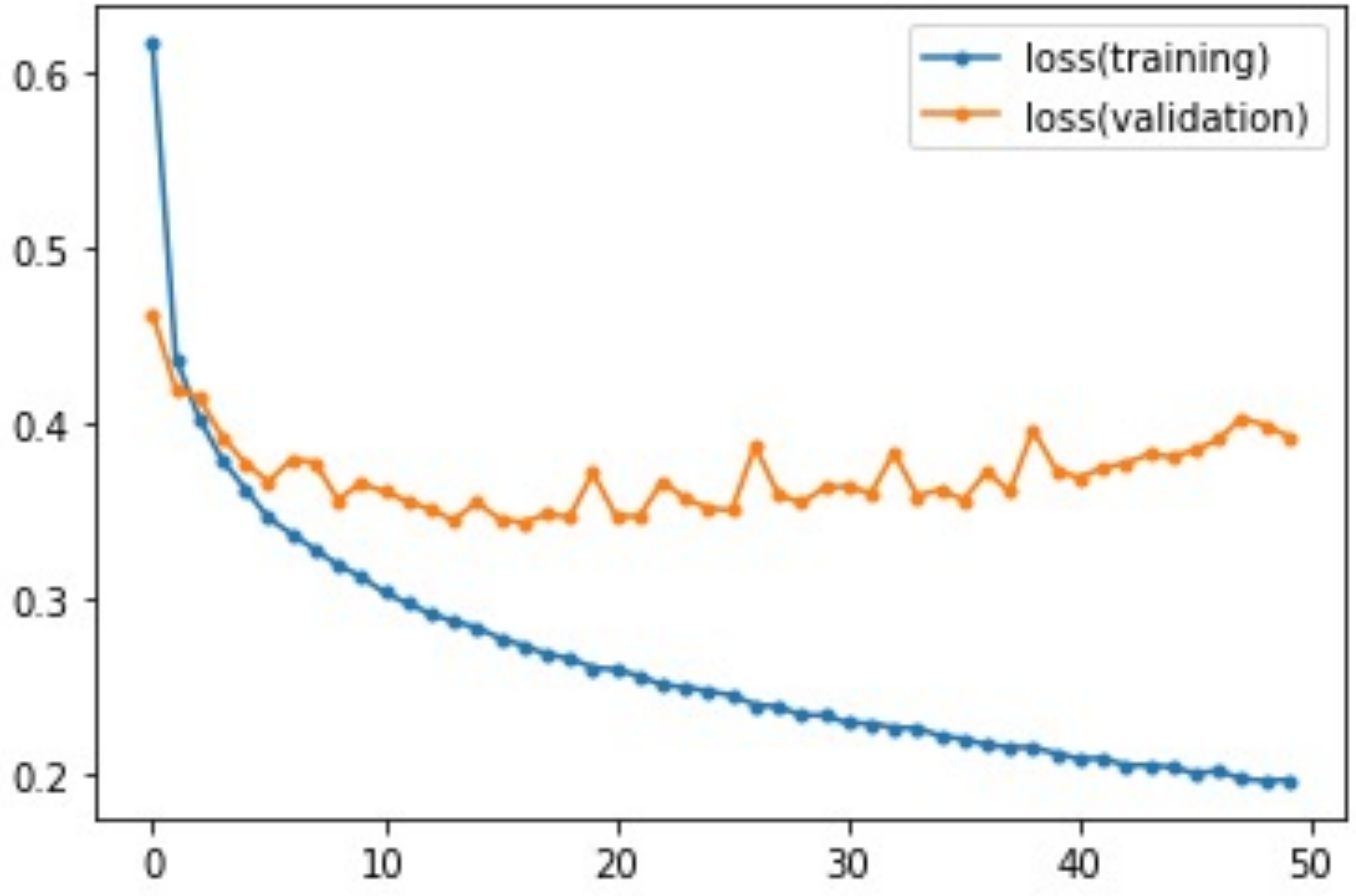
出来上がったモデルを別で用意している10000枚の画像で評価する

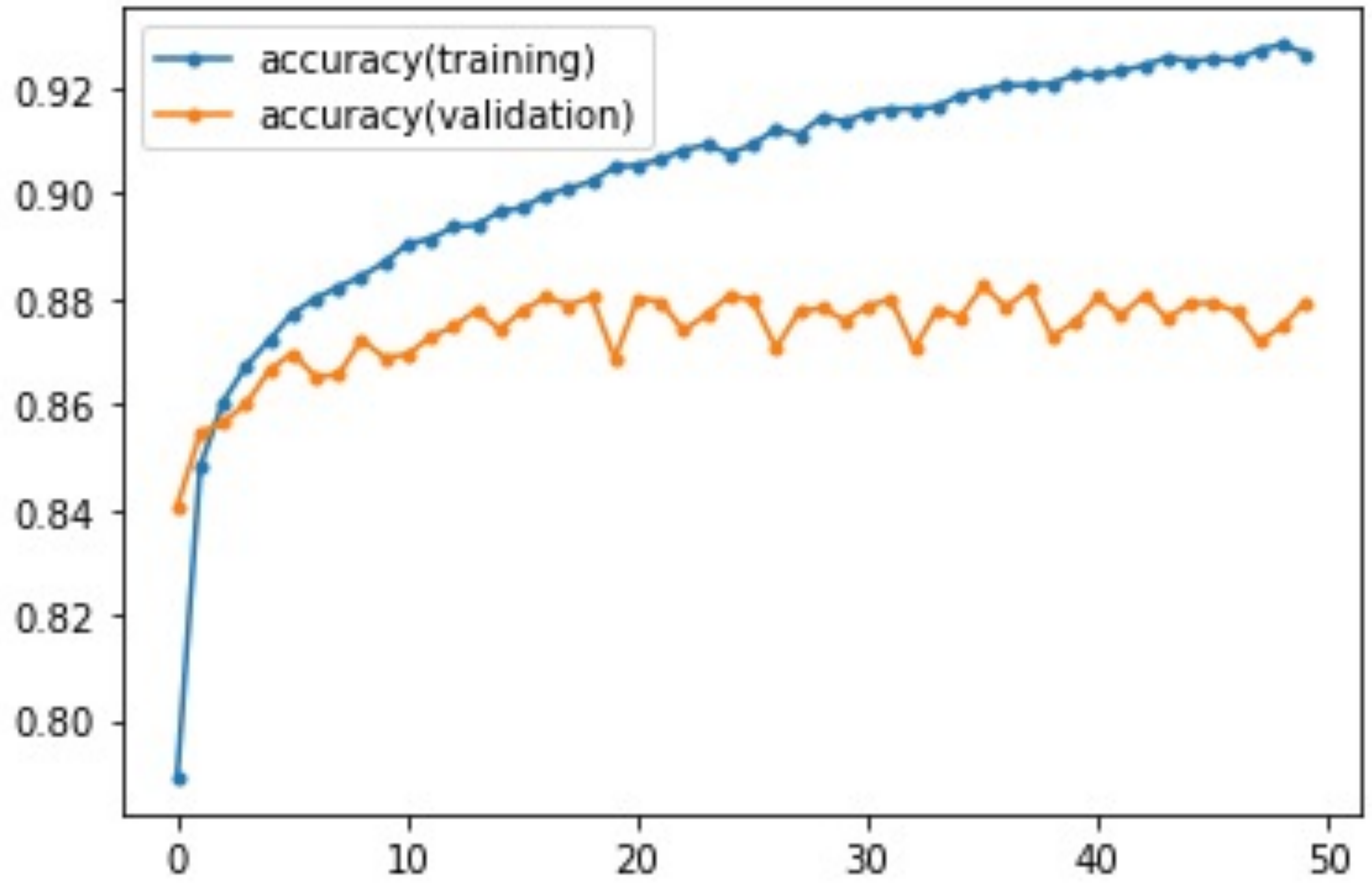
Test loss: 0.41971293091773987
Test accuracy: 0.8759999871253967

score = model.evaluate(特徴量,正解)で、scoreにはmodelを用いた予測結果の損失と正解率が代入される
score[0]で損失、score[1]で正解率が得られる。

```
a = 5
print(a)　出力 5
print('aは')　出力 aは
print('aは',a)　出力 aは5
```

今回のモデルで学習した結果





Test loss: 0.41971293091773987
Test accuracy: 0.8759999871253967

もっと精度をあげたい

ニューロンの数を増やす

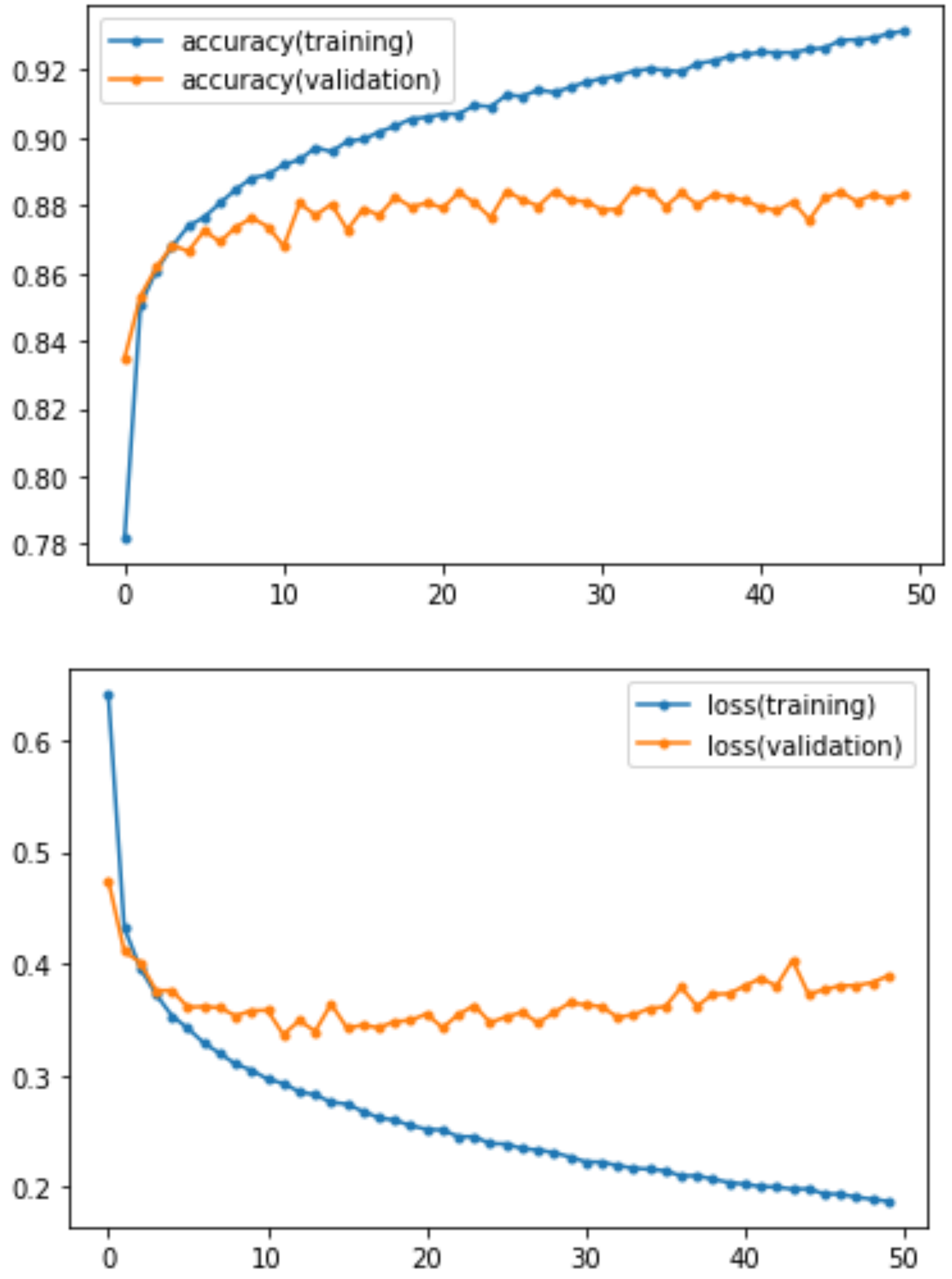
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(32,input_shape=(784,),activation='relu'))
model.add(Dense(10,activation='softmax'))
model.compile(loss='categorical_crossentropy’,

                optimizer='Adam',metrics=['accuracy'])
model.summary()
```

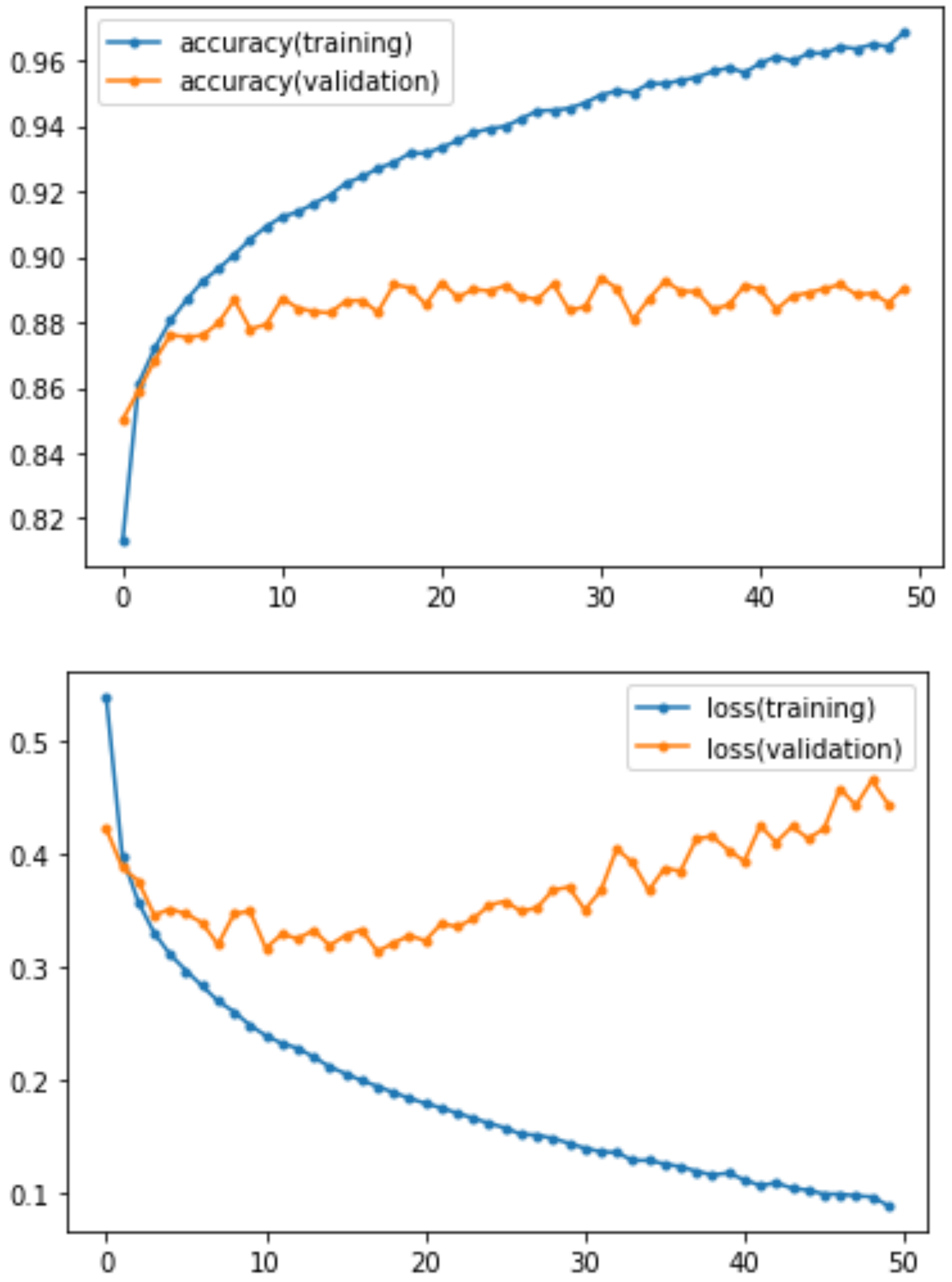
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(128,input_shape=(784,),activation='relu'))
model.add(Dense(10,activation='softmax'))
model.compile(loss='categorical_crossentropy’,

                optimizer='Adam',metrics=['accuracy'])
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 32)                25120
_________________________________________________________________
dense_13 (Dense)             (None, 10)                330
=================================================================
Total params: 25,450
Trainable params: 25,450
Non-trainable params: 0
_________________________________________________________________
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

変数が増えて正解率が少し上昇

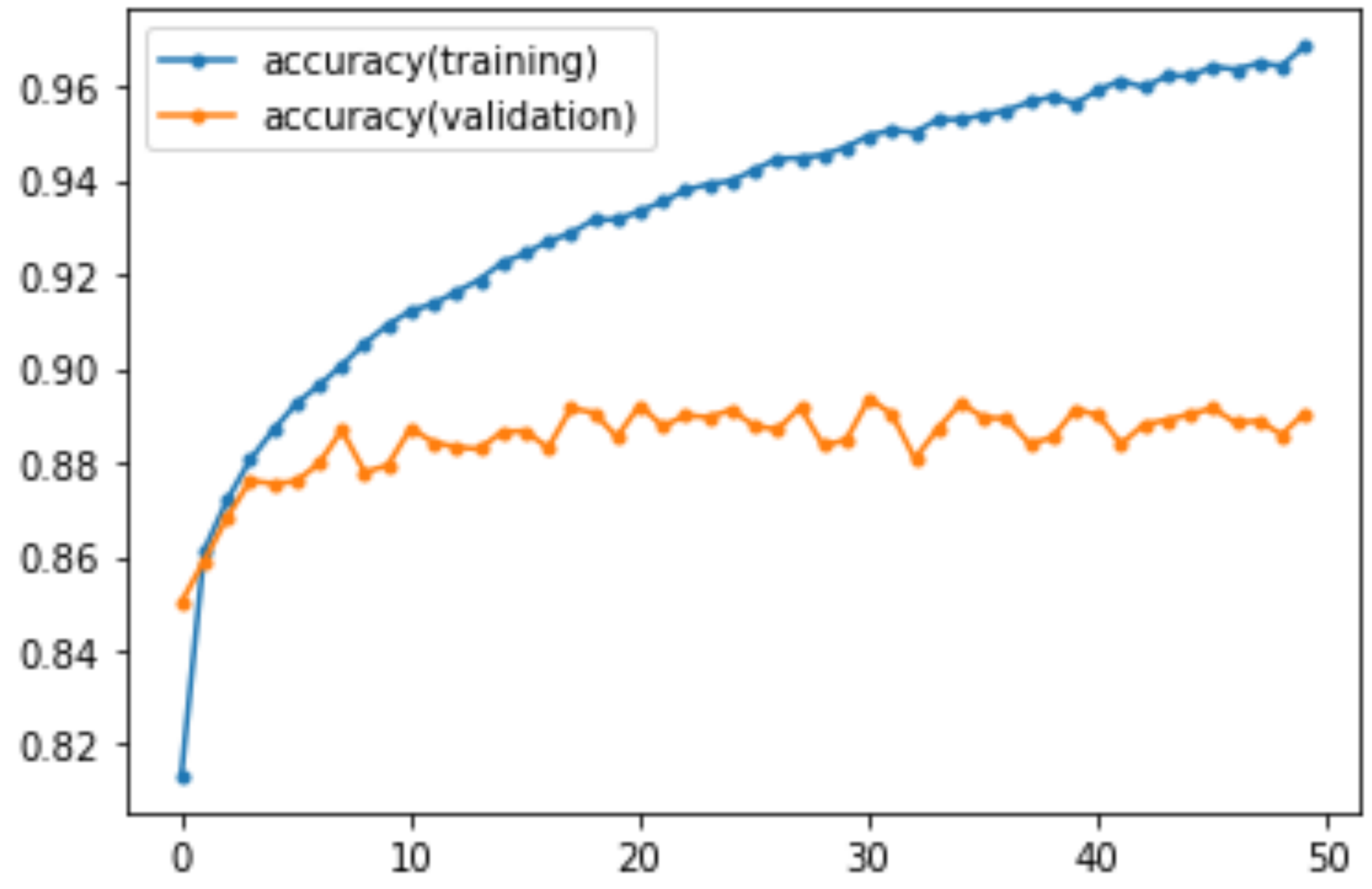




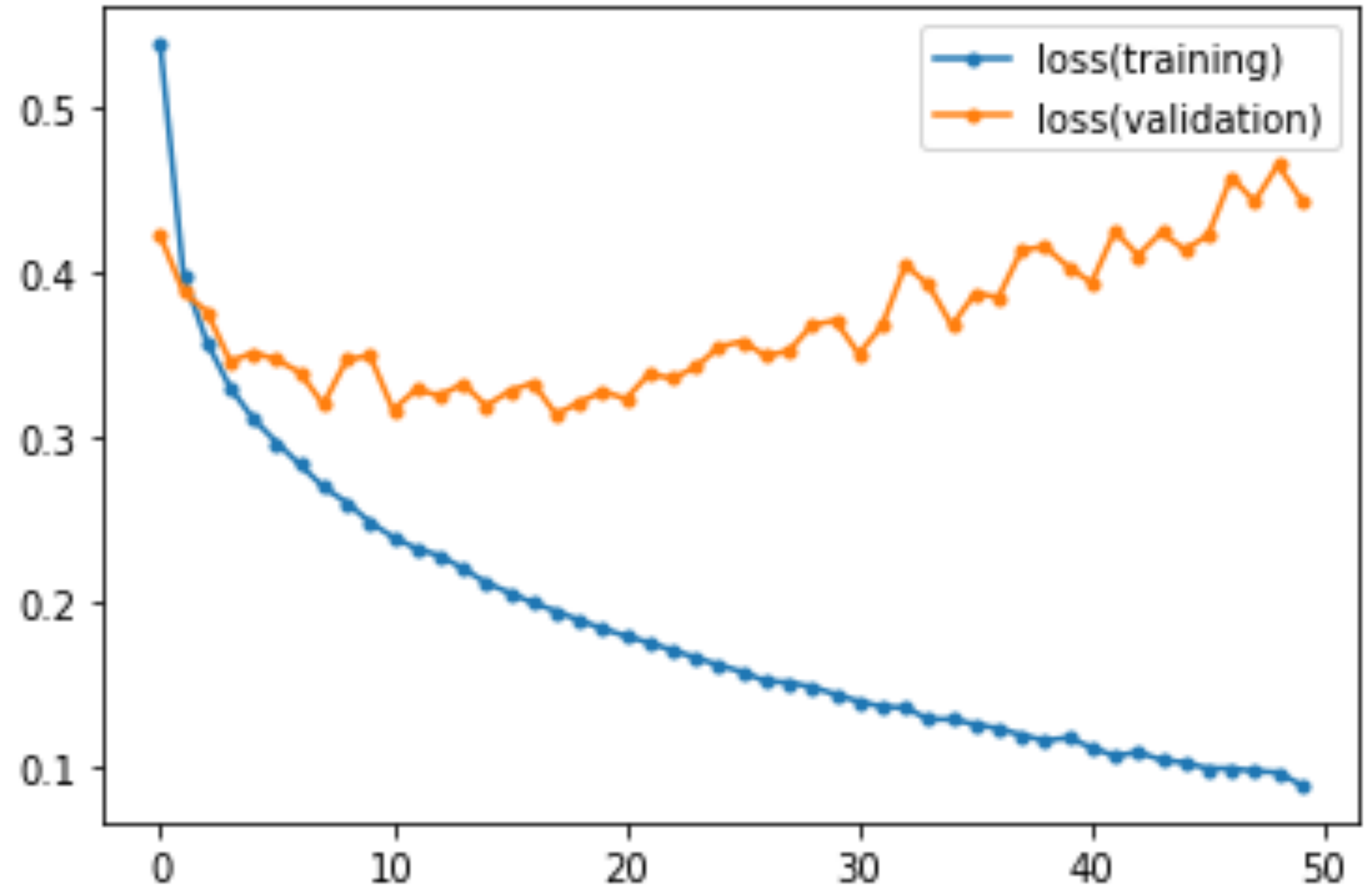
Test loss: 0.41971293091773987
Test accuracy: 0.8759999871253967

Test loss: 0.47562167048454285
Test accuracy: 0.8906000256538391

結果をもう少し考察



accuracy(training)は順調に上がっているが、validationが上がっていない。



trainingの損失率は順調に下がっているが、
validationが下がっていない。

層を追加してみよう

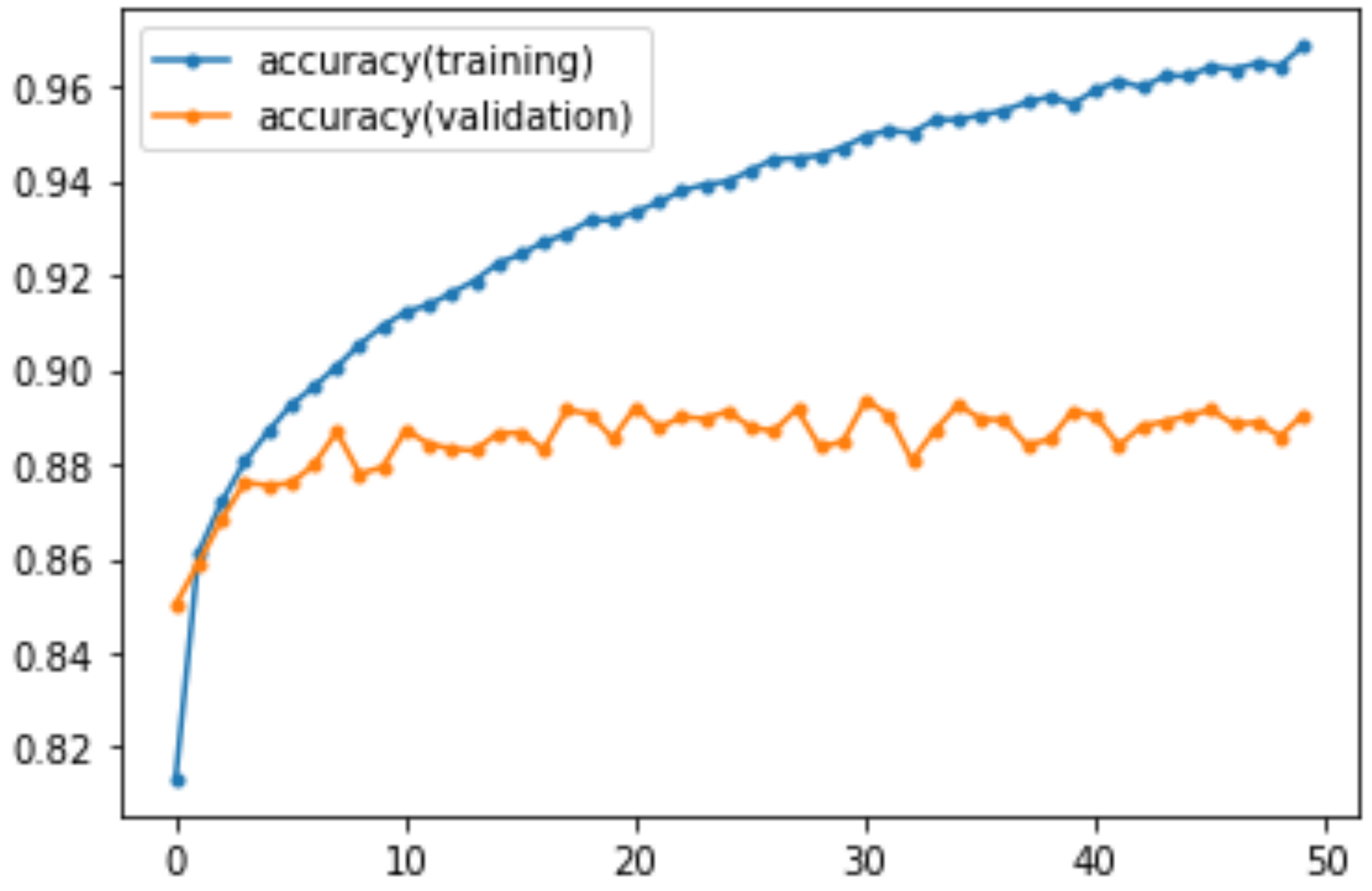
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(128,input_shape=(784,),activation='relu'))
model.add(Dense(10,activation='softmax'))
model.compile(loss='categorical_crossentropy’,

                    optimizer='Adam',metrics=['accuracy'])
model.summary()
```

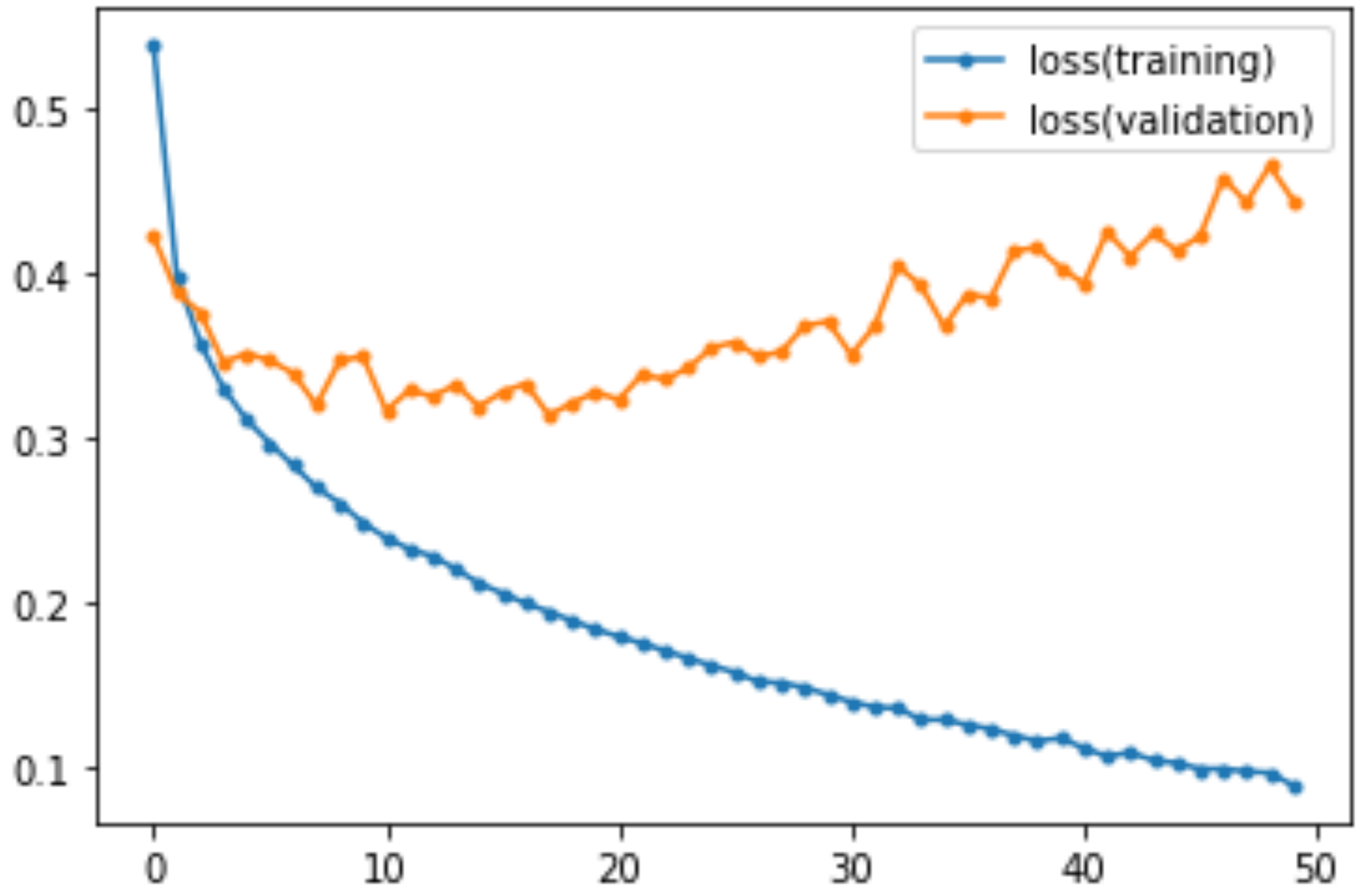
```
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

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

model = Sequential()
model.add(Dense(128,input_shape=(784,),activation='relu’))
model.add(Dense(64,activation='relu’))
model.add(Dense(32,activation='relu'))
model.add(Dense(10,activation='softmax'))
model.compile(loss='categorical_crossentropy’,

                    optimizer='Adam',metrics=['accuracy'])
model.summary()
```

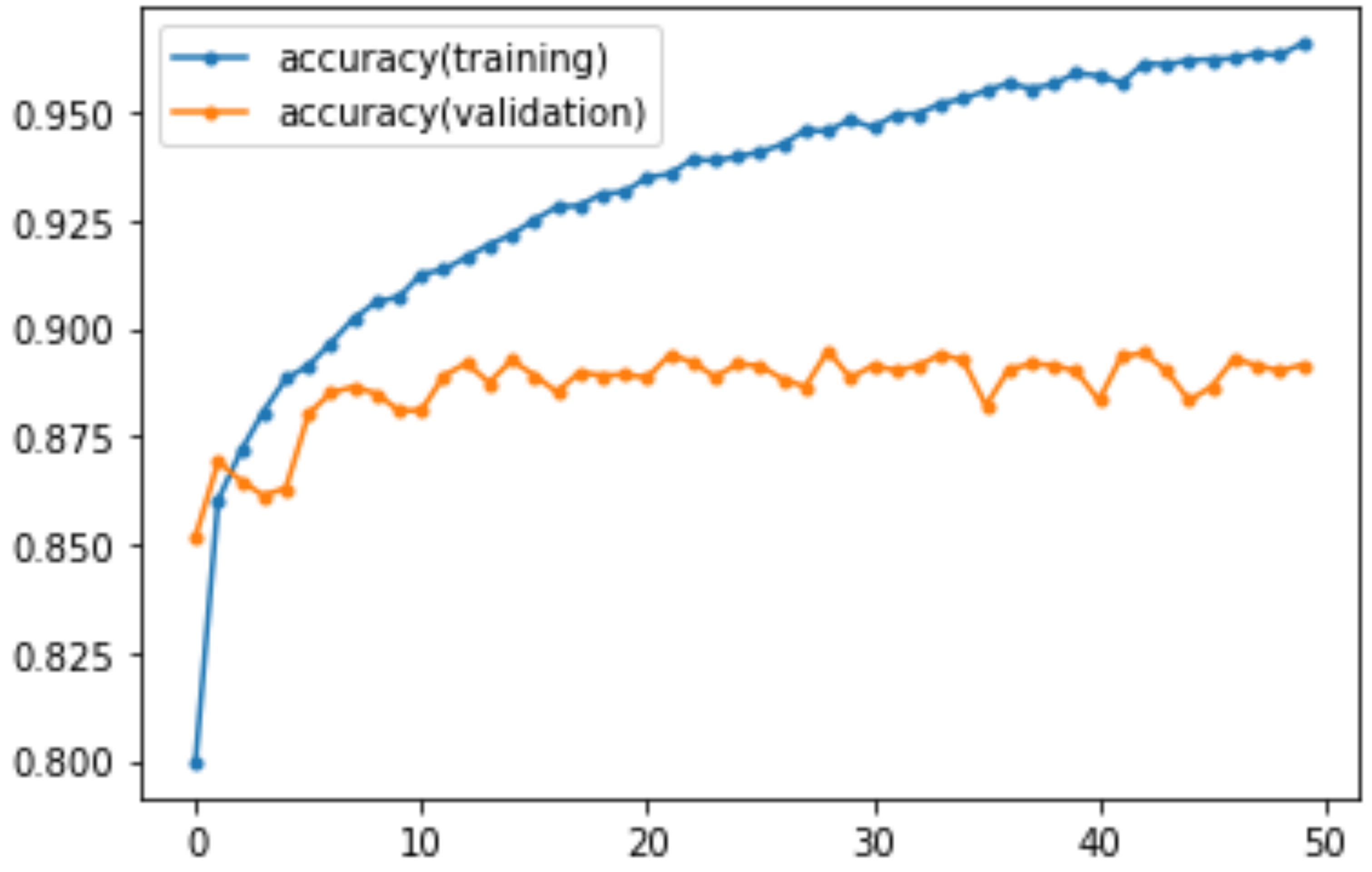
```
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_5 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_6 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_7 (Dense)              (None, 10)                330
=================================================================
Total params: 111,146
Trainable params: 111,146
Non-trainable params: 0
_________________________________________________________________
```

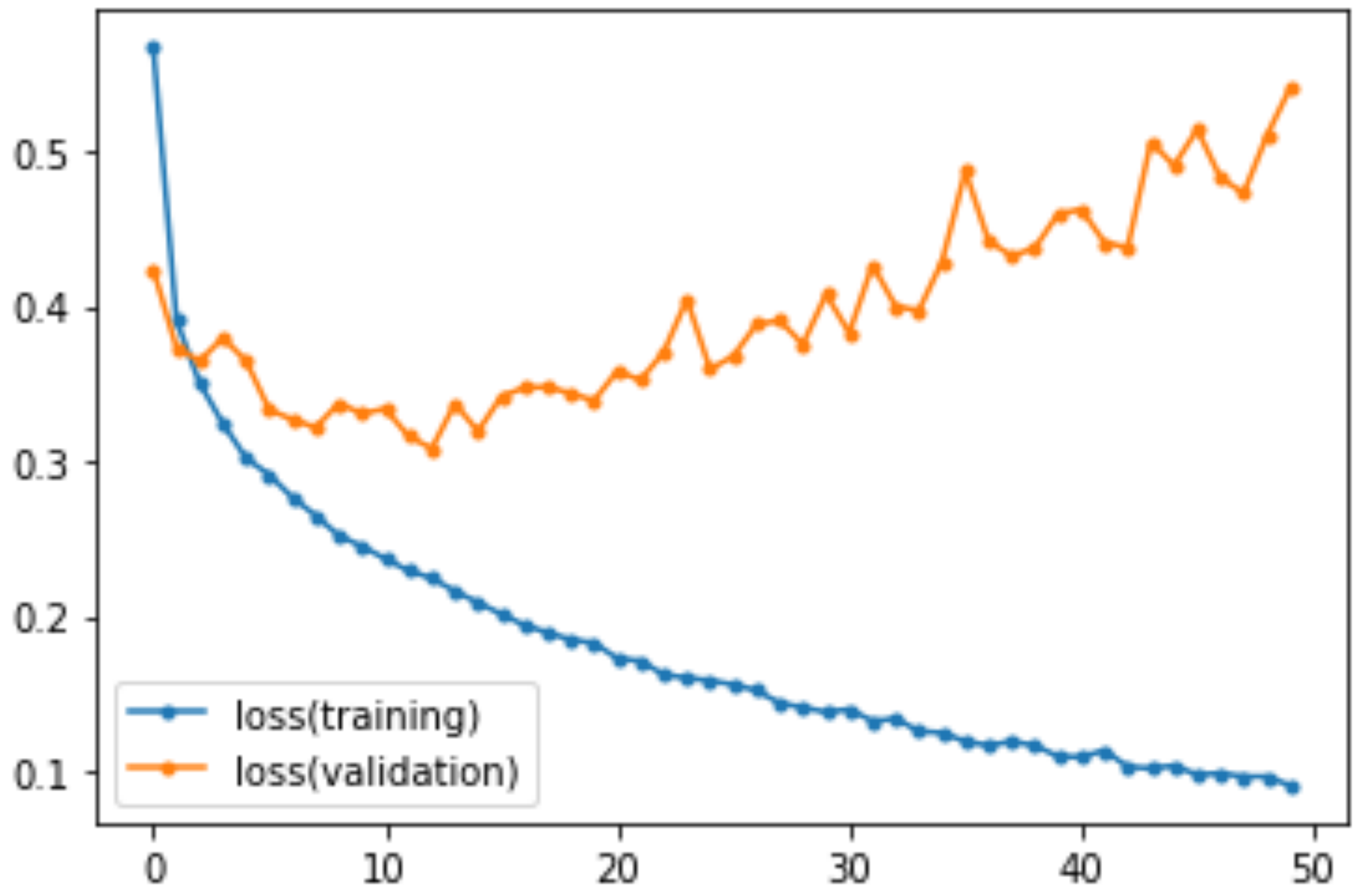
層を増やしても今回のデータでは精度あまり上がっていない





Test loss: 0.47562167048454285
Test accuracy: 0.8906000256538391





Test loss: 0.600957691694641
Test accuracy: 0.8855999708175659

Dropoutを加えてみよう

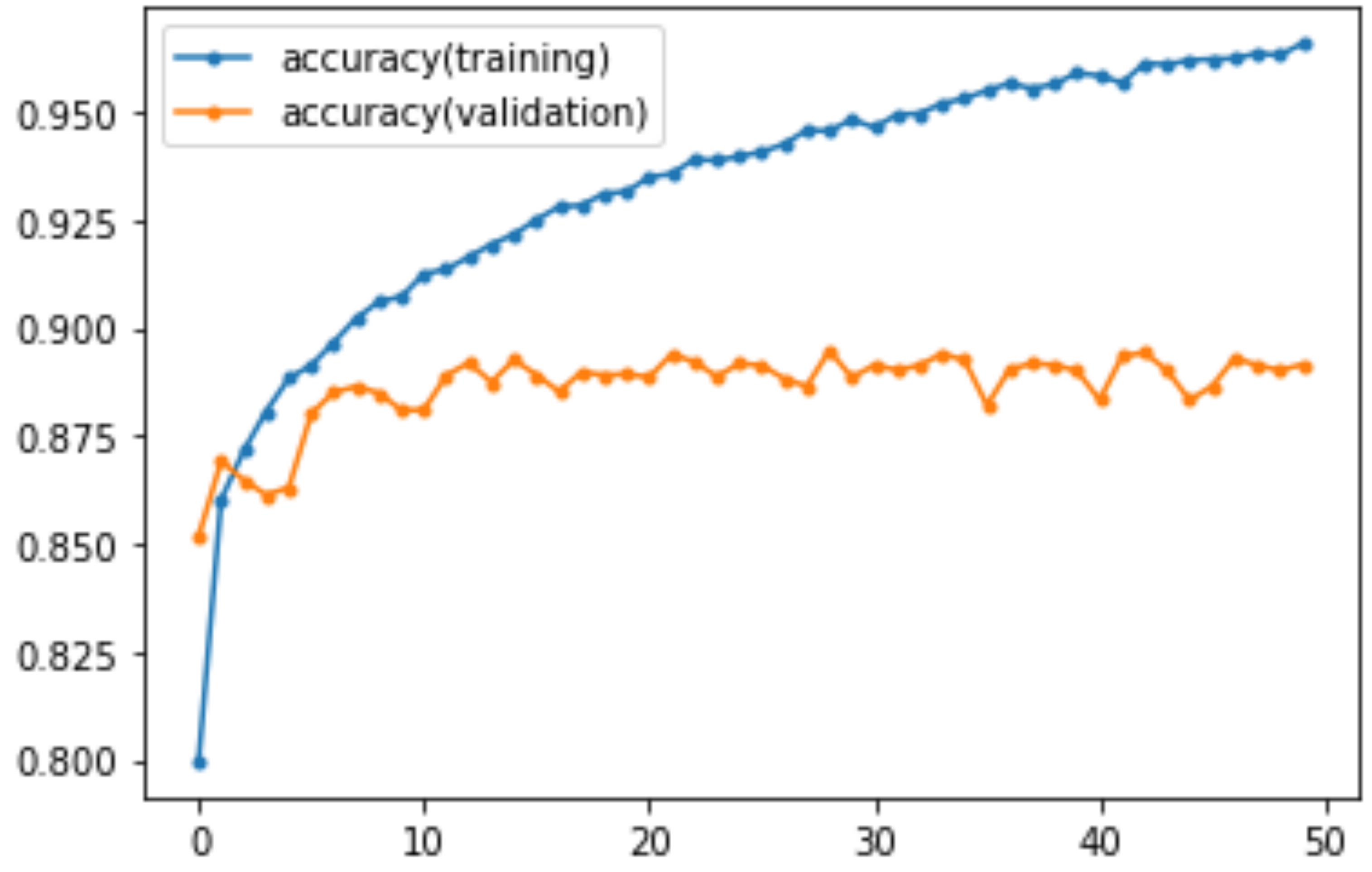
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

model = Sequential()
model.add(Dense(128,input_shape=(784,),activation='relu’))
model.add(Dense(64,activation='relu’))
model.add(Dense(32,activation='relu'))
model.add(Dense(10,activation='softmax'))
model.compile(loss='categorical_crossentropy’,

              optimizer='Adam',metrics=['accuracy'])
model.summary()
```

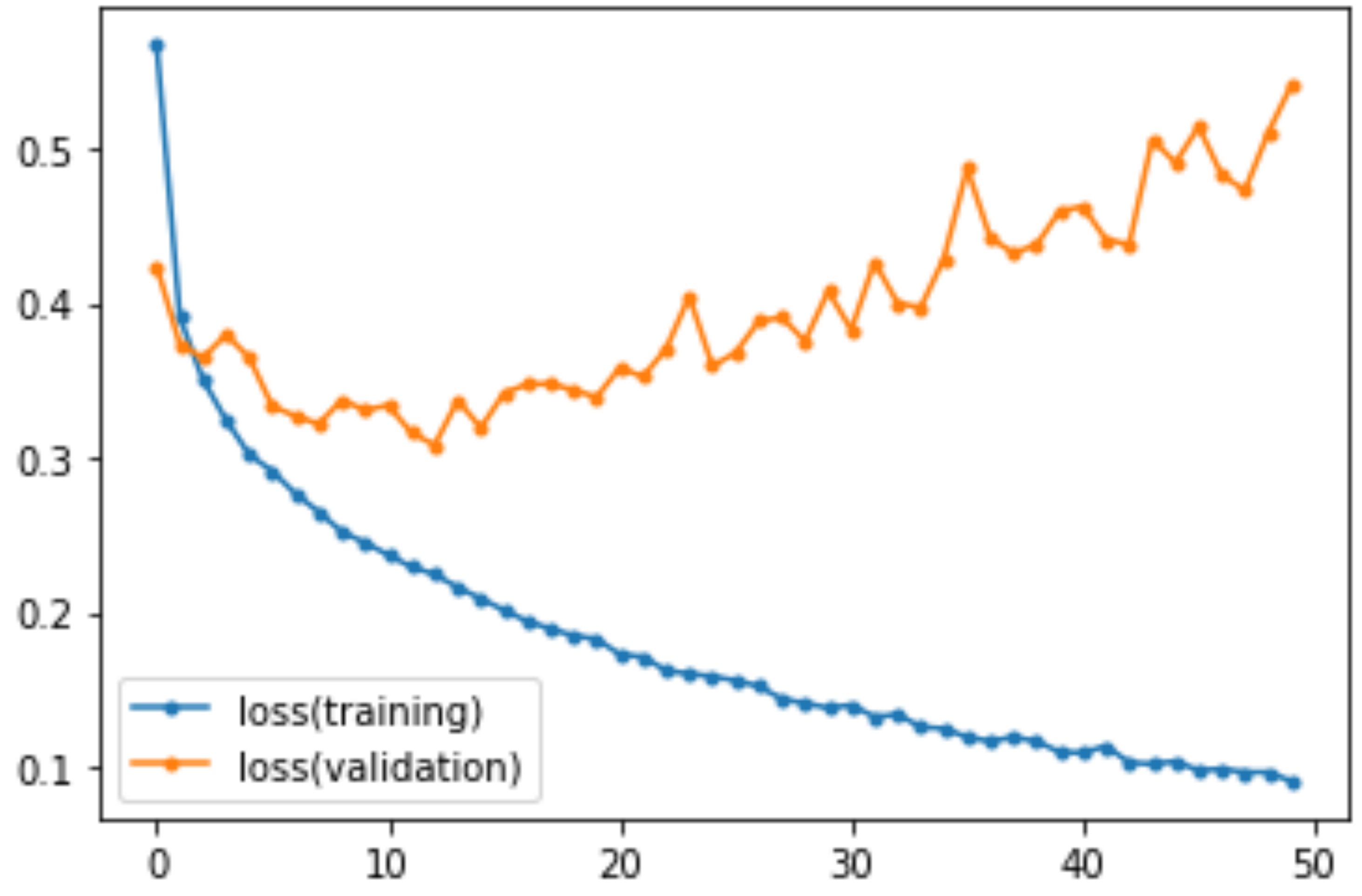
```
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_5 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_6 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_7 (Dense)              (None, 10)                330
=================================================================
Total params: 111,146
Trainable params: 111,146
Non-trainable params: 0
_________________________________________________________________
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

model = Sequential()
model.add(Dense(128,input_shape=(784,),activation='relu’))
model.add(Dropout(0.5))
model.add(Dense(64,activation='relu’))
model.add(Dense(32,activation='relu'))
model.add(Dense(10,activation='softmax'))
model.compile(loss='categorical_crossentropy’,

              optimizer='Adam',metrics=['accuracy'])
model.summary()
```

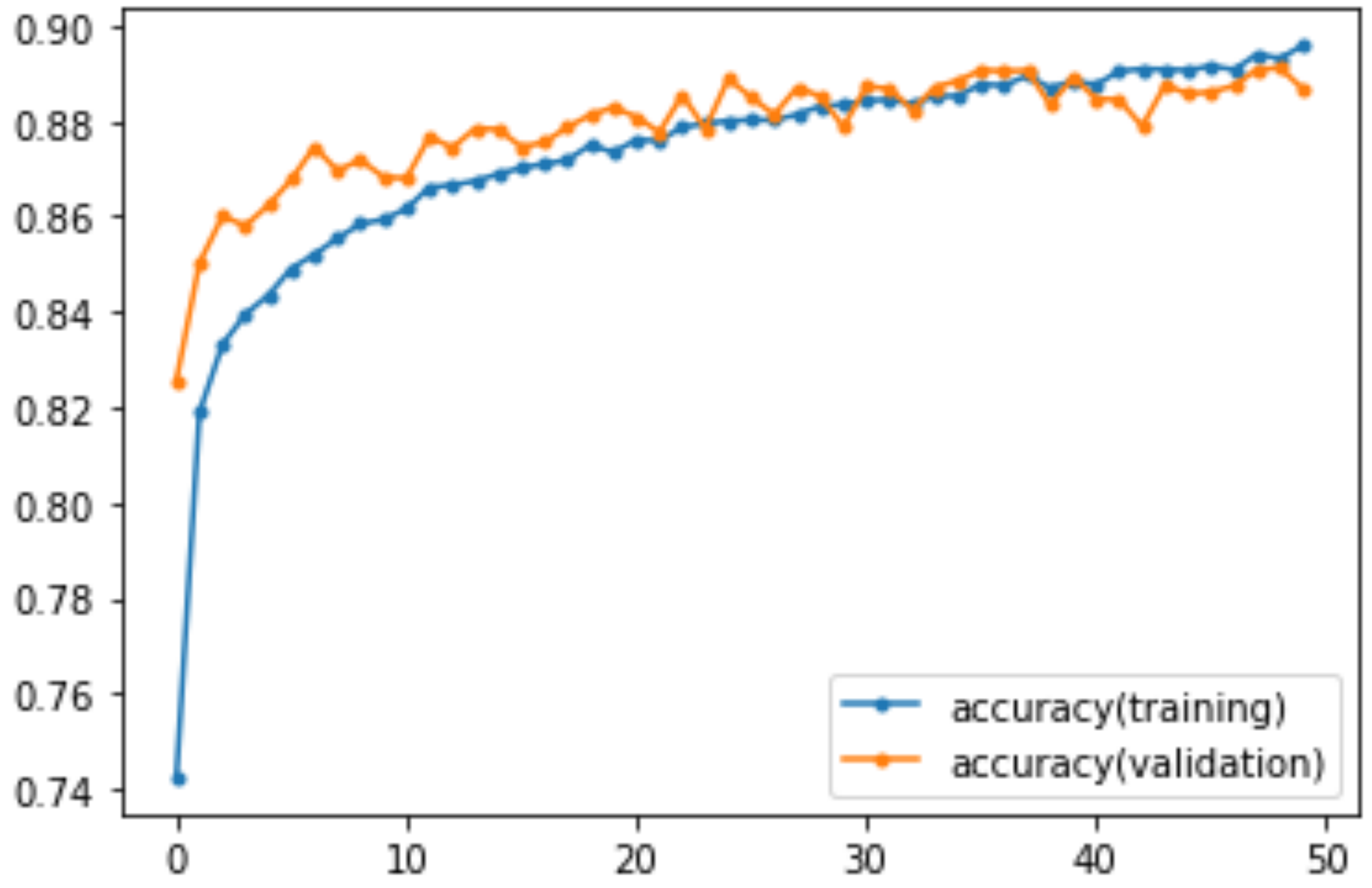
```
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 128)               100480
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_9 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_10 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_11 (Dense)             (None, 10)                330
=================================================================
Total params: 111,146
Trainable params: 111,146
Non-trainable params: 0
_________________________________________________________________
```

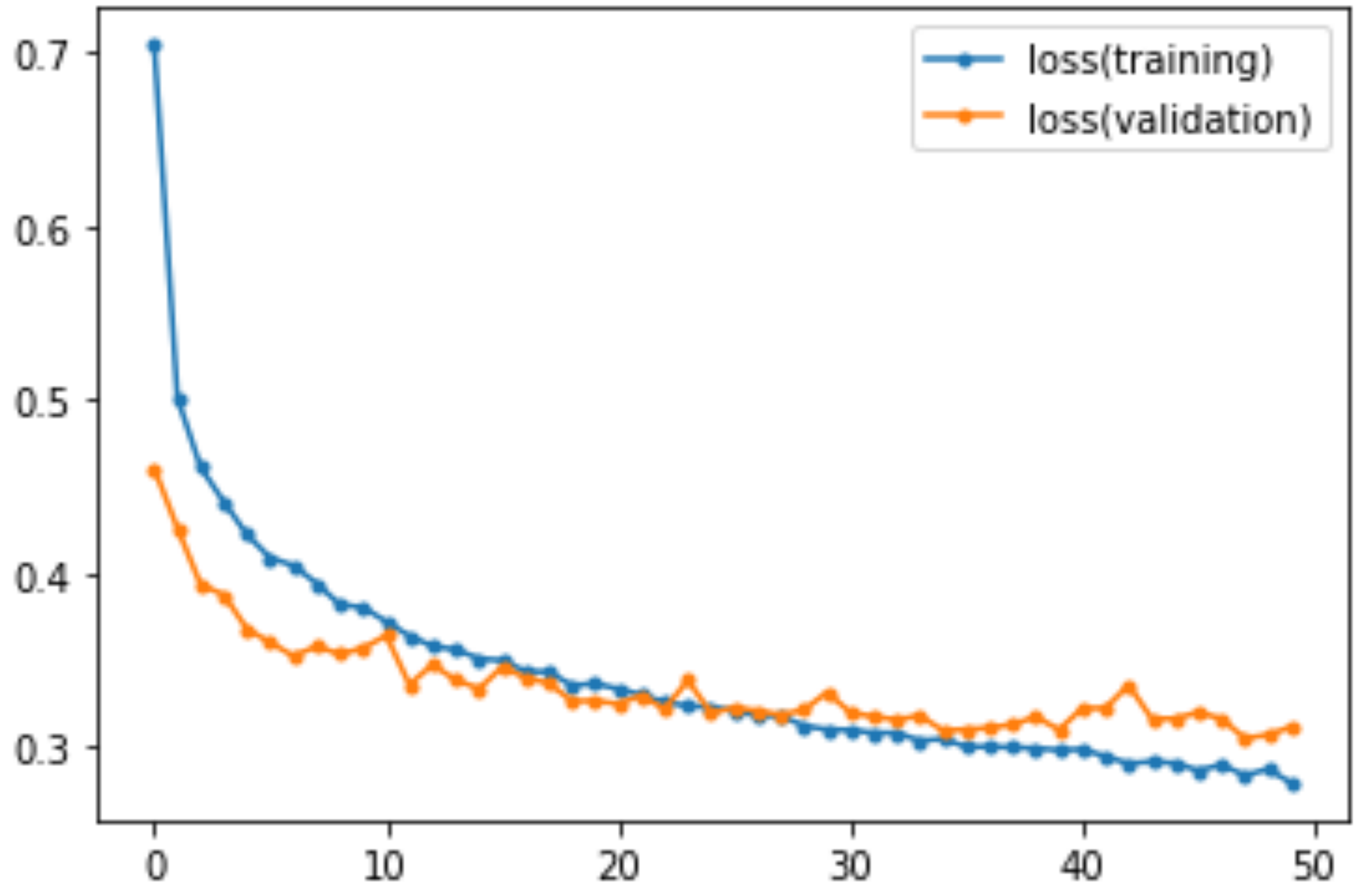
Dropoutを加えると過学習を抑制できる





Test loss: 0.6009576916694641
Test accuracy: 0.8855999708175659





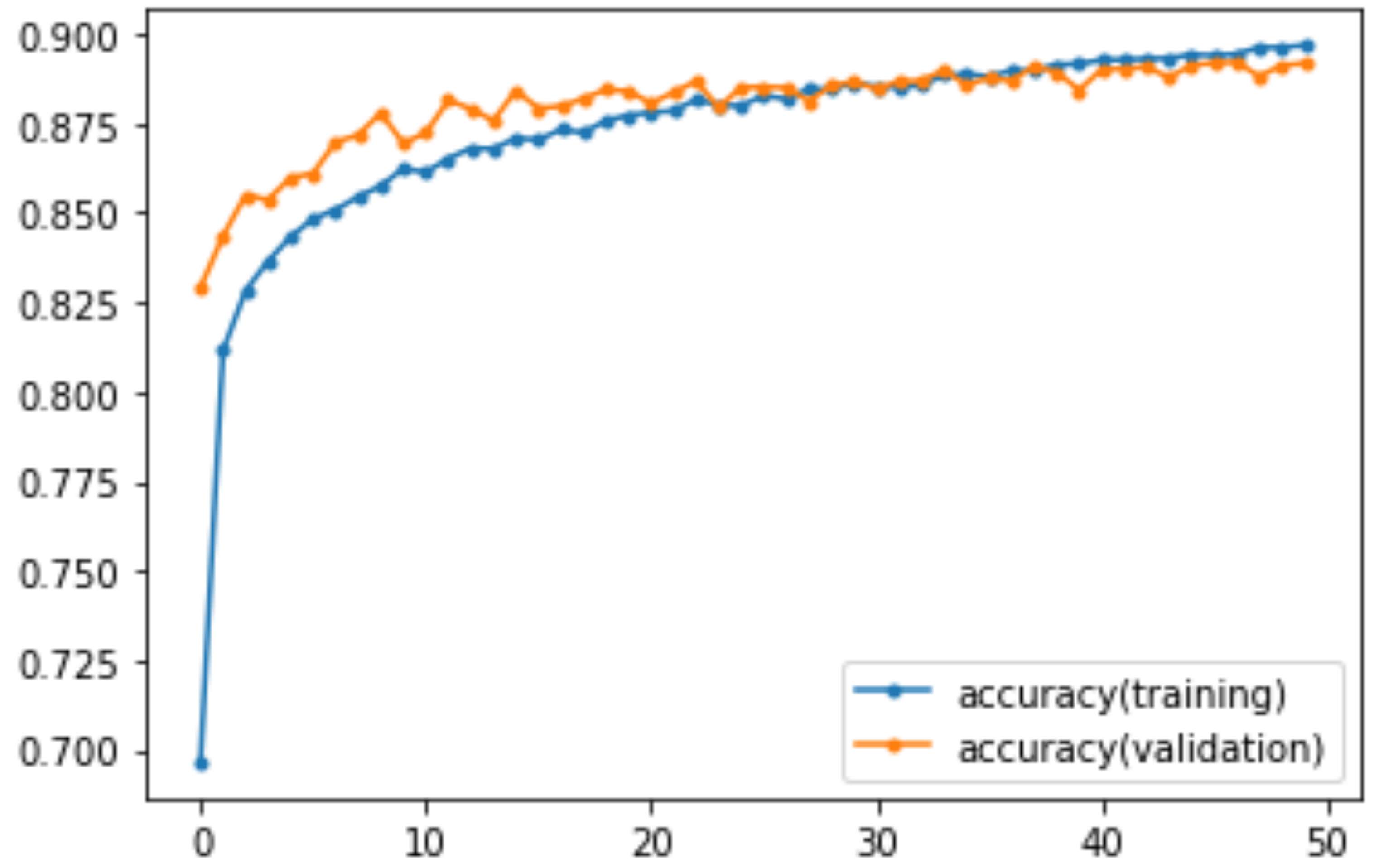
Test loss: 0.3330557644367218
Test accuracy: 0.8855000138282776

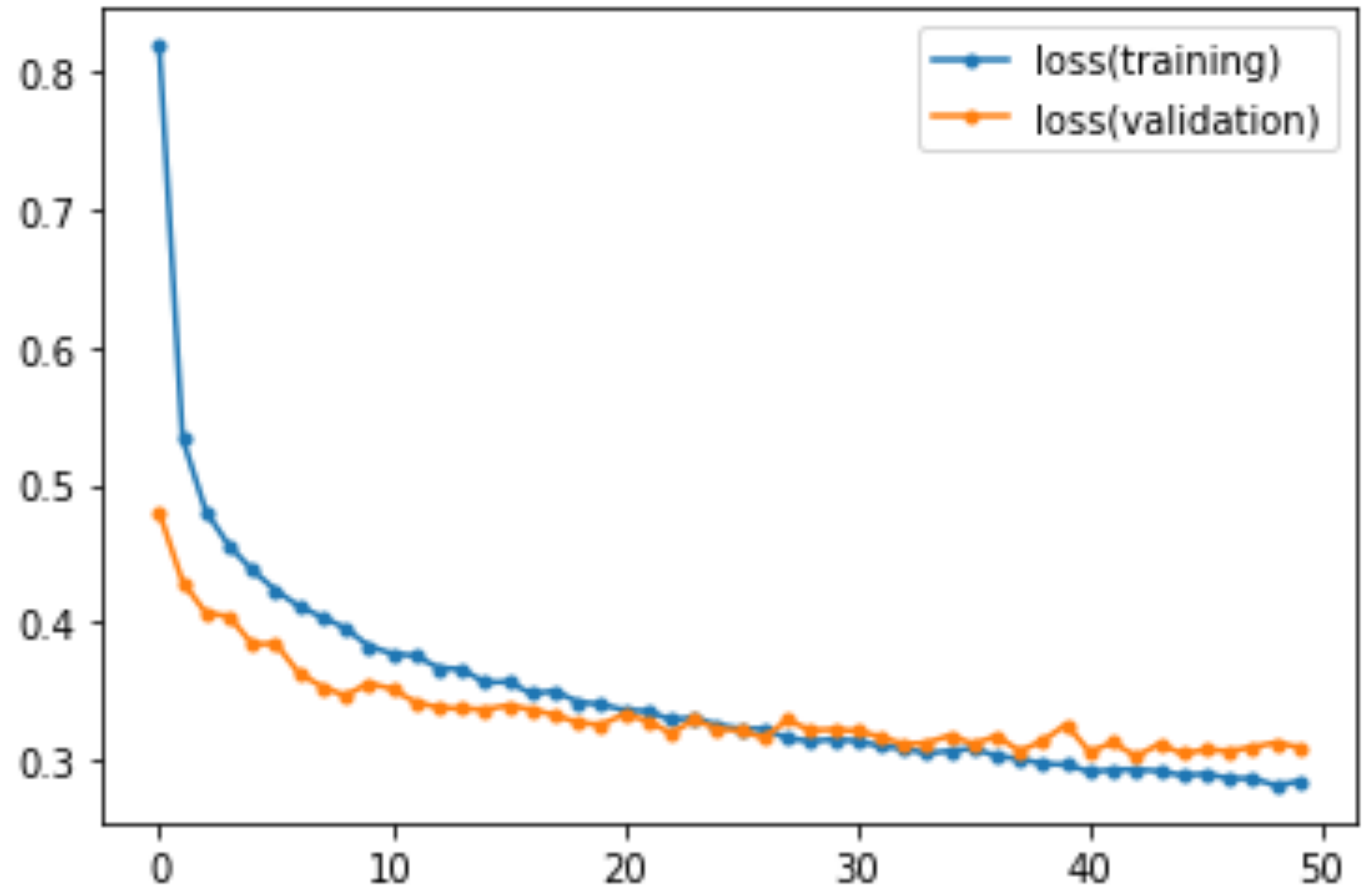
層はいくらでも増やすことが出来ます。

色々試してみましょう。

Layer (type)	Output Shape	Param #
dense_12 (Dense)	(None, 256)	200960
dropout_1 (Dropout)	(None, 256)	0
dense_13 (Dense)	(None, 64)	16448
dense_14 (Dense)	(None, 128)	8320
dropout_2 (Dropout)	(None, 128)	0
dense_15 (Dense)	(None, 64)	8256
dense_16 (Dense)	(None, 32)	2080
dense_17 (Dense)	(None, 10)	330

Total params: 236,394
Trainable params: 236,394
Non-trainable params: 0





Test loss: 0.3416001796722412
Test accuracy: 0.8855000138282776

今回のデータの量、質だと
MLPではパラメータを増やし
ても90%以上の精度は
なかなか出にくいようです。

実際の予測結果を確認してみよう

model.predict(知りたい変数)で予測する

```
test = model.predict(x_test)
print(test[0])
```

10個の数字が返ってくる

[6.8410866e-10 2.5985405e-10 1.5257271e-11 5.5445526e-10 3.2095484e-09 2.4379743e-04 1.7138366e-09 3.0476972e-03 1.6789203e-08 9.9670851e-01]

実際の予測結果を確認してみよう

model.predict(知りたい変数)で予測する

```
test = model.predict(x_test)
print(test[0])
```

10個の数字が返ってくる

[6.8410866e-10 2.5985405e-10 1.5257271e-11 5.5445526e-10 3.2095484e-09 2.4379743e-04 1.7138366e-09 3.0476972e-03 1.6789203e-08 9.9670851e-01]

0 1 2 3 4 5 6 7 8 9

実際の予測結果を確認してみよう

model.predict(知りたい変数)で予測する

```
test = model.predict(x_test)
print(test[0])
```

10個の数字が返ってくる

[6.8410866e-10 2.5985405e-10 1.5257271e-11 5.5445526e-10 3.2095484e-09 2.4379743e-04 1.7138366e-09 3.0476972e-03 1.6789203e-08 9.9670851e-01]

||

[$6.8410866 \times 10^{-10}$ 2.59854×10^{-10} 1.525727×10^{-11} $5.5445526 \times 10^{-10}$ 3.209548×10^{-9} 2.4379743×10^{-4} 1.7138366×10^{-9} 3.0476972×10^{-3} 1.6789203×10^{-8} 9.9670851×10^{-1}]

0.00..	0.00..	0.00..	0.00..	0.00..	0.000243..	0.00..	0.00304..	0.00..	0.96708..
0	1	2	3	4	5	6	7	8	9

実際の予測結果を確認してみよう

model.predict(知りたい変数)で予測する

```
test = model.predict(x_test)
print(test[0])
```

10個の数字が返ってくる

[6.8410866e-10 2.5985405e-10 1.5257271e-11 5.5445526e-10 3.2095484e-09 2.4379743e-04 1.7138366e-09 3.0476972e-03 1.6789203e-08 9.9670851e-01]

||

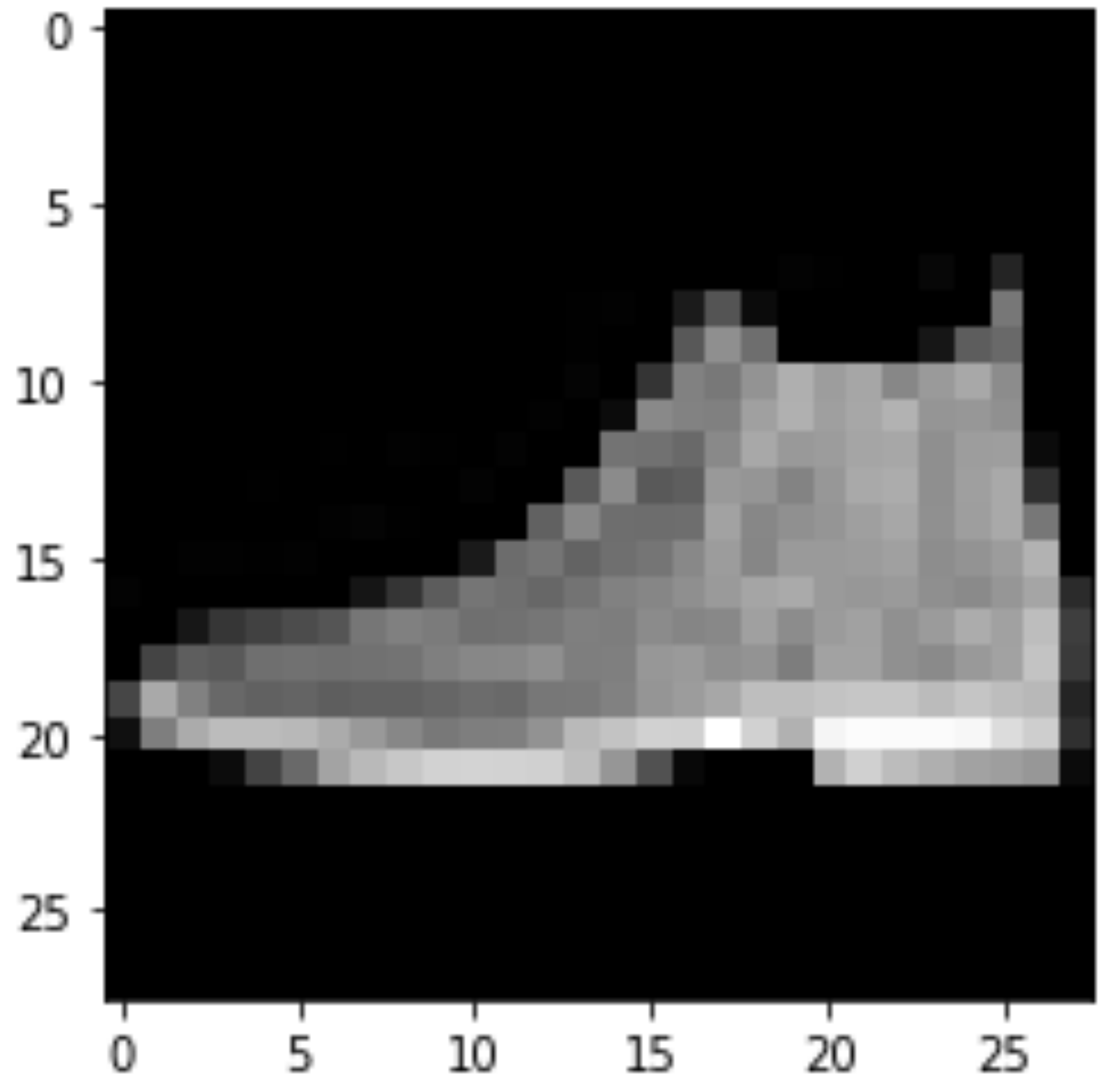
[6.8410866×10^{-10} 2.59854×10^{-10} 1.525727×10^{-11} 5.5445526×10^{-10} 3.209548×10^{-9} 2.4379743×10^{-4} 1.7138366×10^{-9} 3.0476972×10^{-3} 1.6789203×10^{-8} 9.9670851×10^{-1}]

0.00..	0.00..	0.00..	0.00..	0.00..	0.000243..	0.00..	0.00304..	0.00..	0.96708..
0	1	2	3	4	5	6	7	8	9

test[o]はなんだったか？

```
(x_train, y_train), (x_test, y_test) =    fashion_mnist.load_data
plt.imshow(x_test[o],'gray')
plt.show()
print(y_test[o])
```

Ankle boot



9

0 : T-shirt/top、1 : Trouser、2 : Pullover、3 : Dress、4 : Coat、
5 : Sandal、6 : Shirt、7 : Sneaker、8 : Bag、9 : Ankle boot

課題

FASHION-MNISTではなく、MNISTでMLPを実践しなさい

・4層(入力層、出力層含めて)以上にすること
・Dropout()をいれること
(過学習が極力ない形にしてください)

testの10枚目の画像を表示して、
作成したモデルでの予測結果(最大となるクラスとその確率)を示しなさい