医療とAI・ビッグデータ応用
MLP②

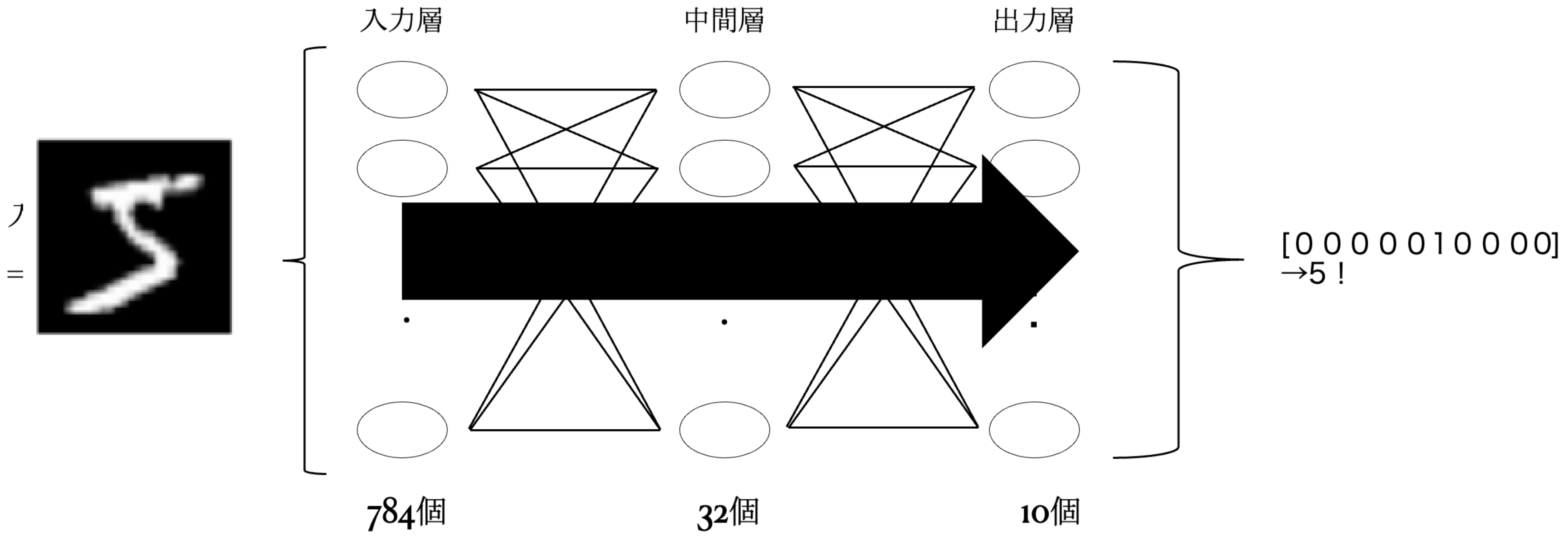
本スライドは、自由にお使いください。
使用した場合は、このQRコードからアンケートに回答をお願いします。

統合教育機構
須藤毅顕

前回までの復習

モデルの作成

```
from keras.models import Sequential
from 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()
```

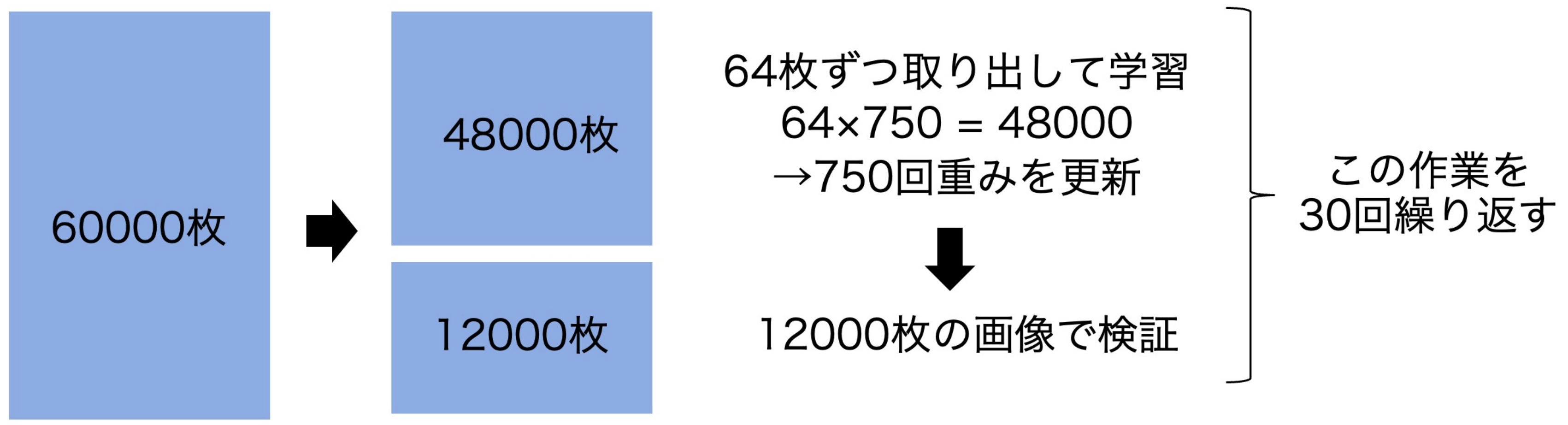


WebClassに「応用5_20231109_template.ipynb」
があるので開いて実行していきましょう
(GPUに変更してください)

学習

```
result = model.fit(x_train, y_train, epochs=30, batch_size=64, validation_split=0.2)
```

```
Epoch 1/30
750/750 [==============================] - 4s 4ms/step - loss: 0.4633 - accuracy: 0.8709 - val_loss: 0.2517 - val_accuracy: 0.9298
Epoch 2/30
750/750 [==============================] - 3s 4ms/step - loss: 0.2401 - accuracy: 0.9309 - val_loss: 0.2020 - val_accuracy: 0.9448
Epoch 3/30
750/750 [==============================] - 3s 4ms/step - loss: 0.1906 - accuracy: 0.9456 - val_loss: 0.1802 - val_accuracy: 0.9498
:
Epoch 28/30
750/750 [==============================] - 3s 4ms/step - loss: 0.0288 - accuracy: 0.9922 - val_loss: 0.1434 - val_accuracy: 0.9629
Epoch 29/30
750/750 [==============================] - 4s 5ms/step - loss: 0.0270 - accuracy: 0.9930 - val_loss: 0.1353 - val_accuracy: 0.9660
Epoch 30/30
750/750 [==============================] - 3s 4ms/step - loss: 0.0264 - accuracy: 0.9930 - val_loss: 0.1381 - val_accuracy: 0.9654
```



予測

最後に再度予測してみる

予測はmodel.predict()

一応名前を変更して学習後はtest2とする

```
test2 = model.predict(x_test)

313/313 [==============================] - 1s 2ms/step
```

2枚目を再度確認

```
print(test2[1])

[8.0701529e-11 3.9249046e-10 9.9999940e-01 8.3521821e-09 1.4485680e-28
 6.2819618e-07 2.5197353e-11 1.3647900e-22 2.3160052e-10 1.9060435e-24]
```

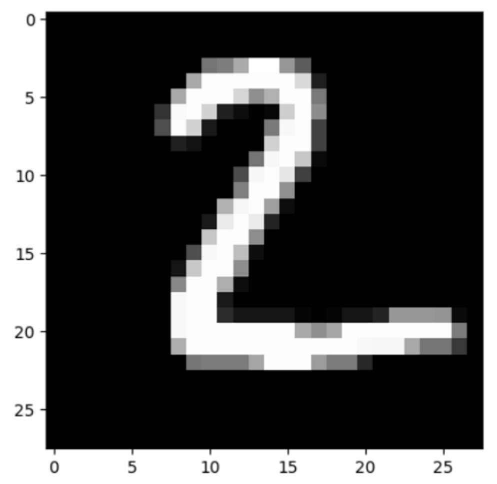
8.07e-11 は 8.07×10^{-11} = 0.0000000000807

```
import numpy as np
print(np.around(test2[1],3))
print(y_test[1])

[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
```

np.aroung(配列, num)で、
配列を少数第num位で四捨五入

99.9999...%で2と予想している

```
import matplotlib.pyplot as plt
plt.imshow(x_test[1],'gray')
plt.show()
```

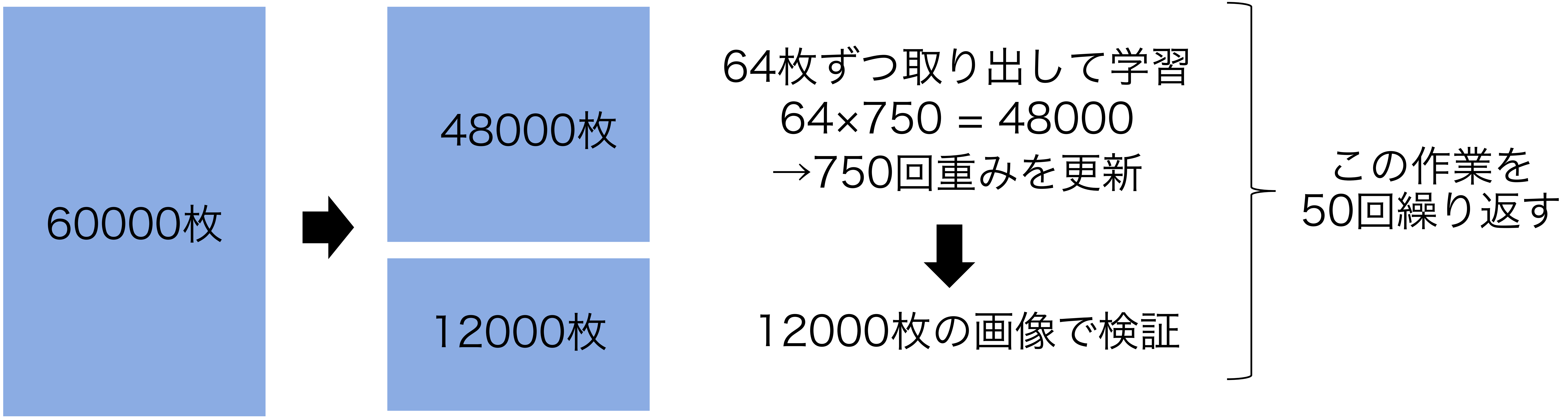



```
print(y_test[1])

2
```

epochs=50で実行し直しましょう

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

```
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
```



結果の作図

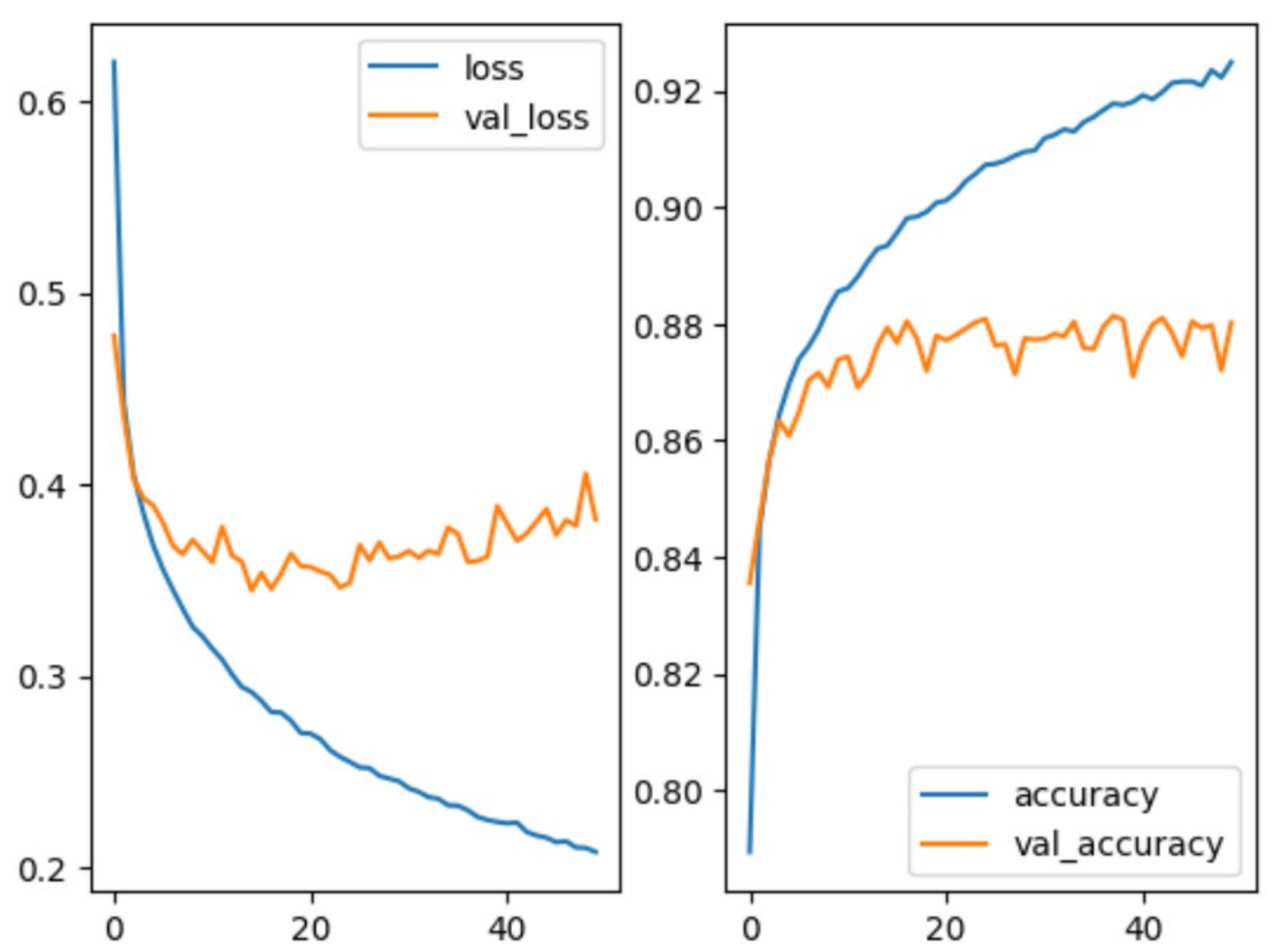
```
import matplotlib.pyplot as plt

plt.subplot(1,2,1)
plt.plot(result.history['loss'],label='loss')
plt.plot(result.history['val_loss'],label='val_loss')
plt.legend()
plt.subplot(1,2,2)
plt.plot(result.history['accuracy'],label='accuracy')
plt.plot(result.history['val_accuracy'],label='val_accuracy')
plt.legend()
plt.show()
```

縦1, 横2の1つ目

誤差(loss)の折れ線グラフ

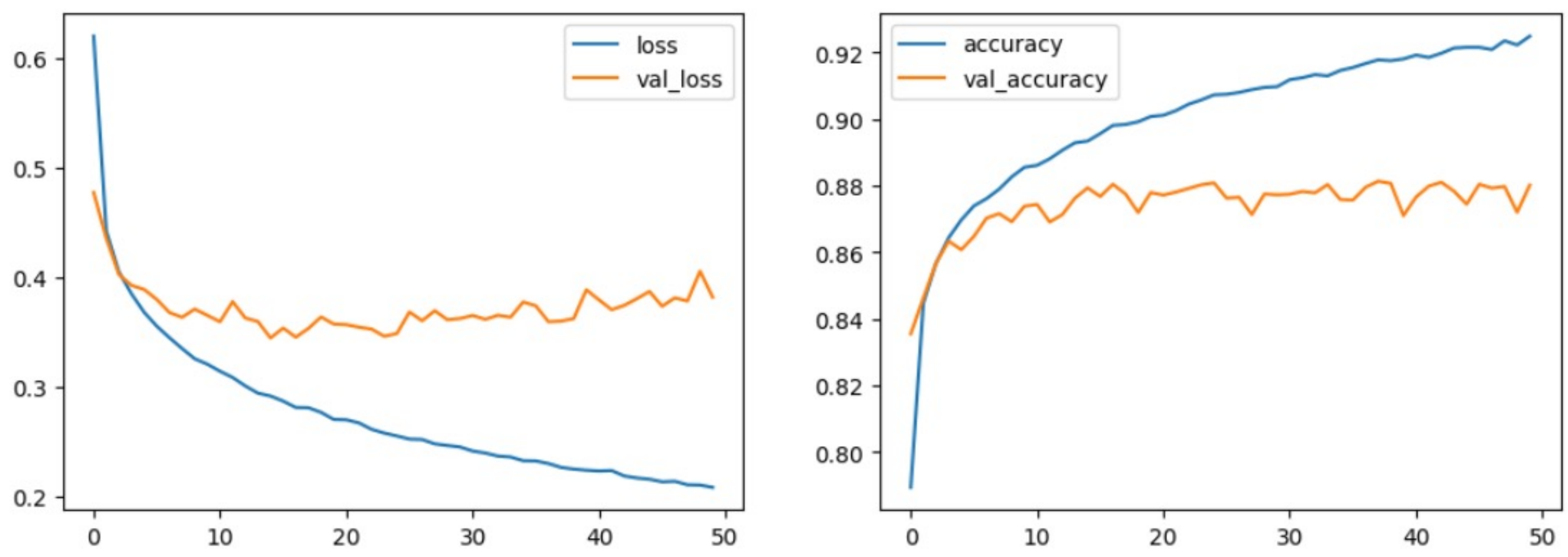
縦1, 横2の2つ目

正解率(accuracy)の折れ線グラフ



結果の作図

```
import matplotlib.pyplot as plt
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(result.history['loss'],label='loss')
plt.plot(result.history['val_loss'],label='val_loss')
plt.legend()
plt.subplot(1,2,2)
plt.plot(result.history['accuracy'],label='accuracy')
plt.plot(result.history['val_accuracy'],label='val_accuracy')
plt.legend()
plt.show()
```



result.historyの中身について

print(result.history)

```
{'loss': [0.6204796433448792, 0.4424095153808594, 0.4049777686595917, 0.38482892513275146, 0.3680422008037567, 0.3553660809993744, 0.34484514594078064,
0.3349671959877014, 0.32561713457107544, 0.3206530511379242, 0.3141988515853882, 0.30852627754211426, 0.3008130192756653, 0.29420095682144165, 0.29154443740844727,
0.28692877292633057, 0.28109729290008545, 0.28085023164749146, 0.27662160992622375, 0.2701963186264038, 0.26984351873397827, 0.26697129011154175, 0.26113253831863403,
0.25769904255867004, 0.2550542652606964, 0.2521992623806, 0.2518135905265808, 0.24781620502471924, 0.24636663496494293, 0.24498197436332703, 0.2412831038236618,
0.23945355415344238, 0.23669078946113586, 0.2358267456293106, 0.23242957890033722, 0.23221394419670105, 0.2298291176557541, 0.22631986439228058, 0.2247573286294937,
0.22372491657733917, 0.22302721440792084, 0.22342391312122345, 0.2184654176235199, 0.21663862466812134, 0.21550878882408142, 0.21314330399036407, 0.2136351317167282,
0.2104269564151764, 0.2101719230413437, 0.2080526500940323],

'accuracy': [0.789354145526886, 0.8447291851043701, 0.8566874861717224, 0.8643541932106018, 0.8697083592414856, 0.8739166855812073, 0.8761041760444641,
0.8789583444595337, 0.8826666474342346, 0.8855208158493042, 0.886104166507721, 0.8880624771118164, 0.8906458616256714, 0.8928750157356262, 0.8933958411216736,
0.8956249952316284, 0.898104190826416, 0.8983749747276306, 0.8991875052452087, 0.9007708430290222, 0.9011458158493042, 0.9025416374206543, 0.9044791460037231,
0.9057499766349792, 0.9072708487510681, 0.9074375033378601, 0.9079999923706055, 0.9088541865348816, 0.9095208048820496, 0.9097291827201843, 0.9118333458900452,
0.9124166369438171, 0.9133541584014893, 0.9129791855812073, 0.9146041870117188, 0.9154791831970215, 0.9166874885559082, 0.9177916646003723, 0.9175416827201843,
0.9180625081062317, 0.9192083477973938, 0.9185208082199097, 0.9197708368301392, 0.9213333129882812, 0.9215624928474426, 0.9215624928474426, 0.9208750128746033,
0.9235208630561829, 0.922249972820282, 0.9248958230018616],

'val_loss': [0.4776163697242737, 0.4357101321220398, 0.40258854627609253, 0.39271479845046997, 0.3889164924621582, 0.3798101842403412, 0.3678809702396393,
0.36349910497665405, 0.37104806303977966, 0.3652504086494446, 0.35947203636169434, 0.37782320380210876, 0.3629121482372284, 0.3596421778202057, 0.34462788701057434,
0.35367244482040405, 0.3453534245491028, 0.35335132479667664, 0.36383312940597534, 0.35738077759742737, 0.35678377747535706, 0.35446837544441223, 0.3527243733406067,
0.34618350863456726, 0.34865111112594604, 0.3683689832687378, 0.3603420555591583, 0.36953479051589966, 0.36129820346832275, 0.3623996675014496, 0.36520129442214966,
0.36162319779396057, 0.36536312103271484, 0.3636190593242645, 0.37748828530311584, 0.37399259209632874, 0.35947826504707336, 0.36005476117134094, 0.3623170256614685,
0.3886697292327881, 0.37945523858070374, 0.37049585580825806, 0.3743937611579895, 0.3805387318134308, 0.38715872168540955, 0.3735528290271759, 0.38132739067077637,
0.37841248512268066, 0.4056456685066223, 0.3818448781967163],

'val_accuracy': [0.8355000019073486, 0.8462499976158142, 0.8567500114440918, 0.8633333444595337, 0.8607500195503235, 0.8647500276565552, 0.8702499866485596,
0.8715833425521851, 0.8691666722297668, 0.8738333582878113, 0.8743333220481873, 0.8690833449363708, 0.8713333606719971, 0.8762500286102295, 0.8793333172798157,
0.8767499923706055, 0.8804166913032532, 0.8774999976158142, 0.871916651725769, 0.877916693687439, 0.8771666884422302, 0.878083348274231, 0.8791666626930237,
0.8802499771118164, 0.8808333277702332, 0.8762500286102295, 0.8765000104904175, 0.8713333606719971, 0.8774999976158142, 0.8772500157356262, 0.8774166703224182,
0.878250002861023, 0.8778333067893982, 0.8803333044052124, 0.8758333325386047, 0.8756666779518127, 0.8794999718666077, 0.8813333511352539, 0.8806666731834412,
0.8709999918937683, 0.8765833377838135, 0.8798333406448364, 0.8809999823570251, 0.878333330154419, 0.8744166493415833, 0.8804166913032532, 0.8792499899864197,
0.8797500133514404, 0.871999979019165, 0.8801666498184204]}
```

result.historyの中身について

print(result.history)

エポック50回分の誤差と正解率

```
{'loss': [0.6204796433448792, 0.4424095153808594, 0.4049777686595917, 0.38482892513275146, 0.3680422008037567, 0.3553660809993744, 0.34484514594078064,
0.3349671959877014, 0.32561713457107544, 0.3206530511379242, 0.3141988515853882, 0.30852627754211426, 0.3008130192756653, 0.29420095682144165, 0.29154443740844727,
0.28692877292633057, 0.28109729290008545, 0.28085023164749146, 0.27662160992622375, 0.2701963186264038, 0.26984351873397827, 0.26697129011154175, 0.26113253831863403,
0.25769904255867004, 0.2550542652606964, 0.2521992623806, 0.2518135905265808, 0.24781620502471924, 0.24636663496494293, 0.24498197436332703, 0.2412831038236618,
0.23945355415344238, 0.23669078946113586, 0.2358267456293106, 0.23242957890033722, 0.23221394419670105, 0.2298291176557541, 0.22631986439228058, 0.2247573286294937,
0.22372491657733917, 0.22302721440792084, 0.22342391312122345, 0.2184654176235199, 0.21663862466812134, 0.21550878882408142, 0.21314330399036407, 0.2136351317167282,
0.2104269564151764, 0.2101719230413437, 0.2080526500940323],

'accuracy': [0.789354145526886, 0.8447291851043701, 0.8566874861717224, 0.8643541932106018, 0.8697083592414856, 0.8739166855812073, 0.8761041760444641,
0.8789583444595337, 0.8826666474342346, 0.8855208158493042, 0.886104166507721, 0.8880624771118164, 0.8906458616256714, 0.8928750157356262, 0.8933958411216736,
0.8956249952316284, 0.898104190826416, 0.8983749747276306, 0.8991875052452087, 0.9007708430290222, 0.9011458158493042, 0.9025416374206543, 0.9044791460037231,
0.9057499766349792, 0.9072708487510681, 0.9074375033378601, 0.9079999923706055, 0.9088541865348816, 0.9095208048820496, 0.9097291827201843, 0.9118333458900452,
0.9124166369438171, 0.9133541584014893, 0.9129791855812073, 0.9146041870117188, 0.9154791831970215, 0.9166874885559082, 0.9177916646003723, 0.9175416827201843,
0.9180625081062317, 0.9192083477973938, 0.9185208082199097, 0.9197708368301392, 0.9213333129882812, 0.9215624928474426, 0.9215624928474426, 0.9208750128746033,
0.9235208630561829, 0.922249972820282, 0.9248958230018616],

'val_loss': [0.4776163697242737, 0.4357101321220398, 0.40258854627609253, 0.39271479845046997, 0.3889164924621582, 0.3798101842403412, 0.3678809702396393,
0.36349910497665405, 0.37104806303977966, 0.3652504086494446, 0.35947203636169434, 0.37782320380210876, 0.3629121482372284, 0.3596421778202057, 0.34462788701057434,
0.35367244482040405, 0.3453534245491028, 0.35335132479667664, 0.36383312940597534, 0.35738077759742737, 0.35678377747535706, 0.35446837544441223, 0.3527243733406067,
0.34618350863456726, 0.34865111112594604, 0.3683689832687378, 0.3603420555591583, 0.36953479051589966, 0.36129820346832275, 0.3623996675014496, 0.36520129442214966,
0.36162319779396057, 0.36536312103271484, 0.3636190593242645, 0.37748828530311584, 0.37399259209632874, 0.35947826504707336, 0.36005476117134094, 0.3623170256614685,
0.3886697292327881, 0.37945523858070374, 0.37049585580825806, 0.3743937611579895, 0.3805387318134308, 0.38715872168540955, 0.3735528290271759, 0.38132739067077637,
0.37841248512268066, 0.4056456685066223, 0.3818448781967163],

'val_accuracy': [0.8355000019073486, 0.8462499976158142, 0.8567500114440918, 0.8633333444595337, 0.8607500195503235, 0.8647500276565552, 0.8702499866485596,
0.8715833425521851, 0.8691666722297668, 0.8738333582878113, 0.8743333220481873, 0.8690833449363708, 0.8713333606719971, 0.8762500286102295, 0.8793333172798157,
0.8767499923706055, 0.8804166913032532, 0.8774999976158142, 0.871916651725769, 0.877916693687439, 0.8771666884422302, 0.878083348274231, 0.879166626930237,
0.8802499771118164, 0.8808333277702332, 0.8762500286102295, 0.8765000104904175, 0.8713333606719971, 0.8774999976158142, 0.8772500157356262, 0.8774166703224182,
0.878250002861023, 0.8778333067893982, 0.8803333044052124, 0.8758333325386047, 0.8756666779518127, 0.8794999718666077, 0.8813333511352539, 0.8806666731834412,
0.8709999918937683, 0.8765833377838135, 0.8798333406448364, 0.8809999823570251, 0.878333330154419, 0.8744166493415833, 0.8804166913032532, 0.8792499899864197,
0.8797500133514404, 0.871999979019165, 0.8801666498184204]}
```

{'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]}

辞書型のデータの取得方法

print(result.history[‘loss’])

[0.6204796433448792, 0.4424095153808594, 0.4049777686595917, 0.38482892513275146, 0.3680422008037567, 0.3553660809993744,
0.34484514594078064, 0.3349671959877014, 0.32561713457107544, 0.3206530511379242, 0.3141988515853882, 0.30852627754211426,
0.3008130192756653, 0.29420095682144165, 0.29154443740844727, 0.28692877292633057, 0.28109729290008545, 0.28085023164749146,
0.27662160992622375, 0.2701963186264038, 0.26984351873397827, 0.26697129011154175, 0.26113253831863403, 0.25769904255867004,
0.2550542652606964, 0.2521992623806, 0.2518135905265808, 0.24781620502471924, 0.24636663496494293, 0.24498197436332703,
0.2412831038236618, 0.23945355415344238, 0.23669078946113586, 0.2358267456293106, 0.23242957890033722, 0.23221394419670105,
0.2298291176557541, 0.22631986439228058, 0.2247573286294937, 0.22372491657733917, 0.22302721440792084, 0.22342391312122345,
0.2184654176235199, 0.21663862466812134, 0.21550878882408142, 0.21314330399036407, 0.2136351317167282, 0.2104269564151764,
0.2101719230413437, 0.2080526500940323]

出力結果は各エポックの誤差がリストになっている

x = [要素,要素,...,要素]

a = [1,1,3,3,3]
これはリスト型

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

b = {‘name’:’sudo’,’age’:36}
これは辞書型

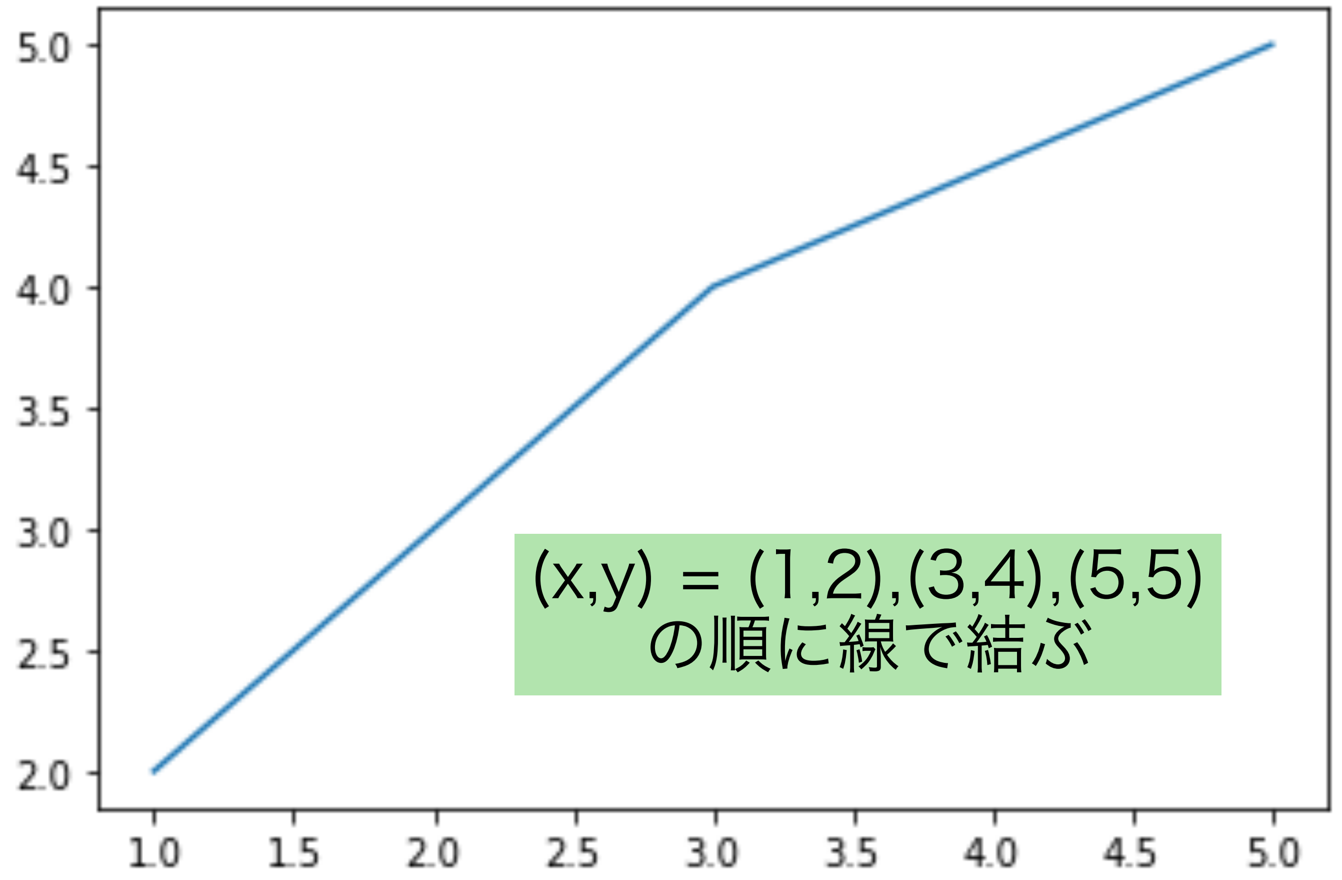
```
b = {'name':'sudo','age':36}
print(type(b))
print(b['name'])
print(b['age'])
```

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

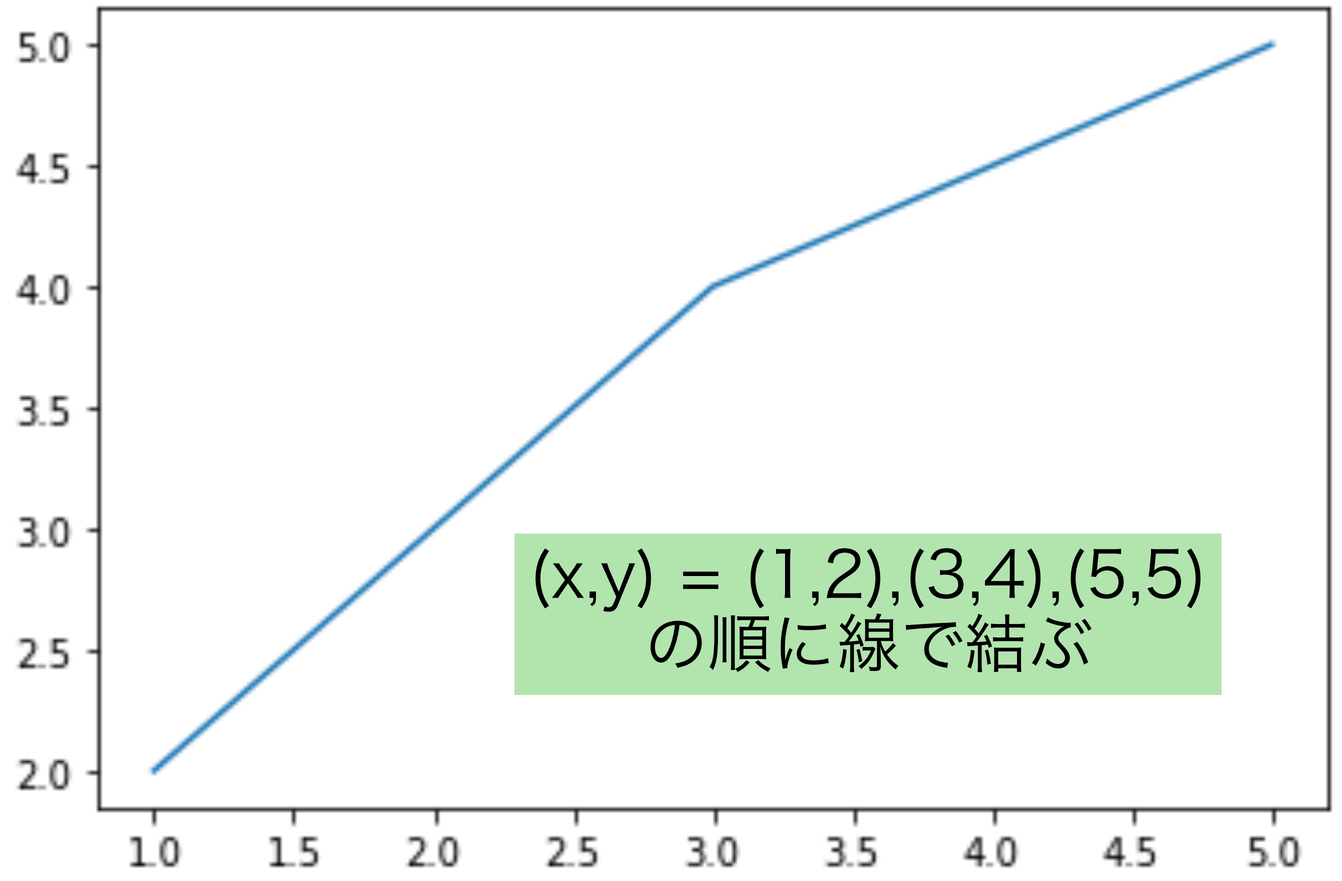
辞書型は変数名[key]で
valueを取り出せる!

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

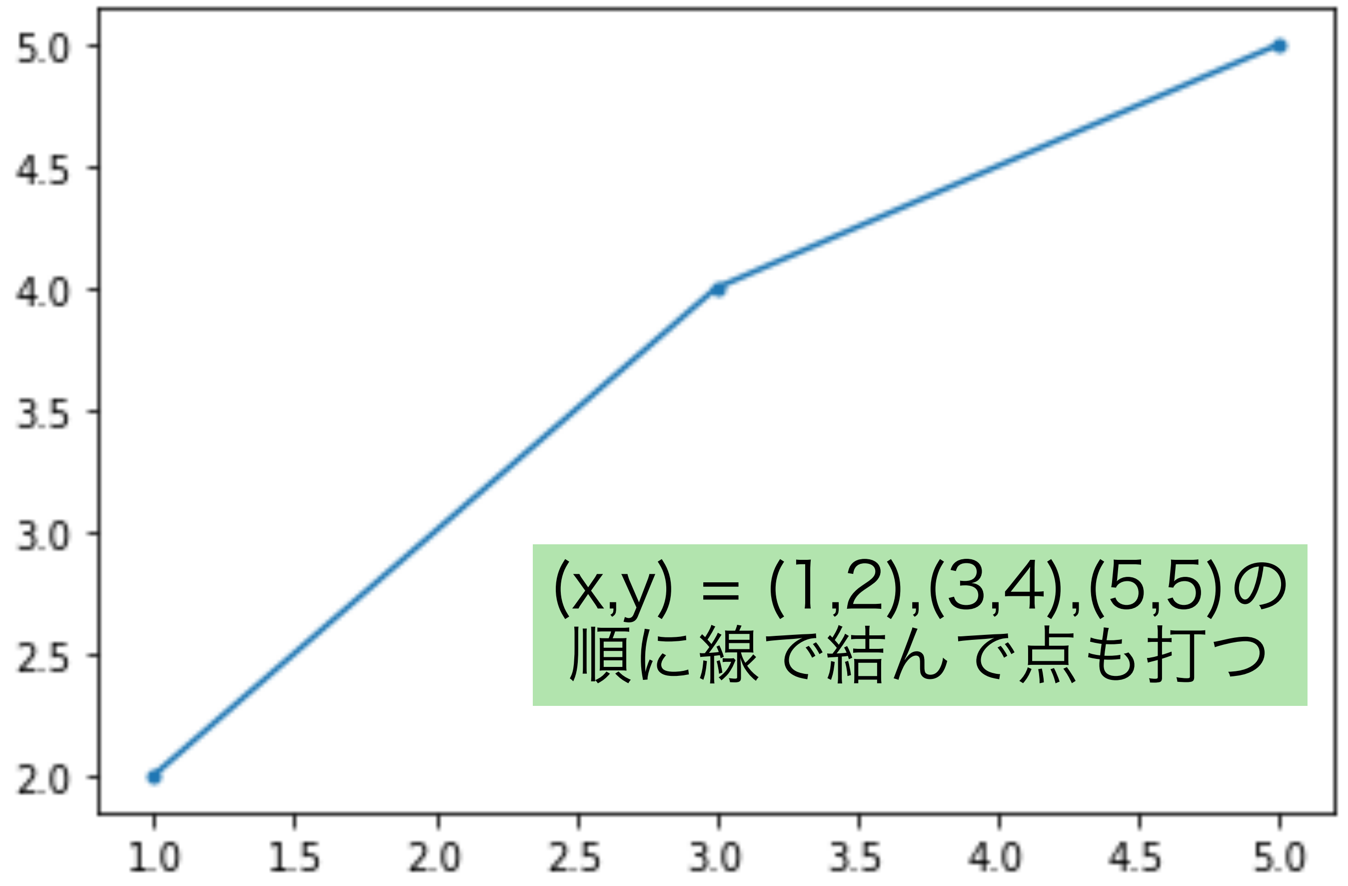
```
x = [1,3,5]
y = [2,4,5]
plt.plot(x,y)
plt.show()
```



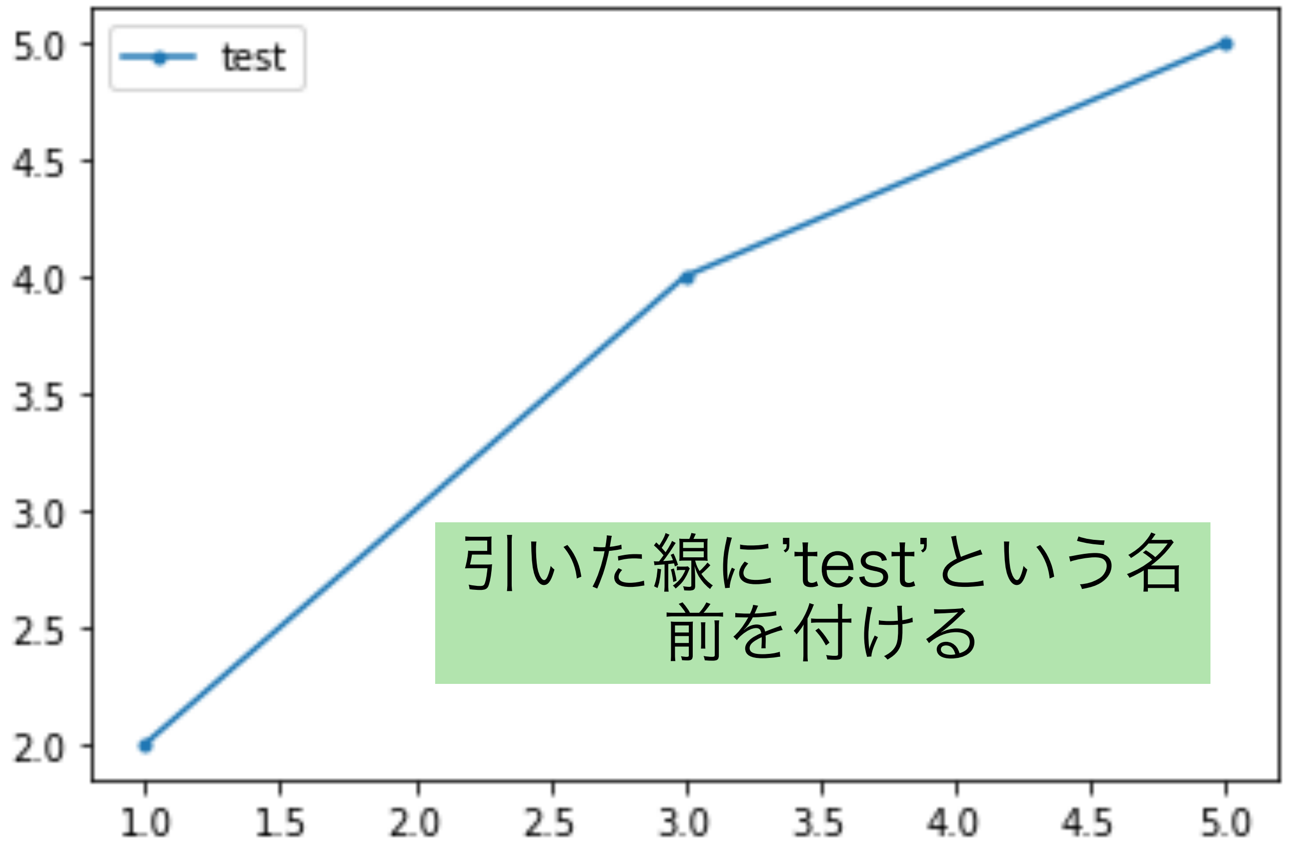
```python
x = [1,3,5]
y = [2,4,5]
plt.plot(x,y)
plt.show()
```



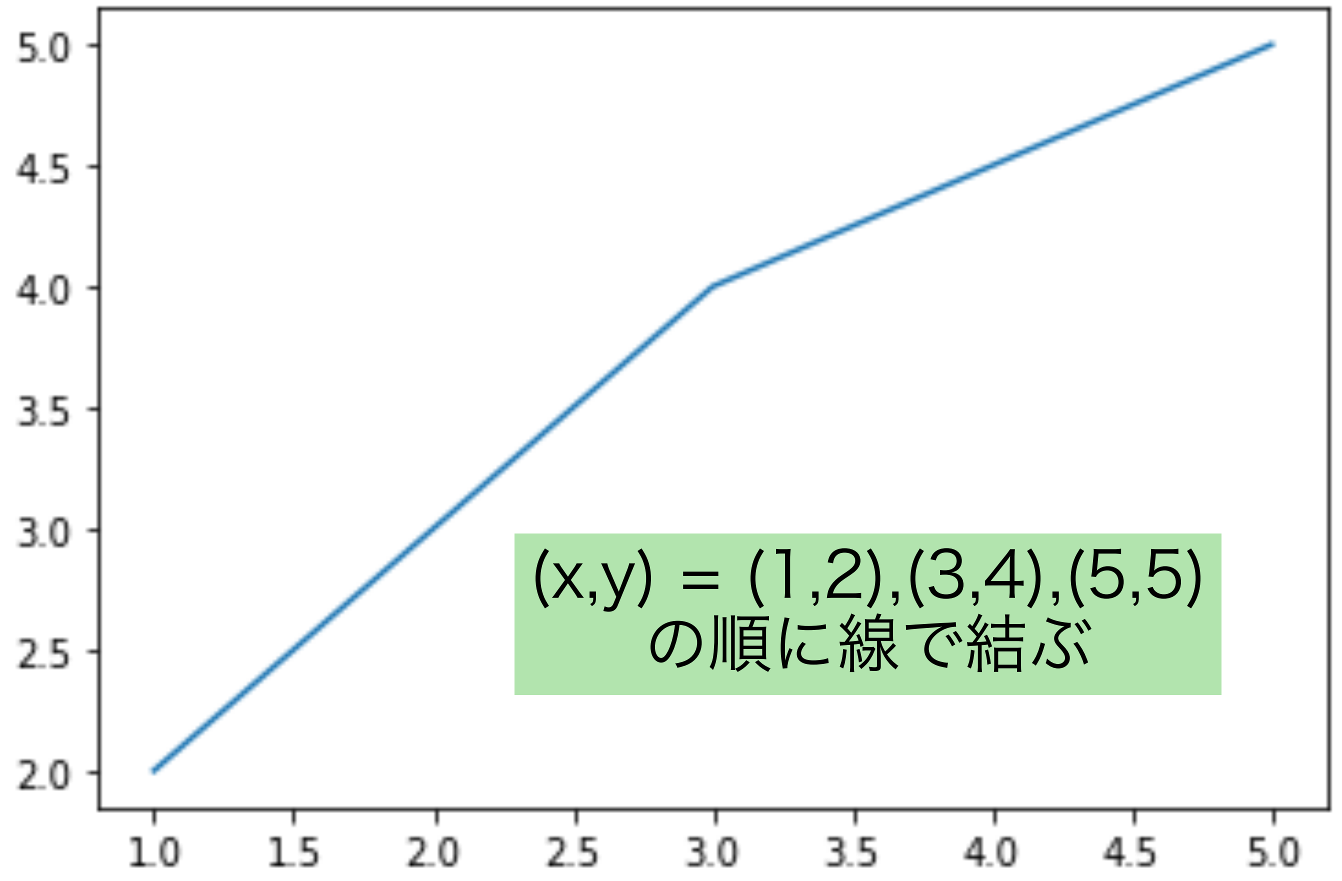

```python
x = [1,3,5]
y = [2,4,5]
plt.plot(x,y,marker='.')
plt.show()
```



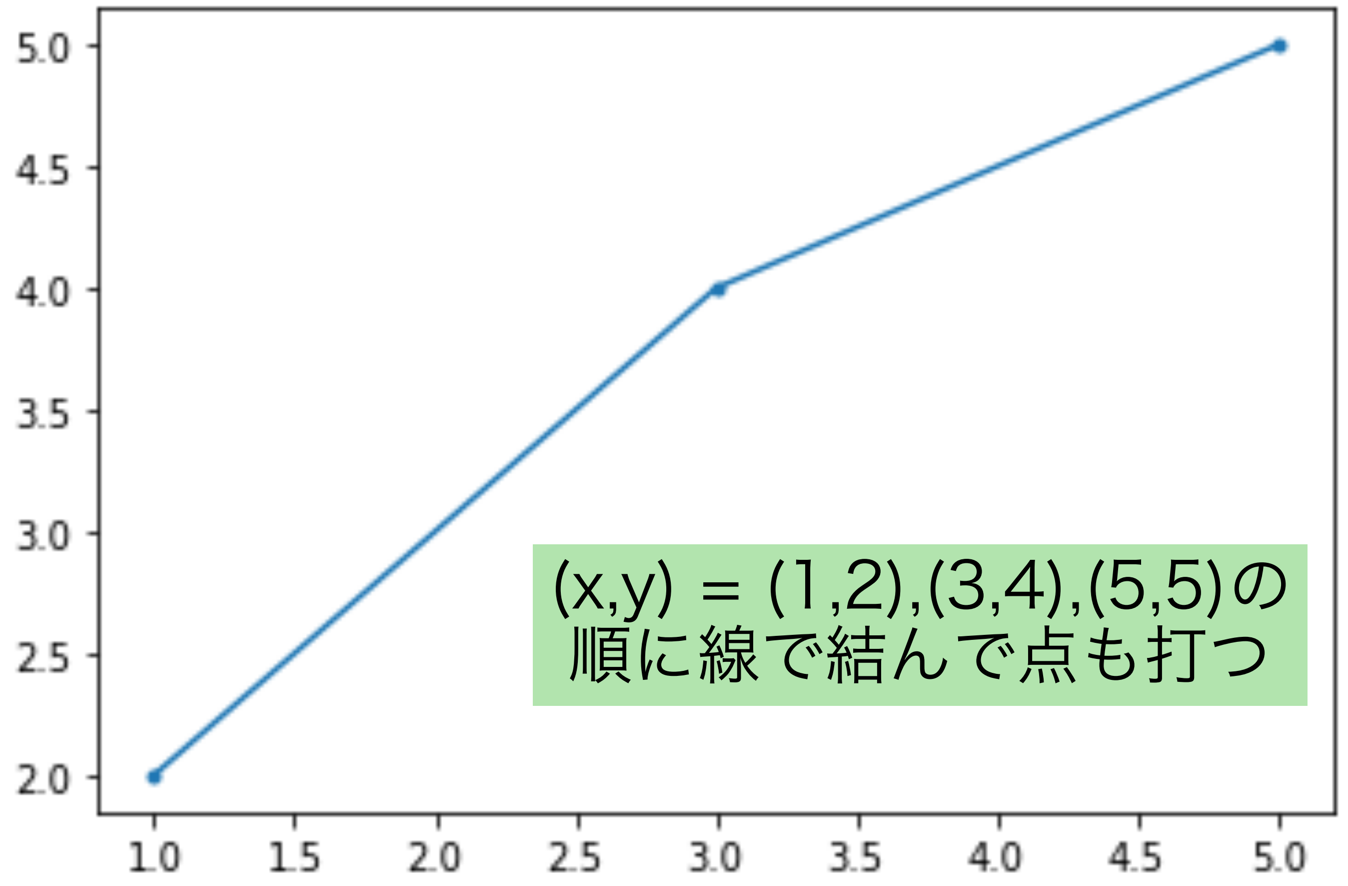

```python
x = [1,3,5]
y = [2,4,5]
plt.plot(x,y,marker='.',label='test')
plt.legend()
plt.show()
```



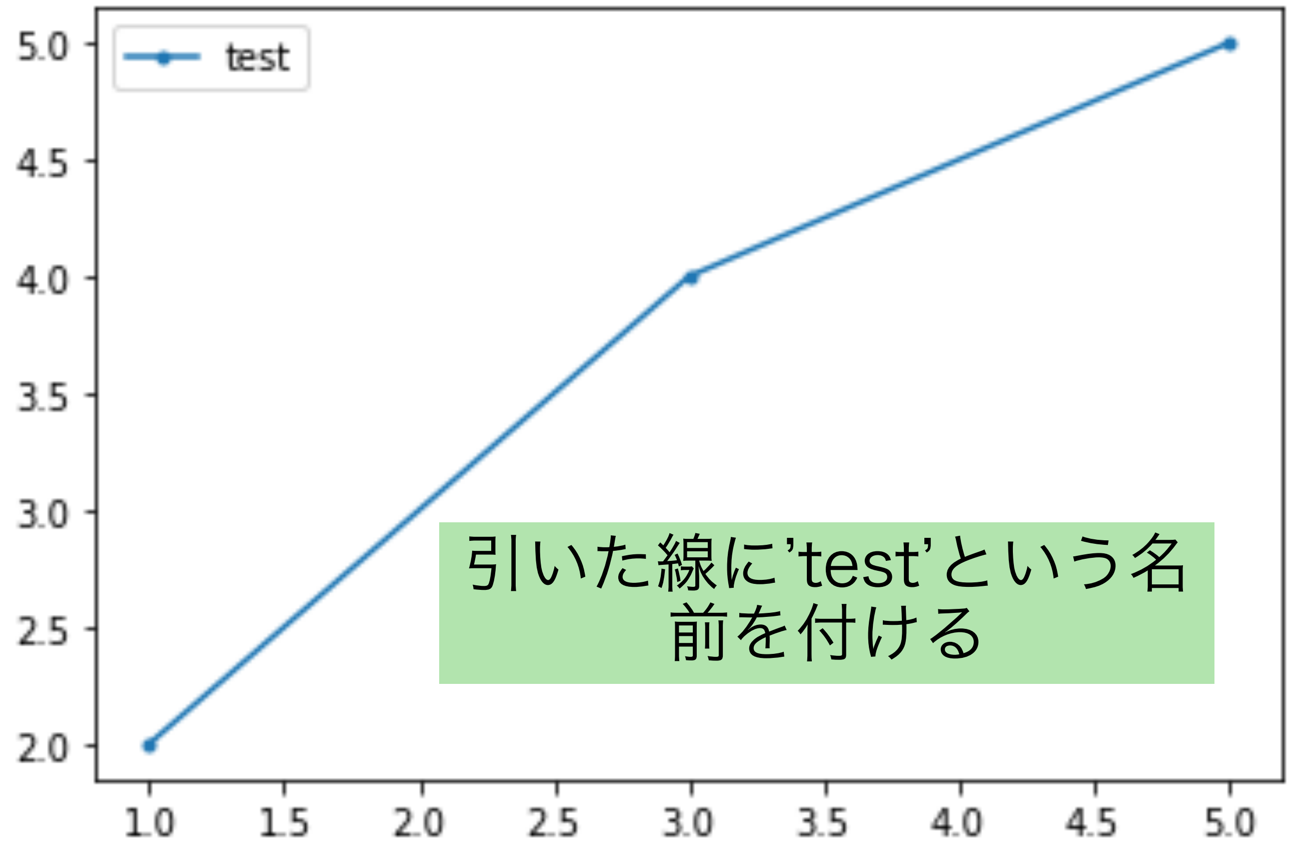
```
x = [1,3,5]
y = [2,4,5]
plt.plot(x,y)
plt.show()
```



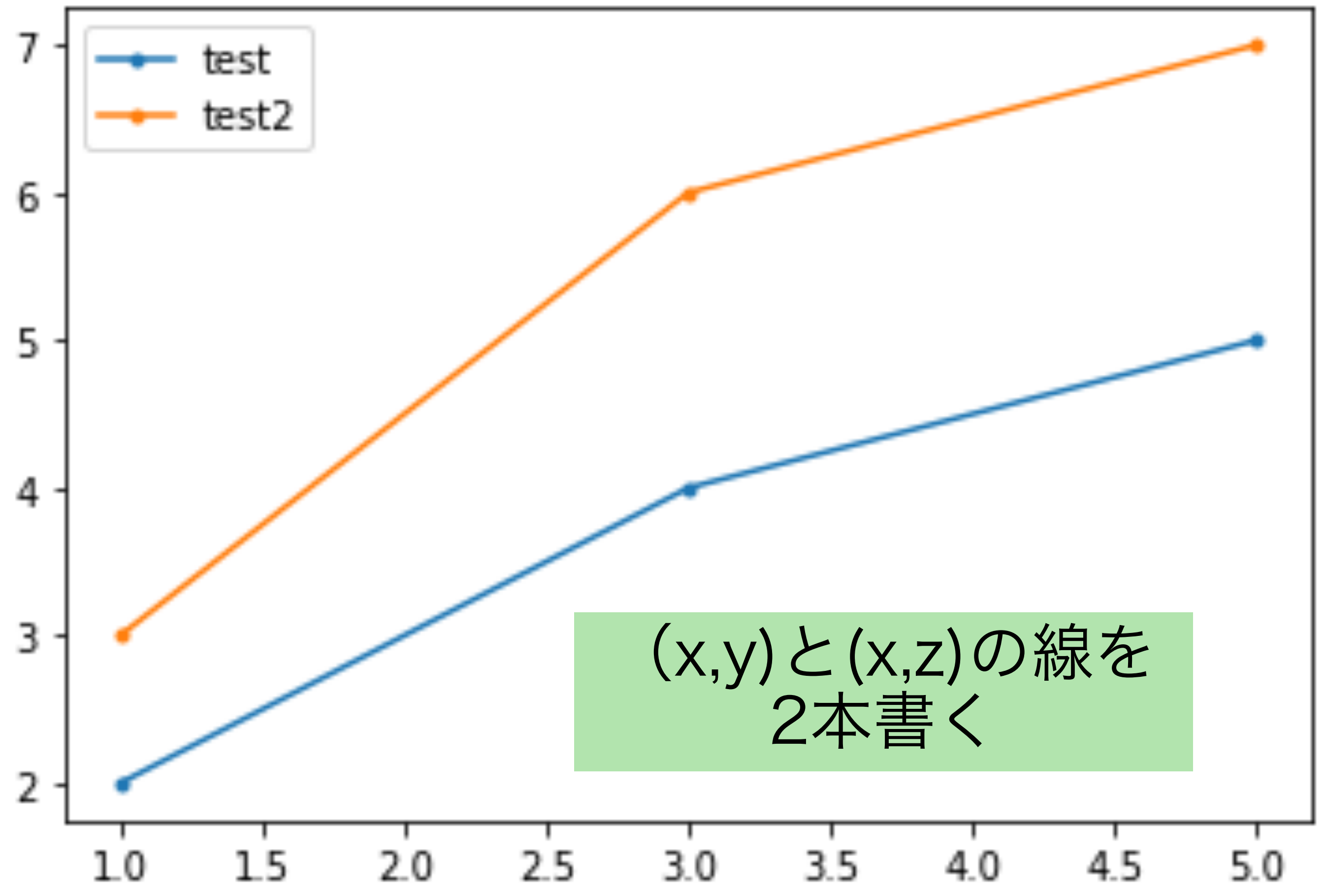

```
x = [1,3,5]
y = [2,4,5]
plt.plot(x,y,marker='.')
plt.show()
```



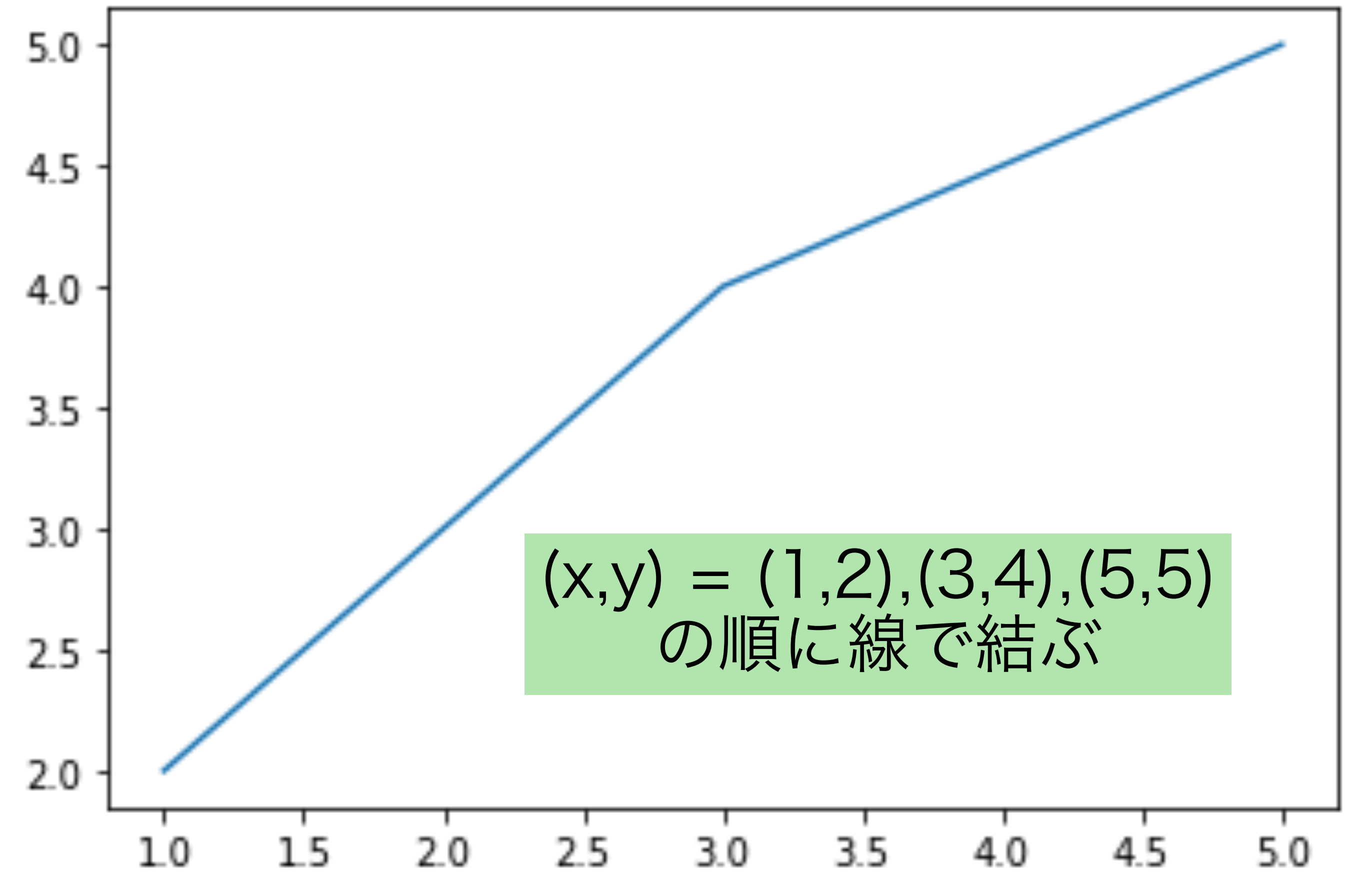

```
x = [1,3,5]
y = [2,4,5]
plt.plot(x,y,marker='.',label='test')
plt.legend()
plt.show()
```



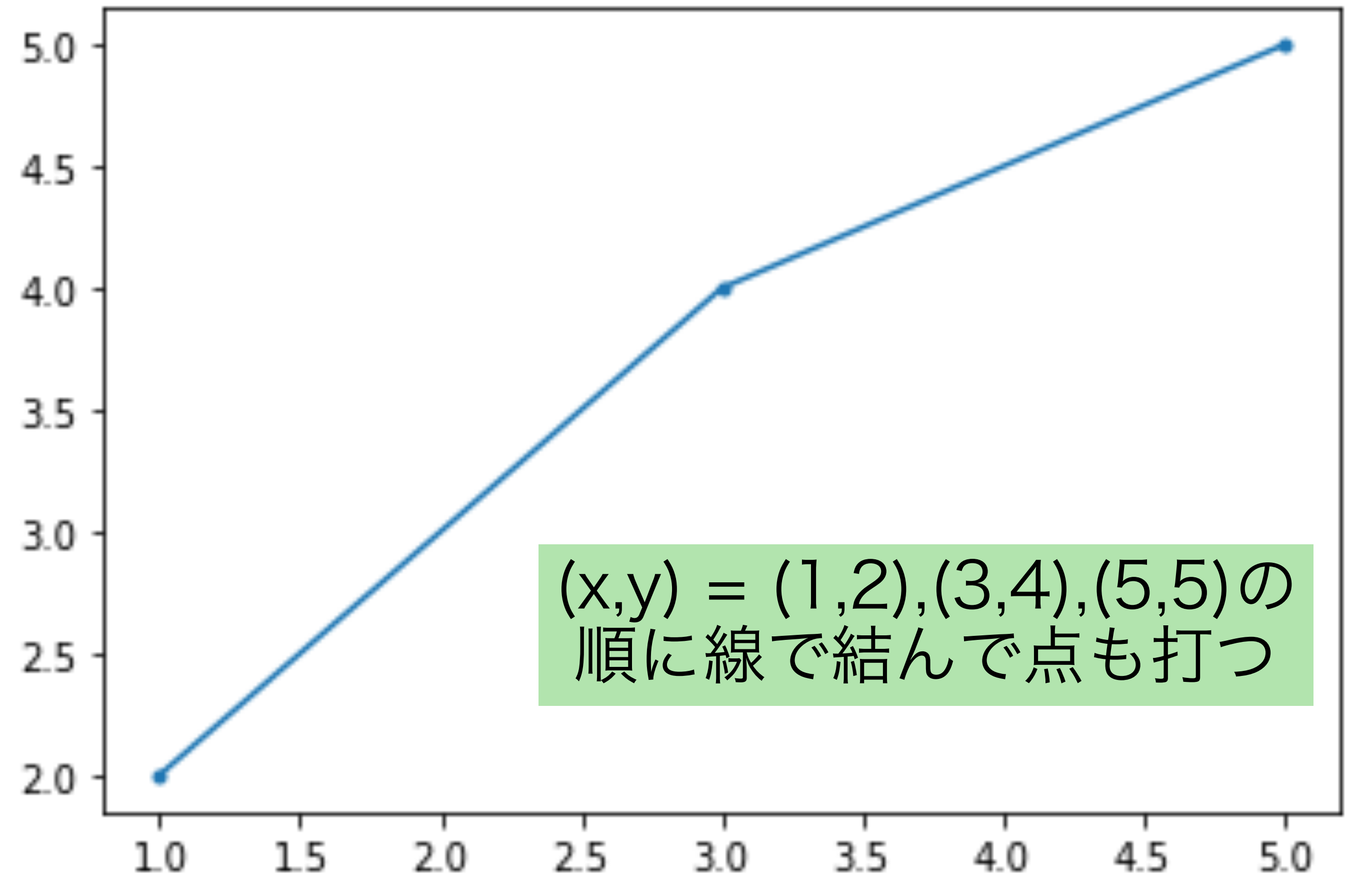

```
x = [1,3,5]
y = [2,4,5]
z = [3,6,7]
plt.plot(x,y,marker='.',label='test')
plt.plot(x,z,marker='.',label='test2')
plt.legend()
plt.show()
```



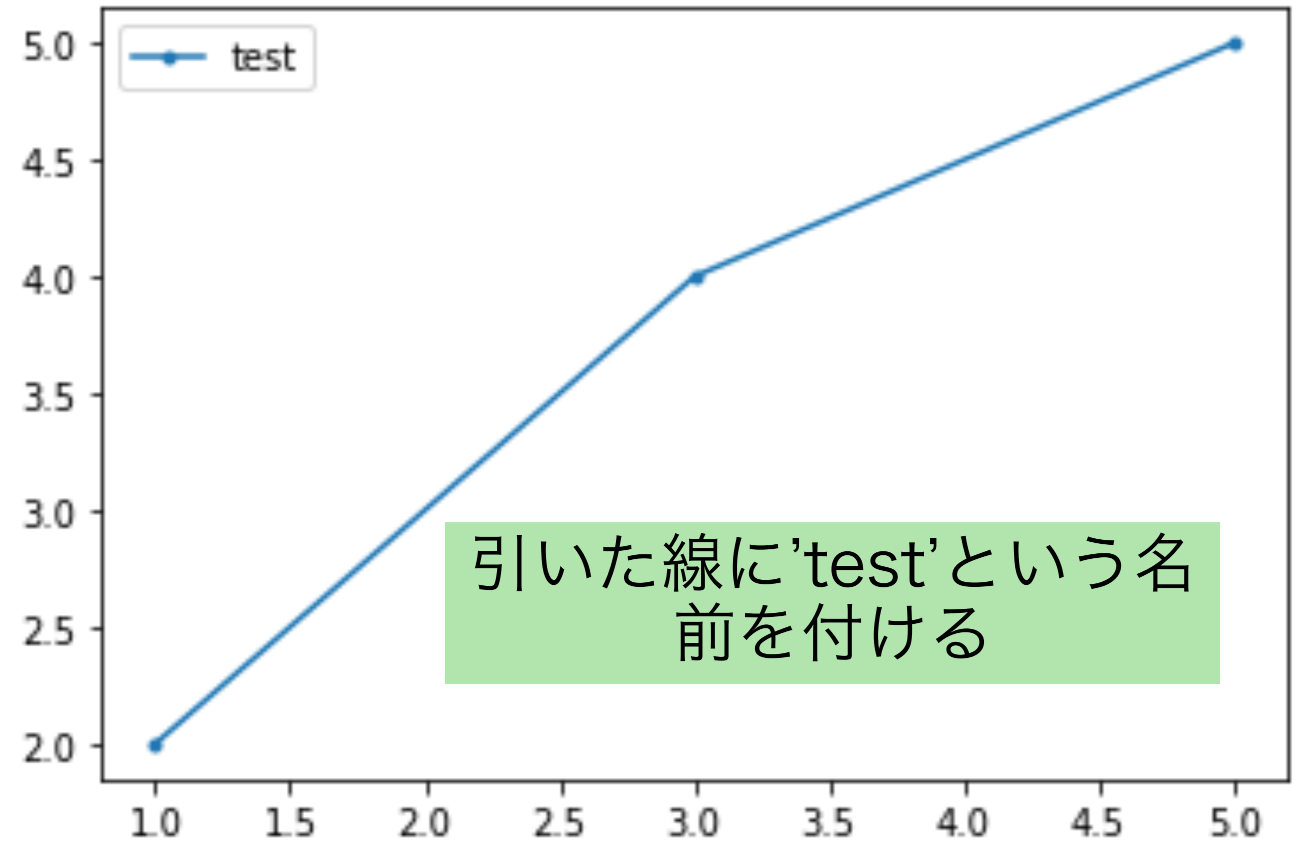
```
x = [1,3,5]
y = [2,4,5]
plt.plot(x,y)
plt.show()
```



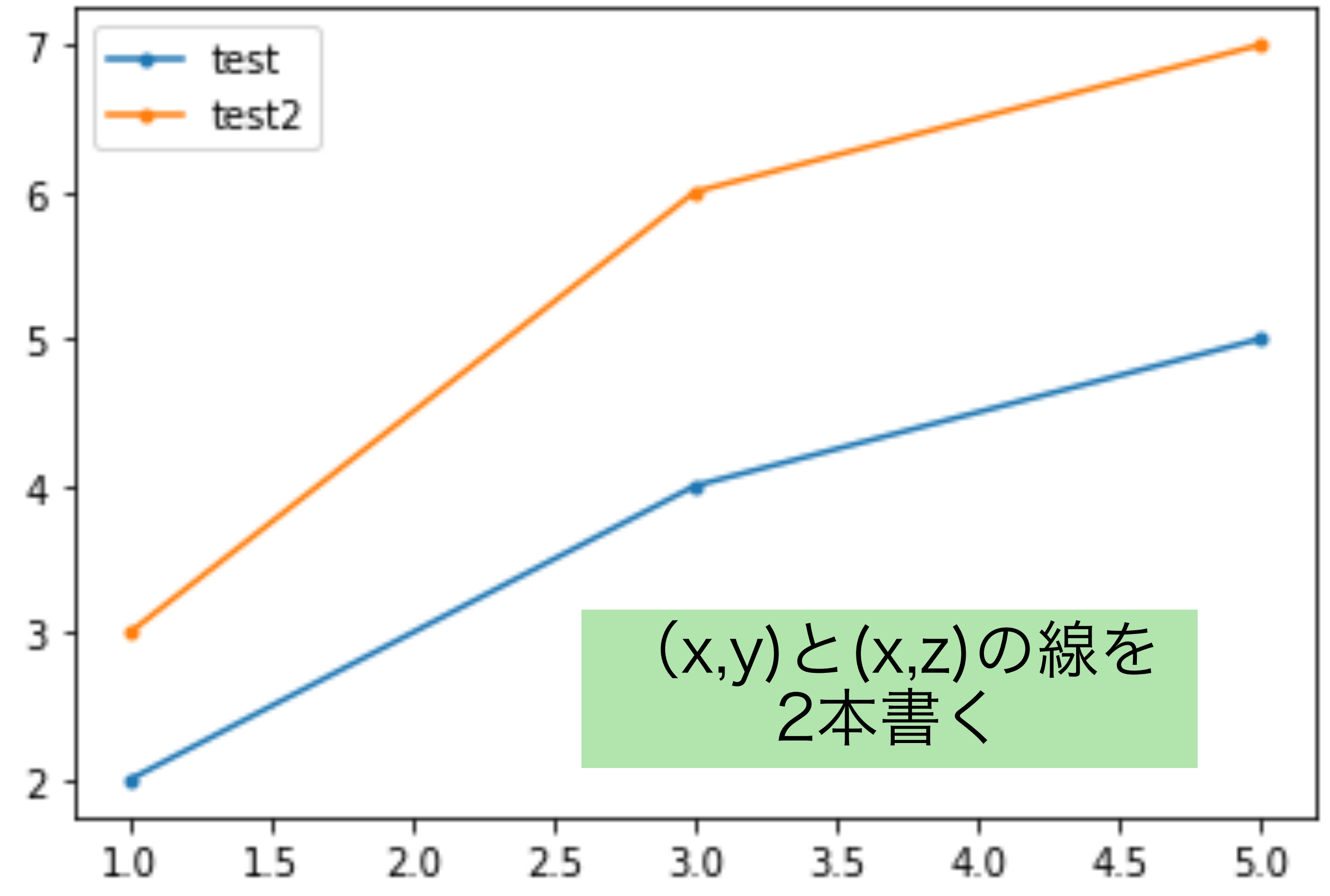

```
x = [1,3,5]
y = [2,4,5]
plt.plot(x,y,marker='.')
plt.show()
```



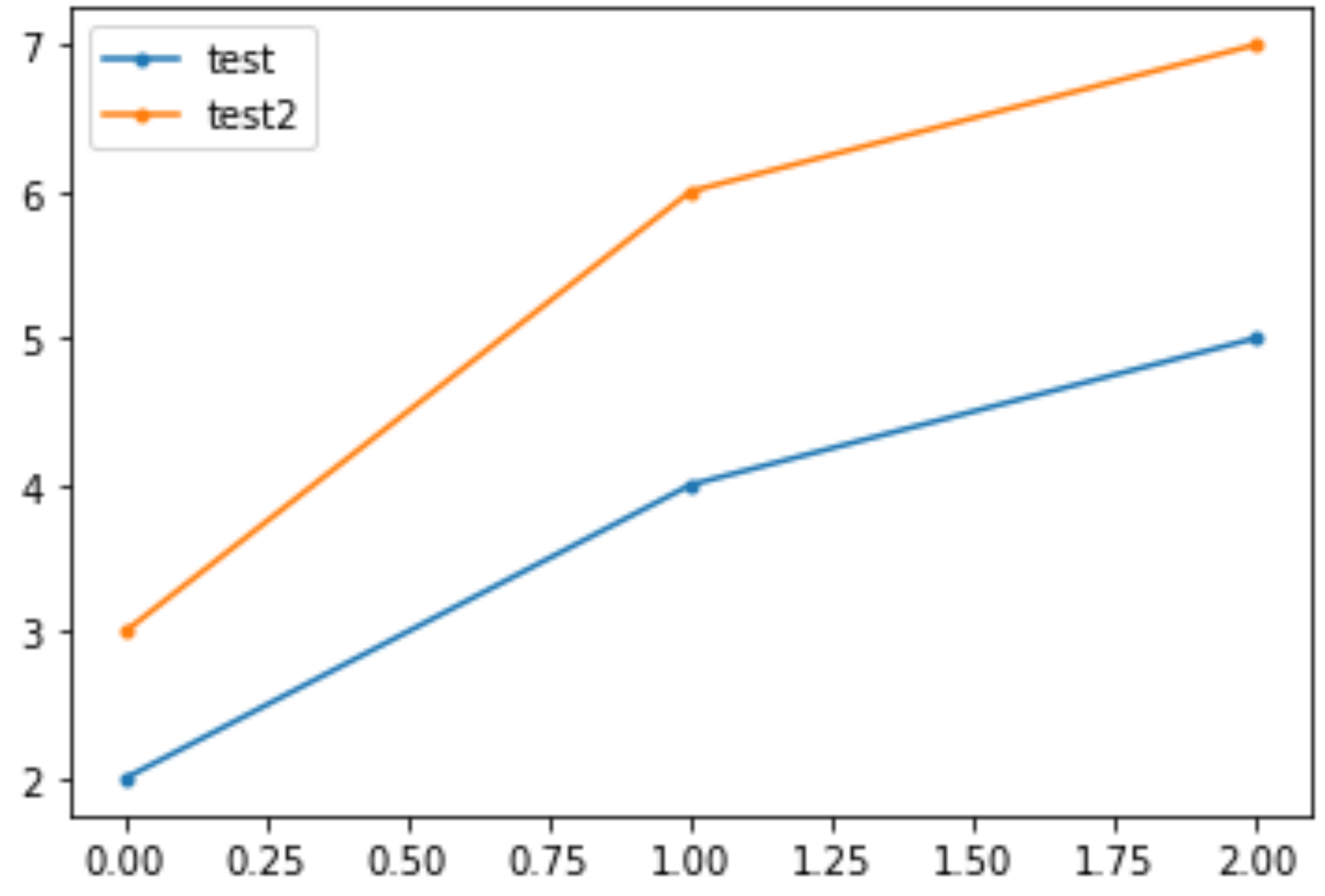

```
x = [1,3,5]
y = [2,4,5]
plt.plot(x,y,marker='.',label='test')
plt.legend()
plt.show()
```




```
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()
```

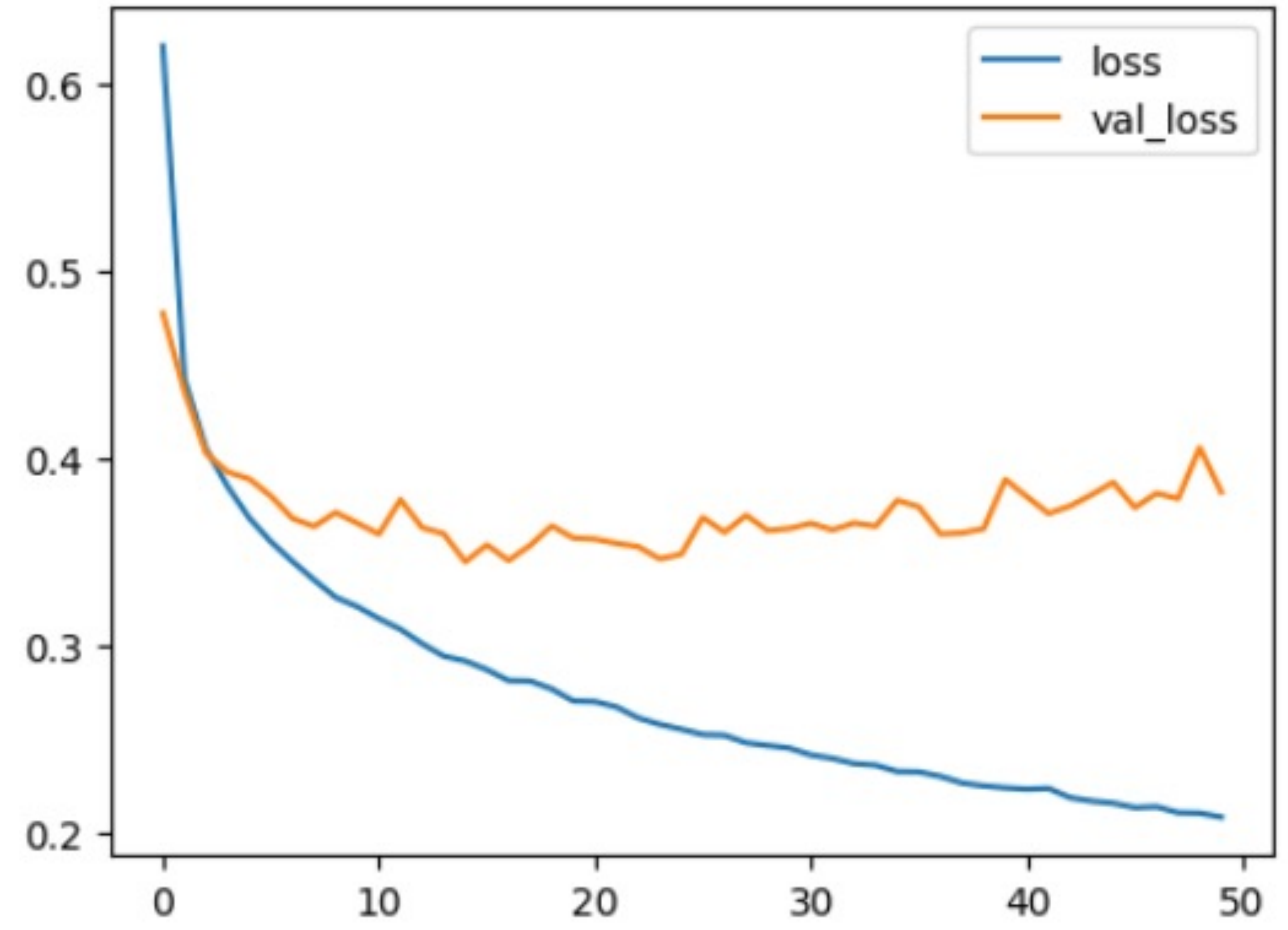


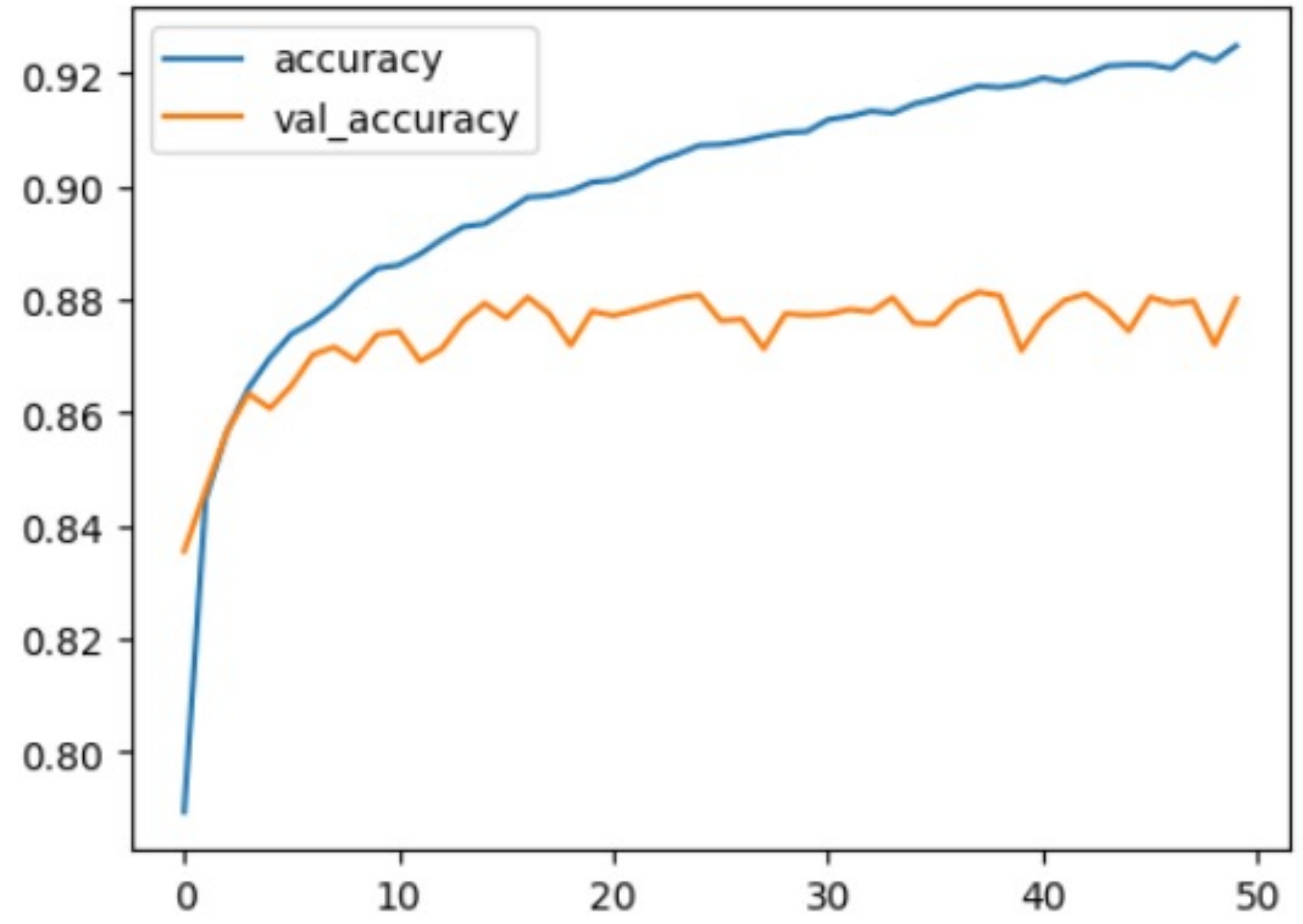
x軸の変数が与えられない時はy軸の個数だけ順に0,1,2,3,..と与えられる。

```
plt.plot(result.history['loss'],label='loss')
plt.plot(result.history['val_loss'],label='val_loss')
```

```
'loss': [0.6204796433448792, 0.4424095153808594, 0.4049777686595917, 0.38482892513275146, 0.3680422008037567,
0.3553660809993744, 0.34484514594078064, 0.3349671959877014, 0.32561713457107544, 0.3206530511379242, 0.3141988515853882,
0.30852627754211426, 0.3008130192756653, 0.29420095682144165, 0.29154443740844727, 0.28692877292633057,
0.28109729290008545, 0.28085023164749146, 0.27662160992622375, 0.2701963186264038, 0.26984351873397827,
0.26697129011154175, 0.26113253831863403, 0.25769904255867004, 0.2550542652606964, 0.2521992623806, 0.2518135905265808,
0.24781620502471924, 0.24636663496494293, 0.24498197436332703, 0.2412831038236618, 0.23945355415344238,
0.23669078946113586, 0.23582674562931061, 0.23242957890033722, 0.23221394419670105, 0.2298291176557541, 0.22631986439228058,
0.2247573286294937, 0.22372491657733917, 0.22302721440792084, 0.22342391312122345, 0.2184654176235199, 0.21663862466812134,
0.21550878882408142, 0.21314330399036407, 0.2136351317167282, 0.2104269564151764, 0.2101719230413437, 0.2080526500940323],

'val_loss': [0.4776163697242737, 0.4357101321220398, 0.40258854627609253, 0.39271479845046997, 0.3889164924621582,
0.3798101842403412, 0.3678809702396393, 0.36349910497665405, 0.37104806303977966, 0.3652504086494446, 0.35947203636169434,
0.37782320380210876, 0.3629121482372284, 0.3596421778202057, 0.34462788701057434, 0.35367244482040405, 0.3453534245491028,
0.35335132479667664, 0.36383312940597534, 0.35738077759742737, 0.35678377747535706, 0.35446837544441223,
0.3527243733406067, 0.34618350863456726, 0.34865111112594604, 0.3683689832687378, 0.3603420555591583, 0.36953479051589966,
0.36129820346832275, 0.3623996675014496, 0.36520129442214966, 0.36162319779396057, 0.36536312103271484, 0.3636190593242645,
0.37748828530311584, 0.37399259209632874, 0.35947826504707336, 0.36005476117134094, 0.3623170256614685, 0.3886697292327881,
0.37945523858070374, 0.37049585580825806, 0.37439376115798950, 0.38053873181343080, 0.38715872168540955, 0.3735528290271759,
0.38132739067077637, 0.37841248512268066, 0.4056456685066223, 0.3818448781967163],
```

どちらも要素の数が50個のリスト
xは[0,1,2,...,49]が省略されている





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

model.evaluate()で評価

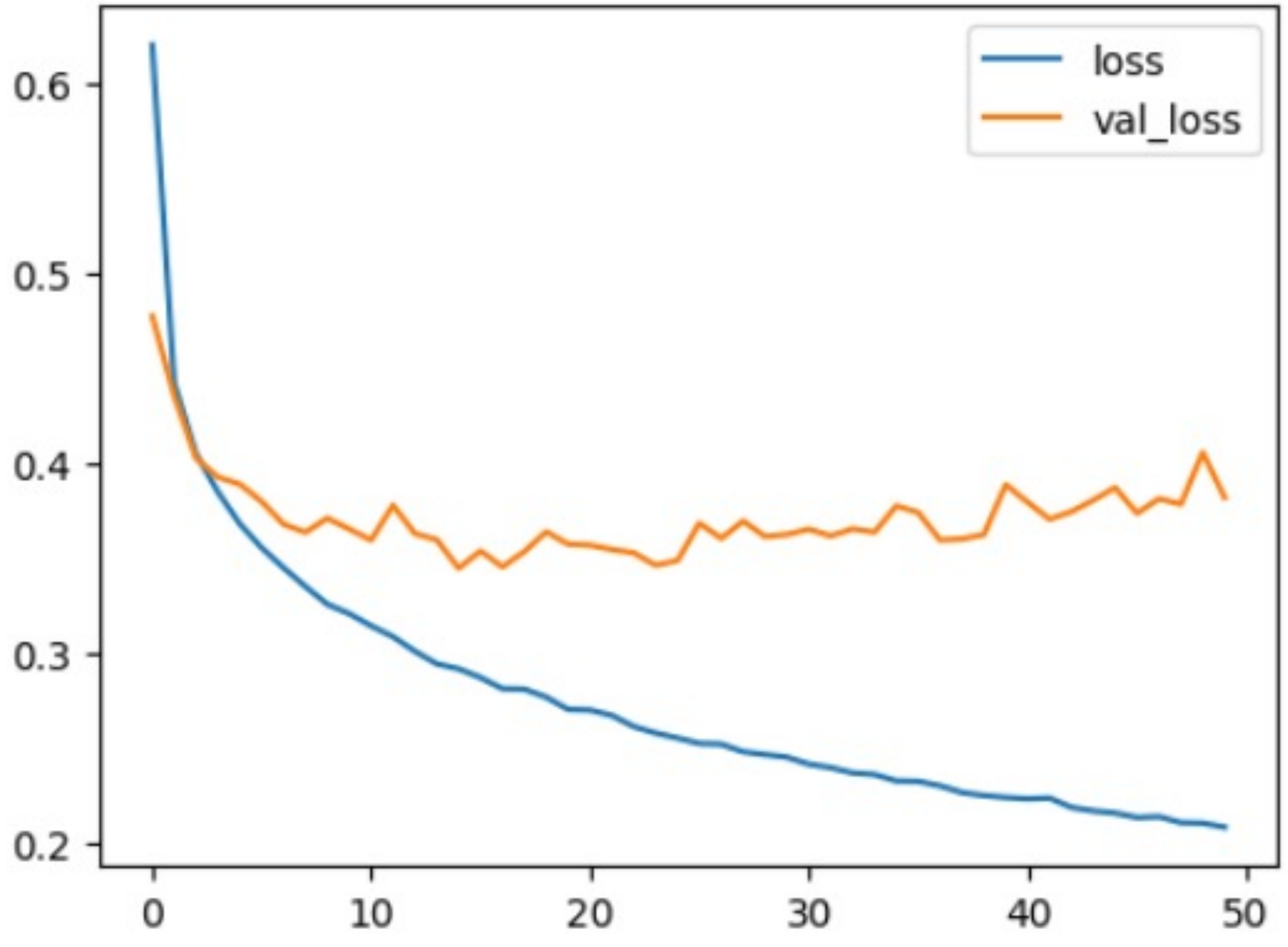
出来上がったモデルを別で用意している10000枚の画像で評価する

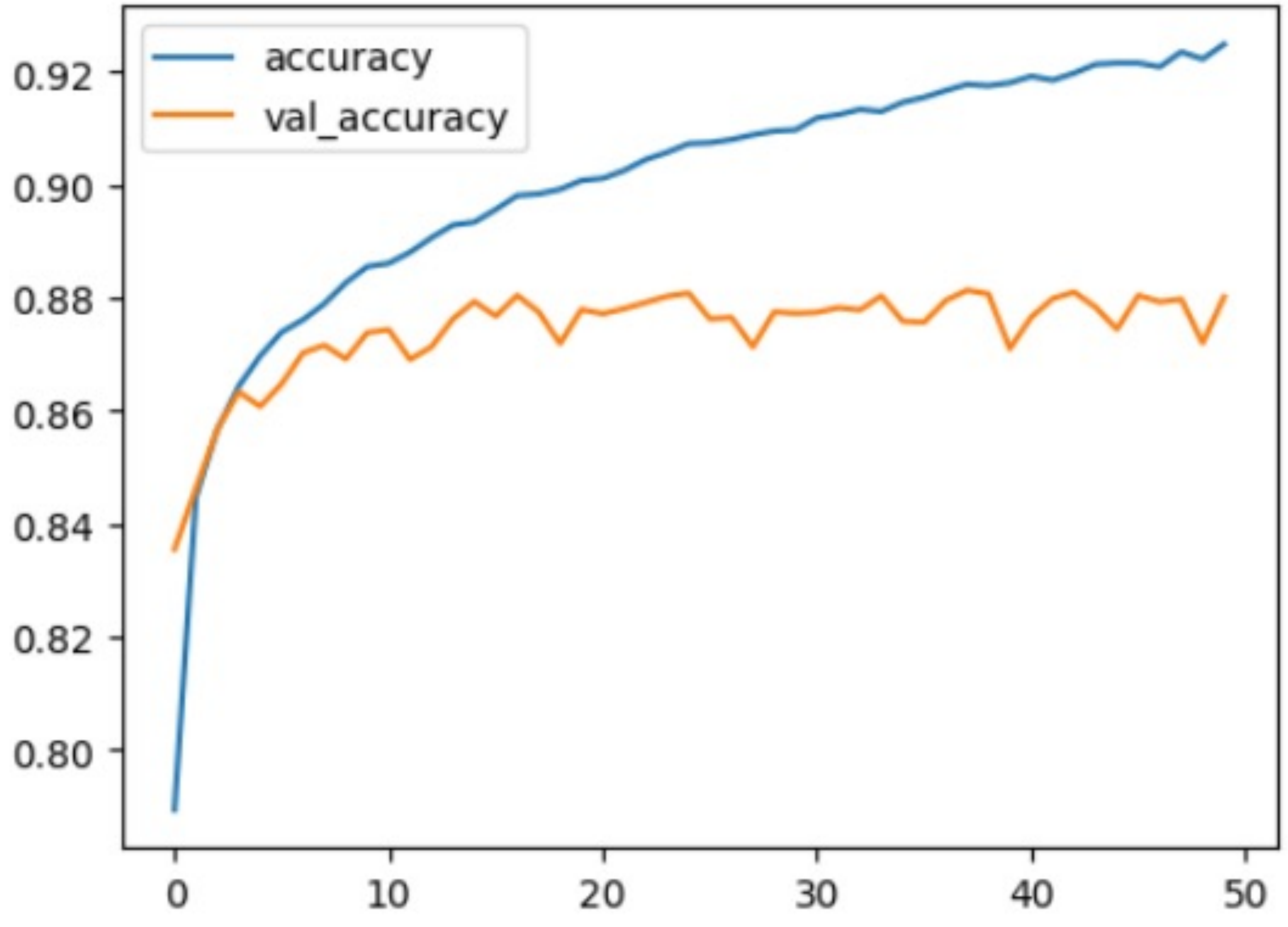
Test loss: 0.40565115213394165
Test accuracy: 0.8694000244140625

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: 0.40565115213394165
Test accuracy: 0.8694000244140625

もっと精度をあげたい

ニューロンの数を増やす

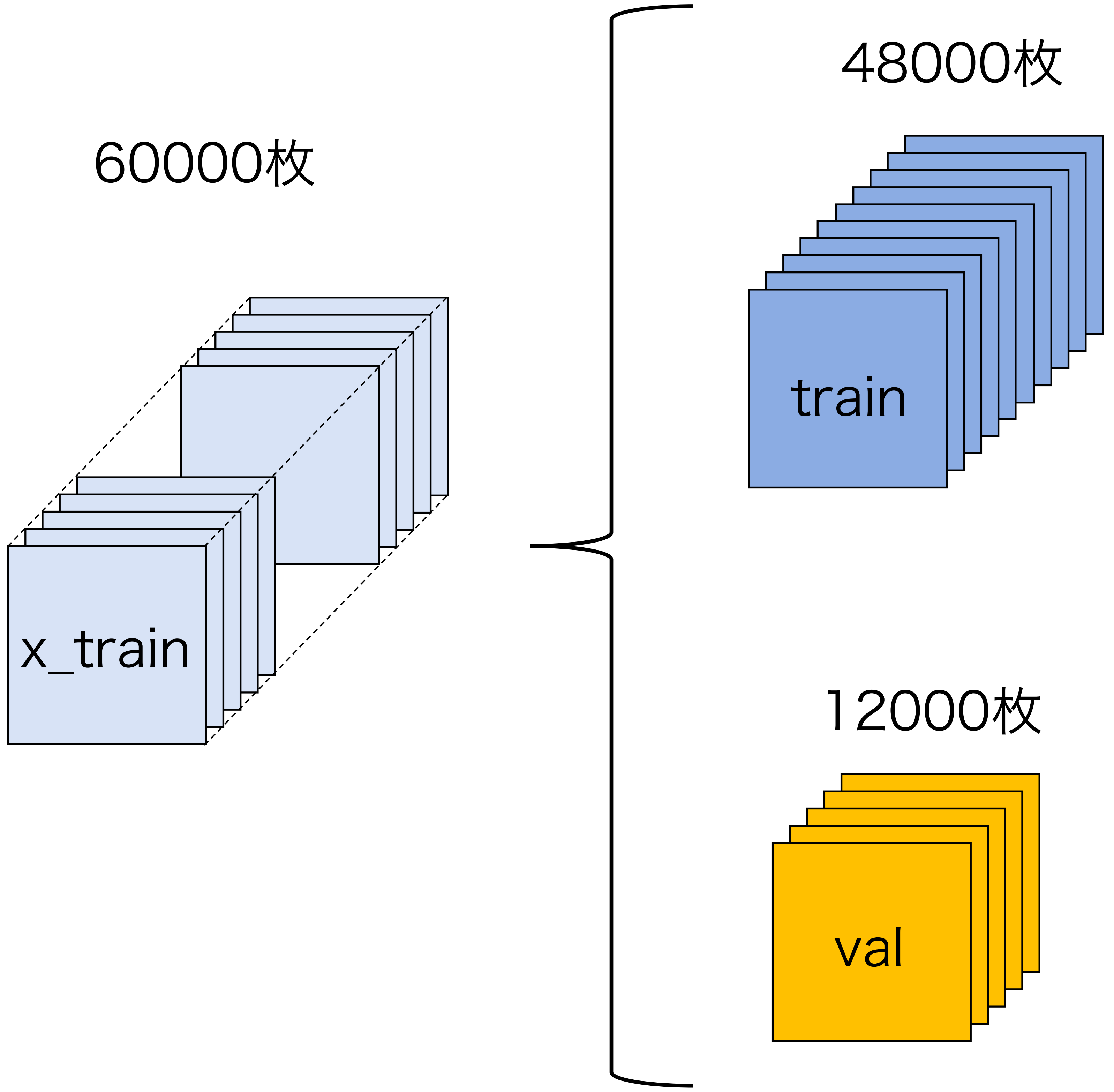
```
model = Sequential()
model.add(Dense(32,input_shape=(784,),activation='relu'))
model.add(Dense(10,activation='softmax'))
model.compile(loss='categorical_crossentropy',
                    optimizer='Adam',metrics=['accuracy'])
model.summary()
```

```
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
_________________________________________________________________
```

```
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()
```

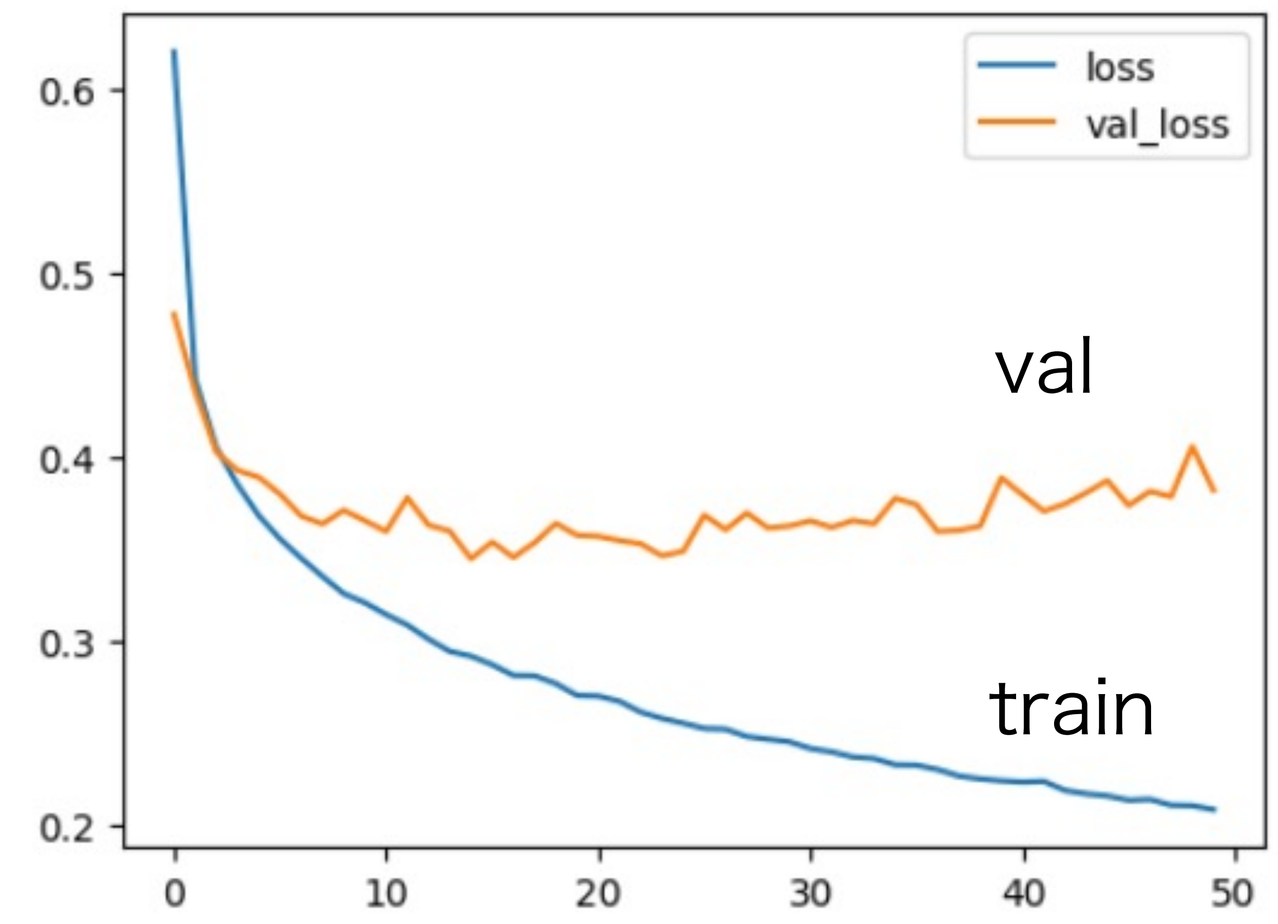
```
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_9 (Dense)              (None, 10)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

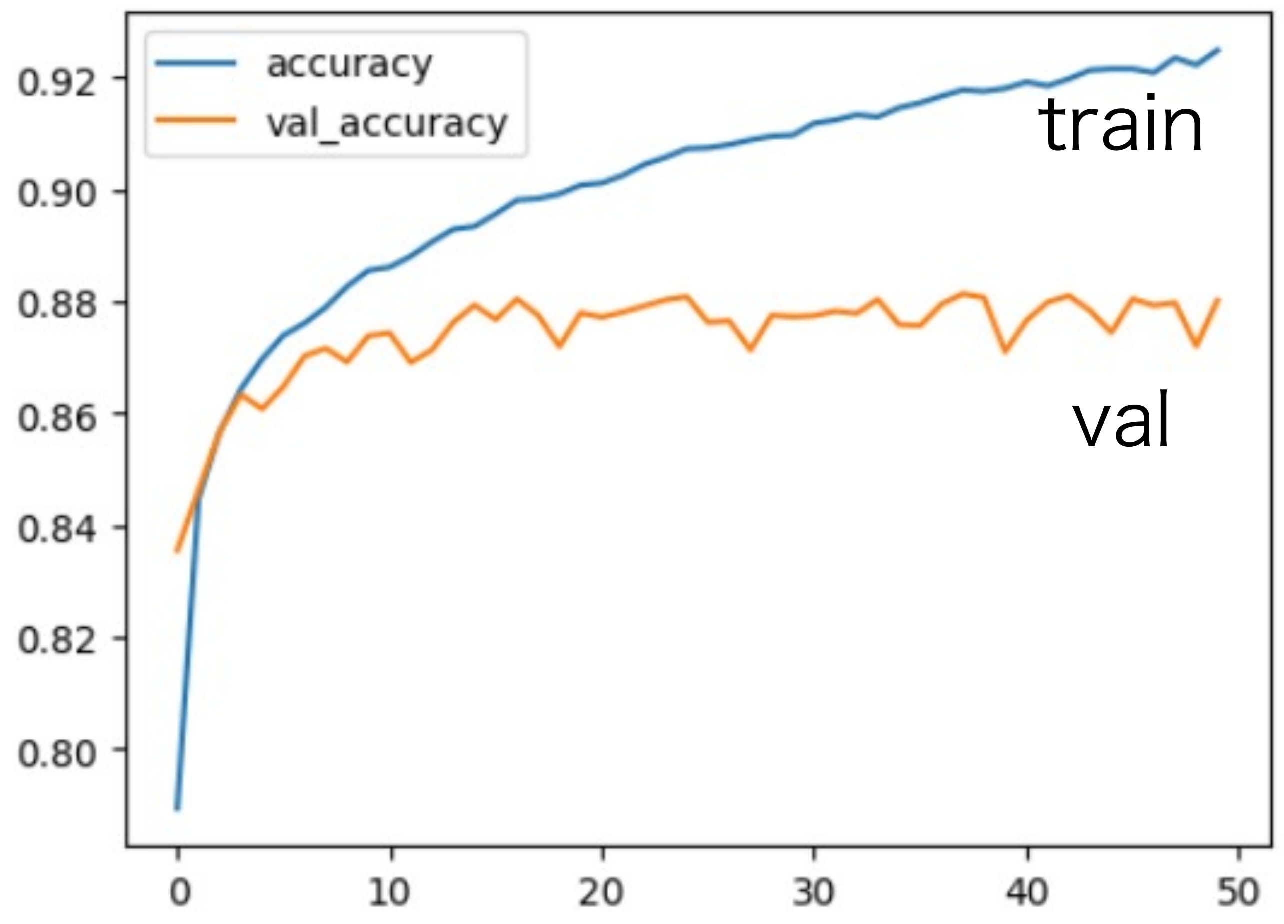
結果の分析

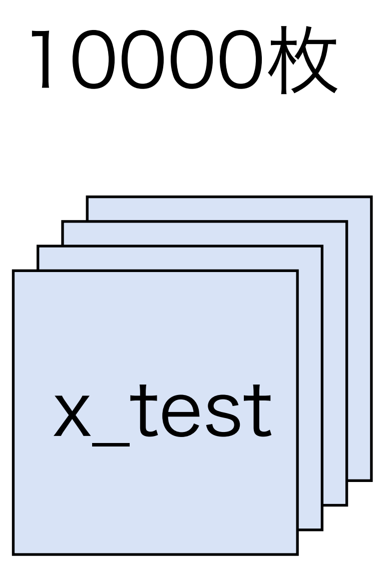


①trainを64枚ずつランダムに取り出して、学習、更新(750回/epoch)

②1epoch終了後、trainとval両方でlossとaccuracyを算出(50epoch)







③最後にtestで最終評価(lossとaccuracy)

Test loss: 0.40565115213394165
Test accuracy: 0.8694000244140625

result.historyの中身について

print(result.history)

```
{'loss': [0.6204796433448792, 0.4424095153808594, 0.4049777686595917, 0.38482892513275146, 0.3680422008037567, 0.3553660809993744, 0.34484514594078064,
0.3349671959877014, 0.32561713457107544, 0.3206530511379242, 0.3141988515853882, 0.30852627754211426, 0.3008130192756653, 0.29420095682144165, 0.29154443740844727,
0.28692877292633057, 0.28109729290008545, 0.28085023164749146, 0.27662160992622375, 0.2701963186264038, 0.26984351873397827, 0.26697129011154175, 0.26113253831863403,
0.25769904255867004, 0.2550542652606964, 0.2521992623806, 0.2518135905265808, 0.24781620502471924, 0.24636663496494293, 0.24498197436332703, 0.241283103823661 8,
0.23945355415344238, 0.23669078946113586, 0.23582674562931 06, 0.23242957890033722, 0.23221394419670105, 0.2298291176557541, 0.22631986439228058, 0.2247573286294937,
0.22372491657733917, 0.22302721440792084, 0.22342391312122345, 0.2184654176235199, 0.21663862466812134, 0.21550878882408142, 0.21314330399036407, 0.2136351317167282,
0.2104269564151764, 0.2101719230413437, 0.2080526500940323],

'accuracy': [0.789354145526886, 0.8447291851043701, 0.8566874861717224, 0.8643541932106018, 0.8697083592414856, 0.8739166855812073, 0.8761041760444641,
0.8789583444595337, 0.8826666474342346, 0.8855208158493042, 0.886104166507721, 0.8880624771118164, 0.8906458616256714, 0.8928750157356262, 0.8933958411216736,
0.8956249952316284, 0.898104190826416, 0.8983749747276306, 0.8991875052452087, 0.9007708430290222, 0.9011458158493042, 0.9025416374206543, 0.9044791460037231,
0.9057499766349792, 0.9072708487510681, 0.9074375033378601, 0.9079999923706055, 0.9088541865348816, 0.9095208048820496, 0.9097291827201843, 0.9118333458900452,
0.912416636943817 1, 0.9133541584014893, 0.9129791855812073, 0.9146041870117188, 0.9154791831970215, 0.9166874885559082, 0.9177916646003723, 0.9175416827201843,
0.9180625081062317, 0.9192083477973938, 0.9185208082199097, 0.9197708368301392, 0.9213333129882812, 0.9215624928474426, 0.9215624928474426, 0.9208750128746033,
0.9235208630561829, 0.922249972820282, 0.9248958230018616],

'val_loss': [0.4776163697242737, 0.4357101321220398, 0.40258854627609253, 0.39271479845046997, 0.3889164924621582, 0.3798101842403412, 0.3678809702396393,
0.36349910497665405, 0.37104806303977966, 0.3652504086494446, 0.35947203636169434, 0.37782320380210876, 0.3629121482372284, 0.3596421778202057, 0.34462788701057434,
0.35367244482040405, 0.3453534245491028, 0.35335132479667664, 0.36383312940597534, 0.35738077759742737, 0.35678377747535706, 0.35446837544441223, 0.3527243733406067,
0.34618350863456726, 0.34865111112594604, 0.3683689832687378, 0.3603420555591583, 0.36953479051589966, 0.3612982034683 2275, 0.36239966750144 96, 0.3652012944221 4966,
0.36162319779396057, 0.36536312103271484, 0.3636190593242645, 0.37748828530311584, 0.37399259209632874, 0.35947826504707336, 0.36005476117134094, 0.3623170256614685,
0.3886697292327881, 0.37945523858070374, 0.37049585580825806, 0.37439376115798 95, 0.3805387318134308, 0.38715872168540955, 0.3735528290271759, 0.3813273906707 7637,
0.37841248512268066, 0.4056456685066223, 0.3818448781967163],

'val_accuracy': [0.8355000019073486, 0.8462499976158142, 0.8567500114440918, 0.8633333444595337, 0.8607500195503235, 0.8647500276565552, 0.8702499866485596,
0.8715833425521851, 0.8691666722297668, 0.8738333582878113, 0.8743333220481873, 0.8690833449363708, 0.8713333606719971, 0.8762500286102295, 0.8793333172798157,
0.8767499923706055, 0.8804166913032532, 0.8774999976158142, 0.871916651725769, 0.877916693687439, 0.8771666884422302, 0.878083348274231, 0.8791666626930237,
0.8802499771118164, 0.8808333277702332, 0.8762500286102295, 0.8765000104904175, 0.8713333606719971, 0.8774999976158142, 0.8772500157356262, 0.8774166703224182,
0.878250002861023, 0.8778333067893982, 0.8803333044052124, 0.8758333325386047, 0.8756666779518127, 0.8794999718666077, 0.8813333511352539, 0.8806666731834412,
0.8709999918937683, 0.8765833377838135, 0.8798333406448364, 0.8809999823570251, 0.878333330154419, 0.8744166493415833, 0.8804166913032532, 0.8792499899864197,
0.8797500133514404, 0.871999979019165, 0.8801666498184204]}
```

'loss': 0.2080526500940323,
'accuracy': 0.9248958230018616,

'val_loss': 0.3818448781967163,
'val_accuracy': 0.8801666498184204

結果の分析

60000枚

x_train

48000枚

train

①trainを64枚ずつランダムに取り出して、学習、更新(750回/epoch)

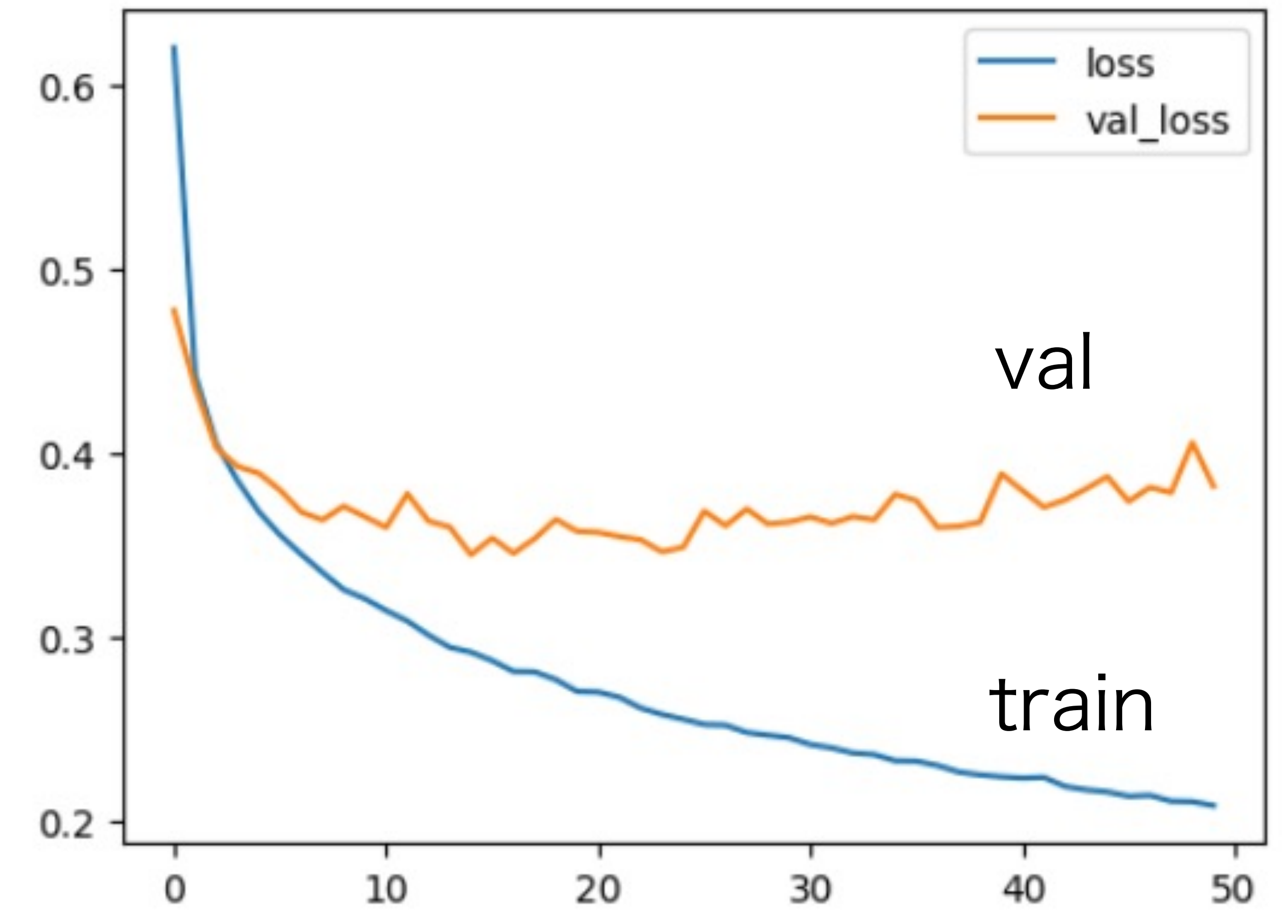
'loss': 0.21
'accuracy': 0.92

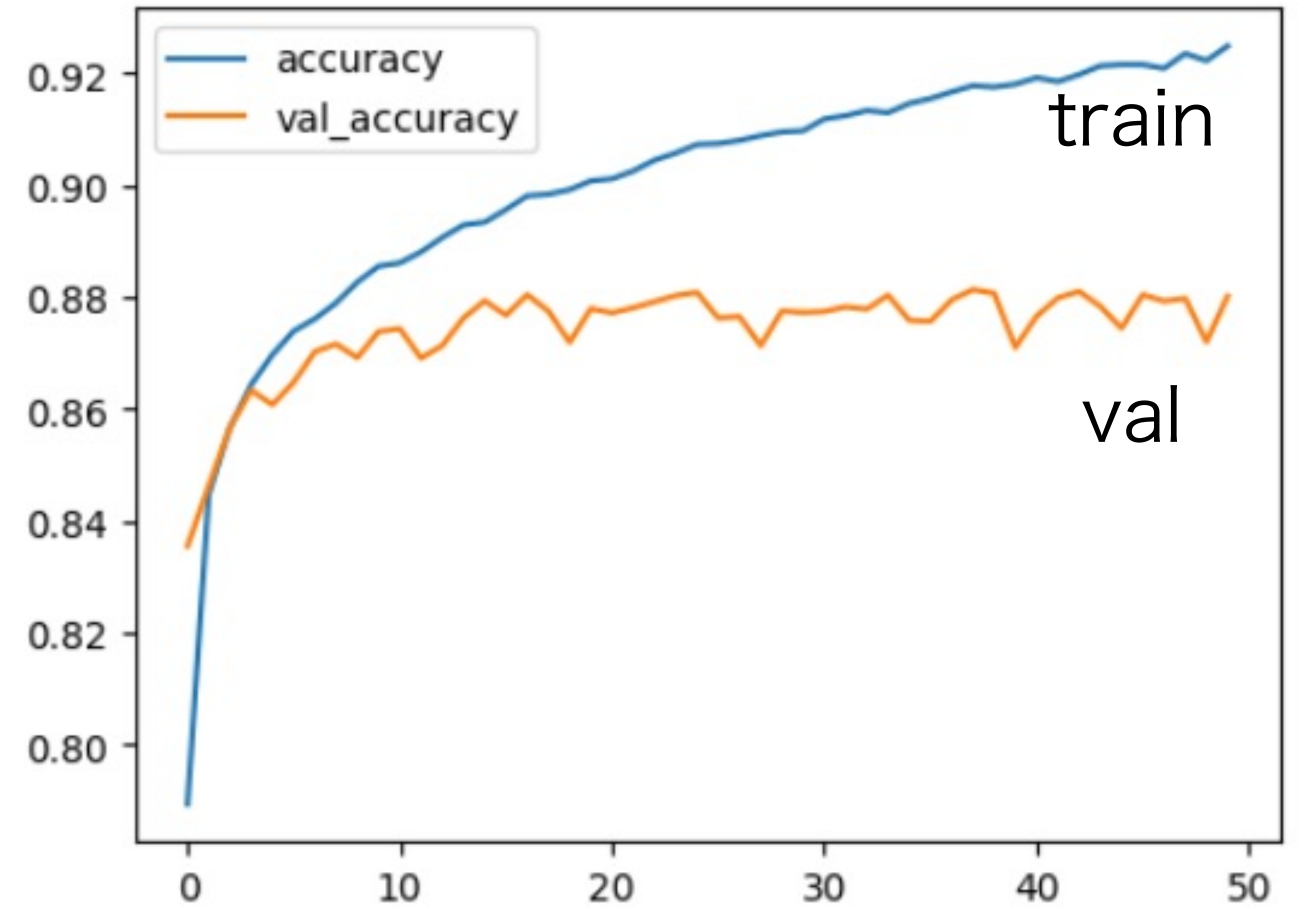
12000枚

val

②1epoch終了後、trainとval両方でlossとaccuracyを算出(50epoch)

'val_loss': 0.38
'val_accuracy': 0.88





10000枚

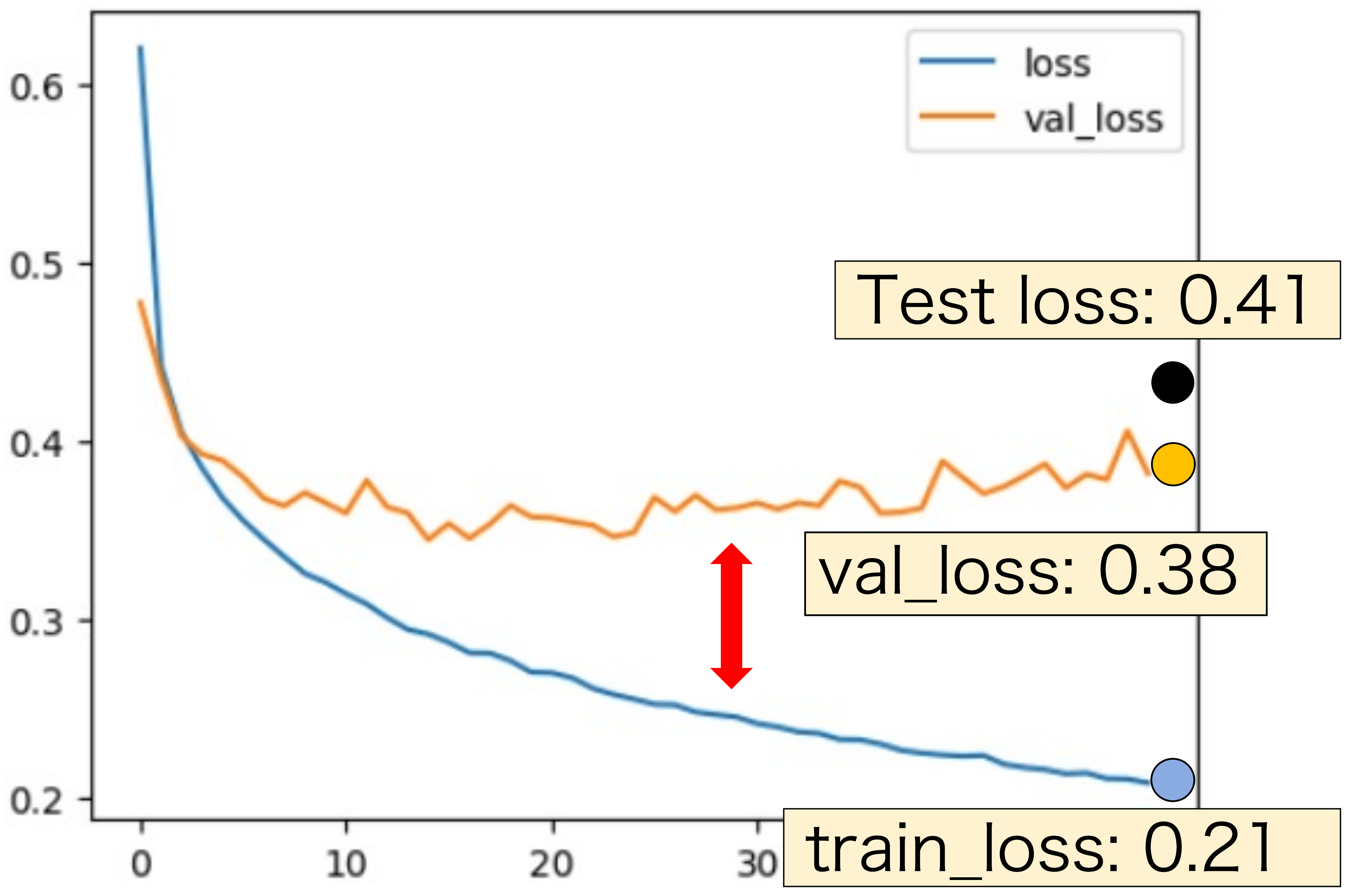
x_test

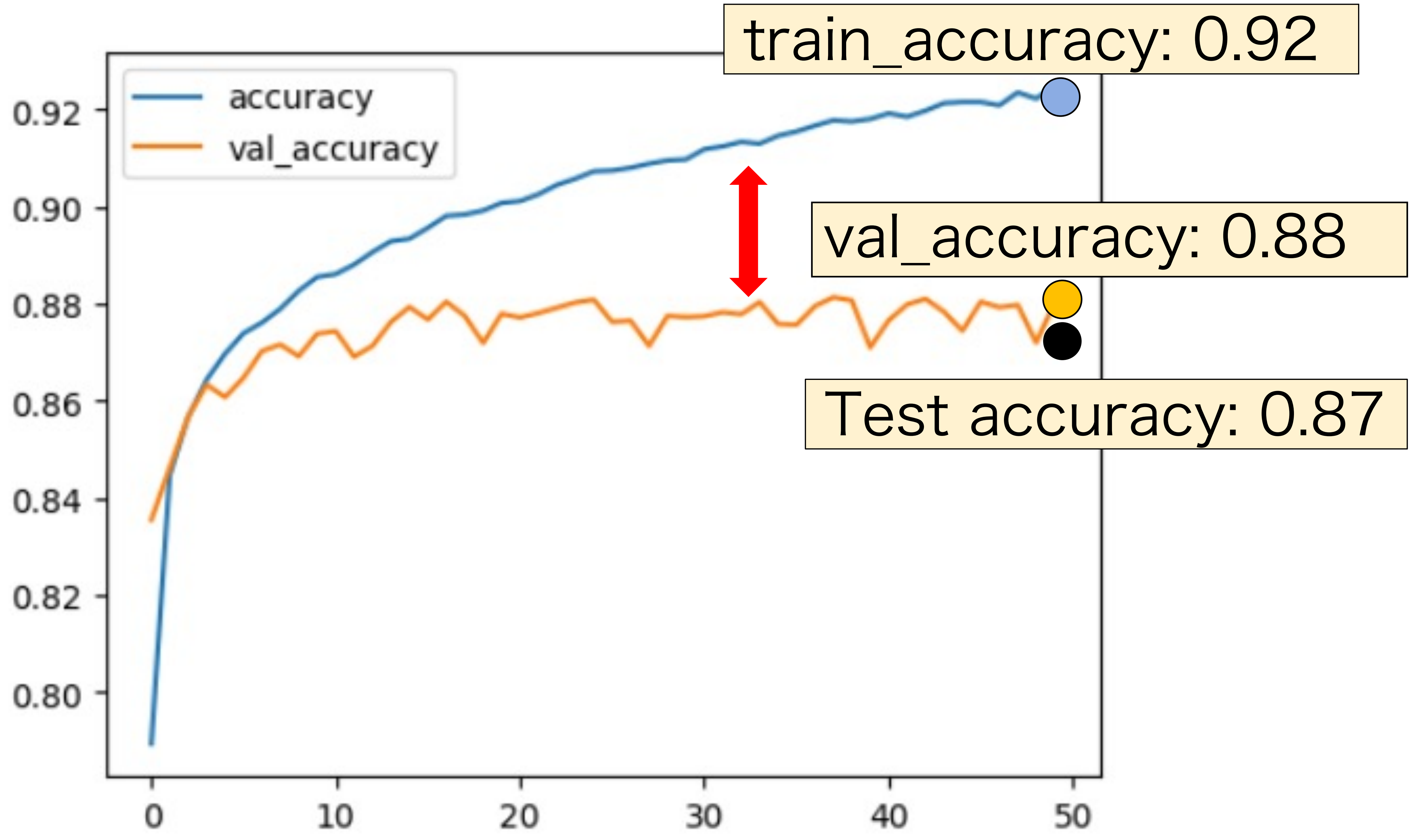
③最後にtestで最終評価
(lossとaccuracy)

Test loss: 0.41
Test accuracy: 0.87

Test loss: 0.40565115213394165
Test accuracy: 0.8694000244140625

trainはlossも小さくaccuracyも高いのに、valは精度が悪い (testも悪い)



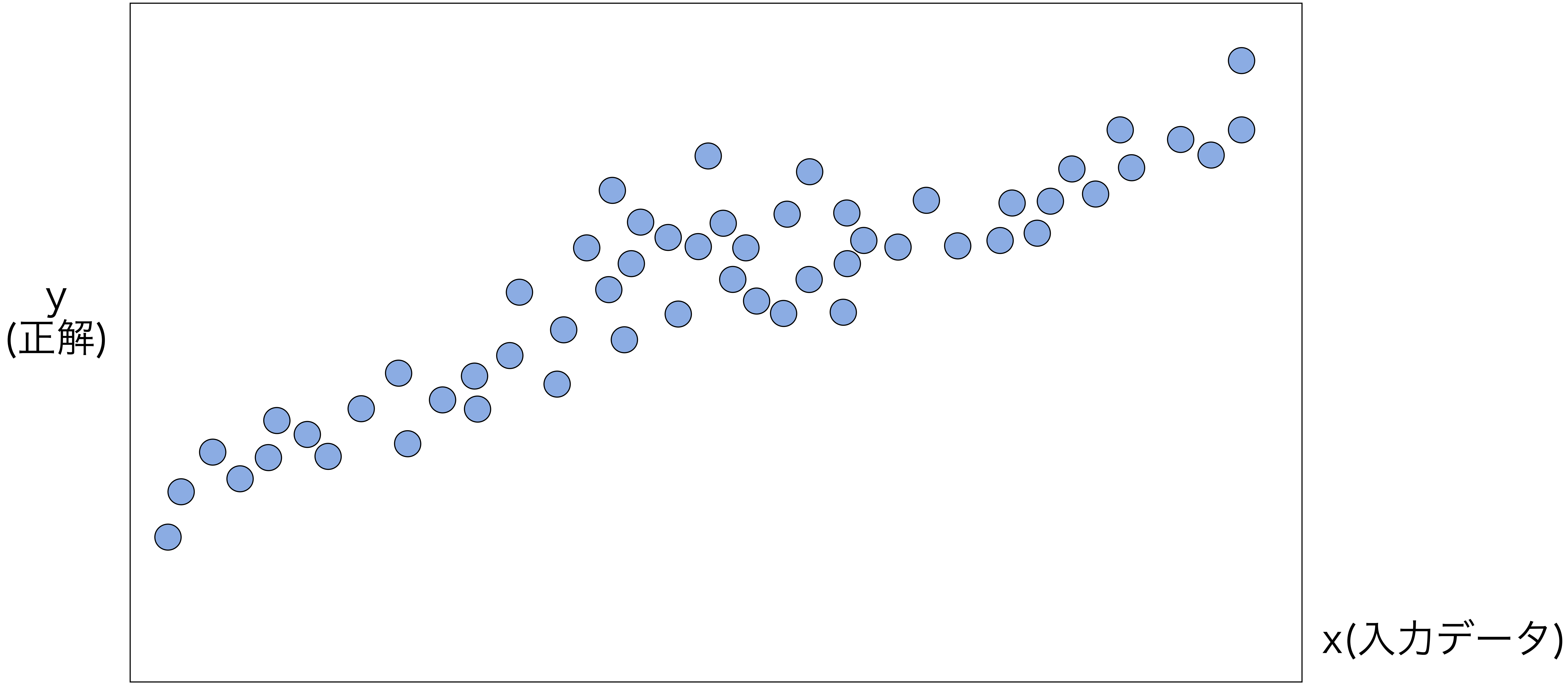


→ trainはうまく学習出来ているのに、valは出来ていない
→ trainのデータに合わせて学習し過ぎ

この状態を過学習と言う

学習のイメージ

学習はデータから当てはまりの良い関数(係数)を導く作業です



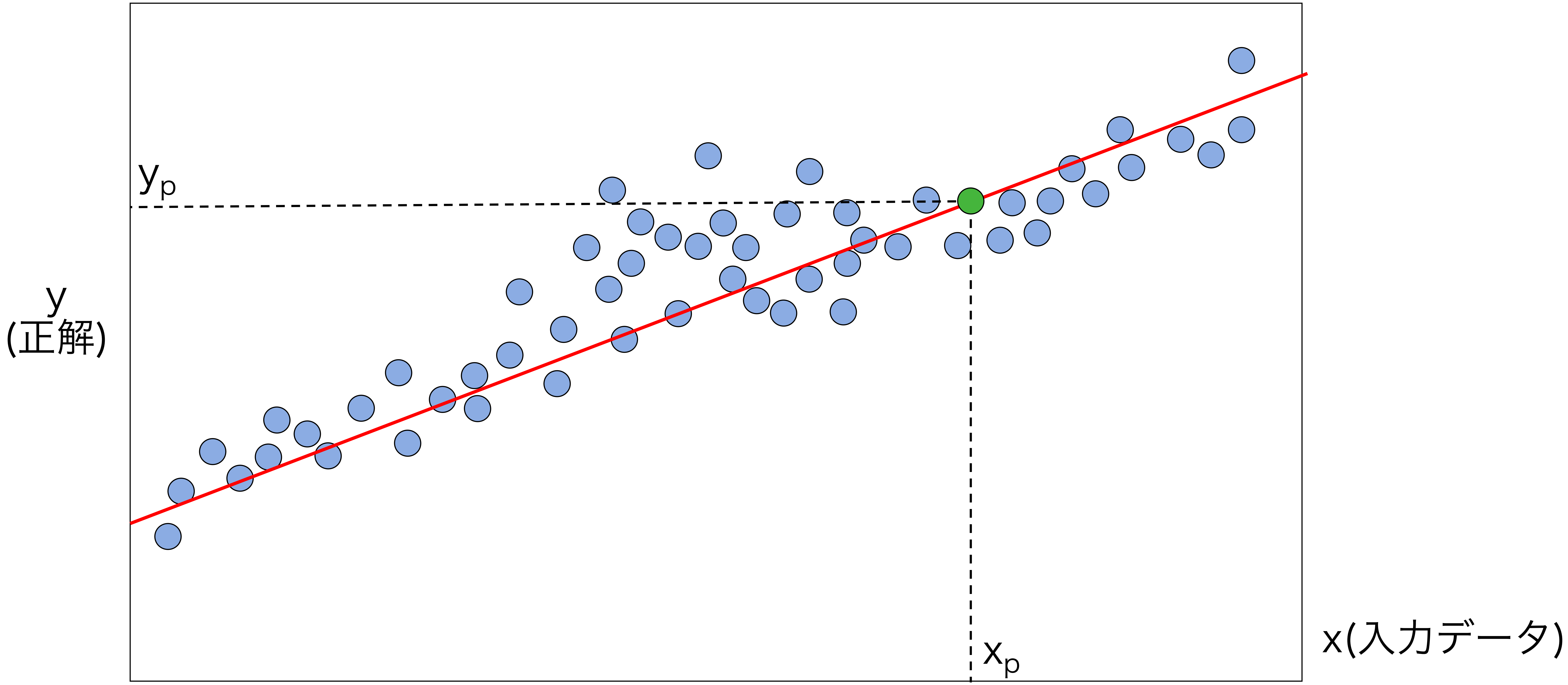
一番シンプルな線形回帰 $f_1(x)$ を想定します(xが増えるとyが一定の割合で増える)

$y = f_1(x) = a_1\ x + a_2$　となり、このa_1とa_2を探します

深層学習のwがa_1、bがa_2に相当します

学習のイメージ

学習はデータから当てはまりの良い関数(係数)を導く作業です



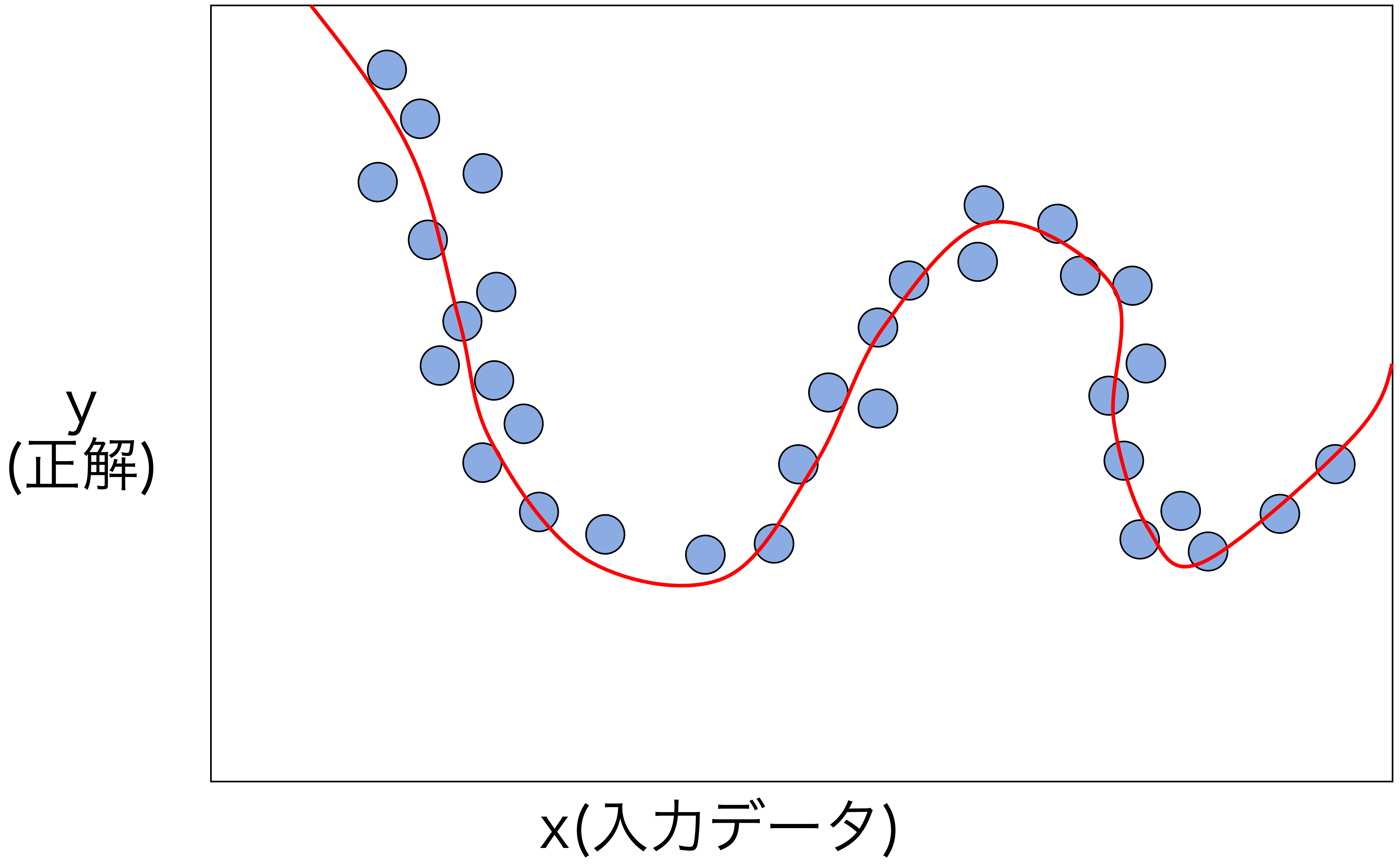
a1とa2が求まると、図のような直線が求まるので、

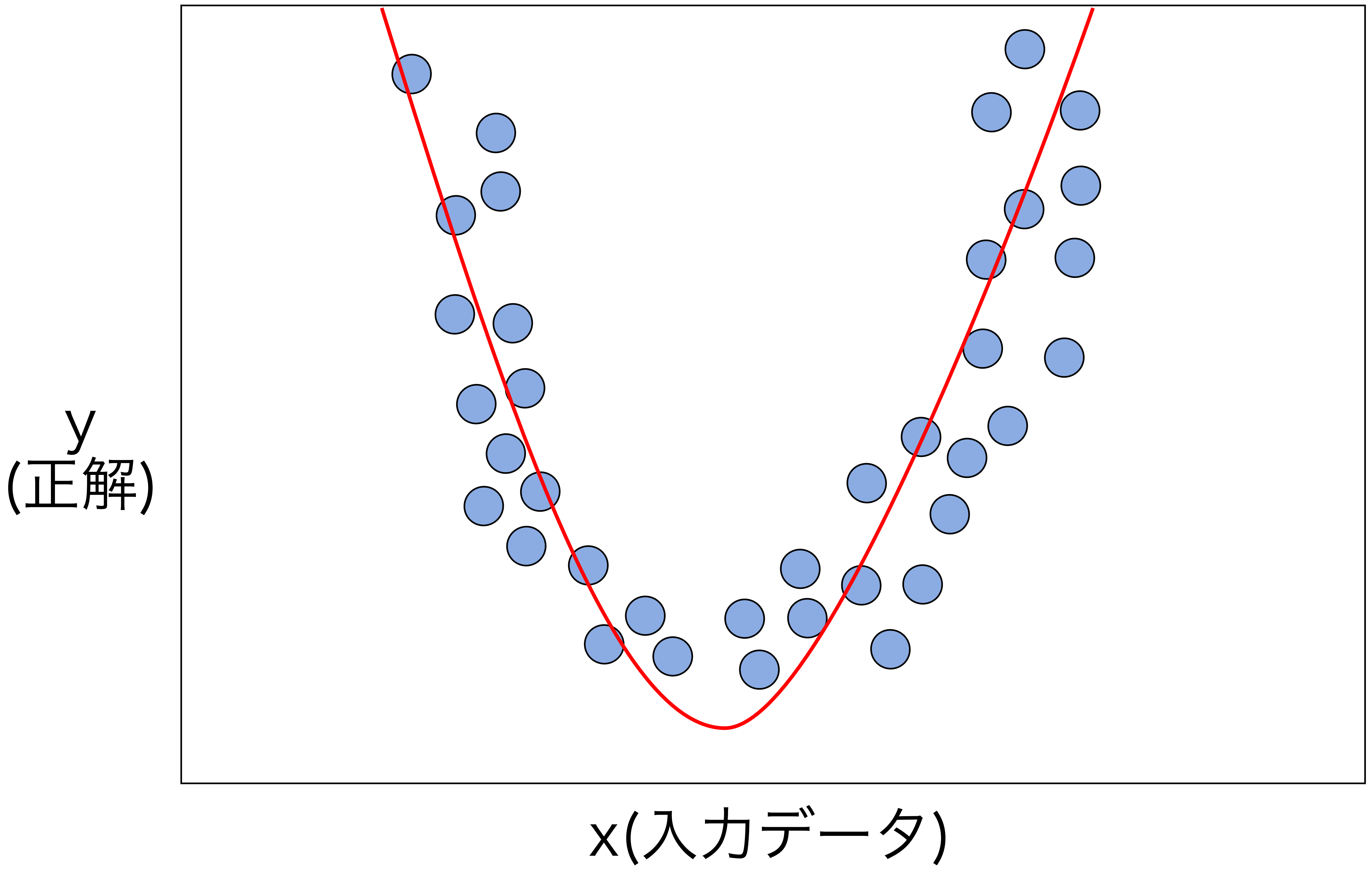
例えば x_pの時の予測結果y_pは、

$y_p = f(x_p) = a_1\ x_p + a_2$ で予測することが出来ます

学習のイメージ

学習はデータから当てはまりの良い関数(係数)を導く作業です

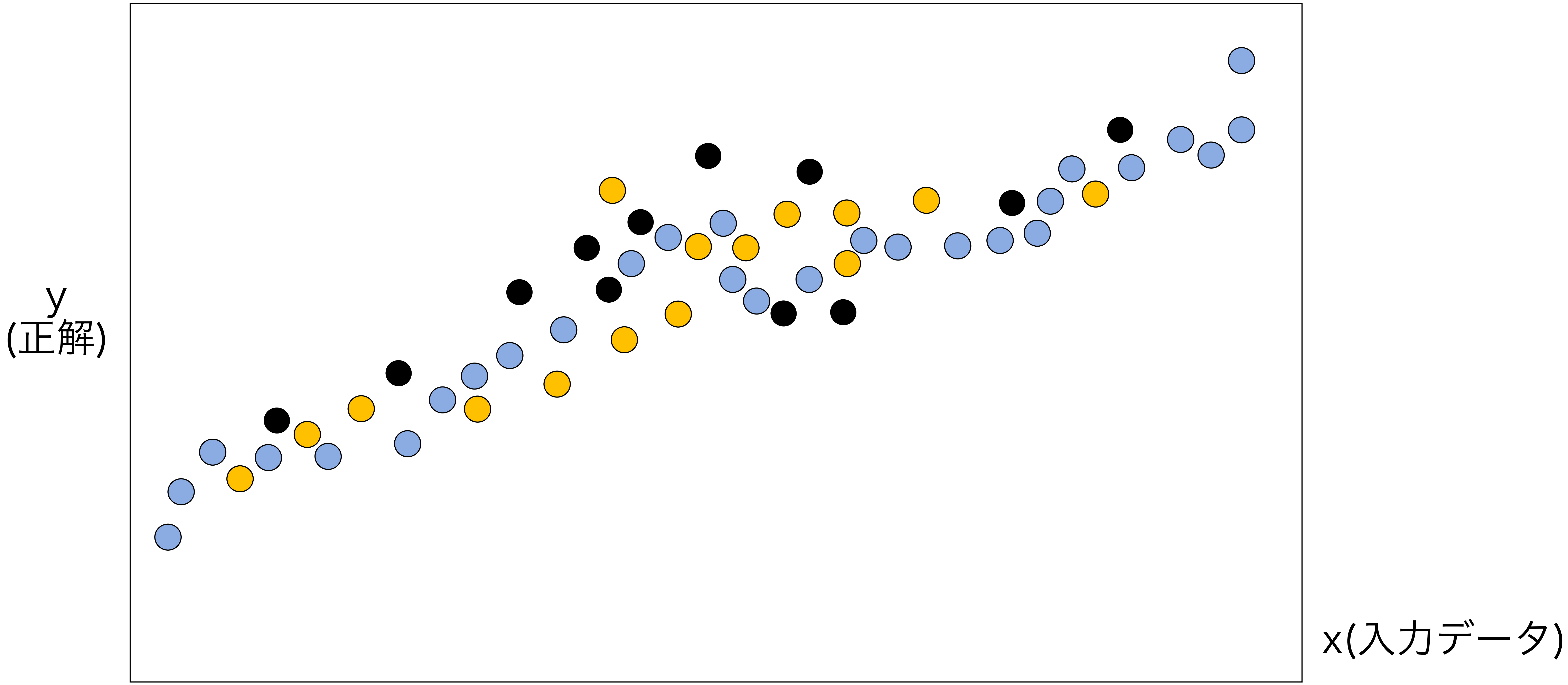




$y = a_1x + a_2$では(どんなa_1、a_2を与えても)直線にしかならない

曲線や3次元空間、4次元以上で表現出来るデータも適切な関数を設定すれば学習出来る

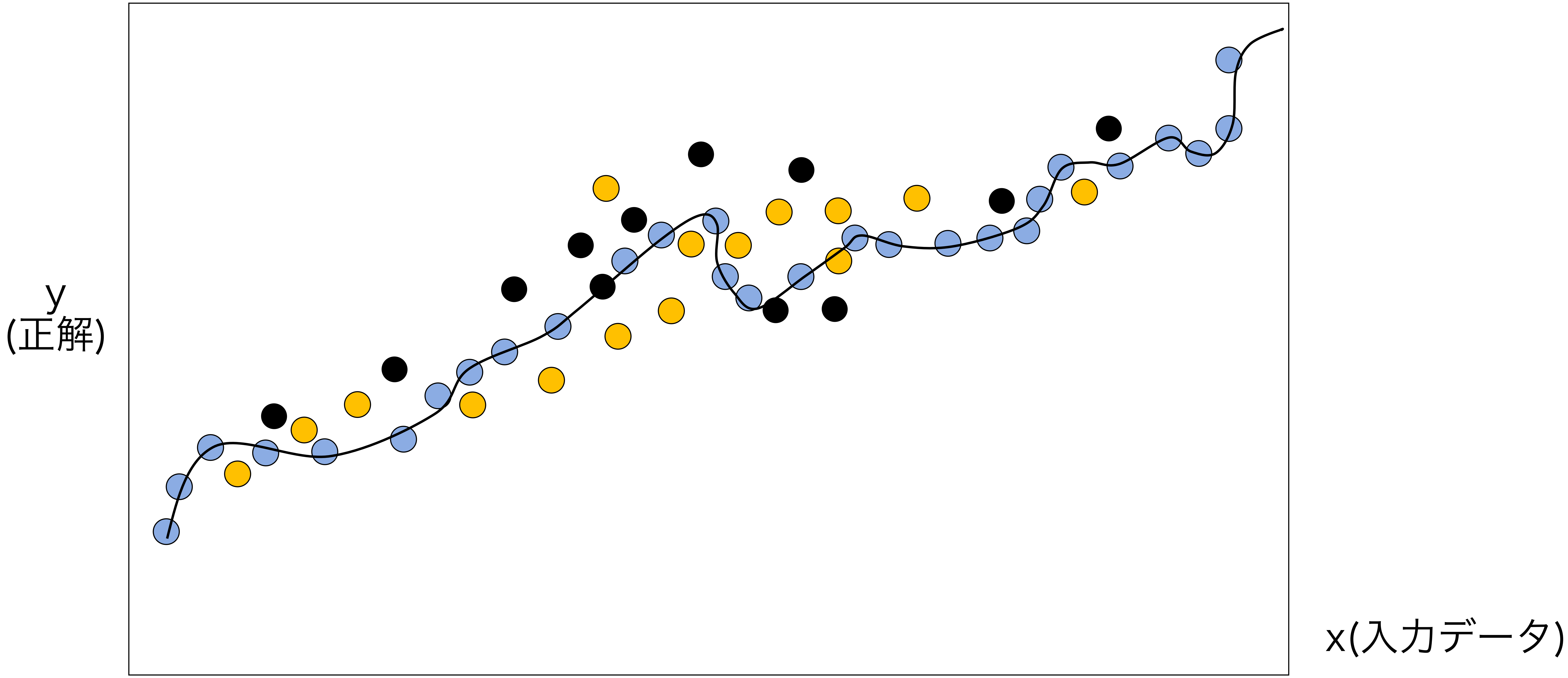
過学習のイメージ



仮に深層学習のデータをまとめて2次元で表したとする
(本当は高次元で作図が難しい)

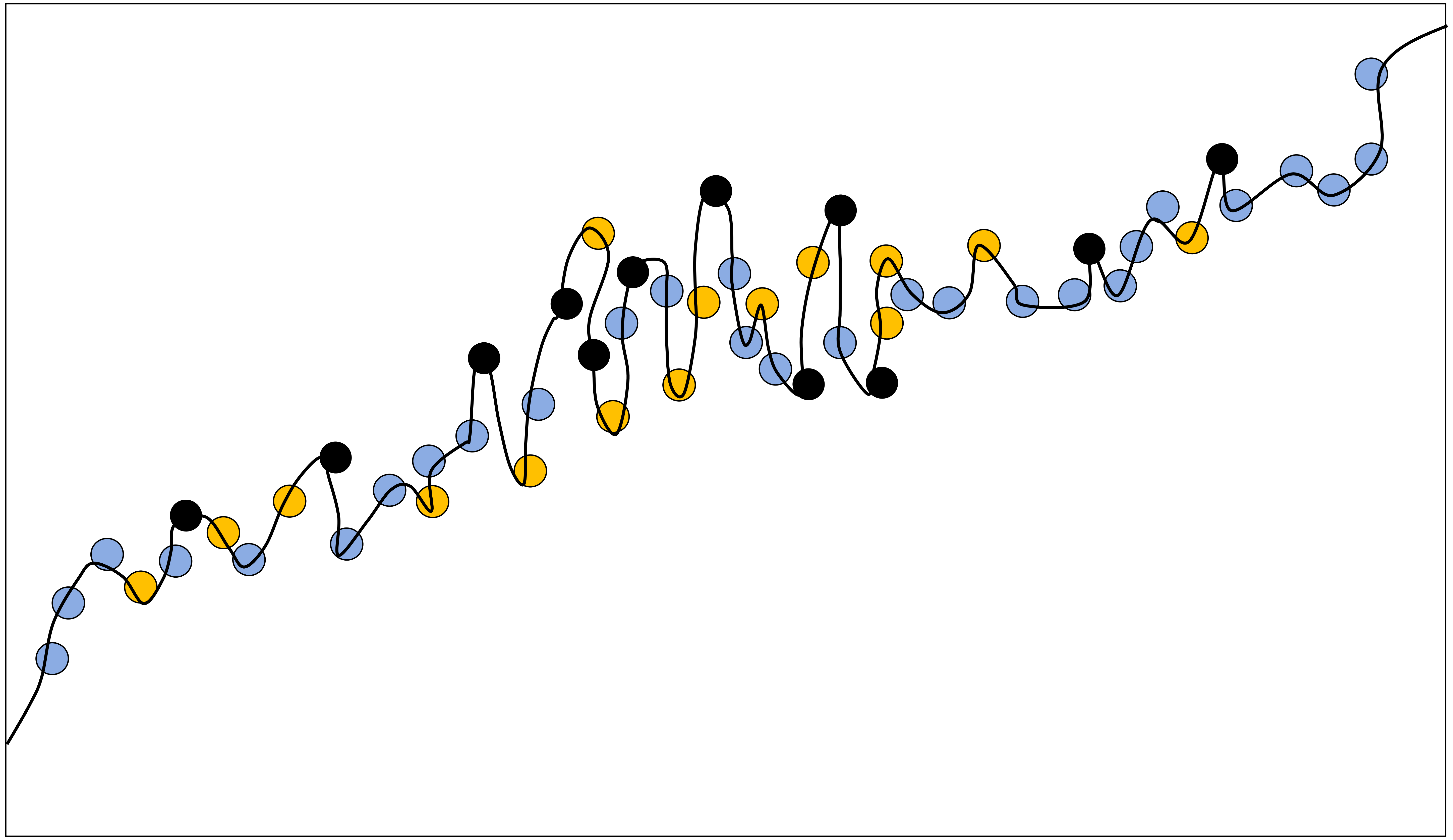
過学習のイメージ

trainはlossも小さくaccuracyも高いのに、valは精度が悪い (testも悪い)



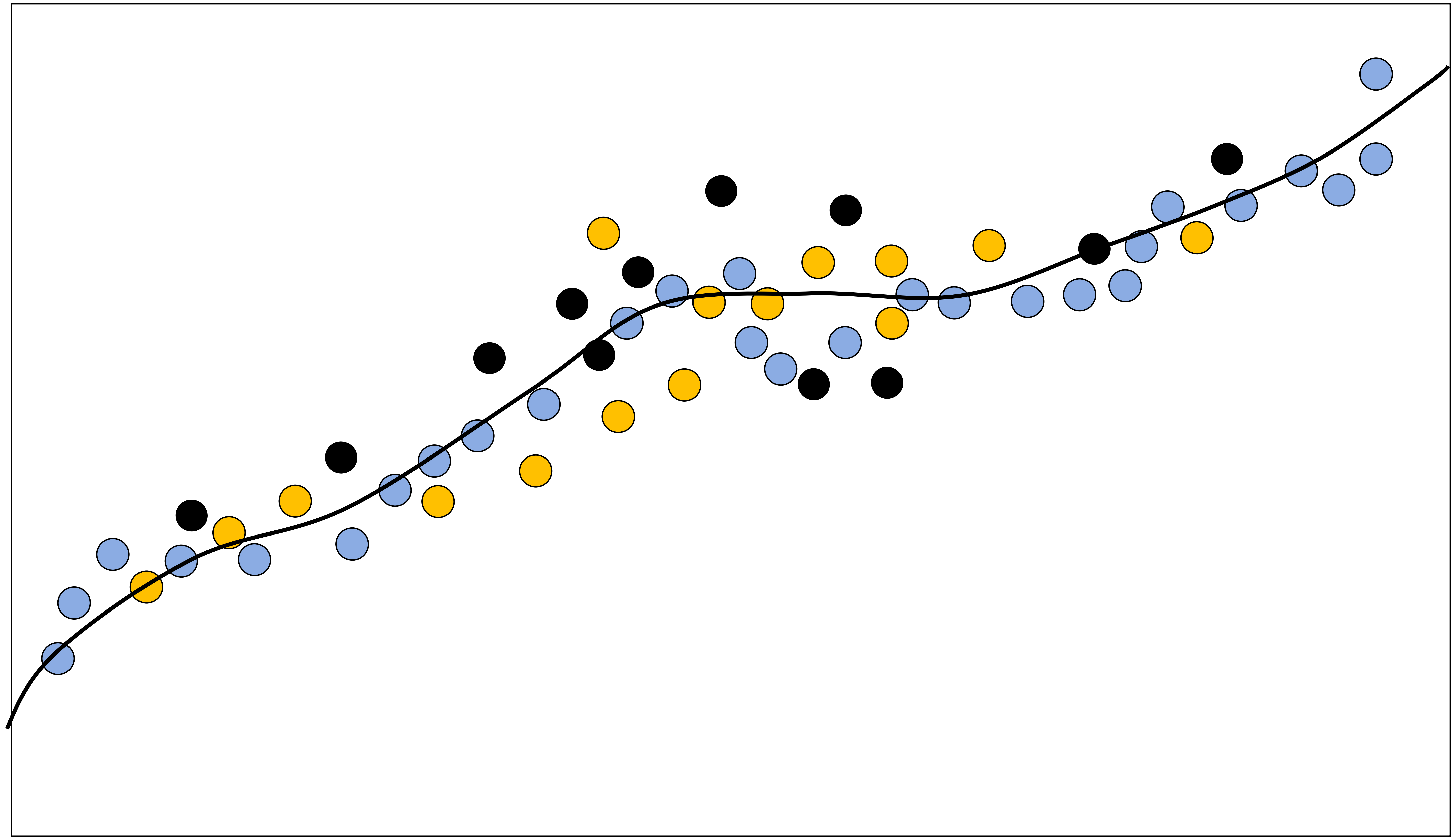
過学習はtrainのデータに合わせて学習し過ぎてしまった状態
この図だとtrainでは精度100%近いが、valやtestでは精度が落ちる

完全に適合しているイメージ



実際はこのようにはなりません

過学習のイメージ



未知のデータでも精度が高くなるようなパラメータを探す作業

モデルの改変や学習用データに偏りがないかなど

ニューロンの数を増やす

```
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()
```

```
_________________________________________________________________
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
_________________________________________________________________
```

```
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
_________________________________________________________________
```

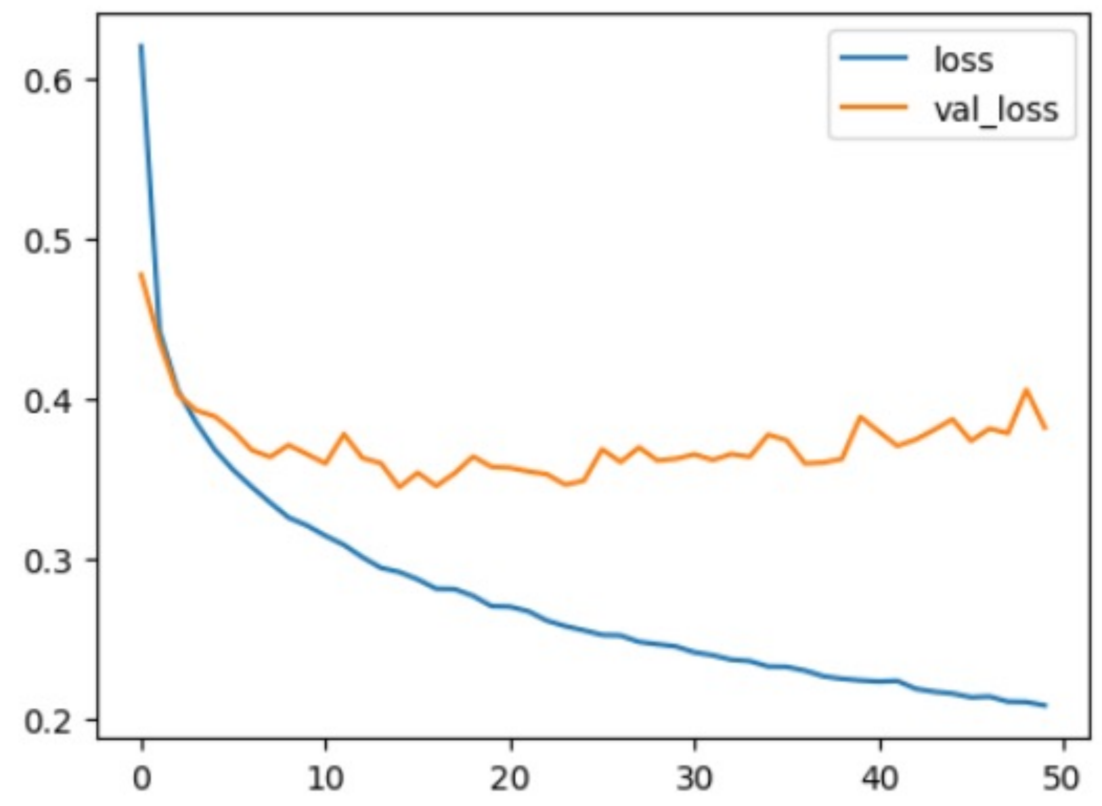
ニューロンの数を増やす

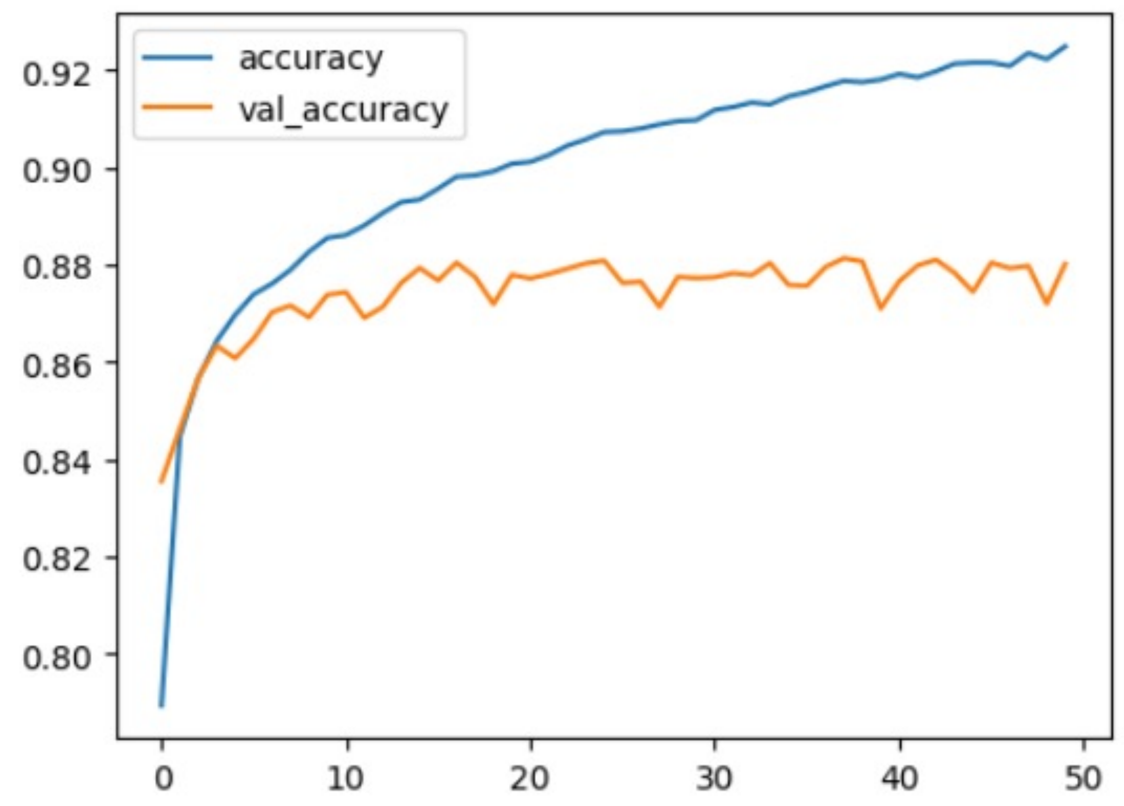
```
model = Sequential()
model.add(Dense(32,input_shape=(784,),activation='relu'))
model.add(Dense(10,activation='softmax'))
model.compile(loss='categorical_crossentropy’,
                    optimizer='Adam',metrics=['accuracy'])
model.summary()
```

```
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()
```

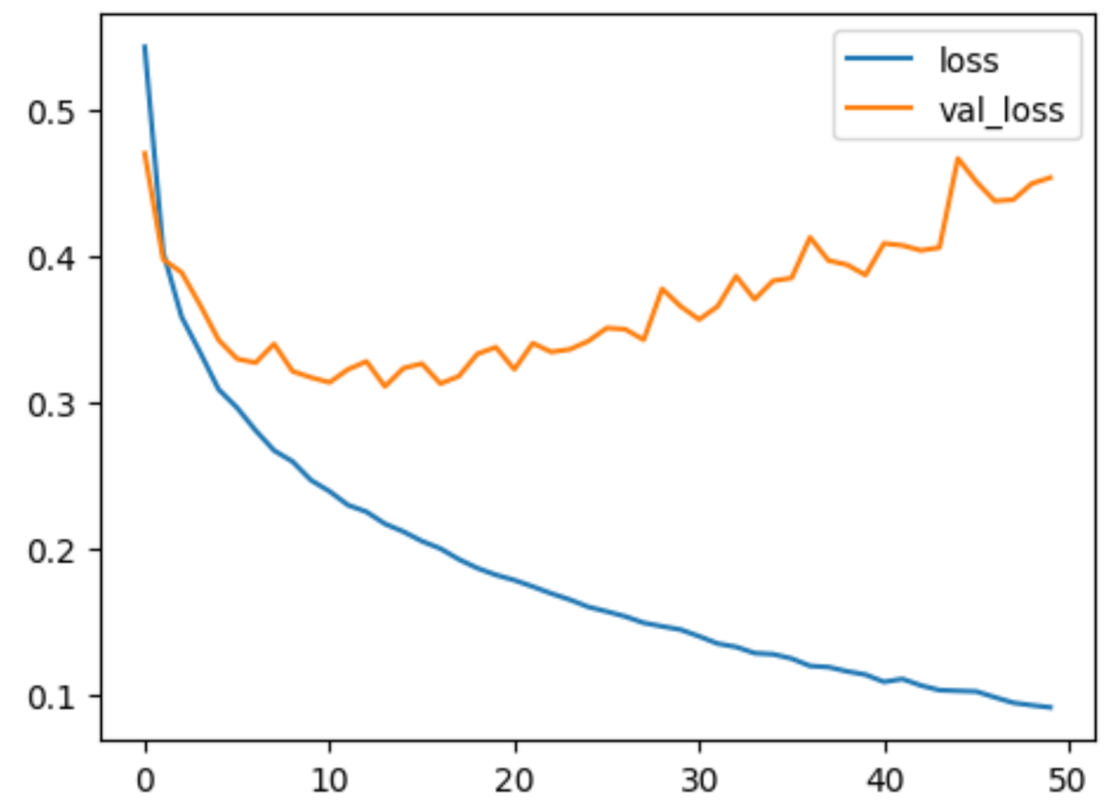
```
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 32)                25120
_________________________________________________________________
dense_13 (Dense)             (None, 10) 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

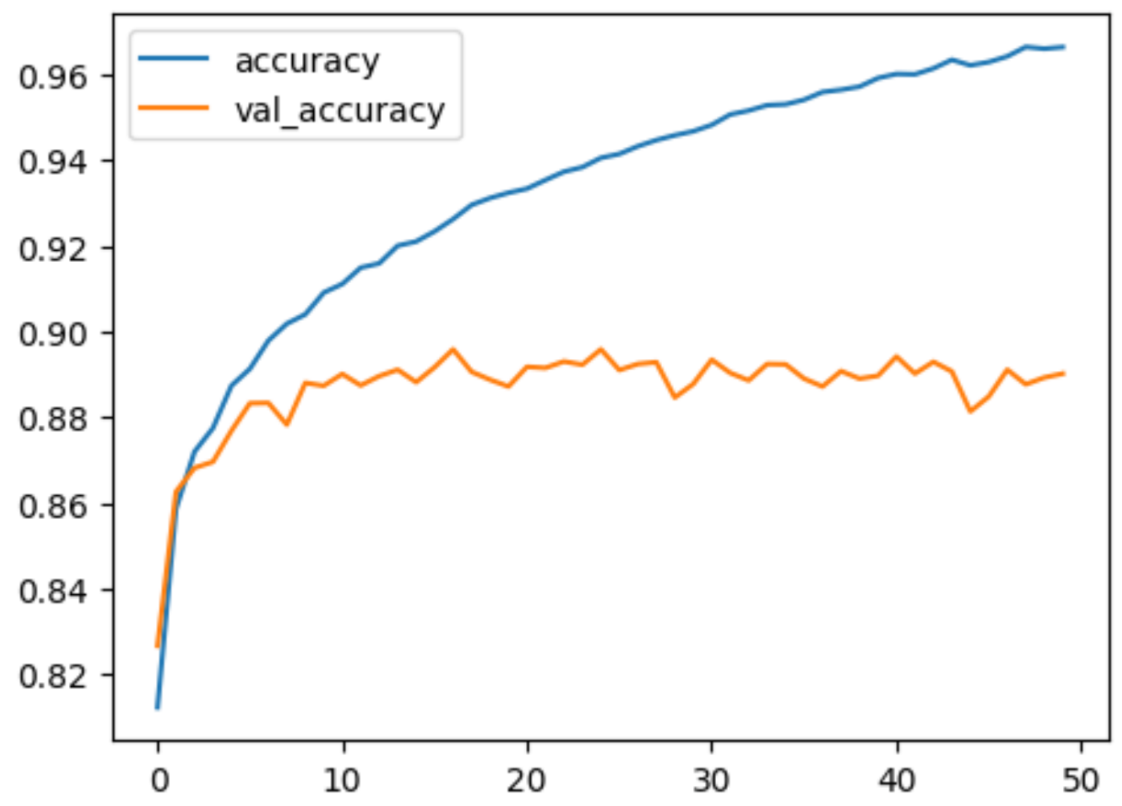
```
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_9 (Dense)              (None, 10)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





Test loss: 0.40565115213394165
Test accuracy: 0.8694000244140625





Test loss: 0.4939178228378296
Test accuracy: 0.885200023651123

層を追加してみよう

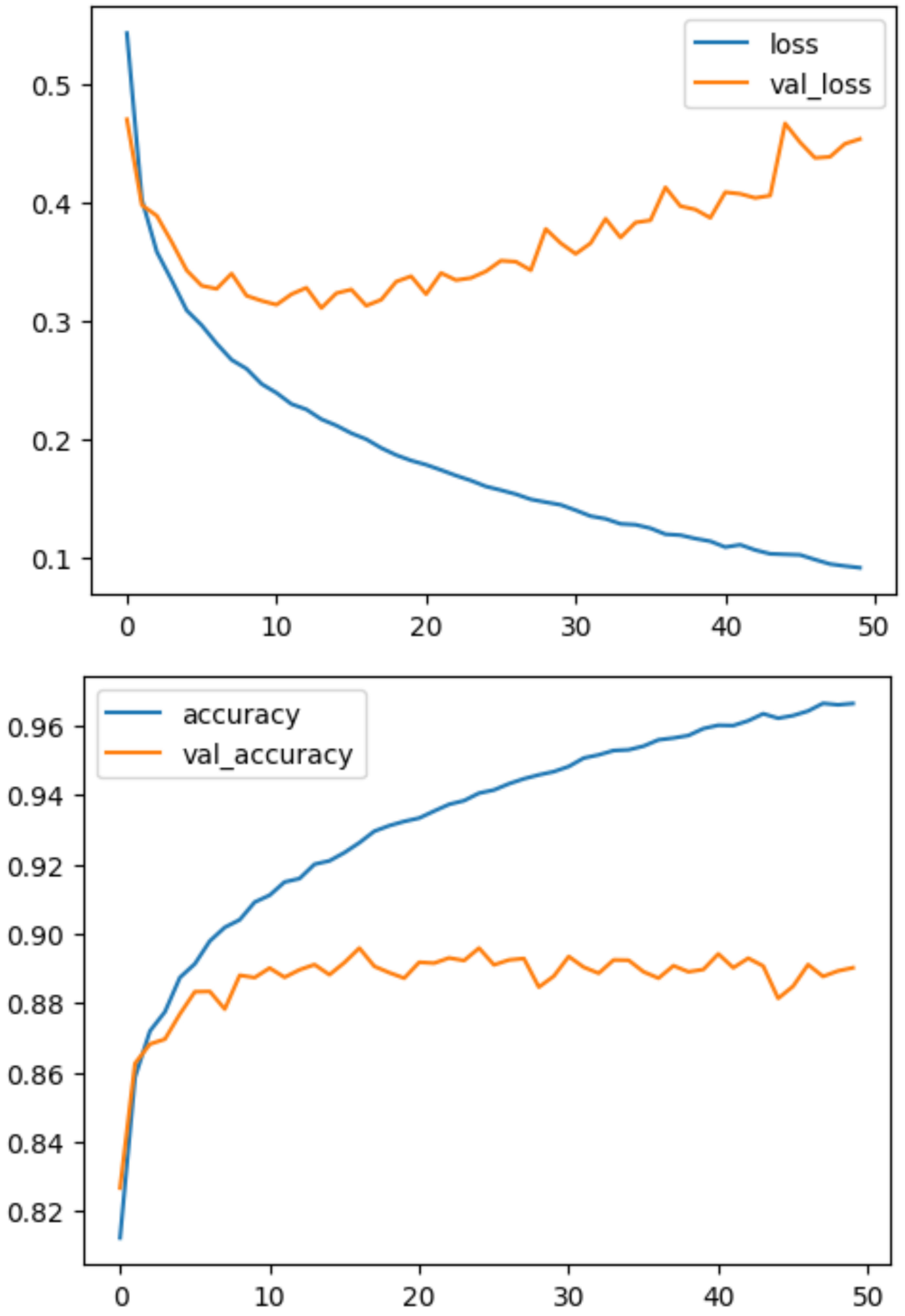
```
model = Sequential()
model.add(Dense(128,input_shape=(784,),activation='relu'))
model.add(Dense(10,activation='softmax'))
model.compile(loss='categorical_crossentropy’,
                    optimizer='Adam',metrics=['accuracy'])
model.summary()
```

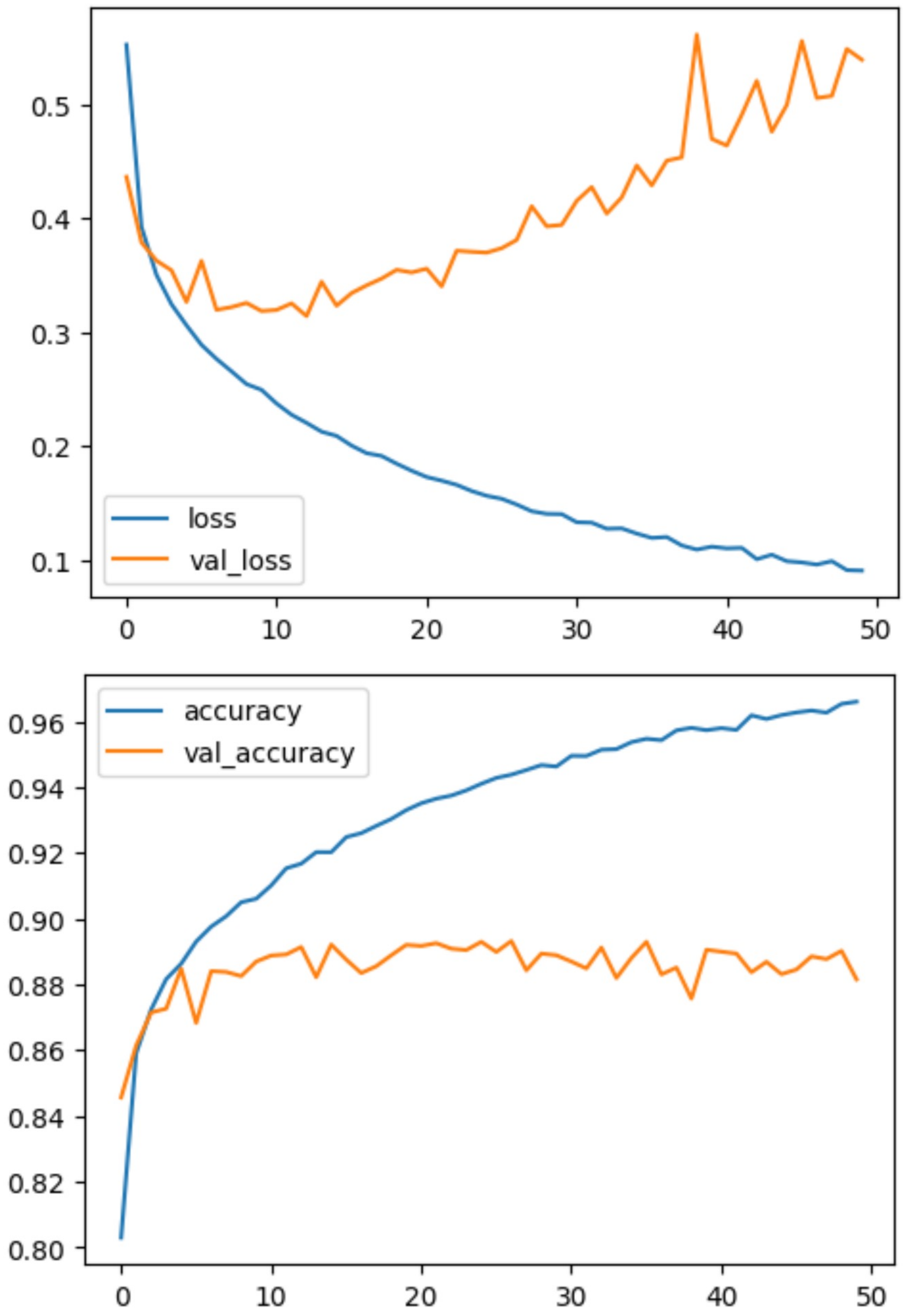
```
model = Sequential()
model.add(Dense(128,input_shape=(784,),activation='relu’))
model.add(Dense(64,activation='relu’))
model.add(Dense(32,activation='relu'))
model.add(Dense(10,activation='softmax'))
model.compile(loss='categorical_crossentropy’,
                    optimizer='Adam',metrics=['accuracy'])
model.summary()
```

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

```
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: 0.4939178228378296
Test accuracy: 0.885200023651123

Test loss: 0.5994181036949158
Test accuracy 0.8773999810218811

なぜ過学習が起きるのか

モデルの複雑さ： モデルが非常に複雑である場合（例えば、パラメータが多すぎる）、そのモデルは訓練データのノイズまで学習してしまう。モデルがデータの真のパターンよりも、データに含まれるランダムな誤差や無関係な特徴を学習してしまう。

データの不足： 訓練データが不十分である場合、モデルは利用可能なデータに過剰に適合してしまう。十分なバリエーションのデータがなければ、モデルが一般化するための「良い」パターンを学ぶことができない。

トレーニングの長さ： トレーニングを長く続けすぎると、モデルが訓練データセットの特異性を学習し、新しいデータに対してうまく一般化できなくなる。

データの品質： データに偏りがあったり、誤った情報が含まれていたりすると、モデルが誤ったパターンを学習する原因となる。

Dropoutを加えてみよう

```
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()
```

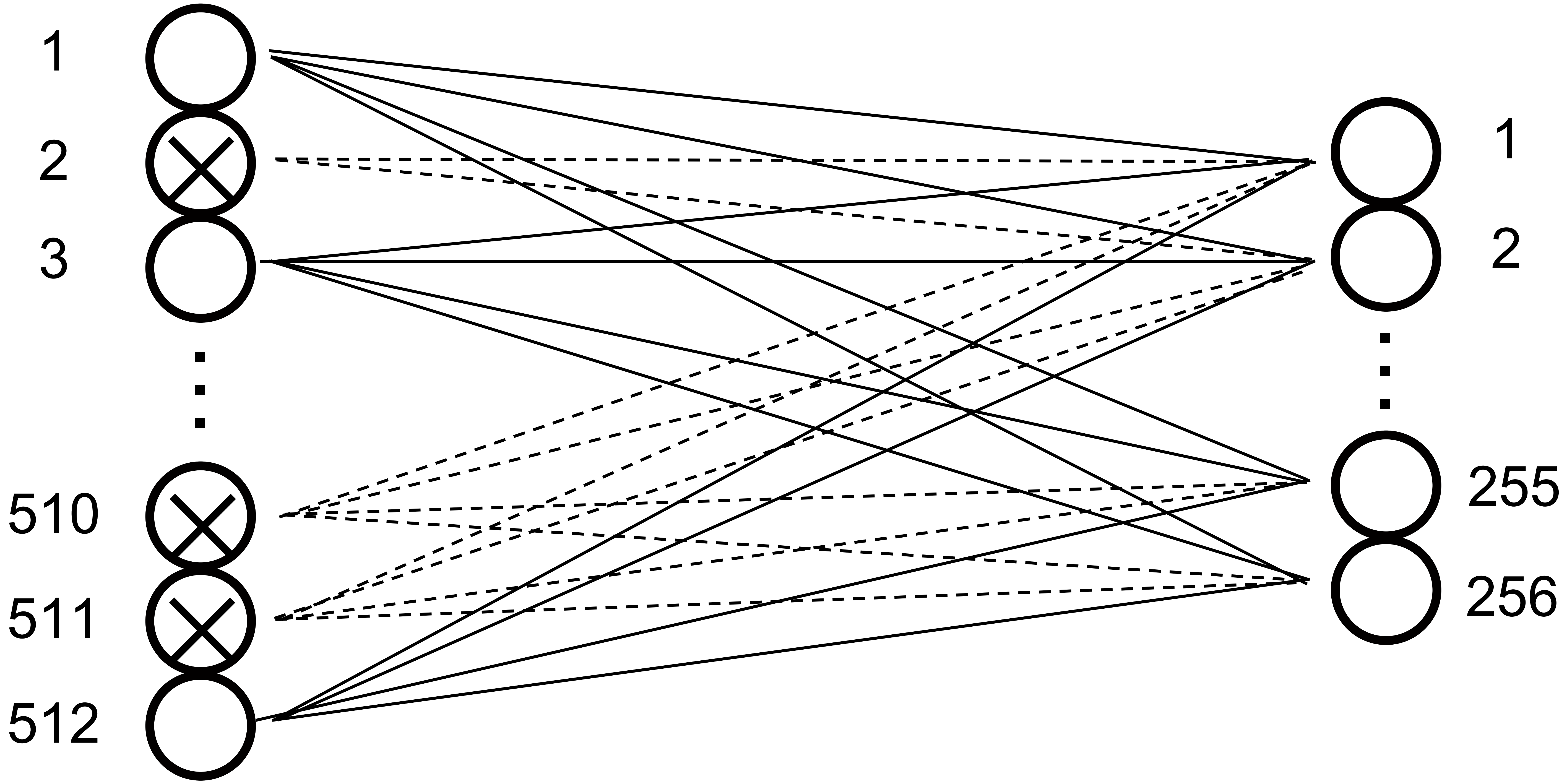
```
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_5 (Dense)              (None, 64)                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 keras.layers import 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()
```

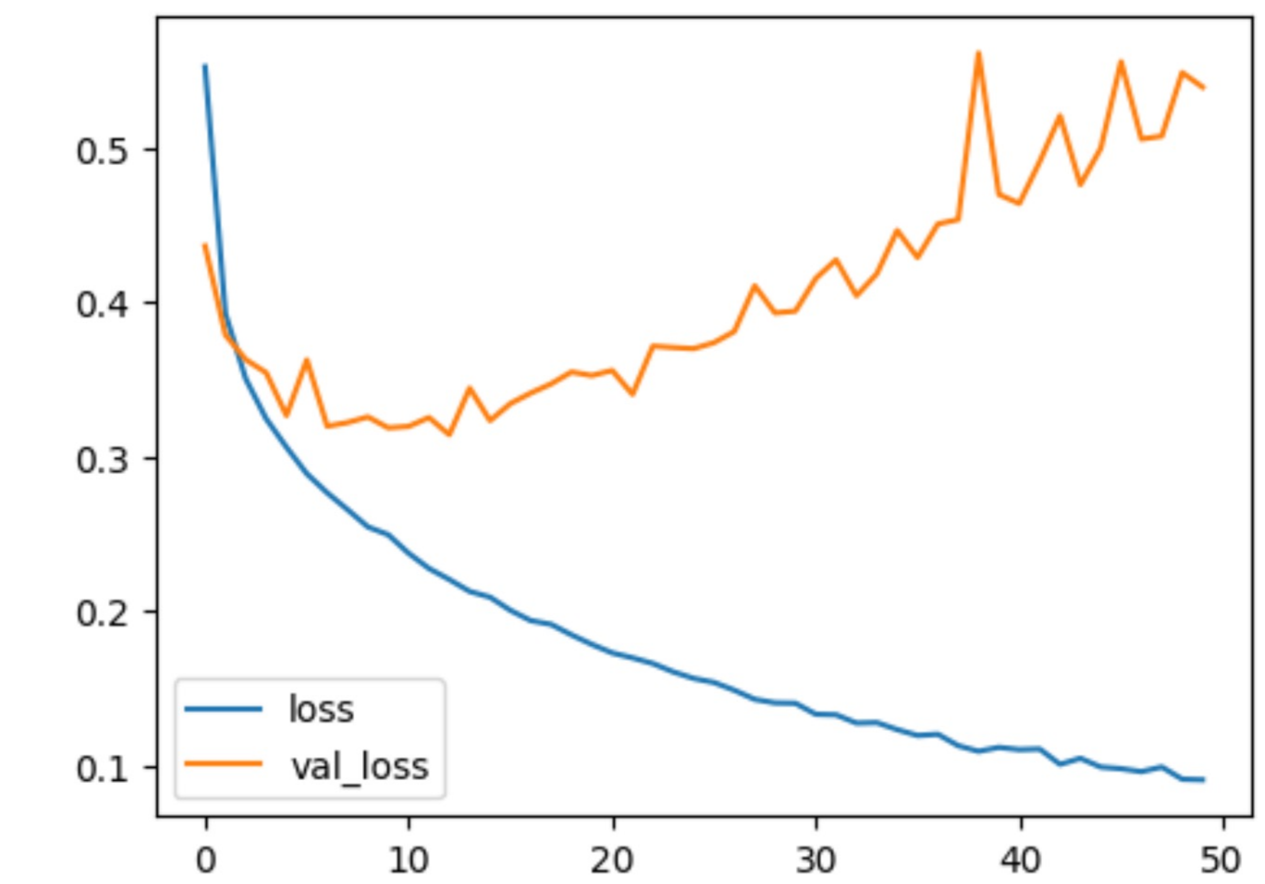
```
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 128)               100480
_________________________________________________________________
dropout (Dropout)            (None, 128)               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
_________________________________________________________________
```

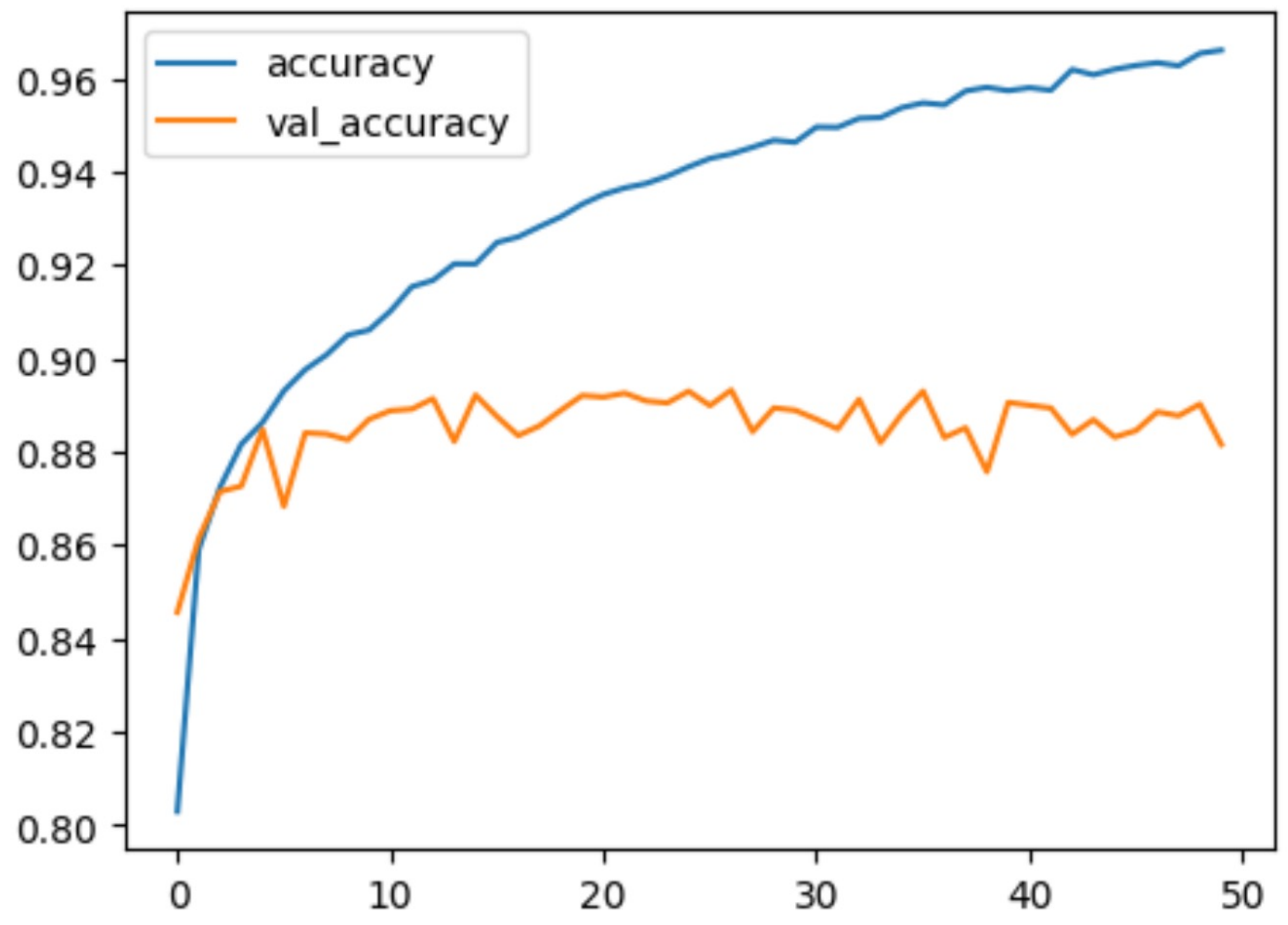
Dropout

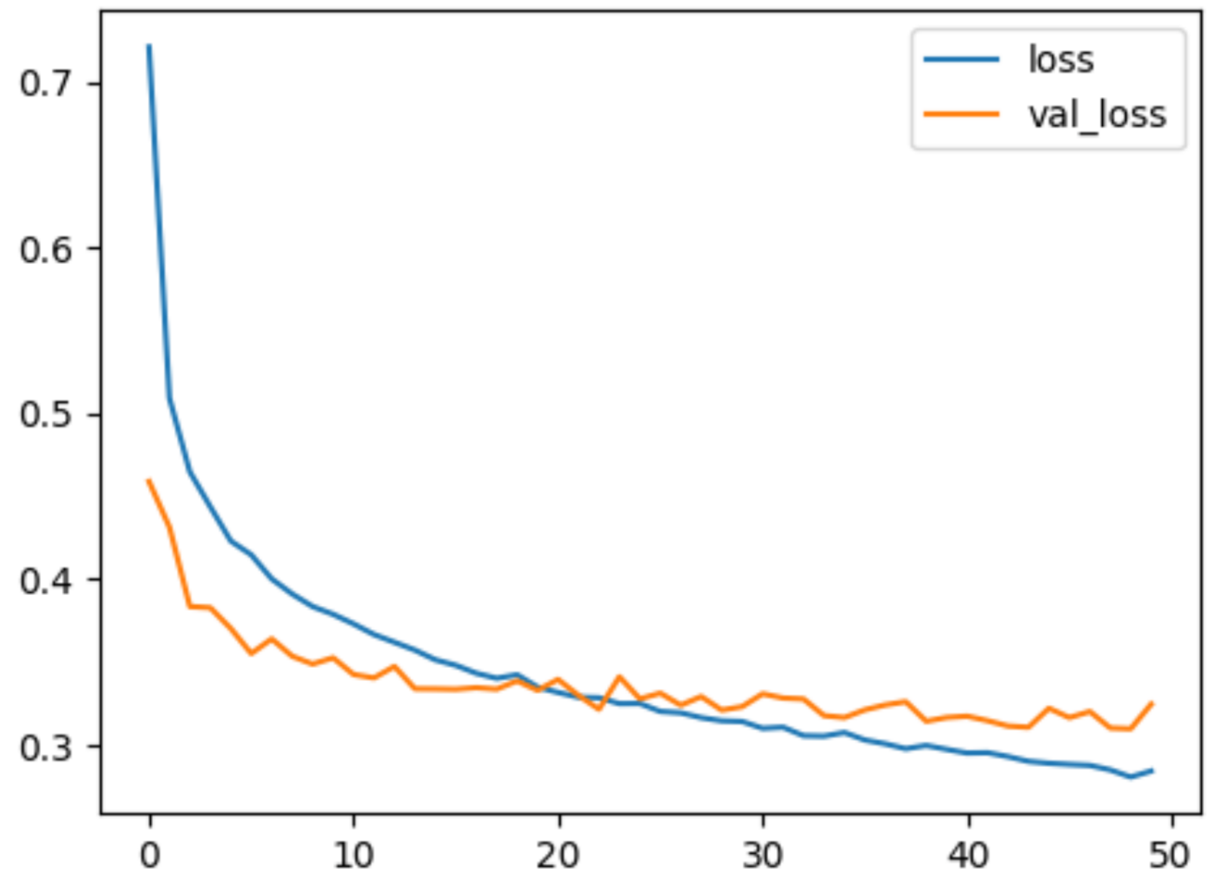
過学習を防ぐための対策の1つで、学習時に設定した確率で出力を０にする手法。
推論時には何も行わず、学習時にのみ行われる。これにより、特定のニューロン
の評価だけに依存し過ぎることを避けて、より頑健な(=データの本質的な構造を捉えた)
特徴を学習することを促す。

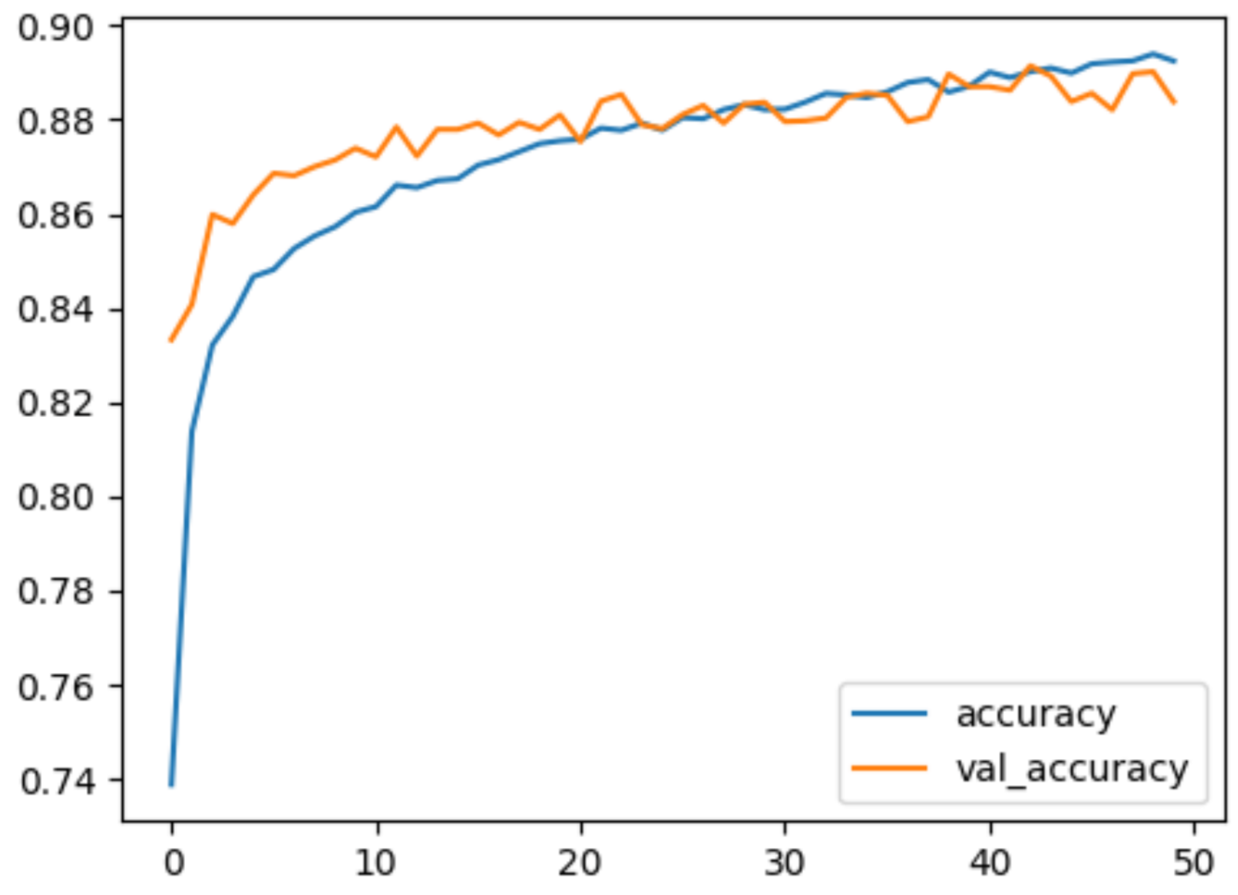


Dropoutを加えると過学習を抑制できる









Test loss: 0.5994181036949158
Test accuracy 0.8773999810218811

Test loss: 0.3330557644367218
Test accuracy: 0.8855000138282776

実際は過学習を抑えつつ精度をどれだけ上げられるかを検討する

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。

課題

・WebClassにある”kadai5.ipynb”をやってみましょう
・実行したら”学籍番号_名前_5.ipynb”という名前で保存して提出して下さい。

締め切りは2週間後の11/23の23:59です。
締め切りを過ぎた課題は受け取らないので注意して下さい