

学習と検証、学習曲線（小画像、2層 の単純なニューラルネットワーク）

（人工知能の実行）

URL: <https://www.kkaneko.jp/ai/ni/index.html>

金子邦彦



トピックス



- 分類を行うニューラルネットワーク
- ニューラルネットワークの作成
- ニューラルネットワークの学習
- 学習曲線
- 学習不足
- 過学習

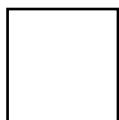
プログラムは、次で公開

https://colab.research.google.com/drive/18Nf9FPFhOvx8_V30z8PdBD2kcyDap8b7?usp=sharing

- **プログラムの再実行, プログラムの変更**には, **Google アカウント**が必要.
- プログラムを変更した場合でも, 特別な操作をしない限り, 他の人には公開されない

画像と画素

画素は、**白と黒の2種類**しかないとする

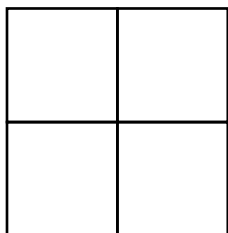


白は 0

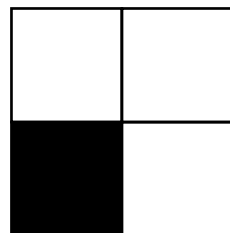


黒は 1 とする

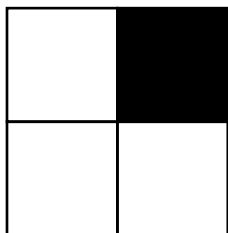
画像のサイズが 2×2 のとき



[0, 0, 0, 0]



[0, 0, 1, 0]



[0, 1, 0, 0]

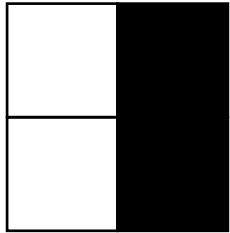


[1, 1, 1, 1]

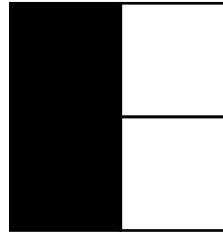
分類の例



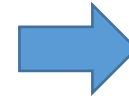
① 縦



[0, 1, 0, 1]



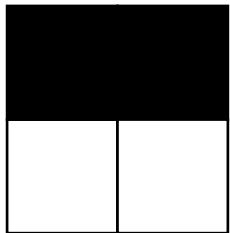
[1, 0, 1, 0]



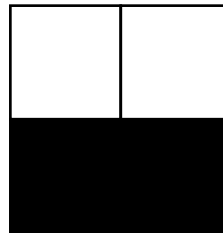
分類結果

1

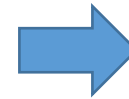
② 横



[1, 1, 0, 0]



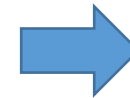
[0, 0, 1, 1]



2

③ それ以外

その他

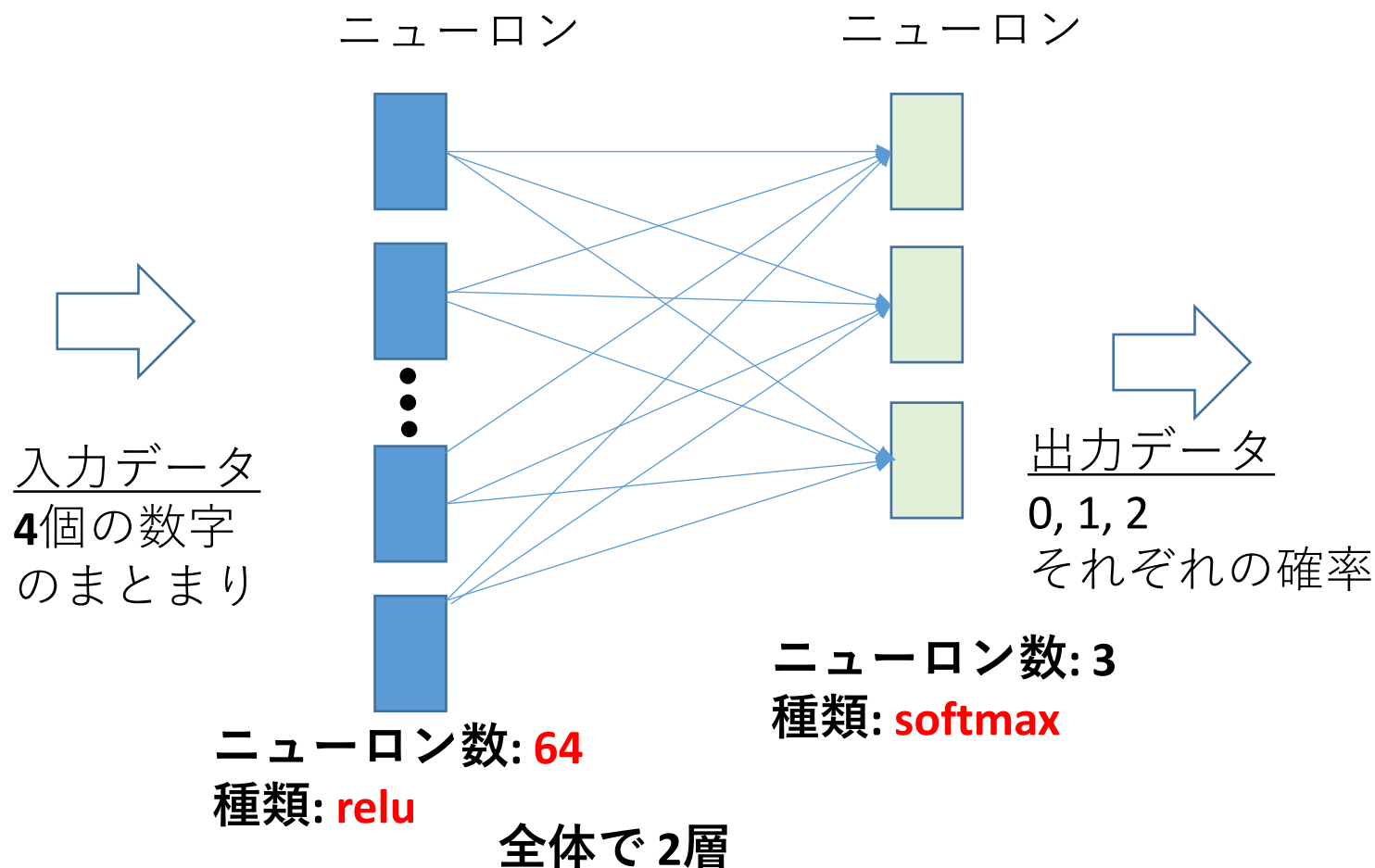


0

作成するニューラルネットワーク



- 1層目：ニューロン数 **64**, 種類は **relu**
- 2層目：ニューロン数 **3**, 種類は **softmax**



ニューラルネットワーク作成のプログラム例



プログラムを使用し,
ニューラルネットワークを作成

```
import tensorflow as tf
```

入力データは **4** 個の数字

1 層目のニューロン数は **64**
種類は **relu**

```
def create_model():
```

```
    return m = tf.keras.models.Sequential([  
        tf.keras.layers.Dense(units=64, input_dim=4, activation='relu'),  
        tf.keras.layers.Dense(units=3, activation='softmax')
```

```
    ])
```

2 層目のニューロン数は **3**
種類は **softmax**

ニューラルネットワーク の作成では、次を設定する

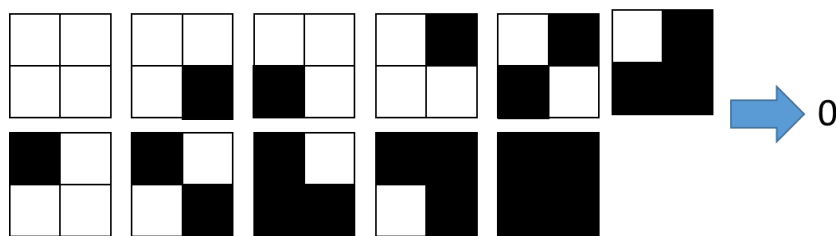
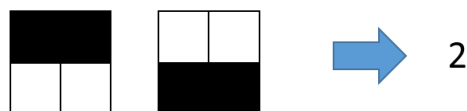
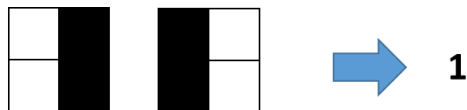
- 入力データでの数値の個数
- **ニューロン** の数 (**層** ごと)
- **ニューロン** の種類 (**層** ごと)

訓練データと検証データ



訓練データ： **学習**に使用

訓練データによる**学習**により、**訓練データ**ではないデータでも分類できる能力（「汎化」という）を獲得



検証データ： **学習の結果を確認**するためのもの。

訓練データとは違うものを使用する。

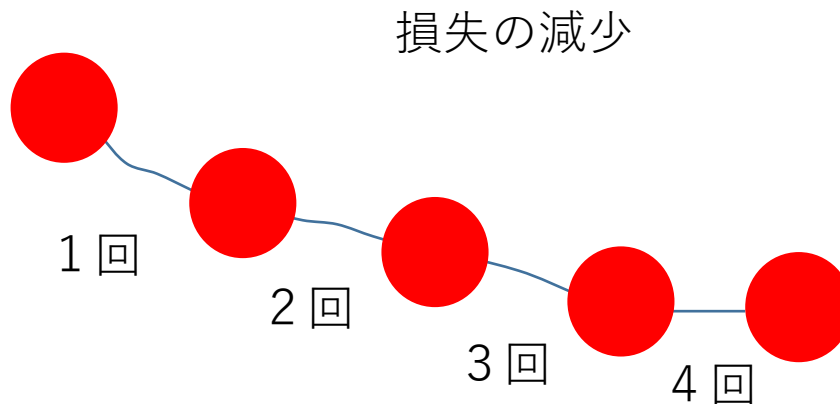


学習の繰り返し

- 同じ**訓練データ**を**繰り返し**使用

訓練データを1回使っただけでは、**学習不足**の場合がある。

繰り返し使用することで、損失をさらに減らす



学習の繰り返しを行うプログラム例



学習の繰り返し回数は **200**

```
EPOCHS=200  
history = m.fit(x=x_train,  
                y=y_train,  
                epochs=EPOCHS,  
                validation_data=(x_test, y_test),  
                callbacks=[tensorboard_callback],  
                verbose=2)
```

訓練データの指定

検証データの指定

学習の繰り返しを行うプログラムの実行結果



同じ訓練データを用いた学習を繰り返し
ながら、検証データで検証

```
super (sub, self, __init__(name, kwargs))
Epoch 1/200
1/1 - 1s - loss: 1.2898 - accuracy: 0.2000 - val_loss: 1.3929 - val_accuracy: 0.0000e+00 - 870ms/epoch - 870ms/step
Epoch 2/200
1/1 - 0s - loss: 1.2638 - accuracy: 0.2000 - val_loss: 1.3274 - val_accuracy: 0.0000e+00 - 51ms/epoch - 51ms/step
Epoch 3/200
1/1 - 0s - loss: 1.2283 - accuracy: 0.2000 - val_loss: 1.2487 - val_accuracy: 0.0000e+00 - 59ms/epoch - 59ms/step
Epoch 4/200
1/1 - 0s - loss: 1.1864 - accuracy: 0.2000 - val_loss: 1.1619 - val_accuracy: 0.0000e+00 - 56ms/epoch - 56ms/step
Epoch 5/200
1/1 - 0s - loss: 1.1409 - accuracy: 0.4000 - val_loss: 1.0714 - val_accuracy: 0.0000e+00 - 74ms/epoch - 74ms/step
Epoch 6/200
1/1 - 0s - loss: 1.0937 - accuracy: 0.4667 - val_loss: 0.9818 - val_accuracy: 0.0000e+00 - 70ms/epoch - 70ms/step
Epoch 7/200
1/1 - 0s - loss: 1.0473 - accuracy: 0.5333 - val_loss: 0.8945 - val_accuracy: 1.0000 - 51ms/epoch - 51ms/step
```

学習の繰り返しごとの

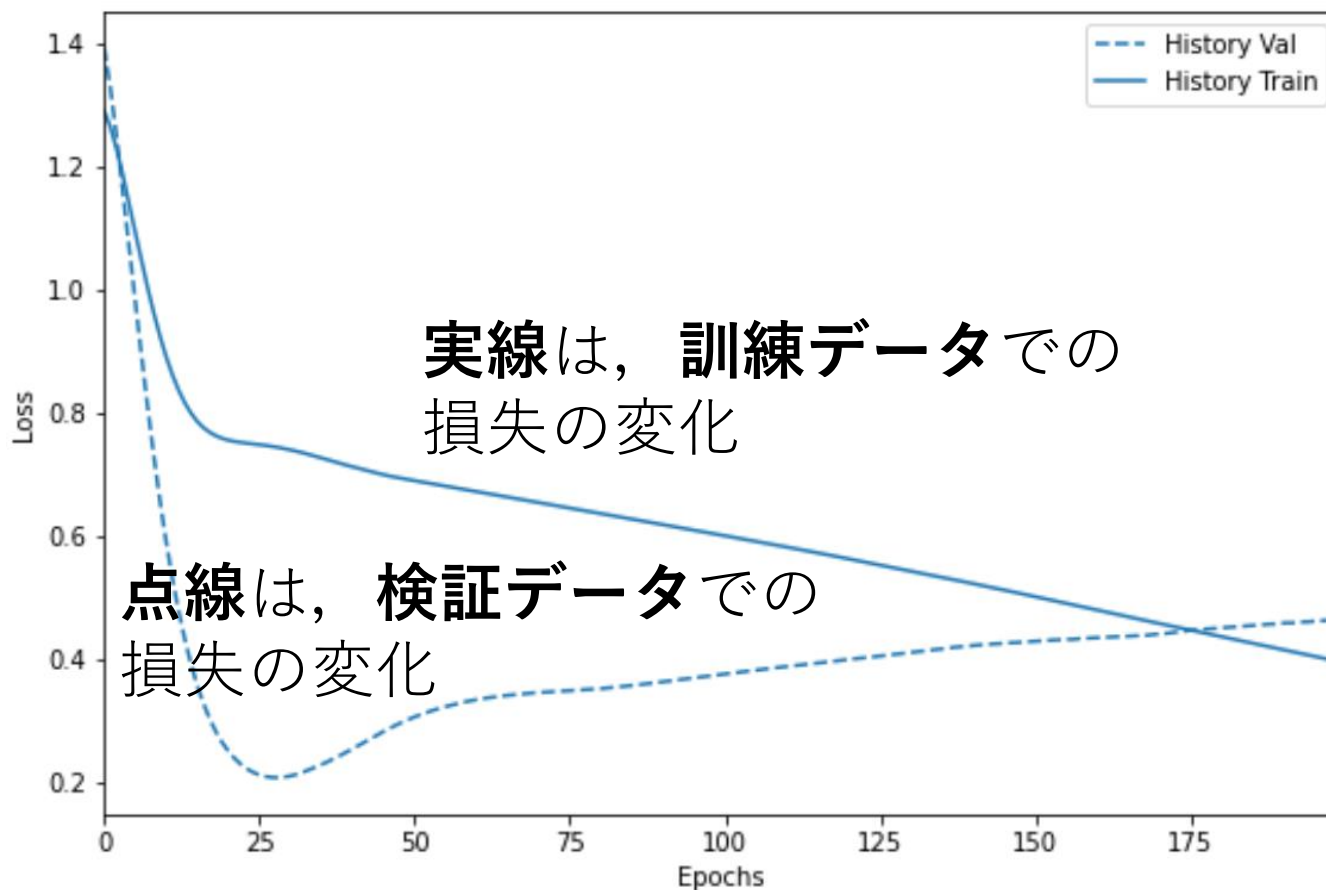
- ・ **訓練データ**での**損失** (loss)
- ・ **検証データ**での**損失** (val_loss)

などの変化を確認できる

学習曲線



学習の繰り返しでの、損失などの変化をプロットしたグラフ



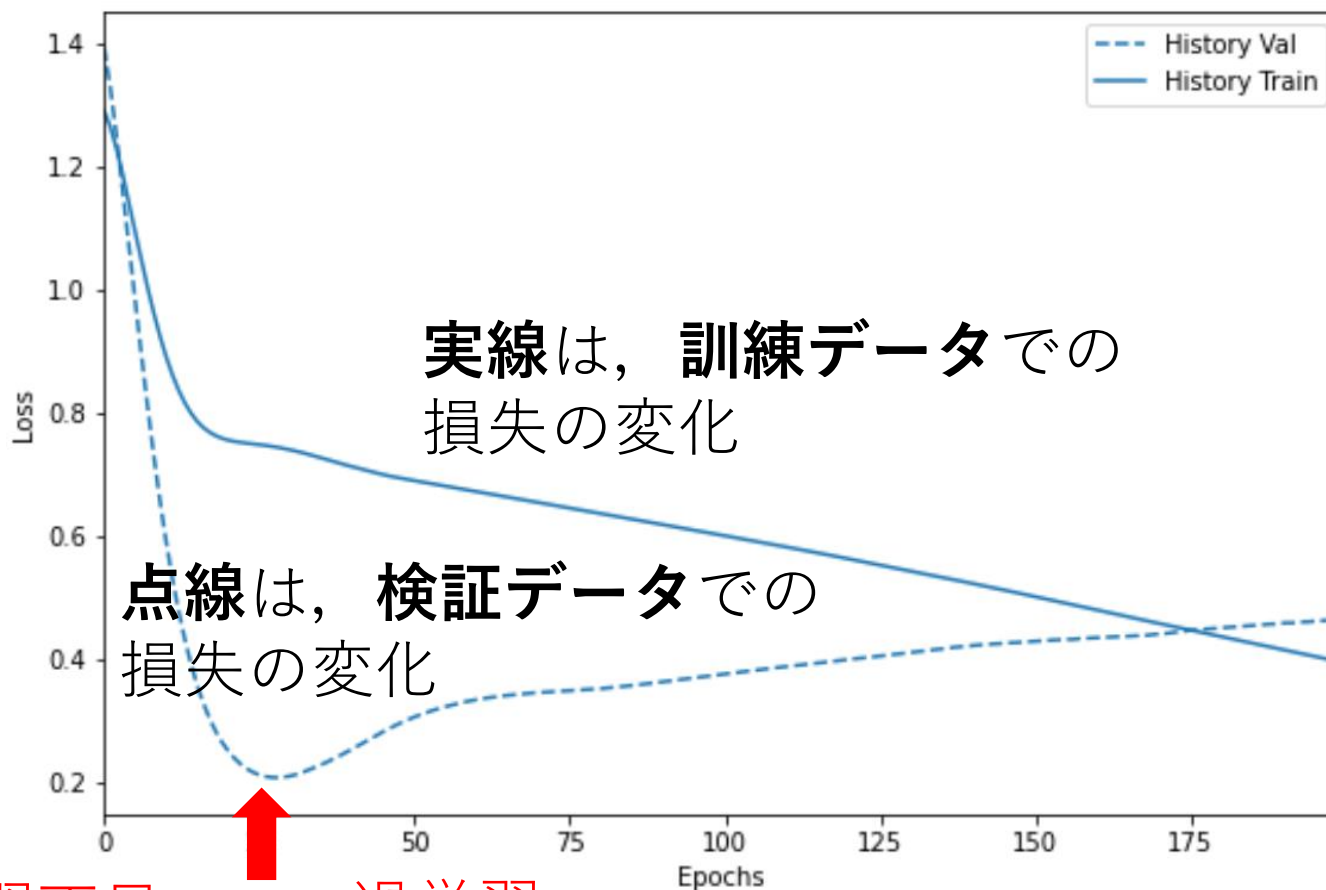
→ 学習の繰り返し

訓練データと検証データでは、違う形になることに注意

学習曲線から読み取れること



損失は少なければ少ないほど良い



学習不足 最善 過学習

学習の繰り返し
13

過学習



検証データでの**検証**により、
訓練データの量不足、
学習の繰り返し過ぎ、
ニューラルネットワークの設定の良く無さ

など、さまざまな原因により、
「**訓練データ**ではうまくいっているのに、
検証データではうまくいっていない
(汎化が**できていない**)」

という現象が起きること

学習曲線の有用性



- **学習曲線**は, **学習不足**や**過学習**についての確認, 検証を行うのに役に立つ

過学習なし



損失

高い

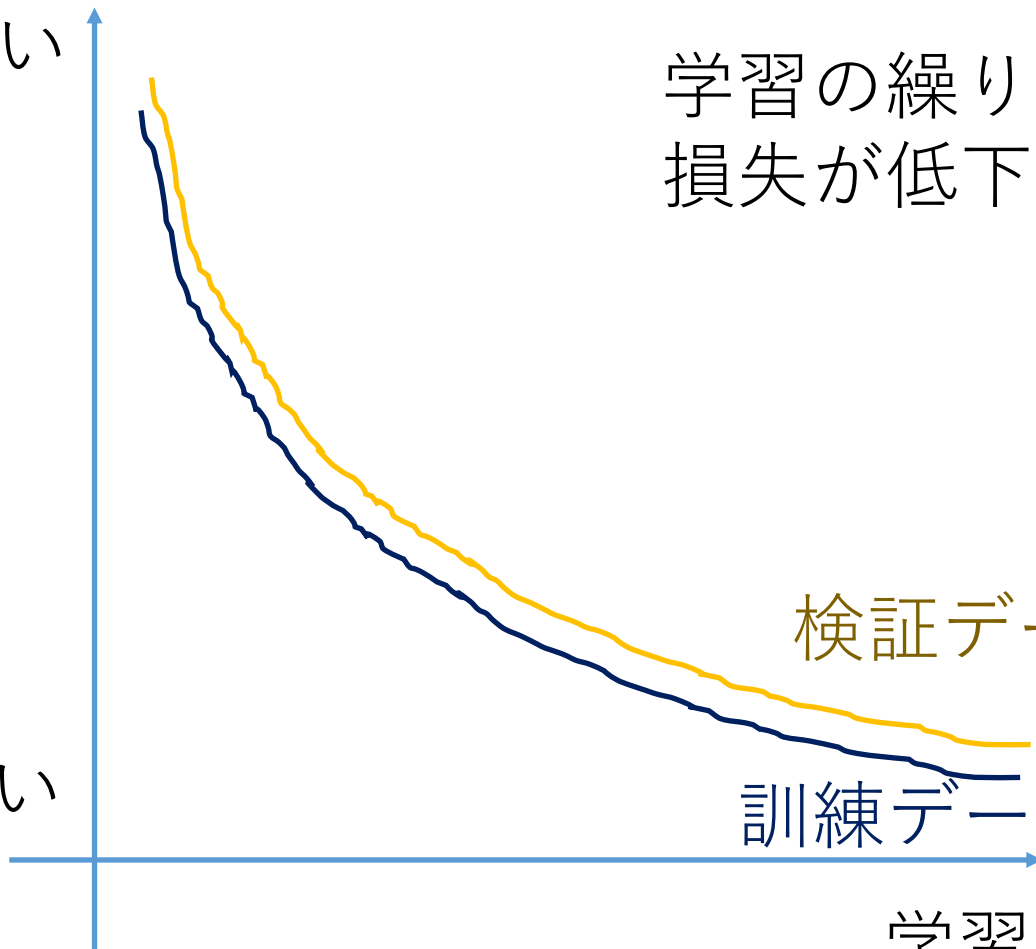
学習の繰り返しとともに
損失が低下

低い

検証データ

訓練データ

学習の繰り返し回数



過学習あり



損失

高い

学習の繰り返しに伴い、
訓練データでの損失は低下しても、
検証データでの損失が低下しない

低い

検証データ

訓練データ

学習の繰り返し回数