学習と検証、学習曲線（Iris データセット）

（人工知能応用）

URL: https://www.kkaneko.jp/cc/ni/index.html

金子邦彦

トピックス

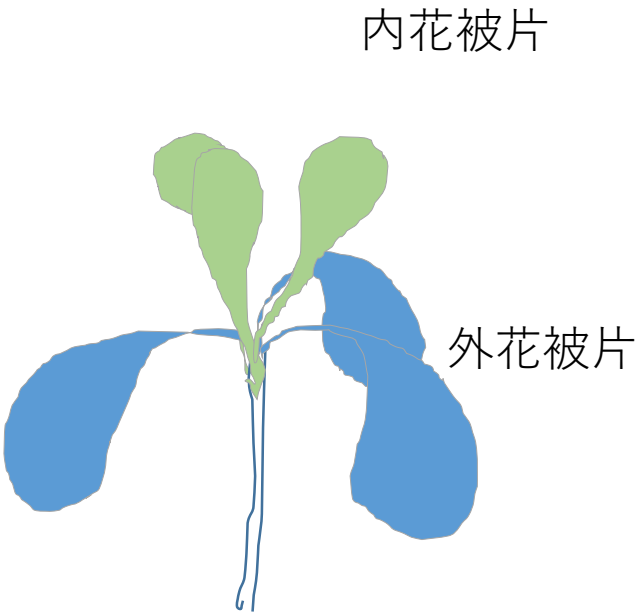
- 分類を行うニューラルネットワーク
- ニューラルネットワークの作成
- ニューラルネットワークの学習
- 学習曲線
- Iris データセット

プログラムは、次で公開

https://colab.research.google.com/drive/18rj0Lyy7rL_JJS9flrGWJR04EuI8tnTm?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

アヤメ属 (Iris)



- 多年草
- 世界に 150種. 日本に 9種.
- 花被片は 6個
- **外花被片**（がいかひへん） Sepal
  3個（大型で下に垂れる）
- **内花被片**（ないかひへん） Petal
  3個（直立する）

Iris データセット

Iris データセット（データ数は 50 × 3）
のうち、先頭 10 行

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
5	3.4	1.5	0.2	setosa
4.4	2.9	1.4	0.2	setosa

外花被片(**Sepal**)の長さと幅　**内花被片**(**Petal**)の長さと幅　**種類**

特徴量　**ラベル**

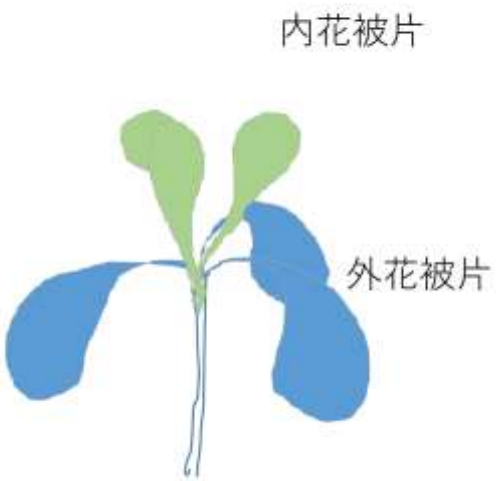
- **3種**のアヤメの**外花被辺**、**内花被片**を計測
- 種類も記録
  - **setosa**
  - **versicolor**
  - **virginica**
- データ数は **50** × **3**

作成者：Ronald Fisher

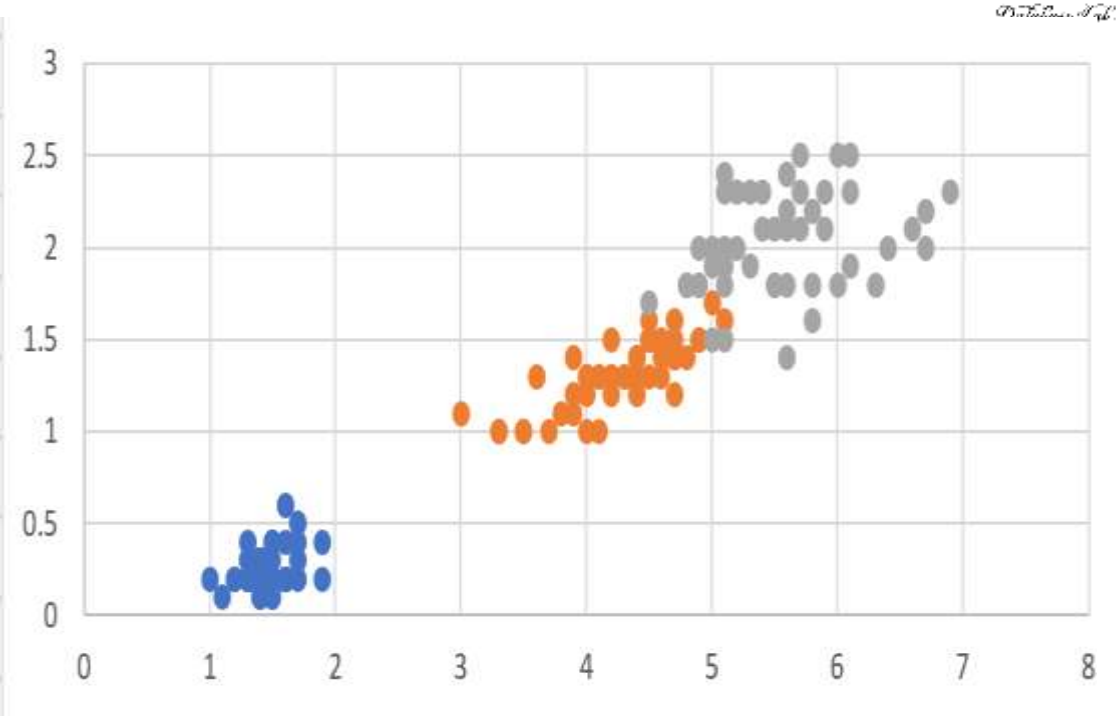
作成年：1936

Iris データセットの散布図

アヤメ属 (Iris)



縦軸：内花被片の**幅**



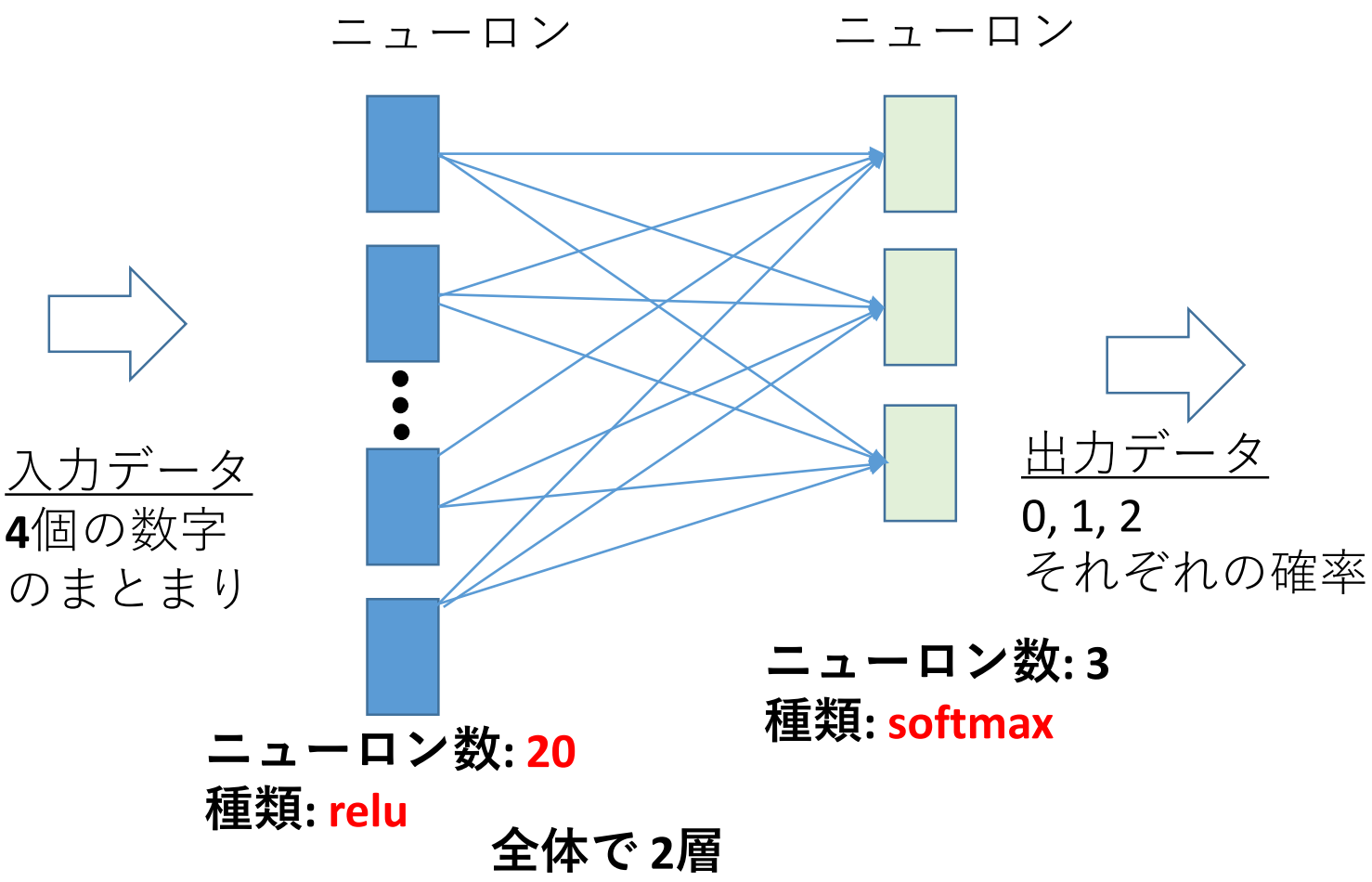
横軸：内花被片の**長さ**

次の**3種類**の分類済みのデータ

setosa
versicolor
virginica

作成するニューラルネットワーク

- **1層目：ニューロン数 20, 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

def create_model():
    return m = tf.keras.models.Sequential([
        tf.keras.layers.Dense(units=20, input_dim=4, activation='relu'),
        tf.keras.layers.Dropout(0.1),
        tf.keras.layers.Dense(units=3, activation='softmax')
    ])
```

入力データは **4**個の数字

１層目のニューロン数は **20**
種類は **relu**

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

Iris データセットから複数行を抜き出して使用

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.

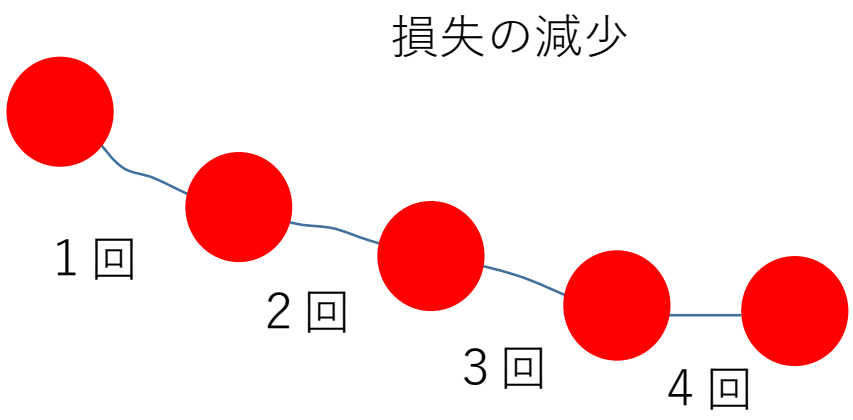
Iris データセットで，訓練データとして使用しない行を使用

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=50
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

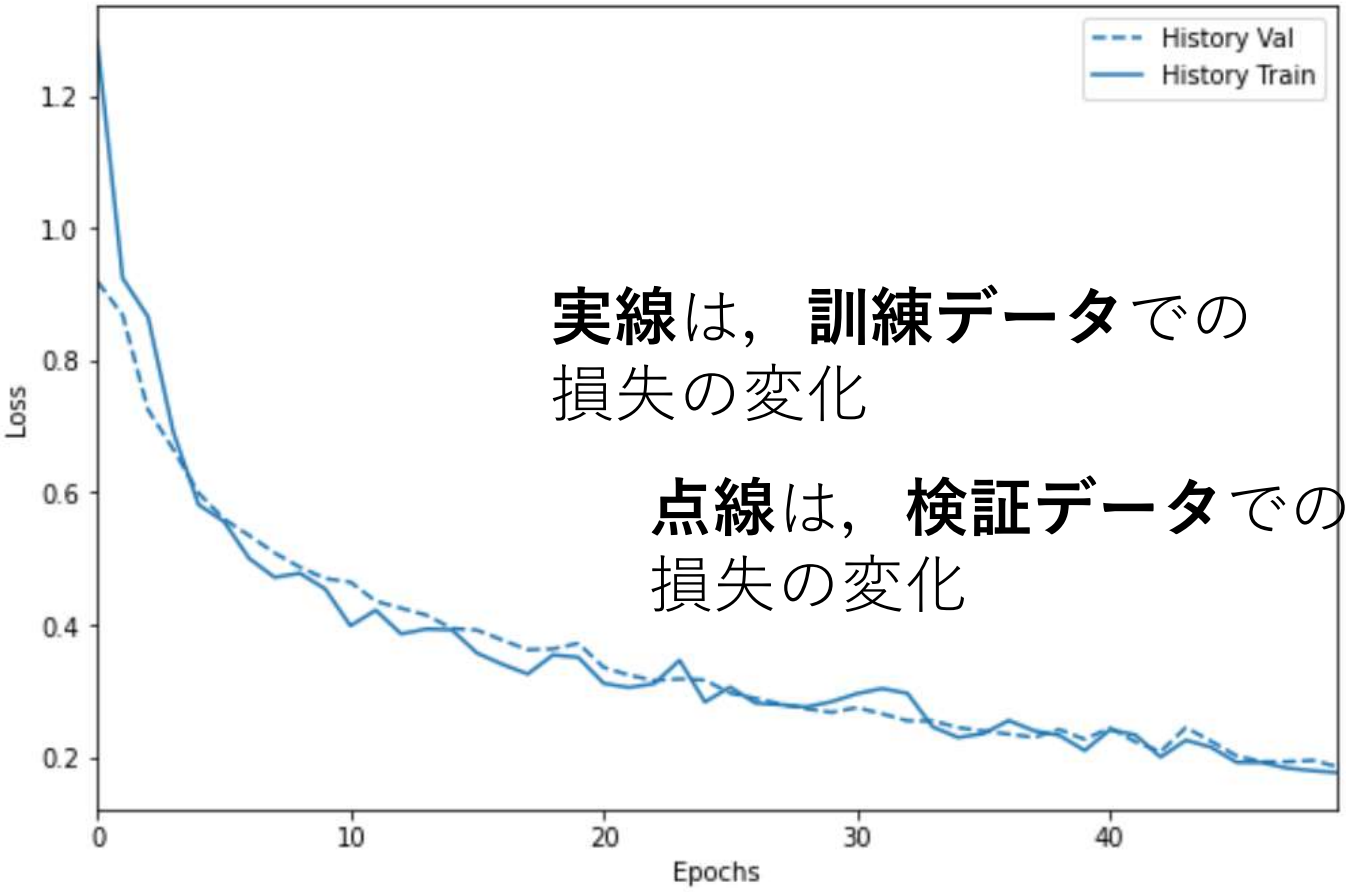
```
Epoch 1/50
3/3 - 1s - loss: 1.2824 - accuracy: 0.3200 - val_loss: 0.9192 - val_accuracy: 0.3333 - 512ms/epoch - 171ms/ste
Epoch 2/50
3/3 - 0s - loss: 0.9241 - accuracy: 0.4933 - val_loss: 0.8683 - val_accuracy: 0.4267 - 64ms/epoch - 21ms/step
Epoch 3/50
3/3 - 0s - loss: 0.8657 - accuracy: 0.4933 - val_loss: 0.7255 - val_accuracy: 0.5733 - 51ms/epoch - 17ms/step
Epoch 4/50
3/3 - 0s - loss: 0.6919 - accuracy: 0.5333 - val_loss: 0.6656 - val_accuracy: 0.6400 - 68ms/epoch - 23ms/step
Epoch 5/50
3/3 - 0s - loss: 0.5817 - accuracy: 0.7200 - val_loss: 0.5993 - val_accuracy: 0.6400 - 66ms/epoch - 22ms/step
Epoch 6/50
3/3 - 0s - loss: 0.5564 - accuracy: 0.6667 - val_loss: 0.5601 - val_accuracy: 0.6400 - 68ms/epoch - 23ms/step
Epoch 7/50
3/3 - 0s - loss: 0.5005 - accuracy: 0.6933 - val_loss: 0.5351 - val_accuracy: 0.6400 - 48ms/epoch - 16ms/step
Epoch 8/50
3/3 - 0s - loss: 0.4719 - accuracy: 0.7333 - val_loss: 0.5089 - val_accuracy: 0.8667 - 65ms/epoch - 22ms/step
Epoch 9/50
3/3 - 0s - loss: 0.4782 - accuracy: 0.8267 - val_loss: 0.4873 - val_accuracy: 0.9200 - 51ms/epoch - 17ms/step
Epoch 10/50
3/3 - 0s - loss: 0.4544 - accuracy: 0.8267 - val_loss: 0.4701 - val_accuracy: 0.7067 - 66ms/epoch - 22ms/step
Epoch 11/50
3/3 - 0s - loss: 0.3987 - accuracy: 0.8133 - val_loss: 0.4647 - val_accuracy: 0.6533 - 67ms/epoch - 22ms/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**



実線は，**訓練データ**での
損失の変化

点線は，**検証データ**での
損失の変化

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