2022/11/17 18:00~19:00

深層学習にふれてみよう(縮小版②)

本教材を使用した際にはお手数ですが、
下記アンケートフォームにご協力下さい。

https://forms.gle/cgej2DL5PvneRhCp8

東京医科歯科大学
統合教育機構
須藤　毅顕

前回の線形回帰の復習

scikit-learnを用いた機械学習の書き方

①学習モデルの選択(今回は線形回帰)
(モデル名) = LinearRegression()

②データを入れて学習させる
(モデル名).fit(説明変数,目的変数)

③傾き(偏回帰係数)と切片(定数項)を求める
(モデル名).coef_　　#傾き
(モデル名).intercept_　#切片

④予測を行う
(モデル名).predict(新たな説明変数)

線形回帰は機械学習の1種

機械学習を実践してみよう！！

機械学習の分類

機械学習

正解データとセットで学習

教師あり学習

- 回帰
  - 与えられたデータから未来のデータを予測
  - 線形回帰など
- 分類
  - 与えられたデータを分類
  - ロジスティック回帰など

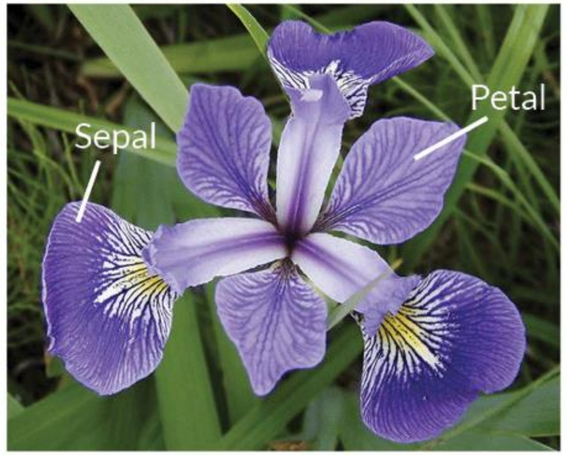
正解データは与えずに学習

教師なし学習

- クラスタリング
  - グループ分け
  - K-means法など
- 次元削減
  - 2次元に可視化
  - PCAなど

ロジスティック回帰分析にトライしよう

あやめのデータ(2.csv)



Iris Versicolor
ブルーフラッグ

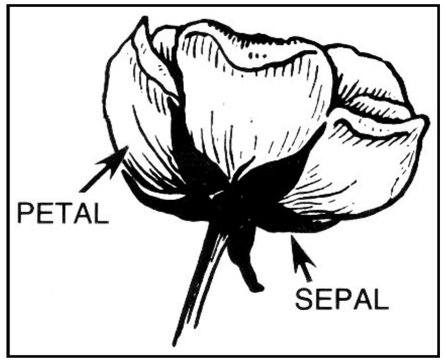
Iris Setosa
ヒオウギアヤメ

Iris Virginica
バージニカ

150行✖6列

変数4つ
- Sepal(がく片)の長さと幅
- Petal(花びら)の長さと幅

正解が3種類
- ヒオウギアヤメ(0)
- ブルーフラッグ(1)
- バージニカ(2)



	A	B	C	D	E	F
1	がく片の長さ	がく片の幅	花びらの長さ	花びらの幅	アヤメの種類_数字	アヤメの種類
2	5.1	3.5	1.4	0.2	0	ヒオウギアヤメ
3	4.9	3	1.4	0.2	0	ヒオウギアヤメ
4	4.7	3.2	1.3	0.2	0	ヒオウギアヤメ
5	4.6	3.1	1.5	0.2	0	ヒオウギアヤメ
6	5	3.6	1.4	0.2	0	ヒオウギアヤメ
7	5.4	3.9	1.7	0.4	0	ヒオウギアヤメ
8	4.6	3.4	1.4	0.3	0	ヒオウギアヤメ
9	5	3.4	1.5	0.2	0	ヒオウギアヤメ
10	4.4	2.9	1.4	0.2	0	ヒオウギアヤメ
11	4.9	3.1	1.5	0.1	0	ヒオウギアヤメ
12	5.4	3.7	1.5	0.2	0	ヒオウギアヤメ
⋮						
50	5.3	3.7	1.5	0.2	0	ヒオウギアヤメ
51	5	3.3	1.4	0.2	0	ヒオウギアヤメ
52	7	3.2	4.7	1.4	1	ブルーフラッグ
53	6.4	3.2	4.5	1.5	1	ブルーフラッグ
54	6.9	3.1	4.9	1.5	1	ブルーフラッグ
55	5.5	2.3	4	1.3	1	ブルーフラッグ
56	6.5	2.8	4.6	1.5	1	ブルーフラッグ
57	5.7	2.8	4.5	1.3	1	ブルーフラッグ
58	6.3	3.3	4.7	1.6	1	ブルーフラッグ
⋮						
141	6.9	3.1	5.4	2.1	2	バージニカ
142	6.7	3.1	5.6	2.4	2	バージニカ
143	6.9	3.1	5.1	2.3	2	バージニカ
144	5.8	2.7	5.1	1.9	2	バージニカ
145	6.8	3.2	5.9	2.3	2	バージニカ
146	6.7	3.3	5.7	2.5	2	バージニカ
147	6.7	3	5.2	2.3	2	バージニカ
148	6.3	2.5	5	1.9	2	バージニカ
149	6.5	3	5.2	2	2	バージニカ
150	6.2	3.4	5.4	2.3	2	バージニカ
151	5.9	3	5.1	1.8	2	バージニカ

まずはデータを読み込んでみよう

pandasというライブラリをインストールしてpdと省略して使います

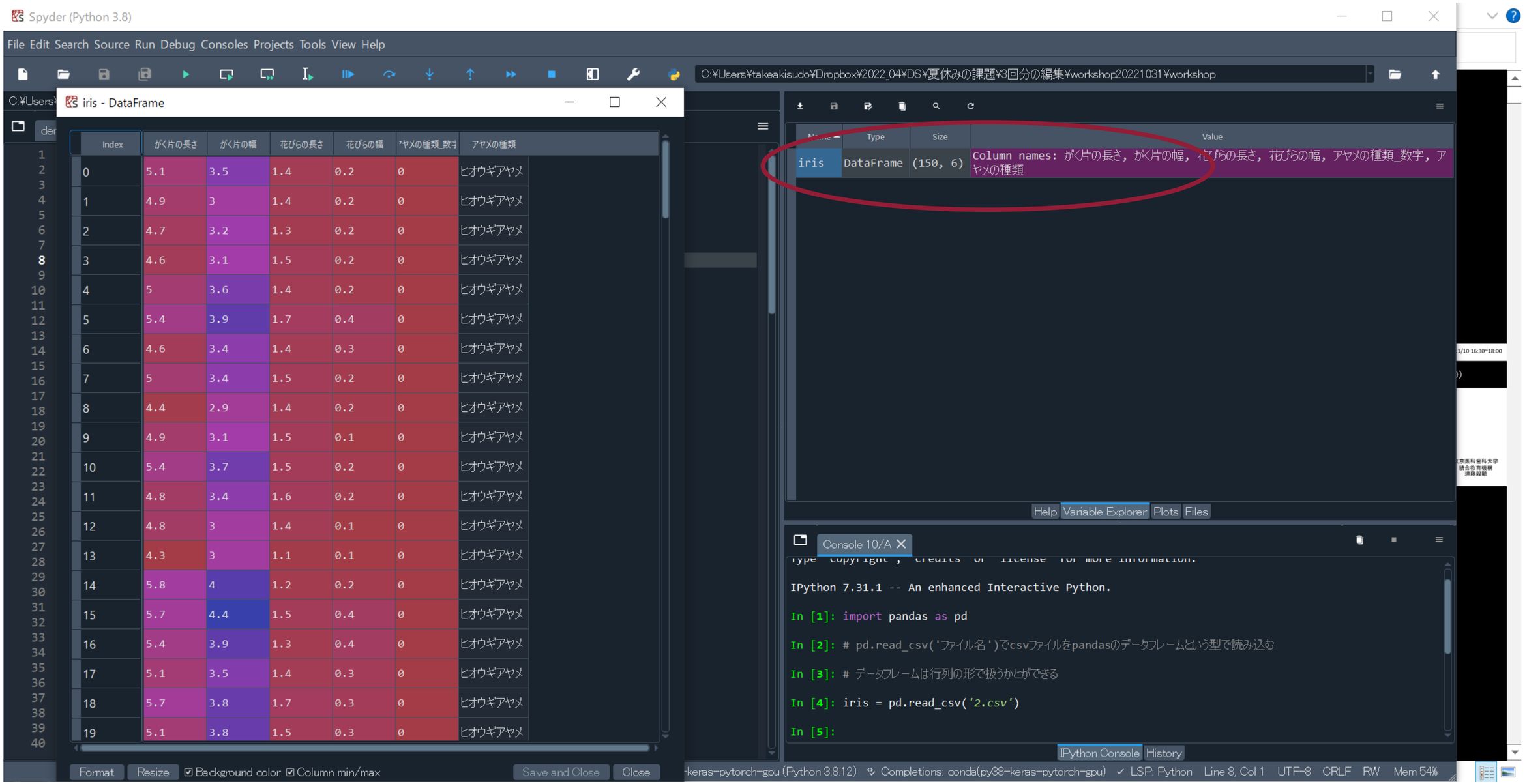
```
import pandas as pd
```

pd.read_csv(‘ファイル名.csv’)でファイルをpandasの形式で読み込みます

```
iris = pd.read_csv('2.csv')
```

pandasで読み込んだデータの型をデータフレームと言います。
今回は読み込んだ2.csvのデータをirisという変数名で格納しています。

変数エクスプローラーのirisをダブルクリックするとデータフレームをみることが出来る



ロジスティック回帰でアヤメを分類する

がく片の長さでヒオウギアヤメとブルーフラッグを分類する

①学習モデルの選択(前回は線形回帰)
(モデル名) = LinearRegression()

②データを入れて学習させる
(モデル名).fit(説明変数,目的変数)

③傾き(偏回帰係数)と切片(定数項)を求める
(モデル名).coef_　　#傾き
(モデル名).intercept_　#切片

④予測を行う
(モデル名).predict(新たな説明変数)

ロジスティック回帰でアヤメを分類する

がく片の長さでヒオウギアヤメとブルーフラッグを分類する

①学習モデルの選択(前回は線形回帰)
(モデル名) = LinearRegression()

②データを入れて学習させる
(モデル名).fit(説明変数,目的変数)

③傾き(偏回帰係数)と切片(定数項)を求める
(モデル名).coef_　　#傾き
(モデル名).intercept_　#切片

④予測を行う
(モデル名).predict(新たな説明変数)

①学習モデルの選択(今回はロジスティック回帰)
(モデル名) = LogisticRegression()

②データを入れて学習させる
(モデル名).fit(説明変数,目的変数)

③傾き(偏回帰係数)と切片(定数項)を求める
(モデル名).coef_　　#傾き
(モデル名).intercept_　#切片

④予測を行う
(モデル名).predict(新たな説明変数)
(モデル名).predict_proba(新たな説明変数)

行の操作

データフレーム[a:b]でa行目(以上)からb行目(未満)までを取り出す

```
a = iris[0:4]
print(a)
```

データフレーム.shapeで行列(2次元配列)の形状を調べることが出来る

```
print(iris.shape)
print(a.shape)
```

行の操作

データフレーム[a:b]でa行目(以上)からb行目(未満)までを取り出す

```
a = iris[0:4]
print(a)
```

データフレーム.shapeで行列(2次元配列)の形状を調べることが出来る

```
print(iris.shape)
print(a.shape)
```

```
In [9]: print(iris.shape)
(150, 6)

In [10]: print(a.shape)
(4, 6)
```

a - DataFrame

Index	がく片の長さ	がく片の幅	花びらの長さ	花びらの幅	ｱﾔﾒの種類_数字	アヤメの種類
0	5.1	3.5	1.4	0.2	0	ヒオウギアヤメ
1	4.9	3	1.4	0.2	0	ヒオウギアヤメ
2	4.7	3.2	1.3	0.2	0	ヒオウギアヤメ
3	4.6	3.1	1.5	0.2	0	ヒオウギアヤメ

列の操作

1列のみを取り出すにはデータフレーム[列の名前]で取り出す

```
b = iris['がく片の長さ']
print(b)
print(b.shape)
```

2列以上を取り出すにはデータフレーム[['列の名前','列の名前']]で取り出す

```
c = iris[['がく片の長さ','がく片の幅']]

print(c)
print(c.shape)
```

b - Series

Index	がく片の長さ
0	5.1
1	4.9
2	4.7
3	4.6
4	5
5	5.4

c - DataFrame

Index	がく片の長さ	がく片の幅
0	5.1	3.5
1	4.9	3
2	4.7	3.2
3	4.6	3.1
4	5	3.6
5	5.4	3.9

列の操作

データフレーム[‘列の名前’]だとデータフレームの形(2次元配列)ではなくなる
2次元配列として取り出すには1列のみであってもデータフレーム[['列の名前']]とする

```
d = iris[['がく片の長さ']]
print(d)
print(d.shape)
```

データフレームを1列だけ
取り出した型をSeriesと言います

Name	Type	Size	Value
a	DataFrame	(4, 6)	Column names: がく片の長さ, がく片の幅, 花びらの長さ, 花びらの幅, アヤメの種類_数字, アヤメの種類
b	Series	(150,)	Series object of pandas.core.series module
c	DataFrame	(150, 2)	Column names: がく片の長さ, がく片の幅
d	DataFrame	(150, 1)	Column names: がく片の長さ
iris	DataFrame	(150, 6)	Column names: がく片の長さ, がく片の幅, 花びらの長さ, 花びらの幅, アヤメの種類_数字, アヤメの種類

学習に用いる説明変数と目的変数を設定する

説明変数(目的を知るために使用する変数)：がく片の長さ→ x

目的変数(目的(分類や予測)となる変数)：アヤメの種類_数字→ y

```
from sklearn.linear_model import LogisticRegression
df = iris[0:100]
x = df[['がく片の長さ']]
y = df[‘アヤメの種類_数字’]
print(x)
print(y)
```

ライブラリの読み込み
irisの1行目から100行目まで(ヒオウギアヤメとブルーフラッグ)を取り出して、dfという名前の変数に代入

学習に用いる説明変数と目的変数を設定する

説明変数(目的を知るために使用する変数)：がく片の長さ→ｘ

目的変数(目的(分類や予測)となる変数)：アヤメの種類_数字→ｙ

```
from sklearn.linear_model import LogisticRegression
df = iris[0:100]
x = df[['がく片の長さ']]
y = df[‘アヤメの種類_数字’]
print(x)
print(y)
```

xに"がく片の長さ"、yに"アヤメの種類_数字”の列の内容を代入
xは2次元配列(線形回帰と同様)

学習に用いる説明変数と目的変数を設定する

説明変数(目的を知るために使用する変数)：がく片の長さ→ｘ

目的変数(目的(分類や予測)となる変数)：アヤメの種類_数字→ｙ

```
from sklearn.linear_model import LogisticRegression
df = iris[0:100]
x = df[['がく片の長さ']]
y = df['アヤメの種類_数字']
print(x)
print(y)
```

xに"がく片の長さ"、yに"アヤメの種類_数字”の列の内容を代入
xは2次元配列(線形回帰と同様)

```
In [25]: print(x)
    がく片の長さ
0         5.1
1         4.9
2         4.7
3         4.6
4         5.0
..        ...
95        5.7
96        5.7
97        6.2
98        5.1
99        5.7

[100 rows x 1 columns]
```

```
In [26]: print(y)
0     0
1     0
2     0
3     0
4     0
     ..
95    1
96    1
97    1
98    1
99    1
Name: アヤメの種類_数字, Length: 100, dtype: int64
```

学習に用いる説明変数と目的変数を設定する

x(DataFrame)とy(Series)をnumpy配列に変換する

変数名 = データフレーム.to_numpy()

```
x2 = x.to_numpy()
y2 = y.to_numpy()
```

x	DataFrame	(100, 1)	Column names: がく片の長さ
y	Series	(100,)	Series object of pandas.core.series module

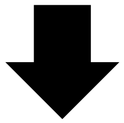
学習に用いる説明変数と目的変数を設定する

x(DataFrame)とy(Series)をnumpy配列に変換する

変数名 = データフレーム.to_numpy()

```
x2 = x.to_numpy()
y2 = y.to_numpy()
```

x	DataFrame	(100, 1)	Column names: がく片の長さ
y	Series	(100,)	Series object of pandas.core.series module



x2	Array of float64	(100, 1)	[[5.1] [4.9]
y2	Array of int64	(100,)	[0 0 0 ... 1 1 1]

データフレームでも学習出来ますが予測の際に警告が出ないようにするために変換してます

ヒオウギアヤメのがく片の長さとがく片の幅でロジスティック回帰

学習は線形回帰と同様

```
model2 = LogisticRegression()
model2.fit(x2, y2)

print(model2.coef_)
print(model2.intercept_)
```

作った学習モデルで分類をする

学習モデルを作ったのでpredictで予測する

がく片の長さが4.5, 5, 7,0の時の予測結果は？

In

```
test4 = [[4.5],[5.0],[7.0]]
result4 = model2.predict(test4)
print(result4)
print("がく片の長さが4.5の時の予測結果は:",result4[0])
print("がく片の長さが5.0の時の予測結果は:",result4[1])
print("がく片の長さが7.0の時の予測結果は:",result4[2])
```

Out

ヒオウギアヤメ(=0)、ヒオウギアヤメ(=0)、ブルーフラッグ(=1)と予測された

作った学習モデルで分類をする

学習モデルを作ったのでpredictで予測する

がく片の長さが4.5, 5, 7,0の時の予測結果は？

In

```
test4 = [[4.5],[5.0],[7.0]]
result4 = model2.predict(test4)
print(result4)
print("がく片の長さが4.5の時の予測結果は:",result4[0])
print("がく片の長さが5.0の時の予測結果は:",result4[1])
print("がく片の長さが7.0の時の予測結果は:",result4[2])
```

Out

```
print("がく片の長さが4.5の時の予測結果は:",result4[0])
がく片の長さが4.5の時の予測結果は: 0

print("がく片の長さが5.0の時の予測結果は:",result4[1])
がく片の長さが5.0の時の予測結果は: 0

print("がく片の長さが7.0の時の予測結果は:",result4[2])
がく片の長さが7.0の時の予測結果は: 1
```

ヒオウギアヤメ(=0)、ヒオウギアヤメ(=0)、ブルーフラッグ(=1)と予測された

作った学習モデルで分類をする

学習モデルを作ったのでpredict_probaで確率を予測する
がく片の長さが4.5, 5, 7,0の時のブルーフラッグの確率は？

In

```
result5 = model2.predict_proba(test4)
print(result5)
print("がく片の長さが4.5の時のヒオウギアヤメとブルーフラッグの確率は:",result5[0])
print("がく片の長さが5.0の時のヒオウギアヤメとブルーフラッグの確率は:",result5[1])
print("がく片の長さが7.0の時のヒオウギアヤメとブルーフラッグの確率は:",result5[2])
```

作った学習モデルで分類をする

学習モデルを作ったのでpredict_probaで確率を予測する
がく片の長さが4.5, 5, 7,0の時のブルーフラッグの確率は？

In

```
result5 = model2.predict_proba(test4)
print(result5)
print("がく片の長さが4.5の時のヒオウギアヤメとブルーフラッグの確率は:",result5[0])
print("がく片の長さが5.0の時のヒオウギアヤメとブルーフラッグの確率は:",result5[1])
print("がく片の長さが7.0の時のヒオウギアヤメとブルーフラッグの確率は:",result5[2])
```

Out

```
print(result5)
[[0.95036498 0.04963502]
 [0.79665518 0.20334482]
 [0.00682033 0.99317967]]

print("がく片の長さが4.5の時のヒオウギアヤメとブルーフラッグの確率は:",result5[0])
がく片の長さが4.5の時のヒオウギアヤメとブルーフラッグの確率は: [0.95036498 0.04963502]

print("がく片の長さが5.0の時のヒオウギアヤメとブルーフラッグの確率は:",result5[1])
がく片の長さが5.0の時のヒオウギアヤメとブルーフラッグの確率は: [0.79665518 0.20334482]

print("がく片の長さが7.0の時のヒオウギアヤメとブルーフラッグの確率は:",result5[2])
がく片の長さが7.0の時のヒオウギアヤメとブルーフラッグの確率は: [0.00682033 0.99317967]
```

がく片の長さが4.5の時は0.0496、5.0の時は0.2033、7.0の時は0.9932と予測された

説明変数を2つ使って分類する

がく片の長さと幅の2つの変数だとどうなる？

```
x = df[[‘がく片の長さ’]]
y = df[‘アヤメの種類_数字’]

x2 = x.to_numpy()
y2 = y.to_numpy()

model2 = LogisticRegression()
model2.fit(x, y)

print(model2.coef_)
print(model2.intercept_)
```

説明変数を2つ使って分類する

がく片の長さと幅の2つの変数だとどうなる？

```
x = df[[‘がく片の長さ’]]
y = df[‘アヤメの種類_数字’]

x2 = x.to_numpy()
y2 = y.to_numpy()

model2 = LogisticRegression()
model2.fit(x, y)

print(model2.coef_)
print(model2.intercept_)
```

```
x = df[['がく片の長さ','がく片の幅']]
y = df['アヤメの種類_数字’]

x2 = x.to_numpy()
y2 = y.to_numpy()

model2 = LogisticRegression()
model2.fit(x2, y2)

print(model2.coef_)
print(model2.intercept_)
```

説明変数を2つ使って分類する

がく片の長さと幅が、(4.5 , 3.2)、(6.0 , 2.5)、(7.0, 6.0)の時は？

```
test = [[4.5,3.2],[6.0,2.5],[7.0,6.0]]
result = model2.predict_proba(test)
print(result)
```

```
print(result)
[[0.95781722 0.04218278]
 [0.02631867 0.97368133]
 [0.97991578 0.02008422]]
```

-1

前半のまとめ

機械学習の一歩目として、
線形回帰とロジスティック回帰を実践しました

①学習モデルの選択(今回は線形回帰)
(モデル名) = LinearRegression()

②データを入れて学習させる
(モデル名).fit(説明変数,目的変数)

③傾き(偏回帰係数)と切片(定数項)を求める
(モデル名).coef_　　#傾き
(モデル名).intercept_　#切片

④予測を行う
(モデル名).predict(新たな説明変数)

①学習モデルの選択(今回はロジスティック回帰)
(モデル名) = LogisticRegression()

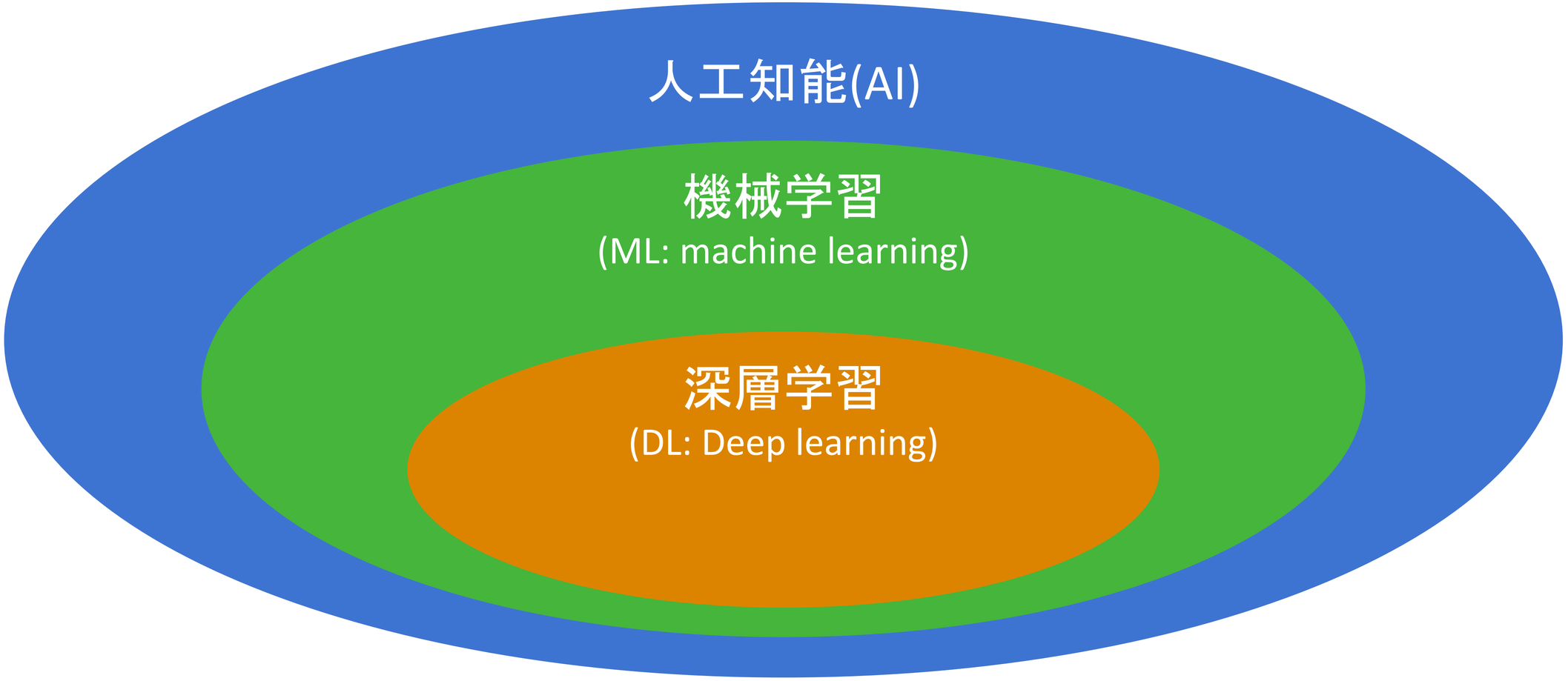
②データを入れて学習させる
(モデル名).fit(説明変数,目的変数)

③傾き(偏回帰係数)と切片(定数項)を求める
(モデル名).coef_　　#傾き
(モデル名).intercept_　#切片

④予測を行う
(モデル名).predict(新たな説明変数)
(モデル名).predict_proba(新たな説明変数)

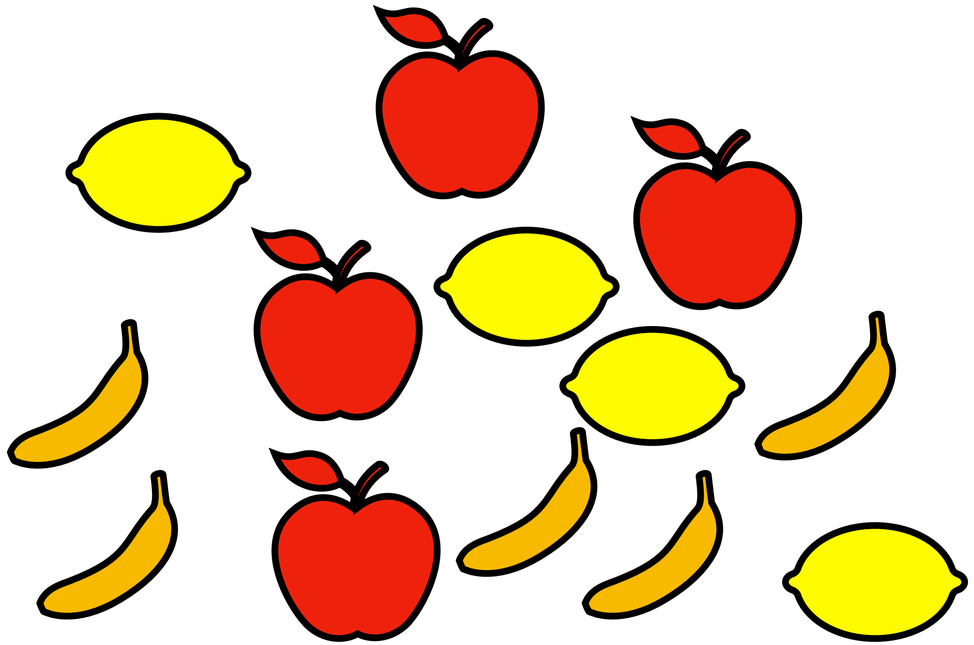
深層学習を実践してみよう！！

- 機械学習は人工知能の1つの分野
- 後で出てくる深層学習は機械学習の１種



深層学習を実践してみよう！！

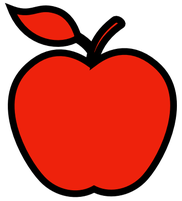
基本的な全体の流れはこれまでと同様



入力

学習

線形回帰
ロジスティック回帰



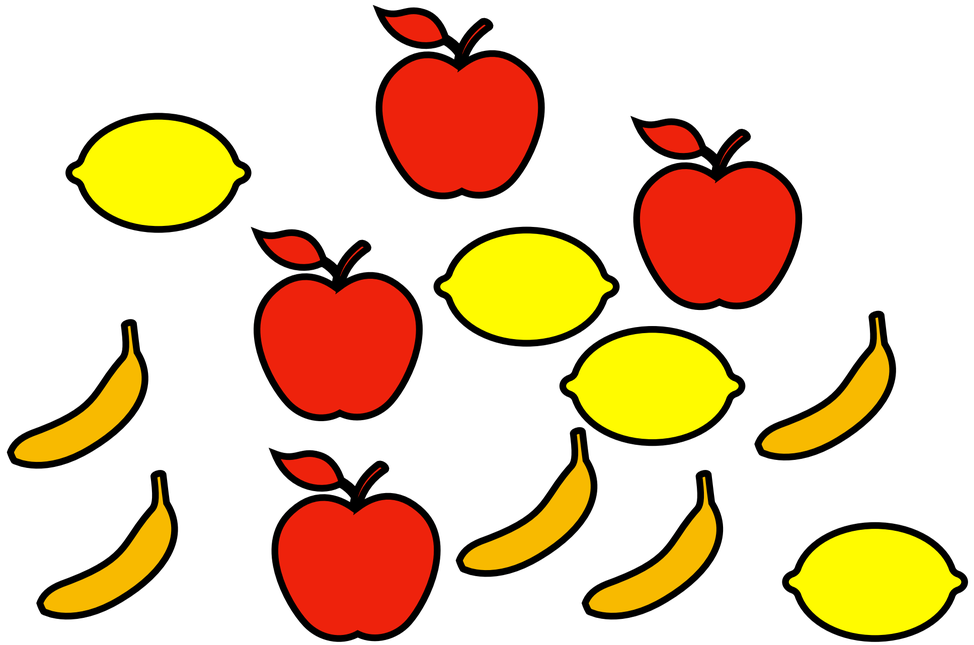
学習済みモデル
(≒分類マシーン)

りんご!!

深層学習では、学習モデルにニューラルネットワークを用いる

深層学習を実践してみよう！！

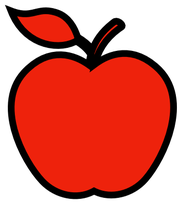
基本的な全体の流れはこれまでと同様



入力

学習

ニューラルネットワーク



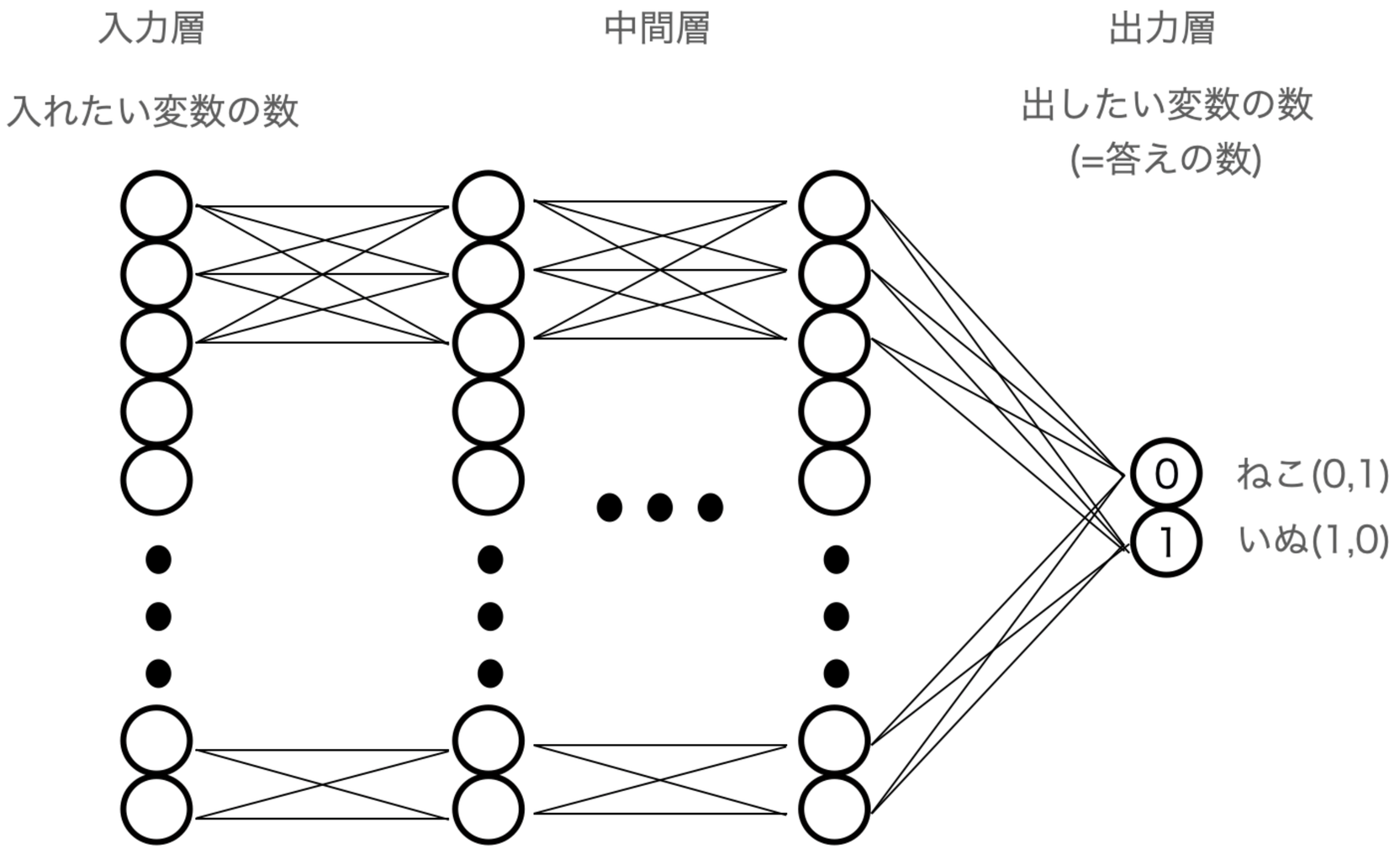
学習済みモデル

(≒分類マシーン)

りんご!!

深層学習では、学習モデルにニューラルネットワークを用いる

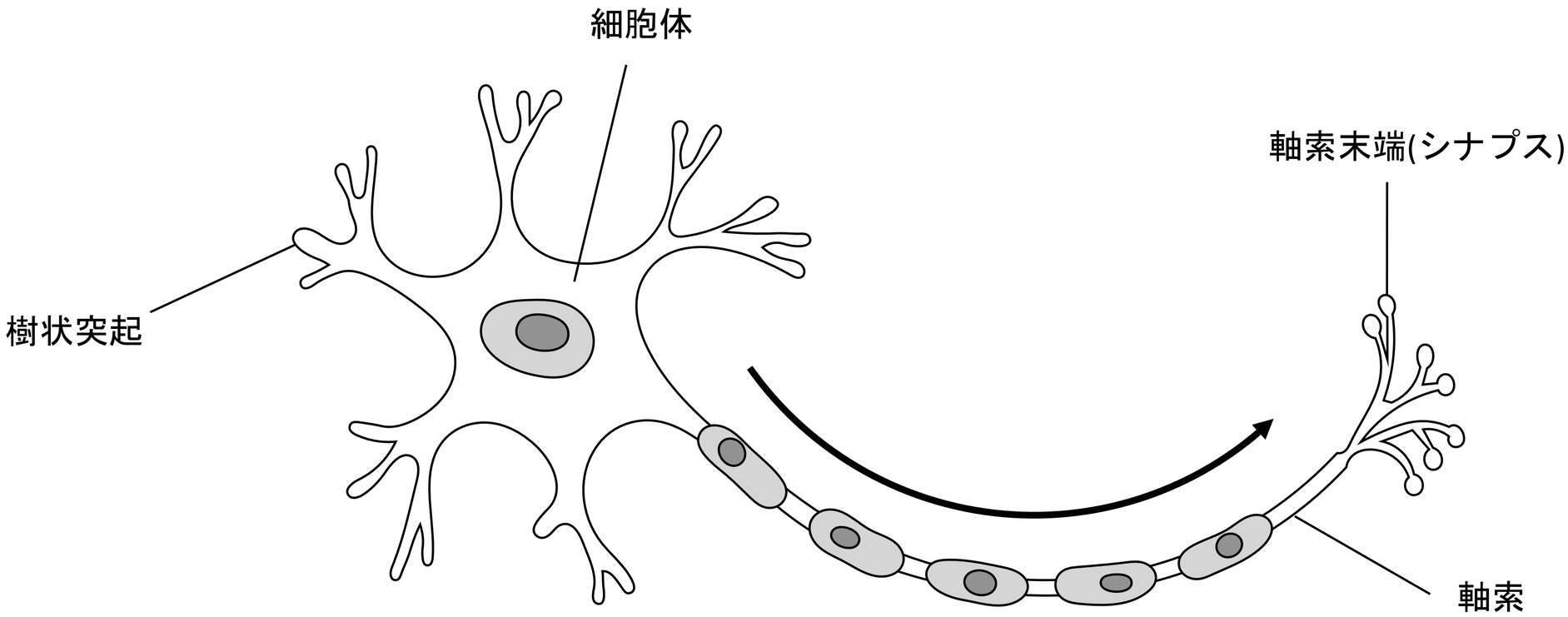
ニューラルネットワークとは



○が全てニューロン、繋がった線の数だけ数式(関数)が存在する。

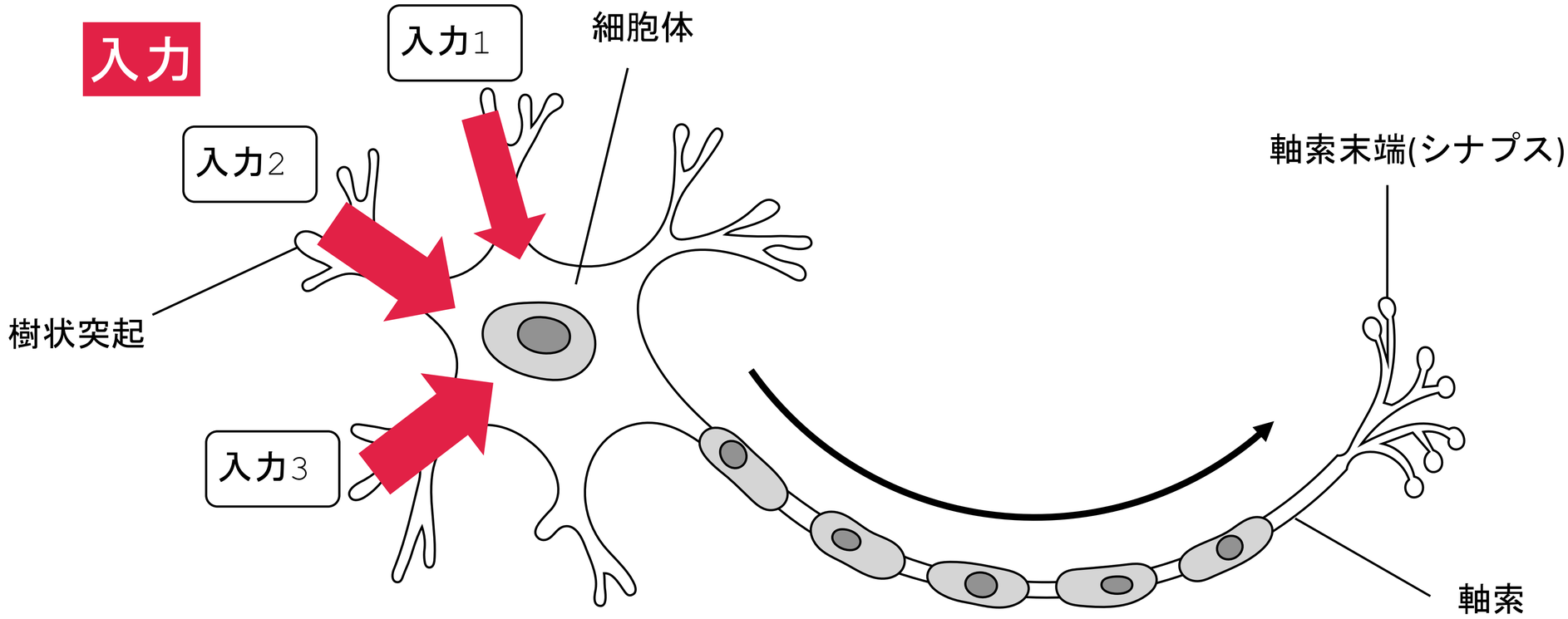
ニューラルネットワークとは

- ニューロンは、樹状突起、細胞体、軸索からなる
- ニューロンは、樹状突起から入力された電気信号が神経細胞内の電位を超えるかどうかの閾値を持っている
- 閾値を超えるとニューロンは興奮状態となり、軸索末端から電気信号が出力される



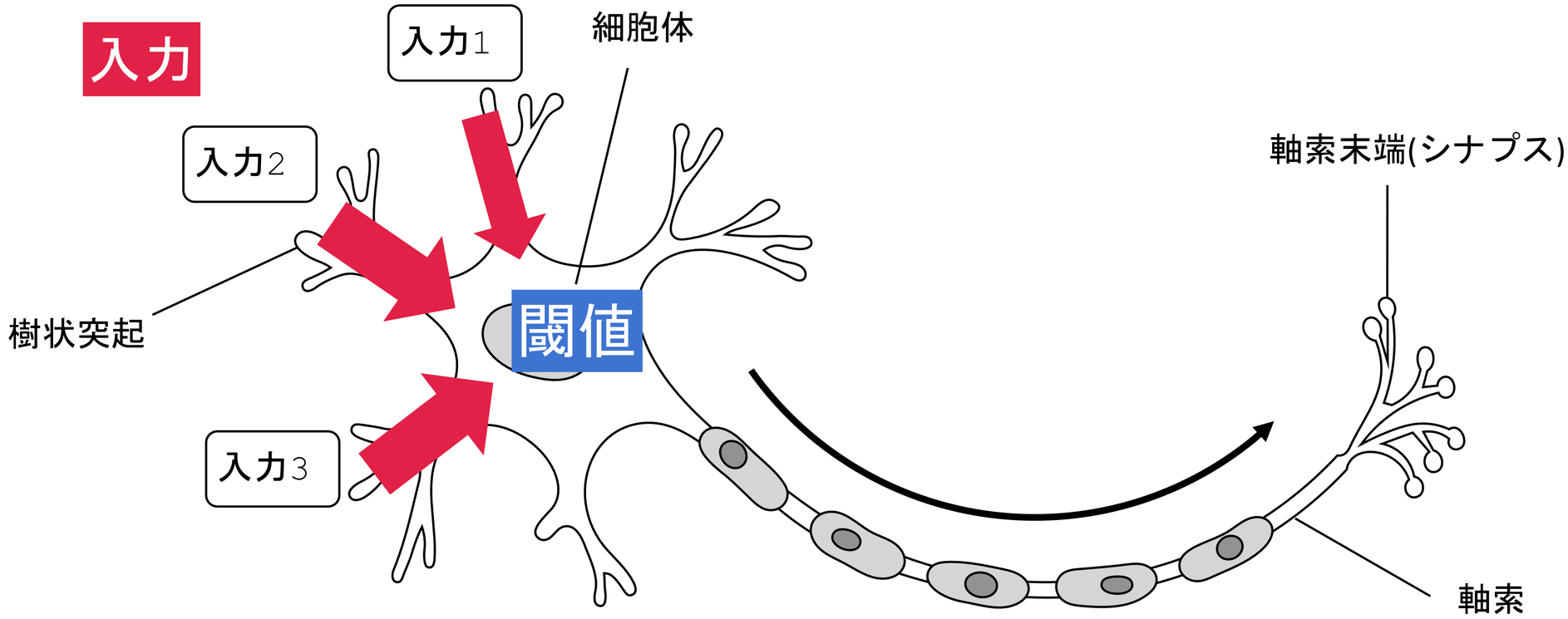
ニューラルネットワークとは

- ニューロンは、樹状突起、細胞体、軸索からなる
- ニューロンは、樹状突起から入力された電気信号が神経細胞内の電位を超えるかどうかの閾値を持っている
- 閾値を超えるとニューロンは興奮状態となり、軸索末端から電気信号が出力される



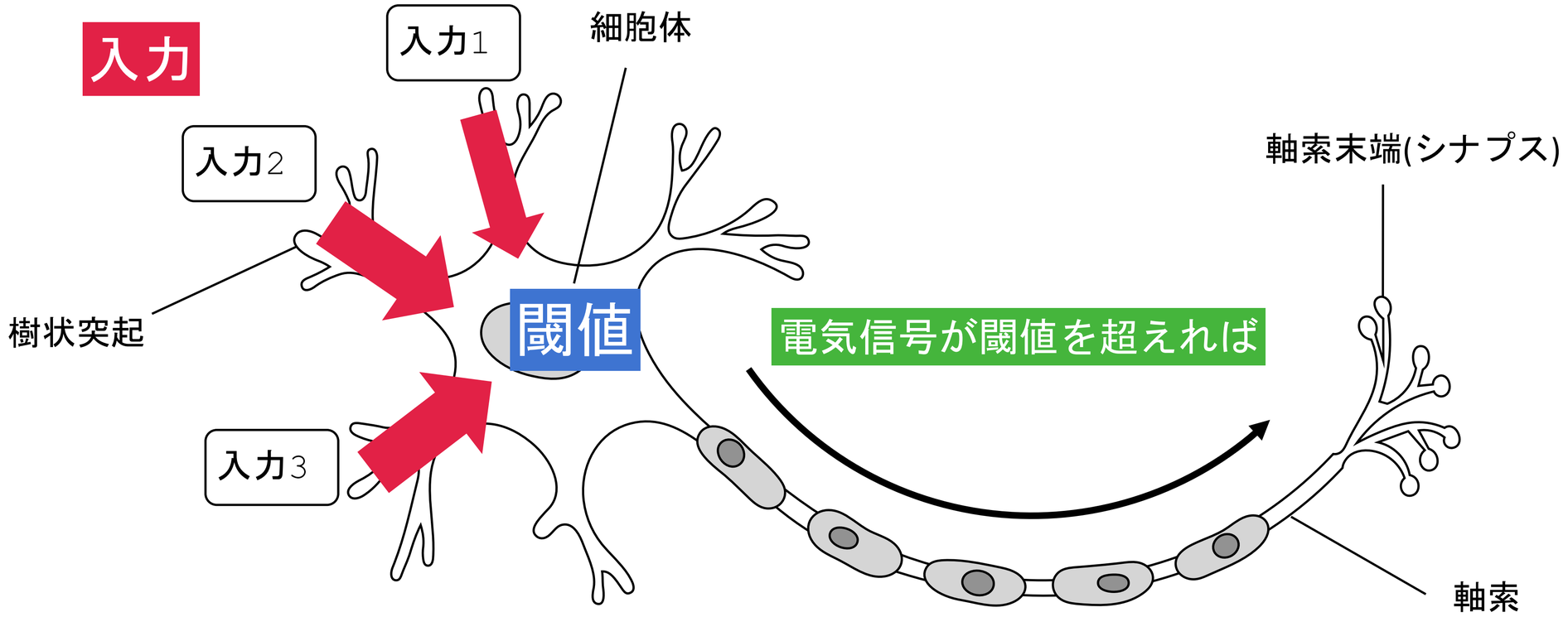
ニューラルネットワークとは

- ニューロンは、樹状突起、細胞体、軸索からなる
- ニューロンは、樹状突起から入力された電気信号が神経細胞内の電位を超えるかどうかの閾値を持っている
- 閾値を超えるとニューロンは興奮状態となり、軸索末端から電気信号が出力される



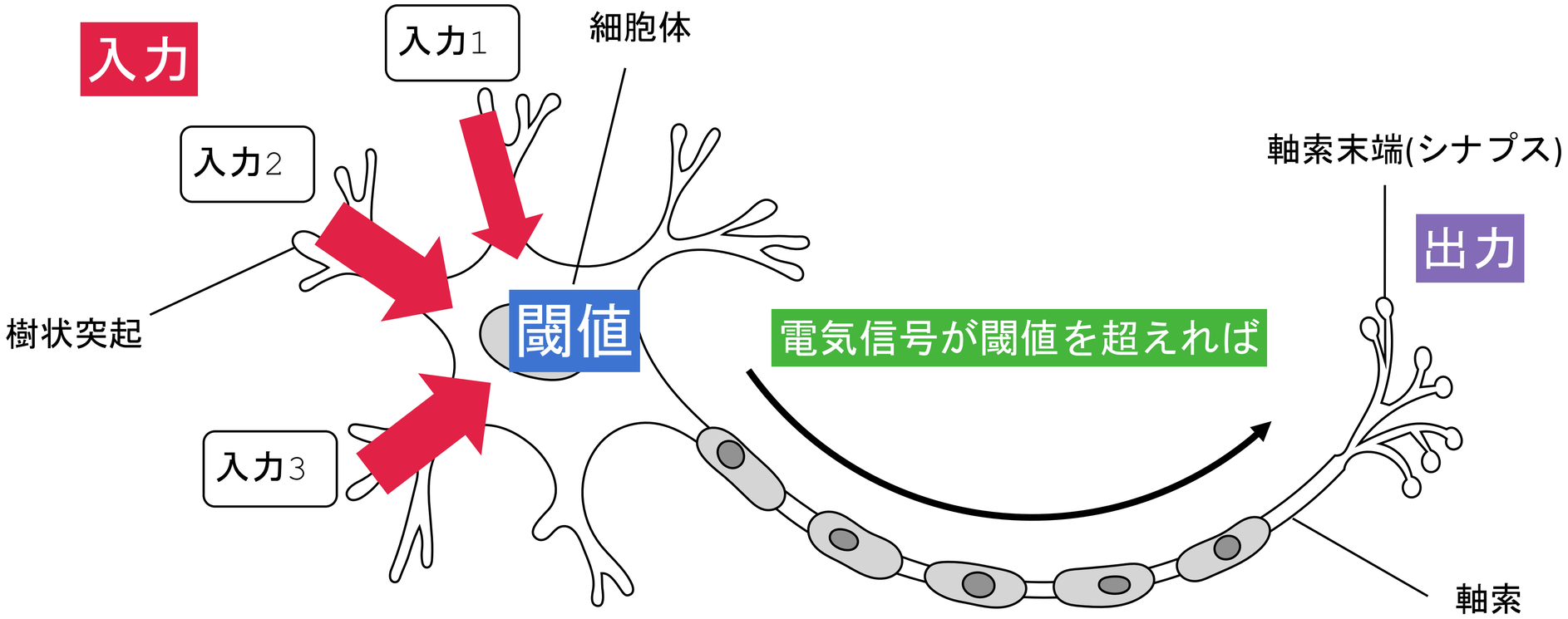
ニューラルネットワークとは

- ニューロンは、樹状突起、細胞体、軸索からなる
- ニューロンは、樹状突起から入力された電気信号が神経細胞内の電位を超えるかどうかの閾値を持っている
- 閾値を超えるとニューロンは興奮状態となり、軸索末端から電気信号が出力される



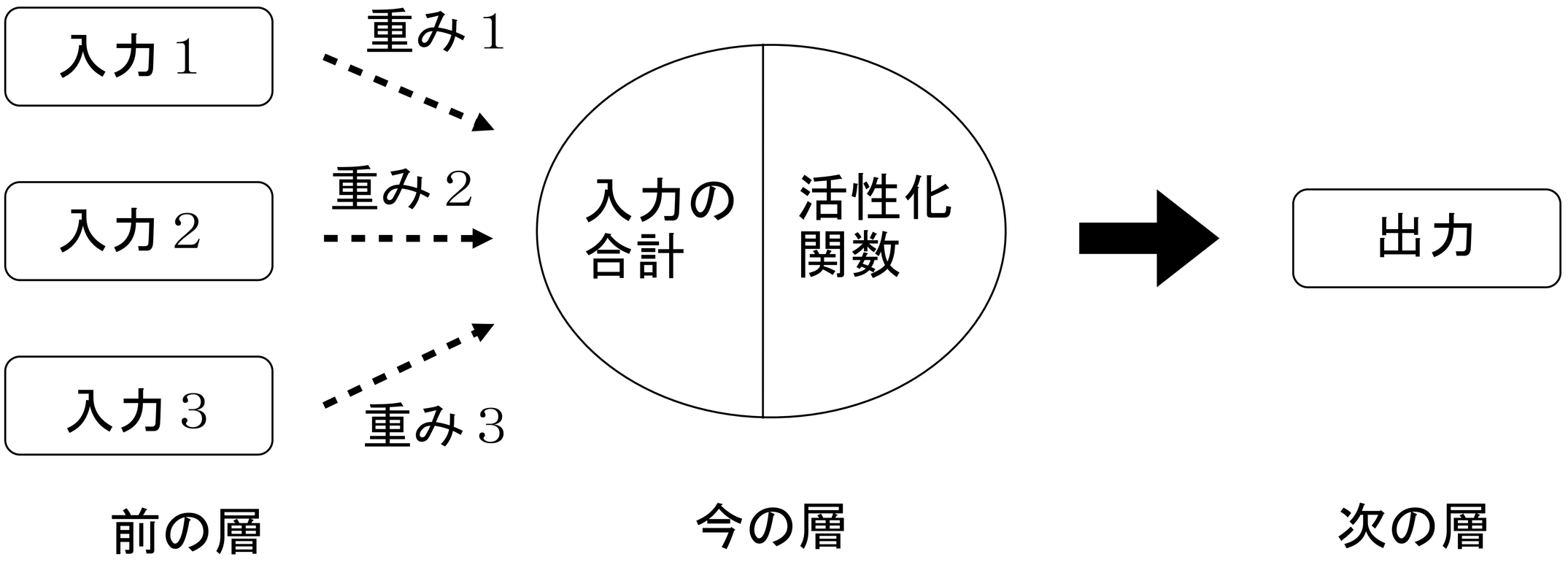
ニューラルネットワークとは

- ニューロンは、樹状突起、細胞体、軸索からなる
- ニューロンは、樹状突起から入力された電気信号が神経細胞内の電位を超えるかどうかの閾値を持っている
- 閾値を超えるとニューロンは興奮状態となり、軸索末端から電気信号が出力される



ニューラルネットワークとは

単一の人工ニューロンはこのようなモデルで表すことができる。



アヤメのデータでの深層学習の実践

あやめのデータ

Iris Versicolor
ブルーフラッグ

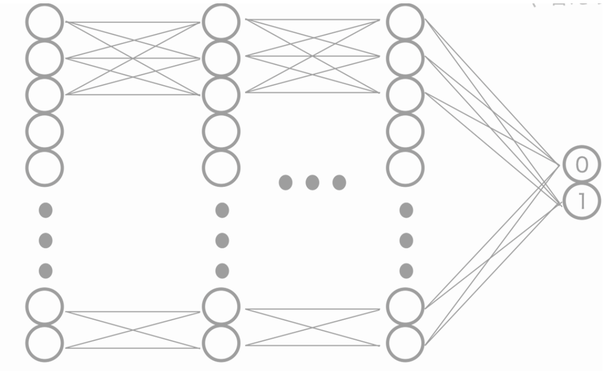
Iris Setosa
ヒオウギアヤメ

変数4つ

がく片の長さ
がく片の幅
花びらの長さ
花びらの幅

あやめのデータは100個、正解が2種類

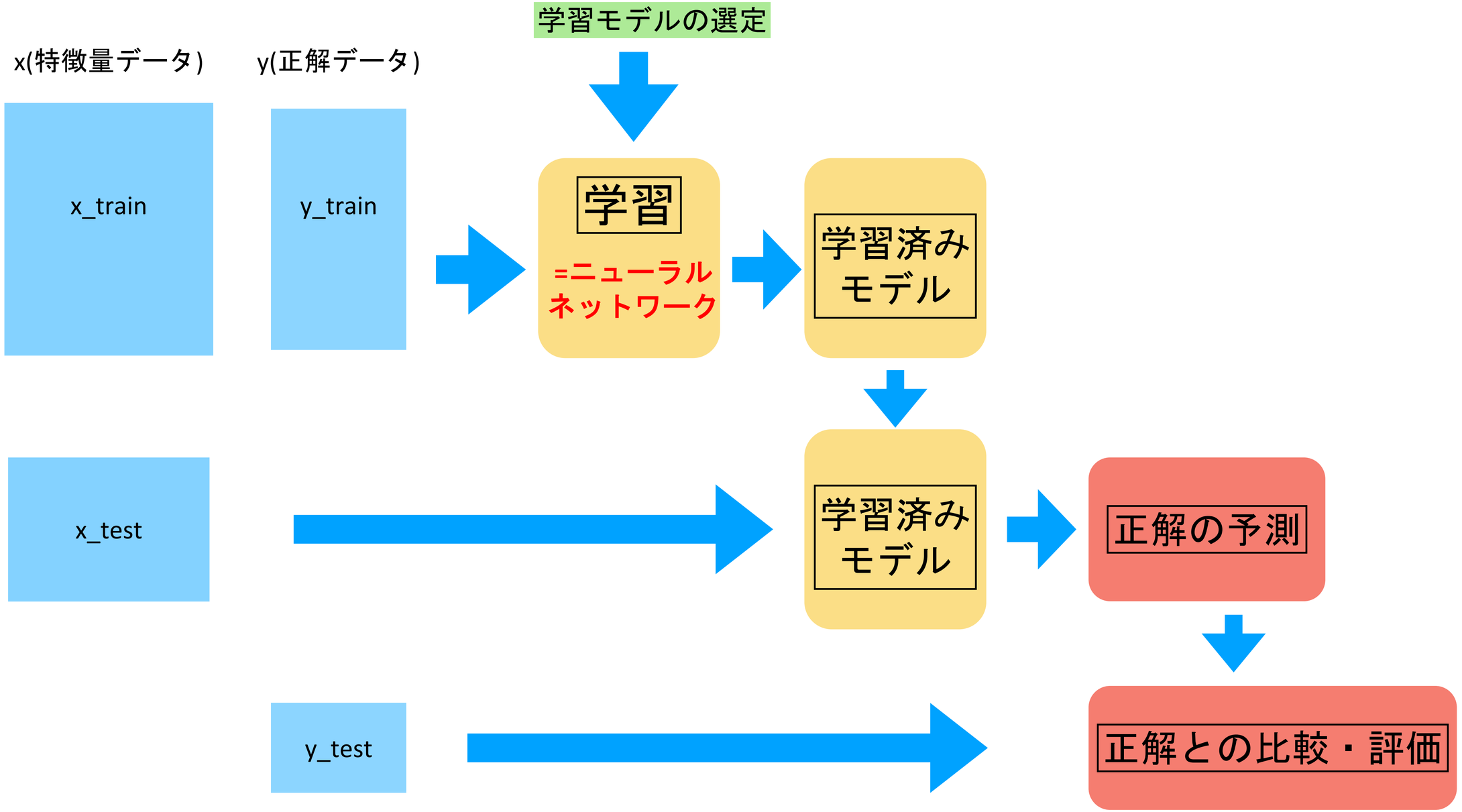
ニューラルネットワーク



分類

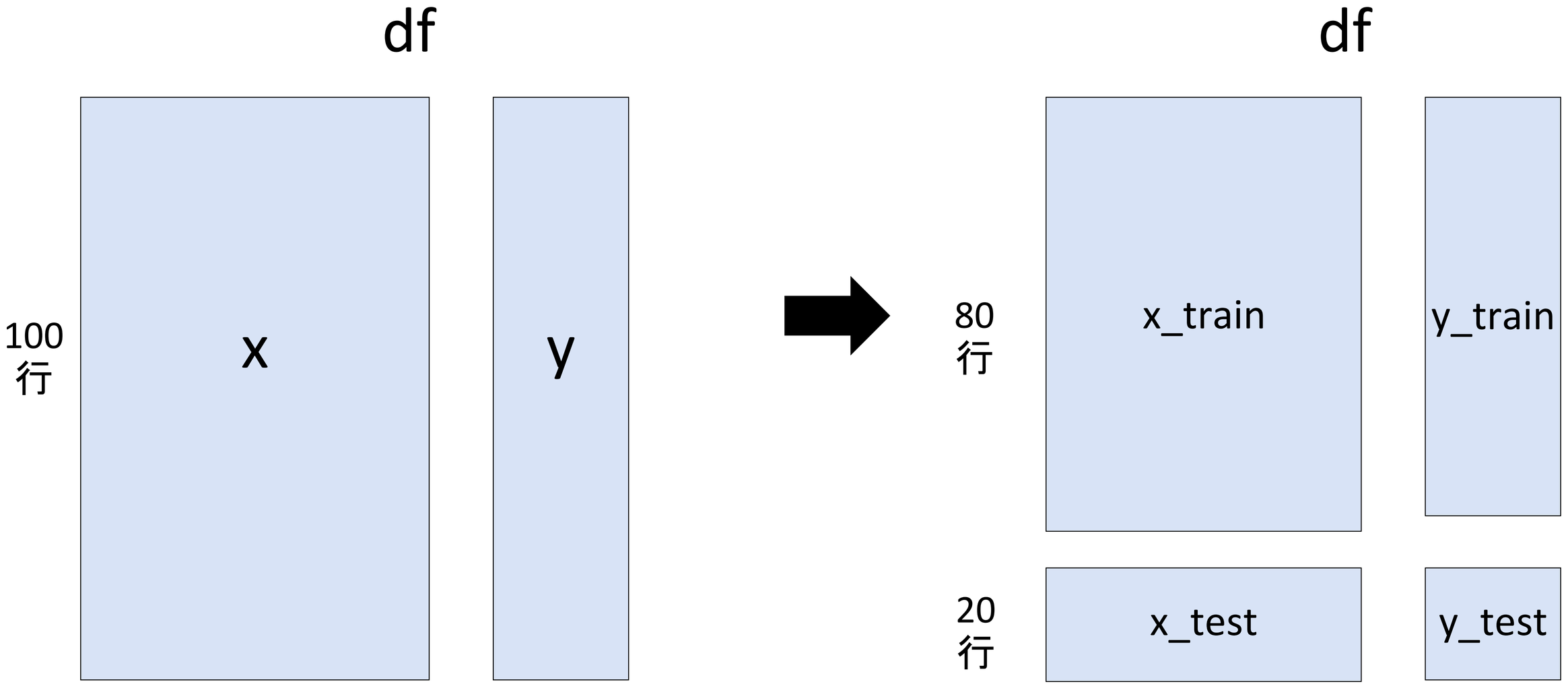
ヒオウギアヤメorブルーフラッグ

深層学習(機械学習)の一般的な流れ



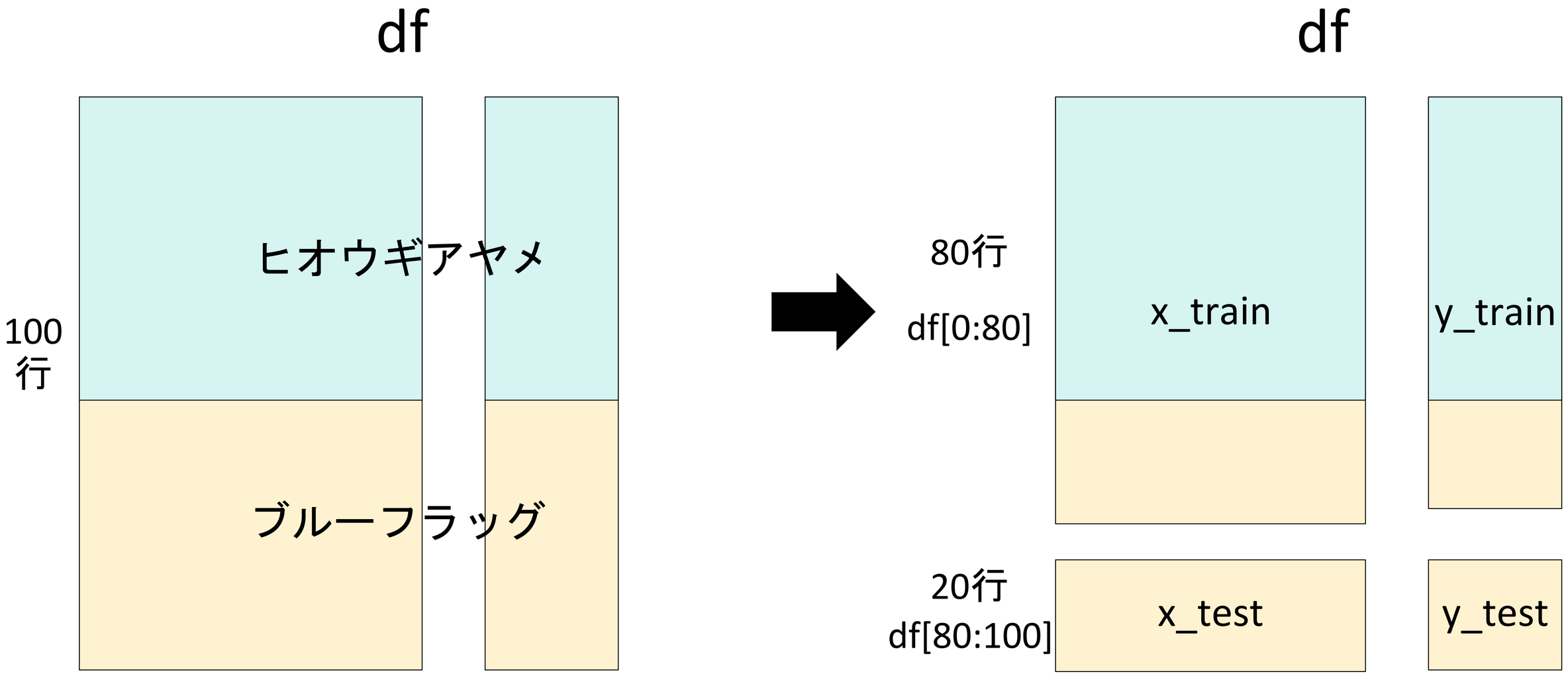
データを学習用とテスト用に分ける

df の80個を学習用のデータ、20個をテスト用のデータに分けたい
(慣例的に7~8割を学習用に用いるのが一般的です)



データを学習用とテスト用に分ける

df : 100行のヒオウギアヤメとブルーフラッグのデータ
　　1~50行目がヒオウギアヤメ、51~100行目がブルーフラッグ
→そのままではテスト用データはすべてブルーフラッグ



データをランダムにシャッフルしてから学習用とテスト用に分ける

rand = df.sample(n=100,random_state=1)

データフレーム.sample(n=取り出したい数,random_state=数字)
でデータフレームからランダムにn個取り出す

df - DataFrame

Index	がく片の長さ	がく片の幅	花びらの長さ	花びらの幅	アヤメの種類_数字	アヤメの種類
0	5.1	3.5	1.4	0.2	0	ヒオウギアヤメ
1	4.9	3	1.4	0.2	0	ヒオウギアヤメ
2	4.7	3.2	1.3	0.2	0	ヒオウギアヤメ
3	4.6	3.1	1.5	0.2	0	ヒオウギアヤメ
4	5	3.6	1.4	0.2	0	ヒオウギアヤメ
5	5.4	3.9	1.7	0.4	0	ヒオウギアヤメ
6	4.6	3.4	1.4	0.3	0	ヒオウギアヤメ
7	5	3.4	1.5	0.2	0	ヒオウギアヤメ
8	4.4	2.9	1.4	0.2	0	ヒオウギアヤメ
9	4.9	3.1	1.5	0.1	0	ヒオウギアヤメ
10	5.4	3.7	1.5	0.2	0	ヒオウギアヤメ
11	4.8	3.4	1.6	0.2	0	ヒオウギアヤメ
12	4.8	3	1.4	0.1	0	ヒオウギアヤメ
13	4.3	3	1.1	0.1	0	ヒオウギアヤメ

Format Resize Background color Column min/max Save and Close Close

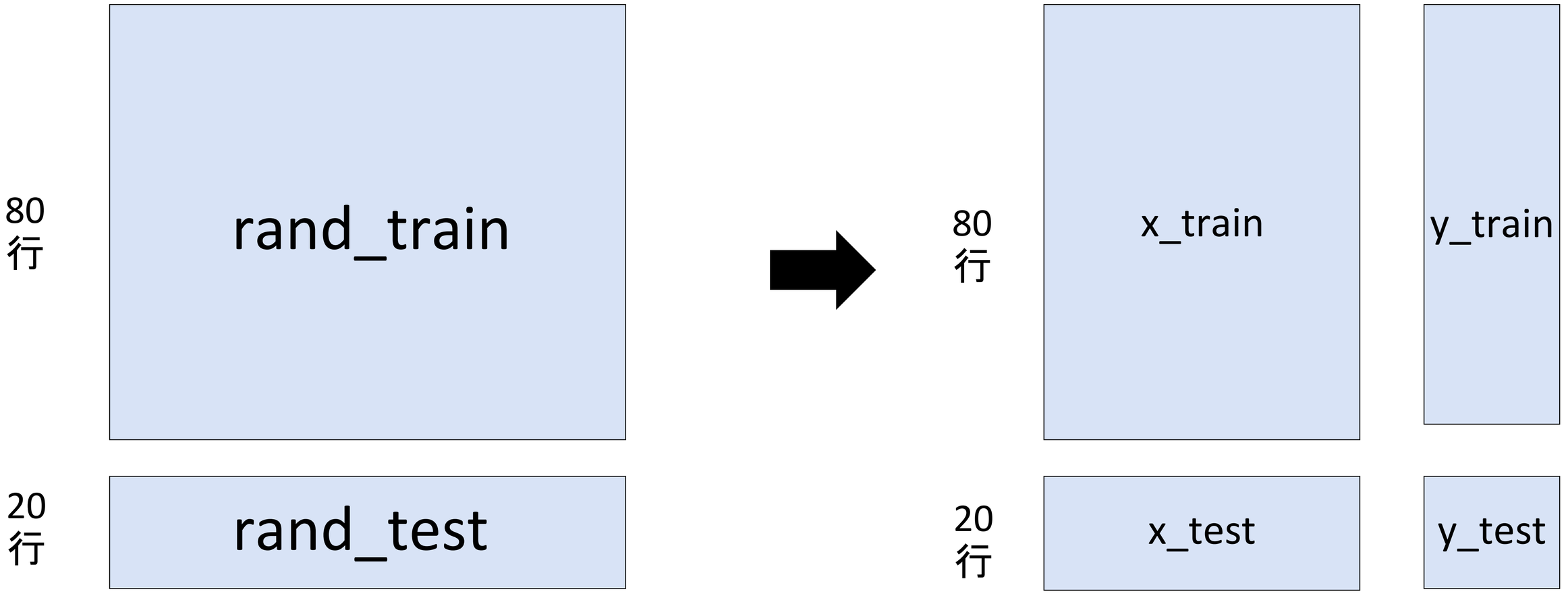
rand - DataFrame

Index	がく片の長さ	がく片の幅	花びらの長さ	花びらの幅	ヤメの種類_数字	アヤメの種類
80	5.5	2.4	3.8	1.1	1	ブルーフラッグ
84	5.4	3	4.5	1.5	1	ブルーフラッグ
33	5.5	4.2	1.4	0.2	0	ヒオウギアヤメ
81	5.5	2.4	3.7	1	1	ブルーフラッグ
93	5	2.3	3.3	1	1	ブルーフラッグ
17	5.1	3.5	1.4	0.3	0	ヒオウギアヤメ
36	5.5	3.5	1.3	0.2	0	ヒオウギアヤメ
82	5.8	2.7	3.9	1.2	1	ブルーフラッグ
69	5.6	2.5	3.9	1.1	1	ブルーフラッグ
65	6.7	3.1	4.4	1.4	1	ブルーフラッグ
92	5.8	2.6	4	1.2	1	ブルーフラッグ
39	5.1	3.4	1.5	0.2	0	ヒオウギアヤメ
56	6.3	3.3	4.7	1.6	1	ブルーフラッグ

Format Resize Background color Column min/max Save and Close Close

データをランダムにシャッフルしてから学習用とテスト用に分ける

```
rand_train = rand[0:80]
rand_test = rand[80:100]
x_train = rand_train[['がく片の長さ','がく片の幅','花びらの長さ','花びらの幅']]
y_train = rand_train['アヤメの種類_数字']
x_test = rand_test[['がく片の長さ','がく片の幅','花びらの長さ','花びらの幅']]
y_test = rand_test['アヤメの種類_数字']
```



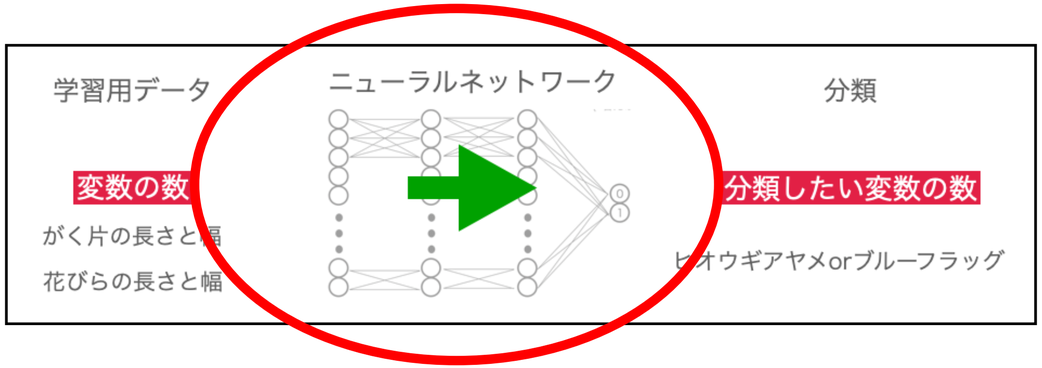
データをランダムにシャッフルしてから学習用とテスト用に分ける

```
rand_train = rand[0:80]
rand_test = rand[80:100]
x_train = rand_train[['がく片の長さ','がく片の幅','花びらの長さ','花びらの幅']]
y_train = rand_train['アヤメの種類_数字']
x_test = rand_test[['がく片の長さ','がく片の幅','花びらの長さ','花びらの幅']]
y_test = rand_test['アヤメの種類_数字']
```

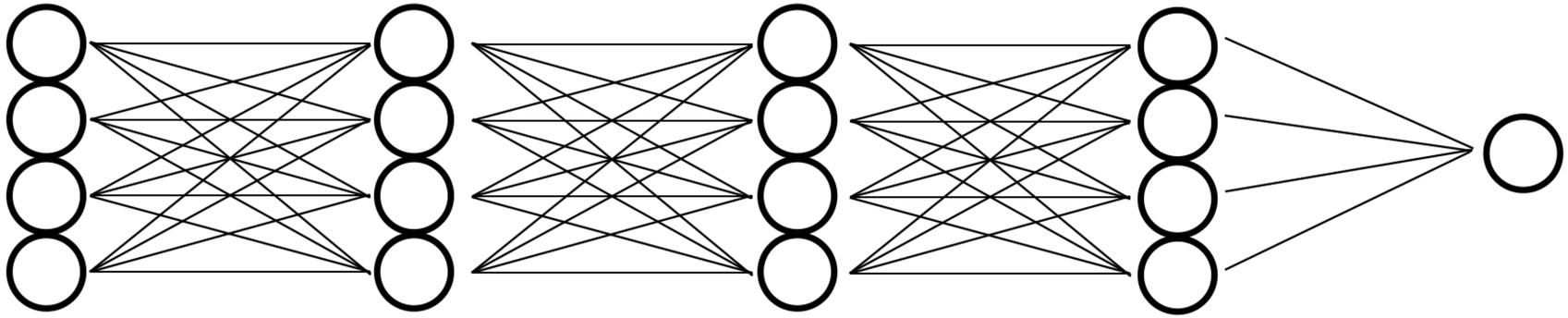
```
x_train = x_train.to_numpy()
y_train = y_train.to_numpy()
x_test = x_test.to_numpy()
y_test = y_test.to_numpy()
```

Numpy配列に置き換える

学習モデルの作成(ニューラルネットワーク)



今回作成する学習モデル



入力層)
ニューロン4つ

１層目
ニューロン4つ

２層目
ニューロン4つ

３層目
ニューロン4つ

４層目(=出力層)
ニューロン1つ

今回はシンプルなネットワークにするため、出力層は1つにしてます。
(2値分類の場合、最後に出る値を確率pが出ると自動的にもう１つの確率は1-pになります。)

学習モデルの作成(ニューラルネットワーク)

ニューラルネットワークを作成していく

モデル名 = Sequential()

ニューラルネットワークを作るモデルも沢山ある中で、
今回はKeras(ケラス)のSequentialモデルを使用します

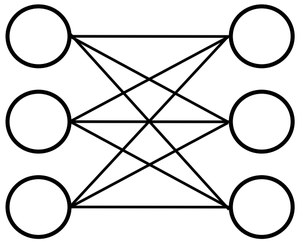
```
dl_model = Sequential()
dl_model.add(Dense(4, activation='relu', input_shape=(4,)))
dl_model.add(Dense(4, activation='relu'))
dl_model.add(Dense(4, activation='relu'))
dl_model.add(Dense(1, activation='sigmoid'))
dl_model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=["accuracy"])
dl_model.summary()
```

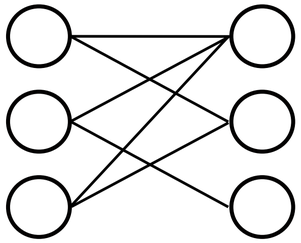
今回はモデル名を”dl_model”とする

学習モデルの作成(ニューラルネットワーク)

モデル名.add()で層の追加を行う

Denseは全ての入力が全てのニューロンと結合している状態(=全結合層という)

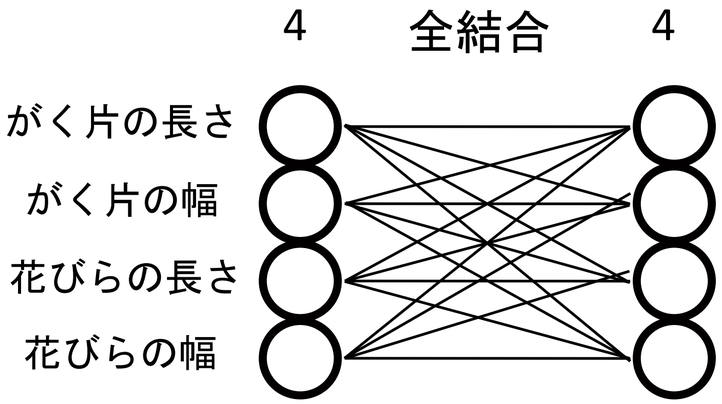
全結合層 

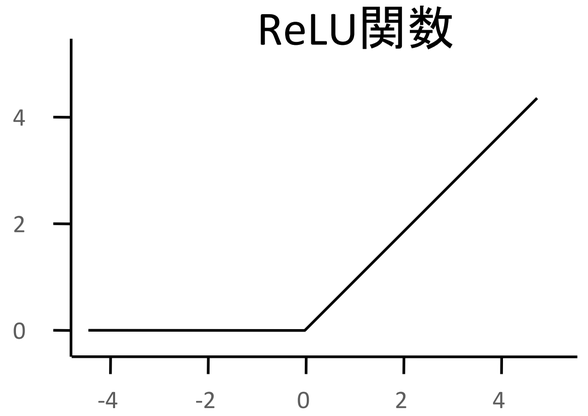
非全結合層 

```
dl_model = Sequential()
dl_model.add(Dense(4, activation='relu', input_shape=(4,)))
dl_model.add(Dense(4, activation='relu'))
dl_model.add(Dense(4, activation='relu'))
dl_model.add(Dense(1, activation='sigmoid'))
dl_model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=["accuracy"])
dl_model.summary()
```

学習モデルの作成(ニューラルネットワーク)

model.add(Dense(出力の変数の数, activation='活性化関数', input_shape=(入力の変数の数,)))



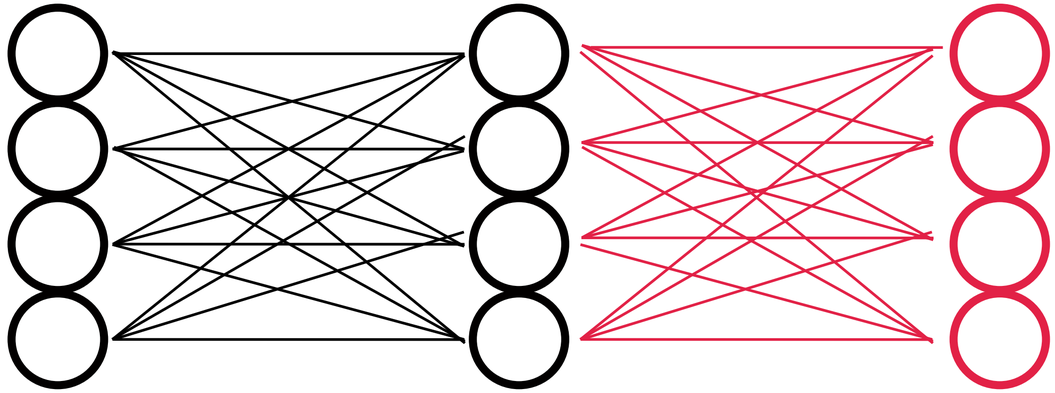


入力の変数4つ、出力の変数4つ
活性化関数はReLUを選択(全結合層)

(入力の4つの変数は、がく片の
長さと幅、花びらの長さと幅、
出力層の数は自由に決められます)

```
dl_model = Sequential()
dl_model.add(Dense(4, activation='relu', input_shape=(4,)))
dl_model.add(Dense(4, activation='relu'))
dl_model.add(Dense(4, activation='relu'))
dl_model.add(Dense(1, activation='sigmoid'))
dl_model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=["accuracy"])
dl_model.summary()
```

学習モデルの作成(ニューラルネットワーク)

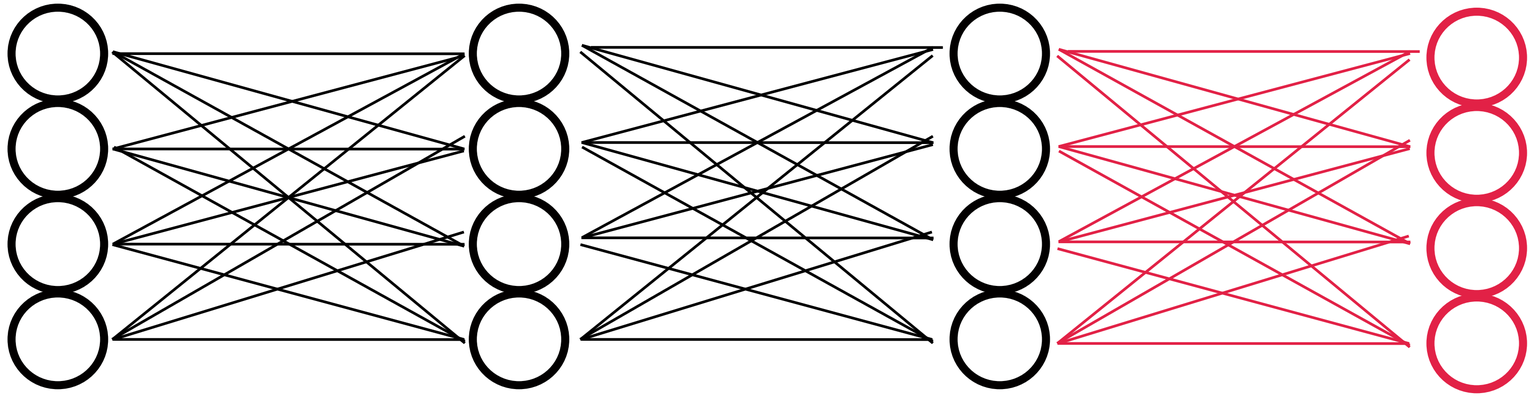


model.add()でさらに層の追加

2回目以降はinput_shapeは不要
入力の変数はそのまま(4つ)、出力の変数4つ
ReLUという(活性化)関数を選択

```
dl_model = Sequential()
dl_model.add(Dense(4, activation='relu', input_shape=(4,)))
dl_model.add(Dense(4, activation='relu'))
dl_model.add(Dense(4, activation='relu'))
dl_model.add(Dense(1, activation='sigmoid'))
dl_model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=["accuracy"])
dl_model.summary()
```

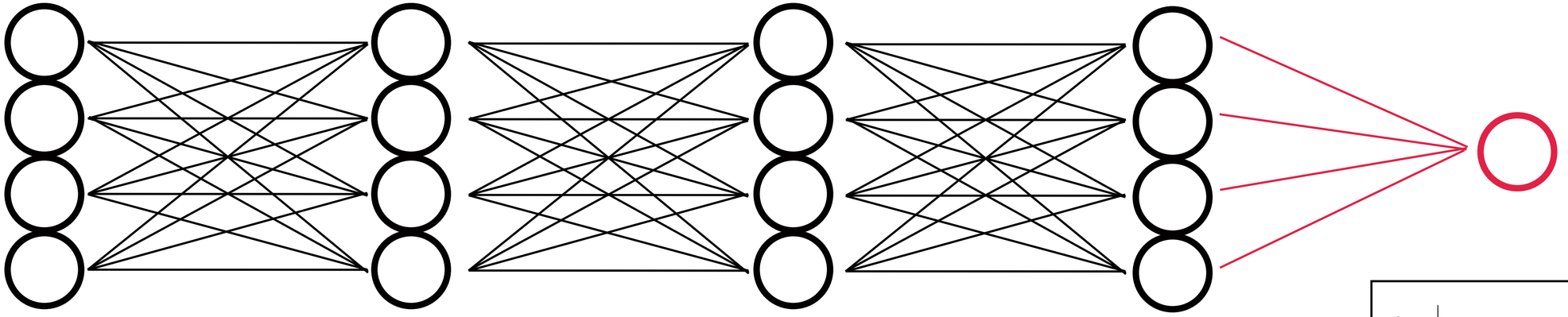
学習モデルの作成(ニューラルネットワーク)

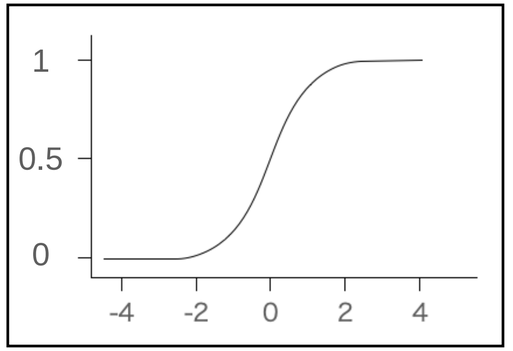


入力の変数はそのまま(4つ)、出力の変数4つ
ReLUという(活性化)関数を選択

```
dl_model = Sequential()
dl_model.add(Dense(4, activation='relu', input_shape=(4,)))
dl_model.add(Dense(4, activation='relu'))
dl_model.add(Dense(4, activation='relu'))
dl_model.add(Dense(1, activation='sigmoid'))
dl_model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=["accuracy"])
dl_model.summary()
```

学習モデルの作成(ニューラルネットワーク)





出力の変数1つ
シグモイド(活性化)関数を選択

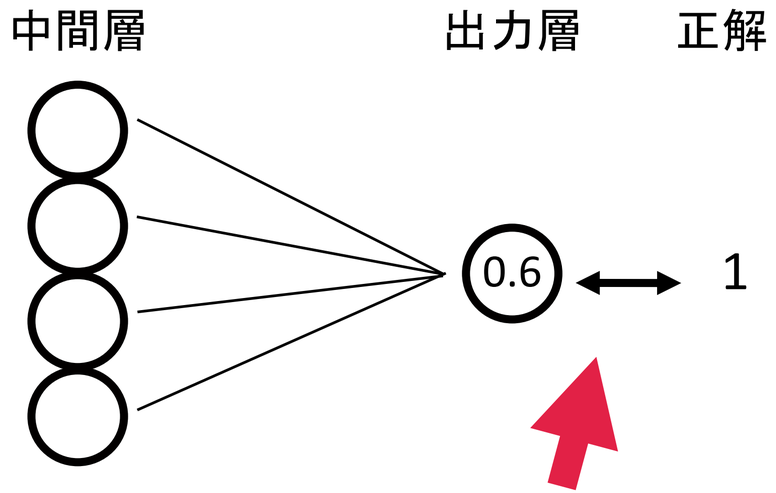
```
dl_model = Sequential()
dl_model.add(Dense(4, activation='relu', input_shape=(4,)))
dl_model.add(Dense(4, activation='relu'))
dl_model.add(Dense(4, activation='relu'))
dl_model.add(Dense(1, activation='sigmoid'))
dl_model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=["accuracy"])
dl_model.summary()
```

学習モデルの作成(ニューラルネットワーク)

model.compile()で、学習時の評価方法の選択をする

loss = “損失関数”, optimizer =‘最適化関数’, metrics=[“評価指標”]

今回は2クラス分類なのでbinary_crossentropyを選択
評価指標は正解率を示すaccuracyを選択



この誤差を0に近づけるように、
誤差逆伝播を行い各パラメーターを調整

(今回はAdamという最適化アルゴリズムを使用)

```
dl_model = Sequential()
dl_model.add(Dense(4, activation='relu', input_shape=(4,)))
dl_model.add(Dense(4, activation='relu'))
dl_model.add(Dense(4, activation='relu'))
dl_model.add(Dense(1, activation='sigmoid'))
dl_model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=["accuracy"])
dl_model.summary()
```

学習モデルの作成(ニューラルネットワーク)

dl_model.summary()は、
作成した学習モデルを要約する

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 4)                 20
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 20
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 20
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 5
=================================================================
Total params: 65
Trainable params: 65
Non-trainable params: 0
_________________________________________________________________
```

```
dl_model = Sequential()
dl_model.add(Dense(4, activation='relu', input_shape=(4,)))
dl_model.add(Dense(4, activation='relu'))
dl_model.add(Dense(4, activation='relu'))
dl_model.add(Dense(1, activation='sigmoid'))
dl_model.compile(loss='binary_crossentropy', optimizer='Adam', metrics=["accuracy"])
dl_model.summary()
```

学習用データでの学習

```
result = dl_model.fit(x_train, y_train, epochs=300)
```

resultに学習過程を記録する。 300回学習させる

epochs

1エポックは1試行のことで、学習用データを１通り使って１エポックと数える
ここでは300回学習させている。

学習用データでの学習

```
result = dl_model.fit(x_train, y_train, epochs=300)
```

resultに学習過程を記録する。 300回学習させる

epochs

1エポックは1試行のことで、学習用データを１通り使って１エポックと数える
ここでは300回学習させている。

```
80/80 [==============================] - 0s 76us/sample - loss: 0.0231 - accuracy: 1.0000
Epoch 290/300
80/80 [==============================] - 0s 69us/sample - loss: 0.0228 - accuracy: 1.0000
Epoch 291/300
80/80 [==============================] - 0s 66us/sample - loss: 0.0225 - accuracy: 1.0000
Epoch 292/300
80/80 [==============================] - 0s 66us/sample - loss: 0.0222 - accuracy: 1.0000
Epoch 293/300
80/80 [==============================] - 0s 76us/sample - loss: 0.0219 - accuracy: 1.0000
Epoch 294/300
80/80 [==============================] - 0s 66us/sample - loss: 0.0216 - accuracy: 1.0000
Epoch 295/300
80/80 [==============================] - 0s 60us/sample - loss: 0.0214 - accuracy: 1.0000
Epoch 296/300
80/80 [==============================] - 0s 64us/sample - loss: 0.0211 - accuracy: 1.0000
Epoch 297/300
80/80 [==============================] - 0s 66us/sample - loss: 0.0208 - accuracy: 1.0000
Epoch 298/300
80/80 [==============================] - 0s 74us/sample - loss: 0.0206 - accuracy: 1.0000
Epoch 299/300
80/80 [==============================] - 0s 82us/sample - loss: 0.0203 - accuracy: 1.0000
Epoch 300/300
80/80 [==============================] - 0s 90us/sample - loss: 0.0201 - accuracy: 1.0000
```

実行するとコンソール画面に学習過程が出力される
loss: 学習用データの誤差、0に近いほどよい
acc: 学習用データの正解率、1に近いほどよい

テスト用データでの検証

```
score = dl_model.evaluate(x_test, y_test)
print("testの誤差は：",score[0],"testの正解率は:",score[1])
```

(変数名) = dl_model.evaluate(x_test, y_test)
→誤差と正解率が算出される

誤差→(変数名)[0]、正解率→(変数名)[1]

```
print("Test loss:", score[0])
Test loss:  0.009264961816370487

print("Test accuracy:", score[1])
Test accuracy: 1.0
```

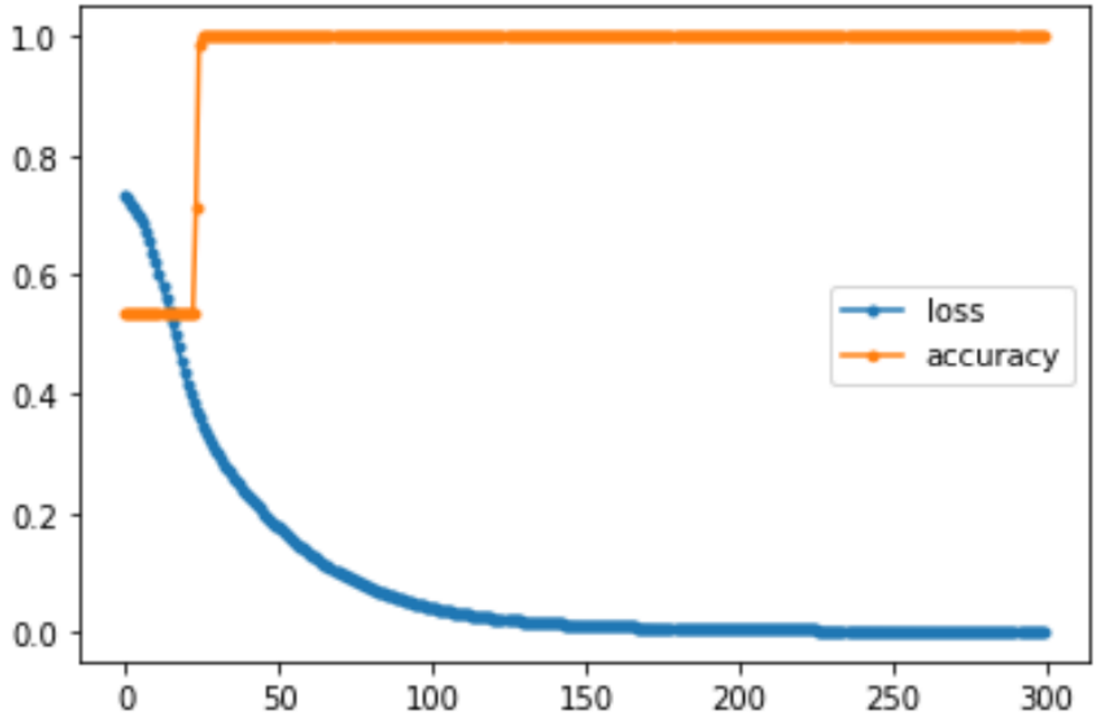
誤差0.9%、正解率100%という結果

結果の可視化

```
# 学習結果の図示
import matplotlib.pyplot as plt

loss = result.history['loss']
acc = result.history['accuracy']
epochs = list(range(0,300))
plt.plot(epochs,loss,marker='.',label='loss')
plt.plot(epochs,acc,marker='.',label='accuracy')
plt.legend()
plt.show()
```

学習する度に正解率が上がって誤差が下がっていることが分かる

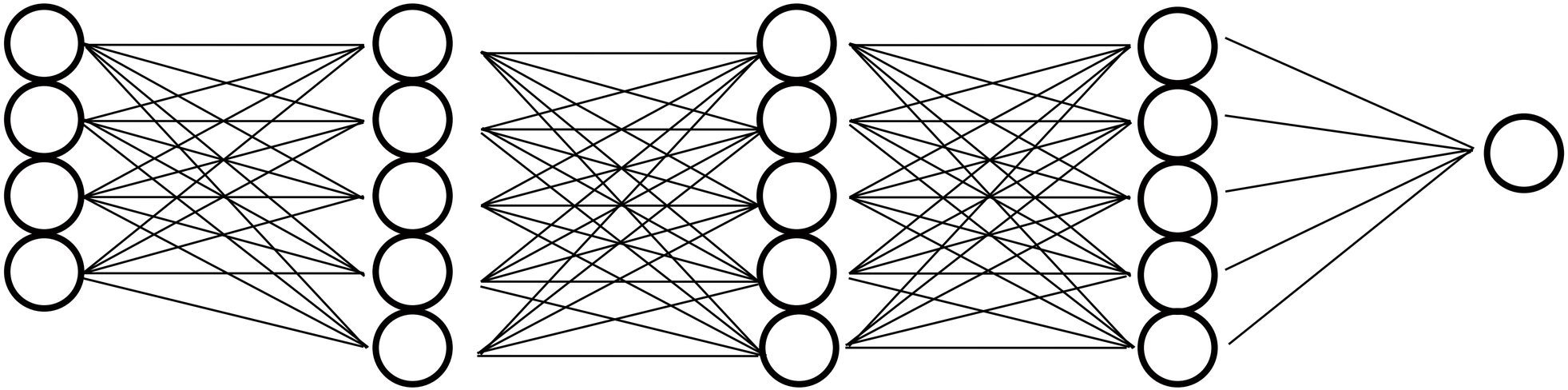


result = model.fit(x_train, y_train, epochs=300)

kerasのfit()は各エポック毎の正解率、誤差を保存出来る
result.history[“accuracy”]が正解率のリスト
result.history[[“loss”]が誤差のリスト

自分たちで実践してみよう

1層目から3層目のニューロンを1つ増やして、
学習の回数を500回にして実践してみよう



課題

・中間層の２層目をニューロン6個、３層目をニューロン3個にして実行しなさい
（モデル名).summary()の出力結果と図を提出して下さい。（エポック数300）

・(余力がある人は)
上のモデルを使って新たなアヤメのデータでブルーフラッグである確率を算出しなさい

No.	がく片の長さ	がく片の幅	花びらの長さ	花びらの幅
1	5.3	2.4	4.8	1.5
2	4.4	2.5	1.8	0.4

ヒント: import numpy as np
(変数) = model.predict((調べたいNumpy配列))
print(変数) = [[(1つ目の確率)]、[(2つ目の確率)],[…] , […], …]

以上で第2回は終了になります。
第3回は肺のレントゲン画像を用いた深層学習を実施してみたいと思います。