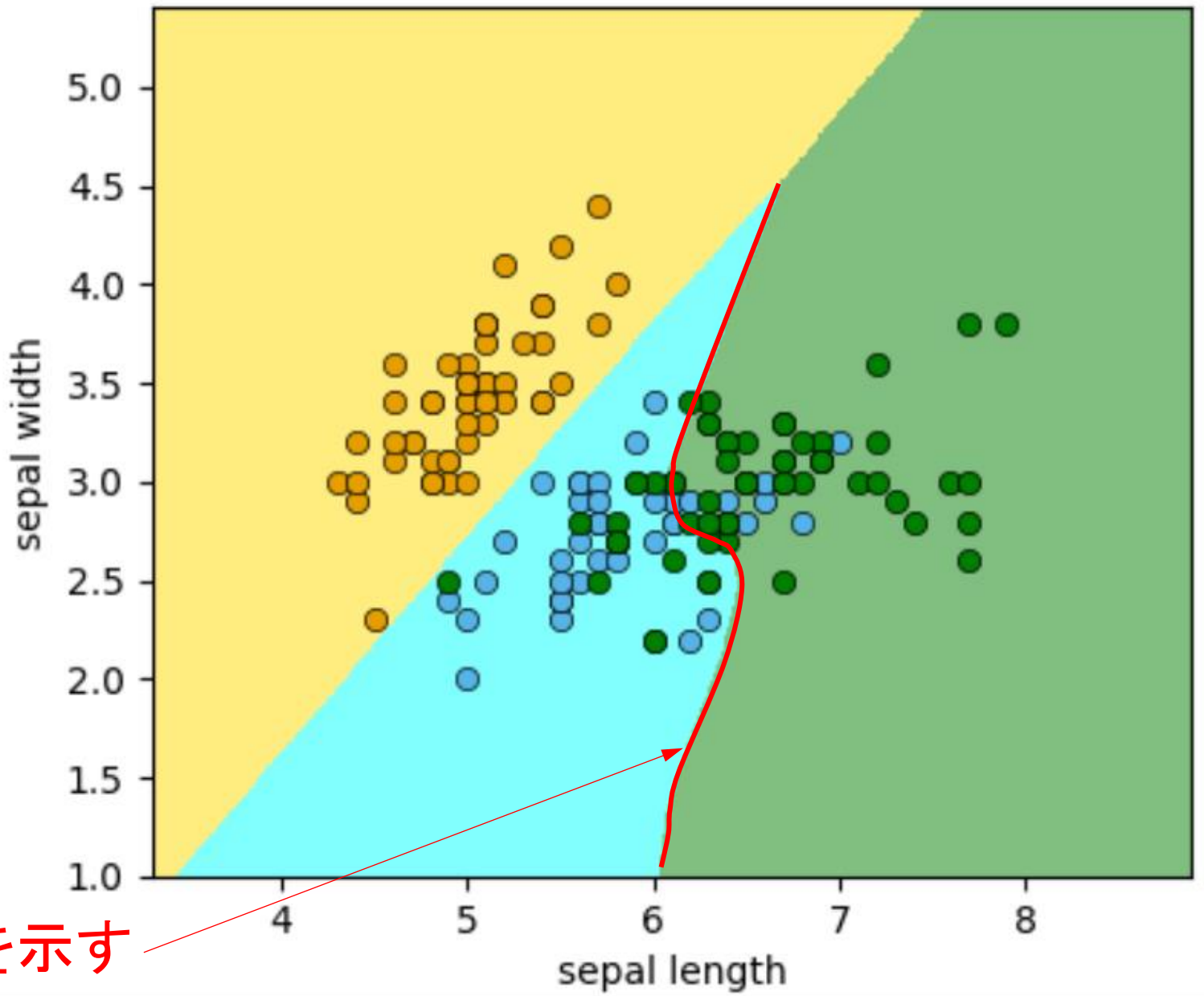
実行結果1

ニューラルネットワークを用いた分類

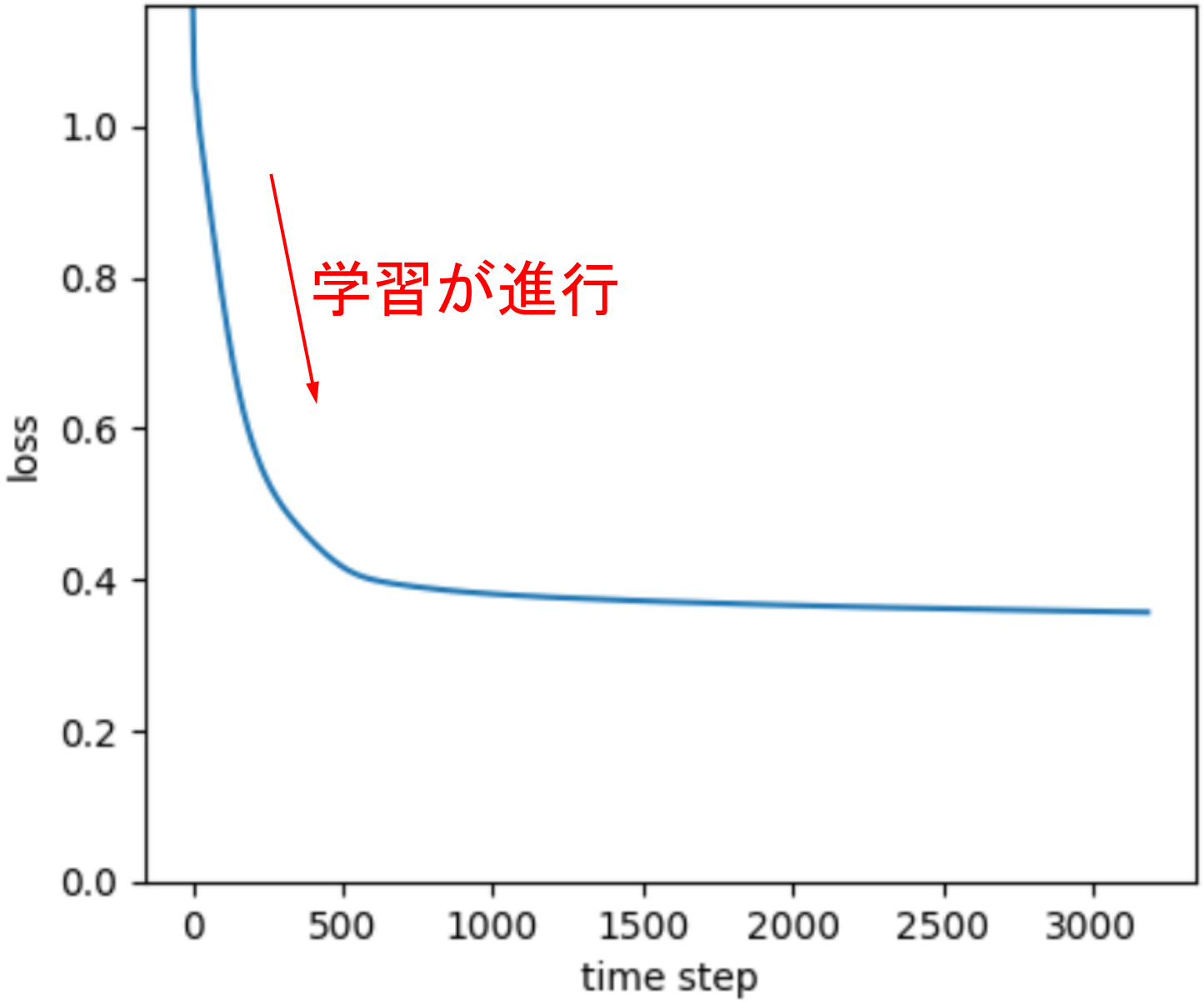
特徴2、種類（クラス）3



超平面での分類を示す

実行結果2

損失関数



分析結果とターゲット(正解)が一致すると0となる

シナプス強度を調節して
損失関数0になるようにする

勾配降下法、最急効果法

←収束の確認は重要

```
1  # -*- coding: utf-8 -*-
2  from sklearn import datasets
3  from sklearn.neural_network import MLPClassifier
4  import numpy as np
5  import matplotlib.pyplot as plt
6  from matplotlib.colors import ListedColormap
7
8  # アヤメのデータをロードし、変数irisに格納
9  iris = datasets.load_iris()
10
11 # 特徴量のセットを変数Xに、ターゲットを変数yに格納
12 X = iris.data
13 y = iris.target
14
15 # 特徴量を外花被片の長さ(sepal length)と幅(sepal width)の
16 # 2つのみに制限(2次元で考えるため)
17 X = X[:, :2]
18
19 # 分類用に多層ニューラルネットワークを用意
20 #ランダムな要素を固定した場合（毎回同じ結果）
21 clf = MLPClassifier(hidden_layer_sizes=(100, ), max_iter=10000, tol=0.00001, random_state=1)
22 # 毎回ランダムな場合
23 #clf = MLPClassifier(hidden_layer_sizes=(100, ), max_iter=10000, tol=0.00001, random_state=None)
24
25 # ニューラルネットワークの学習
26 print('学習中…')
27 clf.fit(X, y)
28
```

ニューラルネットワークを実行

Hidden_layer_sizes=(100,)
100ニューロンからなる中
間層が1層　という意味

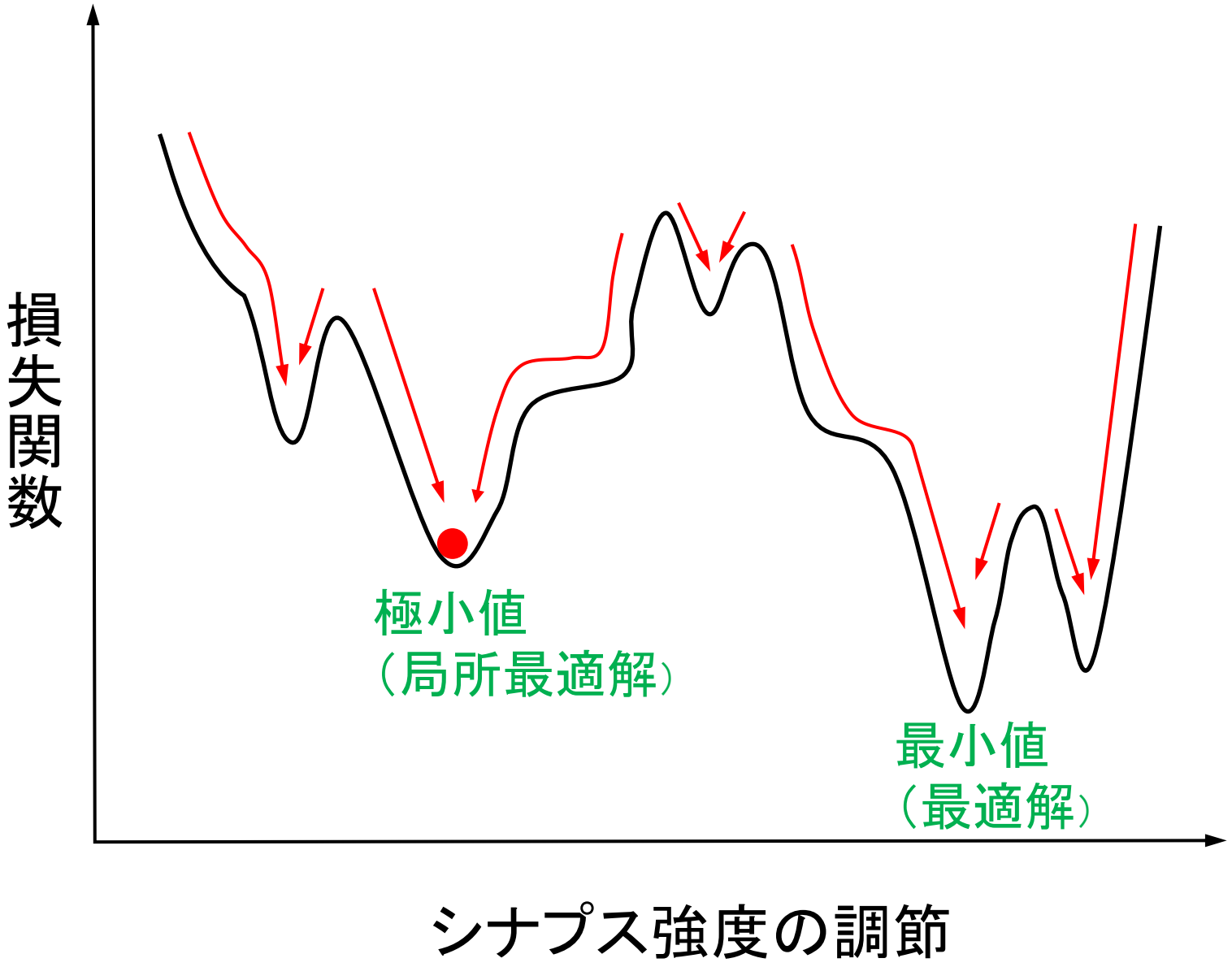
学習を継続する最大ステップ数

この値より3回減らなければ学習終了

学習開始前のシナプス強度を
ランダム乱数で定めている

clf:　MLPClassfier(Multi-layer Perceptron Classfiier)
多層パーセプトロン分類器

```
##### 分類結果を背景の色分けにより表示

# 外花被片の長さ(sepal length)と幅(sepal width)の
# 最小値と最大値からそれぞれ1ずつ広げた領域を
# グラフ表示エリアとする
x_min = min(X[:,0]) - 1
x_max = max(X[:,0]) + 1
y_min = min(X[:,1]) - 1
y_max = max(X[:,1]) + 1

# グラフ表示エリアを縦横400ずつのグリッドに区切る
# (分類クラスに応じて背景に色を塗るため)
XX, YY = np.mgrid[x_min:x_max:400j, y_min:y_max:400j]

# グリッドの点をscikit-learn用の入力に並べなおす
Xg = np.c_[XX.ravel(), YY.ravel()]

# 各グリッドの点が属するクラス(0～2)の予測をZに格納
Z = clf.predict(Xg)

# グリッド上に並べなおす
Z = Z.reshape(XX.shape)

# クラス0 (iris setosa) が薄オレンジ (1, 0.93, 0.5, 1)
# クラス1 (iris versicolor) が薄青 (0.5, 1, 1, 1)
# クラス2 (iris virginica) が薄緑 (0.5, 0.75, 0.5, 1)
cmap0 = ListedColormap([(0, 0, 0, 0), (1, 0.93, 0.5, 1)])
cmap1 = ListedColormap([(0, 0, 0, 0), (0.5, 1, 1, 1)])
cmap2 = ListedColormap([(0, 0, 0, 0), (0.5, 0.75, 0.5, 1)])

# グラフウインドウの大きさを横長に定める。
plt.subplots(figsize=(11.2,4.2))

# 縦に1枚、横に2枚のグラフの1枚目を設定
plt.subplot(121)

# 背景の色を表示
plt.pcolormesh(XX, YY, Z==0, cmap=cmap0)
plt.pcolormesh(XX, YY, Z==1, cmap=cmap1)
plt.pcolormesh(XX, YY, Z==2, cmap=cmap2)

# 軸ラベルを設定
plt.xlabel('sepal length')
plt.ylabel('sepal width')

##### ターゲットに応じた色付きでデータ点を表示

# iris setosa (y=0) のデータのみを取り出す
Xc0 = X[y==0]
# iris versicolor (y=1) のデータのみを取り出す
Xc1 = X[y==1]
# iris virginica (y=2) のデータのみを取り出す
Xc2 = X[y==2]

# iris setosa のデータXc0をプロット
plt.scatter(Xc0[:,0], Xc0[:,1], c='#E69F00', linewidths=0.5, edgecolors='black')
# iris versicolor のデータXc1をプロット
plt.scatter(Xc1[:,0], Xc1[:,1], c='#56B4E9', linewidths=0.5, edgecolors='black')
# iris virginica のデータXc2をプロット
plt.scatter(Xc2[:,0], Xc2[:,1], c='#008000', linewidths=0.5, edgecolors='black')

# 縦に1枚、横に2枚のグラフの2枚目を設定
plt.subplot(122)

# 損失関数のグラフの軸ラベルを設定
plt.xlabel('time step')
plt.ylabel('loss')

# グラフ縦軸の範囲を0以上と定める
plt.ylim(0, max(clf.loss_curve_))

# 損失関数の時間変化を描画
plt.plot(clf.loss_curve_)

# 描画したグラフを表示
plt.show()
```


損失関数
極小値
（局所最適解）
最小値
（最適解）
シナプス強度の調節

特徴4、種類(クラス)3

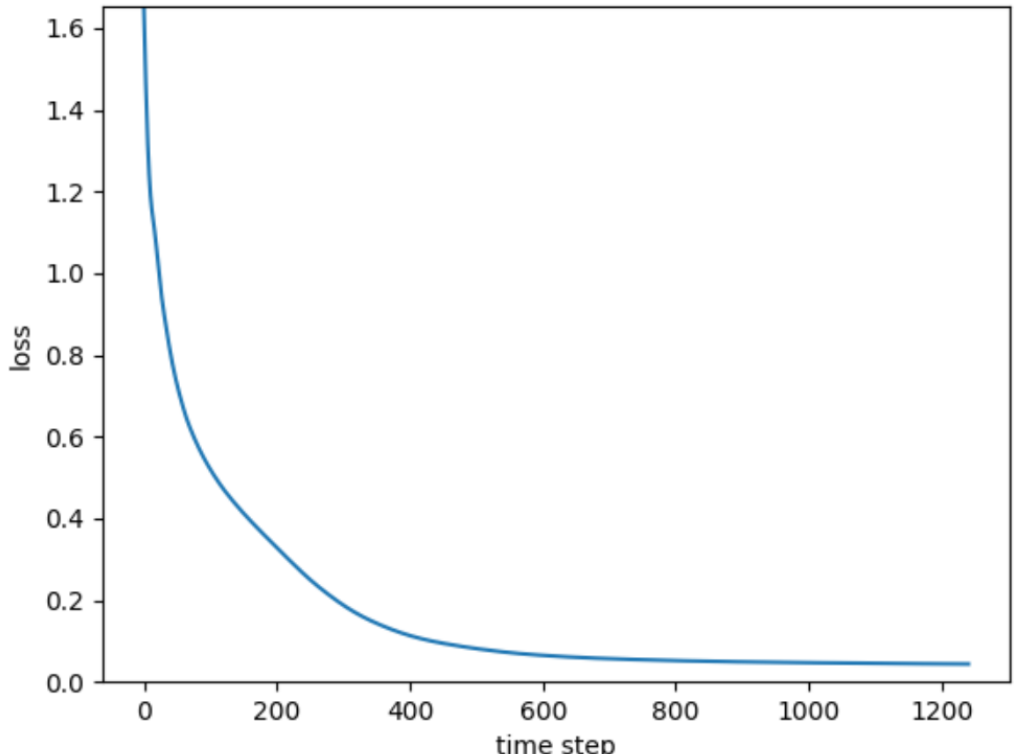
```
# -*- coding: utf-8 -*-
from sklearn import datasets
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt

# アヤメのデータをロードし、変数irisに格納
iris = datasets.load_iris()

# 特徴量のセットを変数Xに、ターゲットを変数yに格納
X = iris.data
y = iris.target

# 分類用に多層ニューラルネットワークを用意
# ランダムな要素を固定した場合（毎回同じ結果）
clf = MLPClassifier(hidden_layer_sizes=(100, ), max_iter=10000, tol=0.00001, random_state=1)
# 毎回ランダムな場合
#clf = MLPClassifier(hidden_layer_sizes=(100, ), max_iter=10000, tol=0.00001, random_state=None)

# ニューラルネットワークの学習
print('学習中…')
clf.fit(X, y)

# データを分類器に与え、予測を得る
result = clf.predict(X)

print('ターゲット')
print(y)
print('機械学習による予測')
print(result)

```

```
30
31 # データ数をtotalに格納
32 total = len(X)
33 # ターゲット（正解）と予測が一致した数をsuccessに格納
34 success = sum(result==y)
35
36 # 正解率をパーセント表示
37 print('正解率')
38 print(100.0*success/total)
39
40 # 損失関数のグラフの軸ラベルを設定
41 plt.xlabel('time step')
42 plt.ylabel('loss')
43
44 # グラフ縦軸の範囲を0以上と定める
45 plt.ylim(0, max(clf.loss_curve_))
46
47 # 損失関数の時間変化を描画
48 plt.plot(clf.loss_curve_)
49
50 # 描画したグラフを表示
51 plt.show()
```

```
学習中…
ターゲット
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
機械学習による予測
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
正解率
98.66666666666667
>>>
```

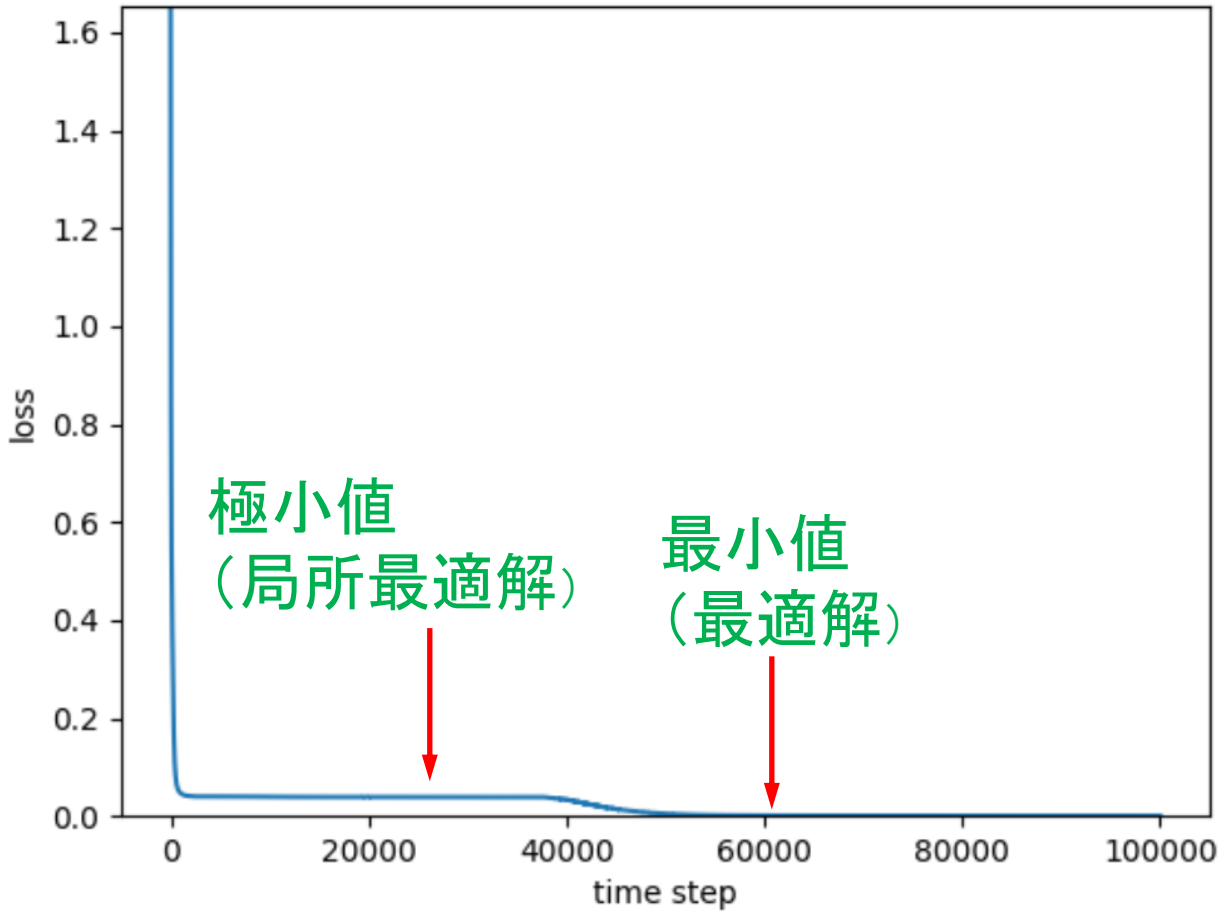
予測がターゲット（正解）と不一致

実行結果



10000 → 100000　　0.0001 → −1

```
13 # 分類用に多層ニューラルネットワークを用意
14 # ランダムな要素を固定した場合（毎回同じ結果）
15 clf = MLPClassifier(hidden_layer_sizes=(100, ), max_iter=100000, tol=-1, random_state=1)
16 # 毎回ランダムな場合
17 #clf = MLPClassifier(hidden_layer_sizes=(100, ), max_iter=10000, tol=0.00001, random_state=None)
```




```
学習中…

Warning (from warnings module):
  File "C:¥Users¥2004059¥AppData¥Local¥Programs¥Python¥Python39¥lib¥site-pac
s¥sklearn¥neural_network¥_multilayer_perceptron.py", line 614
    warnings.warn(
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (100000) reache
d the optimization hasn't converged yet.
ターゲット
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
機械学習による予測
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
正解率
100.0
>>>
```