ニューラルネットワークによる分類，未来予測

https://www.kkaneko.jp/ai/at/index.html

金子邦彦

BY NC SA

1. 教師あり学習の概要、Irisデータセットを用いた分類の実践を解説。
2. 時系列データの特性（例：Facebookイベント数の分析事例）。
3. リカレントニューラルネットワークの特徴、応用。
4. 太陽黒点数のデータを用いたLSTMによる未来予測の事例を解説。
5. 基本から実践的な演習、応用事例までを網羅。

アウトライン

１．教師有り学習

２．Irisデータセット

３．分類

４．時系列データ

５．リカレントニューラルネットワークと LSTM

６．未来予測

4-1 教師有り学習

教師あり学習

訓練データ (x, y) を使用.

(x, y) ・・・ **入力 x** に対する**正解 y**

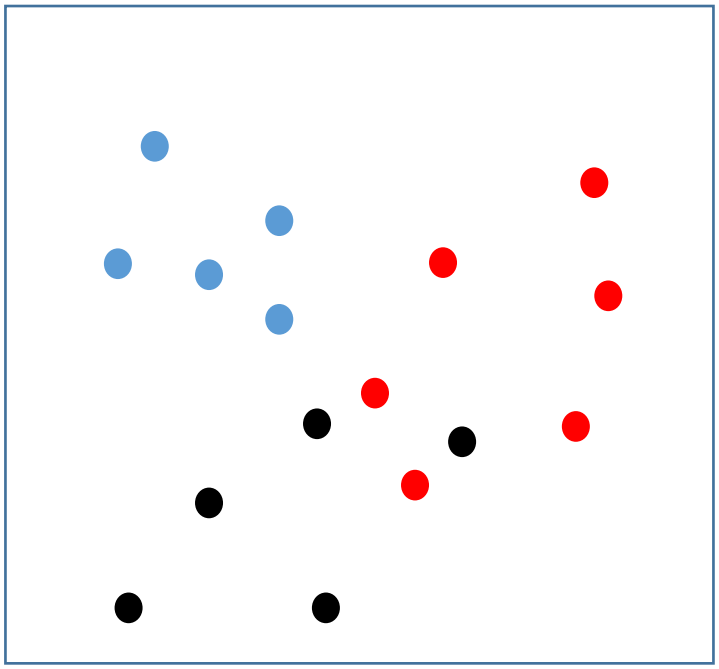
両方を学習に使用.

教師あり学習の主な用途

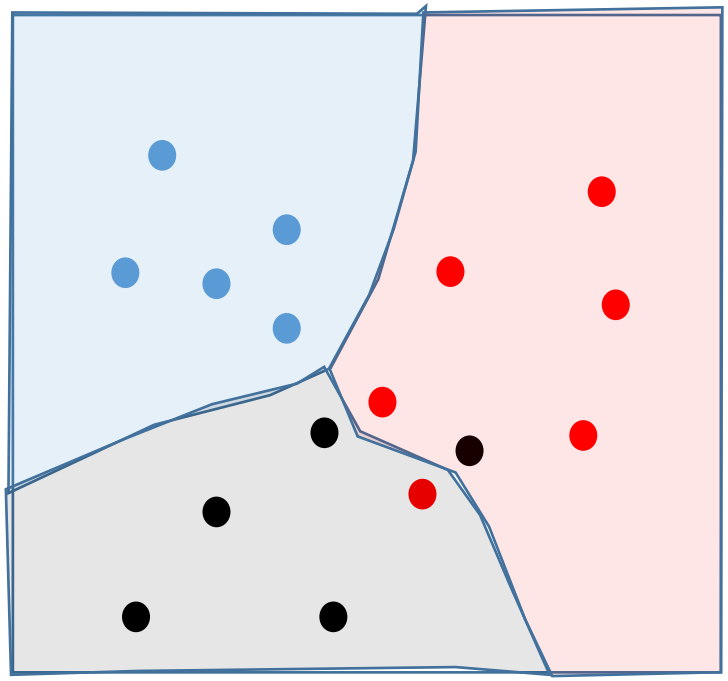
- **分類**：何種類かに分類すること
- **判別**：２種類への分類
- **回帰**：ある量から，別の量を予測するためのもの

分類

訓練データ



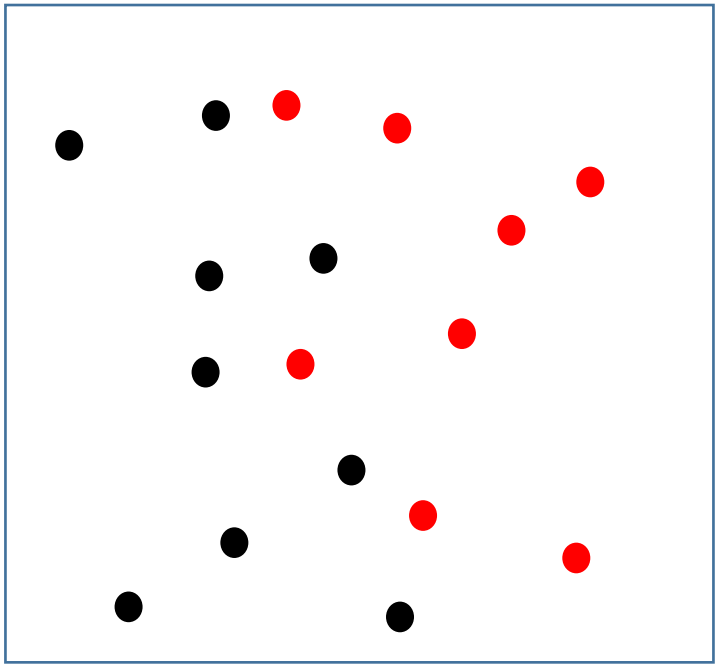
それぞれの範囲を得る



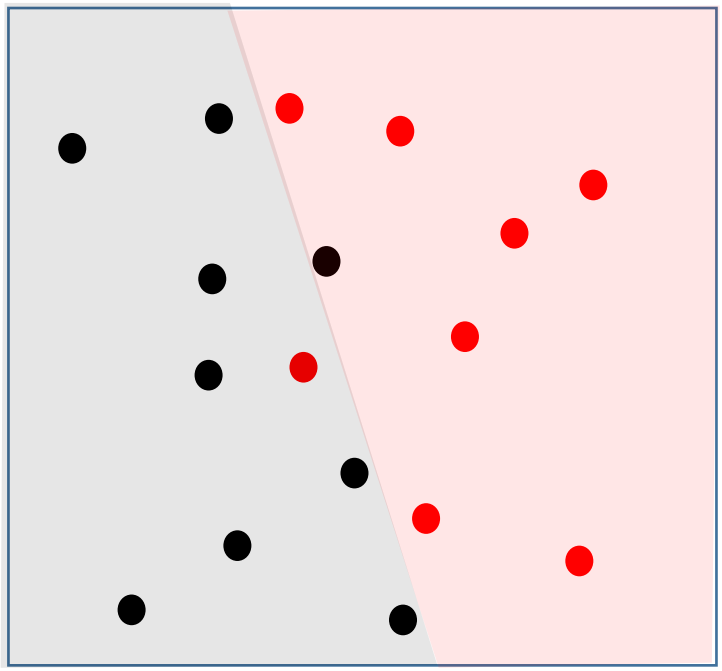
分類：何種類かに分類すること

判別

訓練データ



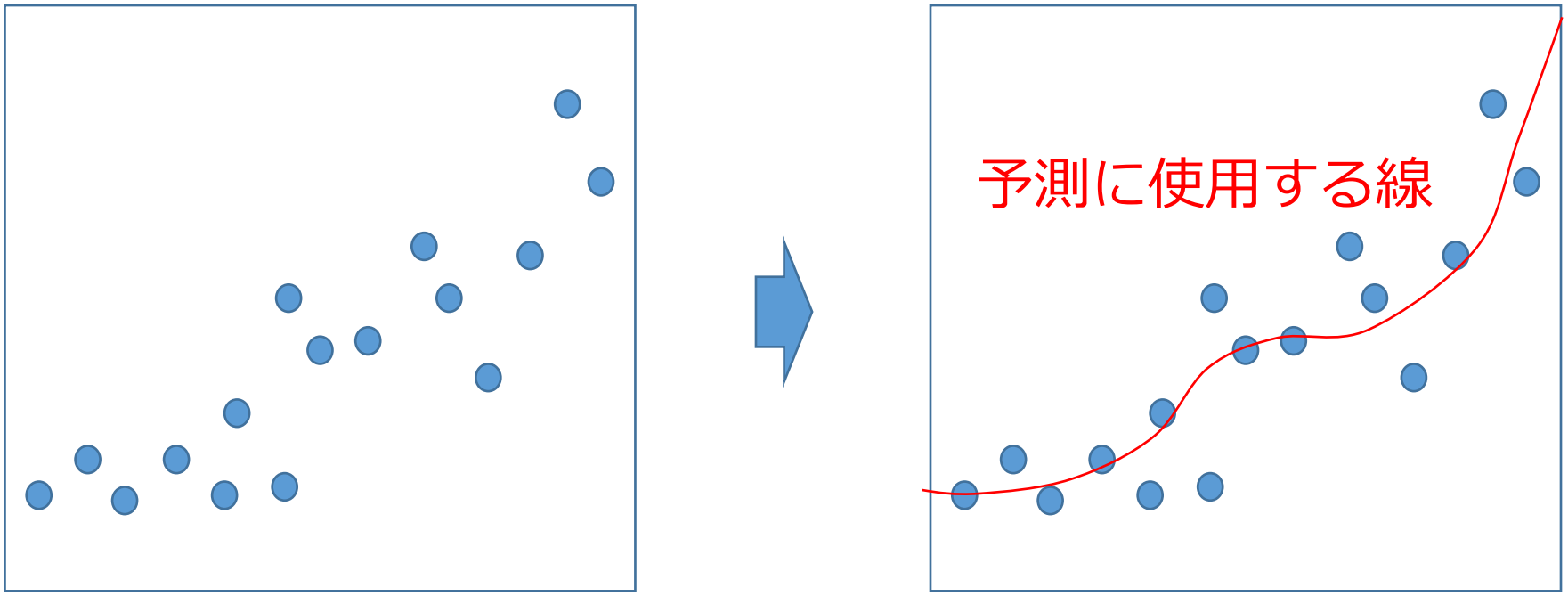
それぞれの範囲を得る



判別：２種類への分類

回帰

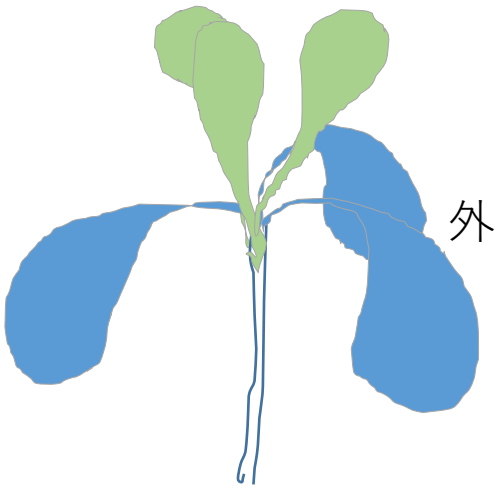
訓練データ



回帰：ある量から，別の量を予測するためのもの
（例）身長と体重の関係，年齢と収入の関係

4-2 Iris データセット

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

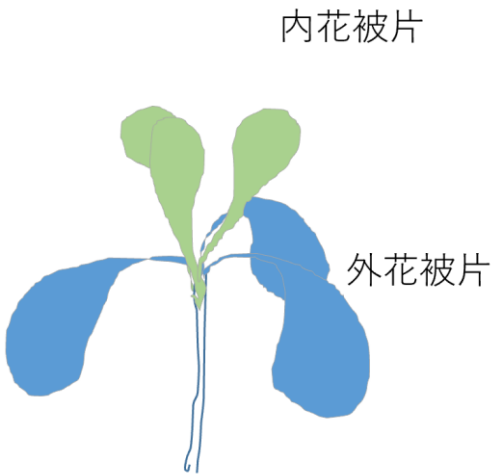
  virginica
- データ数は **50 × 3**

作成者：Ronald Fisher

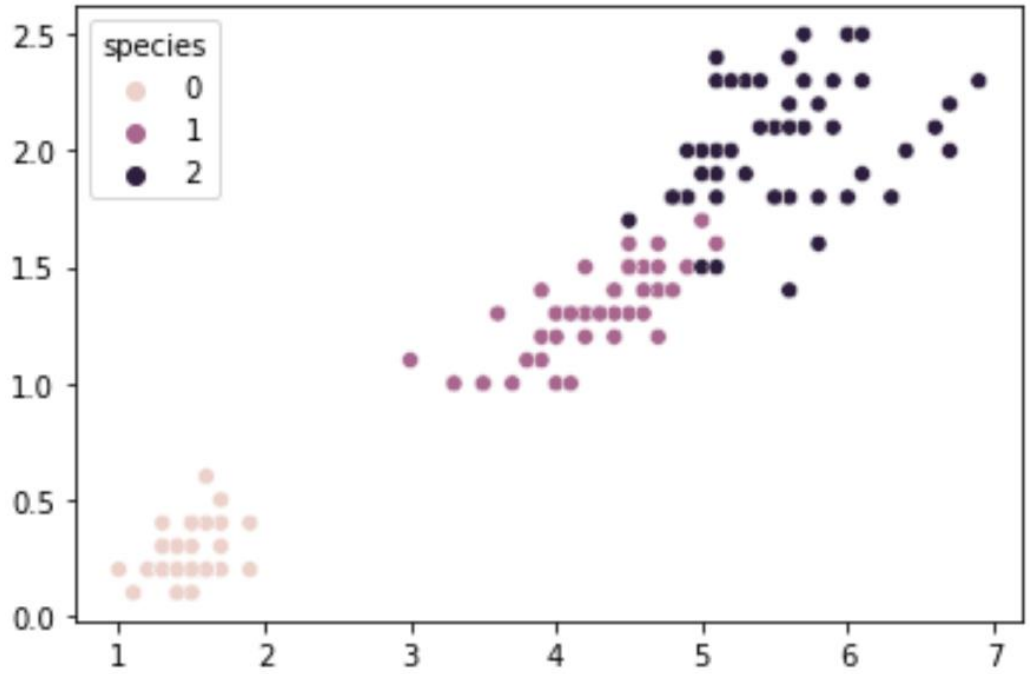
作成年：1936

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

0: **setosa**
1: **versicolor**
2: **virginica**

Iris データセットと配列（アレイ）

Iris データセット

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

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
```

特徴量（数値）
サイズ：**150 × 4**

```
[0
0
0
0
0
0
0
0
0
0
```

ラベル（数値）
サイズ：**150**

setosa → 0
versicolor → 1
virginia → 2

ラベルの数値化

演習

Iris データセットをロードし，特徴量とラベルを表示

ページ１５～１７

【トピックス】

- Google Colaboratory でのプログラム実行
- Iris データセット

Google Colaboratory の使い方概要 ①

Object detection

ファイル 編集 表示 挿入 ランタイム ツール ヘルプ

共有 ログイン

目次

+ コード + テキスト ドライブにコピー 接続 編集

- Copyright 2018 The TensorFlow Hub Authors.
- Object Detection
  - Setup
    - Imports and function definitions
  - Example use
    - Helper functions for downloading images and for visualization.
  - Apply module
    - More images

セクション

Copyright 2018 The TensorFlow Hub Authors.

Licensed under the Apache License, Version 2.0 (the "License");

```
[ ] # Copyright 2018 The TensorFlow Hub Authors. All Rights Reserved.
    #
    # Licensed under the Apache License, Version 2.0 (the "License");
    # you may not use this file except in compliance with the License.
    # You may obtain a copy of the License at
    #
    #     http://www.apache.org/licenses/LICENSE-2.0
    #
    # Unless required by applicable law or agreed to in writing, software
    # distributed under the License is distributed on an "AS IS" BASIS,
    # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    # See the License for the specific language governing permissions and
    # limitations under the License.
    # ==============================================================================
```

実行

コードセル

Object Detection

View on TensorFlow.org | Run in Google Colab | View on GitHub | Download notebook | See TF Hub models

This Colab demonstrates use of a TF-Hub module trained to perform object detection.

Setup

```
[ ] #@title Imports and function definitions
```

Imports and function definitions

Google Colaboratory ノートブック

コードセルの再実行や変更には,

Google アカウントでのログインが必要

Google Colaboratory の使い方概要 ②

実行

```
[ ] files = ['a.png', 'b.png','c.png', '126.png', '127.png']
```

コードセル

6．顔検出

顔検出は，**写真やビデオの中の顔を検出すること**．顔とそれ以外のオブジェクトを区別することも行う．顔検出の結果は，バウンディングボックスで得られるのが普通である．

次のプログラムは，Dlib を用いて，画像からの顔検出を行う。

- 「**dets = cnn_face_detector(img, 6)**」・・・**顔検出**の実行
- 「**cv2.rectangle(disp, (d.rect.left(), d.rect.top()), (d.rect.right(), d.rect.bottom()), (255, 0, 0), 1)**」・・・**顔検出の結果**を**四角形で表示**

結果は，赤い四角で表示される．１，３，４，５番目の画像（a.png, c.png, 126.png, 127.png）からは，顔が検出される．２番目の画像（b.png, 手で顔を覆い隠したもの）からは顔が検出されない．**少し隠れていたり，顔が傾いていても顔検出ができる**が，**大きく隠れていると顔検出できない**.

実行結果が長いので，スクロールして**全体を確認**すること.

謝辞: ここの Python プログラムは，Dlib に付属の cnn_face_detector.py を書き換えて使用している

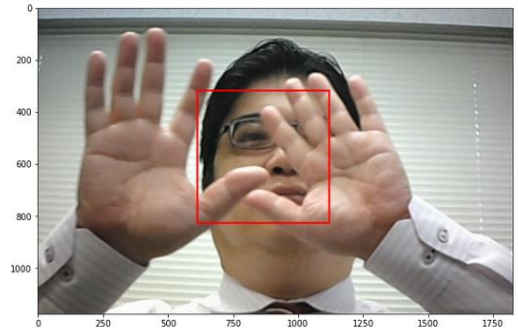
テキストセル

実行

```
import sys
import dlib
import os
import urllib.request
import cv2
import matplotlib.pyplot as plt

cnn_face_detector = dlib.cnn_face_detection_model_v1('mmod_human_face_detector.dat')

for f in files:
    print("*** file: {} ***".format(f))
    img = dlib.load_rgb_image(f)
    dets = cnn_face_detector(img, 1)
    print("Number of faces detected: {}".format(len(dets)))
    for i, d in enumerate(dets):
        print("Detection {}: Left: {} Top: {} Right: {} Bottom: {} Confidence: {}".format(
            i, d.rect.left(), d.rect.top(), d.rect.right(), d.rect.bottom(), d.confidence))
    disp = img.copy()
    for i, d in enumerate(dets):
        cv2.rectangle(disp, (d.rect.left(), d.rect.top()), (d.rect.right(), d.rect.bottom()), (255, 0, 0), 8)
    plt.figure(figsize=(10,10))
    plt.imshow(disp)
    plt.show()
```

コードセル

```
*** file: a.png ***
Number of faces detected: 1
Detection 0: Left: 614 Top: 319 Right: 1121 Bottom: 827 Confidence: 0.20801100134849548
```




```
*** file: b.png ***
Number of faces detected: 0
```

- **WEBブラウザでアクセス**
- **コードセル**は **Python プログラム**.
  各自の Google アカウントでログインすれば，変更，再実行可能

一番上のコードセルから順々に実行

Iris データセットは Python でも利用可能

ソースコード

```
from sklearn.datasets import load_iris
iris = load_iris()
x, y = iris.data, iris.target
print(x)
print(y)
```

x に**特徴量**を，yに**ラベル**を格納

```
from sklearn.datasets import load_iris
iris = load_iris()
x, y = iris.data, iris.target
print(x)
print(y)
```

```
[6.2 2.9 4.3 1.3]
[5.1 2.5 3.  1.1]
[5.7 2.8 4.1 1.3]
[6.3 3.3 6.  2.5]
[5.8 2.7 5.1 1.9]
[7.1 3.  5.9 2.1]
[6.3 2.9 5.6 1.8]
[6.5 3.  5.8 2.2]
[7.6 3.  6.6 2.1]
[4.9 2.5 4.5 1.7]
[7.3 2.9 6.3 1.8]
[6.7 2.5 5.8 1.8]
[7.2 3.6 6.1 2.5]
[6.5 3.2 5.1 2. ]
[6.4 2.7 5.3 1.9]
[6.8 3.  5.5 2.1]
[5.7 2.5 5.  2. ]
[5.8 2.8 5.1 2.4]
[6.4 3.2 5.3 2.3]
[6.5 3.  5.5 1.8]
[7.7 3.8 6.7 2.2]
[7.7 2.6 6.9 2.3]
[6.  2.2 5.  1.5]
```

実行結果

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
 [4.6 3.6 1.  0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
 [5.  3.  1.6 0.2]
 [5.  3.4 1.6 0.4]
 [5.2 3.5 1.5 0.2]
 [5.2 3.4 1.4 0.2]
 [4.7 3.2 1.6 0.2]
 [4.8 3.1 1.6 0.2]
 [5.4 3.4 1.5 0.4]
 [5.2 4.1 1.5 0.1]
 [5.5 4.2 1.4 0.2]
 [4.9 3.1 1.5 0.2]
 [5.  3.2 1.2 0.2]
```

x を表示している部分

```
[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 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]
```

y を表示している部分

4-3 分類

Iris データセットの分類

- **花びらの長さと幅**から，**花の種類**を予測

花びら4つの数値（長さと幅の2つによる計4つの値）に基づいて、**3種類の花**（アイリスセットーサ、アイリスバーシクル、アイリスバージニカ）の**一つを予測**する

- **利用可能なデータを分割**

150個を，120個と 30個のように分割

120個：**訓練データ**

30個：**検証データ**（過学習や学習不足の可能性を評価）

演習

Iris データセットを１２０個と３０個に分割

ページ２１

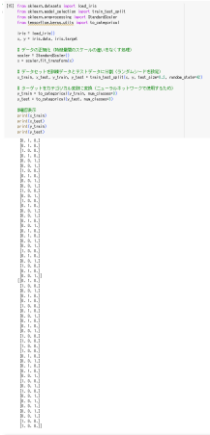
【トピックス】

- Google Colaboratory でのプログラム実行
- Irisデータセットの分割

Iris データセットの利用可能なデータを分割

ソースコード

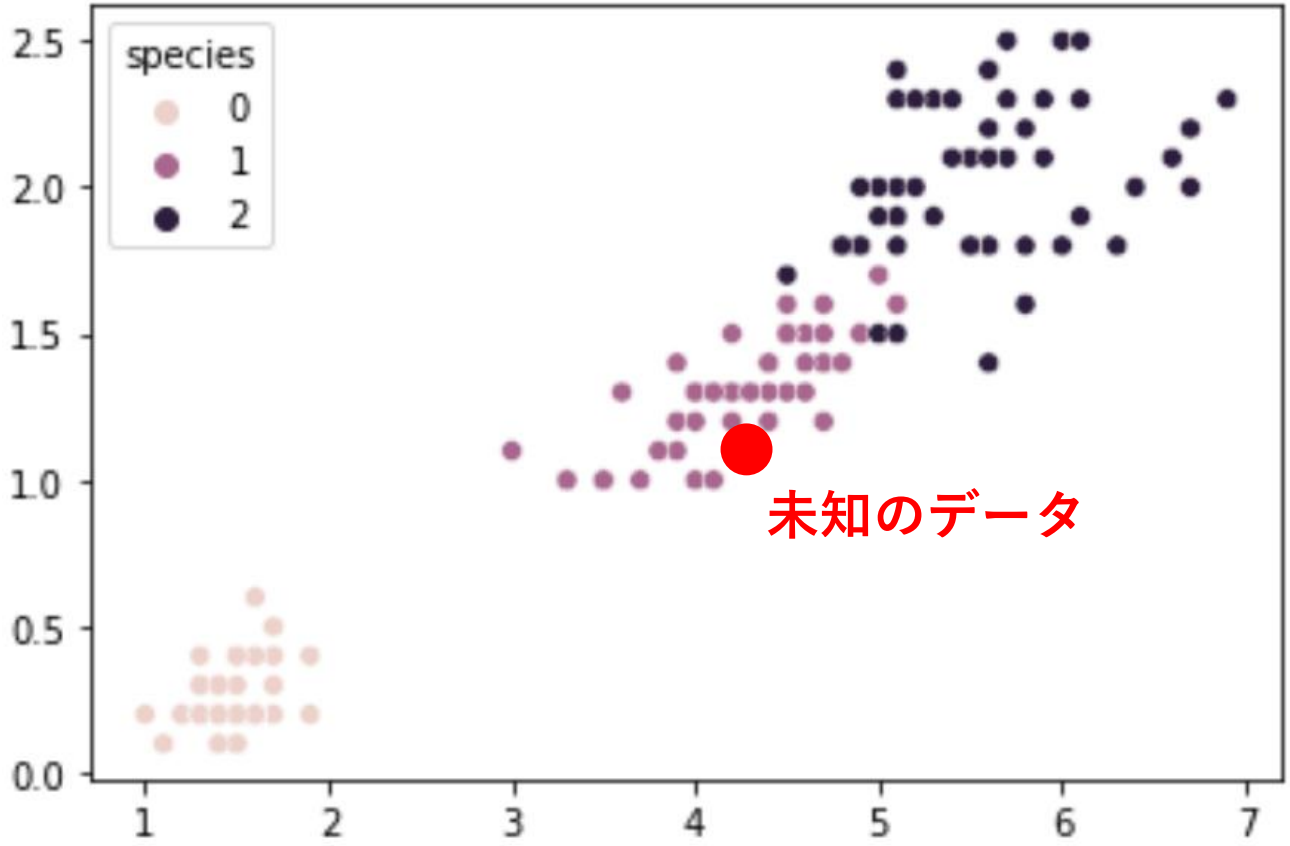
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.utils import to_categorical

iris = load_iris()
x, y = iris.data, iris.target

# データの正規化（特徴量間のスケールの違いをなくす処理）
scaler = StandardScaler()
x = scaler.fit_transform(x)

# データセットを訓練データとテストデータに分割（ランダムシードを設定）
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# ターゲットをカテゴリカル変数に変換（ニューラルネットワークで使用するため）
y_train = to_categorical(y_train, num_classes=3)
y_test = to_categorical(y_test, num_classes=3)

#確認表示
print(x_train)
print(x_test)
print(y_train)
print(y_test)
```



実行結果

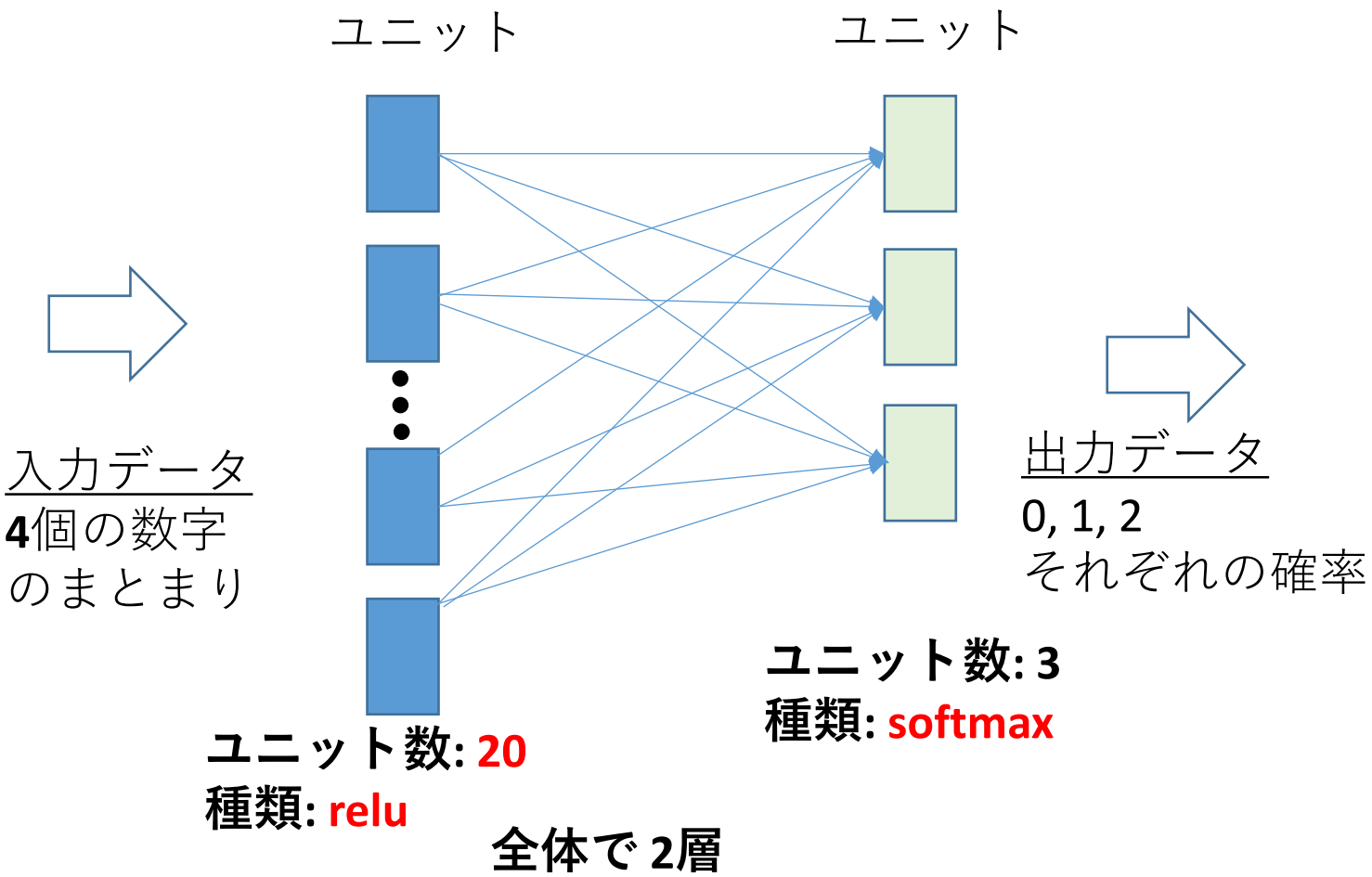
分類

- 学習により、「未知のデータを分類できる能力を獲得」と考えることもできる



分類を行うニューラルネットワークの例

- **1層目：ユニット数 20, 種類は relu**
- **2層目：ユニット数 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 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層目のユニット数は **20**
種類は **relu**

２層目のユニット数は **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
- **ユニット**の数（**層**ごと）
- **ユニット**の種類（**層**ごと）

ニューラルネットワークの学習を行うプログラム例

学習の繰り返し回数は **50**

```
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

演習

Iris データセットを用いた学習，分類の実行

ページ２７

【トピックス】

- **Google Colaboratory でのプログラム実行**
- **ニューラルネットワークの作成**
- **学習の実行**
- **分類の実行**

Iris データセットによる学習と分類の実行

ソースコード

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import tensorflow as tf
from tensorflow.keras.utils import to_categorical

iris = load_iris()
x, y = iris.data, iris.target

# データの正規化（特徴量間のスケールの違いをなくす処理）
scaler = StandardScaler()
x = scaler.fit_transform(x)

# データセットを訓練データとテストデータに分割（ランダムシードを設定）
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

# ターゲットをカテゴリカル変数に変換（ニューラルネットワークで使用するため）
y_train = to_categorical(y_train, num_classes=3)
y_test = to_categorical(y_test, num_classes=3)

def create_model():
    return 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')
    ])

m = create_model()
m.compile(optimizer='adam',
          loss='categorical_crossentropy',
          metrics=['accuracy'])
EPOCHS = 50
history = m.fit(x = x_train,
                y = y_train,
                epochs = EPOCHS,
                validation_data = (x_test, y_test),
                verbose = 2)

m.evaluate(x_test, y_test)
# 分類を実行
predictions = m.predict(x_test)

# 確率が最大のクラスを選択
predicted_classes = np.argmax(predictions, axis=1)
actual_classes = np.argmax(y_test, axis=1)

# アイリスのクラス名
class_names = iris.target_names

# 予測結果と実際のクラスを表示
for i in range(len(predicted_classes)):
    print(f"分類結果: {class_names[predicted_classes[i]]}, 実際のクラス: {class_names[actual_classes[i]]}")
```

```
Epoch 38/50
4/4 - 0s - loss: 0.5121 - accuracy: 0.8167 - val_loss: 0.4648 - val_accuracy: 0.8333 - 34ms/epoch - 9ms/step
Epoch 39/50
4/4 - 0s - loss: 0.4910 - accuracy: 0.8167 - val_loss: 0.4580 - val_accuracy: 0.8333 - 36ms/epoch - 9ms/step
Epoch 40/50
4/4 - 0s - loss: 0.4794 - accuracy: 0.8333 - val_loss: 0.4514 - val_accuracy: 0.8667 - 39ms/epoch - 10ms/step
Epoch 41/50
4/4 - 0s - loss: 0.4056 - accuracy: 0.8250 - val_loss: 0.4450 - val_accuracy: 0.8667 - 56ms/epoch - 14ms/step
Epoch 42/50
4/4 - 0s - loss: 0.4851 - accuracy: 0.8083 - val_loss: 0.4388 - val_accuracy: 0.8667 - 38ms/epoch - 10ms/step
Epoch 43/50
4/4 - 0s - loss: 0.4755 - accuracy: 0.8333 - val_loss: 0.4328 - val_accuracy: 0.8667 - 37ms/epoch - 9ms/step
Epoch 44/50
4/4 - 0s - loss: 0.4081 - accuracy: 0.8000 - val_loss: 0.4271 - val_accuracy: 0.8667 - 36ms/epoch - 9ms/step
Epoch 45/50
4/4 - 0s - loss: 0.4814 - accuracy: 0.8333 - val_loss: 0.4215 - val_accuracy: 0.8667 - 37ms/epoch - 9ms/step
Epoch 46/50
4/4 - 0s - loss: 0.4594 - accuracy: 0.8250 - val_loss: 0.4160 - val_accuracy: 0.8667 - 39ms/epoch - 10ms/step
Epoch 47/50
4/4 - 0s - loss: 0.4302 - accuracy: 0.8583 - val_loss: 0.4105 - val_accuracy: 0.8667 - 54ms/epoch - 13ms/step
Epoch 48/50
4/4 - 0s - loss: 0.4567 - accuracy: 0.8250 - val_loss: 0.4052 - val_accuracy: 0.8667 - 41ms/epoch - 10ms/step
Epoch 49/50
4/4 - 0s - loss: 0.4552 - accuracy: 0.8333 - val_loss: 0.4001 - val_accuracy: 0.8667 - 36ms/epoch - 9ms/step
Epoch 50/50
4/4 - 0s - loss: 0.4540 - accuracy: 0.8333 - val_loss: 0.3951 - val_accuracy: 0.8667 - 36ms/epoch - 9ms/step
1/1 [==============================] - 0s 25ms/step - loss: 0.3951 - accuracy: 0.8667
1/1 [==============================] - 0s 53ms/step
分類結果: virginica, 実際のクラス: versicolor
分類結果: setosa, 実際のクラス: setosa
分類結果: virginica, 実際のクラス: virginica
分類結果: virginica, 実際のクラス: versicolor
分類結果: virginica, 実際のクラス: versicolor
分類結果: setosa, 実際のクラス: setosa
分類結果: versicolor, 実際のクラス: versicolor
分類結果: virginica, 実際のクラス: virginica
分類結果: versicolor, 実際のクラス: versicolor
分類結果: versicolor, 実際のクラス: versicolor
分類結果: virginica, 実際のクラス: virginica
分類結果: setosa, 実際のクラス: setosa
分類結果: setosa, 実際のクラス: setosa
分類結果: setosa, 実際のクラス: setosa
分類結果: setosa, 実際のクラス: setosa
分類結果: virginica, 実際のクラス: versicolor
分類結果: virginica, 実際のクラス: virginica
分類結果: versicolor, 実際のクラス: versicolor
分類結果: versicolor, 実際のクラス: versicolor
分類結果: virginica, 実際のクラス: virginica
分類結果: setosa, 実際のクラス: setosa
分類結果: virginica, 実際のクラス: virginica
分類結果: setosa, 実際のクラス: setosa
分類結果: virginica, 実際のクラス: virginica
分類結果: virginica, 実際のクラス: virginica
分類結果: virginica, 実際のクラス: virginica
分類結果: virginica, 実際のクラス: virginica
分類結果: virginica, 実際のクラス: virginica
分類結果: setosa, 実際のクラス: setosa
分類結果: setosa, 実際のクラス: setosa
```

実行結果

学習についての表示部分

- 繰り返し学習を行う．訓練データにより，モデルが更新される．
- 各更新後，検証データを用いて，性能を評価（検証データを分類し，正解と比較して，精度を検証）．その結果は，学習不足や過学習の可能性を判断するのに使用

```
Epoch 1/50
4/4 - 1s - loss: 1.0542 - accuracy: 0.4333 - val_loss: 1.0251 - val_accuracy: 0.5000 - 1s/epoch - 301ms/step
Epoch 2/50
4/4 - 0s - loss: 1.0010 - accuracy: 0.6167 - val_loss: 0.9974 - val_accuracy: 0.6667 - 67ms/epoch - 17ms/step
Epoch 3/50
4/4 - 0s - loss: 0.9844 - accuracy: 0.6667 - val_loss: 0.9708 - val_accuracy: 0.7333 - 53ms/epoch - 13ms/step
Epoch 4/50
4/4 - 0s - loss: 0.9569 - accuracy: 0.7083 - val_loss: 0.9452 - val_accuracy: 0.7667 - 53ms/epoch - 13ms/step
Epoch 5/50
4/4 - 0s - loss: 0.9344 - accuracy: 0.7083 - val_loss: 0.9204 - val_accuracy: 0.7667 - 50ms/epoch - 12ms/step
Epoch 6/50
4/4 - 0s - loss: 0.9048 - accuracy: 0.7667 - val_loss: 0.8968 - val_accuracy: 0.7667 - 67ms/epoch - 17ms/step
Epoch 7/50
4/4 - 0s - loss: 0.8733 - accuracy: 0.7750 - val_loss: 0.8739 - val_accuracy: 0.8000 - 68ms/epoch - 17ms/step
Epoch 8/50
4/4 - 0s - loss: 0.8716 - accuracy: 0.7750 - val_loss: 0.8519 - val_accuracy: 0.8000 - 90ms/epoch - 22ms/step
Epoch 9/50
4/4 - 0s - loss: 0.8406 - accuracy: 0.7667 - val_loss: 0.8313 - val_accuracy: 0.8000 - 74ms/epoch - 19ms/step
Epoch 10/50
4/4 - 0s - loss: 0.8318 - accuracy: 0.7667 - val_loss: 0.8112 - val_accuracy: 0.8000 - 59ms/epoch - 15ms/step
Epoch 11/50
4/4 - 0s - loss: 0.8092 - accuracy: 0.8000 - val_loss: 0.7917 - val_accuracy: 0.8000 - 75ms/epoch - 19ms/step
Epoch 12/50
4/4 - 0s - loss: 0.7769 - accuracy: 0.8000 - val_loss: 0.7729 - val_accuracy: 0.8000 - 78ms/epoch - 20ms/step
Epoch 13/50
4/4 - 0s - loss: 0.7654 - accuracy: 0.7917 - val_loss: 0.7547 - val_accuracy: 0.8333 - 53ms/epoch - 13ms/step
```

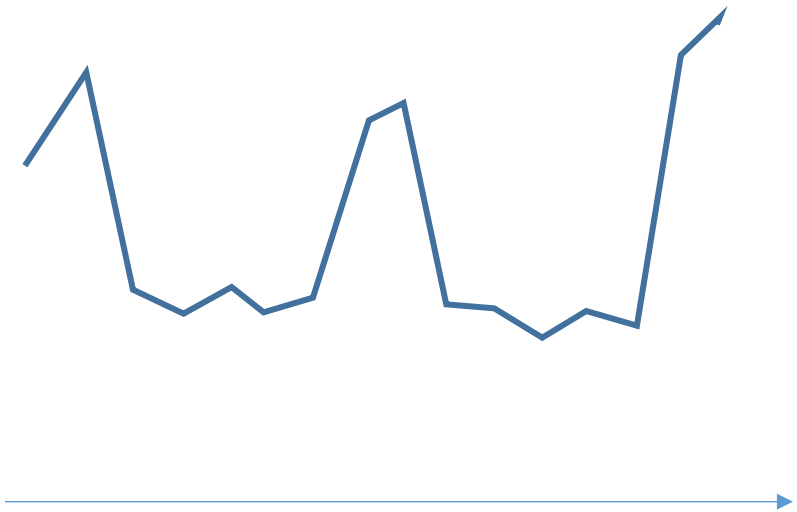
検証データについて，
予測結果と正解を比べての
精度 (accuracy)

分類結果についての表示部分

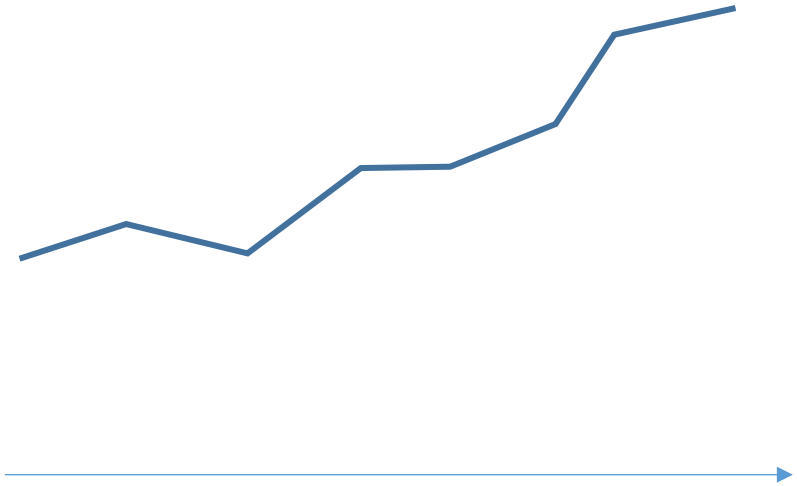
- 検証データを分類した結果と正解と表示

```
1/1 [==============================] - 0s 53ms/step
分類結果: virginica, 実際のクラス: versicolor
分類結果: setosa, 実際のクラス: setosa
分類結果: virginica, 実際のクラス: virginica
分類結果: virginica, 実際のクラス: versicolor
分類結果: virginica, 実際のクラス: versicolor
分類結果: setosa, 実際のクラス: setosa
分類結果: versicolor, 実際のクラス: versicolor
分類結果: virginica, 実際のクラス: virginica
分類結果: versicolor, 実際のクラス: versicolor
分類結果: versicolor, 実際のクラス: versicolor
分類結果: virginica, 実際のクラス: virginica
分類結果: setosa, 実際のクラス: setosa
分類結果: setosa, 実際のクラス: setosa
分類結果: setosa, 実際のクラス: setosa
分類結果: setosa, 実際のクラス: setosa
分類結果: virginica, 実際のクラス: versicolor
分類結果: virginica, 実際のクラス: virginica
分類結果: versicolor, 実際のクラス: versicolor
分類結果: versicolor, 実際のクラス: versicolor
分類結果: virginica, 実際のクラス: virginica
分類結果: setosa, 実際のクラス: setosa
分類結果: virginica, 実際のクラス: virginica
分類結果: setosa, 実際のクラス: setosa
分類結果: virginica, 実際のクラス: virginica
分類結果: virginica, 実際のクラス: virginica
分類結果: virginica, 実際のクラス: virginica
分類結果: virginica, 実際のクラス: virginica
分類結果: virginica, 実際のクラス: virginica
分類結果: setosa, 実際のクラス: setosa
分類結果: setosa, 実際のクラス: setosa
```

4-4. 時系列データ



過去の量から，周期性を分析し活用



過去の量から，トレンドを分析し活用

時系列データ

時系列データは,

昨日の気温は**15**度

今日の気温が**13**度

のように，**時間とともに，値が変化するデータ**

時系列データの例

year	month	day	sn_value	s
1848	12	23	353	
1848	12	24	240	
1848	12	25	275	
1848	12	26	352	
1848	12	27	268	
1848	12	28	285	
1848	12	29	343	
1848	12	30	340	
1848	12	31	238	
1849	1	1	287	

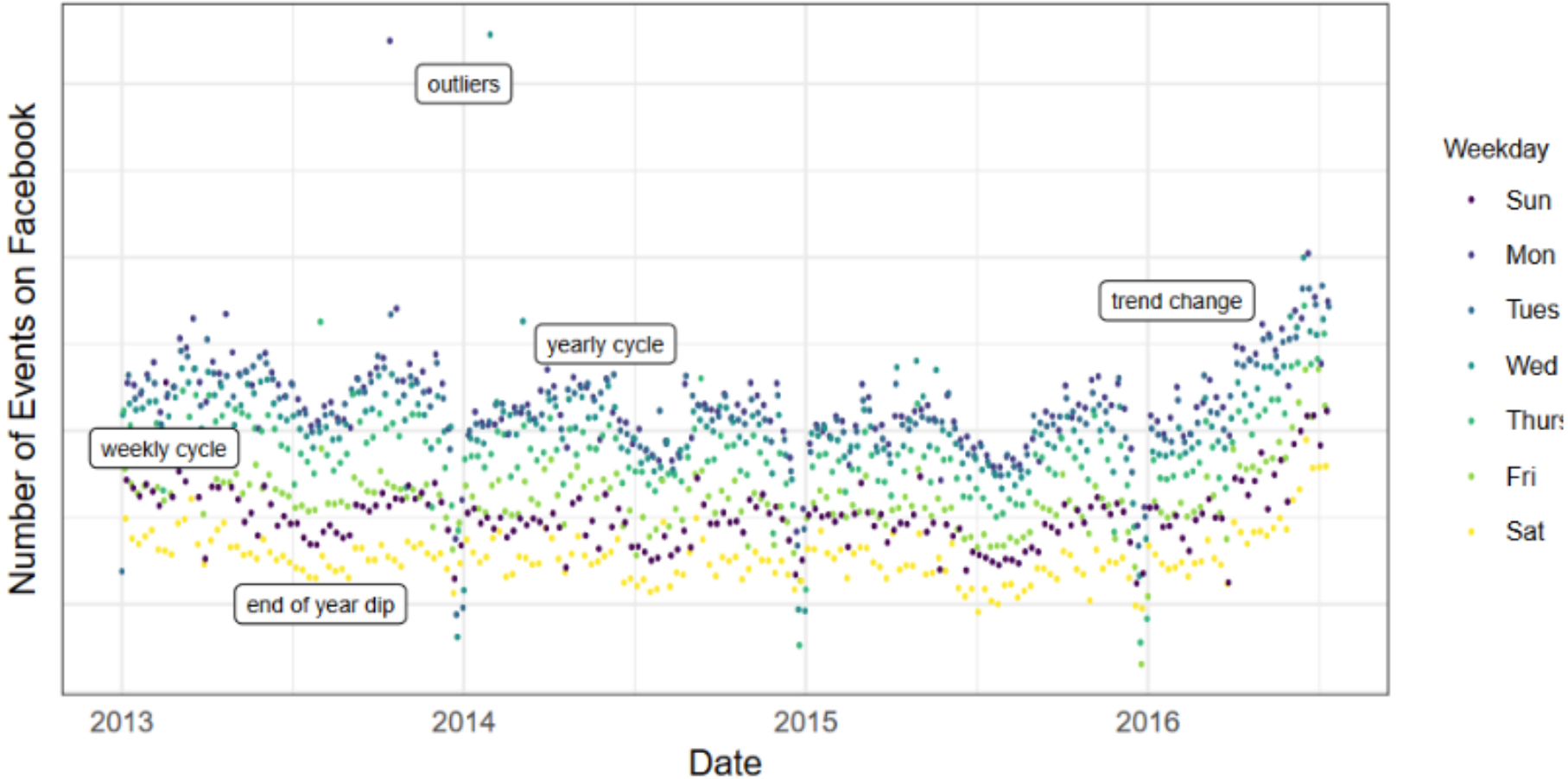
太陽の黒点数の変化

時系列データの特性

- **周期性**：　週単位，月単位，年単位
- **イベント**：　正月，クリスマス，４月頭
- **長期的な傾向**：　増加傾向，一定，減少傾向

- **誤差や，突発的な変化・変動**

Facebook イベント数の分析例



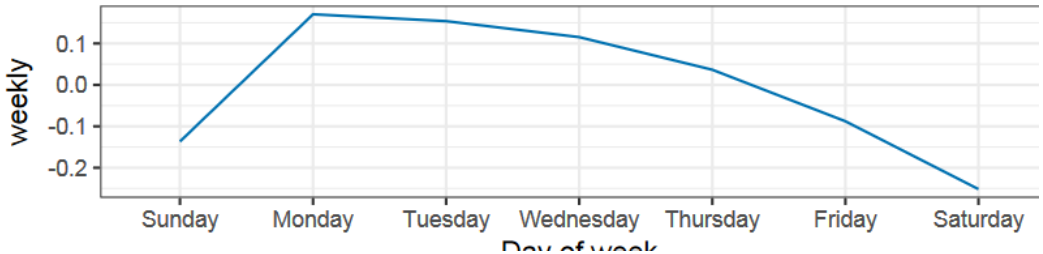
曜日で色を変えて**プロット**　⇒　曜日単の周期性を読み取る

Taylor SJ, Letham B. 2017. Forecasting at scale. PeerJ Preprints 5:e3190v2
https://doi.org/10.7287/peerj.preprints.3190v2

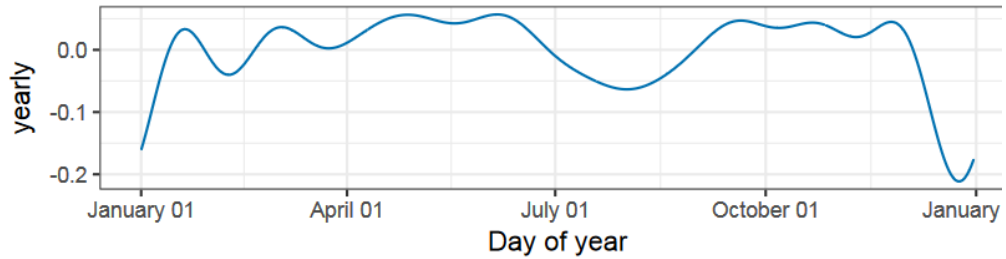
Facebook イベント数の分析例

周期性の分析

元データをProphet で処理した結果



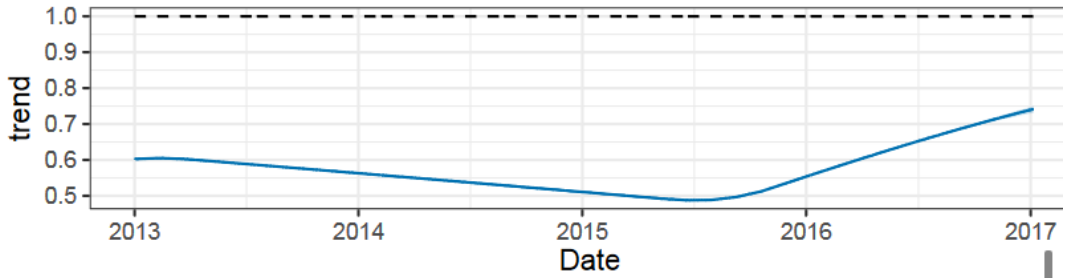
土日は少ない



年末年始は少ない

トレンドの分析

元データをProphet で処理した結果



2015年からは増加

Taylor SJ, Letham B. 2017. Forecasting at scale. PeerJ Preprints 5:e3190v2
https://doi.org/10.7287/peerj.preprints.3190v2

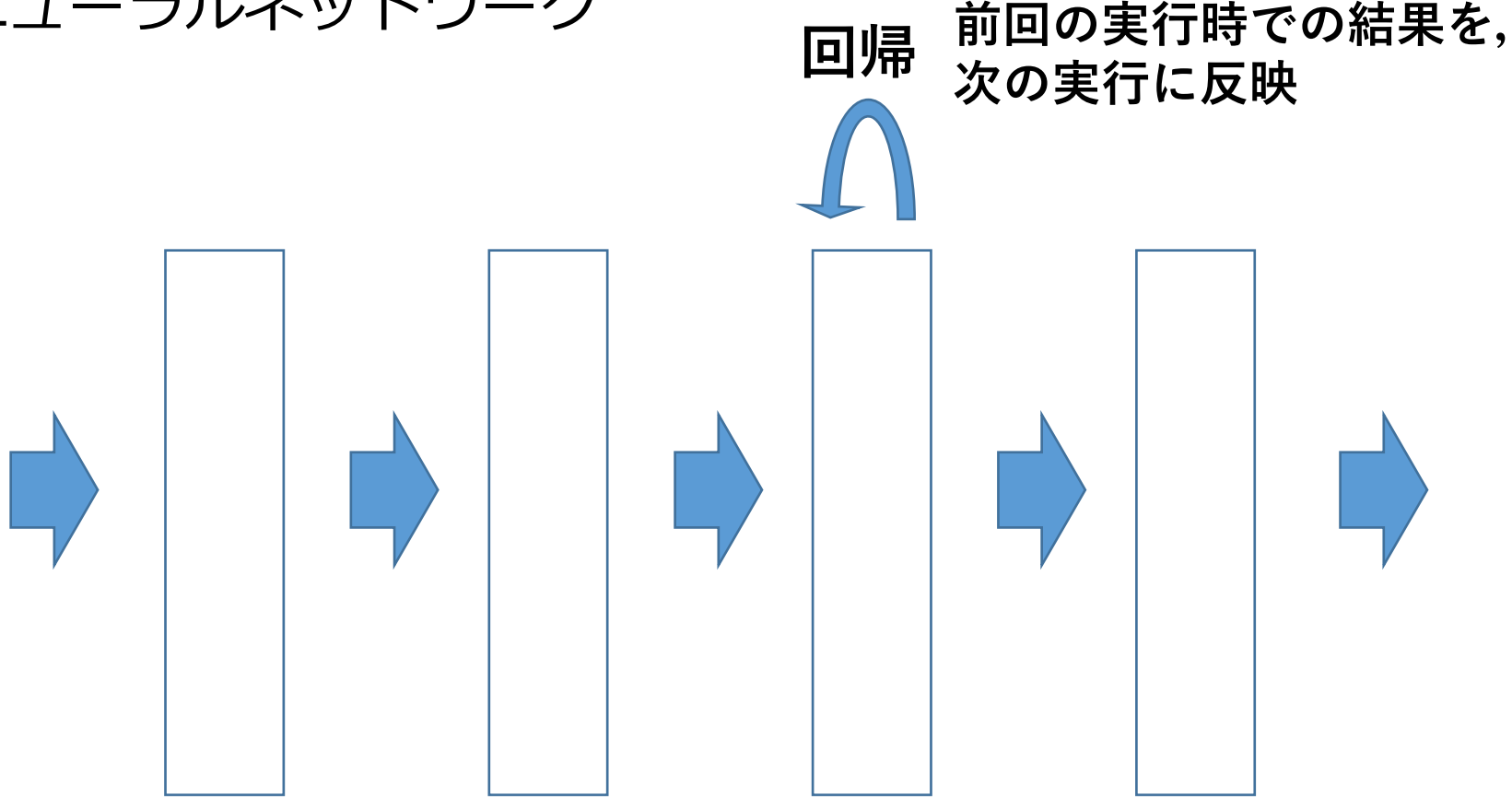
まとめ

- **時系列データ**は，**時間とともに，値が変化するデータ**
- **時系列データ**から，**周期性**や**トレンド**などを読み取ることができる

4-5. リカレント
ニューラルネットワークとLSTM

リカレントニューラルネットワーク

リカレントニューラルネットワークは
回帰により**過去の情報を保持**する
ニューラルネットワーク

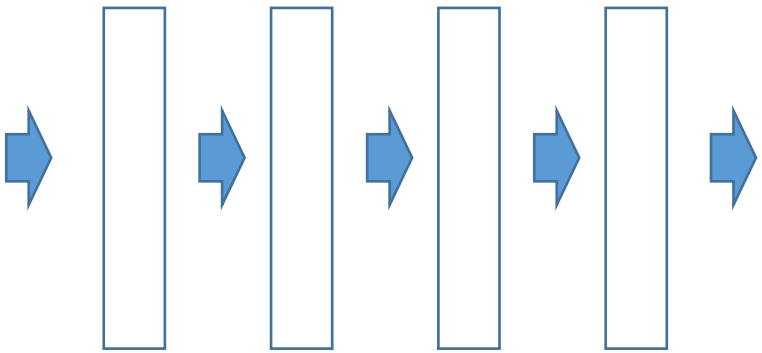


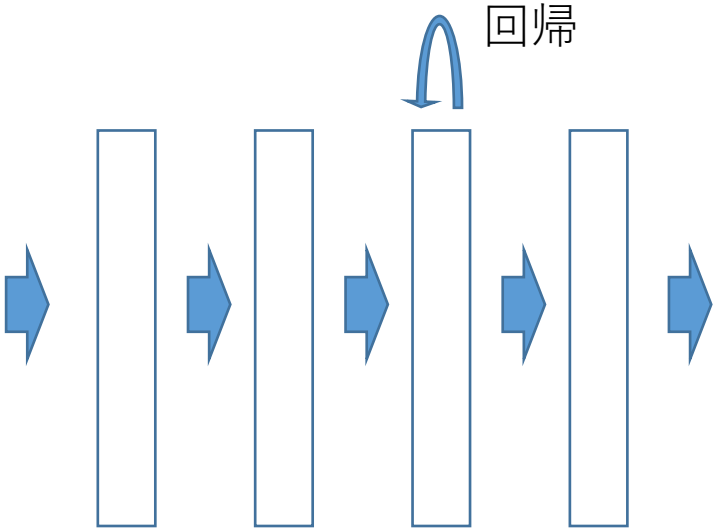
リカレントニューラルネットワーク

- **回帰**により，**過去の情報を保持**．前回の実行時での結果の一部が，次の実行に反映される

- **時系列データ**など，**データの並びを扱う能力**を持つ

フィードフォワードとリカレントニューラルネットワーク





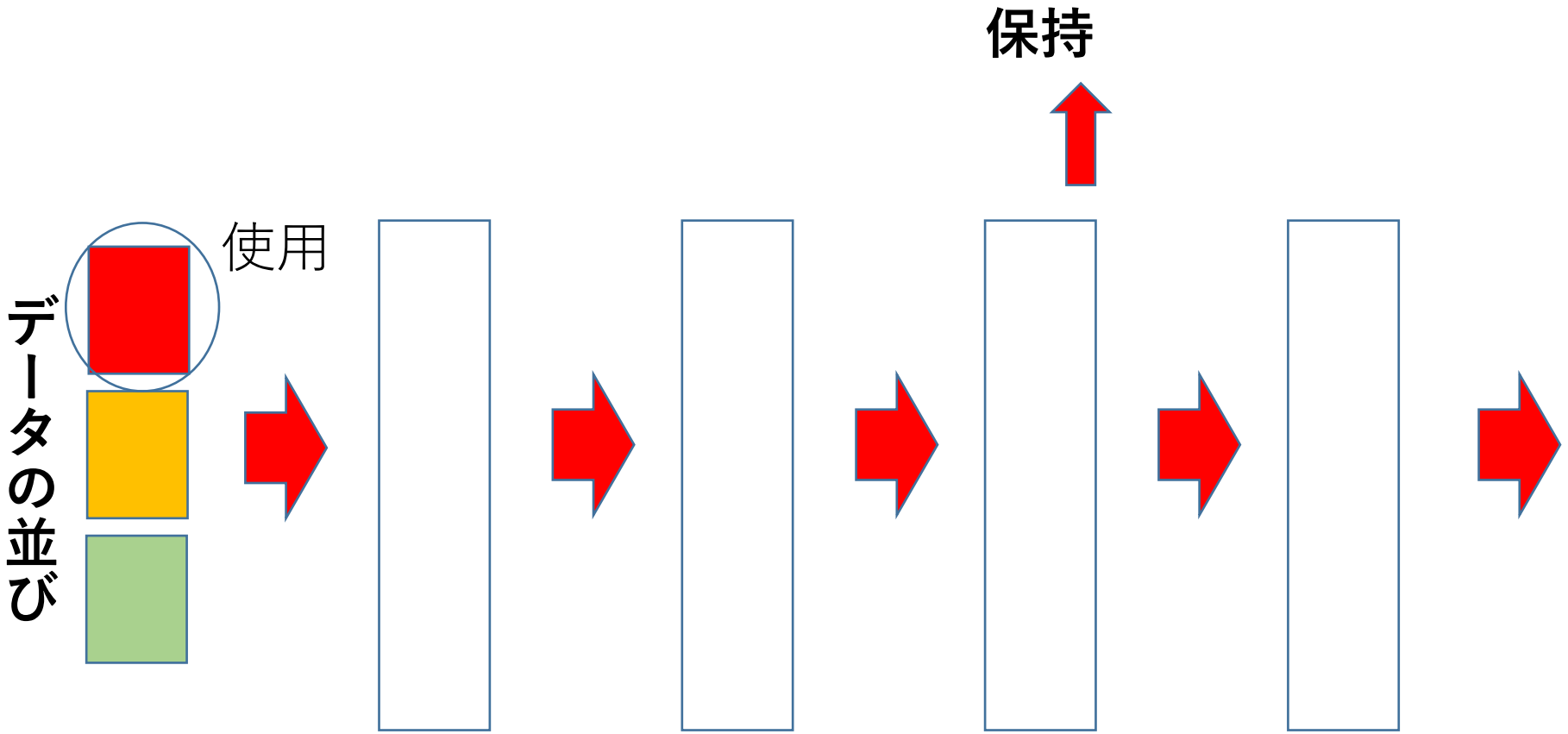
フィードフォワードネットワーク

ある層の出力を，次の層が受け取る

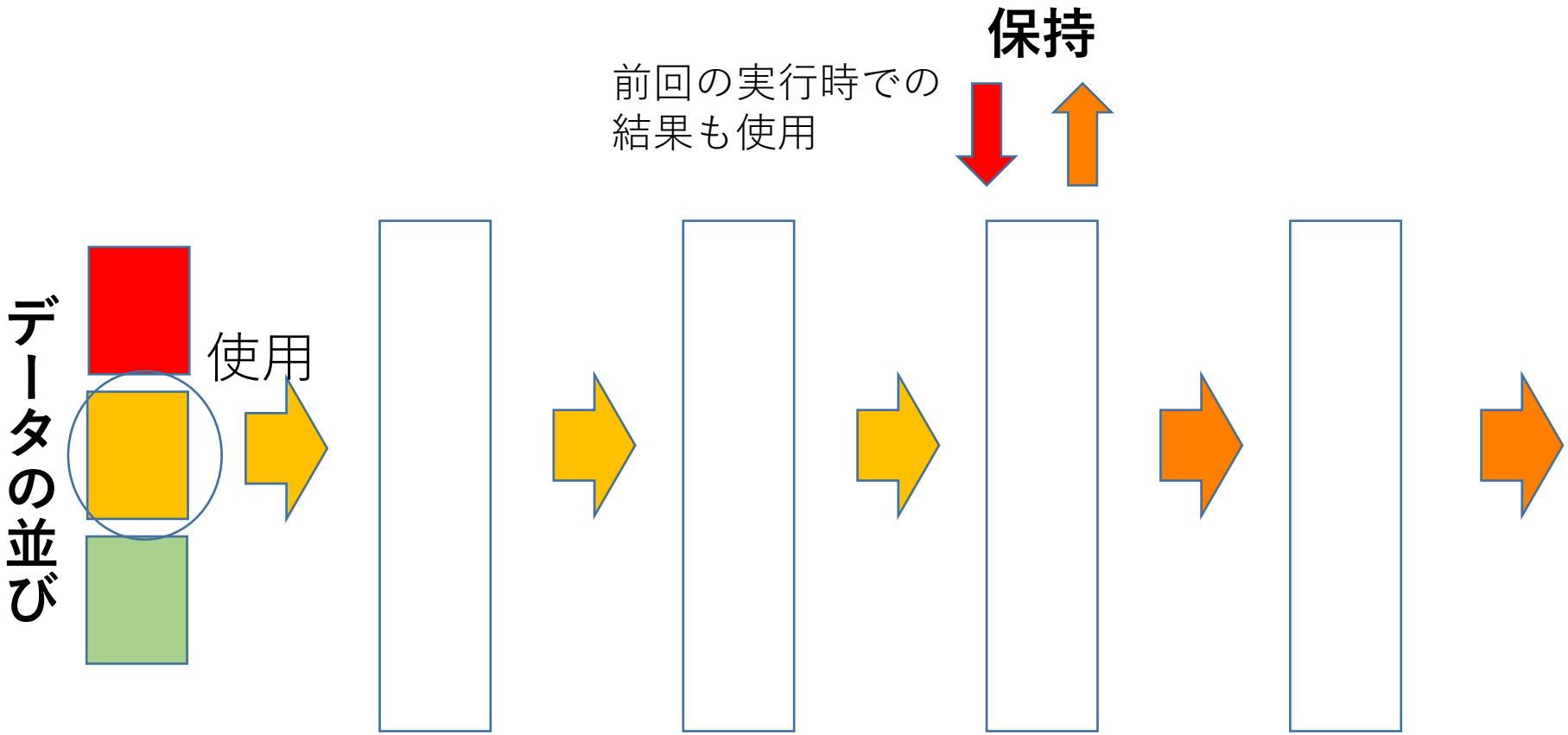
リカレントニューラルネットワーク

回帰により，前回の実行時での結果の一部が，次の実行に反映される．

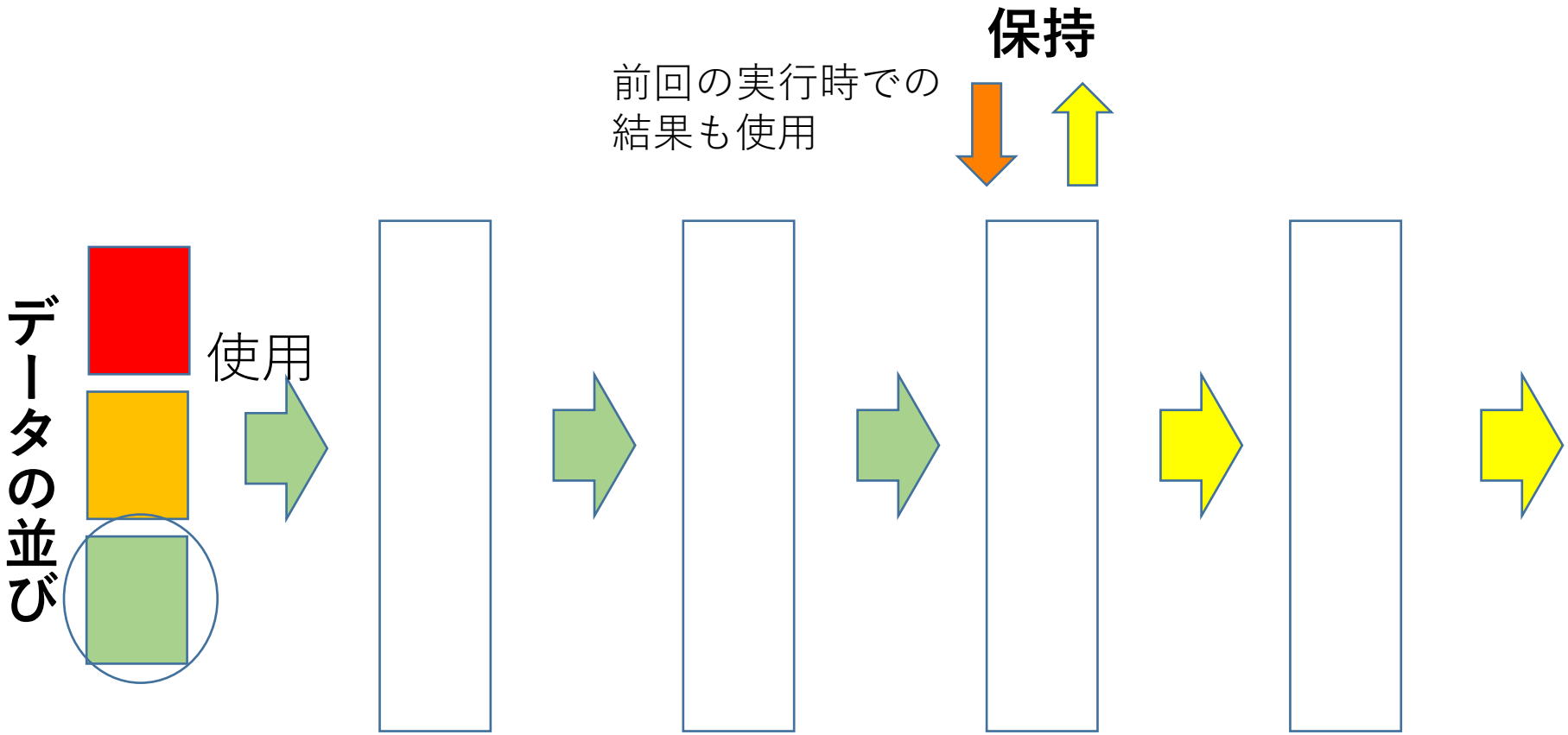
リカレントニューラルネットワークの動作イメージ ①



リカレントニューラルネットワークの動作イメージ ②



リカレントニューラルネットワークの動作イメージ ③



リカレントニューラルネットワークの応用

- **時系列データを用いた予測**

  データの並びを扱う

- **手書き文字認識**

  筆記の動きを扱う

- **音声認識**

  音の並びを扱う

- **「言葉」の理解，翻訳，テキスト生成，プログラム生成**

  単語の並びを扱う

LSTM 誕生の背景

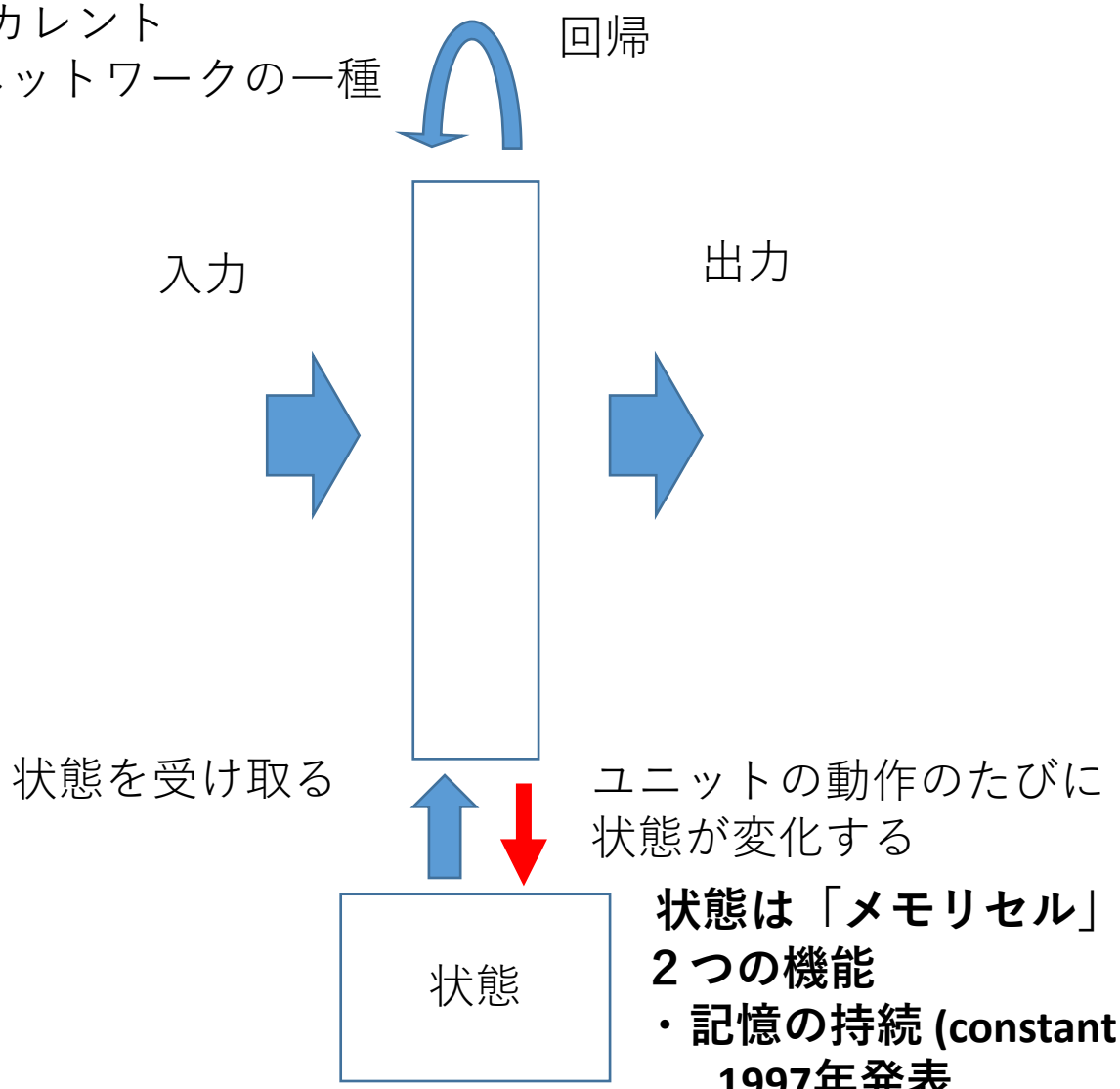
- **リカレントニューラルネットワーク**では，**長期に及ぶ**過去の情報の保持が困難 （1991年, 1994年に理論的根拠が示された）

- 長期に及ぶ過去の情報を保持しようとすると，問題が発生（勾配消失や勾配爆発により，学習がうまくいかなくなる）

Y Bengio 1, P Simard, P Frasconi,
Learning long-term dependencies with gradient descent is difficult,
IEEE Trans Neural Netw, 1994;5(2):157-66, doi: 10.1109/72.279181, 1994.

LSTM の仕組み

LSTM は，リカレント
ニューラルネットワークの一種



状態は「メモリセル」に記憶されている

２つの機能

- **記憶の持続 (constant error carousel)**
  1997年発表
- **記憶の忘却 (forget gate)**
  1999 年発表

LSTM の特徴

- **LSTM**は，**リカレントニューラルネットワーク**の一種
- **メモリセル**は，状態として，**同じ値の長期の記憶の保持**を可能とする
- **リカレントニューラルネットワーク**の**弱点**ともいわれる「**長期に及ぶ**過去の情報の保持が困難」であることを**解決**
- 1997, 1999年発表の技術．その後，手書き文字認識，音声認識，自動翻訳など数多くの応用

まとめ

- **時系列データ**は，**時間とともに，値が変化するデータ**
- **時系列データ**から，**周期性**や**トレンド**などを読み取ることができる
- **リカレントニューラルネットワーク**
  - **回帰**により，**過去の情報を保持**
  - **時系列データ**など**データの並びを扱う能力**を持つ

    予測，手書き文字認識，音声認識，言葉の理解，翻訳，テキスト生成，プログラム生成　など

  - **長期に及ぶ過去の情報の保持**のため，**LSTM** が考案された

4-6. 未来予測

① LSTM による予測

1. 使用するページ:

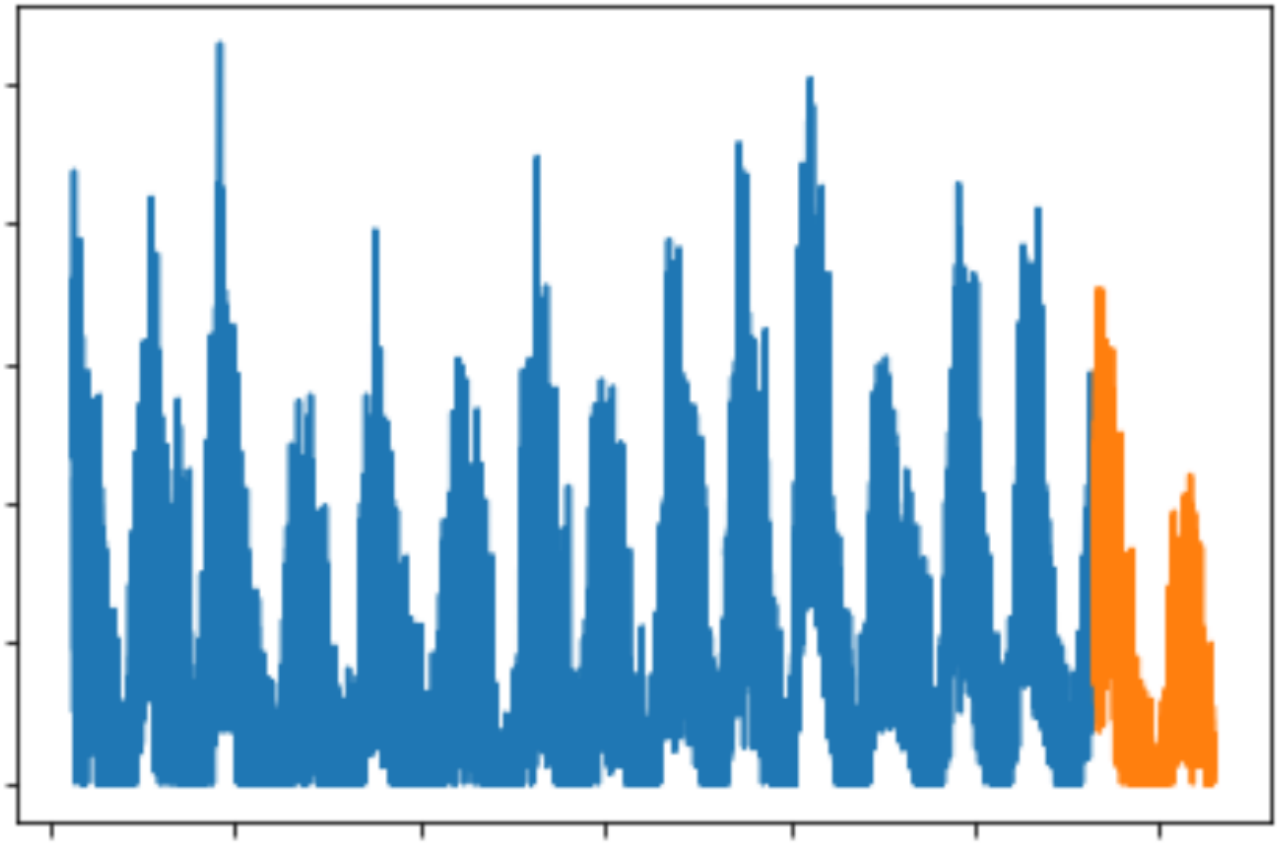
https://colab.research.google.com/drive/1qxh5l0iEPUm-QRTuEBSqBLd3V9KvqItK?usp=sharing

2. 必要な事前知識

LSTM もニューラルネットワークの一種であり，学習のさせ方などは，ふつうのニューラルネットワークと同じであること

3. 各自で実行すること

実際に実行し、予測を試す.



太陽の黒点数の変化　　予測

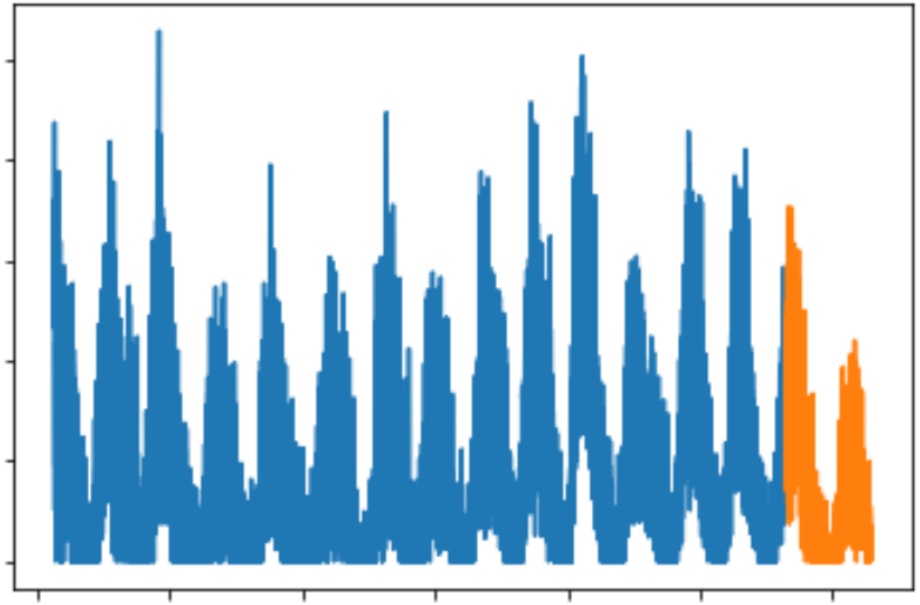
1848年～1999年のデータを用いて，2000年以降を予測
（ディープニューラルネットワークによる予測）

予測では，過去の観測値から「次の日（つまり一日分）の予測」を行うことを繰り返している．

みどころ

10日分のデータから
11日後を予測

これを
上手く使って

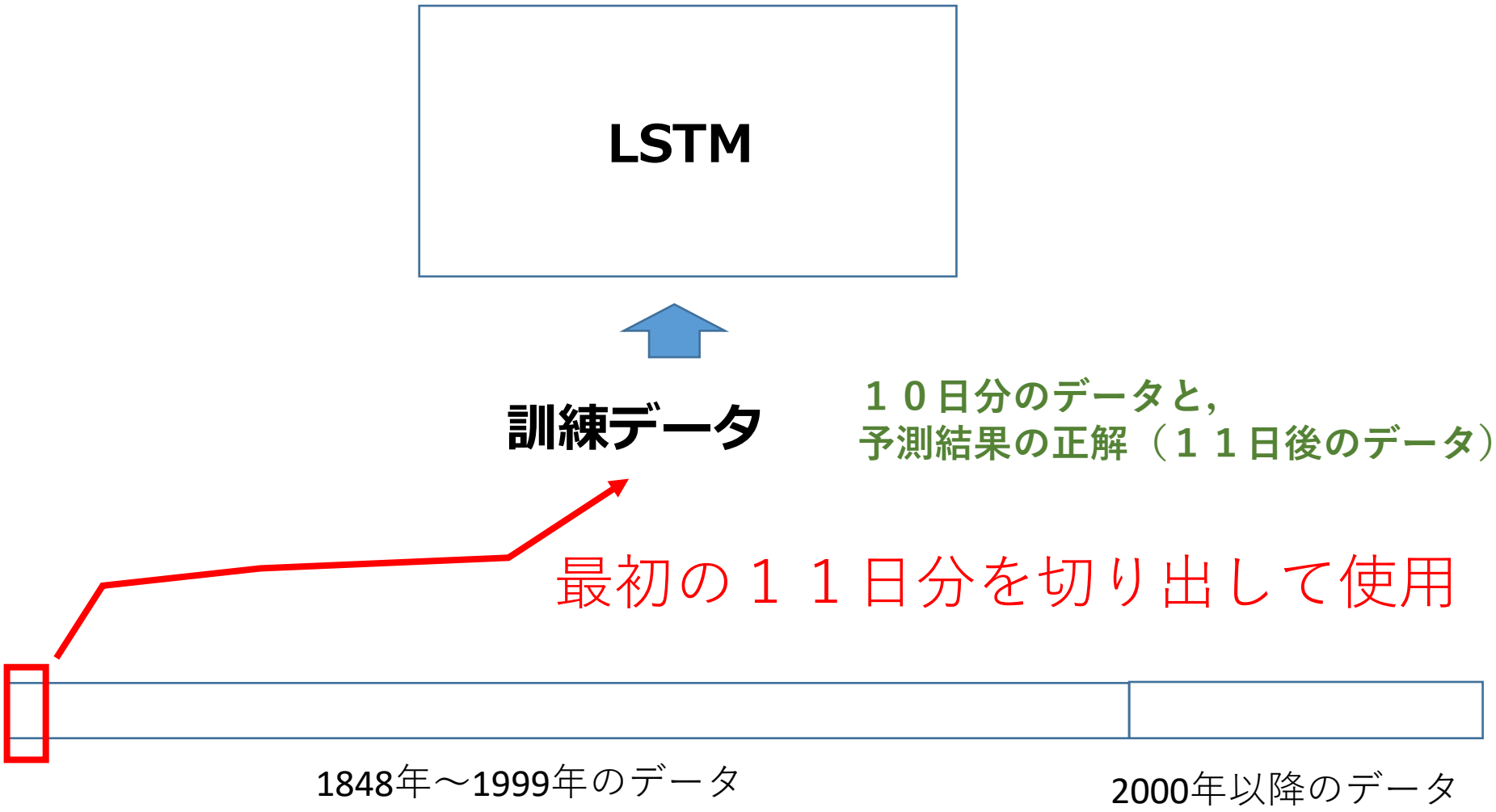


太陽の黒点
数の変化

予測

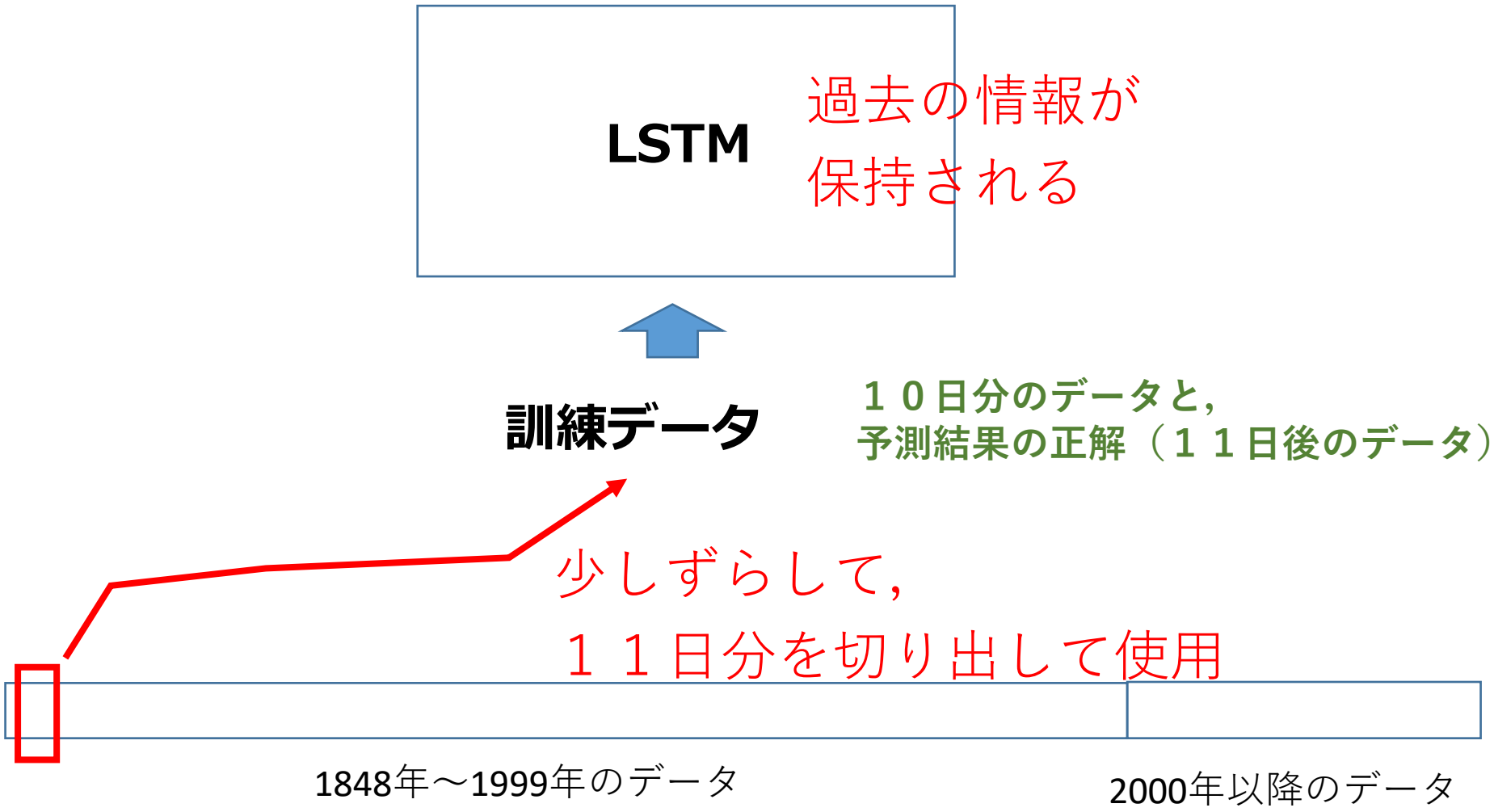
1848年～1999年のデータを用いて，2000年以降を予測

①



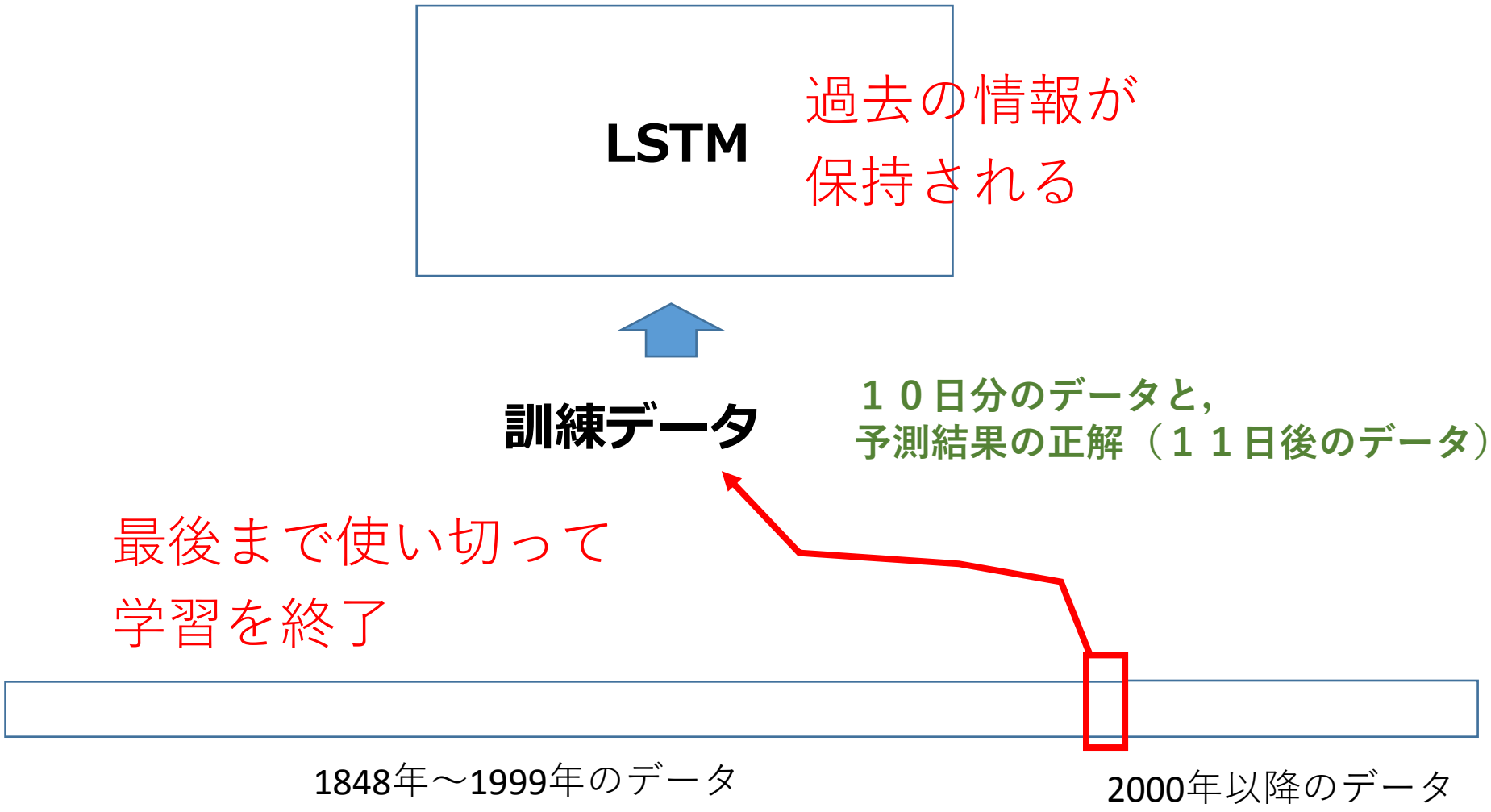
LSTM での学習の繰り返し

②



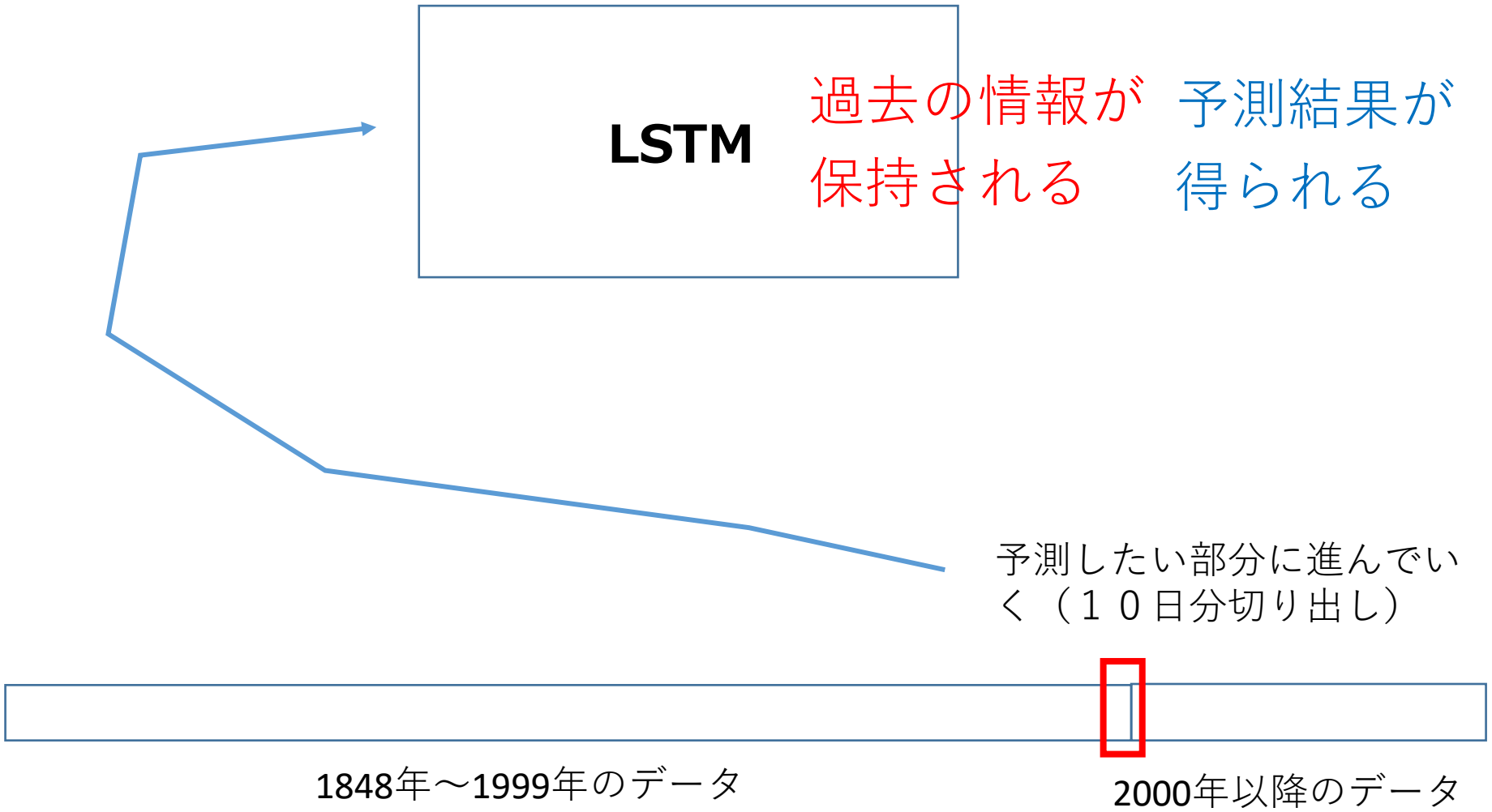
LSTM での学習の繰り返し

③



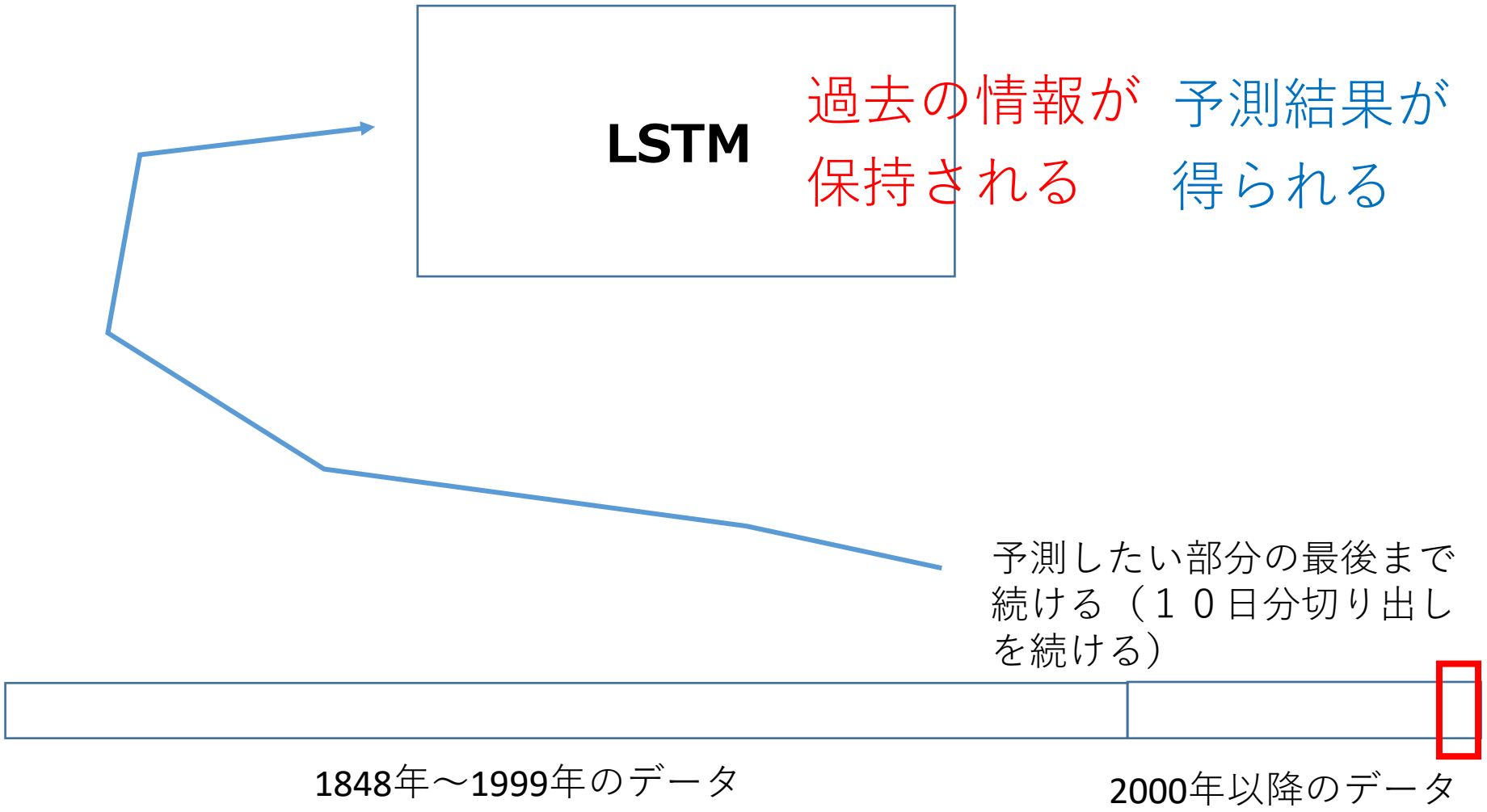
LSTM での学習の繰り返し

④

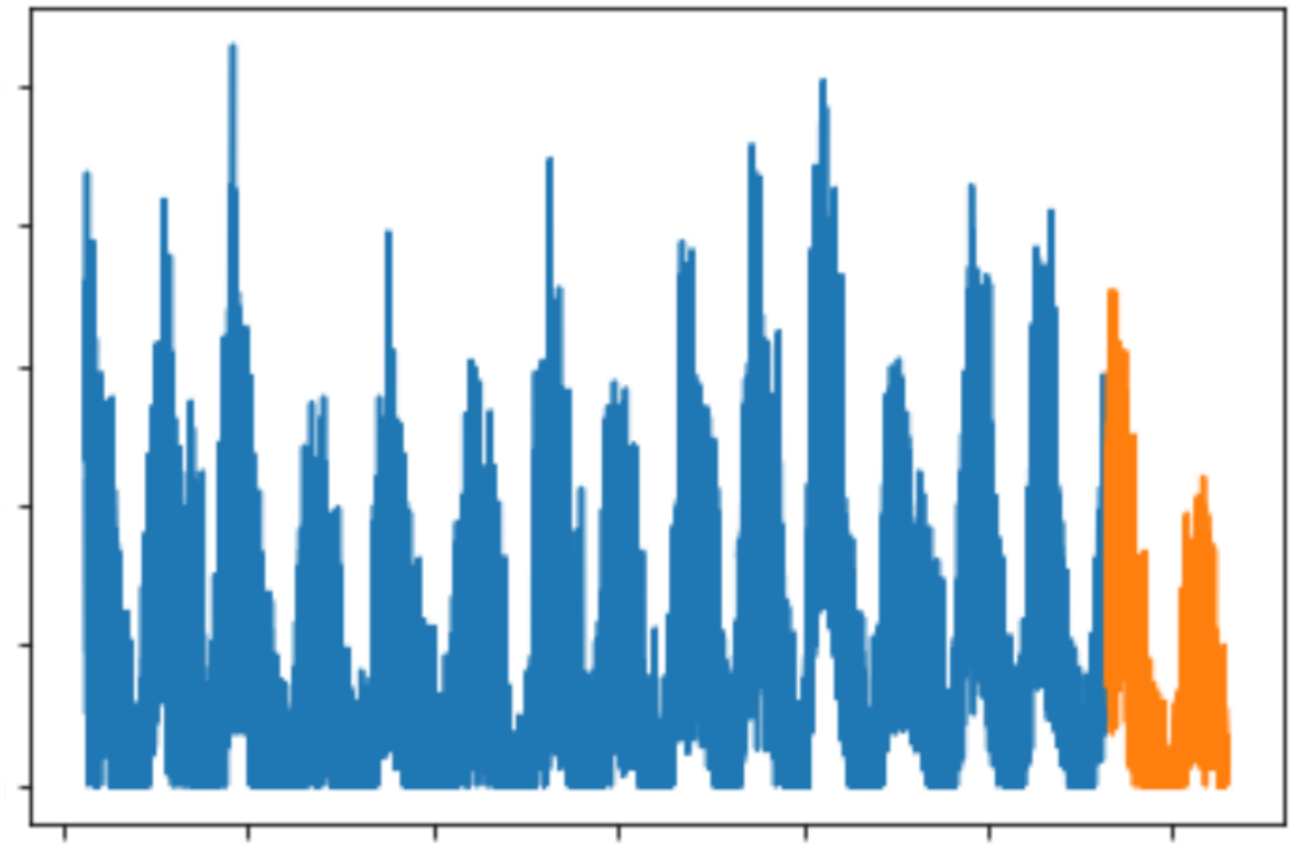

LSTM
過去の情報が
保持される
予測結果が
得られる
予測したい部分に進んでいく（１０日分切り出し）
1848年～1999年のデータ
2000年以降のデータ

LSTMによる予測

⑤


LSTM
過去の情報が
保持される
予測結果が
得られる
予測したい部分の最後まで
続ける（１０日分切り出し
を続ける）
1848年～1999年のデータ
2000年以降のデータ

LSTMによる予測



太陽の黒点数の変化　　予測

1848年～1999年のデータを用いて，2000年以降を予測
（ディープニューラルネットワークによる予測）

全体まとめ

- **教師あり学習**は，訓練データ(x, y)を用いて行われる．（入力xとその正解y）
- **学習**では，**訓練データ**を用いて繰り返しモデルが更新され、各更新後に**検証データを用いて性能が評価**されます。
- **時系列データ**は，**時間とともに値が変化するデータ**であり、**周期性**や**トレンド**を分析するために使用される．
- **時系列データの分析**に，**リカレントニューラルネットワーク**（RNN）が適用される場合がある。**過去の情報から未来を予測**することも可能．
- リカレントニューラルネットワークの一種であるLSTMは、状態の保持と忘却を行う機能を持つ．リカレントニューラルネットワークが持つ「長期に及ぶ過去の情報の保持が困難」であるという問題を解決．

授業の学ぶ意義と満足感

- 教師あり学習の概要を、実践的な演習で修得。機械学習の基本を体験的に理解。学びの楽しさを実感。
- 時系列データの特性、分析手法を学び、実社会での活用可能性を確認。
- リカレントニューラルネットワークやLSTMの仕組みを理解。AIによる時系列データ解析の実践的手法を知り、AIエンジニアとしてのスキルを向上。
- 実データを使った演習で、データから価値を引き出すAIエンジニアとしてのスキルを向上。学習意欲も向上。