



学習分析学会

Japanese Society for Learning Analytics

Keras/Python3で学ぶ ディープラーニング による時系列データ解析入門【実践編】

— 第2部 —

主催：学習分析学会

日時：2019年2月2日(土) 10：30～17：45

会場：株式会社フोटロン21F会議室

お願い

- 学内のゼミや授業などでお使いいただくときは、講師まで一報いただけると励みになります。
- 有償セミナーのテキストため、組織を超えて広く配布することをご遠慮ください。

第2部目次

はじめに 学習分析と時系列データ

第1章 高速フーリエ変換による周波数分析（講義）

第2章 高速フーリエ変換による周波数分析（プログラム）

第3章 音声ファイルの取り扱いと可視化

第4章 オートエンコーダによる異常検知（講義）

第5章 オートエンコーダによる異常検知（プログラム）

第6章 1次元畳み込みニューラルネットワークによる異常検知（講義）

第7章 1次元畳み込みニューラルネットワークによる異常検知（プログラム）

第8章 人の活動の識別：概要（講義）

第9章 人の活動の識別：データの基礎分析と前処理（プログラム）

第10章 人の活動の識別：ディープラーニングによる識別（プログラム）

タイムテーブル

① 10:30-12:10

昼休憩 (45分)

② 12:55-14:25

休憩 (10分)

③ 14:35-16:05

休憩 (10分)

④ 16:15-17:45

講師

佐藤伸也（さとうしんや） shinya.sato.mito@vc.ibaraki.ac.jp

茨城大学 全学教育機構 准教授

東京理科大学理工学部情報科学科卒業。同大学院博士課程を経て、2002年姫路獨協大学経済情報学部専任講師、2004年より同大学准教授。2006年から1年間キングスカレッジ・ロンドンにて客員研究員。現在、茨城大学にて、線形論理の計算的解釈に関する研究、プログラミング教育や情報教育の他、Pythonによる授業アンケートデータ等の集計などに携わる。PhD（イギリス・サセックス大学）。

卯木輝彦（うのきてるひこ） unoki@photron.co.jp

株式会社フोटロン 研究開発センター長

東京理科大学理学部応用数学科卒業。沖電気工業、TDKを経て、2006年よりフोटロン。この間、1995年東京理科大学理工学部情報科学科受託研究員、1996年より2年間、走行支援道路システム開発機構研究員。2017年からは、IMAGICA GROUP リサーチディレクターを兼務。

現在は、映像を活用した教育支援システムの研究開発に従事。

学習分析学会理事、教育システム情報学会 産学連携委員。

はじめに

この資料は、学習分析学会主催セミナー「Keras/Python3で学ぶ ディープラーニングによる時系列データ解析入門【実践編】」のテキストです。2017年度より学習分析学会で実施しているディープラーニングセミナーシリーズ「Keras x Python 3で学ぶディープラーニング」の第4回目にあたります。

今回は、時間粒度が比較的細かい時系列データに焦点をあてました。課題として異常検知と分類問題の2つを取り上げています。Pythonプログラムに慣れていない人でも、一日でディープラーニングのプログラムを把握し、実行できるようになることを目指しています。

テキストは講義形式を想定した解説部分と、実行可能なプログラムの部分に分かれています。プログラムは、Jupyter Notebook形式で作成しています。講師と一緒にプログラムをステップバイステップで実行することで、このテキストと同等のものが完成します。

講義では実行環境としてGoogle Colaboratoryを使います。Jupyter Notebookの動作環境があればローカルPC上で実行することもできます。プログラムコードは各自が自由に変更することができます。試行錯誤でパラメータを変更したり、別なデータを使ってみたりと、自由に試してみてください。

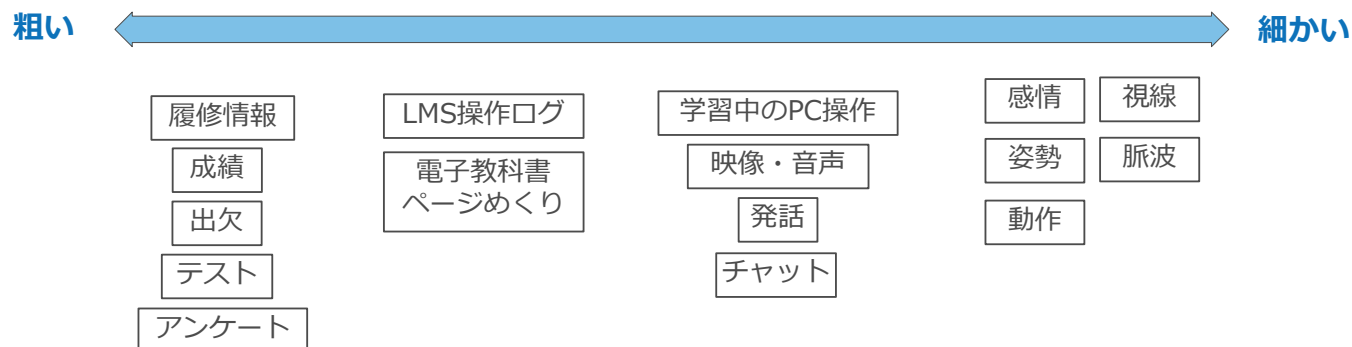
学習分析と時系列データ

学習分析で扱われるデータには、さまざまな時間粒度のものがあります。

本セミナーシリーズ（学習分析学会主催「Keras/Pythonで学ぶ…」）では、

- 2017年度は、時間とは無関係なデータを用いて成績予測問題や、画像分類問題を
 - 2018年12月は、年月日など比較的時間粒度の粗い時系列データでの回帰問題を
- 取り上げました。

今回は、より時間粒度の細かいデータを取り扱います。



第1章

高速フーリエ変換による周波数分析（講義）

第1章では、高速フーリエ変換（FFT）による周波数分析の基本的な考え方を把握します。周波数分析の結果は、ニューラルネットワークに入力する特徴量の一つとして利用することができます。

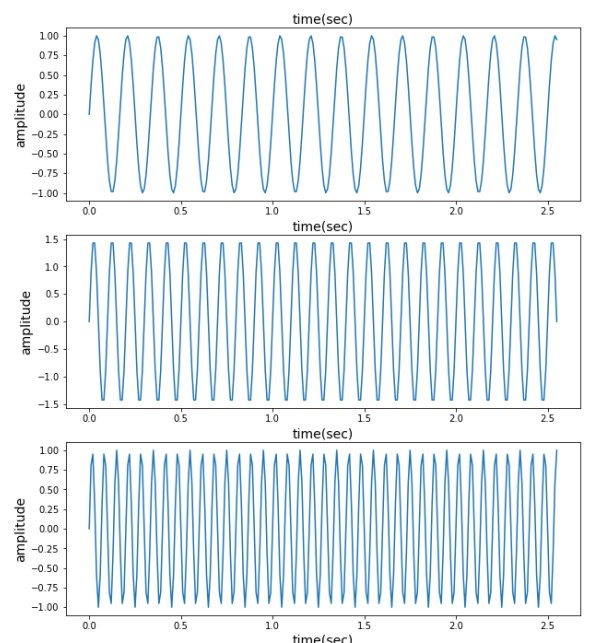
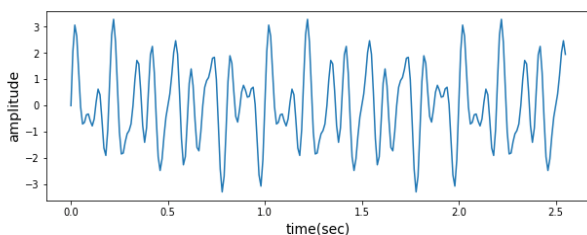
厳密性は追求せずに、概要を理解することが目標です。フーリエ変換の理論やFFTの内部処理などは本セミナーの範囲を超えるため説明しません。より進んだ内容は他の書籍等を参照してください。

FFTのプログラミングは、第2章で扱います。

周波数分析とは

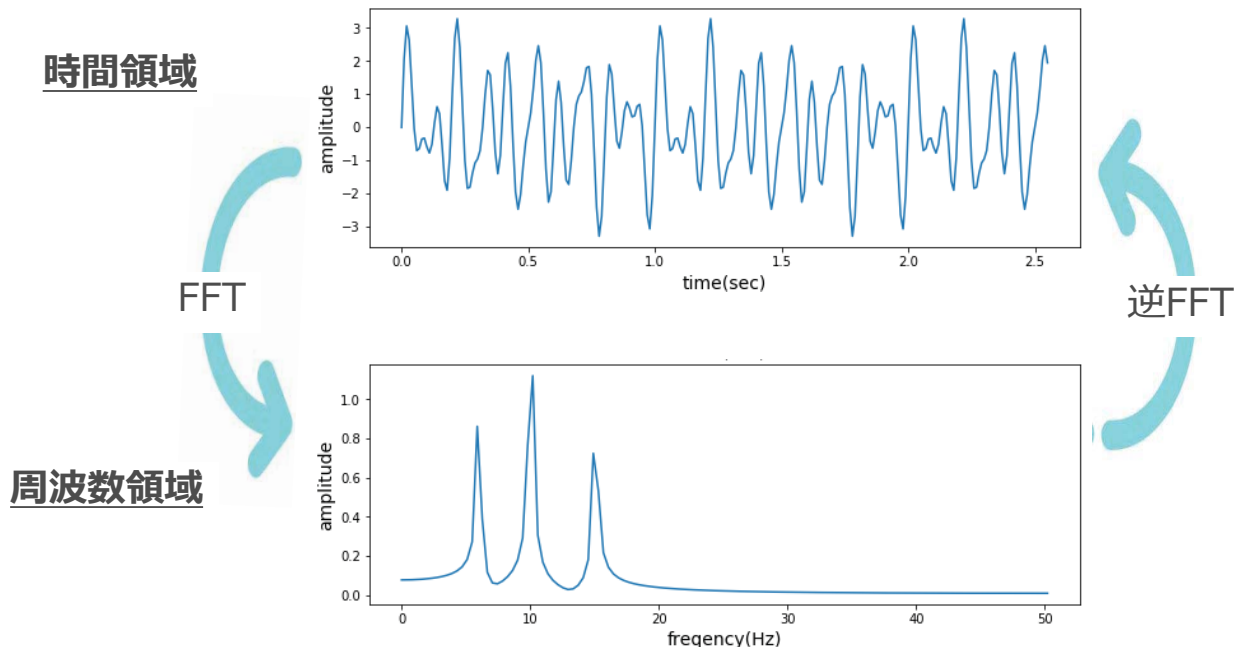
- 周期性のあるデータは、正弦波の和として表現することができます。
- どのような周波数の正弦波が含まれているのか調べることを「周波数分析」といいます。
- 周波数分析の手法の一つが「フーリエ変換」です。

複雑な波形も、正弦波の足し合わせで表現できる



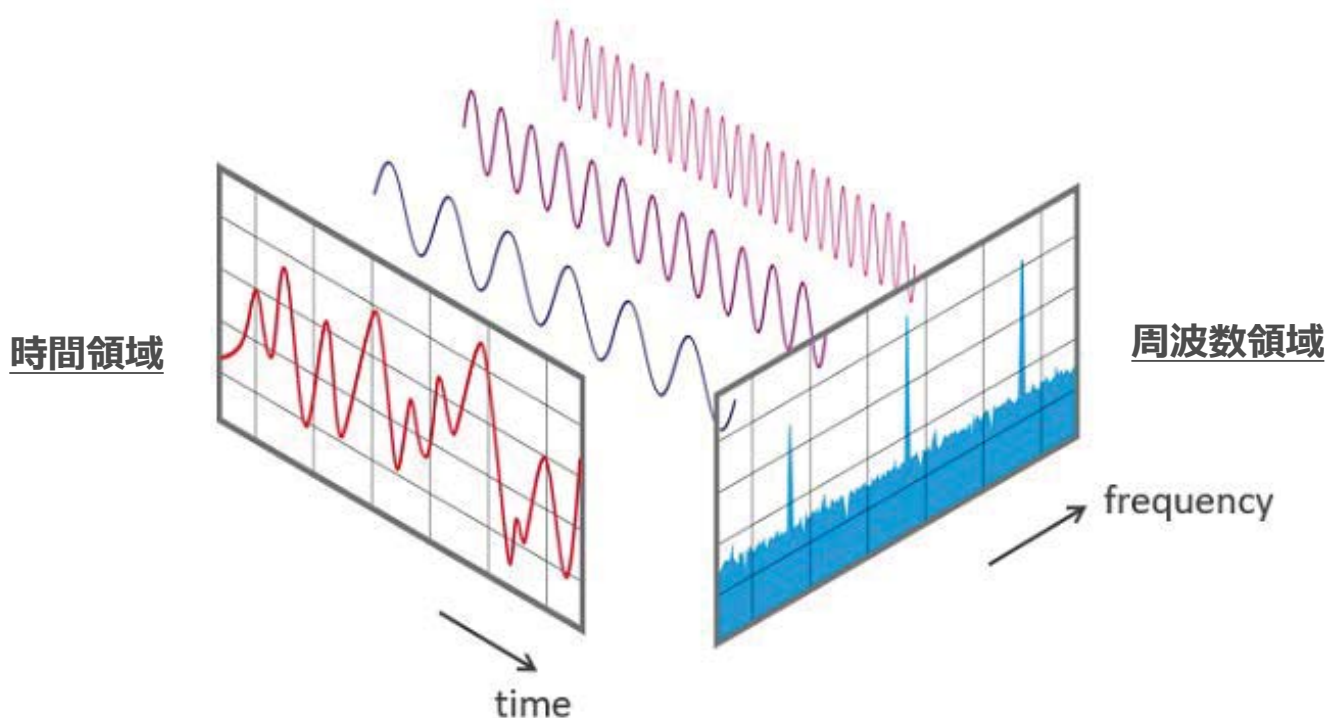
FFT (Fast Fourier Transform) 高速フーリエ変換

- 「離散フーリエ変換 (Discrete Fourier Transform、DFT)」は、離散化されたフーリエ変換です。デジタル信号の周波数分析に利用されます。
- FFTは、DFTを計算機上で高速に計算するアルゴリズムです。
「時間領域」の信号を「周波数領域」の信号に変換します。
- 周波数領域から時間領域への変換は、逆フーリエ高速変換 (Inverse FFT) でできます。



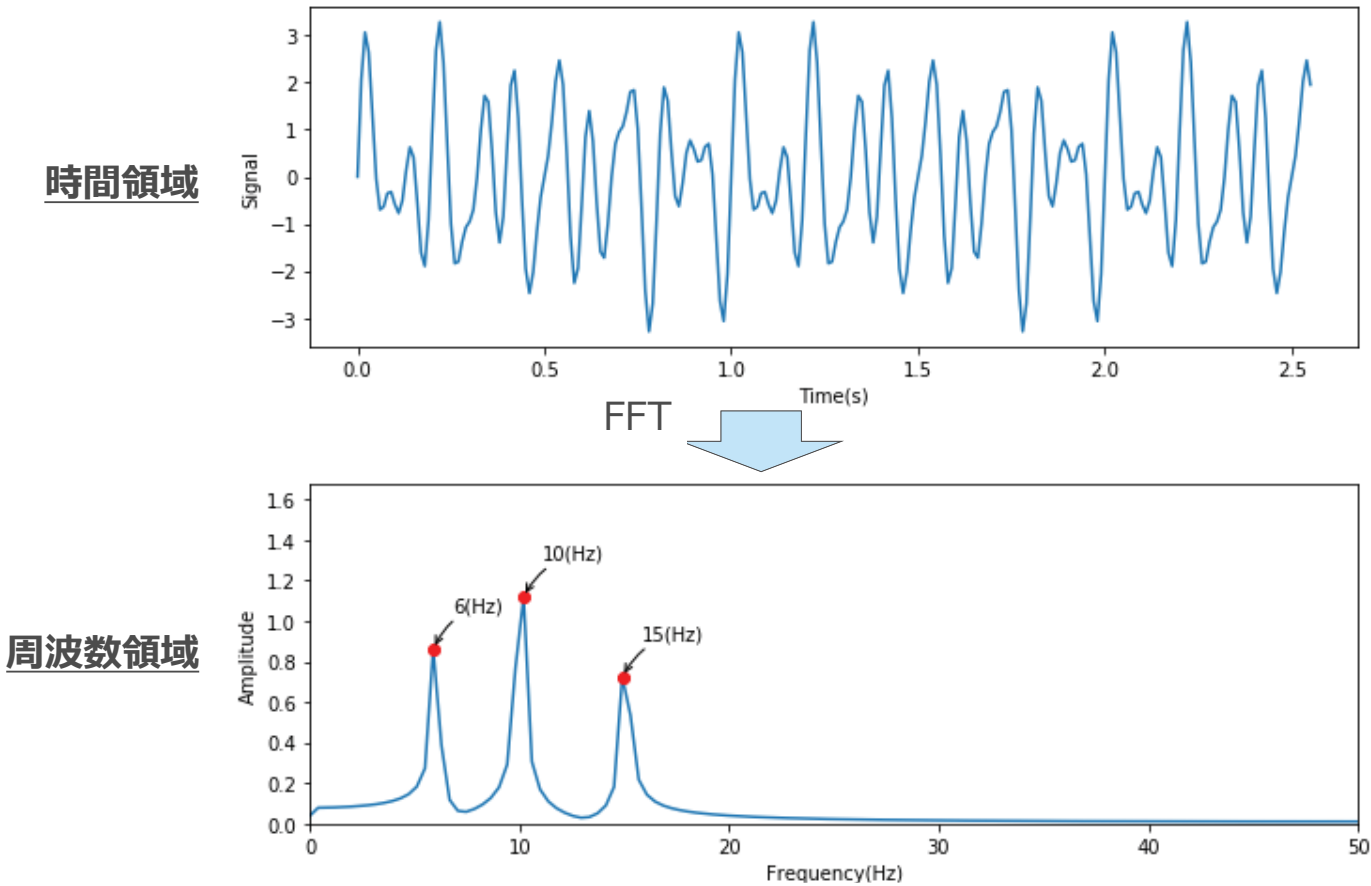
FFT (Fast Fourier Transformation) 高速フーリエ変換

時間領域と周波数領域信号の概観



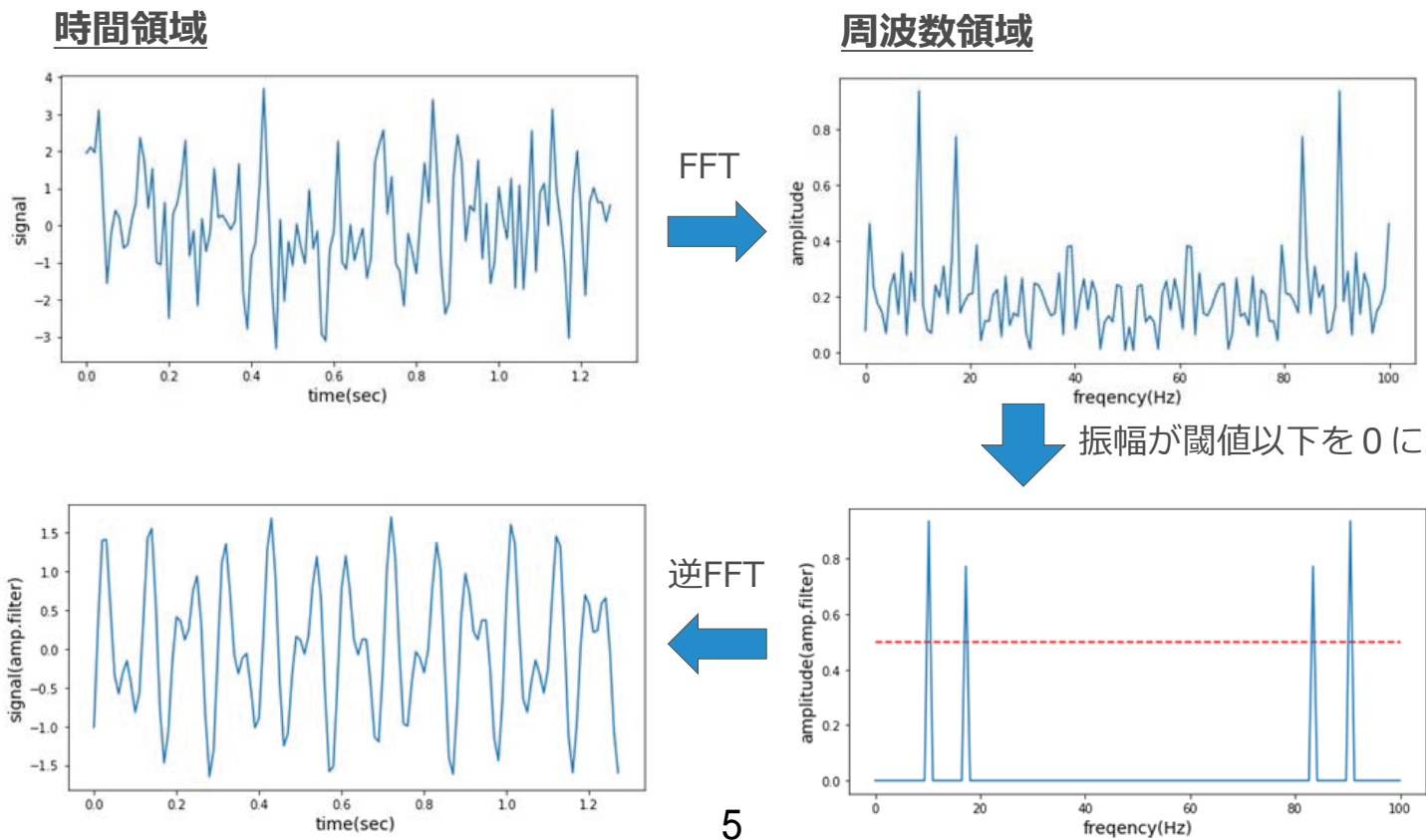
(<https://www.nti-audio.com/ja/%E3%82%B5%E3%83%9D%E3%83%BC%E3%83%88/%E6%B8%AC%E5%AE%9A%E3%83%8E%E3%82%A6%E3%83%8F%E3%82%A6/%E9%AB%98%E9%80%9F%E3%83%95%E3%83%BC%E3%83%AA%E3%82%A8%E5%A4%89%E6%8F%9B> より)

何に使えるの ①特徴量の抽出



何に使えるの ②ノイズ除去

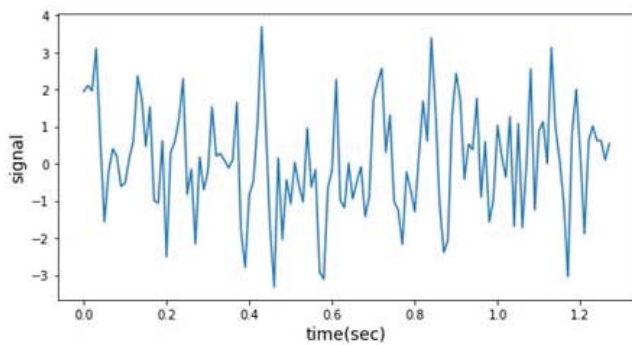
ノイズを除去することにより、時間領域での機械学習の精度が高まる



何に使えるの ③フィルター

ローパスフィルターの例

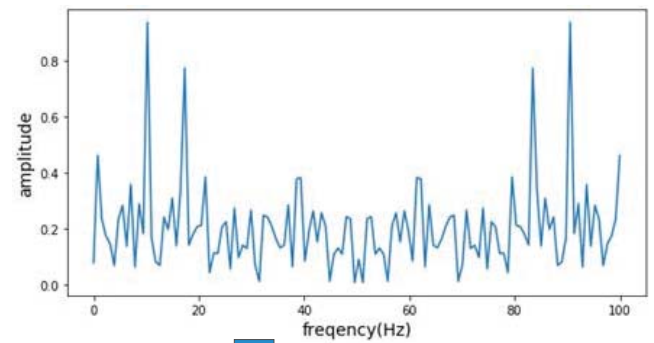
時間領域



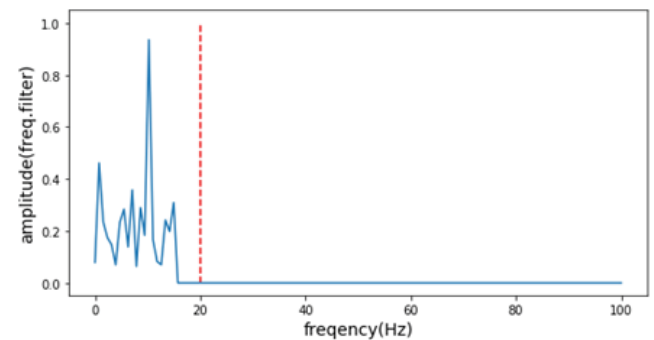
FFT



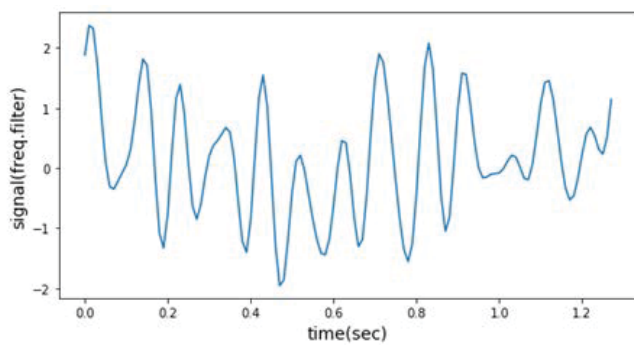
周波数領域



高周波成分をカット



逆FFT



参考文献

- 金谷健一 「これなら分かる応用数学教室—最小二乗法からウェーブレットまで」 共立出版(2003)
- E.Oran Brigham(著)、宮川、今井(訳)「高速フーリエ変換」 科学技術出版社 (1985)

第2章 高速フーリエ変換による周波数分析（プログラム）

第2章では、Numpyライブラリに含まれる高速フーリエ変換（FFT）モジュールを用いて、周波数分析を行います。

周波数分析は、周期性のあるデータの概要把握に役に立ちます。周波数分析の結果はニューラルネットワークの特徴量として利用することができます。機械学習の精度を上げるため、時系列データのノイズ除去や特定の周波数範囲のデータを削除するフィルター処理は有効です。

難しい数式展開を知らなくても、Pythonを使えば簡単に周波数分析ができることを体感してください。本章では、Matplotlibを使った可視化のテクニックについても紹介します。

学習項目

1. 準備
2. データの読み込みと確認
3. 高速フーリエ変換（FFT）
4. 逆フーリエ高速変換（IFFT）
5. ローパスフィルター
6. ピークの検出と可視化

使用するファイル

- 2_FFT.ipynb（このファイルです）
- data/fft.csv（データファイル）

データ

- 練習用に作成した波形データです。
- サンプリング周波数は100Hzで、256点のサンプルが含まれています。すなわち、10ms間隔（1秒あたり100個）で、2.56秒分のデータからなります。
- csvファイルの1列目は最初のサンプルを0秒とした時刻、2列目は対応する時刻の値です。

1. 準備

必要なライブラリの読み込み

- Pandas：csvファイルの読み込みに使います。
- NumPy：FFTの機能を使います。
- Matplotlib：グラフ描画ライブラリです。データの可視化に使います。Pandasのplotよりも細かい指定ができます。
- SciPy：高水準の科学技術計算ライブラリです。SciPyに含まれるSignalパッケージを使ってピーク値の検出を行います。

```
In [1]: 1 import numpy as np
        2 import matplotlib.pyplot as plt
        3 import matplotlib as mpl
        4 import pandas as pd
        5 from scipy import signal
```

グラフのデフォルト表示サイズの変更

セミナー中に画面をプロジェクターに投影するため、図の大きさを大きく変更しています。通常は不要です。

※ 以後の章でも断りなく同様の設定変更を行います。

```
In [2]: 1 mpl.rcParams['figure.figsize'] = 14, 4
```

カレントディレクトリの移動

- 本セミナーでは、データの保存場所としてGoogle Driveを利用します。
- Google Driveをマウントし、
- 相対パスとしてデータにアクセスできるように、カレントディレクトリを移動しておきます。
- Google Driveを使わない場合には、このセルをコメントアウトしてください。

※ 以後の章でも断りなく同様のカレントディレクトリの移動を行います。

```
In [3]: 1 # from google.colab import drive
        2 # drive.mount('/gdrive')
        3 # %cd "/gdrive/My Drive/Colab Notebooks/jasla_rensyuu_20190202"
```

2. データの読み込みと確認

CSVファイルからデータを読み込む

- csvファイルから Pandas の DataFrame としてデータを読み込みます。
- read_csv 関数を使います。
- パラメータとして、ファイル名を指定します。
- 変数 df に、データを読み込んでみましょう。
- csvファイルの1列目はDataFrameのindexとします。index_colパラメータで指定します。

```
In [4]: 1 df = pd.read_csv('data/fft.csv', index_col=0)
```

データのサイズの確認

- データの行と列の数を確認するには、shape を使います。
- shapeは関数ではないので、カッコが不要です。

```
In [5]: 1 df.shape
```

```
Out[5]: (256, 1)
```

- 読み込んだデータがSeriesではなく、1列のデータからなるDataFrameであることに注意が必要です。

データの一部を表示

- データの一部を表示するには、head関数や、tail関数を使います。
- head関数とtail関数は、表示する行数を引数にとります。（デフォルト値:5）

```
In [6]: 1 df.head()
```

```
Out[6]:
```

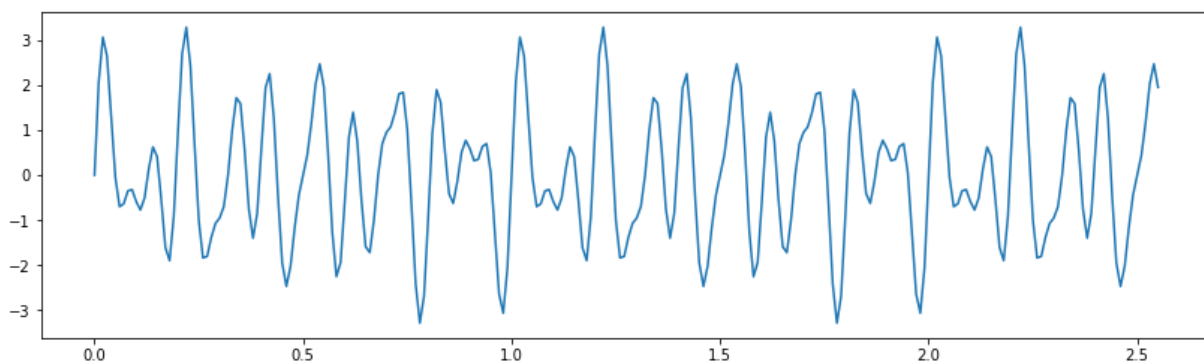
	f
0.00	0.000000
0.01	2.058819
0.02	3.062188
0.03	2.640429
0.04	1.291919

グラフの表示

- matplotlibライブラリのpyplotモジュールに含まれるplotメソッドを使います。
- matplotlib.pyplotは、pltという名前でimportしていました。
- plot関数にDataFrameを渡します。

```
In [7]: 1 plt.plot(df)
```

```
Out[7]: [<matplotlib.lines.Line2D at 0x1eea60a9c50>]
```



(メモ) 可視化ライブラリについて

本セミナーシリーズでは、2018年1月にはSeabornを、2018年12月にはPandasのplot関数を中心に紹介しました。今回は主にmatplotlibを使います。Seabornは機能豊富です。Pandas plotはDataFrameの可視化がとてもお手軽にできます。matplotlibは初学者にとってとっつきにくい部分がありますがPandasのplotよりもグラフの体裁を細かく指定できます。

3. 高速フーリエ変換 (FFT)

FFTの実行

- numpyライブラリに含まれるfftモジュールのfft関数を使います。(<https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.fft.fft.html>)
- FFTの処理結果は、numpyのndarrayで、値は複素数で取得できます。
- FFTでは、データの個数を2のn乗個にする必要があります。

```
In [8]: 1 F = np.fft.fft(df['f'])
```

FFTの出力結果から絶対値を求め、振幅のスケールを合わせる

- 結果を確認するためには、絶対値を求め複素数を実数に変換します。
- 信号の振幅のスケールに合わせるために、データの数で割って2倍します。
- 絶対値の計算には、numpyのabs関数を使います。

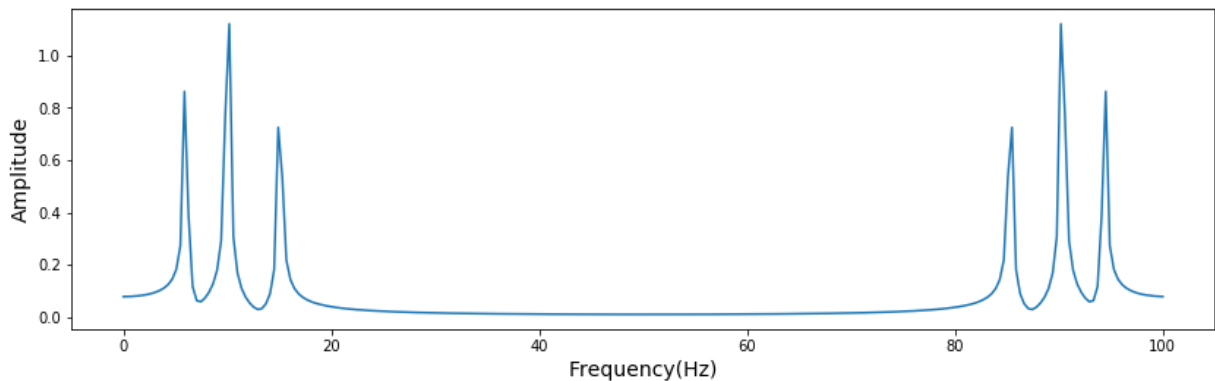
```
In [9]: 1 F_abs = np.abs(F[:F.size]) / df.size * 2
```

結果をグラフで表示

- matplotlibのplot関数を使って、結果を表示してみます。
- matplotlibでは軸のラベルを指定することができます。
 - 横軸のラベルは、xlabelで指定します。
 - 縦軸のラベルは、ylabelで指定します。
 - フォントのサイズも指定できます。
- 横軸を周波数にしましょう。
 - FFTは、サンプリング周波数までの値が出力されます。
 - 横軸の値を決めるため、numpyのlinspace関数で、等差数列を作ります。
 - linspace関数には、パラメータとして(数列の始点, 数列の終点, 分割数)を指定します。

```
In [10]: 1 plt.xlabel('Frequency(Hz)', fontsize=14)
2 plt.ylabel('Amplitude', fontsize=14)
3 fq = np.linspace(0, 100, df.size)
4 plt.plot(fq, F_abs)
```

Out[10]: [<matplotlib.lines.Line2D at 0x1eea60de470>]

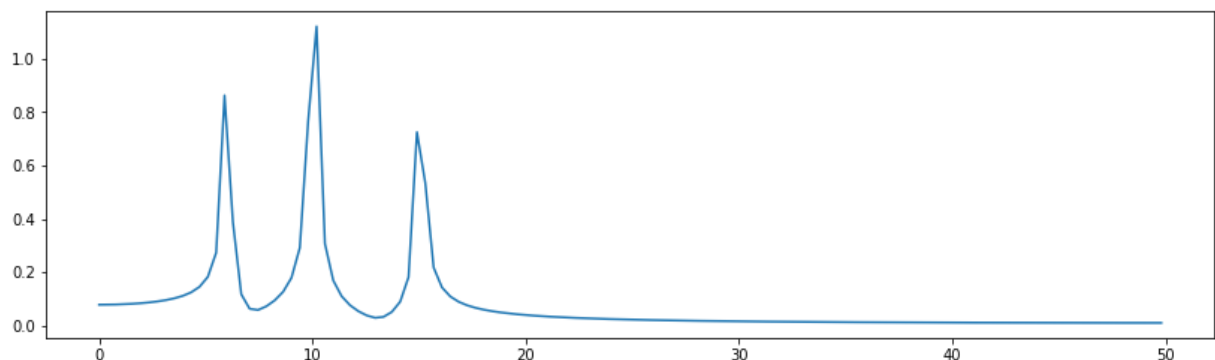


有効範囲のみを表示する

- FFTの結果の有効範囲は、サンプリング周波数の1/2までです。
- サンプリング周波数の1/2の値を、ナイキスト周波数と呼びます。
- FFTにより得られる結果は、ナイキスト周波数を境に左右対称となります。
- 右半分を左半分の虚像とよび、無視する必要があります。

```
In [11]: 1 N = int(F_abs.size / 2)
2 plt.plot(fq[:N], F_abs[:N])
```

Out[11]: [<matplotlib.lines.Line2D at 0x1eea6166cc0>]



以上で、時間領域の信号を、周波数領域に変換することができました。元の波形は、3つの周波数の正弦波から構成されていることが分かりました。

4. 逆フーリエ高速変換（IFFT）

FFTの結果に対して逆フーリエ高速変換（IFFT）を行うことで元の波形に戻ることができるか、確認してみましょう。

逆FFTの実行

- numpyライブラリに含まれるfftモジュールのifft関数を使います。(<https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.fft.ifft.html>)
- FFTの結果をそのままifft関数に渡して逆FFTを試みましょう。
- IFFTの結果は、複素数で戻ってきます。
- 実部を取り出してプロットしてみましょう。実部は.realプロパティで取得することができます。

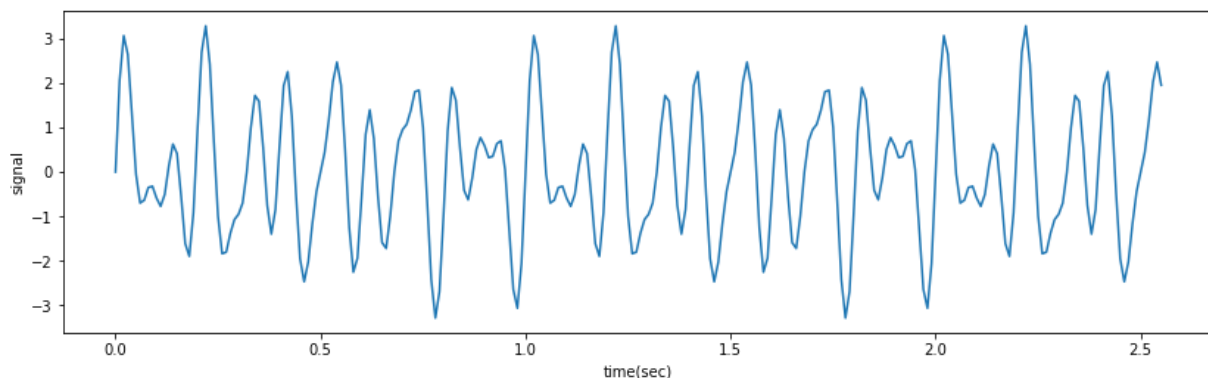
```
In [12]: 1 fi = np.fft.ifft(F)
        2 fi_real = fi.real
```

結果をグラフで表示

- matplotlibのplot関数を使って、結果を表示してみます。
- 横軸は時間です。単位を秒にしましょう。

```
In [13]: 1 plt.xlabel('time(sec)')
        2 plt.ylabel('signal')
        3 t = np.arange(0, fi_real.size/100, 1/100) # 時間軸を秒にする
        4 plt.plot(t, fi_real)
```

Out[13]: [matplotlib.lines.Line2D at 0x1eea61922e8]



FFTの結果を逆FFTすることで、元の波形とほぼ同じ形に戻ることが分かりました。

5. ローパスフィルター

FFTの結果（周波数領域）から、指定した値より高い周波数のデータカットします。カット後のデータを逆FFTをし、先ほどの結果と比べてみましょう。

データの書き換え

- 閾値を変数fcに設定します。ここでは8Hzとしました。
- 元のデータを残しておきたいので、FFTの結果Fを別な変数F1にコピーします。
- fcよりも高い周波数のデータをゼロに書き換えます。

```
In [14]: 1 fc = 8
        2 F1 = F.copy()
        3 F1[(fq > fc)] = 0
```

逆FFTの実行

- numpy.fft.ifft関数を使います。
- 実部を取り出して振幅を元のスケールに戻します。すでに虚像をカットしているのでスケールを合わせるために値を2倍にします。

```
In [15]: 1 F_ifft = np.fft.ifft(F1)
        2 F_ifft_real = F_ifft.real * 2
```

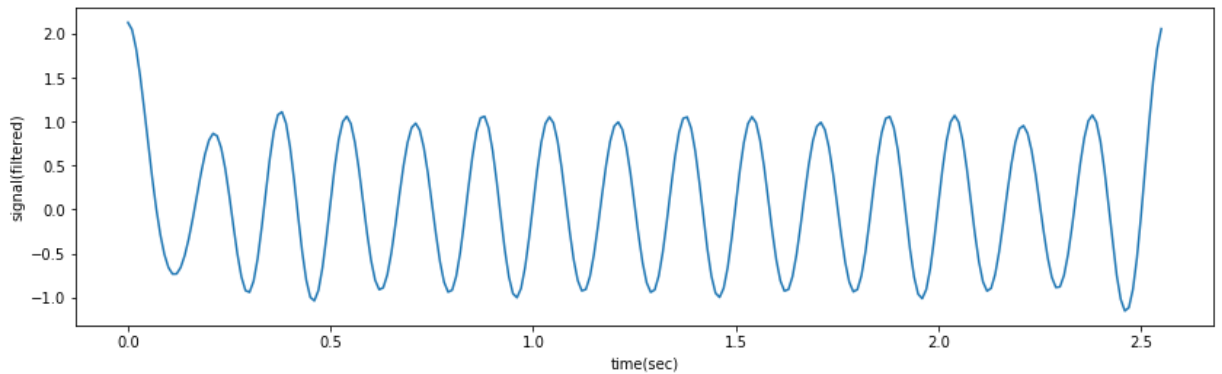
結果をグラフで表示

- matplotlibのplotを使って、結果を表示してみます。

- 横軸は時間です。単位を秒にしましょう。

```
In [16]: 1 t = np.arange(0, F_ifft_real.size/100, 1/100) # 時間軸を秒にする
2 plt.xlabel('time(sec)')
3 plt.ylabel('signal(filtered)')
4 plt.plot(t, F_ifft_real)
```

Out[16]: [<matplotlib.lines.Line2D at 0x1eea7235278>]



6. ピークの検出と可視化

SciPyのsignalパッケージにはピーク（極大、極小）を自動で検出する便利な機能があります。検出されたピーク値をmatplotlibのアノテーションを使って可視化してみましょう。

ピークの検出

- signal.argrelmax関数で極大値が、signal.argrelmin関数で極小値のインデックスが取得できます
- 今回は、signal.argrelmax関数を使って、極大値を検出してみましょう。
- orderパラメータで検出の際に比較をする両側のデータの範囲を指定できます。
- 結果を変数max_idxに代入しましょう。
- 結果はndarrayのtupleで戻ってきます。ndarrayとして取り出すため[0]を付けています。

```
In [17]: 1 max_idx = signal.argrelmax(F_abs, order=1)[0]
2
3 # 虚像(後半)を除外
4 max_idx = max_idx[(max_idx <= 100/2)]
```

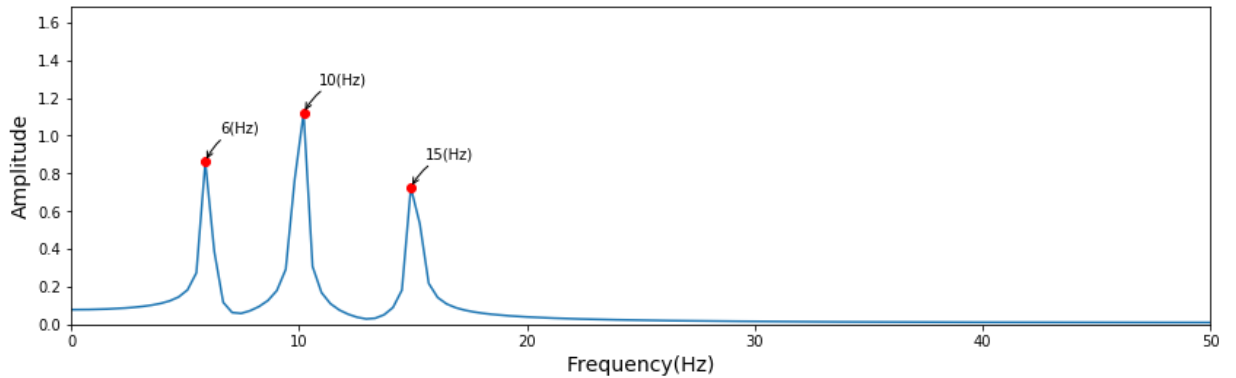
結果をグラフで表示

- matplotlibのplotを使って、結果を表示します。
- annotate関数でアノテーションをつけることができます。ピークに周波数をテキストで表示してみました。
(https://matplotlib.org/api/_as_gen/matplotlib.pyplot.annotate.html
(https://matplotlib.org/api/_as_gen/matplotlib.pyplot.annotate.html))

```

In [18]: 1 plt.xlabel('Frequency(Hz)', fontsize=14)
2 plt.ylabel('Amplitude', fontsize=14)
3
4 plt.axis([0, 100/2, 0, max(F_abs)*1.5])
5 plt.plot(fq, F_abs)
6 plt.plot(fq[max_idx], F_abs[max_idx], 'ro')
7
8 # グラフにピークの周波数をテキストで表示
9 for i in range(len(max_idx)):
10     plt.annotate(' {0:.0f} (Hz)'.format(np.round(fq[max_idx[i]])),
11                 xy = (fq[max_idx[i]], F_abs[max_idx[i]]),
12                 xytext = (10, 20),
13                 textcoords = 'offset points',
14                 arrowprops = dict(arrowstyle="->",
15                                   connectionstyle="arc3,rad=.2")
16     )

```



第3章 音声ファイルの取り扱いと可視化

第3章では、音声ファイル（wav形式）の読み込み方法と、可視化のテクニックについて学びます。Pythonで音声ファイルを読み込む方法は、数多くあります。ここでは、次の2つのライブラリによる方法を実行します。

1. SciPy (<https://docs.scipy.org/doc/scipy/reference/> (<https://docs.scipy.org/doc/scipy/reference/>))

- 高水準の科学技術計算パッケージ
- 統計、線形代数、信号処理、FFTなど、多くの機能がふくまれています。

2. LibROSA (<https://librosa.github.io/librosa/index.html> (<https://librosa.github.io/librosa/index.html>))

- 音楽と音声の解析のためのパッケージ
- テンポとビートを推定する機能があります。テンポやビートは、ニューラルネットワークの特徴量として利用することができます。

また、本章では特定のフォルダのファイルをまとめて読み込み、可視化をするテクニックについても紹介します。

学習項目

1. 準備
2. SciPyによるデータの読み込みと確認
3. LibROSAによるデータの読み込みと確認
4. フォルダ内のファイルをまとめて可視化
5. Matplotlibの二つの記法

使用するファイル

- 3_heartbeat-sounds.ipynb（このファイルです）
- data/heartbeat/フォルダ以下、複数のwavファイル(音声データファイル)

データセット

- 「Classifying Heart Sounds Challenge」は、2012年に開催されたデータ分析コンペです。コンペの課題の一つは心音の分類でした。(<http://www.peterjbentley.com/heartchallenge/> (<http://www.peterjbentley.com/heartchallenge/>))
- 本章では、Classifying Heart Sounds Challengeで使用されたデータ（812ファイル）のうちの一部の抜粋し、使用します。
- データは、iStethoscope ProというiPhoneアプリで取得した心音です。
- 1秒から30秒までの異なる長さのwav形式の音声ファイルです。

※Bentley, P. and Nordehn, G. and Coimbra, M. and Mannor, S., "The PASCAL Classifying Heart Sounds Challenge 2011 (CHSC2011) Results",

1. 準備

必要なライブラリの読み込み

- NumPy：グラフの軸の値の設定などに使用します。
- os：ファイル名の取得に使用します。OSに依存している機能を利用するためのモジュールです。
- glob：ファイル名の取得に利用します。ワイルドカードや正規表現を使ってファイルの検索ができます。
- Matplotlib：データの可視化に使用します。
- SciPy：音声ファイルの取り込みに利用します。SciPyのioパッケージに含まれるwavfileモジュールを使用します。
- LibROSA：音声ファイルの取り込みに利用します。

```
In [1]: 1 import numpy as np
        2 import os
        3 from glob import glob
        4
        5 from scipy.io import wavfile
        6 import librosa as lr
        7
        8 import matplotlib.pyplot as plt
        9 import matplotlib as mpl
```

グラフのデフォルト表示サイズの変更

```
In [2]: 1 mpl.rcParams['figure.figsize'] = 12, 4
```

カレントディレクトリの移動

```
In [3]: 1 # from google.colab import drive
        2 # drive.mount('/gdrive')
        3 # %cd "/gdrive/My Drive/Colab Notebooks/jasla_rensyuu_20190202"
```

2. SciPyによるデータの読み込みと確認

wavファイルの読み込み

- scipy.ioパッケージに含まれるwavfileモジュールを使って、wavファイルを読み込みましょう。
- read関数の引数にファイル名を指定します。
- 戻り値は、サンプルレートと、numpy array形式の数値データです。

```
In [4]: 1 rate, data = wavfile.read('data/heartbeat/extrahls__201102241217.wav')
```

```
In [5]: 1 rate
```

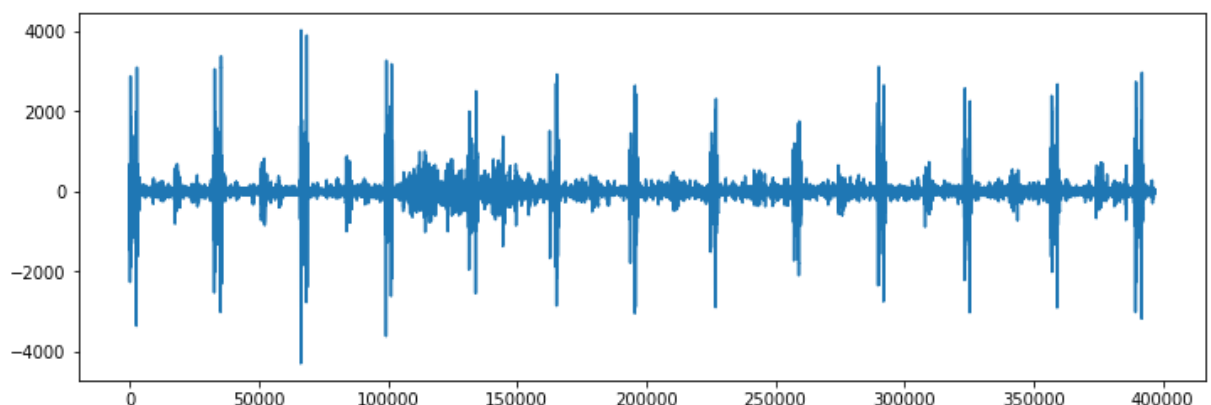
Out[5]: 44100

グラフを表示

- matplotlibのplot関数を使います。
- plot関数は、第1引数のデータがx軸に、第2引数のデータがy軸となります。第2引数を省略した場合には0から一連の番号がつけます。

```
In [6]: 1 plt.plot(data)
```

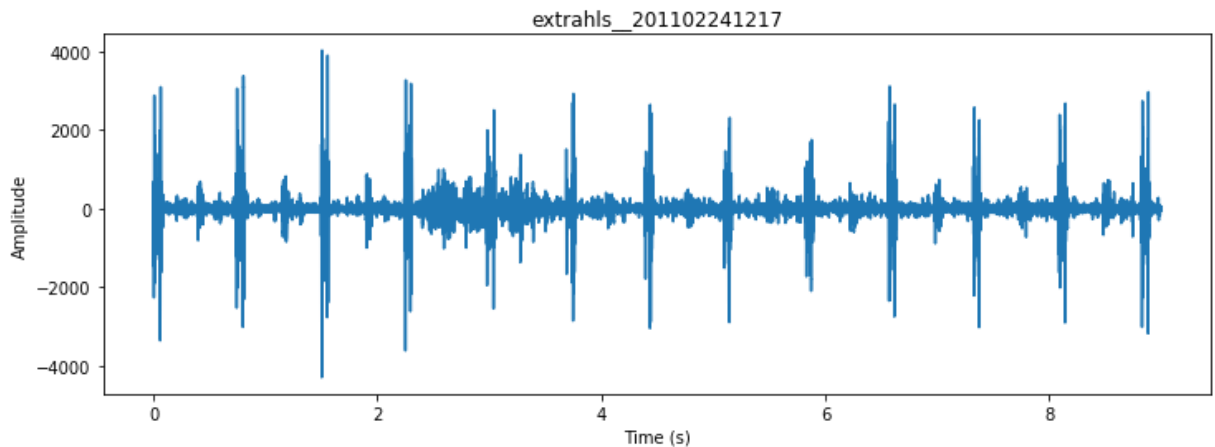
Out[6]: [<matplotlib.lines.Line2D at 0x1c3c6187048>]



- 横軸の単位を「秒」にしてみましょう。データの位置をサンプルレートで割ることで秒に変換ができます。
- タイトルや、軸のラベルも表示してみましょう。

```
In [7]: 1 sec = np.arange(0, len(data)) / rate
        2
        3 plt.title('extrahls_201102241217')
        4 plt.xlabel('Time (s)')
        5 plt.ylabel('Amplitude')
        6 plt.plot(sec, data)
```

Out[7]: [matplotlib.lines.Line2D at 0x1c3c64f1080]



3. LibROSAによるデータの読み込みと確認

wavファイルの読み込み

- load関数を使って、wavファイルを読み込みましょう。
- load関数の引数にファイル名を指定します。
- srは、サンプルレートを指定するパラメータです。元のファイルのサンプリングレートで出力するためにsrパラメータにNoneを指定します。デフォルトは22050で、デフォルトのままではダウンサンプリングがされてしまいます。
- 戻り値は、numpy array形式の数値データと、サンプルレートです。wavfile.readとは順番が異なることに注意が必要です。

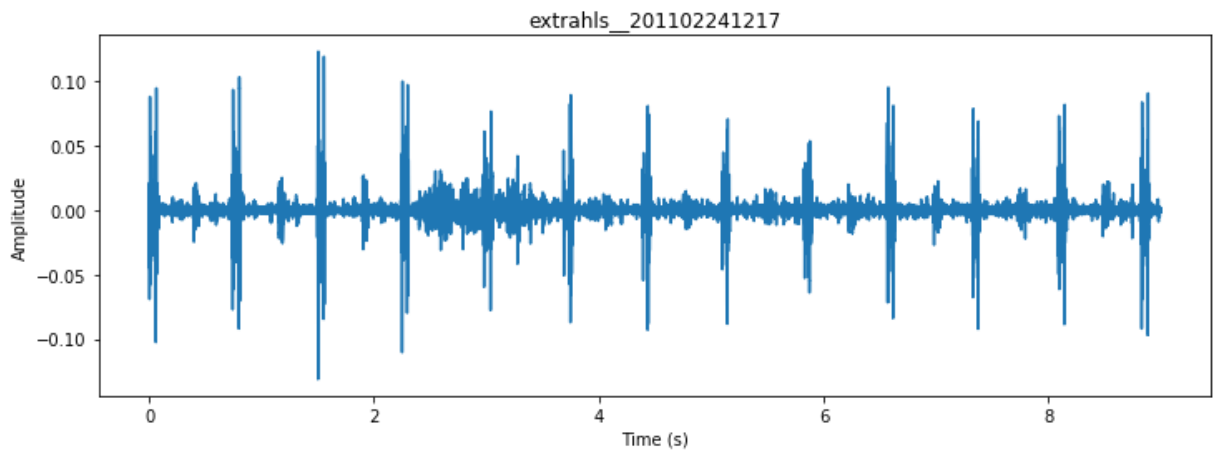
```
In [8]: 1 audio, sfreq = lr.load('data/heartbeat/extrahls_201102241217.wav',
        2                      sr=None)
```

グラフを表示

- matplotlibのplot関数を使います。
- 横軸の単位を「秒」にしてみましょう。

```
In [9]: 1 sec = np.arange(0, len(audio)) / sfreq
2
3 plt.title('extrahls_201102241217')
4 plt.xlabel('Time (s)')
5 plt.ylabel('Amplitude')
6 plt.plot(sec, audio)
```

Out[9]: [<matplotlib.lines.Line2D at 0x1c3c659d2e8>]



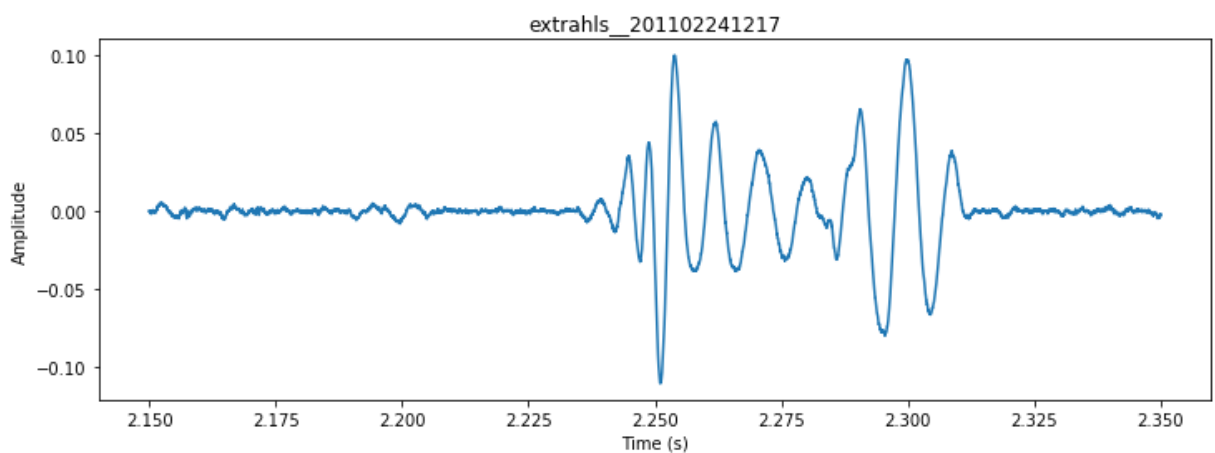
wavファイルの一部分だけ読み込む

- load関数は、offsetパラメータで読み込み開始位置を指定できます。
- durationパラメータで読み込む長さを指定できます。
- load関数の引数に、読み出しの位置と範囲を指定してみましょう。

```
In [10]: 1 offset = 2.15
2 duration = 0.2
3 audio, sfreq = lr.load('data/heartbeat/extrahls_201102241217.wav',
4                        sr=None,
5                        offset=offset,
6                        duration=duration)
```

```
In [11]: 1 sec = np.arange(0, len(audio)) / sfreq + offset
2
3 plt.title('extrahls_201102241217')
4 plt.xlabel('Time (s)')
5 plt.ylabel('Amplitude')
6 plt.plot(sec, audio)
```

Out[11]: [<matplotlib.lines.Line2D at 0x1c3c65433c8>]



4. フォルダ内のファイルをまとめて可視化

wavファイルのリストを作成

- globモジュールを使うと、ワイルドカードや正規表現を使って条件を満たすファイル名の一覧を取得できます。
- wavファイルのリストを作ってみましょう。

```
In [12]: 1 files = glob('data/heartbeat/*.wav')
```

- glob関数で取得したwavファイルの数を確認してみましょう。
- len関数により、リストの要素数を取得できます。
- 後で利用できるように、変数nにファイル数を代入しておきます。

```
In [13]: 1 n = len(files)
        2 n
```

```
Out[13]: 20
```

まとめて可視化

グラフタイトルのための準備

- ファイル名を各表のタイトルに使いましょう。
- os.path.basenameを使うと、ファイル名だけ取り出すことができます。
- 拡張子「.wav」を削除するため、各ファイル名の最後の4文字をスライスで取り除きます。

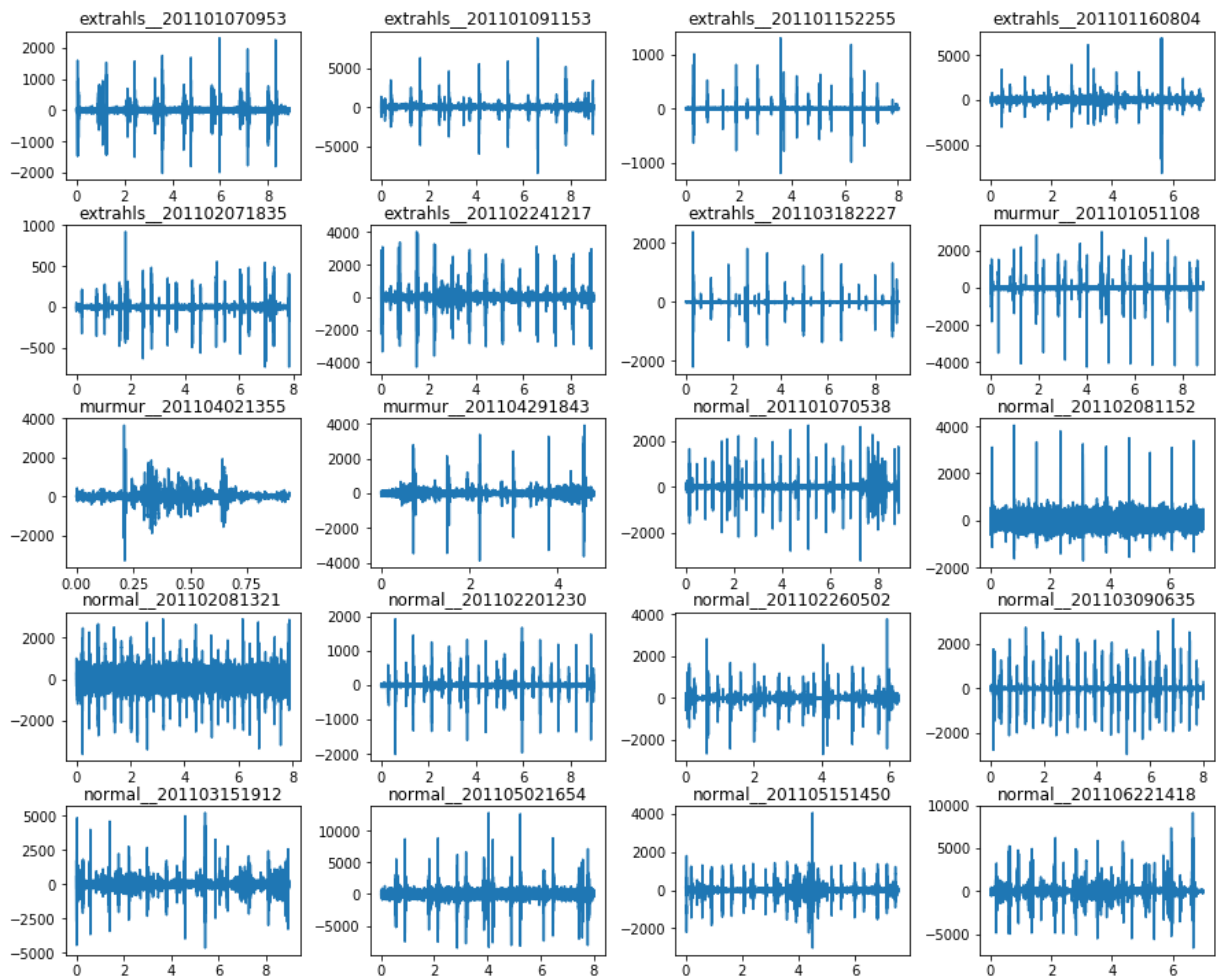
```
In [14]: 1 os.path.basename(files[0])[:-4]
```

```
Out[14]: 'extrahls__201101070953'
```

subplotでグラフ表示

- plt.subplotを使うと、一つの図に複数のグラフをまとめることができます。
- subplot関数のパラメータに、(行数, 列数, 何番目か)を指定します。カンマを省略することもできます。
- plt.subplot(2, 1, 1) と plt.subplot(211) は同じ意味になります。
- subplots_adjust関数を使うと、グラフ間の隙間を調整することができます。(デフォルト値は、0.2インチ)

```
In [15]: 1 plt.figure(figsize = (15, 15))
2 plt.subplots_adjust(wspace=0.3, hspace=0.3) # デフォルト0.2
3 for i in range(n):
4     rate, data = wavfile.read(files[i])
5     sec = np.arange(0, len(data)) / rate
6     plt.subplot(n//4+1, 4, i+1)
7     plt.plot(sec, data)
8     plt.title(os.path.basename(files[i])[:-4])
```



5. Matplotlibの二つの記法

「オブジェクト指向API」 vs 「Pyplot」

matplotlibにはグラフを作る際に二つの記法（流儀）があります。一つのプログラムでこれら二つが混在して使われていることも多く、初心者が混乱する原因になっています。本セミナーでは、後者を採用しています。確認をしながらインタラクティブに表示をする場合、後者が便利です。

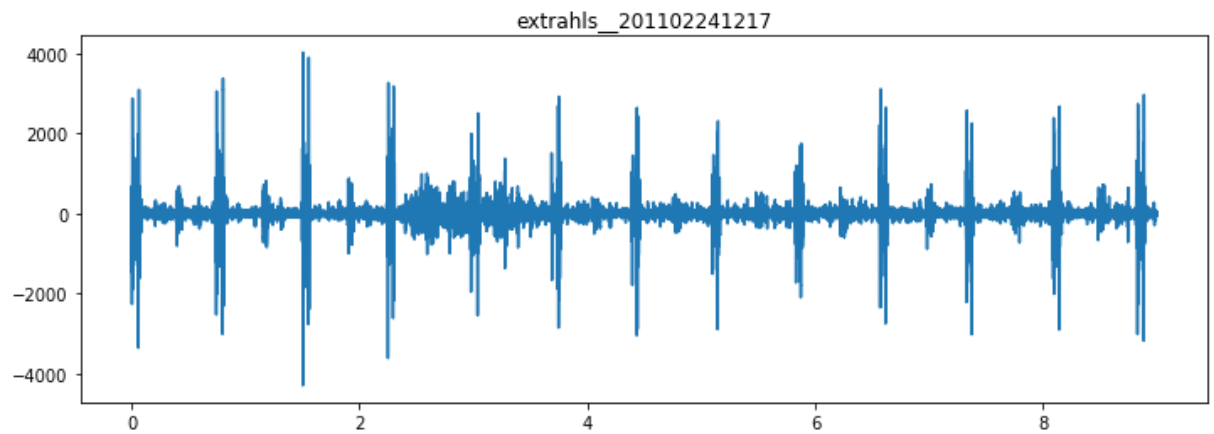
- オブジェクト指向インターフェース
 - 「fig, ax = plt.subplots()」などでオブジェクトを作成し、その後「ax.plot()」などを使う記法です。
- Pyplotインターフェース
 - 「plt.xxx」ですべて記述します。MATLABをまねた記法です。
 - current figureやcurrent axesと呼ばれるオブジェクトが自動で作成されます。

※（参考）A note on the Object-Oriented API vs Pyplot (<https://matplotlib.org/tutorials/introductory/lifecycle.html#a-note-on-the-object-oriented-api-vs-pyplot/>)

オブジェクト指向APIを使った例

```
In [16]: 1 rate, data = wavfile.read('data/heartbeat/extrahls_201102241217.wav')
2 sec = np.arange(0, len(data)) / rate
3
4 fig, ax = plt.subplots()
5 ax.set_title('extrahls_201102241217')
6 ax.plot(sec, data)
```

Out[16]: [<matplotlib.lines.Line2D at 0x1c3d480ab00>]



第4章

オートエンコーダによる異常検知（講義）

ディープラーニングにより時系列データの異常を検知する方法は、数多くあります。今回のセミナーでは2つの方法を取りあげます。第4章では課題の概要と、2つの方法のうちのひとつである「オートエンコーダによる異常検知」について説明します。

学習項目

1. 概要
2. オートエンコーダとは
3. オートエンコーダによる異常検知
4. オートエンコーダのプログラミング演習概要

1. 概要

細粒度時系列データを用いた学習支援

研究の例：

- 学習者の「集中度」「エンゲージメントの度合い」などをカメラやセンサーのデータなどから推定し、変化点をとらえて、教員、学習者へのフィードバック
- グループ学習の変化点を検出し、録画映像にアノテーションを付与し、振り返り支援

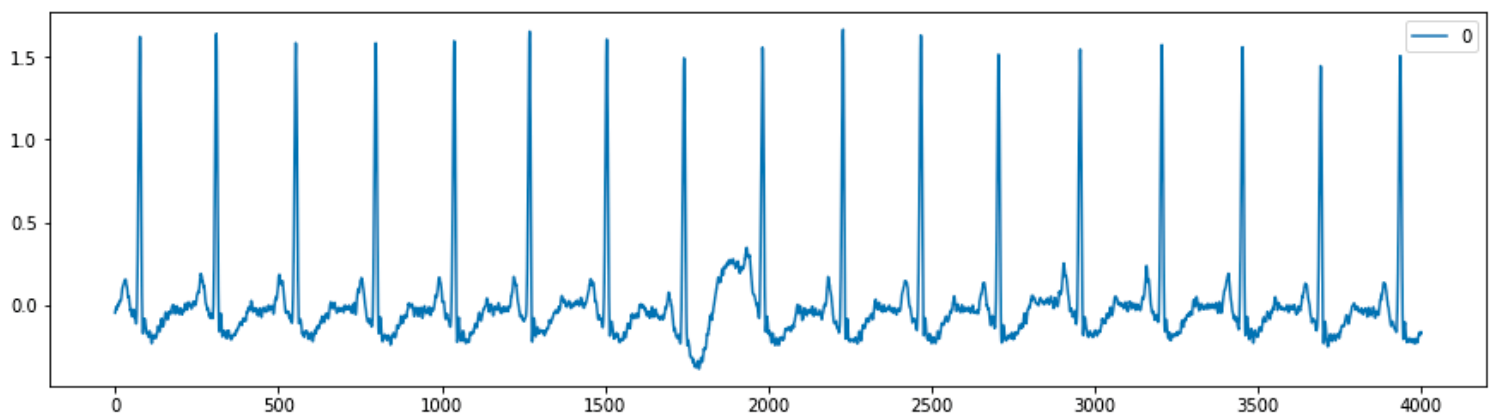


- 研究は増えている
- 公開されているデータはほとんどない



心電図データ

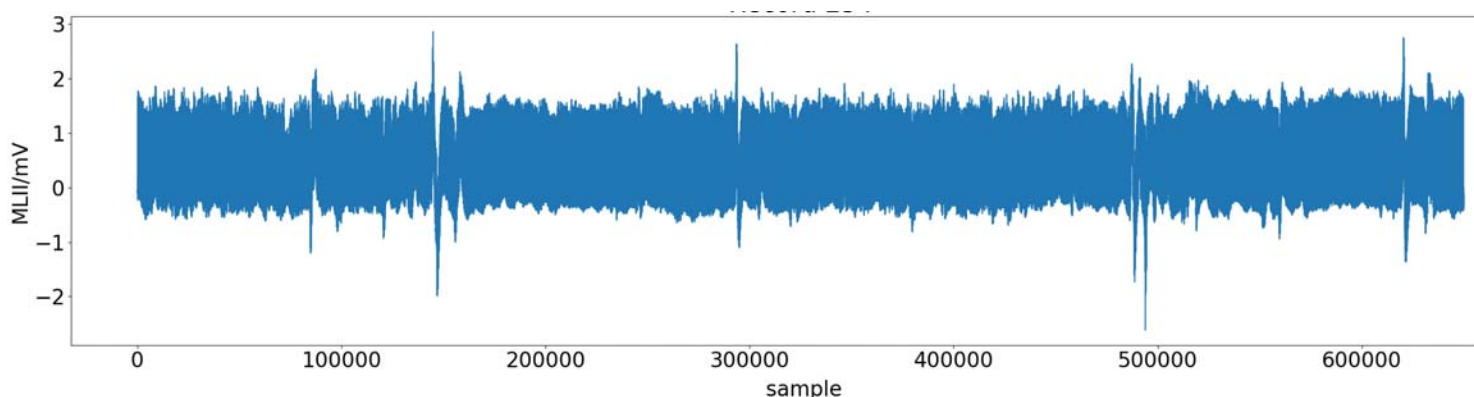
どこが異常かわかりますか？



手元にあるデータはだいたい正常

- 異常データを、十分に確保できない
- 異常の種類が、まんべんなく含まれているとは限らない
- 教師あり学習では、正解ラベルが必要

約30分の心電図データ



使用するデータセット

MIT-BIH Arrhythmia Database

(<https://www.physionet.org/physiobank/database/mitdb/>)

- Physionetで公開されている心電図データ
- 23歳から89歳までの男女47人の被験者から得られた48時間30分のデータです。
- 各データは、MLII、V1(またはV2,V5)の2チャンネルのデータが含まれています。
- 2人以上の心臓専門医が付与した約11万個のアノテーションデータとともに提供されています。アノテーションには、ピークの位置なども含まれています。
(<https://www.physionet.org/physiobank/database/html/mitdbdir/intro.htm#annotations>)
- Webページには確認すべきデータのポイントについての説明があります。
(<https://www.physionet.org/physiobank/database/html/mitdbdir/records.htm#234>)
- データを読み込むためのPythonライブラリ (wfdb) が提供されています。

今回のセミナーでは前処理済みのデータを、csvファイルで配布します

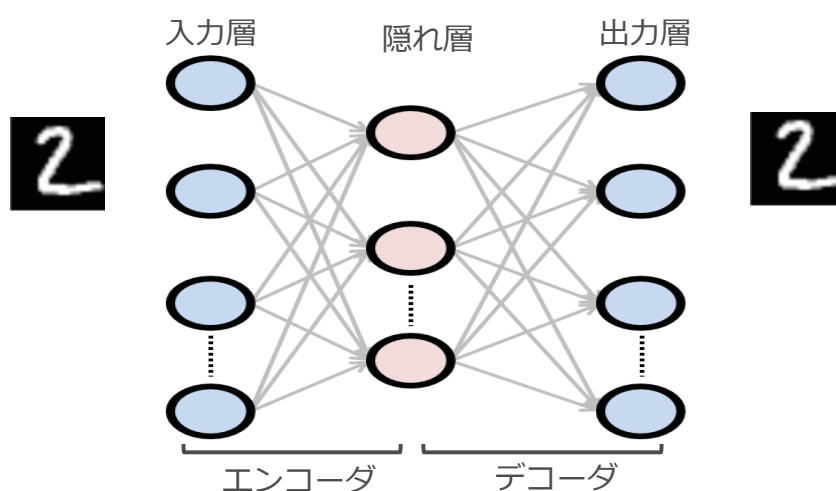
今回とりあげる2つの異常検知方法

次の2種類の方法で異常検知を実行します。

1. オートエンコーダ (Autoencoder、自己符号化器)
 - 訓練データには異常データが含まれていないという前提で、学習をする
 - 「正解ラベル」が不要
2. 1次元畳み込みニューラルネットワーク (1D Convolutional Neural Network, 1D-CNN)
 - 学習するデータに正常か異常かの「正解ラベル」が必要

2. オートエンコーダとは

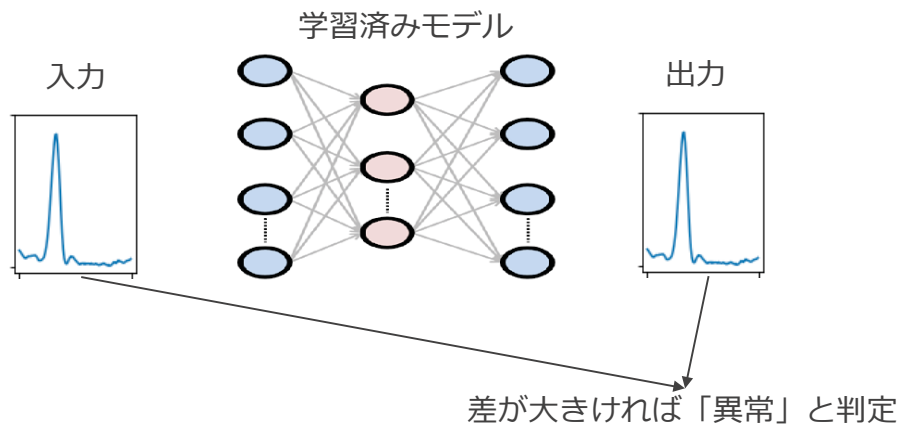
入力と出力が同じになるように、ニューラルネットを学習させる。



- 入力層のユニット数と出力層のユニット数が同じ、教師データを入力データと同じにする。
- 学習アルゴリズム等は一般のニューラルネットと同様。
- 入力層から隠れ層までを「エンコーダ」、中間層から出力層までを「デコーダ」と呼ぶ。
- 隠れ層のユニット数が入力層のユニット数より少ない場合、データを低次元に圧縮していることになる。
- 隠れ層のユニット数を入力層のユニット数より多くし、学習時に「制約」を設けてほとんどの中間層のユニットの値を0にするモデルを「スパースオートエンコーダ」と呼ぶ。

3. オートエンコーダによる異常検知

**未知のデータを学習済みのモデルに入力し
入力と出力の差の大きいものを異常と判定する。**



- 正常なデータで学習をしたモデルは、正常データを復元できる。
- モデルが学習していないものは復元できない。

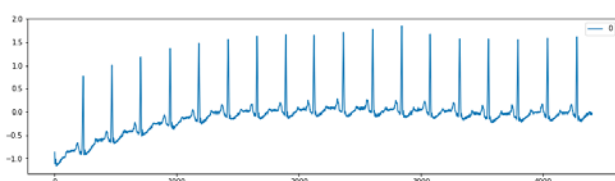
4. オートエンコーダのプログラミング演習概要

目標

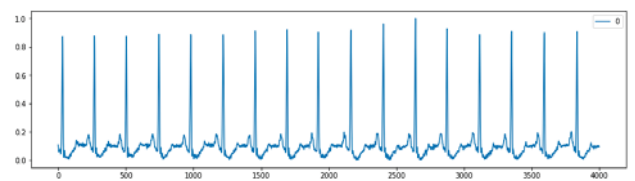
- MIT-BIH Arrhythmia Databaseで公開されている心電図データから、異常（不整脈）を検出します。

データセット

- ID234のデータのみを使います。
- 2チャンネル分（MLII、V1）のデータのうち、MLIIのみを使用します。
- 約65万点のデータから、訓練用とテスト用にそれぞれ4000点（約11秒）のデータを切り出して、さらに1/2にダウンサンプリングして使用します。
- 前処理として、トレンドの除去、および0から1の範囲へのスケーリング処理を行い、csvファイルに保存しました。
- 欠損値や、数値以外のデータは、含まれていません。



体動や呼吸に伴い基線が揺れる



- トレンドの除去
- スケーリング

ディープラーニングのプログラムの流れ（再掲）

■ディープラーニングのプログラムは、次のような流れで実行します。

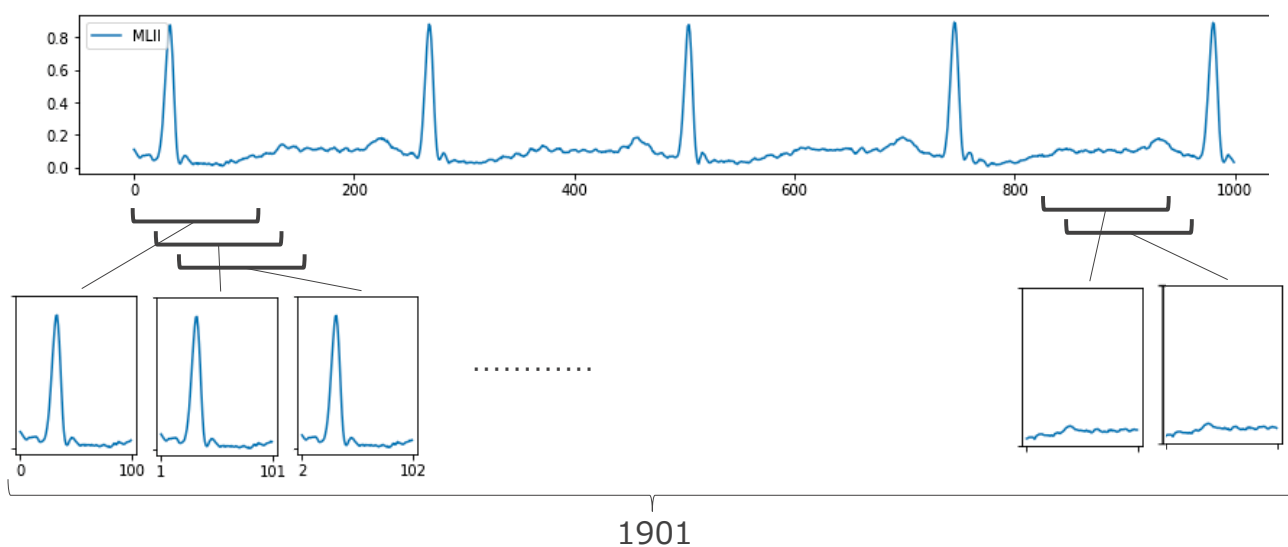
- ① データ準備：欠損値の処理、正規化など
- ② モデル定義：レイヤー構成、ノード数などの決定
- ③ パラメータの設定：損失関数、最適化手法等
- ④ トレーニング：実行回数などを決める
- ⑤ 評価：テストデータで評価
- ⑥ 必要に応じて、②に戻る

以下では、上記流れ①～⑤の各項目をより詳しく確認しておきます。

① データ準備

1. モデル入力用データの作成

- 開始位置を1点ずつずらしながら、100点ずつ切り出します。（ウィンドウサイズ100）
- 2000点のデータなので、1901行×100点のデータができる。



② モデル定義

- Kerasには2つの利用可能なモデルがあります(<https://keras.io/ja/models/about-keras-models/>)
 - Sequentialモデル
 - functional APIとともに用いるモデルクラス
- 今回は、functional APIを使います。
 - functional APIは複数の出力があるモデルや有向非巡回グラフ、共有レイヤーを持ったモデルなどの複雑なモデルを定義するためのインターフェースです。
- functional APIの使い方
 1. Input関数で入力データの次元を指定する。shape=(784,)の場合、784次元のデータが任意の個数あるデータを入力とする。
 2. 加えたいレイヤー関数に、それ以前のレイヤー情報を指定して、変数に入れます。
レイヤーとは、全結合層であればDenseなど (<https://keras.io/ja/layers/core/>) のことです。
 3. Model関数で入力と出力を指定します。

Keras公式ドキュメントに書かれている全結合層の例

```
inputs = Input(shape=(784,))
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)
model = Model(inputs=inputs, outputs=predictions)
```

784の入力次元、64ユニットの中間層が2層、10ユニットの出力

今回のモデル

- スパースオートエンコーダを使います。
 - 各層のユニット数は、入力層100、隠れ層160、出力層100とします。
 - 正則化によって、レイヤーの出力に制約を課することができます。
中間層の正則化項にL1出力正則化を用います。 (<https://keras.io/ja/regularizers/>)

```
input_layer = Input(shape=(100,))
encoded = Dense(160, activation='relu',
                activity_regularizer=regularizers.l1(1))(input_layer)
decoded = Dense(100, activation='relu')(encoded)
autoencoder = Model(inputs=input_layer, outputs=decoded)
```

③ パラメータの設定

- モデルの学習を始める前に、compile関数を用いてどのような学習処理を行なうかを設定します。
- compileでは、最適化アルゴリズム、損失関数などを指定します。

例：損失関数に平均二乗誤差、最適化アルゴリズムにAdamを指定した場合

```
model.compile(loss='mean_squared_error',  
              optimizer='adam')
```

今回の設定：損失関数にバイナリクロスエントロピー、最適化アルゴリズムにAdaDelta

```
autoencoder.compile(loss='binary_crossentropy',  
                   optimizer='adadelta',)
```

④ トレーニング

- 入力データと教師データを使って、モデルのトレーニング（学習）を行います。今回はオートエンコーダなので、入力データと教師データに同一のデータを使用します。
- モデルをトレーニングには、fit関数を使います。
- fit関数には、入力データと教師データ、ミニバッチのサイズ(batch_size)、エポック数などを渡します。
- verbose=0とすると、トレーニング状況のログを画面に表示しません。

例

```
model.fit(  
    X_train,  
    y_train,  
    epochs=100,  
    batch_size=32,  
    verbose=1)
```

オートエンコーダの設定

```
autoencoder.fit(X_train,  
                X_train,  
                epochs=1000,  
                batch_size=None,  
                verbose=0)
```


⑤ 評価

- テストデータを使って、トレーニング済みモデルの評価を行います。
- predict関数を使い、トレーニング済みモデルに未知の新しいデータ（テストデータ）を適用します。predict関数の戻り値は、Numpy Arrayです。
- predict関数の結果と、テストデータの比較をして、差が大きいものを異常と判定します。
- 判定基準は対象とする問題領域に応じて、適当に設定する必要があります。今回は、差の最大値の80%以上をとる点を異常とみなし、検出します。

例

```
y_pred = model.predict(X_test)
```


第5章 オートエンコーダによる異常検知（プログラム）

第5章では、オートエンコーダによる異常検知のプログラムを実行します。

学習項目

1. 準備
2. データの準備
3. モデルの定義
4. パラメータの設定
5. トレーニング
6. 評価

使用するファイル

- 5_autoencoder.ipynb（このファイルです）
- data/ecg_data/down234_148000_152000.csv (訓練用データ)
- data/ecg_data/down234_152000_156000.csv (テスト用データ)

データセット

- MIT-BIH Arrhythmia Databaseで公開されている心電図データのうち、ID234のデータを使用します。
- 元データのサンプリング周波数は360Hzです。2チャンネル分（MLII、V1）のデータが含まれています。
- 前処理として、サンプリング周波数を元データの半分の180Hzにするダウンサンプリング、トレンドの除去、および0から1の範囲へのスケーリングを行いました。
- 約65万点から、元データの148000点目から151999点までを訓練用に、152000から159999点までをテスト用に切り出し、それぞれ別のcsvファイルとしました。
- 各csvファイルには、約11秒（=2000point/180Hz）のデータが含まれます。csvファイルの1列目にはMLII、2列目にはV1の2列のデータからなります。

1. 準備

必要なライブラリの読み込み

- NumPy：モデル入力用のデータを作成するために使用します。
- Matplotlib：データの可視化に使用します。
- keras：ニューラルネットワークのためのライブラリです。

```
In [1]: 1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import matplotlib as mpl
5
6 from keras.layers import Input, Dense
7 from keras.models import Model
8 from keras import regularizers
9 from keras import backend as K
```

Using TensorFlow backend.

バックエンドのグラフ識別子をリセット

- clear_session関数はバックエンドのTensorFlowのグラフを壊し、新たなものを作成します。
(https://keras.io/ja/backend/#clear_session (https://keras.io/ja/backend/#clear_session))

```
In [2]: 1 K.clear_session()
```

グラフのデフォルト表示サイズの変更

```
In [3]: 1 mpl.rcParams['figure.figsize'] = 15, 4
```

カレントディレクトリの移動

```
In [4]: 1 # from google.colab import drive
2 # drive.mount('/gdrive')
3 # %cd "/gdrive/My Drive/Colab Notebooks/jasla_rensyuu_20190202"
```

2. データの準備

CSVファイルからデータを読み込む

- csvファイルから Pandas の DataFrame としてデータを読み込みます。
- 訓練用データを変数 train_df に、テストデータを変数 test_df に読み込んでみましょう。
- read_csv 関数を使います。
- 今回は、MLIIのみを使用するため、csvファイルの1列目だけを読み込みます。usecols/パラメータで読み込む列を指定します。1列目の指定は、0となります。

```
In [5]: 1 train_df = pd.read_csv('data/ecg_data/down234_148000_152000.csv', usecols=[0])
2 test_df = pd.read_csv('data/ecg_data/down234_152000_156000.csv', usecols=[0])
3 # train_df = pd.read_csv('data/ecg_data/scaled234_148000_152000.csv', usecols=[0])
4 # test_df = pd.read_csv('data/ecg_data/scaled234_152000_156000.csv', usecols=[0])
```

データのサイズを確認してみましょう

- データの行と列の数を確認するには、shape を使います。
- shapeは関数ではないので、カッコが不要です。

```
In [6]: 1 train_df.shape
```

```
Out[6]: (2000, 1)
```

```
In [7]: 1 test_df.shape
```

```
Out[7]: (2000, 1)
```

データの一部を表示してみましょう

- データの一部を表示するには、head関数や、tail関数を使います。
- head関数とtail関数は、表示する行数を引数にとります。（デフォルト値:5）

```
In [8]: 1 train_df.head()
```

```
Out[8]:
```

	MLII
0	0.109128
1	0.088258
2	0.066945
3	0.058160
4	0.071976

```
In [9]: 1 test_df.head()
```

Out[9]:

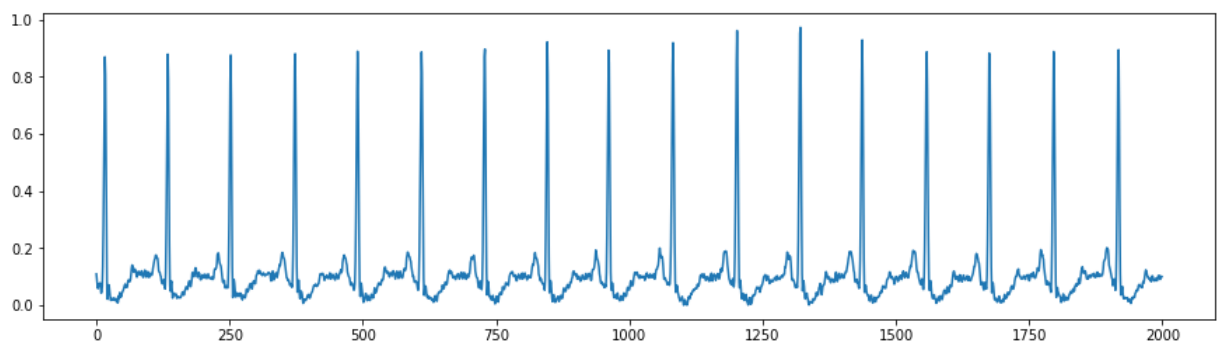
	MLII
0	0.164886
1	0.172314
2	0.177290
3	0.179869
4	0.187242

グラフを表示してみましょう。

- matplotlibのplotメソッドを使います。

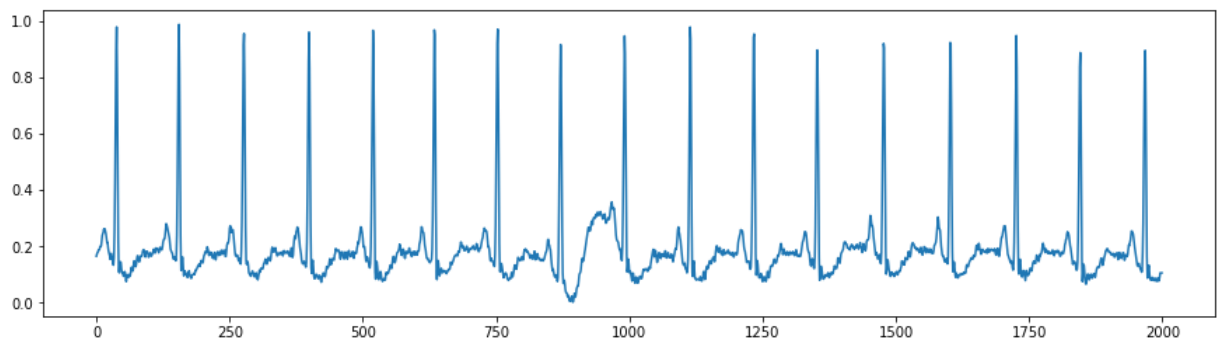
```
In [10]: 1 plt.plot(train_df)
```

Out[10]: [matplotlib.lines.Line2D at 0x2cea0b2f048>]



```
In [11]: 1 plt.plot(test_df)
```

Out[11]: [matplotlib.lines.Line2D at 0x2cea0b47748>]



モデル入力用データの作成

- 位置を1点ずつずらしながら指定されたウィンドウサイズのデータを順に切り出す関数を定義します。入力はDataFrameとウィンドウサイズ、出力はDataFrameとします。
- 訓練データを `X_train` に、テストデータを `X_test` に代入します。
- `head` 関数などで、内容を確認しておきましょう。

```
In [12]: 1 def subseq(x, w):
2         seq = np.zeros((len(x) - w + 1, w))
3         x_np = x.iloc[:, 0]
4         for i in range(len(x) - w + 1):
5             seq[i, :] = x_np[i:w+i]
6         return pd.DataFrame(seq)
7
8     w_size = 100 # ウィンドウサイズ
9     X_train = subseq(train_df, w_size)
10    X_test = subseq(test_df, w_size)
```

```
In [13]: 1 X_train.head()
```

```
Out[13]:
```

	0	1	2	3	4	5	6	7	8	9	...	90
0	0.109128	0.088258	0.066945	0.058160	0.071976	0.070855	0.074548	0.078003	0.061329	0.042165	...	0.113487
1	0.088258	0.066945	0.058160	0.071976	0.070855	0.074548	0.078003	0.061329	0.042165	0.047974	...	0.122801
2	0.066945	0.058160	0.071976	0.070855	0.074548	0.078003	0.061329	0.042165	0.047974	0.066254	...	0.109664
3	0.058160	0.071976	0.070855	0.074548	0.078003	0.061329	0.042165	0.047974	0.066254	0.134413	...	0.101539
4	0.071976	0.070855	0.074548	0.078003	0.061329	0.042165	0.047974	0.066254	0.134413	0.267482	...	0.098413

5 rows × 100 columns

3. モデルの定義

- Kerasのfunctional APIを使います。
- 各層のユニット数は、入力層100、隠れ層160、出力層100の3層のスパースオートエンコーダとします。
- 中間層の正則化項にL1出力正則化を用います。

```
In [14]: 1 l1=10e-7 # L1正則化のパラメータ
2 enc_dim = 160 # 隠れ層のユニット数
3
4 input_layer = Input(shape=(w_size,))
5 encoded = Dense(enc_dim, activation='relu',
6                 activity_regularizer=regularizers.l1(l1))(input_layer)
7 decoded = Dense(w_size, activation='relu')(encoded)
8 autoencoder = Model(inputs=input_layer,
9                     outputs=decoded)
```

- summary関数で、モデルのが概要を確認できます。

```
In [15]: 1 autoencoder.summary()
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 100)	0
dense_1 (Dense)	(None, 160)	16160
dense_2 (Dense)	(None, 100)	16100
Total params: 32,260		
Trainable params: 32,260		
Non-trainable params: 0		

4. パラメータの設定

- compile関数を用いてどのような学習処理を行なうかを設定を行います。

- 今回は、損失関数にbinary_crossentropy、最適化アルゴリズムにadadelatを指定します。
- 細かいパラメータは、すべてデフォルトのままとしました。

```
In [16]: 1 autoencoder.compile(optimizer='adadelat',
2               loss='binary_crossentropy')
```

5. トレーニング

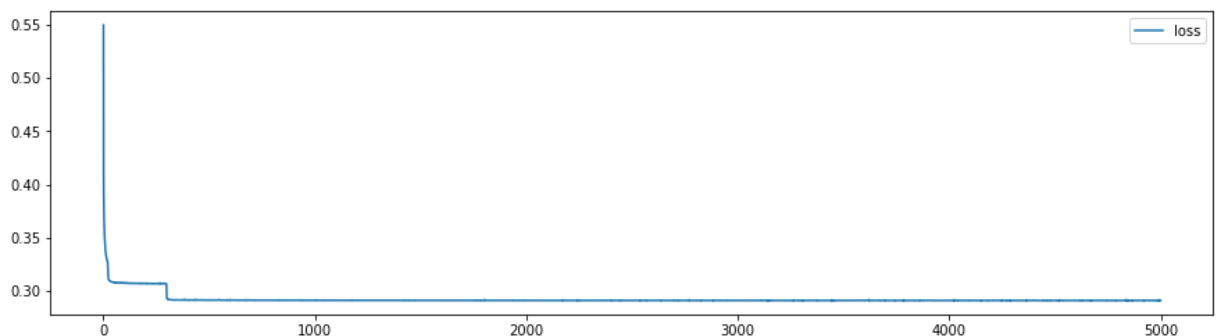
- 入力データと教師データにX_trainを使って、モデルのトレーニング（学習）を行います。
- モデルをトレーニングには、fit関数を使います。
- fit関数には、トレーニングの入力データと教師データ、ミニバッチのサイズ(batch_size)、エポック数などを渡します。
- verbose=0とすると、トレーニング状況のログを画面に表示しません。

```
In [17]: 1 epochs = 5000
2 history = autoencoder.fit(X_train,
3                           X_train,
4                           epochs=epochs,
5                           batch_size=None,
6                           verbose=0)
```

- fit関数の戻り値のhistoryオブジェクトには、各エポックでのloss関数の推移などが辞書型のデータとして記録されます。
- plot関数で表示してみましょう。

```
In [18]: 1 pd.DataFrame(history.history['loss'], columns=['loss']).plot()
```

Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x2ce9a42eb38>



6. 評価

テストデータとオートエンコーダの出力との比較

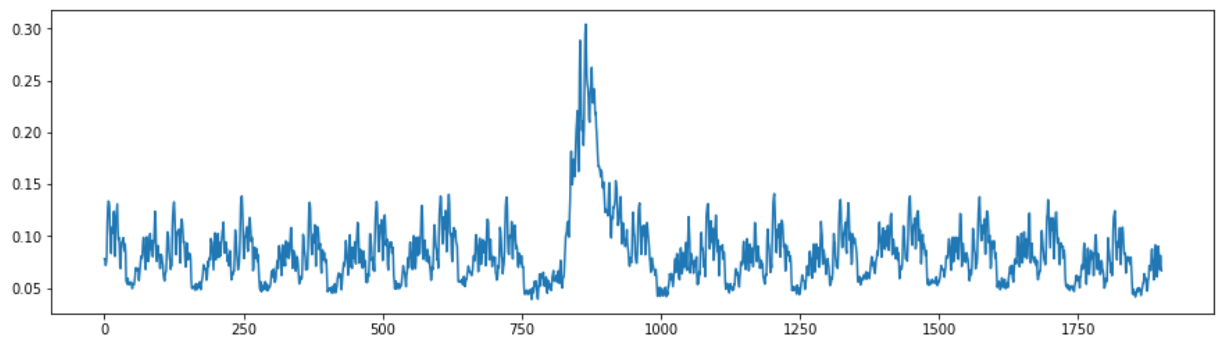
- テストデータを使って、トレーニング済みモデルの評価を行います。
- predict関数を使い、トレーニング済みモデルにテストデータを適用します。predict関数の戻り値は、Numpy Arrayです。

```
In [19]: 1 decoded = pd.DataFrame(autoencoder.predict(X_test))
```

- 入力したテスト用データと、モデルの出力との距離として、各要素の差の二乗和の平方根を求めます。
- 結果をplotしてみましょう。

```
In [20]: 1 dist = np.sqrt( np.sum( (decoded - X_test)**2, axis=1))
        2 dist.plot()
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x2cea2655dd8>

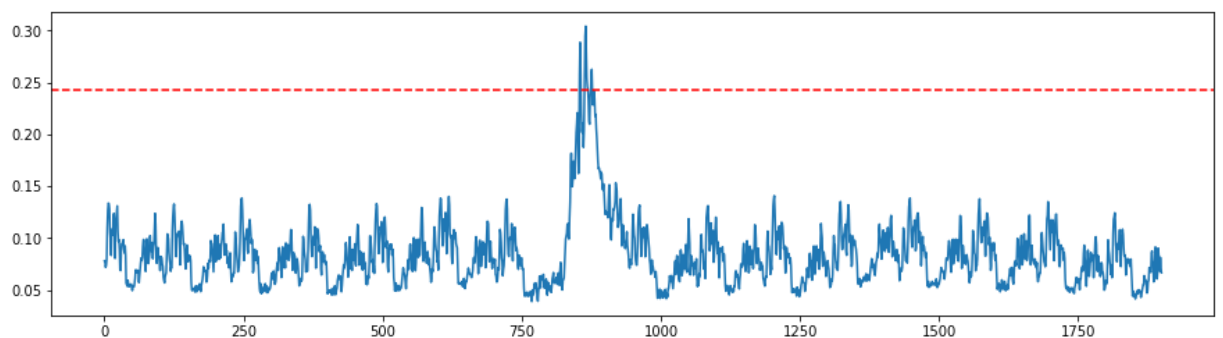


異常の判定

- 閾値の設定は、対象とするに問題領域に応じて適当に定める必要があります。
- 今回はテストデータに異常が含まれているという前提で、ピークの80%の値を閾値とし、閾値を超えるものを異常とみなします。

```
In [21]: 1 ax = dist.plot()
        2 ax.axhline(y=dist.max() * 0.8, color='red', linestyle='--')
```

Out[21]: <matplotlib.lines.Line2D at 0x2cea268e828>



- 閾値を超える点のインデックスを取得してみます。

```
In [22]: 1 dist[dist > dist.max() * 0.8].index
```

Out[22]: Int64Index([854, 855, 856, 863, 864, 865, 866, 867, 868, 875, 876], dtype='int64')

第6章

1次元畳み込みニューラルネットワーク による異常検知（講義）

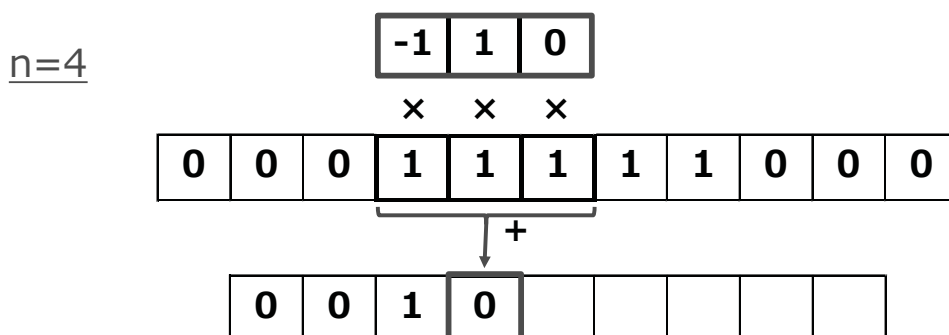
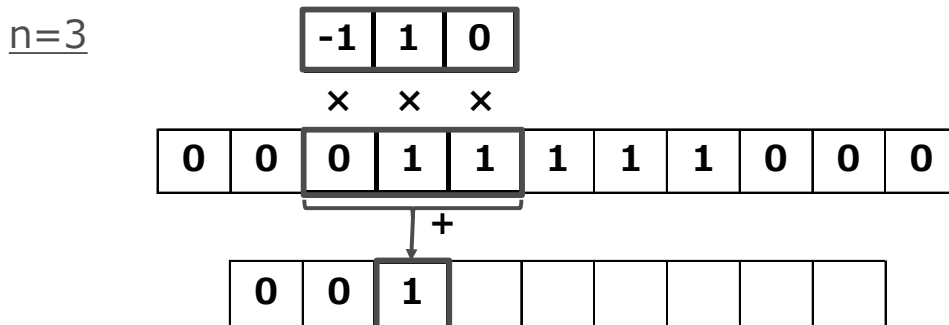
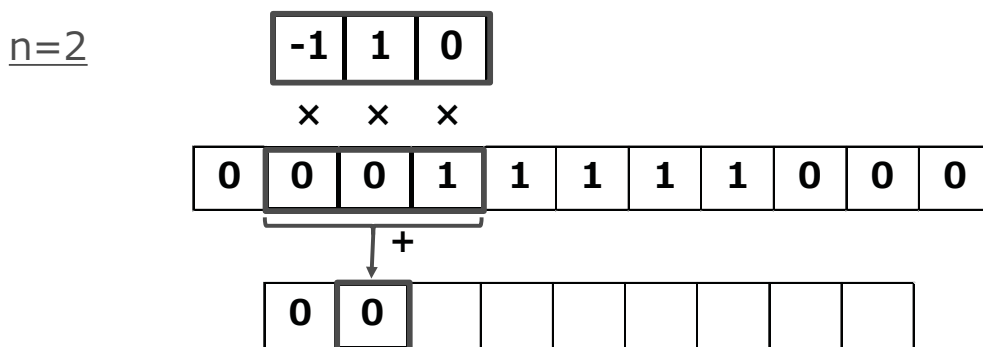
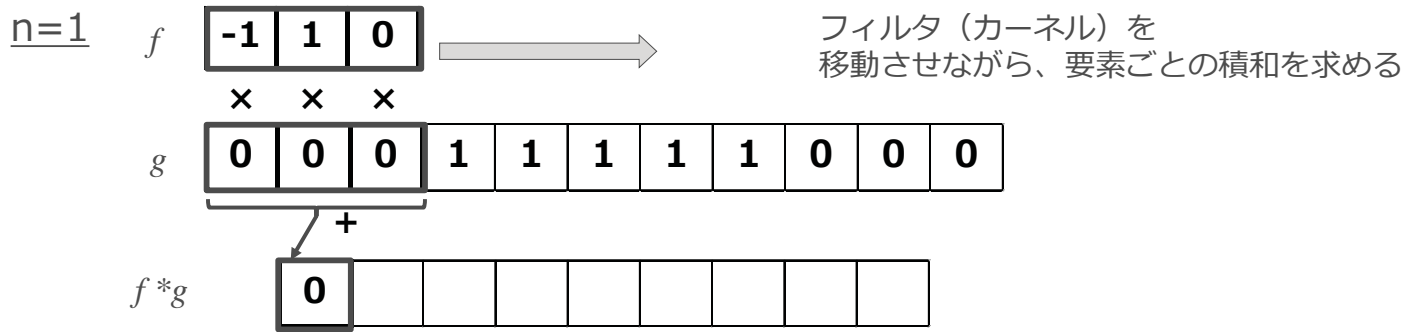
第6章では、もうひとつの異常検知の方法「1次元畳み込みニューラルネットワーク(1d-CNN)による異常検知」について説明します。

学習項目

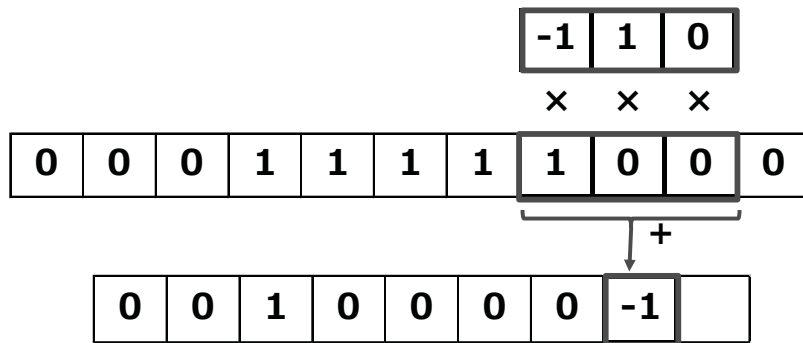
1. 畳み込み
2. 畳み込み層（Convolution layers）
3. プーリング層（Pooling layers）
4. 平滑化（Flatten）
5. 層のスタッキング
6. 1D-CNNによるプログラミング演習概要

1. 畳み込み

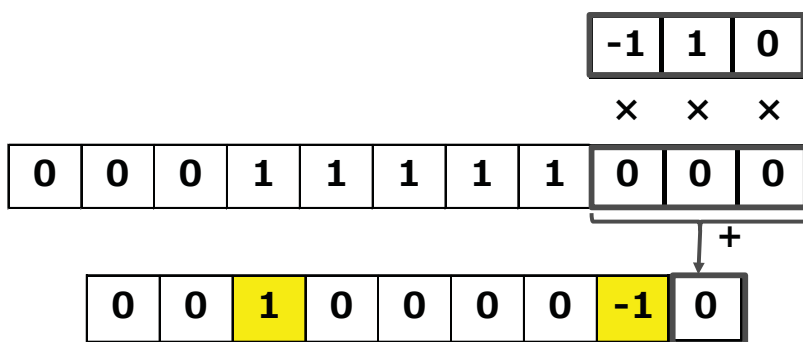
1次元離散畳み込み (1d Discrete convolution) の例



n=8



n=9

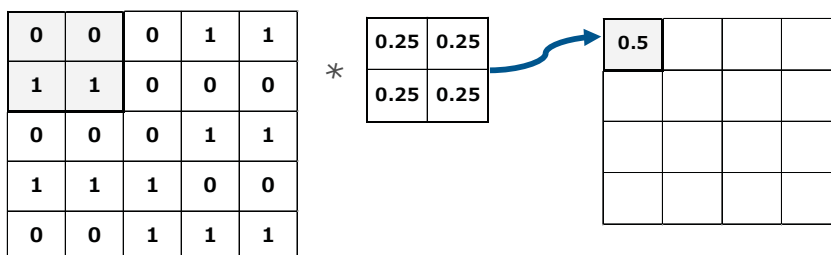


-1	1	0
----	---	---

変化点（エッジ）を検出

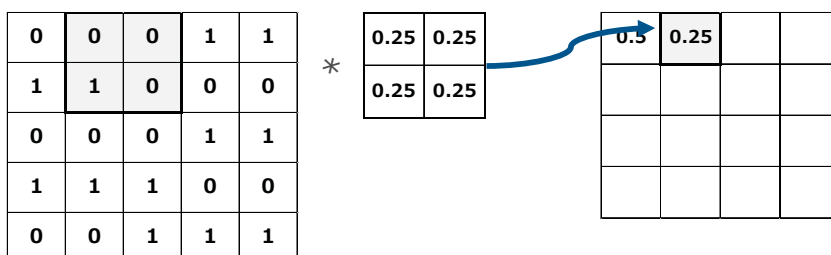
2次元離散畳み込み（2d Discrete convolution）の例

n=1

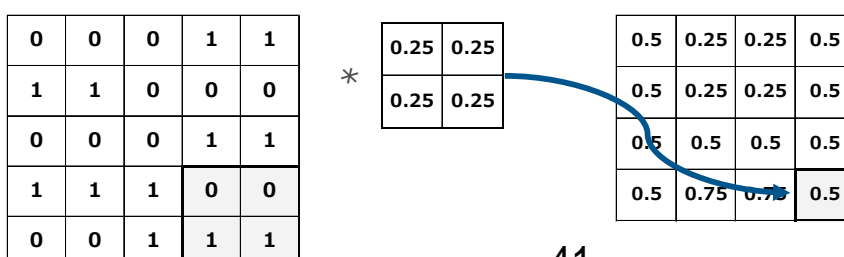


フィルターを移動させながら、要素ごとの積和を求める

n=2



n=16



0.25	0.25
0.25	0.25

平滑化（ぼかし）
フィルター

その他のフィルターの例

0	-1	0
0	1	0
0	0	0

横方向のエッジ

0	0	0
-1	1	0
0	0	0

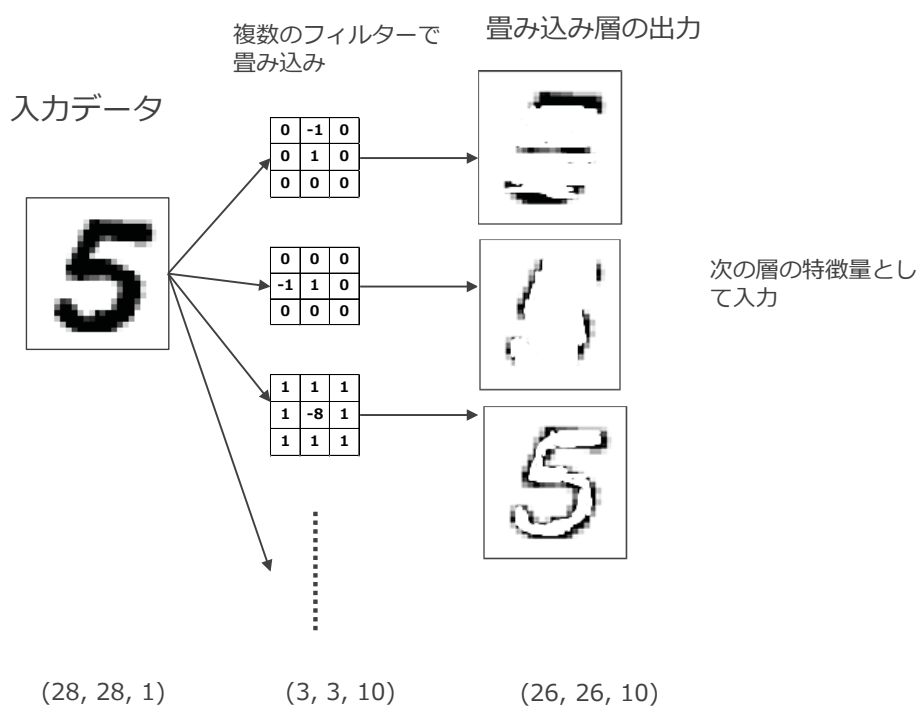
縦方向のエッジ

1	1	1
1	-8	1
1	1	1

縦、横、ななめ方向のエッジ

他にも、特定のパターンだけに反応する
パターンマッチングなど

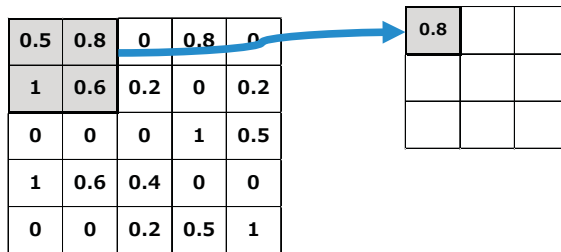
2. 畳み込み層 (Convolution layers)



3. プーリング層 (Pooling layers)

畳み込み層の後に適用されることが多い。次元を圧縮する。

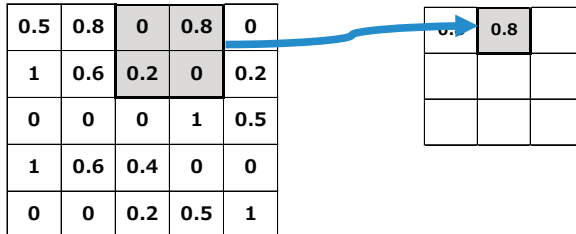
0.5	0.8	0	0.8	0
1	0.6	0.2	0	0.2
0	0	0	1	0.5
1	0.6	0.4	0	0
0	0	0.2	0.5	1



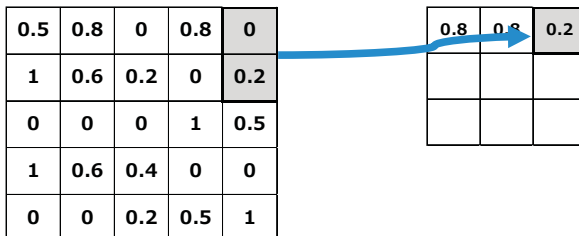
Max poolingの場合、
部分領域ごとに最大値を選択

他に、average poolingなどあり

0.5	0.8	0	0.8	0
1	0.6	0.2	0	0.2
0	0	0	1	0.5
1	0.6	0.4	0	0
0	0	0.2	0.5	1



0.5	0.8	0	0.8	0
1	0.6	0.2	0	0.2
0	0	0	1	0.5
1	0.6	0.4	0	0
0	0	0.2	0.5	1

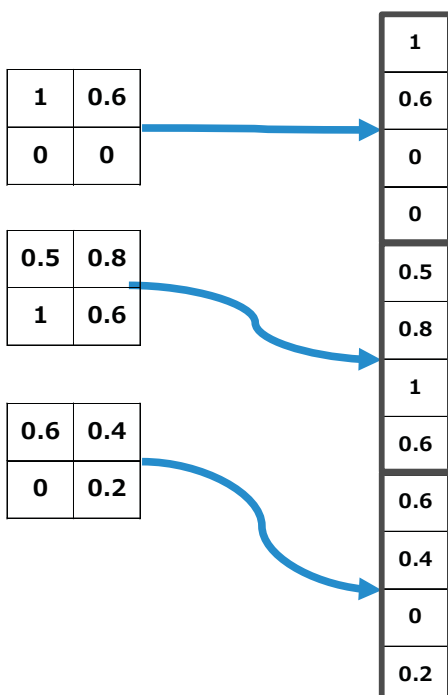


プーリング層の効果
• 位置変化に対して強くなる
• 計算負荷を下げる

4. 平滑化 (Flatten)

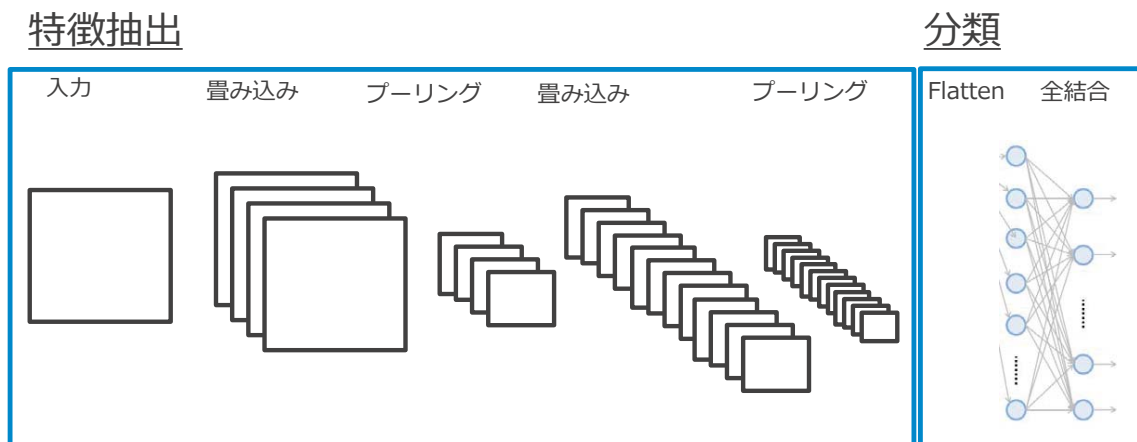
入力を平滑化する。全結合層の前に使われることが多い

1	0.6
0	0



1
0.6
0
0
0.5
0.8
1
0.6
0.6
0.4
0
0.2

5. 層のスタッキング



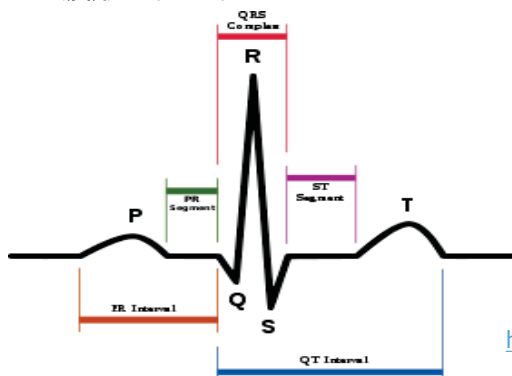
6. 1D-CNNによるプログラミング演習概要

目標

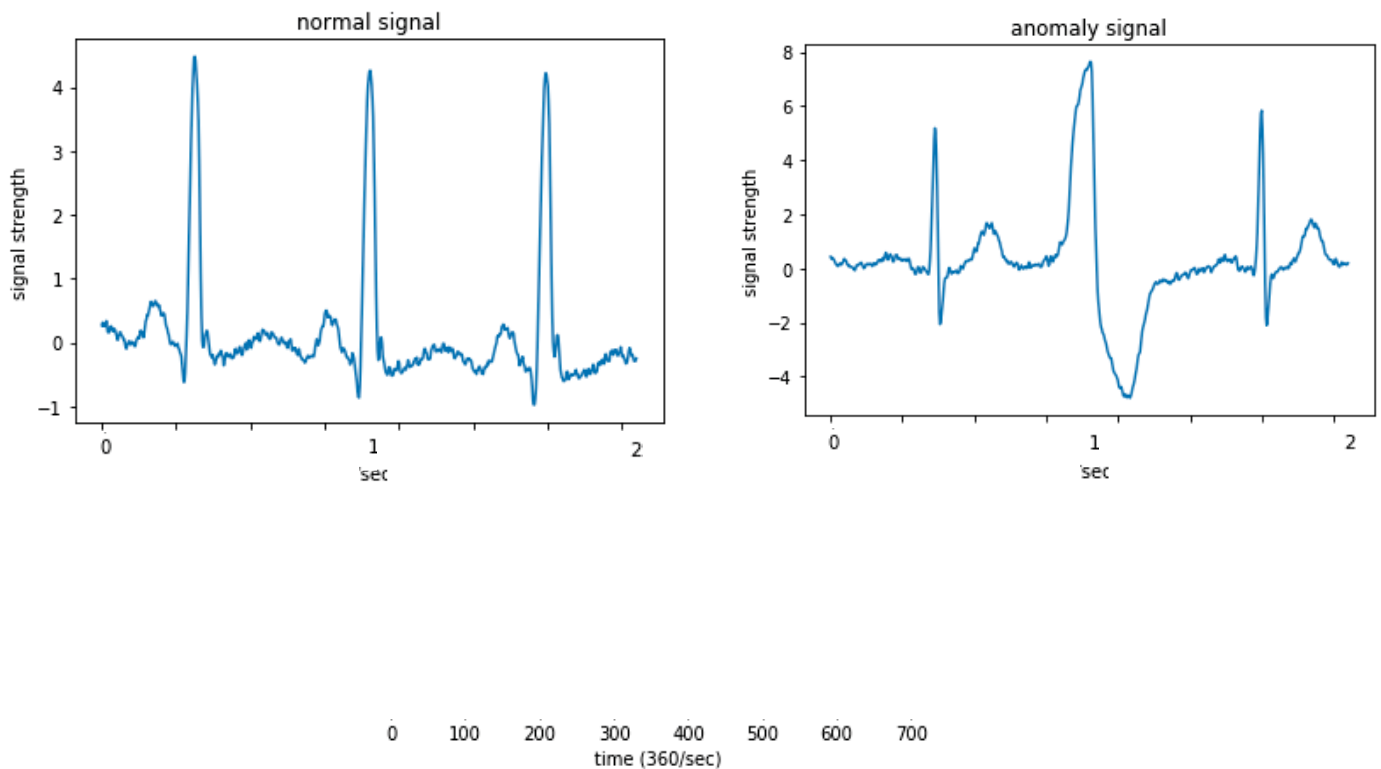
- MIT-BIH Arrhythmia Databaseで公開されている心電図データから、異常（不整脈）を検出します。

データセット

- 公開されている48レコードのうち、ID122, 209, 220, 223, 234の5つを使います。
- 前処理として、標準化とダウンサンプリングを行いました。サンプリング周波数は元データの半分の180Hzです。
- 付与されていたアノテーションを基に、Rのピークを中心に前後1秒ずつ、計2秒を切り出して一つのデータとしました。1データあたり、2秒分で360点からなります。
- 各データには、正常か異常のラベルがついています。
- 波形のデータをつにまとめX.csvに、対応する正解ラベルをy.csvに、csv形式で保存しました。



- データの例



ディープラーニングのプログラムの流れ（再掲）

■ディープラーニングのプログラムは、次のような流れで実行します。

- ① データ準備：欠損値の処理、正規化など
- ② モデル定義：レイヤー構成、ノード数などの決定
- ③ パラメータの設定：損失関数、最適化手法等
- ④ トレーニング：実行回数などを決める
- ⑤ 評価：テストデータで評価
- ⑥ 必要に応じて、②に戻る

以下では、上記流れ①～⑤の各項目をより詳しく確認しておきます。

① データ準備

1. モデル入力用データの作成

- 異常が認められるデータは、全体の4%以下です。
- 通常のカテゴリ問題では、各クラスの比率が極端に異なる場合には、多いデータの削除や、少ないデータの増強などでバランスをとることがありますが、今回はデータの比率の調整を行わずに実行してみます。
- データは、60%を訓練データに、40%をテスト用データにします。
- データの分割には、sklearnライブラリに含まれるtrain_test_split関数を使います。

2. 正解レベルのOne-hot表現への変換

0:正常 1:異常		正常	異常
0	→	1	0
0		1	0
1		0	1
0		1	0
1		0	1

② モデル定義

- Sequentialモデルを使います。
- 1次元の畳み込み層は、Conv1Dです。フィルターの数と、カーネルのサイズを指定します。
- 全結合層（Dense）に渡す前に、Flattenを呼び平滑化をしましょう。
- Dropoutも追加できます。

今回のモデル

```
model = Sequential()  
model.add(Conv1D(filters=64, kernel_size=10, input_shape=(X_train.shape[1], 1)))  
model.add(Activation('relu'))  
model.add(Flatten())  
model.add(Dropout(0.4))  
model.add(Dense(128, activation='relu'))  
model.add(Dense(n_class))  
model.add(Activation('softmax'))
```


③ パラメータの設定

- モデルの学習を始める前に、compile関数を用いてどのような学習処理を行なうかを設定します。
- compileでは、最適化アルゴリズム、損失関数などを指定します。

例：損失関数にカテゴリカルクロスエントロピー、最適化アルゴリズムに確率的勾配降下法（SGD）を指定

```
model.compile(loss='categorical_crossentropy',  
              optimizer='sgd')
```

④ トレーニング

- 入力データと教師データ（正解ラベル）を使って、モデルのトレーニング（学習）を行います。
- モデルをトレーニングには、fit関数を使います。
- fit関数には、トレーニングの入力データと教師データ、ミニバッチのサイズ(batch_size)、エポック数などを渡します。

⑤ 評価

- テストデータを使って、トレーニング済みモデルの評価を行います。
- トレーニング済みモデルを使って、新しいデータから予測を行うには、predict関数を使います。
- 混同行列 (confusion matrix) や、正解率 (accuracy) などでも評価をします。

(参考) 評価指標についてのわかりやすい解説

<http://ibisforest.org/index.php?F%E5%80%A4>

<https://pythondatascience.plavox.info/scikit-learn/%E5%88%86%E9%A1%9E%E7%B5%90%E6%9E%9C%E3%81%AE%E3%83%A2%E3%83%87%E3%83%AB%E8%A9%95%E4%BE%A1>

第7章 1次元畳み込みニューラルネットワークによる異常検知 (プログラム)

第7章では、1次元畳み込みニューラルネットワークによる異常検知のプログラムを実行します。

学習項目

1. 準備
2. データの準備
3. モデルの定義
4. パラメータの設定
5. トレーニング
6. 評価

使用するファイル

- 7_1dCNN.ipynb (このファイルです)
- data/X.csv (心電図の波形データ)
- data/y.csv (正解ラベル)

データセット

- MIT-BIH Arrhythmia Databaseで公開されている心電図データから一部を抜粋し、本セミナー用に前処理を行いました。
- 元データでRとラベルのついた波形のピークの前後1秒ずつ、計2秒を切り出して一つのデータとしました。事前に標準化を行いました。
- サンプリングレート180Hzで、一つのデータは2秒分で360点からなります。
- 各データには、正常か異常かの正解ラベルがついています。
- 波形のデータはX.csvに、対応する正解ラベルはy.csvに保存されています。
- 正解ラベルは数値が割り当てられており、0が正常、1が異常です。

1. 準備

必要なライブラリの読み込み

- NumPy : モデル入力用のデータを作成するために使用します。
- Matplotlib : データの可視化に使用します。
- keras : ニューラルネットワークのためのライブラリです。
- scikit-learn : 機械学習用のライブラリです。データの分割、結果の評価に使用します。

```
In [1]: 1 import matplotlib.pyplot as plt
2 import numpy as np
3 import pandas as pd
4
5 from keras.models import Sequential
6 from keras.layers import Dense, Activation, Flatten, Conv1D, Dropout
7 from keras.optimizers import SGD
8 from keras.utils.np_utils import to_categorical
9
10 from sklearn.model_selection import train_test_split
11 from sklearn.metrics import accuracy_score
12 from sklearn.metrics import confusion_matrix
13 from sklearn.metrics import classification_report
14 import keras.backend as K
```

Using TensorFlow backend.

バックエンドのグラフ識別子をリセット

```
In [2]: 1 K.clear_session
```

```
Out[2]: <function keras.backend.tensorflow_backend.clear_session()>
```

カレントディレクトリの移動

```
In [3]: 1 # from google.colab import drive
2 # drive.mount('/gdrive')
3 # %cd "/gdrive/My Drive/Colab Notebooks/jasla_rensyuu_20190202"
```

2. データの準備

CSVファイルからデータを読み込む

- csvファイルから Pandas の DataFrame としてデータを読み込みます。
- 波形データを変数 `x_df` に、正解ラベルを変数 `y_df` に読み込んでみましょう。
- `X.csv` はヘッダー行を含まないファイルのため、`read_csv` 関数のパラメータで `header=None` を指定します。

```
In [4]: 1 x_df = pd.read_csv('data/X.csv', header=None)
2 y_df = pd.read_csv('data/y.csv')
```

データのサイズを確認してみましょう

- データの行と列の数を確認するには、`shape` を使います。
- `shape` は関数ではないので、カッコが不要です。

```
In [5]: 1 x_df.shape
2
```

```
Out[5]: (12260, 360)
```

```
In [6]: 1 y_df.shape
```

```
Out[6]: (12260, 1)
```

データの一部を表示してみましょう

- データの一部を表示するには、`head` 関数や、`tail` 関数を使います。
- 他に長いデータの場合、`.T` を使って転置をすると、見やすくなる場合があります。

```
In [7]: 1 y_df.head(20).T
```

```
Out[7]:
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

異常値の割合を確認してみましょう

- `Series` の `value_counts` 関数を使うとデータの値の出現回数を数えることができます。
- `y_df` は `DataFrame` のなので、列を指定して `Series` とします。
- `value_counts` 関数の `normalize` パラメータを `True` とすると、割合が分かります。

```
In [8]: 1 y_df['y'].value_counts(True)
```

```
Out[8]: 0    0.961093
        1    0.038907
        Name: y, dtype: float64
```

トレーニングデータとテストデータに分割

- データを訓練データと、テストデータに分けます。60%を訓練データに、40%をテストデータに使用します。
- データの分割は、scikit-learnライブラリのtrain_test_splitが便利です。
- train_test_splitは、第1引数に入力データ、第2引数に正解ラベルを渡します。test_sizeではテストデータの割合を0.0~1.0の実数で指定します。デフォルトではランダムにシャッフルして分割します。random_stateに任意の整数を指定して、乱数のシードを固定することで、同じ分割を再現できます。
- トレーニング用データを変数x_trainとy_trainに、テスト用データを変数x_testとy_testに代入しましょう。
- 分割後のデータのサイズや、異常値の割合も確認しておきましょう。

```
In [9]: 1 x_train, x_test, y_train, y_test = train_test_split(x_df, y_df, test_size=0.4, random_state=0)
```

```
In [10]: 1 x_train.shape, x_test.shape
```

```
Out[10]: ((7356, 360), (4904, 360))
```

```
In [11]: 1 y_train['y'].value_counts(normalize=True)
```

```
Out[11]: 0    0.9618
        1    0.0382
        Name: y, dtype: float64
```

```
In [12]: 1 y_test['y'].value_counts(normalize=True)
```

```
Out[12]: 0    0.960033
        1    0.039967
        Name: y, dtype: float64
```

3階のテンソルへの変換

- Kerasの1次元の畳み込みレイヤーConv1Dへの入力、Shapeが(batch_size, steps, input_dim)の3階テンソルにします。(<https://keras.io/ja/layers/convolutional/>)
- reshape関数で変換をし、変換後のデータは、それぞれ変数 X_train と X_testに代入しましょう。

```
In [13]: 1 X_train = x_train.values.reshape(x_train.shape[0], x_train.shape[1], 1)
        2 X_test = x_test.values.reshape(x_test.shape[0], x_test.shape[1], 1)
```

```
In [14]: 1 X_train.shape, X_test.shape
```

```
Out[14]: ((7356, 360, 1), (4904, 360, 1))
```

正解レベルのOne-hot表現への変換

- 正解ラベルをOne-hot表現に変換します。すなわち「1列で、0（正常）か1（異常）で表現されていた正解ラベルデータ」を、「2列で、1列目が1で2列目が0ならば正常、1列目が0で2列目が1ならば異常」と表現するようにします。
- Kerasのto_categorical関数を利用します。
- この変換は2値分類問題のときには必須ではありませんが、多値分類問題のときには必要な処理になります。
- 変換後の正解ラベルデータを、y_train_cおよびy_test_cに代入します。

```
In [15]: 1 n_class = 2
        2 y_train_c = to_categorical(y_train, n_class)
        3 y_test_c = to_categorical(y_test, n_class)
```

```
In [16]: 1 y_train_c.shape, y_test_c.shape
```

```
Out[16]: ((7356, 2), (4904, 2))
```

3. モデルの定義

- KerasのSequentialモデルを使います。
- 1次元の畳み込み層は、Conv1Dです。フィルターの数と、カーネルのサイズを指定します。
- 全結合層（Dense）に渡す前に、Flattenを呼びましょう。
- Dropoutも追加できます。

```
In [17]: 1 model = Sequential()  
2 model.add(Conv1D(filters=64, kernel_size=10, input_shape=(X_train.shape[1], 1)))  
3 model.add(Activation('relu'))  
4 model.add(Flatten())  
5 model.add(Dropout(0.4))  
6 model.add(Dense(128, activation='relu'))  
7 model.add(Dense(n_class))  
8 model.add(Activation('softmax'))
```

- summary関数で、モデルのが概要を確認できます。

```
In [18]: 1 model.summary()
```

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 351, 64)	704
activation_1 (Activation)	(None, 351, 64)	0
flatten_1 (Flatten)	(None, 22464)	0
dropout_1 (Dropout)	(None, 22464)	0
dense_1 (Dense)	(None, 128)	2875520
dense_2 (Dense)	(None, 2)	258
activation_2 (Activation)	(None, 2)	0
Total params: 2,876,482		
Trainable params: 2,876,482		
Non-trainable params: 0		

4. パラメータの設定

- compile関数を用いてどのような学習処理を行なうかを設定を行います。
- 今回は、損失関数にカテゴリカルクロスエントロピー、最適化アルゴリズムに確率的勾配降下法（SGD）を指定します。
- 細かいパラメータは、すべてデフォルトのままとしました。

```
In [19]: 1 model.compile(loss='categorical_crossentropy',  
2                 optimizer='sgd')
```

5. トレーニング

- 入力データと教師データ（正解ラベル）を使って、モデルのトレーニング（学習）を行います。
- モデルをトレーニングには、fit関数を使います。

- fit関数には、トレーニングの入力データと教師データ、ミニバッチのサイズ(batch_size)、エポック数などを渡します。

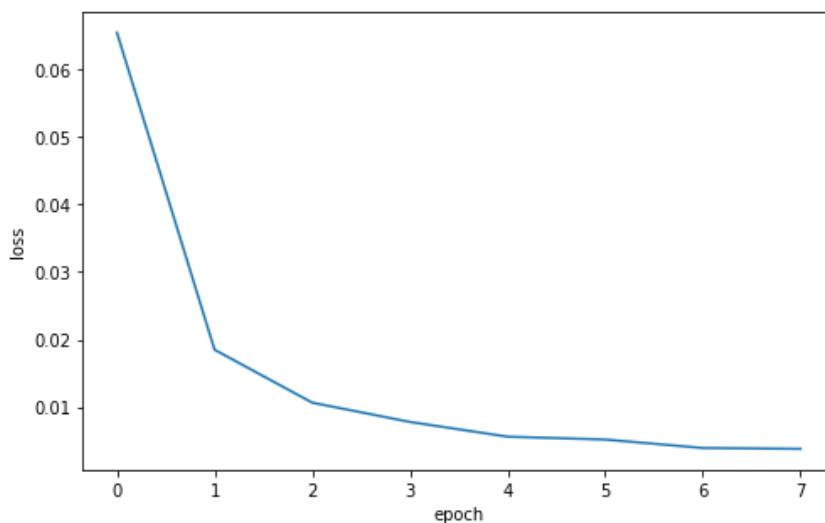
In [20]:

```
1 epochs = 8
2 history = model.fit(X_train, y_train_c, epochs=epochs)
```

```
Epoch 1/8
7356/7356 [=====] - 8s 1ms/step - loss: 0.0655
Epoch 2/8
7356/7356 [=====] - 7s 986us/step - loss: 0.0185
Epoch 3/8
7356/7356 [=====] - 7s 950us/step - loss: 0.0106
Epoch 4/8
7356/7356 [=====] - 7s 953us/step - loss: 0.0078
Epoch 5/8
7356/7356 [=====] - 7s 949us/step - loss: 0.0056
Epoch 6/8
7356/7356 [=====] - 7s 995us/step - loss: 0.0052
Epoch 7/8
7356/7356 [=====] - 7s 937us/step - loss: 0.0039
Epoch 8/8
7356/7356 [=====] - 7s 946us/step - loss: 0.0038
```

In [21]:

```
1 plt.figure(figsize=(8, 5))
2 plt.plot(history.history['loss'])
3 plt.ylabel('loss')
4 plt.xlabel('epoch')
5 plt.show()
```



6. 評価

- テストデータを使って、トレーニング済みモデルの評価を行います。

テストデータからトレーニング済みモデルを使って予測

- predict_classes関数にテストデータを渡すことで、トレーニング済みモデルによる推定結果を得ることができます。

In [22]:

```
1 y_pred = model.predict_classes(X_test)
```

各種評価指標の算出

- 分類問題のモデル評価には、scikit-learnの関数を使うのが便利です。
- accuracy_score関数で、正解率を算出できます。

```
In [23]: 1 accuracy_score(y_test, y_pred)
```

```
Out[23]: 0.9987765089722676
```

- confusion_matrix関数で、混同行列を表示することができます。

```
In [24]: 1 confusion_matrix(y_test, y_pred)
```

```
Out[24]: array([[4705,    3],
               [    3,   193]], dtype=int64)
```

- 下記では混同行列を見やすくするためのpretty_confusion_matrix関数を定義しました。

```
In [25]: 1 def pretty_confusion_matrix(y_true, y_pred, labels=["False", "True"]):
2         cm = confusion_matrix(y_true, y_pred)
3         pred_labels = ['Predicted ' + l for l in labels]
4         df = pd.DataFrame(cm, index=labels, columns=pred_labels)
5         return df
6
7         pretty_confusion_matrix(y_test, y_pred, ['Normal', 'Abnormal'])
```

```
Out[25]:
```

	Predicted Normal	Predicted Abnormal
Normal	4705	3
Abnormal	3	193

- classification_report関数では、精度 (Precision)や、検出率 (Recall) を表示することができます。

```
In [26]: 1 print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	4708
1	0.98	0.98	0.98	196
micro avg	1.00	1.00	1.00	4904
macro avg	0.99	0.99	0.99	4904
weighted avg	1.00	1.00	1.00	4904

第 8 章

人の活動の識別：概要（講義）

本章では、第8章から第10章までで取り上げる課題およびデータセットの概要について説明します。

人の活動の識別：概要

目標

- スマートフォンの加速度センサー等のデータを使って、6種類の人の活動を識別します。

データセットについて

「Human Activity Recognition Using Smartphones Data Set」

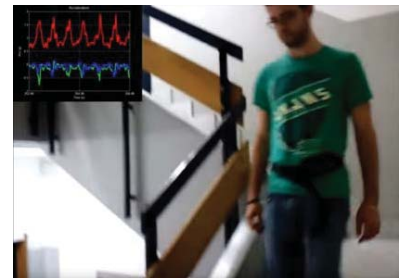
（スマートフォンを使った人間活動の識別）

(<https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones>)

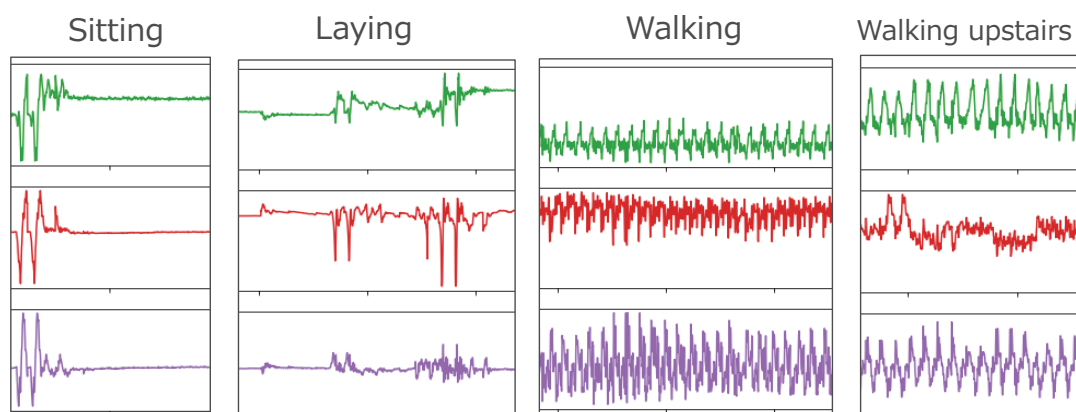
- UC Irvineは、機械学習コミュニティへのサービスとして、2019年1月現在468のデータセットをリポジトリで管理をしています。
- 機械学習の練習に使えるデータセットがUCI Machine Learning Repositoryから無料で入手することができます。
- このデータの人気は高く、アクセス数は第9位です。インターネット上で、多くの良質なサンプルプログラムを見つけることができます。

データセット

- 19歳から48歳の30人が、スマートフォンを(Samsung Galaxy S II)を腰に着用し、次の6種類の活動を行った際のデータです。
 1. 歩行 : WALKING
 2. 階段上り : WALKING_UPSTAIRS
 3. 階段下り : WALKING_DOWNSTAIRS
 4. 着席 : SITTING
 5. 直立 : STANDING
 6. 横たえる : LAYING (lyingでなく laying?)
- リポジトリでは、周波数領域の情報などを含む561の特徴量を抽出したデータも提供されていますが、本セミナーでは、生データを使います。
- 各3軸の直線加速度、角加速度、ジャイロの9つのデータからなります。
- 128サンプルごとに、正解ラベルがついています。



- データの例



第9章 人の活動の識別：データの基礎分析と前処理（プログラム）

第9章では、人の活動識別のためのデータセットを概観し、スケーリングなどの前処理を実施します。また、ディープラーニングの入力データとして次章ですぐに使えるように、3階のテンソルに変換してファイルに保存します。

学習項目

1. 準備
2. データの読み込み
3. データの概要の確認
4. ディープラーニング入力用にデータを変換
5. ファイルに保存

使用するファイル

- 9_har_eda.ipynb（このファイルです）
- data/uci_har/ 以下にある20のtxtファイル

データセット

- UCI Machine Learning Repositoryで公開されているHuman Activity Recognition Using Smartphones Data Set から抜粋したもの。
- データは、公開時点で訓練用データとテスト用データに分かれている。
- ファイルは、スペース区切りのテキストファイル。

1. 準備

必要なライブラリの読み込み

- Pandas：ファイルの読み込みやデータの確認に使用します。
- NumPy：モデル入力用のデータを作成するために使用します。
- Seaborn：可視化のためのライブラリです。Matplotlibのラッパーです。
- Sicket-learn：スケーリングに使用します。
- keras.utils.np_utils：カテゴリカルデータをOne-hot エンコーディングするために使用します。

```
In [1]: 1 import numpy as np
        2 import pandas as pd
        3 import seaborn as sns
        4 from sklearn.preprocessing import MinMaxScaler
        5 from keras.utils.np_utils import to_categorical
```

Using TensorFlow backend.

カレントディレクトリの移動

```
In [2]: 1 # from google.colab import drive
        2 # drive.mount('/gdrive')
        3 # %cd "/gdrive/My Drive/Colab Notebooks/jasla_rensyuu_20190202"
```

参照変数の準備

- アクティビティの番号と内容の対応が取りやすくなるように、辞書型の変数を定義しておきます。
- class_num は、アクティビティの数を代入しておきます。
- nrowsには、ファイルの読み込み行数を設定します。すべてのデータを読み込む場合はNoneを設定します。1回の実行に時間がかかるため、デバッグ中は少ないデータで実行ができるようにします。

```
In [3]: 1 class_num = 6
2 class_names = {1:'Walking',
3                2:'Walk upstairs',
4                3:'Walk downstairs',
5                4:'Sitting',
6                5:'Standing',
7                6:'Laying'}
8
9 # ファイルの読み込み行数を指定します。最大数はtrain:7352, test:2947です。
10 # nrows = 1000
11 nrows = None
```

2. データの読み込み

- データ読み込み用の関数を定義し、まとめて読み込む行数を指定できるようにします。
- 関数内でのファイルの読み込みには、Pandasのread_csv関数を使います。
- データファイルがスペース区切りのため、sepパラメータに“¥s+”を指定します。
- 一つのファイルを、一つのDataFrameとします。
- 必要に応じて、shapeや、headなどでサイズや内容を確認してくみてください。

```
In [4]: 1 def read_csv(filename, nrows=nrows):
2         return pd.read_csv(filename, header=None, sep="¥s+", nrows=nrows)
3
4 # トレーニングデータ
5 y_train = read_csv('data/uci_har/y_train.txt')
6 total_acc_x_train = read_csv('data/uci_har/total_acc_x_train.txt')
7 total_acc_y_train = read_csv('data/uci_har/total_acc_y_train.txt')
8 total_acc_z_train = read_csv('data/uci_har/total_acc_z_train.txt')
9 body_acc_x_train = read_csv('data/uci_har/body_acc_x_train.txt')
10 body_acc_y_train = read_csv('data/uci_har/body_acc_y_train.txt')
11 body_acc_z_train = read_csv('data/uci_har/body_acc_z_train.txt')
12 body_gyro_x_train = read_csv('data/uci_har/body_gyro_x_train.txt')
13 body_gyro_y_train = read_csv('data/uci_har/body_gyro_y_train.txt')
14 body_gyro_z_train = read_csv('data/uci_har/body_gyro_z_train.txt')
15
16 # テストデータ
17 y_test = read_csv('data/uci_har/y_test.txt')
18 total_acc_x_test = read_csv('data/uci_har/total_acc_x_test.txt')
19 total_acc_y_test = read_csv('data/uci_har/total_acc_y_test.txt')
20 total_acc_z_test = read_csv('data/uci_har/total_acc_z_test.txt')
21 body_acc_x_test = read_csv('data/uci_har/body_acc_x_test.txt')
22 body_acc_y_test = read_csv('data/uci_har/body_acc_y_test.txt')
23 body_acc_z_test = read_csv('data/uci_har/body_acc_z_test.txt')
24 body_gyro_x_test = read_csv('data/uci_har/body_gyro_x_test.txt')
25 body_gyro_y_test = read_csv('data/uci_har/body_gyro_y_test.txt')
26 body_gyro_z_test = read_csv('data/uci_har/body_gyro_z_test.txt')
```

```
In [5]: 1 body_gyro_z_test.shape
```

```
Out[5]: (2947, 128)
```

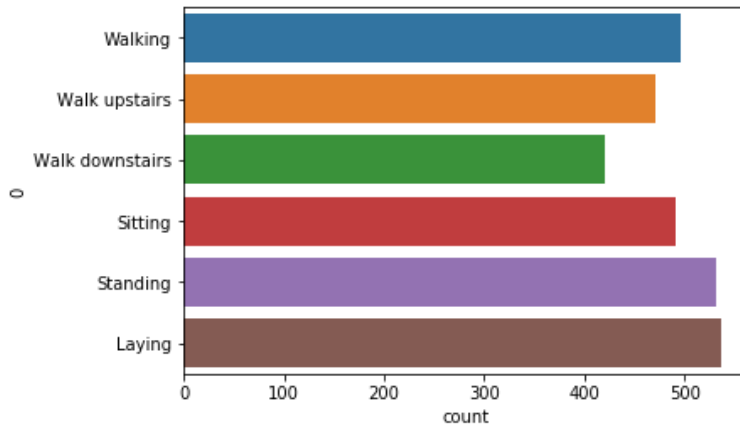
3. データの概要の確認

アクティビティによってデータ数に偏りがないか見てみる

- 可視化ライブラリとして、seabornを使います。
- seabornは、snsという名前でimportしてあります。
- seabornのcountplot関数で、ヒストグラムが表示してみましょう。
- orderパラメータで、出力する順番を指定できます。
- 辞書型のオブジェクトは、.keys()はキーリストを、values()は値のリストを取得できます。
- 以下では、map関数を使って、活動の番号を名前に変換しています。

```
In [6]: 1 sns.countplot(y=y_test[0].map(class_names), order = class_names.values())
```

```
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x2cad82bc320>
```

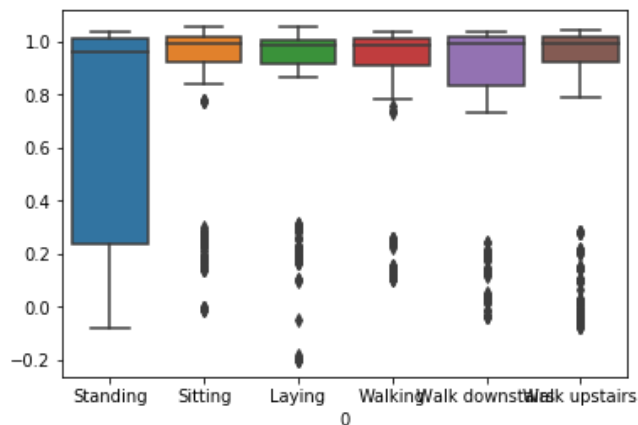


箱ひげ図で値の分布を見る

- アクティビティごとに箱ひげ図の位置が異なっていれば、識別しやすいといえます。

```
In [7]: 1 sns.boxplot(x=y_test[0].map(class_names), y=total_acc_x_train.mean(axis=1))
```

```
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x2caddbb0d30>
```

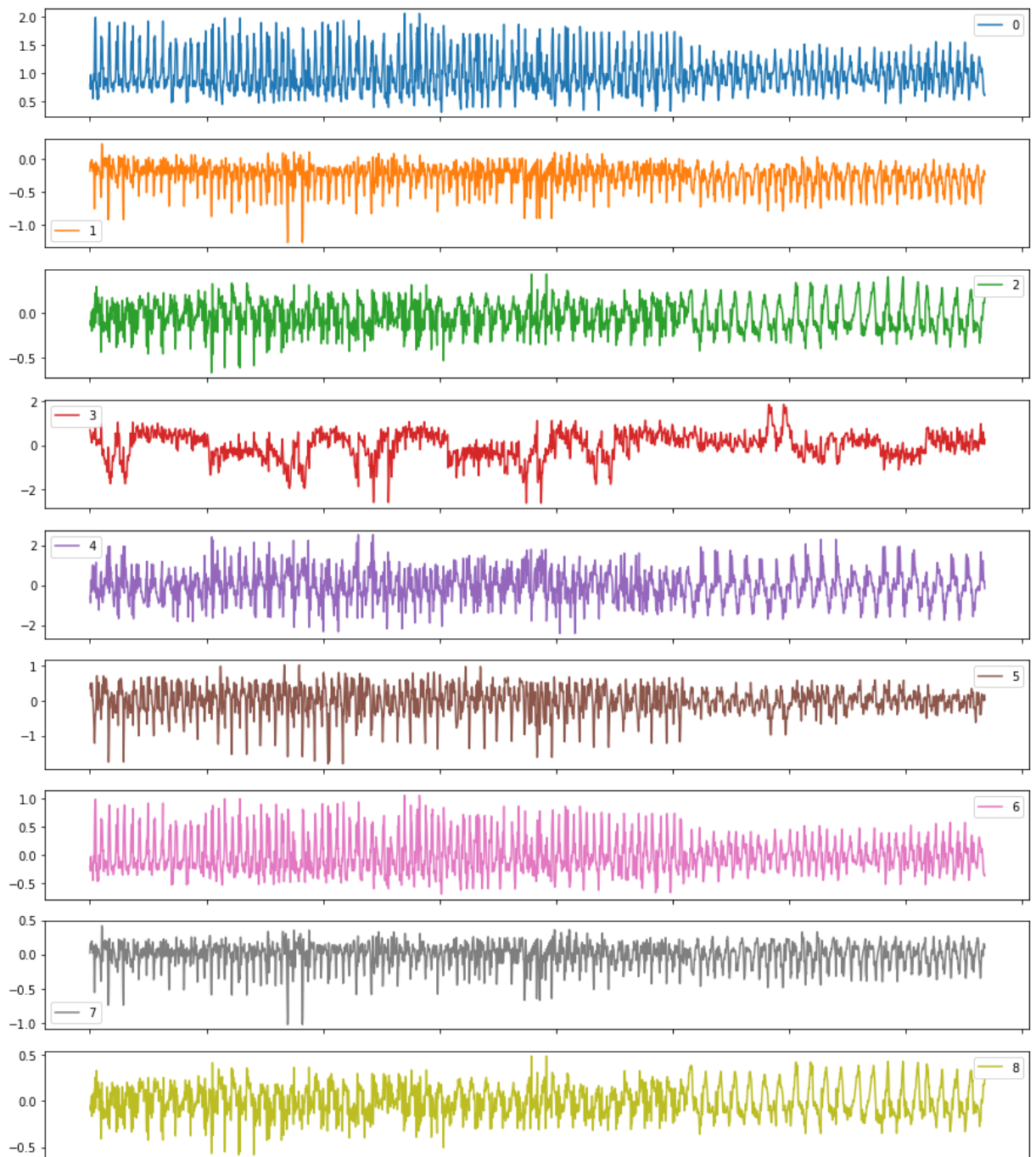


センサー別にデータを表示してみる

- 多くのデータを一度にみたいので、データを一列にならべます。pandasのstack関数を使います。
- i番目からn個のデータを並べて表示してみましょう。

```
In [8]: 1 i=130
2 n=160
3 pd.concat([total_acc_x_train[i:n].stack(), total_acc_y_train[i:n].stack(), total_acc_z_train[i:n].stack(),
4            body_gyro_x_train[i:n].stack(), body_gyro_y_train[i:n].stack(), body_gyro_z_train[i:n].stack(),
5            body_acc_x_train[i:n].stack(), body_acc_y_train[i:n].stack(), body_acc_z_train[i:n].stack()
6            axis=1).plot(subplots=True, figsize=(15, 20))
```

```
Out[8]: array([<matplotlib.axes._subplots.AxesSubplot object at 0x000002CADDDE06A0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x000002CADDCEBFBE0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x000002CADDCE0080>,
<matplotlib.axes._subplots.AxesSubplot object at 0x000002CADD164E0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x000002CADD3D940>,
<matplotlib.axes._subplots.AxesSubplot object at 0x000002CADD67DA0>,
<matplotlib.axes._subplots.AxesSubplot object at 0x000002CADD96240>,
<matplotlib.axes._subplots.AxesSubplot object at 0x000002CADDDE6D8>,
<matplotlib.axes._subplots.AxesSubplot object at 0x000002CADDDE710>],
dtype=object)
```



4. ディープラーニング入力用にデータを変換

- Kerasのモデルに入力できるよう、データのShapeを (サンプル数, シーケンス長, 次元数) の3階のテンソルに変換します。

```
In [9]: 1 def to_3darray(df_list):
2     array_list = []
3     for df in df_list:
4         nd_array = np.array(df)
5         array_list.append(
6             np.reshape(nd_array,
7                         (nd_array.shape[0], nd_array.shape[1], 1)))
8     return np.concatenate(array_list, axis=2)
9
10 # トレーニングデータの特徴量
11 train_df_list = [total_acc_x_train, total_acc_y_train, total_acc_z_train,
12                  body_acc_x_train, body_acc_y_train, body_acc_z_train,
13                  body_gyro_x_train, body_gyro_y_train, body_gyro_z_train]
14 X_train = to_3darray(train_df_list)
15
16 # テストデータの特徴量
17 test_df_list = [total_acc_x_test, total_acc_y_test, total_acc_z_test,
18                 body_acc_x_test, body_acc_y_test, body_acc_z_test,
19                 body_gyro_x_test, body_gyro_y_test, body_gyro_z_test]
20 X_test = to_3darray(test_df_list)
```

- ScikitLearnのMinMaxScalerを使って、データを0から1の範囲にスケーリングしておきます。

```
In [10]: 1 sc = MinMaxScaler()
2 X_train_scf = sc.fit_transform(X_train.reshape(X_train.shape[0] * X_train.shape[1], X_train.shape[2]))
3 X_train = X_train_scf.reshape(X_train.shape)
4 X_test_scf = sc.transform(X_test.reshape(X_test.shape[0] * X_test.shape[1], X_test.shape[2]))
5 X_test = X_test_scf.reshape(X_test.shape)
```

- 正解ラベルをone-hotエンコーディングします。
- 取り扱いを容易にするため、1から6の範囲のIDを、0から5の範囲に変換します。

```
In [11]: 1 Y_train = to_categorical(y_train - 1)
2 Y_test = to_categorical(y_test - 1)
```

- データのShapeを確認しておきましょう。

```
In [12]: 1 X_train.shape, y_train.shape, Y_train.shape, X_test.shape, y_test.shape, Y_test.shape
```

```
Out[12]: ((7352, 128, 9), (7352, 1), (7352, 6), (2947, 128, 9), (2947, 1), (2947, 6))
```

5. ファイルに保存

- 作成したNumPy ndarrayをファイルに保存します。NumPyのsave関数を使います。拡張子は.npyになります。
- 読み込みたいときは、load関数を使います。

```
In [13]: 1 np.save('data/X_train', X_train)
2 np.save('data/Y_train', Y_train)
3 np.save('data/X_test', X_test)
4 np.save('data/Y_test', Y_test)
```


第10章 人の活動の識別：ディープラーニングによる識別（プログラム）

第10章では、第9章で前処理を行った人の活動識別のためのデータセットに対して、ディープラーニングで識別を行います。

学習項目

1. 準備
2. データの準備
3. モデルの定義
4. パラメータの設定
5. トレーニング
6. 評価

使用するファイル

- 10_har_dnn.ipynb（このファイルです）
- data/X_train.npy (訓練用データ)
- data/X_test.npy (テスト用データ)
- data/y_train.npy (訓練用正解ラベルデータ)
- data/y_test.npy (テスト用正解ラベルデータ)

データセット

- UCI Machine Learning Repositoryで公開されているHuman Activity Recognition Using Smartphones Data Set から一部のデータを抜粋し、前処理を行ったもの。
- ファイルは、NumPyで読み込み可能なバイナリ形式の.npyファイル。

1. 準備

必要なライブラリの読み込み

- NumPy：モデル入力用のデータを作成するために使用します。
- Matplotlib：データの可視化に使用します。
- Sicket-learn：metricsパッケージ内のモジュールをモデルの評価に使用します。
- keras：ニューラルネットワークのためのライブラリです。

```
In [1]: 1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 from sklearn.metrics import accuracy_score
5 from sklearn.metrics import confusion_matrix
6 from sklearn.metrics import classification_report
7
8 from keras.models import Sequential
9 from keras.layers import Dense, Activation, Dropout, Flatten
10 from keras.layers.convolutional import Conv1D, MaxPooling1D
11 import keras.backend as K
```

Using TensorFlow backend.

カレントディレクトリの移動

```
In [2]: 1 # from google.colab import drive
2 # drive.mount('/gdrive')
3 # %cd "/gdrive/My Drive/Colab Notebooks/jasla_rensyuu_20190202"
```

参照変数の準備

- アクティビティの番号と内容の対応が取りやすくなるように、辞書型の変数を定義しておきます。
- `class_num` は、アクティビティの数を代入しておきます。

```
In [3]: 1 class_num = 6
        2 class_names = {1:'Walking',
        3                   2:'Walk upstairs',
        4                   3:'Walk downstairs',
        5                   4:'Sitting',
        6                   5:'Standing',
        7                   6:'Laying'}
```

混同行列用関数の定義

- 混同行列を見やすくする関数を定義しておきます。

```
In [4]: 1 def pretty_confusion_matrix(y_true, y_pred, labels=["False", "True"]):
        2     cm = confusion_matrix(y_true, y_pred)
        3     pred_labels = ['(pred) ' + l for l in labels]
        4     df = pd.DataFrame(cm, index=labels, columns=pred_labels)
        5     return df
```

2. データの読み込み

- 第9章で作成したnpzファイルを読み込みます。

```
In [5]: 1 X_train = np.load('data/X_train.npz')
        2 Y_train = np.load('data/Y_train.npz')
        3 X_test = np.load('data/X_test.npz')
        4 Y_test = np.load('data/Y_test.npz')
```

バックエンドのグラフ識別子をリセット

```
In [6]: 1 K.clear_session()
```

3. モデルの定義

- 2層の1次元畳み込み層を使うモデルとしました。
- `summary`関数で、モデルの概要を確認しておきましょう。

```
In [7]: 1 n_steps = X_train.shape[1]
2 n_features = X_train.shape[2]
3 n_outputs = Y_train.shape[1]
4
5 model = Sequential()
6 model.add(Conv1D(filters=32, kernel_size=8, activation='relu', input_shape=(n_steps, n_features)))
7 model.add(Conv1D(filters=16, kernel_size=4, activation='relu'))
8 model.add(Dropout(0.2))
9 model.add(MaxPooling1D(pool_size=2))
10 model.add(Flatten())
11 model.add(Dense(100, activation='relu'))
12 model.add(Dense(n_outputs, activation='softmax'))
13
14 model.summary()
```

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 121, 32)	2336
conv1d_2 (Conv1D)	(None, 118, 16)	2064
dropout_1 (Dropout)	(None, 118, 16)	0
max_pooling1d_1 (MaxPooling1D)	(None, 59, 16)	0
flatten_1 (Flatten)	(None, 944)	0
dense_1 (Dense)	(None, 100)	94500
dense_2 (Dense)	(None, 6)	606
Total params: 99,506		
Trainable params: 99,506		
Non-trainable params: 0		

4. パラメータの設定

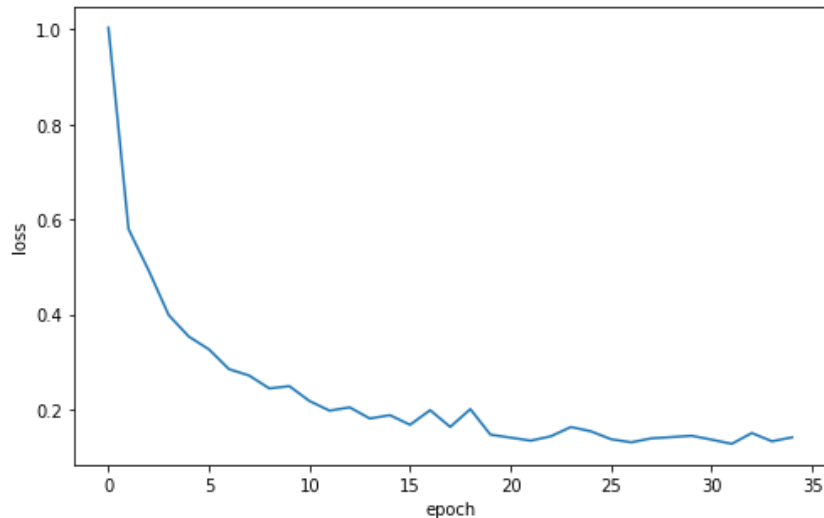
```
In [8]: 1 model.compile(loss='categorical_crossentropy', optimizer='adam')
```

5. トレーニング

- fit関数でモデルのトレーニングを実行します。
- 損失関数の値の推移図をグラフで確認しておきましょう。

```
In [9]: 1 epochs = 35
2 history = model.fit(X_train, Y_train, epochs=epochs, verbose=0)
3
4 plt.figure(figsize=(8, 5))
5 plt.plot(history.history['loss'])
6 plt.ylabel('loss')
7 plt.xlabel('epoch')
```

Out[9]: Text(0.5, 0, 'epoch')



6. 評価

テストデータからトレーニング済みモデルを使って予測

- predict_classes関数にテストデータを渡すことで、トレーニング済みモデルによる推定結果を得ることができます。
- テストデータの正解ラベルY_testはOne-Hot表現になっているため、0から5の範囲の値に戻しておきます。

```
In [10]: 1 y_pred = model.predict_classes(X_test)
2 y_true = np.argmax(Y_test, axis=1)
```

正解率

- accuracy_score関数で、正解率を算出できます。

```
In [11]: 1 accuracy_score(y_true, y_pred)
```

Out[11]: 0.8785205293518833

混同行列

- 混同行列を確認しましょう。見やすく表示するためにpretty_confusion_matrix関数を定義していました。

```
In [12]: 1 pretty_confusion_matrix(y_true, y_pred, class_names.values())
```

Out[12]:

	(pred) Walking	(pred) Walk upstairs	(pred) Walk downstairs	(pred) Sitting	(pred) Standing	(pred) Laying
Walking	456	19	2	0	19	0
Walk upstairs	37	412	6	0	16	0
Walk downstairs	5	41	374	0	0	0
Sitting	1	0	0	325	164	1
Standing	0	0	0	47	485	0
Laying	0	0	0	0	0	537

Classification report

- classification_report関数では、精度 (Precision)や、検出率 (Recall) を表示することができます。

```
In [13]: 1 print(classification_report(y_true, y_pred))
```

	precision	recall	f1-score	support
0	0.91	0.92	0.92	496
1	0.87	0.87	0.87	471
2	0.98	0.89	0.93	420
3	0.87	0.66	0.75	491
4	0.71	0.91	0.80	532
5	1.00	1.00	1.00	537
micro avg	0.88	0.88	0.88	2947
macro avg	0.89	0.88	0.88	2947
weighted avg	0.89	0.88	0.88	2947

