Change point detection in Python with `ruptures`

Digital French-German Summer School with Industry 2020

Charles Truong[1]

[1]Centre Borelli
Université Paris-Saclay
ENS Paris-Saclay, CNRS

Wednesday 24th June

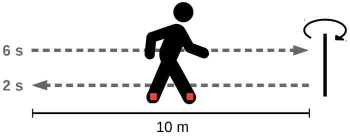
Introduction

- Change point detection is a common task when dealing with non-stationary time series.
- Application example: automatic diagnosis of neurologically impaired patients [Truong et al., 2019a].

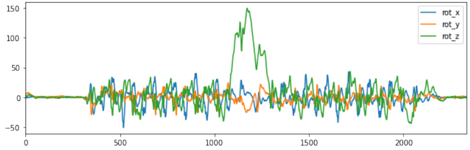
Healthy and pathological subjects underwent a fixed protocol:

- standing still,
- walking 10m,
- turning around,
- walking back,
- standing still.

Protocol schema



Angular velocity (lower back sensor, sampling 100 Hz)



- Can also be applied to finance, industrial monitoring, public health monitoring, etc. [Truong et al., 2020].

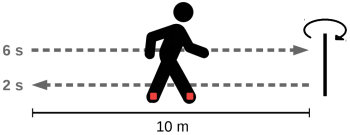
Introduction

- Change point detection is a common task when dealing with non-stationary time series.
- Application example: automatic diagnosis of neurologically impaired patients [Truong et al., 2019a].

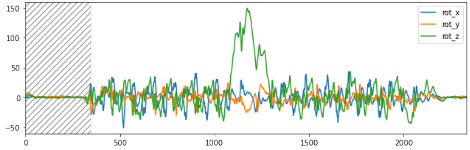
Healthy and pathological subjects underwent a fixed protocol:

- **standing still**,
- walking 10m,
- turning around,
- walking back,
- standing still.

Protocol schema



Angular velocity (lower back sensor, sampling 100 Hz)



- Can also be applied to finance, industrial monitoring, public health monitoring, etc. [Truong et al., 2020].

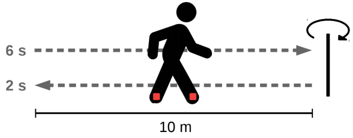
Introduction

- Change point detection is a common task when dealing with non-stationary time series.
- Application example: automatic diagnosis of neurologically impaired patients [Truong et al., 2019a].

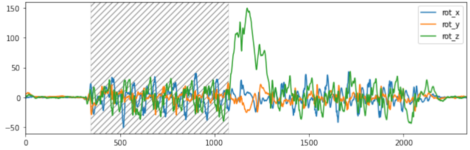
Healthy and pathological subjects underwent a fixed protocol:

- standing still,
- **walking 10m**,
- turning around,
- walking back,
- standing still.

Protocol schema



Angular velocity (lower back sensor, sampling 100 Hz)



- Can also be applied to finance, industrial monitoring, public health monitoring, etc. [Truong et al., 2020].

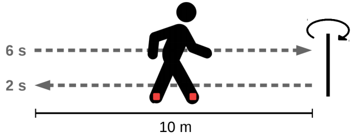
Introduction

- Change point detection is a common task when dealing with non-stationary time series.
- Application example: automatic diagnosis of neurologically impaired patients [Truong et al., 2019a].

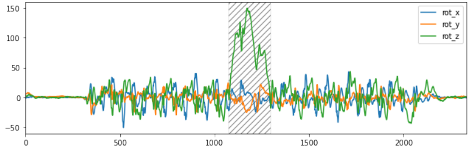
Healthy and pathological subjects underwent a fixed protocol:

- standing still,
- walking 10m,
- **turning around**,
- walking back,
- standing still.

Protocol schema



Angular velocity (lower back sensor, sampling 100 Hz)



- Can also be applied to finance, industrial monitoring, public health monitoring, etc. [Truong et al., 2020].

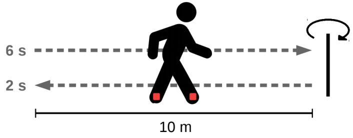
Introduction

- Change point detection is a common task when dealing with non-stationary time series.
- Application example: automatic diagnosis of neurologically impaired patients [Truong et al., 2019a].

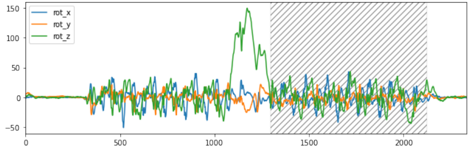
Healthy and pathological subjects underwent a fixed protocol:

- standing still,
- walking 10m,
- turning around,
- **walking back**,
- standing still.

Protocol schema



Angular velocity (lower back sensor, sampling 100 Hz)



- Can also be applied to finance, industrial monitoring, public health monitoring, etc. [Truong et al., 2020].

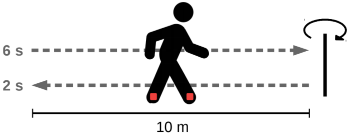
Introduction

- Change point detection is a common task when dealing with non-stationary time series.
- Application example: automatic diagnosis of neurologically impaired patients [Truong et al., 2019a].

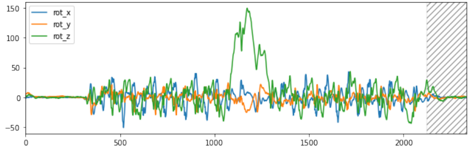
Healthy and pathological subjects underwent a fixed protocol:

- standing still,
- walking 10m,
- turning around,
- walking back,
- **standing still**.

Protocol schema



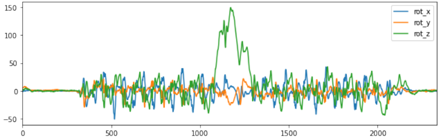
Angular velocity (lower back sensor, sampling 100 Hz)



- Can also be applied to finance, industrial monitoring, public health monitoring, etc. [Truong et al., 2020].

What is change point detection?

- Change point detection consists in finding the temporal boundaries between homogeneous time periods.
- Informally: “multivariate signal $\longrightarrow$ list of change point indexes”

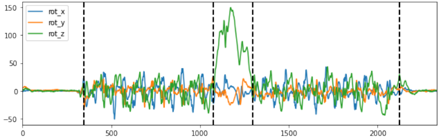


x	y	z
8.139	8.337	-21.055
6.964	9.881	-20.693
4.317	9.993	-19.309
1.752	8.950	-16.941
-0.305	7.356	-13.143
-2.320	7.384	-7.361
-3.312	8.467	-2.530
-3.697	10.891	2.523
-2.622	13.363	5.863
-2.728	11.761	4.473

$\longrightarrow [t_1, t_2, t_3, \ldots]$

What is change point detection?

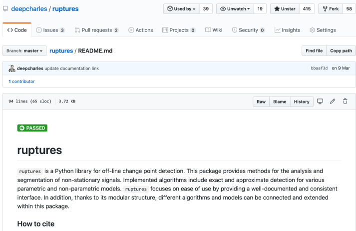
- Change point detection consists in finding the temporal boundaries between homogeneous time periods.
- Informally: “multivariate signal $\longrightarrow$ list of change point indexes”



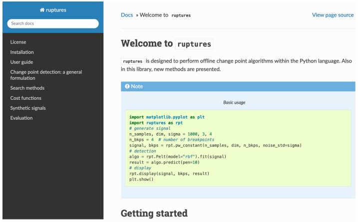
x	y	z
8.139	8.337	-21.055
6.964	9.881	-20.693
4.317	9.993	-19.309
1.752	8.950	-16.941
-0.305	7.356	-13.143
-2.320	7.384	-7.361
-3.312	8.467	-2.530
-3.697	10.891	2.523
-2.622	13.363	5.863
-2.728	11.761	4.473

$\longrightarrow [t_1, t_2, t_3, \ldots]$

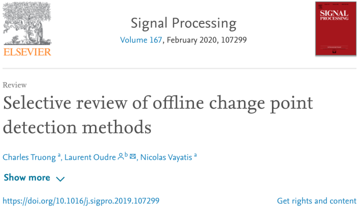
ruptures: a Python library



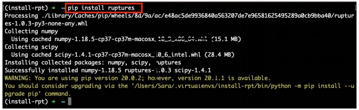
Github page (`github.com/deepcharles/ruptures`)



Documentation



Associated publication [Truong et al., 2020]



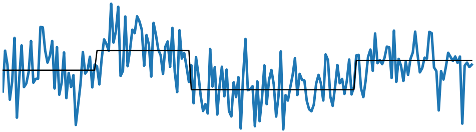
How to install

Links and information on the Github page.

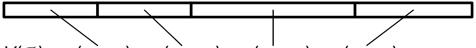
Table of contents

General principle

How to choose a segmentation?



$\mathcal{T} = \{t_1, t_2, t_3\}$



$V(\mathcal{T}) = c(y_{0..t_1}) + c(y_{t_1..t_2}) + c(y_{t_2..t_3}) + c(y_{t_3..T})$

The "best segmentation" is the minimizer, denoted $\widehat{\mathcal{T}}$, of a criterion $V(\mathcal{T})$:

$$V(\mathcal{T}) := \sum_{k=0}^{K} c(y_{t_k..t_{k+1}}).$$

Cost example: $c(y) = \sum_t (y_t - \bar{y})^2$.

Problem 1.
Fixed number K of change points:

$$\widehat{\mathcal{T}} := \arg\min_{\mathcal{T}} \; V(\mathcal{T}) \quad \text{s.t. } |\mathcal{T}| = K.$$

Problem 2.
Unknown number of change points:

$$\widehat{\mathcal{T}} := \arg\min_{\mathcal{T}} \; V(\mathcal{T}) \; + \; \mathrm{pen}(\mathcal{T})$$

where $\mathrm{pen}(\mathcal{T})$ measures the complexity of a segmentation $\mathcal{T}$.

Detection methods are the combination of three elements [Truong et al., 2020].

Cost function | Search method | Constraint

Criterion $V(\mathcal{T})$ to minimize: $$V(\mathcal{T}) := \sum_{k=0}^{K} c(y_{t_k..t_{k+1}}).$$

Problem 1.
Fixed number K of change points:

$$\widehat{\mathcal{T}} := \arg\min_{\mathcal{T}} V(\mathcal{T}) \quad \text{s.t.} \quad |\mathcal{T}| = K.$$

Problem 2.
Unknown number of change points:

$$\widehat{\mathcal{T}} := \arg\min_{\mathcal{T}} V(\mathcal{T}) + \text{pen}(\mathcal{T})$$

where $\text{pen}(\mathcal{T})$ measures the complexity of a segmentation $\mathcal{T}$.

General principle

`ruptures`: a Python library

- A modular architecture.

[319]:
```
import ruptures as rpt

signal = get_signal(...)  # user defined
```

First import and data loading.

[329]:
```
# cost function
c = rpt.costs.CostL2()
```

Choosing the cost function.
Here, $c(y) = \sum_t (y_t - \bar{y})^2$.

[330]:
```
# search method
algo = rpt.Binseg(jump=5, min_size=10, custom_cost=c)
```

Choosing the search method.
Here, binary segmentation.

[331]:
```
# fit algo
algo.fit(signal)
```

Fitting the algorithm.

[332]:
```
# predict change points
# fixed number of changes
bkps = algo.predict(n_bkps=10)
# or penalized detection
bkps = algo.predict(pen=50)
```

Choosing the constraint.
Then detecting the change points (“predict”).

[333]:
```
from ruptures.metrics import hausdorff

error = hausdorff(true_bkps, bkps)
```

Measuring the detection accuracy.

Table of contents

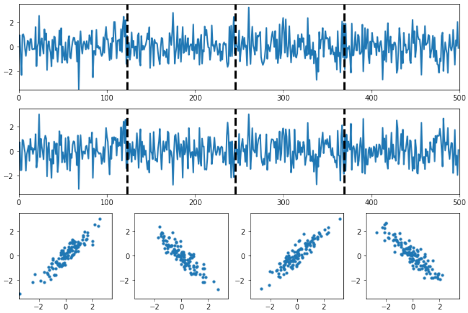
A simple example

ruptures in action

Import and generate signal

[91]:
```
import ruptures as rpt

signal, bkps = rpt.pw_normal(n_samples=500)
```



Choose a method and detect change points

[130]:
```
# cost for multivariate Gaussian
cost = rpt.costs.CostNormal()
# search method: binary segmentation
algo = rpt.Binseg(custom_cost=cost)
# fit
algo.fit(signal)
# predict
prediction = algo.predict(3)
# print results
print(f"True change points:\t\t{bkps}")
print(f"Predicted change points:\t{prediction}")
```

```
True change points:         [123, 246, 369, 500]
Predicted change points:    [120, 245, 370, 500]
```

Measure accurary

[143]:
```
from ruptures.metrics import hausdorff
print(hausdorff(bkps, prediction))
```

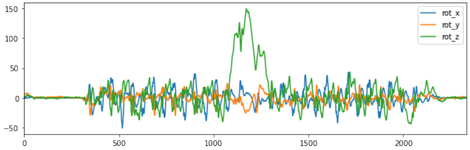
```
3.0
```

Gait analysis

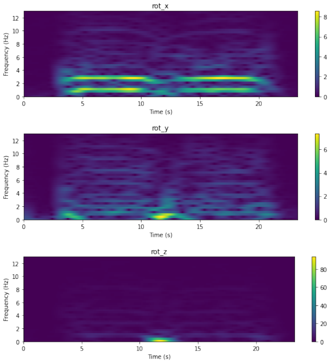
ruptures in action

- To simplify the detection task, the signal is transformed (here, short-term Fourier transform).
- Then mean-shifts are detected.

Angular velocity (lower back sensor, sampling 100 Hz)



Signal transformation

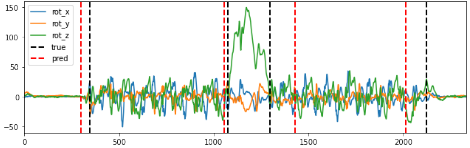


Gait analysis

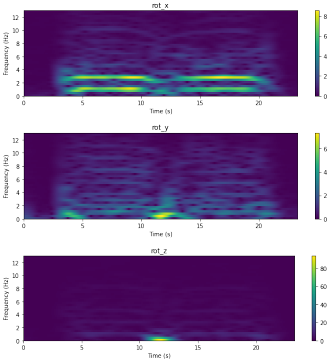
`ruptures` in action

- To simplify the detection task, the signal is transformed (here, short-term Fourier transform).
- Then mean-shifts are detected.

Angular velocity (lower back sensor, sampling 100 Hz)



Signal transformation



[246]:

```python
# cost for multivariate mean-shifts
cost = rpt.costs.CostL2()
# search method: binary segmentation
algo = rpt.Binseg(custom_cost=cost)
# fit
algo.fit(X)
# predict
prediction = algo.predict(4)
# error
error = hausdorff(true_bkps, prediction)/100
# print results
print(f"True change points:\t\t{true_bkps}")
print(f"Predicted change points:\t{prediction}")
print(f"Max error: {error:.2f} sec")
```

```
True change points:         [347, 1075, 1297, 2123, 2332]
Predicted change points:    [300, 1055, 1430, 2015, 2332]
Max error: 1.33 sec
```

Table of contents

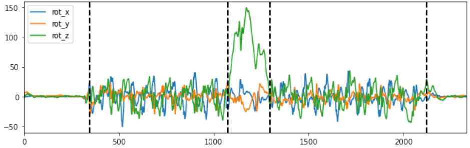
Supervised change point detection

General principle

- How to integrate expert knowledge to calibrate the change point detection? [Truong et al., 2019b]

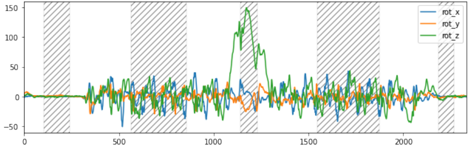
The expert provides the target segmentation: either full or partial label.

Full label



The exact change point locations are provided.

Partial label



Only homogeneous periods (hatched areas) are provided (weakly supervised).

Labels are transformed into constraints. Intuitively, the problem is:

- Learn a transformation Ψ such that
  $d(\Psi(x_t), \Psi(x_s)) \leq u$ if x_t and x_s similar
  $d(\Psi(x_t), \Psi(x_s)) \geq l$ if x_t and x_s dissimilar

($u > 0$ and $l > 0$)

Two samples are *similar* if they belong to the same regime.
Two samples are *dissimilar* if they belong to consecutive regimes.

Deep change point detection

Supervised change point detection

This setting can be used to learn a deep representation.

Here, two layers of temporal separable convolutions and max-pooling (with `tensorflow`).

[256]:
```python
def get_model():
    model = keras.Sequential([
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),])
    return model
```

Learning phase

[262]:
```python
X, y = get_Xy(8)
model = get_model()
model.compile(loss=loss_func,
              optimizer=keras.optimizers.Adam(1e-2))
history = model.fit(X, y, epochs=101, verbose=0,
                    callbacks=CustomCallback())
model.summary()
```

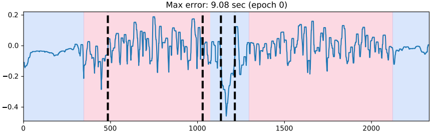
```
Model: "sequential_33"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
separable_conv1d_77 (Separab (None, None, 5)           140
_________________________________________________________________
max_pooling1d_73 (MaxPooling (None, None, 5)           0
_________________________________________________________________
separable_conv1d_78 (Separab (None, None, 1)           166
_________________________________________________________________
max_pooling1d_74 (MaxPooling (None, None, 1)           0
=================================================================
Total params: 306
Trainable params: 306
Non-trainable params: 0
_________________________________________________________________
```

Prediction phase

[267]:
```python
true_bkps = y[0].numpy().tolist()
transformed = model(X)[0].numpy()
prediction = rpt.Binseg(jump=5).fit(transformed).predict(4)
error = hausdorff(true_bkps, prediction)/100
# print results
print(f"True change points:\t\t{true_bkps}")
print(f"Predicted change points:\t{prediction}")
print(f"Max error: {error:.2f} sec")
```

```
True change points:             [347, 1075, 1297, 2123, 2332]
Predicted change points:        [355, 1075, 1295, 2090, 2332]
Max error: 0.33 sec
```

Epoch by epoch (epoch 0)



True segmentation: alternating colors.

Predicted segmentation: dashed lines.

This setting can be used to learn a deep representation.

Here, two layers of temporal separable convolutions and max-pooling (with `tensorflow`).

[256]:
```
def get_model():
    model = keras.Sequential([
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),])
    return model
```

Learning phase

[262]:
```
X, y = get_Xy(8)
model = get_model()
model.compile(loss=loss_func,
              optimizer=keras.optimizers.Adam(1e-2))
history = model.fit(X, y, epochs=101, verbose=0,
                    callbacks=CustomCallback())
model.summary()
```

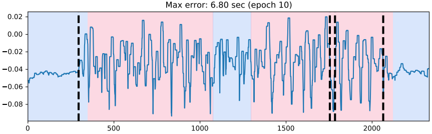
```
Model: "sequential_33"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
separable_conv1d_77 (Separab (None, None, 5)           140
_________________________________________________________________
max_pooling1d_73 (MaxPooling (None, None, 5)           0
_________________________________________________________________
separable_conv1d_78 (Separab (None, None, 1)           166
_________________________________________________________________
max_pooling1d_74 (MaxPooling (None, None, 1)           0
=================================================================
Total params: 306
Trainable params: 306
Non-trainable params: 0
_________________________________________________________________
```

Prediction phase

[267]:
```
true_bkps = y[0].numpy().tolist()
transformed = model(X)[0].numpy()
prediction = rpt.Binseg(jump=5).fit(transformed).predict(4)
error = hausdorff(true_bkps, prediction)/100
# print results
print(f"True change points:\t\t{true_bkps}")
print(f"Predicted change points:\t{prediction}")
print(f"Max error: {error:.2f} sec")
```

```
True change points:          [347, 1075, 1297, 2123, 2332]
Predicted change points:     [355, 1075, 1295, 2090, 2332]
Max error: 0.33 sec
```

Epoch by epoch (epoch 10)



True segmentation: alternating colors.

Predicted segmentation: dashed lines.

Deep change point detection

Supervised change point detection

This setting can be used to learn a deep representation.

Here, two layers of temporal separable convolutions and max-pooling (with `tensorflow`).

[256]:
```python
def get_model():
    model = keras.Sequential([
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),])
    return model
```

Learning phase

[262]:
```python
X, y = get_Xy(8)
model = get_model()
model.compile(loss=loss_func,
              optimizer=keras.optimizers.Adam(1e-2))
history = model.fit(X, y, epochs=101, verbose=0,
                    callbacks=CustomCallback())
model.summary()
```

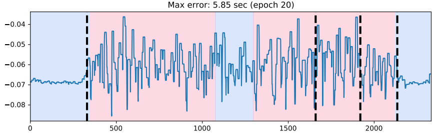
```
Model: "sequential_33"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
separable_conv1d_77 (Separab (None, None, 5)           140
_________________________________________________________________
max_pooling1d_73 (MaxPooling (None, None, 5)           0
_________________________________________________________________
separable_conv1d_78 (Separab (None, None, 1)           166
_________________________________________________________________
max_pooling1d_74 (MaxPooling (None, None, 1)           0
=================================================================
Total params: 306
Trainable params: 306
Non-trainable params: 0
_________________________________________________________________
```

Prediction phase

[267]:
```python
true_bkps = y[0].numpy().tolist()
transformed = model(X)[0].numpy()
prediction = rpt.Binseg(jump=5).fit(transformed).predict(4)
error = hausdorff(true_bkps, prediction)/100
# print results
print(f"True change points:\t\t{true_bkps}")
print(f"Predicted change points:\t{prediction}")
print(f"Max error: {error:.2f} sec")
```

```
True change points:          [347, 1075, 1297, 2123, 2332]
Predicted change points:     [355, 1075, 1295, 2090, 2332]
Max error: 0.33 sec
```

Epoch by epoch (epoch 20)



True segmentation: alternating colors.

Predicted segmentation: dashed lines.

Deep change point detection

Supervised change point detection

This setting can be used to learn a deep representation.

Here, two layers of temporal separable convolutions and max-pooling (with `tensorflow`).

[256]:

```python
def get_model():
    model = keras.Sequential([
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),])
    return model
```

Learning phase

[262]:

```python
X, y = get_Xy(8)
model = get_model()
model.compile(loss=loss_func,
              optimizer=keras.optimizers.Adam(1e-2))
history = model.fit(X, y, epochs=101, verbose=0,
                    callbacks=CustomCallback())
model.summary()
```

```
Model: "sequential_33"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
separable_conv1d_77 (Separab (None, None, 5)           140
_________________________________________________________________
max_pooling1d_73 (MaxPooling (None, None, 5)           0
_________________________________________________________________
separable_conv1d_78 (Separab (None, None, 1)           166
_________________________________________________________________
max_pooling1d_74 (MaxPooling (None, None, 1)           0
=================================================================
Total params: 306
Trainable params: 306
Non-trainable params: 0
_________________________________________________________________
```

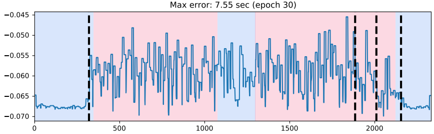
Prediction phase

[267]:

```python
true_bkps = y[0].numpy().tolist()
transformed = model(X)[0].numpy()
prediction = rpt.Binseg(jump=5).fit(transformed).predict(4)
error = hausdorff(true_bkps, prediction)/100
# print results
print(f"True change points:\t\t{true_bkps}")
print(f"Predicted change points:\t{prediction}")
print(f"Max error: {error:.2f} sec")
```

```
True change points:          [347, 1075, 1297, 2123, 2332]
Predicted change points:     [355, 1075, 1295, 2090, 2332]
Max error: 0.33 sec
```

Epoch by epoch (epoch 30)



True segmentation: alternating colors.

Predicted segmentation: dashed lines.

Deep change point detection

Supervised change point detection

This setting can be used to learn a deep representation.

Here, two layers of temporal separable convolutions and max-pooling (with `tensorflow`).

[256]:

```
def get_model():
    model = keras.Sequential([
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),])
    return model
```

Learning phase

[262]:

```
X, y = get_Xy(8)
model = get_model()
model.compile(loss=loss_func,
              optimizer=keras.optimizers.Adam(1e-2))
history = model.fit(X, y, epochs=101, verbose=0,
                    callbacks=CustomCallback())
model.summary()
```

```
Model: "sequential_33"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
separable_conv1d_77 (Separab (None, None, 5)           140
_________________________________________________________________
max_pooling1d_73 (MaxPooling (None, None, 5)           0
_________________________________________________________________
separable_conv1d_78 (Separab (None, None, 1)           166
_________________________________________________________________
max_pooling1d_74 (MaxPooling (None, None, 1)           0
=================================================================
Total params: 306
Trainable params: 306
Non-trainable params: 0
_________________________________________________________________
```

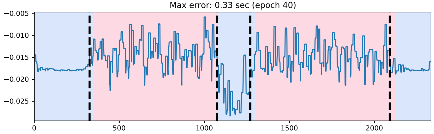
Prediction phase

[267]:

```
true_bkps = y[0].numpy().tolist()
transformed = model(X)[0].numpy()
prediction = rpt.Binseg(jump=5).fit(transformed).predict(4)
error = hausdorff(true_bkps, prediction)/100
# print results
print(f"True change points:\t\t{true_bkps}")
print(f"Predicted change points:\t{prediction}")
print(f"Max error: {error:.2f} sec")
```

```
True change points:          [347, 1075, 1297, 2123, 2332]
Predicted change points:     [355, 1075, 1295, 2090, 2332]
Max error: 0.33 sec
```

Epoch by epoch (epoch 40)



True segmentation: alternating colors.

Predicted segmentation: dashed lines.

This setting can be used to learn a deep representation.

Here, two layers of temporal separable convolutions and max-pooling (with `tensorflow`).

[256]:
```python
def get_model():
    model = keras.Sequential([
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),])
    return model
```

Learning phase

[262]:
```python
X, y = get_Xy(8)
model = get_model()
model.compile(loss=loss_func,
              optimizer=keras.optimizers.Adam(1e-2))
history = model.fit(X, y, epochs=101, verbose=0,
                    callbacks=CustomCallback())
model.summary()
```

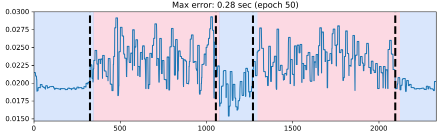
```
Model: "sequential_33"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
separable_conv1d_77 (Separab (None, None, 5)           140
_________________________________________________________________
max_pooling1d_73 (MaxPooling (None, None, 5)           0
_________________________________________________________________
separable_conv1d_78 (Separab (None, None, 1)           166
_________________________________________________________________
max_pooling1d_74 (MaxPooling (None, None, 1)           0
=================================================================
Total params: 306
Trainable params: 306
Non-trainable params: 0
_________________________________________________________________
```

Prediction phase

[267]:
```python
true_bkps = y[0].numpy().tolist()
transformed = model(X)[0].numpy()
prediction = rpt.Binseg(jump=5).fit(transformed).predict(4)
error = hausdorff(true_bkps, prediction)/100
# print results
print(f"True change points:\t\t{true_bkps}")
print(f"Predicted change points:\t{prediction}")
print(f"Max error: {error:.2f} sec")
```

```
True change points:           [347, 1075, 1297, 2123, 2332]
Predicted change points:      [355, 1075, 1295, 2090, 2332]
Max error: 0.33 sec
```

Epoch by epoch (epoch 50)



True segmentation: alternating colors.

Predicted segmentation: dashed lines.

Deep change point detection

Supervised change point detection

This setting can be used to learn a deep representation.

Here, two layers of temporal separable convolutions and max-pooling (with `tensorflow`).

[256]:
```python
def get_model():
    model = keras.Sequential([
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),])
    return model
```

Learning phase

[262]:
```python
X, y = get_Xy(8)
model = get_model()
model.compile(loss=loss_func,
              optimizer=keras.optimizers.Adam(1e-2))
history = model.fit(X, y, epochs=101, verbose=0,
                    callbacks=CustomCallback())
model.summary()
```

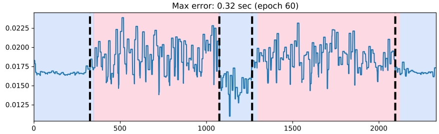
```
Model: "sequential_33"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
separable_conv1d_77 (Separab (None, None, 5)           140
_________________________________________________________________
max_pooling1d_73 (MaxPooling (None, None, 5)           0
_________________________________________________________________
separable_conv1d_78 (Separab (None, None, 1)           166
_________________________________________________________________
max_pooling1d_74 (MaxPooling (None, None, 1)           0
=================================================================
Total params: 306
Trainable params: 306
Non-trainable params: 0
_________________________________________________________________
```

Prediction phase

[267]:
```python
true_bkps = y[0].numpy().tolist()
transformed = model(X)[0].numpy()
prediction = rpt.Binseg(jump=5).fit(transformed).predict(4)
error = hausdorff(true_bkps, prediction)/100
# print results
print(f"True change points:\t\t{true_bkps}")
print(f"Predicted change points:\t{prediction}")
print(f"Max error: {error:.2f} sec")
```

```
True change points:         [347, 1075, 1297, 2123, 2332]
Predicted change points:    [355, 1075, 1295, 2090, 2332]
Max error: 0.33 sec
```

Epoch by epoch (epoch 60)



True segmentation: alternating colors.

Predicted segmentation: dashed lines.

This setting can be used to learn a deep representation.

Here, two layers of temporal separable convolutions and max-pooling (with `tensorflow`).

```
[256]:
def get_model():
    model = keras.Sequential([
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),])
    return model
```

Learning phase

```
[262]:
X, y = get_Xy(8)
model = get_model()
model.compile(loss=loss_func,
              optimizer=keras.optimizers.Adam(1e-2))
history = model.fit(X, y, epochs=101, verbose=0,
                    callbacks=CustomCallback())
model.summary()
```

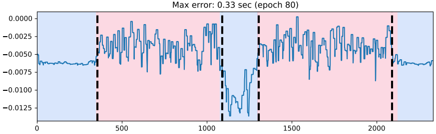
```
Model: "sequential_33"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
separable_conv1d_77 (Separab (None, None, 5)           140
_________________________________________________________________
max_pooling1d_73 (MaxPooling (None, None, 5)           0
_________________________________________________________________
separable_conv1d_78 (Separab (None, None, 1)           166
_________________________________________________________________
max_pooling1d_74 (MaxPooling (None, None, 1)           0
=================================================================
Total params: 306
Trainable params: 306
Non-trainable params: 0
_________________________________________________________________
```

Prediction phase

```
[267]:
true_bkps = y[0].numpy().tolist()
transformed = model(X)[0].numpy()
prediction = rpt.Binseg(jump=5).fit(transformed).predict(4)
error = hausdorff(true_bkps, prediction)/100
# print results
print(f"True change points:\t\t{true_bkps}")
print(f"Predicted change points:\t{prediction}")
print(f"Max error: {error:.2f} sec")
```

```
True change points:          [347, 1075, 1297, 2123, 2332]
Predicted change points:     [355, 1075, 1295, 2090, 2332]
Max error: 0.33 sec
```

Epoch by epoch (epoch 80)



True segmentation: alternating colors.

Predicted segmentation: dashed lines.

Deep change point detection

Supervised change point detection

This setting can be used to learn a deep representation.

Here, two layers of temporal separable convolutions and max-pooling (with `tensorflow`).

[256]:
```python
def get_model():
    model = keras.Sequential([
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),])
    return model
```

Learning phase

[262]:
```python
X, y = get_Xy(8)
model = get_model()
model.compile(loss=loss_func,
              optimizer=keras.optimizers.Adam(1e-2))
history = model.fit(X, y, epochs=101, verbose=0,
                    callbacks=CustomCallback())
model.summary()
```

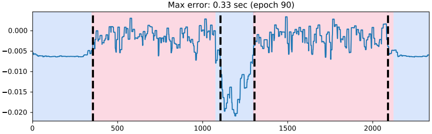
```
Model: "sequential_33"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
separable_conv1d_77 (Separab (None, None, 5)           140
_________________________________________________________________
max_pooling1d_73 (MaxPooling (None, None, 5)           0
_________________________________________________________________
separable_conv1d_78 (Separab (None, None, 1)           166
_________________________________________________________________
max_pooling1d_74 (MaxPooling (None, None, 1)           0
=================================================================
Total params: 306
Trainable params: 306
Non-trainable params: 0
_________________________________________________________________
```

Prediction phase

[267]:
```python
true_bkps = y[0].numpy().tolist()
transformed = model(X)[0].numpy()
prediction = rpt.Binseg(jump=5).fit(transformed).predict(4)
error = hausdorff(true_bkps, prediction)/100
# print results
print(f"True change points:\t\t{true_bkps}")
print(f"Predicted change points:\t{prediction}")
print(f"Max error: {error:.2f} sec")
```

```
True change points:           [347, 1075, 1297, 2123, 2332]
Predicted change points:      [355, 1075, 1295, 2090, 2332]
Max error: 0.33 sec
```

Epoch by epoch (epoch 90)



True segmentation: alternating colors.

Predicted segmentation: dashed lines.

Deep change point detection

Supervised change point detection

This setting can be used to learn a deep representation.

Here, two layers of temporal separable convolutions and max-pooling (with `tensorflow`).

[256]:
```python
def get_model():
    model = keras.Sequential([
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),
        keras.layers.SeparableConv1D(...),
        keras.layers.MaxPool1D(...),])
    return model
```

Learning phase

[262]:
```python
X, y = get_Xy(8)
model = get_model()
model.compile(loss=loss_func,
              optimizer=keras.optimizers.Adam(1e-2))
history = model.fit(X, y, epochs=101, verbose=0,
                    callbacks=CustomCallback())
model.summary()
```

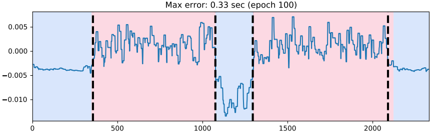
```
Model: "sequential_33"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
separable_conv1d_77 (Separab (None, None, 5)           140
_________________________________________________________________
max_pooling1d_73 (MaxPooling (None, None, 5)           0
_________________________________________________________________
separable_conv1d_78 (Separab (None, None, 1)           166
_________________________________________________________________
max_pooling1d_74 (MaxPooling (None, None, 1)           0
=================================================================
Total params: 306
Trainable params: 306
Non-trainable params: 0
_________________________________________________________________
```

Prediction phase

[267]:
```python
true_bkps = y[0].numpy().tolist()
transformed = model(X)[0].numpy()
prediction = rpt.Binseg(jump=5).fit(transformed).predict(4)
error = hausdorff(true_bkps, prediction)/100
# print results
print(f"True change points:\t\t{true_bkps}")
print(f"Predicted change points:\t{prediction}")
print(f"Max error: {error:.2f} sec")
```

```
True change points:          [347, 1075, 1297, 2123, 2332]
Predicted change points:     [355, 1075, 1295, 2090, 2332]
Max error: 0.33 sec
```

Epoch by epoch (epoch 100)



True segmentation: alternating colors.

Predicted segmentation: dashed lines.

Conclusion

- Code for those experiments will be available on my GitHub `github.com/deepcharles`.
- Data set is already published [Truong et al., 2019a].
- New methods are frequently implemented in ruptures.
- Extensions to graph/network data soon.

References

Truong, C., Barrois-Müller, R., Moreau, T., Provost, C., Vienne-Jumeau, A., Moreau, A., Vidal, P.-P., Vayatis, N., Buffat, S., Yelnik, A., Ricard, D., and Oudre, L. (2019a).
A data set for the study of human locomotion with inertial measurements units.
Image Processing On Line, 9.

Truong, C., Oudre, L., and Vayatis, N. (2019b).
Supervised kernel change point detection with partial annotations.
In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5, Brighton, UK.

Truong, C., Oudre, L., and Vayatis, N. (2020).
Selective review of offline change point detection methods.
Signal Processing, 167.