

Change point detection in Python with ruptures

Digital French-German Summer School with Industry 2020

Charles Truong¹

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

Wednesday 24th June

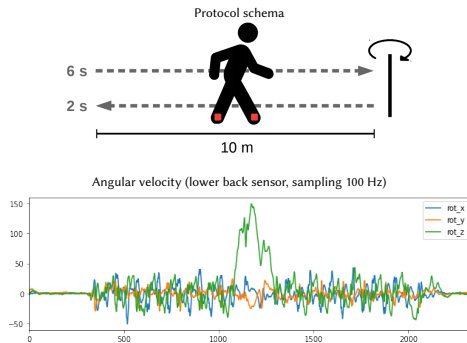


Introduction

- ▶ 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.



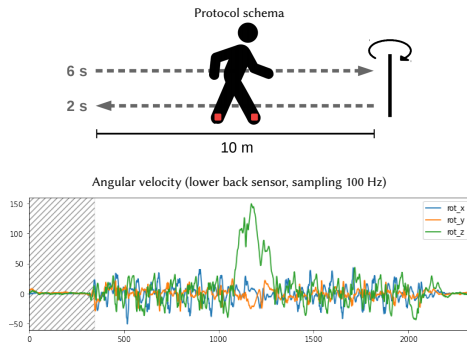
- ▶ Can also be applied to finance, industrial monitoring, public health monitoring, etc. [Truong et al., 2020].

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.



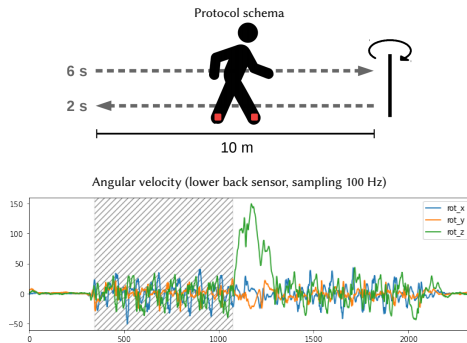
- ▶ Can also be applied to finance, industrial monitoring, public health monitoring, etc. [Truong et al., 2020].

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.



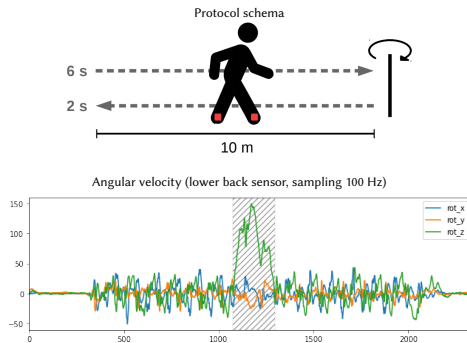
- ▶ Can also be applied to finance, industrial monitoring, public health monitoring, etc. [Truong et al., 2020].

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.



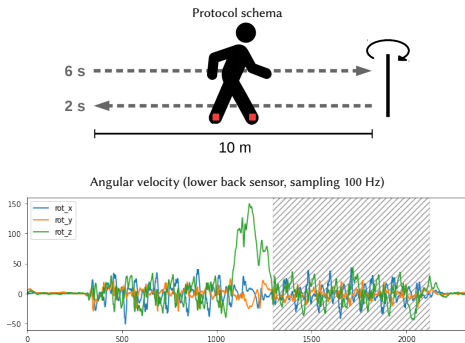
- ▶ Can also be applied to finance, industrial monitoring, public health monitoring, etc. [Truong et al., 2020].

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.



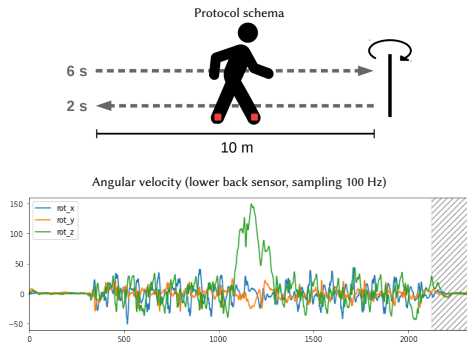
- ▶ Can also be applied to finance, industrial monitoring, public health monitoring, etc. [Truong et al., 2020].

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:

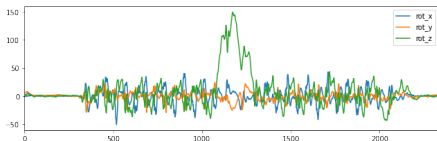
- standing still,
- walking 10m,
- turning around,
- walking back,
- **standing still.**



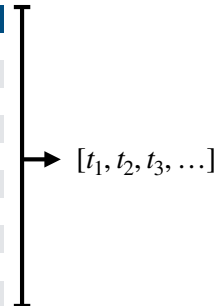
- ▶ Can also be applied to finance, industrial monitoring, public health monitoring, etc. [Truong et al., 2020].

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”

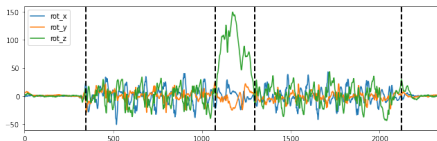


x	y	z
8.139	8.337	-21.055
6.964	9.881	-20.693
4.317	9.993	-19.309
1.752	8.950	-16.941
-0.305	7.356	-13.143
-2.320	7.384	-7.361
-3.312	8.467	-2.530
-3.697	10.891	2.523
-2.622	13.363	5.863
-2.728	11.761	4.473



What is change point detection?

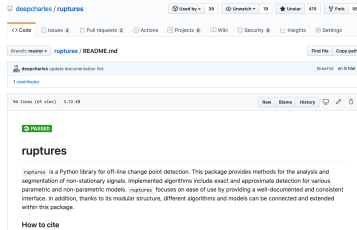
- ▶ Change point detection consists in finding the temporal boundaries between homogeneous time periods.
- ▶ Informally: “multivariate signal \longrightarrow list of change point indexes”



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

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

ruptures: a Python library



deepcharles/ruptures

Used by: 39 | Unwatch: 19 | Watch: 416 | Fork: 58

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

Branch: master | ruptures / README.md | Final file | Copy path

deepcharles update documentation link | 1 contributor

64 lines (45 view) | 3.72 KB | Run | Blame | History

PASSED

ruptures

ruptures is a Python library for off-line change point detection. This package provides methods for the analysis and segmentation of non-stationary signals. Implemented algorithms include exact and approximate detection for various parametric and non-parametric models. ruptures focuses on ease of use by providing a well-documented and consistent interface. In addition, thanks to its modular structure, different algorithms and models can be connected and extended within this package.

How to cite

Github page (github.com/deepcharles/ruptures)



Review

Selective review of offline change point detection methods

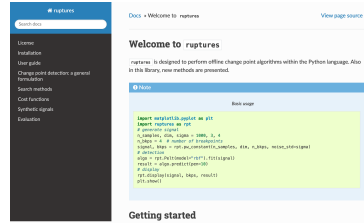
Charles Truong^a, Laurent Oudre^{a,b}, Nicolas Vayatis^a

Show more

<https://doi.org/10.1016/j.sigpro.2019.107299>

Get rights and content

Associated publication [Truong et al., 2020]



ruptures

Search docs

User guide | Installation | User guide | Change point detection: a general | Formulation | Search methods | Cost functions | Synthetic signals | Evaluation

Docs > Welcome to ruptures | View page source

Welcome to ruptures

ruptures is designed to perform offline change point algorithms within the Python language. Also in this library, new methods are presented.

How to use

Back stage

```
import matplotlib.pyplot as plt
import ruptures as rpt
# generate signal
n_samples, dim, npts = 1000, 3, 4
n_kpts = 4 # number of breakpoints
signal, kpts = rpt._generate_signal(n_samples, dim, n_kpts, noise_std=0.1)
# detection
algo = rpt.HMMModel(rpt.LTICSignal)
results = algo.predict(npts)
# display
plt.plot(signal, kpts, results)
plt.show()
```

Getting started

Documentation

```
(install-rpt) ~ - pip install ruptures
Processing ./library/caches/pip/whl/c/8d/9a/cc/e48a5d993684b0c563287de7e905162549528a8c39bb048/ruptur
es-1.0.3-py3-none-any.whl
Collecting numpy
Using cached numpy-1.18.5-cp37-cp37m-macosx_10_9_arm64_t80201903.whl (35.1 MB)
Collecting scipy
Using cached scipy-1.4.1-cp37-cp37m-macosx_10_9_arm64_t80201903.whl (28.4 MB)
Installing collected packages: numpy, scipy, ruptures
Successfully installed numpy-1.18.5 scipy-1.4.1 ruptures-1.0.3
WARNING: You are using pip version 20.0.2; however, version 20.1.1 is available.
You should consider upgrading via the '/Users/Sara/.virtualenvs/install-rpt/bin/python -m pip install --u
pgrade pip' command.
(install-rpt) ~ -
```

How to install

Links and information on the Github page.

Table of contents

1. Introduction

2. What is change point detection?

3. General principle of ruptures

4. ruptures in action

A simple example

Gait analysis

5. Supervised change point detection

General principle

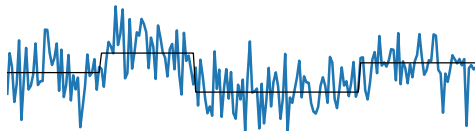
Deep detection

6. Conclusion

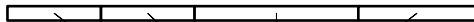
General principle

ruptures: a Python library

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 $\hat{\mathcal{T}}$, 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:

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

Problem 2.

Unknown number of change points:

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

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

General principle

ruptures: a Python library

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

Cost function

Search method

Constraint

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

Problem 1.

Fixed number K of change points:

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

Problem 2.

Unknown number of change points:

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

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

General principle

ruptures: a Python library

► A modular architecture.

```
[319]: 1 import ruptures as rpt
      2
      3 signal = get_signal(...) # user defined
```

First import and data loading.

```
[329]: 1 # cost function
      2 c = rpt.costs.CostL2()
```

Choosing the **cost function**.

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

```
[330]: 1 # search method
      2 algo = rpt.Binseg(jump=5, min_size=10, custom_cost=c)
```

Choosing the **search method**.

Here, binary segmentation.

```
[331]: 1 # fit algo
      2 algo.fit(signal)
```

Fitting the algorithm.

```
[332]: 1 # predict change points
      2 # fixed number of changes
      3 bkps = algo.predict(n_bkps=10)
      4 # or penalized detection
      5 bkps = algo.predict(pen=50)
```

Choosing the **constraint**.

Then detecting the change points ("predict").

```
[333]: 1 from ruptures.metrics import hausdorff
      2
      3 error = hausdorff(true_bkps, bkps)
```

Measuring the detection accuracy.

Table of contents

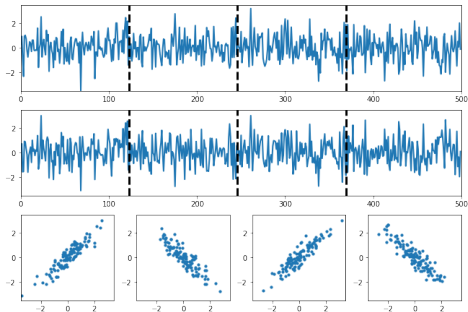
- 1. Introduction
- 2. What is change point detection?
- 3. General principle of ruptures
- 4. **ruptures in action**
 - A simple example
 - Gait analysis
- 5. Supervised change point detection
 - General principle
 - Deep detection
- 6. Conclusion

A simple example

ruptures in action

Import and generate signal

```
[91]: 1 import ruptures as rpt
      2
      3 signal, bkps = rpt.pw_normal(n_samples=500)
```



Choose a method and detect change points

```
[130]: 1 # cost for multivariate Gaussian
      2 cost = rpt.costs.CostNormal()
      3 # search method: binary segmentation
      4 algo = rpt.Binseg(custom_cost=cost)
      5 # fit
      6 algo.fit(signal)
      7 # predict
      8 prediction = algo.predict(3)
      9 # print results
     10 print(f"True change points:\t\t{bkps}")
     11 print(f"Predicted change points:\t{prediction}")
```

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

Measure accuracy

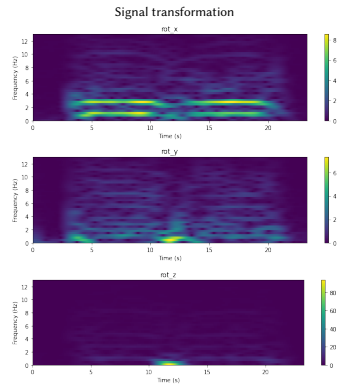
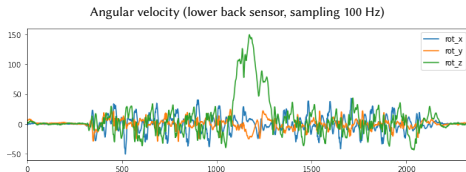
```
[143]: 1 from ruptures.metrics import hausdorff
      2 print(hausdorff(bkps, prediction))
```

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.

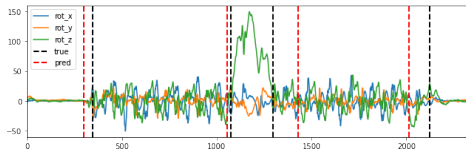


Gait analysis

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)



```
[246]: 1 # cost for multivariate mean-shifts
2 cost = rpt.costs.CostL2()
3 # search method: binary segmentation
4 algo = rpt.Binseg(custom_cost=cost)
5 # fit
6 algo.fit(X)
7 # predict
8 prediction = algo.predict(4)
9 # error
10 error = hausdorff(true_bkps, prediction)/100
11 # print results
12 print(f"True change points:\t\t{true_bkps}")
13 print(f"Predicted change points:\t{prediction}")
14 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

Signal transformation

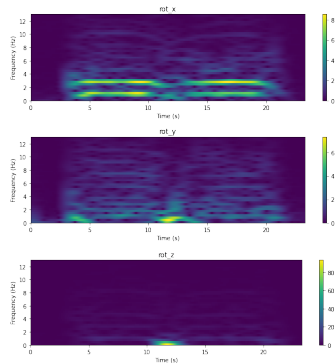


Table of contents

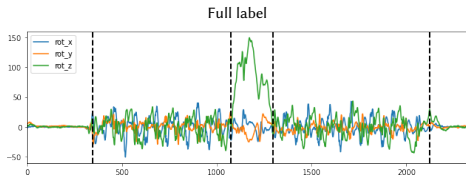
1. Introduction
2. What is change point detection?
3. General principle of ruptures
4. ruptures in action
 - A simple example
 - Gait analysis
5. Supervised change point detection
 - General principle
 - Deep detection
6. Conclusion

Supervised change point detection

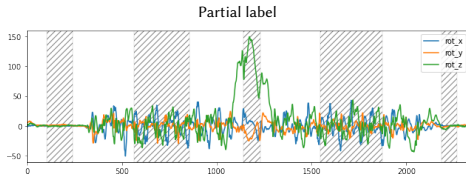
General principle

- 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.



The exact change point locations are provided.



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

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

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

($u > 0$ 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]: 1 def get_model():
2     model = keras.Sequential([
3         keras.layers.SeparableConv1D(...),
4         keras.layers.MaxPool1D(...),
5         keras.layers.SeparableConv1D(...),
6         keras.layers.MaxPool1D(...),])
7     return model
```

Learning phase

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

Model: "sequential_33"

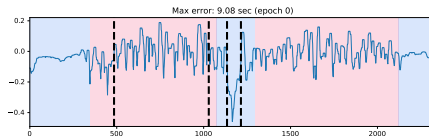
Layer (type)	Output Shape	Param #
=====		
separable_conv1d_77 (Separab	(None, None, 5)	140
max_pooling1d_73 (MaxPooling	(None, None, 5)	0
separable_conv1d_78 (Separab	(None, None, 1)	166
max_pooling1d_74 (MaxPooling	(None, None, 1)	0
=====		
Total params: 306		
Trainable params: 306		
Non-trainable params: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

Prediction phase

```
[267]: 1 true_bkps = y[0].numpy().tolist()
2 transformed = model(X)[0].numpy()
3 prediction = rpt.Binseg(jump=5).fit(transformed).predict(4)
4 error = hausdorff(true_bkps, prediction)/100
5 # print results
6 print(f"True change points:\t\t{true_bkps}")
7 print(f"Predicted change points:\t{prediction}")
8 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.

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]: 1 def get_model():
2     model = keras.Sequential([
3         keras.layers.SeparableConv1D(...),
4         keras.layers.MaxPool1D(...),
5         keras.layers.SeparableConv1D(...),
6         keras.layers.MaxPool1D(...),])
7     return model
```

Learning phase

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

Model: "sequential_33"

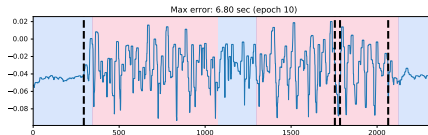
Layer (type)	Output Shape	Param #
=====		
separable_conv1d_77 (Separab	(None, None, 5)	140
max_pooling1d_73 (MaxPooling	(None, None, 5)	0
separable_conv1d_78 (Separab	(None, None, 1)	166
max_pooling1d_74 (MaxPooling	(None, None, 1)	0
=====		
Total params: 306		
Trainable params: 306		
Non-trainable params: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

Prediction phase

```
[267]: 1 true_bkps = y[0].numpy().tolist()
2     transformed = model(X)[0].numpy()
3     prediction = rpt.Binseg(jump=5).fit(transformed).predict(4)
4     error = hausdorff(true_bkps, prediction)/100
5     # print results
6     print(f"True change points:\t\t{true_bkps}")
7     print(f"Predicted change points:\t{prediction}")
8     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]: 1 def get_model():
2     model = keras.Sequential([
3         keras.layers.SeparableConv1D(...),
4         keras.layers.MaxPool1D(...),
5         keras.layers.SeparableConv1D(...),
6         keras.layers.MaxPool1D(...),])
7     return model
```

Learning phase

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

Model: "sequential_33"

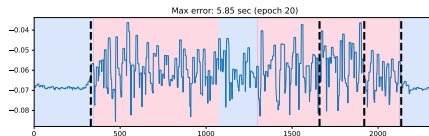
Layer (type)	Output Shape	Param #
=====		
separable_conv1d_77 (Separab	(None, None, 5)	140
max_pooling1d_73 (MaxPooling	(None, None, 5)	0
separable_conv1d_78 (Separab	(None, None, 1)	166
max_pooling1d_74 (MaxPooling	(None, None, 1)	0
=====		
Total params: 306		
Trainable params: 306		
Non-trainable params: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

Prediction phase

```
[267]: 1 true_bkps = y[0].numpy().tolist()
2 transformed = model(X)[0].numpy()
3 prediction = rpt.Binseg(jump=5).fit(transformed).predict(4)
4 error = hausdorff(true_bkps, prediction)/100
5 # print results
6 print(f"True change points:\t\t{true_bkps}")
7 print(f"Predicted change points:\t{prediction}")
8 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]: 1 def get_model():
2       model = keras.Sequential([
3           keras.layers.SeparableConv1D(...),
4           keras.layers.MaxPool1D(...),
5           keras.layers.SeparableConv1D(...),
6           keras.layers.MaxPool1D(...),])
7       return model
```

Learning phase

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

Model: "sequential_33"

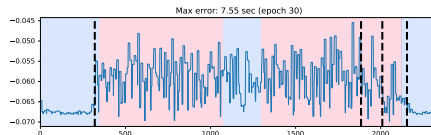
Layer (type)	Output Shape	Param #
=====		
separable_conv1d_77 (Separab	(None, None, 5)	140
max_pooling1d_73 (MaxPooling	(None, None, 5)	0
separable_conv1d_78 (Separab	(None, None, 1)	166
max_pooling1d_74 (MaxPooling	(None, None, 1)	0
=====		
Total params: 306		
Trainable params: 306		
Non-trainable params: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

Prediction phase

```
[267]: 1 true_bkps = y[0].numpy().tolist()
2 transformed = model(X)[0].numpy()
3 prediction = rpt.Binseg(jump=5).fit(transformed).predict(4)
4 error = hausdorff(true_bkps, prediction)/100
5 # print results
6 print(f"True change points:\t\t{true_bkps}")
7 print(f"Predicted change points:\t{prediction}")
8 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]: 1 def get_model():
2     model = keras.Sequential([
3         keras.layers.SeparableConv1D(...),
4         keras.layers.MaxPool1D(...),
5         keras.layers.SeparableConv1D(...),
6         keras.layers.MaxPool1D(...),])
7     return model
```

Learning phase

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

Model: "sequential_33"

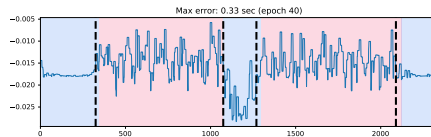
Layer (type)	Output Shape	Param #
=====		
separable_conv1d_77 (Separab	(None, None, 5)	140
max_pooling1d_73 (MaxPooling	(None, None, 5)	0
separable_conv1d_78 (Separab	(None, None, 1)	166
max_pooling1d_74 (MaxPooling	(None, None, 1)	0
=====		
Total params: 306		
Trainable params: 306		
Non-trainable params: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

Prediction phase

```
[267]: 1 true_bkps = y[0].numpy().tolist()
2 transformed = model(X)[0].numpy()
3 prediction = rpt.Binseg(jump=5).fit(transformed).predict(4)
4 error = hausdorff(true_bkps, prediction)/100
5 # print results
6 print(f"True change points:\t\t{true_bkps}")
7 print(f"Predicted change points:\t{prediction}")
8 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.

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]: 1 def get_model():
2       model = keras.Sequential([
3           keras.layers.SeparableConv1D(...),
4           keras.layers.MaxPool1D(...),
5           keras.layers.SeparableConv1D(...),
6           keras.layers.MaxPool1D(...),])
7       return model
```

Learning phase

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

Model: "sequential_33"

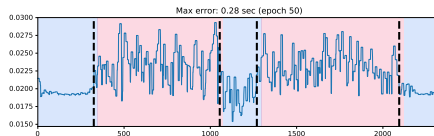
Layer (type)	Output Shape	Param #
=====		
separable_conv1d_77 (Separab	(None, None, 5)	140
max_pooling1d_73 (MaxPooling	(None, None, 5)	0
separable_conv1d_78 (Separab	(None, None, 1)	166
max_pooling1d_74 (MaxPooling	(None, None, 1)	0
=====		
Total params: 306		
Trainable params: 306		
Non-trainable params: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

Prediction phase

```
[267]: 1 true_bkps = y[0].numpy().tolist()
2 transformed = model(X)[0].numpy()
3 prediction = rpt.Binseg(jump=5).fit(transformed).predict(4)
4 error = hausdorff(true_bkps, prediction)/100
5 # print results
6 print(f"True change points:\t\t{true_bkps}")
7 print(f"Predicted change points:\t{prediction}")
8 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]: 1 def get_model():
2       model = keras.Sequential([
3           keras.layers.SeparableConv1D(...),
4           keras.layers.MaxPool1D(...),
5           keras.layers.SeparableConv1D(...),
6           keras.layers.MaxPool1D(...),])
7       return model
```

Learning phase

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

Model: "sequential_33"

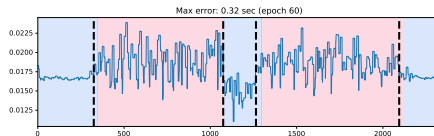
Layer (type)	Output Shape	Param #
separable_conv1d_77 (Separab	(None, None, 5)	140
max_pooling1d_73 (MaxPooling	(None, None, 5)	0
separable_conv1d_78 (Separab	(None, None, 1)	166
max_pooling1d_74 (MaxPooling	(None, None, 1)	0
Total params: 306		
Trainable params: 306		
Non-trainable params: 0		

Prediction phase

```
[267]: 1 true_bkps = y[0].numpy().tolist()
2       transformed = model(X)[0].numpy()
3       prediction = rpt.Binseg(jump=5).fit(transformed).predict(4)
4       error = hausdorff(true_bkps, prediction)/100
5       # print results
6       print(f"True change points:\t\t{true_bkps}")
7       print(f"Predicted change points:\t{prediction}")
8       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.

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]: 1 def get_model():
2     model = keras.Sequential([
3         keras.layers.SeparableConv1D(...),
4         keras.layers.MaxPool1D(...),
5         keras.layers.SeparableConv1D(...),
6         keras.layers.MaxPool1D(...),])
7     return model
```

Learning phase

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

Model: "sequential_33"

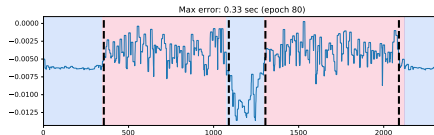
Layer (type)	Output Shape	Param #
=====		
separable_conv1d_77 (Separab	(None, None, 5)	140
max_pooling1d_73 (MaxPooling	(None, None, 5)	0
separable_conv1d_78 (Separab	(None, None, 1)	166
max_pooling1d_74 (MaxPooling	(None, None, 1)	0
=====		
Total params: 306		
Trainable params: 306		
Non-trainable params: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

Prediction phase

```
[267]: 1 true_bkps = y[0].numpy().tolist()
2 transformed = model(X)[0].numpy()
3 prediction = rpt.Binseg(jump=5).fit(transformed).predict(4)
4 error = hausdorff(true_bkps, prediction)/100
5 # print results
6 print(f"True change points:\t\t{true_bkps}")
7 print(f"Predicted change points:\t{prediction}")
8 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]: 1 def get_model():
2       model = keras.Sequential([
3           keras.layers.SeparableConv1D(...),
4           keras.layers.MaxPool1D(...),
5           keras.layers.SeparableConv1D(...),
6           keras.layers.MaxPool1D(...),])
7       return model
```

Learning phase

```
[262]: 1 X, y = get_Xy(8)
2       model = get_model()
3       model.compile(loss=loss_func,
4                     optimizer=keras.optimizers.Adam(1e-2))
5       history = model.fit(X, y, epochs=101, verbose=0,
6                           callbacks=CustomCallback())
7       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]: 1 true_bkps = y[0].numpy().tolist()
2       transformed = model(X)[0].numpy()
3       prediction = rpt.Binseg(jump=5).fit(transformed).predict(4)
4       error = hausdorff(true_bkps, prediction)/100
5       # print results
6       print(f"True change points:\t\t{true_bkps}")
7       print(f"Predicted change points:\t{prediction}")
8       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]: 1 def get_model():
2     model = keras.Sequential([
3         keras.layers.SeparableConv1D(...),
4         keras.layers.MaxPool1D(...),
5         keras.layers.SeparableConv1D(...),
6         keras.layers.MaxPool1D(...),])
7     return model
```

Learning phase

```
[262]: 1 X, y = get_Xy(8)
2 model = get_model()
3 model.compile(loss=loss_func,
4               optimizer=keras.optimizers.Adam(1e-2))
5 history = model.fit(X, y, epochs=101, verbose=0,
6                   callbacks=CustomCallback())
7 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]: 1 true_bkps = y[0].numpy().tolist()
2 transformed = model(X)[0].numpy()
3 prediction = rpt.Binseg(jump=5).fit(transformed).predict(4)
4 error = hausdorff(true_bkps, prediction)/100
5 # print results
6 print(f"True change points:\t\t{true_bkps}")
7 print(f"Predicted change points:\t{prediction}")
8 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.