1 Tunka-Rex Virtual Observatory

1.1 ICRC2021

V. Lenok, D. Kostunin, O. Kopylova, P. Bezyazeekov, D. Wochele, F. Polgart, S. Golovachev, V. Sotnikov, E.Sotnikova for the Tunka-Rex collaboration

1.1.1 Introduction

- The radio detection techniques is successfully applied to ultra-high energy particle physics since first decade of this century.
- A number of detectors are in sucessfully operation, next-generation prototypes are on the construction stage
- **Huge amount of data acquired by radio detectors\ aimed at air-shower experiments are left unused**
- ⇒ We combine the experience gained by the radio astronomers and astroparticle physicists to publish the radio datain multi-purpose *Virtual Observatory*

1.1.2 The science cases and aims of the observatory

All-component values of a radio field recorded with high-resolution timing bring us to the following science cases: - Studies of the radio background in the frequency band of 30-80˜MHz - Searching for radio transients - Training of neural networks for background tagging - Outreach and education

Tunka-Rex instrument is an ideal testbench to try this approach → *Tunka-Rex Virtual Observatory (TRVO)*

1.1.3 Tunka-Rex timeline

1.1.4 Application of TRVO

- Storage of data itself
  - Tunka-Rex data (measurements and simulations)
  - Almarac data (in progress)
- Studies of the radio background
  - RFI library module
  - Module for 21cm cosmology tasks
- Outreach and education
  - The first worksop of Mathematical center in Akademgorodok

1.1.5 Scope of this notebook

We give an example of application of TRVO described in PoS(ICRC2021)421 with combination of autoencoder developed for Tunka-Rex: PoS(ICRC2021)223

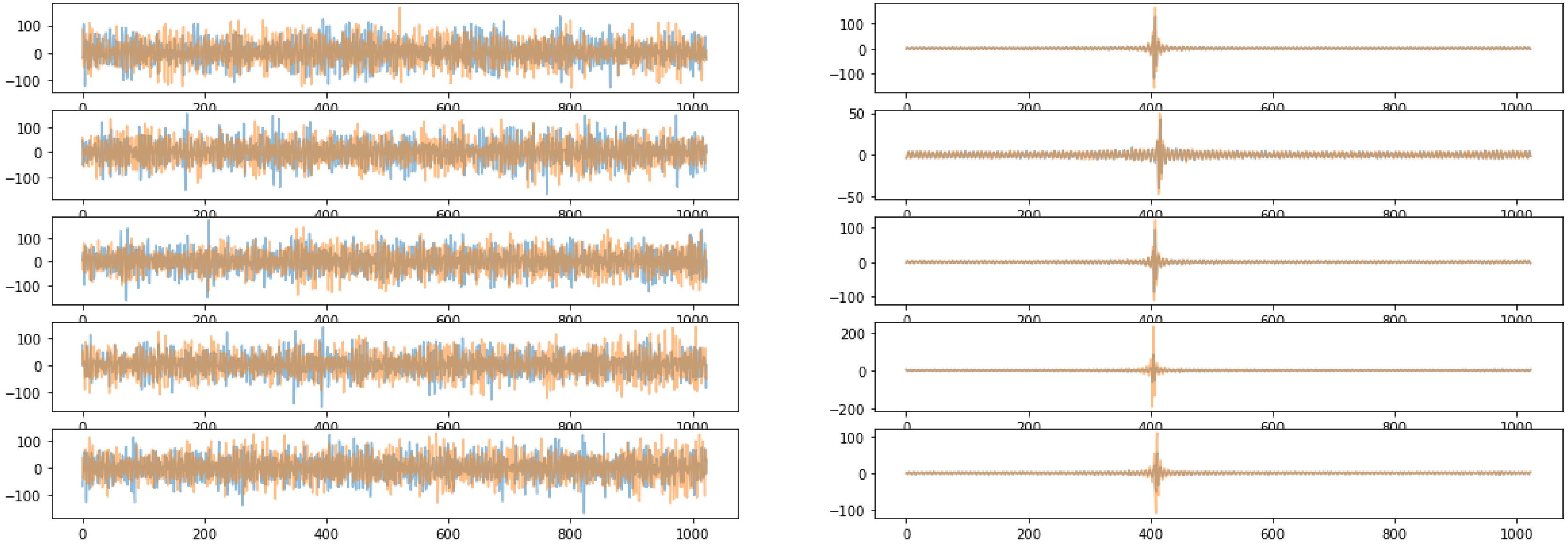
We use PostreSQL database which contains events described in `trvo.core.Schema` Here one can see the example of data selection using SQLAlchemy framework:

[1]:
```
from trvo.core.Schema import Trace
from trvo.icrc import db, db_sim

import matplotlib.pyplot as plt
import numpy as np

# configuring query (just taking 5 events)
nevents = 5

# example of filtering and limiting query (with real data)
Q = db.query(Trace).filter(Trace.station == 5).limit(nevents)

# example of quering simulation data
Q_sim = db_sim.query(Trace).limit(nevents)
```

We have selected five traces for real measurements and simulations. Since the rate of high-energy events is very low, most of the real data contains just radio background, which can be used as background for the testing of autoencoder

[2]:
```
# plotting two channels
plt.clf()
fig, axs = plt.subplots(5, 2)
for ev_id in range(nevents) :
    for ch in (0,1) :
        axs[ev_id,0].plot(Q.all()[ev_id].traces[ch], label = "Ch %s" % ch, alpha = 0.5)
        axs[ev_id,1].plot(Q_sim.all()[ev_id].traces[ch], label = "Ch %s" % ch, alpha = 0.5)
plt.gcf().set_size_inches(20, 7)
plt.show()
```

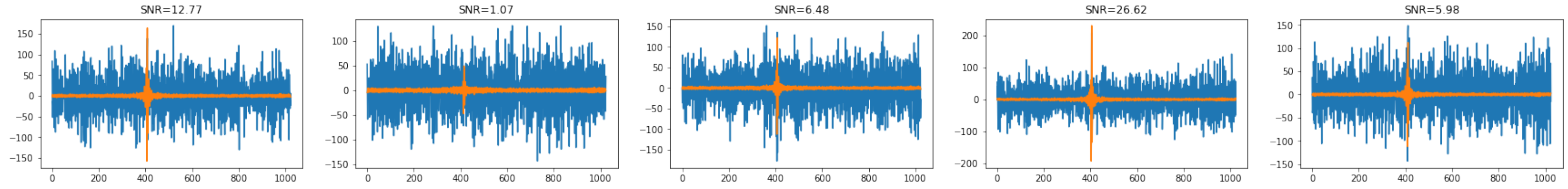
```
<Figure size 432x288 with 0 Axes>
```



Now we mix simulated signals with real background and estimate signal to noise ratio (SNR) as SNR $= S^2/N^2$

[3]:
```
ch = 1
plt.clf()
fig, axs = plt.subplots(1, 5)
plt.gcf().set_size_inches(30, 3)
for ev_id in range(nevents) :
    signal = Q_sim.all()[ev_id].traces[ch]
    noise = Q.all()[ev_id].traces[ch]
    sum = signal+noise
    snr = np.max(np.abs(signal))**2/np.std(noise)**2
    axs[ev_id].plot(sum)
    axs[ev_id].plot(signal)
    axs[ev_id].set_title("SNR=" + str(round(snr,2)))
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In this part we prepare out data in format acceptible by autoencoder: performing upsampling with factor $\times 16$ and select 4096 counts from the upsampled trace. The resulting traces are then normalized to have amplitudes in range of $[0; 1]$:

[4]:
```
import pytrex.units as u
from pytrex.signaltools import Trace as T
import numpy as np

# we need this for the normalization of data for autoencoder
def normalize(array):
    result = []
    coeff = []
    for signal in array:
        c = np.max(signal)-np.min(signal)
        signal = signal/c + 0.5
        result.append(signal)
        coeff.append(c)
    return np.array(result), np.array(coeff)

ch = 1

signal_arr = []
sum_arr = []
snr_arr = []

low_cut = 4000
```

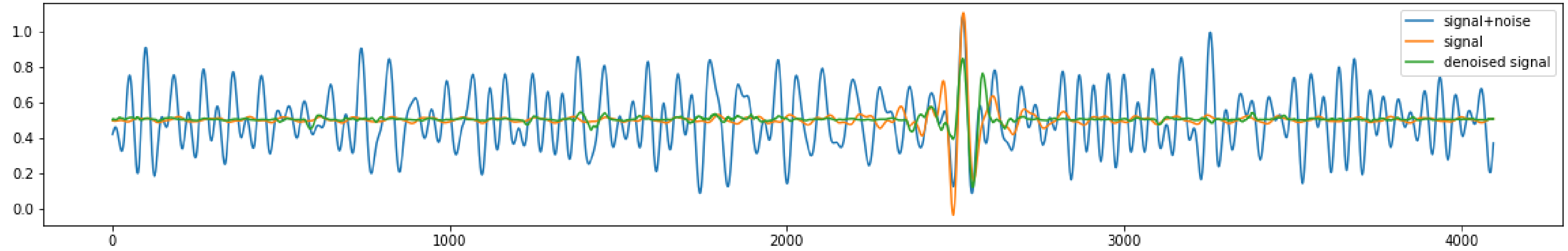
```
# preparing data for autoencoder
for ev_id in range(nevents) :
    signal = Q_sim.all()[ev_id].traces[ch]
    noise = Q.all()[ev_id].traces[ch]
    sum = signal+noise
    snr_arr.append(np.max(signal)**2 / np.std(noise)**2)
    # we need to resample them for neural network
    signal_proc = T(values = signal, timestamps = np.arange(0,5120*u.ns,5*u.ns)).resample(16384)
    sum_proc = T(values = sum, timestamps = np.arange(0,5120*u.ns,5*u.ns)).resample(16384)
    signal_arr.append(signal_proc.values[low_cut:low_cut+4096])
    sum_arr.append(sum_proc.values[low_cut:low_cut+4096])
sum_arr, coeff = normalize(np.array(sum_arr))
signal_arr = np.array(signal_arr)
signal_arr[:,] = signal_arr/coeff[:,None] + 0.5
```

Now we denoise our traces:

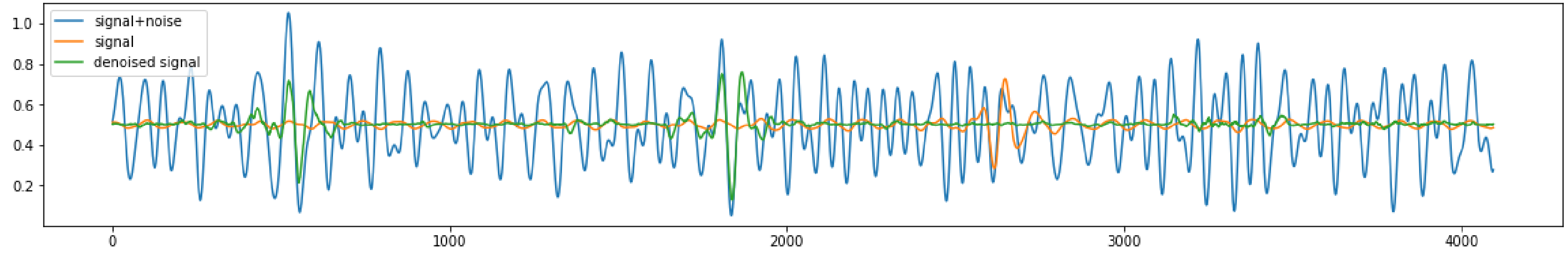
[5]:
```
from keras.models import load_model
autoencoder = load_model("/home/jovyan/work/trvo_icrc/trvo-master/autoencoder.h5")
shape_0 = sum_arr.shape[0]
shape_1 = sum_arr.shape[1]
predict = autoencoder.predict(np.reshape(sum_arr, (shape_0, shape_1, 1)))
predict = np.asarray(predict)
```

[6]:
```
for i,r in enumerate(sum_arr):
    print(str(i+1) + ") SNR = " + str(snr_arr[i]))
    plt.clf()
    plt.gcf().set_size_inches(20, 3)
    plt.plot(sum_arr[i], label = 'signal+noise')
    plt.plot(signal_arr[i], label = 'signal')
    plt.plot(predict[i,:,0], label = 'denoised signal')
    plt.legend()
    plt.show()
```

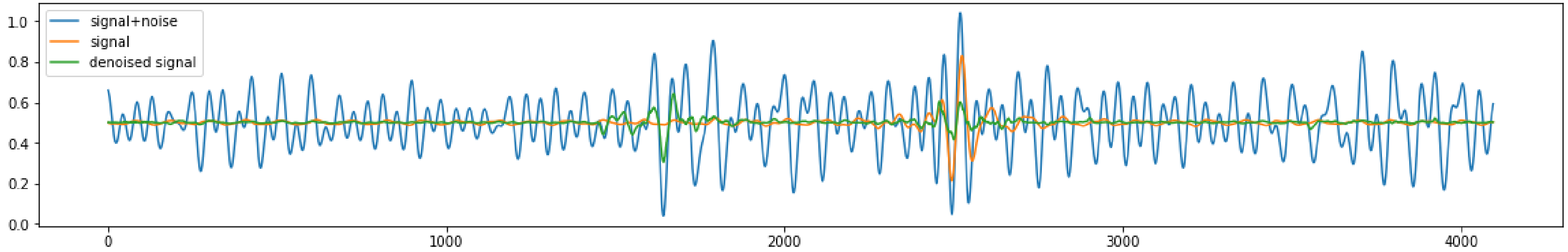
1) SNR = 12.769458133938706



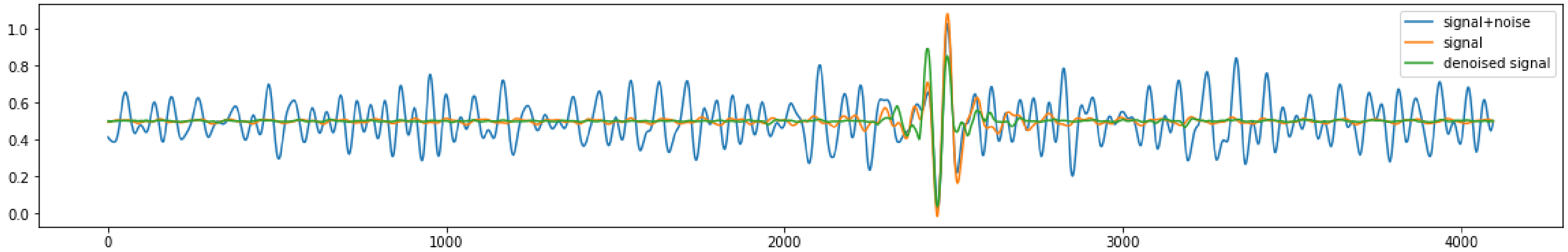
2) SNR = 1.0659592672345821



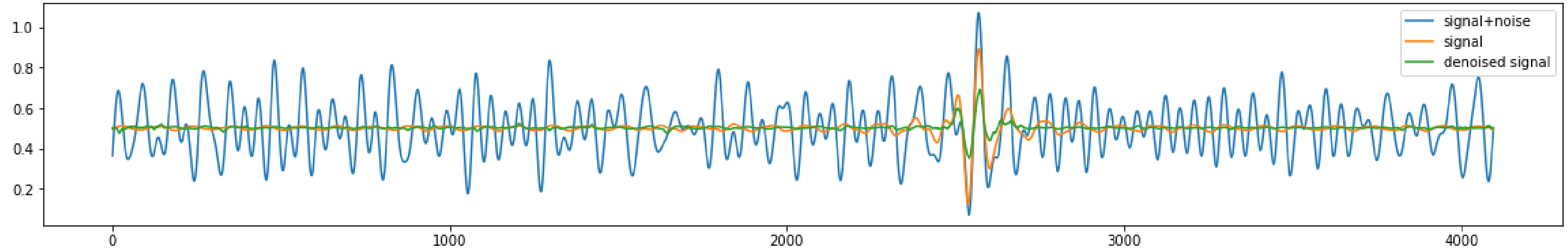
3) SNR = 6.476745516046857



4) SNR = 26.615271645859554



5) SNR = 5.935937596070467



1.1.6 Obtained results

For the standard analysis we use a cut of SNR < 12 and one can see that autoencoder successfully recovers events with high levels of SNR (trace 1 and 4). For the rest traces with low SNR autoencoder either is not able to reconstruct signal at all (trace 2 with SNR ≈ 1) or adds false positives detections (trace 3) as well as failed with amplitude reconstruction (trace 5).

1.1.7 Summary

This short notebook gives an example of application of Tunka-Rex Virtual Observatory, particularly how to use its data wiht autoencoder. One can play with other events and different data analysis ideas. The source code of TRVO is published under free license: https://gitlab.ikp.kit.edu/tunkarex/trvo