GSoC Final Report

Google Summer of Code

Anomaly detection and recovery with a Deep-learning-based approach

(Project associated with GNSS-SDR)

Contributor: Seonwoo Kim / Mentor: Luis Esteve Elfau

Introduction:

GNSS systems and software are directly exposed to various threats. Especially in this project, I focused on detecting spoofing attacks using Deep-learning based approach. Since the GNSS-SDR receiver records various property logs in terms of the received signal, we can get more information over the naive PVT data. Based on the instruction from my mentor, I went through related materials and now working on the TEXBAT dataset. The dataset was designed for the general use of developing methods of detecting or mitigating spoofing attacks. According to the knowledge I got from the related papers, I focused on two signal processing blocks(tracking, and observables). And I found out that the C/N0 value constantly shows an outstanding deviation when spoofing happens. I implement more research on this variable and tried a naive Autoencoder model(with LSTM) on this data. Additionally, I tried a statistical method using the Bollinger band to detect the attack moment.

Objectives:

Develop a spoofing-detection model which can be applied to GNSS-SDR

Key Skills:

GNSS, Network-Spoofing, Pandas, Numpy, Geojson, Keras, Tensorflow, C++, Python

Deliveries(link):

- Extract from the bin file : https://drive.google.com/drive/folders/13-jZOEQsWxXss04g2HMTHZNyb0Px-pMa?usp=sharing
- Graphs of extracted observables : https://drive.google.com/drive/folders/1Q5OBJr1BatXJFdhs3lIylh445pgUkmyo?usp=sharing
- Github repository for the code : https://github.com/seki5405/gnss_spoof_detector.git

TImeline:

5/21/22	The initial connection with my mentor
5/25/22	Instruction to be familiar with GNSS-SDR, and implement My-first-position-fix
~6/02/22	Done with My-first-position-fix, cover GNSS-SDR docs & ASEN5090
~6/08/22	Tried to receive real-time GNSS signal via USRP B200 and HackRF
6/13/22	Instruction to go through project-related materials
~6/18/22	Done with reading 4 papers(2 Anti-spoofing papers, 2 Deep Learning based papers)
6/22/22	Instruction to go through TEXBAT papers(3)
~6/23/22	Did research on Anomaly detection models(SVM, Autoencoder)
~7/01/22	Download the TEXBAT dataset, and tried to run it via GNSS-SDR
~7/06/22	Investigate observables from different modules(Acquisition, Tracking, Observable)
~7/12/22	Investigate more on narrow-downed several variables
~7/17/22	Organize the picked variables for further use
~7/19/22	Tried to apply the Autoencoder model based on LSTM
~7/22/22	Tried statistical method(Bollinger band) to use the boundary based detection
~7/31/22	Solving the issues regarding the data alignment (Changed frequency)
~8/10/22	Select variables from the module and feed them into autoencoder model
~8/15/22	Modify monitoring-client to make a redirection to python file(detector)
~8/20/22	Develop collector.py and collect observables from monitoring-client
~8/24/22	Build Linear autoencoder model and train (train.ipynb)
~8/28/22	Build detector.py which lively inspect transmitted data from the monitoring block
~8/31/22	Finalizing the codes and set the repository
~9/01/22	Fill out the README.md
~9/05/22	Write the final report

Description of Work:

Dataset Analysis

The TEXBAT dataset was presented in 2012 for simulating GNSS receiver under spoofing attack. There are six high-fidelity digital recordings of live static and dynamic GPS L1 C/A spoofing tests conducted by the Radionavigation Laboratory of the University of Texas at Austin. Each scenario is revealing obvious anomalies that future GPS receivers could be designed to detect.

The dataset is composed of 8 different scenarios.

Scenario Designation	Spoofing Type	Platform Mobility	Power Adv. (dB)	Frequency Lock	Noise Padding	Size (GB)
1: Static Switch	N/A	Static	N/A	Unlocked	Enabled	43
2: Static Overpowered Time Push	Time	Static	10	Unlocked	Disabled	42.5
3: Static Matched-Power Time Push	Time	Static	1.3	Locked	Disabled	42.6
4: Static Matched-Power Pos. Push	Position	Static	0.4	Locked	Disabled	42.6
5: Dynamic Overpowered Time Push	Time	Dynamic	9.9	Unlocked	Disabled	38.9
6: Dynamic Matched-Power Pos. Push	Position	Dynamic	0.8	Locked	Disabled	38.9

Scenario description (ds1 to ds6)

Additional 2 data based on different scenarios were also released after 4 years.

- The 7th scenario is based on the cleanStati.bin dataset. It's power-matched time push scenario much like ds3.bin but more subtle because it employs carrier phase alignment between the spoofing and authentic signals.
- The 8th scenario is identical to the ds7.bin scenario but it also includes unpredictable low-rate security code. This is called Zero-delay security code estimation and replay attack.

I explored the dataset and extract observations as below;

- Extracted observables & PVT(LINK)

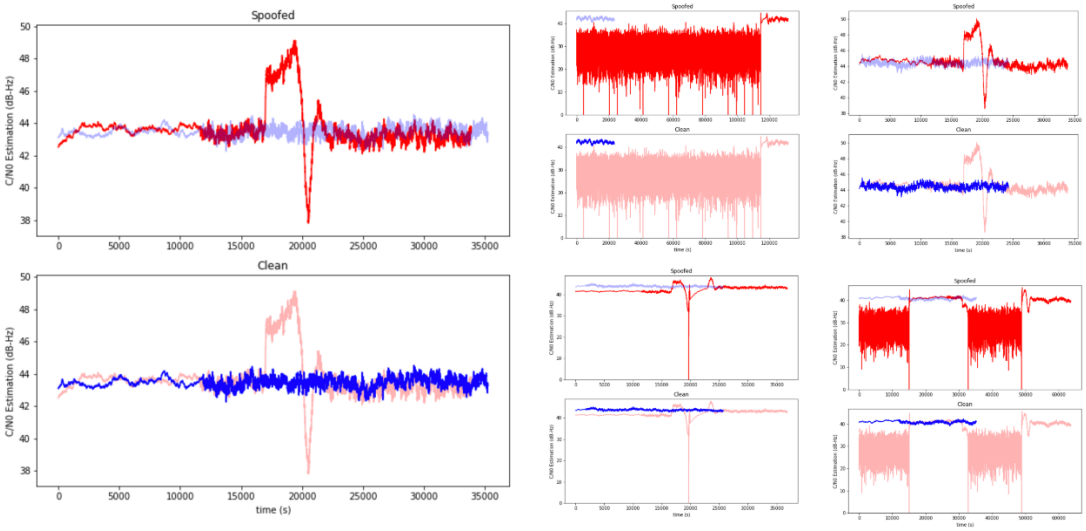
  Each signal processing block has different variables and it's possible to extract the variables by setting 'dump' option in `conf` file. I extracted variables from Acquisition, Tracking, Observable, PVT modules. After that I implemented EDA to find useful variables. For the analysis, I put the spoofing data's plot with clean data's plot to compare them visually.

  [파일 목록 캡쳐 화면!!!]

- Graph from the extracted observables (LINK)

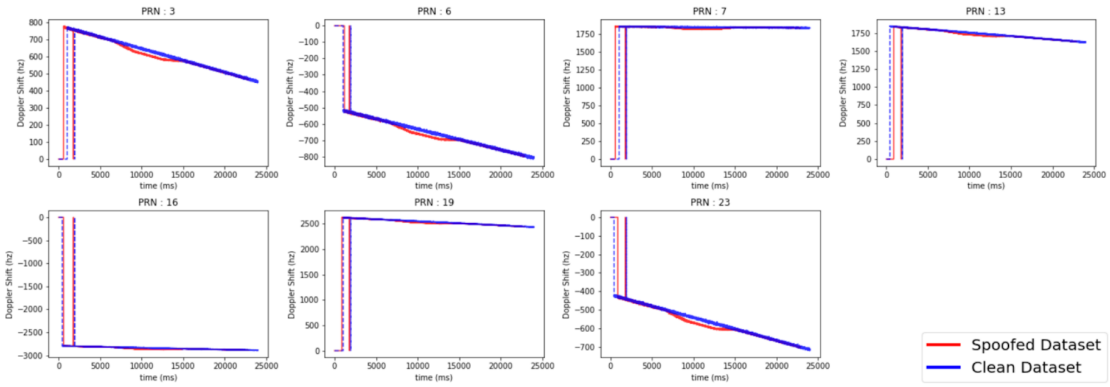
  From EDA, I found some notable observables; C/N0, Doppler Shift, Pseudorange, Prompt_i, Prompt_q.

  - cn0_db_hz : C/N0 estimation, in dB-Hz.

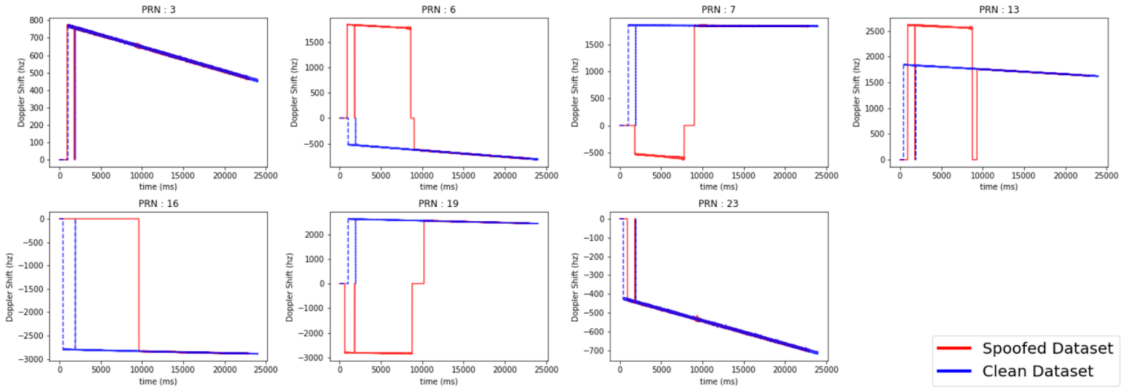


  C/N0 is the one of the most frequently used variables to find anomaly in GNSS. So I investigated from the beginning. As I expected, C/N0 showed notable difference between spoofed and clean data. In normal status, C/N0 tends to be stable while it fluctuates unexpectedly in case of spoofing attack.

  - carrier_doppler_hz : Doppler shift, in Hz.



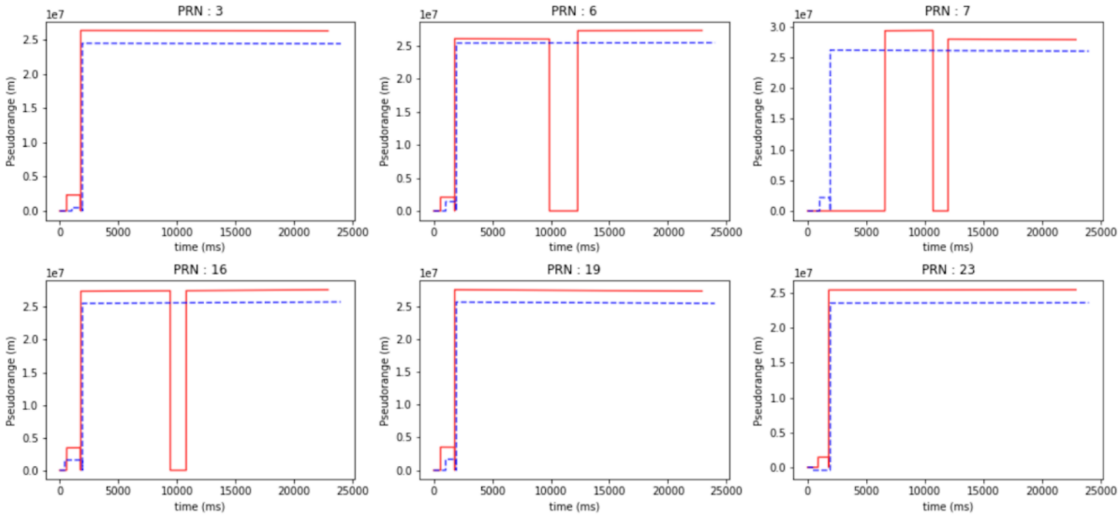
Case 1 : Slight fluctuation during linear decrease/increase



Case 2 : Abrupt extreme jump/drop

Carrier_doppler_hz(Doppler Shift) is the value which typically shows gradual increase/decrease in stationary status since the only satellite move makes gradual change. It also cannot show abrupt jump or drop in case of movement. However, I observed abrupt jump/drop from the spoofed signal(case 2). Also slight fluctuation was also observed from the stationary status.

- Pseudorange : Difference between the time of reception and the time of transmissinon.



Since the Pseudorange is calculated as Doppler Shift multiplied by the negative of carrier wavelength, it’s supposed to be static. However, the abrupt jump/drop was observed in spoofed scenarios.

Code

Code developed during the summer is separated from gnss-sdr repository. The repository is composed of three things(folders):

- ipynb files: Includes all analyzed and approaches (LINK)
- monitoring-client: Modified based on the existing monitoring-client
  - Changed processing variables(2→6)

```
// Print table header.
attron(A_REVERSE);
printw("%3s%6s%10s%17s%14s%14s\n", "CH", "PRN", "Prompt_i", "Prompt_q", "CN0 [dB-Hz]", "Doppler [Hz]");
attroff(A_REVERSE);
```

  - Implement additional socket communication to detector module

```
#include <arpa/inet.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#define PORT 5736


int main(int argc, char* argv[])
{

    printf("Now connecting Spoofing detector...\n");
    int sock = 0, valread, client_fd;
    struct sockaddr_in serv_addr;
    char buffer[1024] = { 0 };
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("\n Socket creation error \n");
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        printf("\n Invalid address/ Address not supported \n");
        return -1;
    }

    if ((client_fd = connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr))) < 0) {
        printf("\n Connection failed. Check the Spoofing detector \n");
        return -1;
    }
```

  - Data extraction&transmission from the monitoring-client to python file
    - Save data for each channel to `line` and combine it to `log`

```
bool Gnss_Synchro_Udp_Source::print_table(std::string& log)
{
    log = "";
    if (read_gnss_synchro(stocks))
        {
            populate_channels(stocks);

            clear();  // Clear the screen.

            // Print table header.
            attron(A_REVERSE);
            printw("%3s%6s%10s%17s%14s%14s\n", "CH", "PRN", "Prompt_i", "Prompt_q", "CN0 [dB-Hz]", "Doppler [Hz]");
            attroff(A_REVERSE);

            for(auto const& ch: channels)
            {
                int channel_id = ch.first;
                gnss_sdr::GnssSynchro data = ch.second;

                printw("%3d%6d%10d%17f%14f%14f\n", channel_id, data.prn(), (int)data.prompt_i(), data.prompt_q(), data.cn0_db_hz(), data.carrier_doppler_hz());
                std::string line;
                line.resize(71);
                sprintf((char*)line.c_str(), "%3d,%6d,%10d,%17f,%14f,%14f, ", channel_id, data.prn(), (int)data.prompt_i(), data.prompt_q(), data.cn0_db_hz(), c
                log += line;
            }

            refresh();  // Update the screen.
```

- Transmit `log` to detector(collector) module

```
            std::string log;
            log.resize(569);

            while (true)
                {
                    udp_source.print_table(log);
                    log += "\n";
                    send(sock, log.c_str(), 569, 0);
                }
```

- spoof_detector
  - `collector.py` : Save the streamed data for the future utilization

```
import socket
import re
import sys
import os

HOST = '127.0.0.1'
PORT = 5736

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind((HOST, PORT))
server_socket.listen()
client_socket, addr = server_socket.accept()
print('Connected by ', addr)

f = open(os.path.join('data', sys.argv[1]), "w")
```

```
i = 1
while True:
    data = client_socket.recv(800)

    if not data:
        break

    line = data.decode()
    line = re.sub(r'[^\x00-\x7F]', '', line)
    print('H',line, 'F')

    i += 1

    if i <= 8:
        continue
    f.write(line)

f.close()
client_socket.close()
server_socket.close()
```

- Autoencoder(`train.ipynb`)

```
def AutoEncoder_model():
  inputs = Input(shape=(4,))
  encoder1 = Dense(3, activation='tanh')(inputs)
  encoder2 = Dense(2, activation='tanh')(encoder1)
  decoder1 = Dense(3, activation='tanh')(encoder2)
  decoder2 = Dense(4, activation='tanh')(decoder1)
  outputs = Dense(4, activation='softmax')(decoder2)
  model = Model(inputs, outputs)
  model.compile(optimizer='adam',loss='mean_squared_error')
  return model

autoencoder = AutoEncoder_model()
```

```
autoencoder.fit(X_train, X_train, epochs = 100, batch_size=128,
validation_data=(X_train, X_train))
```

- `detector.py` : Load the trained model and determine the occurance of spoofing

```
import socket
import numpy as np
from tensorflow import keras

HOST = '127.0.0.1'
PORT = 5736
THRESHOLD = 80

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind((HOST, PORT))
server_socket.listen()
client_socket, addr = server_socket.accept()
print('Connected by ', addr)
```

```
X_1 = np.zeros(6)

autoencoder = keras.models.load_model('autoencoder_model')
timestep = 1
prev_data = {
        1 : np.zeros([1,4]),
        2 : np.zeros([1,4]),
        3 : np.zeros([1,4]),
        4 : np.zeros([1,4]),
        5 : np.zeros([1,4]),
        6 : np.zeros([1,4]),
        7 : np.zeros([1,4]),
        8 : np.zeros([1,4])
    }

while True:
    data = client_socket.recv(569)

    print('.', end=' ')
    timestep += 1

    if not data:
        break

    try:
        data = list(map(float, data.decode().split(', ')[:-1]))
    except:
        continue

    data = {i+1:np.expand_dims(data[6*i+2:6*(i+1)], axis=0) for i in range(8)}

    new_data = {k:data[k]-prev_data[k] for k in data}

    prev_data = data

    prds = {i+1:autoencoder.predict(new_data[i+1],verbose=0) for i in range(8)}

    errors = {i+1:np.mean(np.power(new_data[i+1]-prds[i+1], 2), axis=1) for i in range(8)}

    for i in range(8):
        if errors[i+1] > THRESHOLD:
            print()
            print(f"[Time:{timestep}] Suspicious Signal on Channel[{i+1}] (Autoencoder Error : {errors[i+1]})")

client_socket.close()
server_socket.close()
```

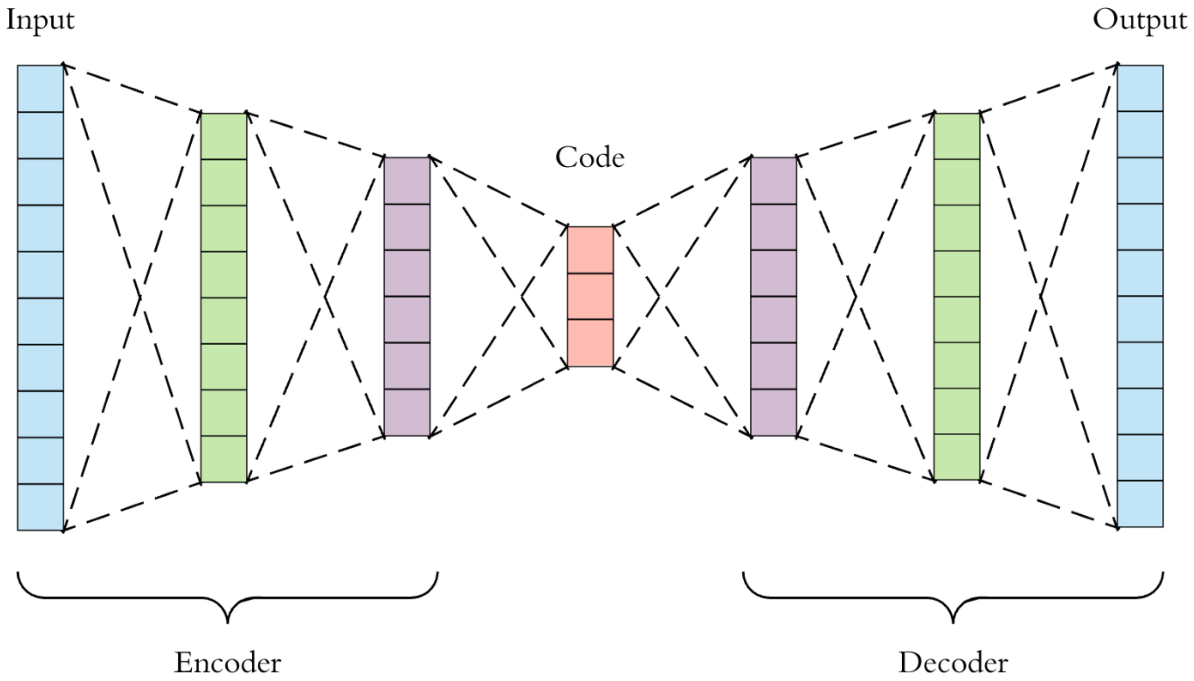
Variables

I chose 4 variables from tracking module based on the observation I found during EDA. The variables show visible difference in their pattern and the amplitude.

- prompt_i : Value of the Prompt correlator in the In-phase component.
- prompt_q : Value of the Prompt correlator in the Quadrature component.
- cn0_db_hz : C/N0 estimation, in dB-Hz.
- carrier_doppler_hz : Doppler shift, in Hz.

Autoencoder

Initially developed LSTM based autoencoder based on the thought that the streamed consecutive data will be well processed with RNN type models. But changed to use Linear autoencoder at last because LSTM based model requires a lot of calculation which is not proper to process streamed data. Also, the Linear model also worked well on detecting spoofing attacks.



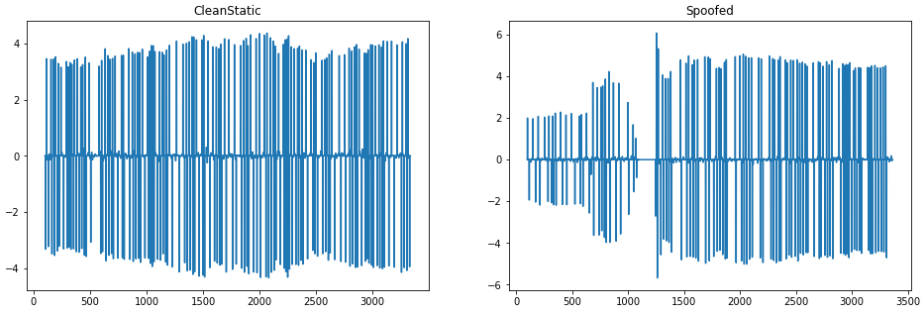
Linear Autoencoder model structure

Before feeding the data, I preprocessed data by normalizing and getting `diff()` (Getting offset from the previous timestep).

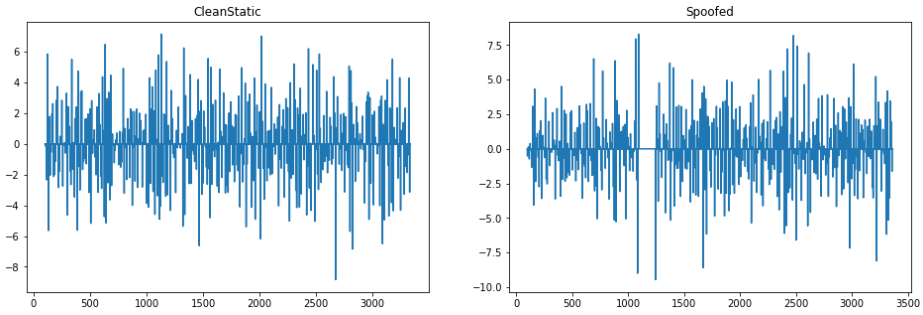
```
for dic in dicts:
  for prn in list(dic.keys())[:-1]:
    df = dic[prn]
    scaler = StandardScaler()
    scaler.fit(df)
    df_scaled = scaler.transform(df)
    dic[prn] = pd.DataFrame(data=df_scaled, index=df.index, columns=df.columns)
```

Normalization using StandardScaler()

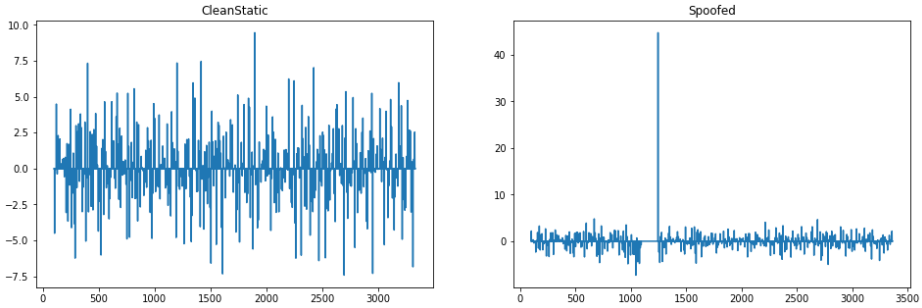
And here’re the preprocessed data with comparison with CleanStatic data.



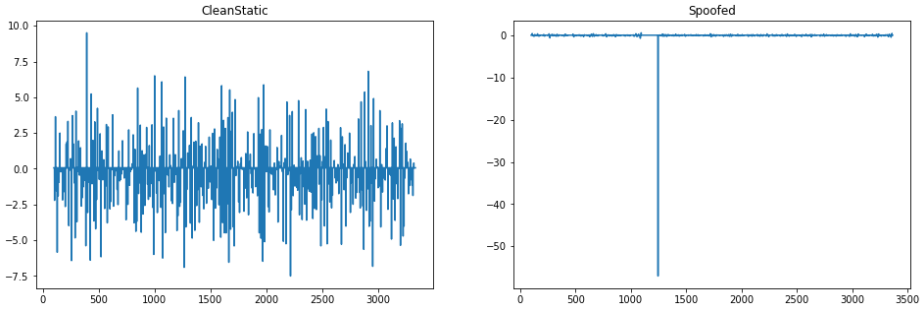
Prompt_i



Prompt_q



cn0_db_hz



carrier_doppler_hz

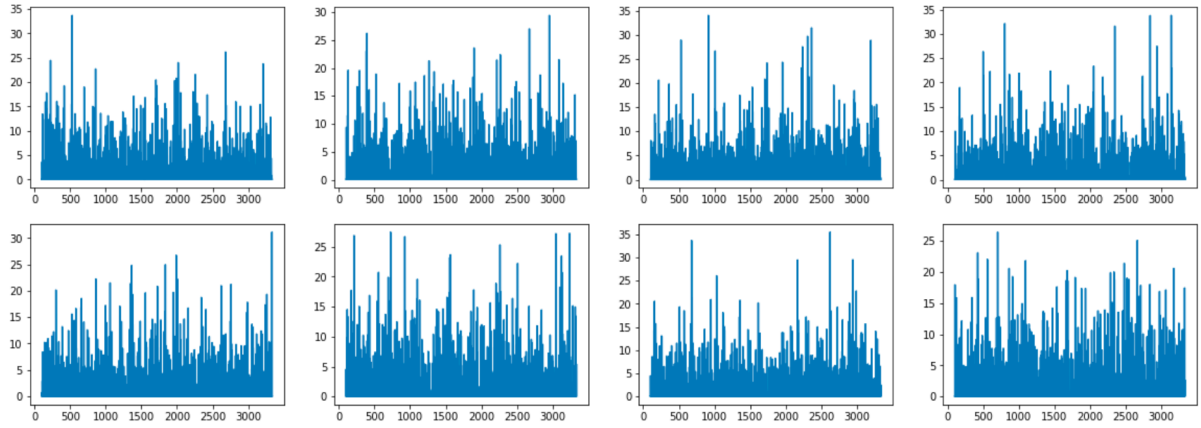
As we can see, there's a starting of attack at around 1200th timestep. And we can observe some weird patterns.

I trained the autoencoder model with `cleanStatic` scenario which is normal. The autoencoder model is trained to generate the output of which label is its input. So the model is trained to generate the input for the normal data, which shows higher Mean Squared Error for the abnormal(Spoofed) data.

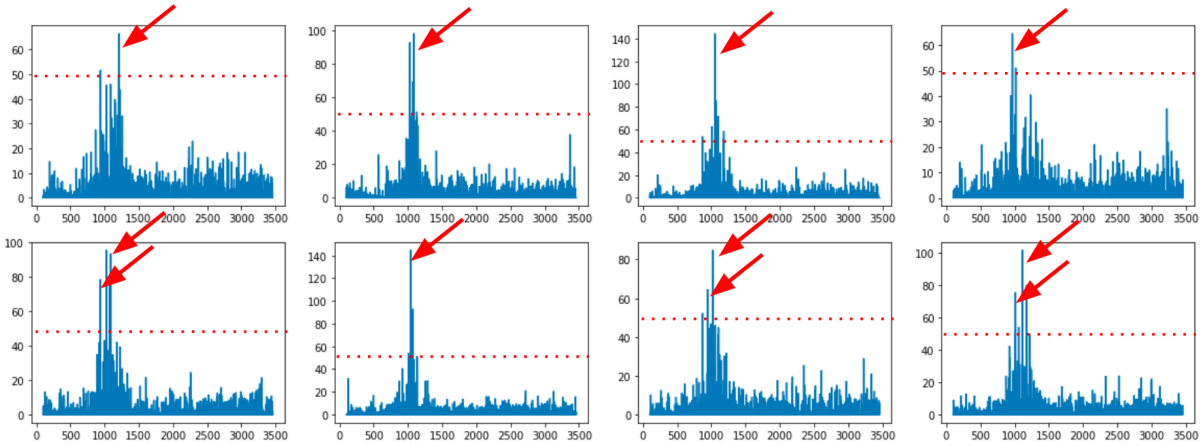
$$MSE = \frac{1}{n}\Sigma\underbrace{\left(y - \hat{y}\right)^2}_{\text{The square of the difference between actual and predicted}}$$

Mean Squared Error equation

Typically, the model shows more than 50 of MSE(Mean Squared Error) for the Spoofed timestep.



MSE graph for cleanStatic dataset, from this, I set the threshold to 50.



MSE graph of 8 channels from the 3rd scenario. The model triggers alerts if the error goes over 50.

Communication between monitoring client - detector

Since the model was developed with Python, I implemented socket communication between the monitoring-client and the detector. On top of the original visualization of monitoring-client, I added construction of log data which consists of data from 8 channels which is implemented in `gnss_synchro_udp_source.cc`. Plus, `main.cc` connects to `detector.py` (`collector.py`) after setting up the socket configuration. And repeatedly transmit the log.

Detector

The detector is designed to work in real time. It receives streamed data from the monitoring-client and parse it into different channel data. The parsed data is preprocessed and fed to the pre-trained model. Based on the gap between the prediction and the input, it can get MSE(Mean Squared Error) for each channel. If the MSE value goes over the designated threshold(50 in the experiment), it will trigger alarm message.

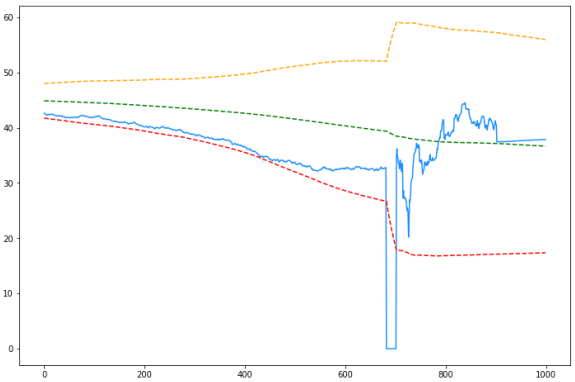
```
[Time:3133] Suspicious Signal on Channel[4] (Autoencoder Error : [2.72639292e+11])
. . . . . .
 . . . . . .
. . . . . .
[Time:3324] Suspicious Signal on Channel[1] (Autoencoder Error : [2.10932371e+12])
[Time:3324] Suspicious Signal on Channel[2] (Autoencoder Error : [5.51157818e+11])
[Time:3324] Suspicious Signal on Channel[3] (Autoencoder Error : [2.36333906e+12])
[Time:3324] Suspicious Signal on Channel[4] (Autoencoder Error : [3.16050932e+12])
```

Triggered alert messages (detector.py)

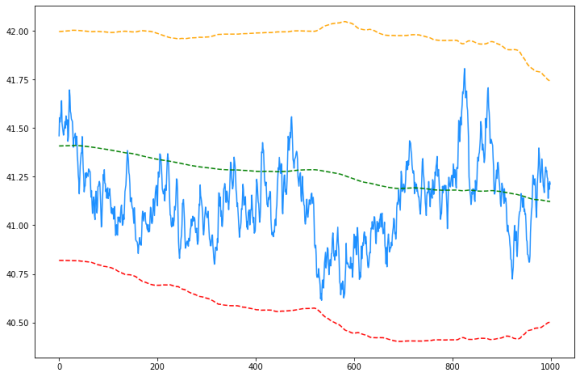
Subsidiary: Bollingerband

While considering the streamed data processing, came up with an idea of using statistical method called Bollingerband. It consists of Moving average line and it's upper band and lower band which maintain equal offset from the moving average line. It doesn't require any training of Deep Neural network.

Used `cn0_db_hz` to check its performance, and it worked well but parameter optimization seems to be necessary.



Spoofed signal



Normal signal

How to run the code:

- Installation(monitoring-client)
  - You may need to rebuild the monitoring-client.
  - The basic instructions are same with the origin. Here's the link of the tutorial.
  - Copy `gnss_synchro_udp.source.cc` and `main.cc` from our repository.

    (It contains socket communication and log processing)
- Execution order
  - The monitoring-client is supposed to connect with `detector.py`.
  - So it's mandatory to run `detector.py` first and monitoring-client after that.
  - Lastly, if you run `gnss-sdr`, it will start streaming.

```
# needs to run the commands in the working directory
> python3 detector.py
> ./monitoring-client 1234
> gnss-sdr --config_file=./sample.conf
```

Results:

- Measurement method

  To evaluate the model's performance, I feed the spoofed signal data by running gnss-sdr with monitor on. While running gnss-sdr, detector receives the data and split the data based on channels. The splitted data goes through the autoencoder model, and the model triggers an alert if the error between the output and input is over 50 (This was the threshold I set based on the experiemental result). I calculate the accuracy of channels by channel by checking if the alert is triggered.
- Measurement result

Scenario	#1	#2	#3	#4	#7	#8
Alert Triggered	7/8	8/8	6/8	5/8	0/8	8/8
Accuracy(%)	87.5	100	75	62.5	0	100

Considering that the triggered alert in any channel notify users to check if there is an spoofing attack, the model works in all scenarios except the 7th scenario.

The scenario tries to match the power-level with the clean input which may work on C/N0. From that, I figure out that C/N0 is the key factor to detect spoofing attack.

Conclusion:

The autoencoder model looked reliable to detect spoofing attack. It doesn't detect clearly for the seventh scenario but worked well with other scenarios. It means that the model generally works well to raise an alert for spoofing attack. However, due to the processing speed of `detector.py`, it doesn't work robustly in real time. It seems that multi-threading or extra sampling might help solving this.

- Future Work:
  - Prove the concept in the real world environment : It may require additional setup to spoofing signal leakage while receiving the real-time signal.
  - Using more variables : The monitor module can process only limited amount of variables. Because of that, I couldn't help excluding some variables I chose during EDA. If adding more variables by editing the `monitoring_client`, it may enhance the model's performance.
  - Multiprocessing : The `detector` prototype is working with a single thread and there seems to some amount of delay in visualizing. Multiprocessing with separated visual expression may helps.

Bibliography:

1. T. E. Humphreys et al., "The Texas Spoofing Test Battery: Toward a Standard for Evaluating GPS Signal Authentication Techniques," 2012.
2. T. Humphreys et al., "TEXBAT DATA SETS 7 AND 8.", utexas.
3. A. Lemmenes, et al., "Detailed Analysis of the TEXBAT Datasets Using a High Fidelity Software GPS Receiver," *Proceedings of the 29th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2016)*, Nov. 2016.
4. Harshad Sathaye, "Anti-Spoofing Techniques for GNSS Receivers", 2021, https://harshadsathaye.com/gsoc21/#ftnt2
5. A. Ranganathan et al., "SPREE: a spoofing resistant GPS receiver", Mobicom 16, Oct. 2016.
6. S. Semanjski et al., "Use of Supervised Machine Learning for GNSS Signal Spoofing Detection with Validation on Real-World Meaconing and Spoofing Data—Part I," *Sensors*, vol. 20, no. 4, p. 1171, Feb. 2020.
7. S. Semanjski et al., "Use of Supervised Machine Learning for GNSS Signal Spoofing Detection with Validation on Real-World Meaconing and Spoofing Data—Part II," *Sensors*, vol. 20, no. 7, p. 1806, Mar. 2020.
8. A. Shafique et al., "Detecting Signal Spoofing Attack in UAVs Using Machine Learning Models," *IEEE Access*, vol. 9, pp. 93803–93815, 2021.
9. Z. Chen, C. K. Yeo, B. S. Lee and C. T. Lau, "Autoencoder-based network anomaly detection," *2018 Wireless Telecommunications Symposium (WTS)*, 2018, pp. 1-5.