Time Series Anomaly Detection

LSTM vs GAN

SALHI Lamia
ELKHEIR Yassine
LABARRE Vincent

What is the problem ?

- Proliferation of temporal observation data
- Detect anomalies becomes essential
  - A time point or period where a system behaves unusually
- The data used here to train/test models

Presentation Plan :

1. Anomaly Detection using LSTM Autoencoder
    a. LSTM autoencoder
    b. Data Pre-Processing
    c. Model Construction
    d. Visualisation and Performance
2. Anomaly Detection using GAN
    a. GAN Model
        i. Signal Generation
        ii. Error Calculation
    b. Data Pre-Processing
    c. Visualisation and Performance
3. Comparison LSTM/GAN
    a. Metrics comparison
    b. interpretation

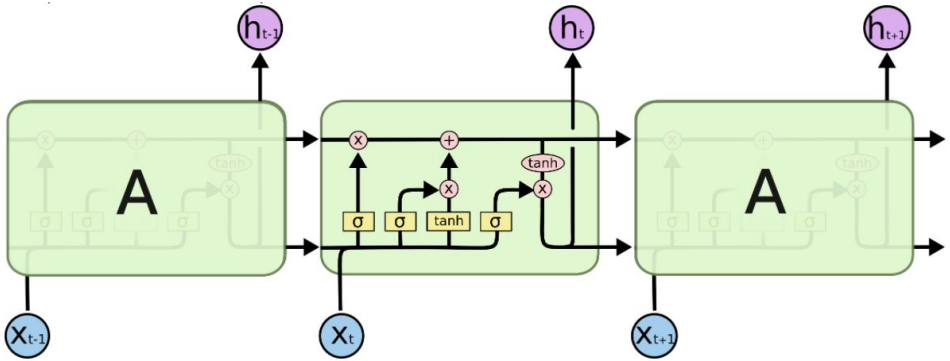
1. Anomaly Detection using LSTM :

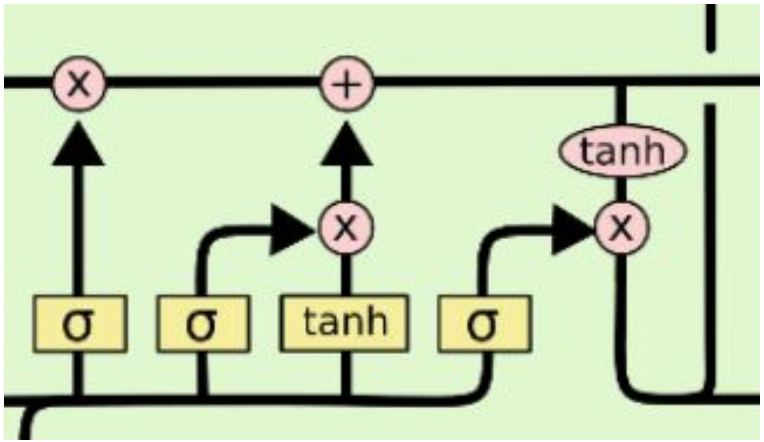
1.a LSTM Autoencoder

What is LSTM ?

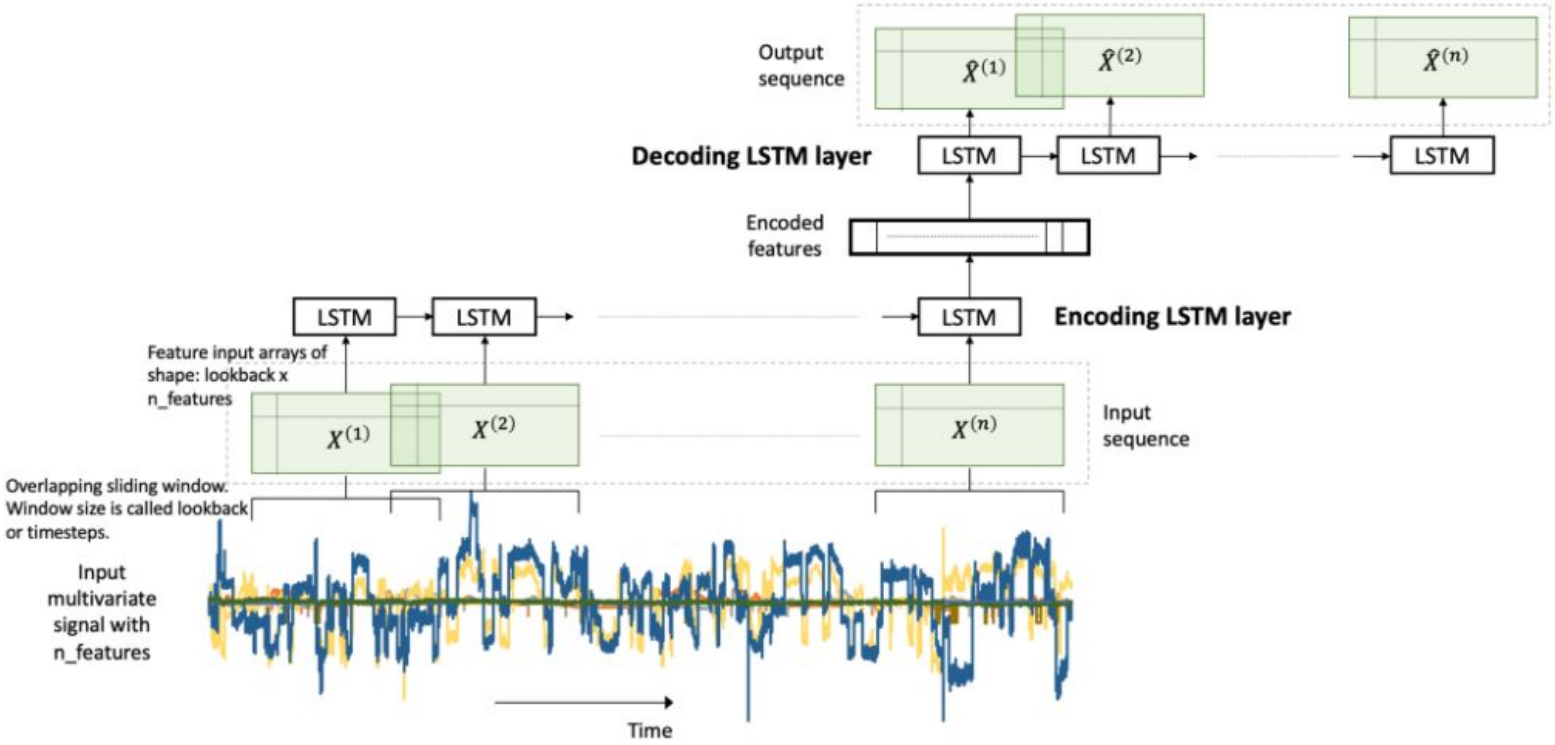
1. LSTM is a type of Recurrent Neural Network (RNN). RNNs, in general, and LSTM, specifically, are used on sequential or time series data.

2. LSTM are known for its ability to extract both long- and short- term effects of pasts event





1.a LSTM Autoencoder



1.b Data Pre-Processing :

The input data to an LSTM model is a 3-dimensional array. The shape of the array is *samples* x *lookback* x *features*. Let's understand them,

- ***Samples*:** This is simply the number of observations, or in other words, the number of data points.
- ***Lookback*:** LSTM models are meant to look at the past. Meaning, at time *t* the LSTM will process data up to (*t-lookback*) to make a prediction.
- ***Features*:** It is the number of features present in the input data.

1.c Model Construction :

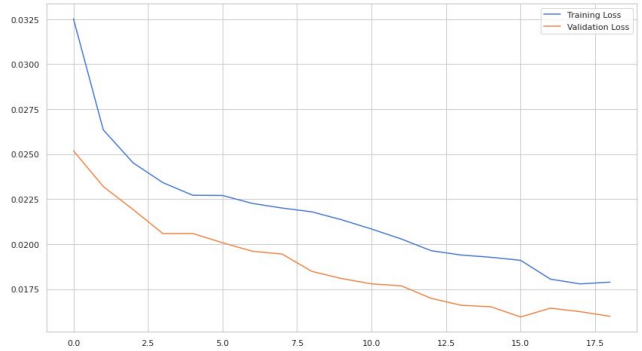
- **Encoder of One layer (128), Activation : Relu By Default**
- **Decoder of One Layer (128), Activation : Relu By Default**
- **Adding DropOut with probability of 0.2**
- **We use RepeatVector to repeat our vector output returned by last layer in encoder LSTM. This vector is repeated TIMESTEPS time since the 1st layer in the decoder - decoder_lstm requires a 3-D input compressed sequence.**

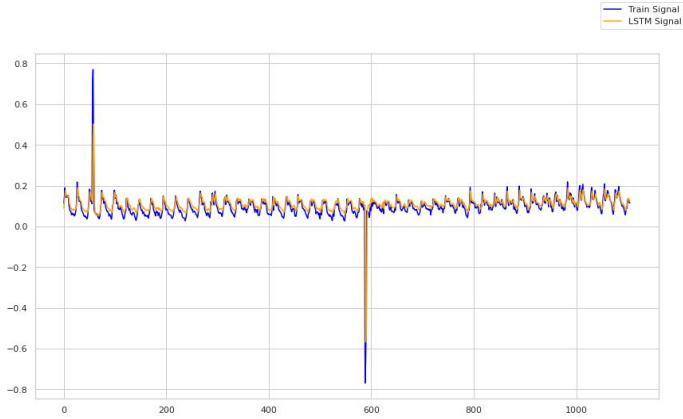
CODE :

```
model = Sequential([
    LSTM(128, input_shape=(timesteps, num_features)),
    Dropout(0.2),
    RepeatVector(timesteps),
    LSTM(128, return_sequences=True),
    Dropout(0.2),
    TimeDistributed(Dense(num_features))
])
```

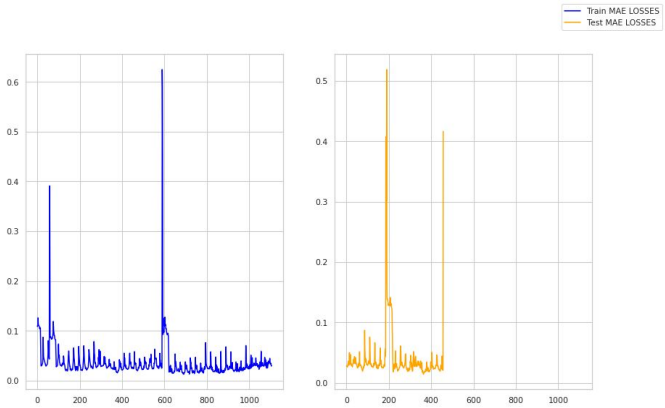
1.d Model evaluation :

Loss Functions MAE- Training And Validation





$$\left\| \hat{X} - X \right\|$$



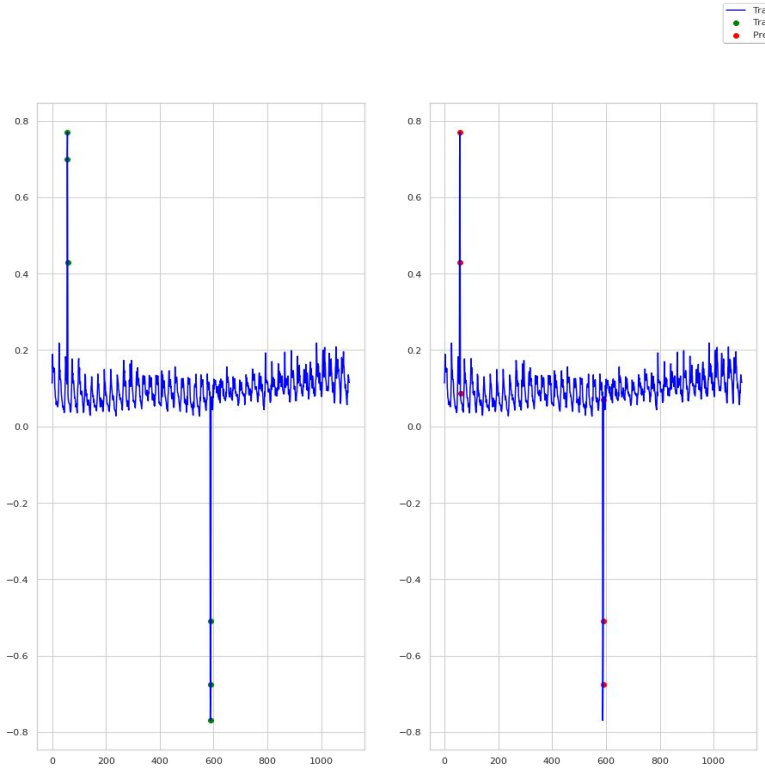
1.d Model evaluation :

how do we select Anomalies ?

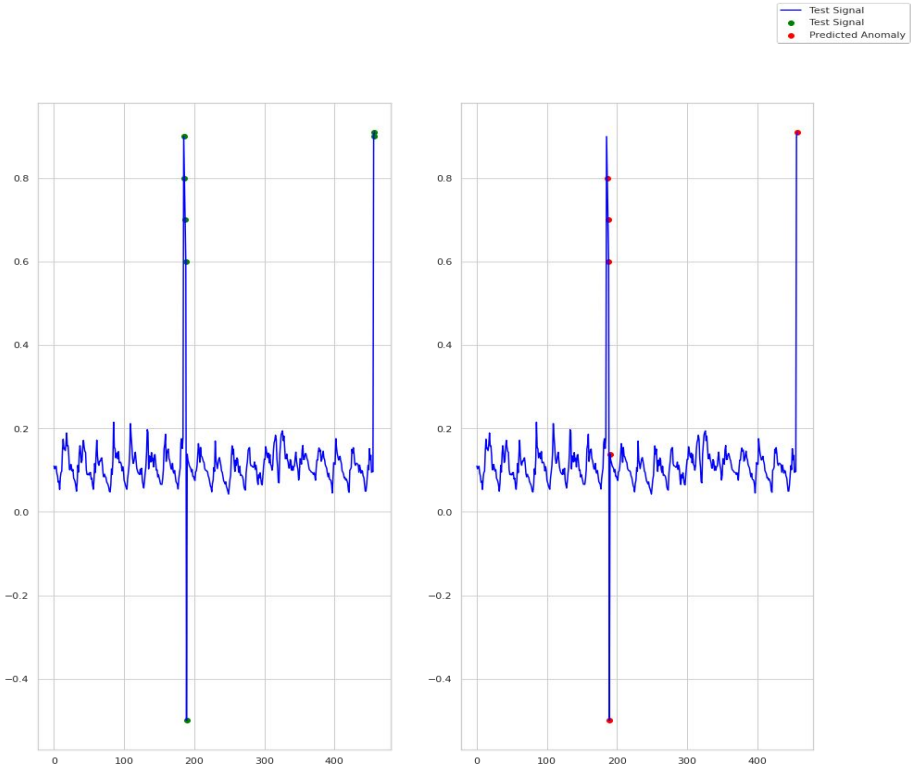
By Setting A Threshold

In our Case, We have set a threshold of 0.2

1.d Model evaluation :



Accuracy on Train : 99.70%



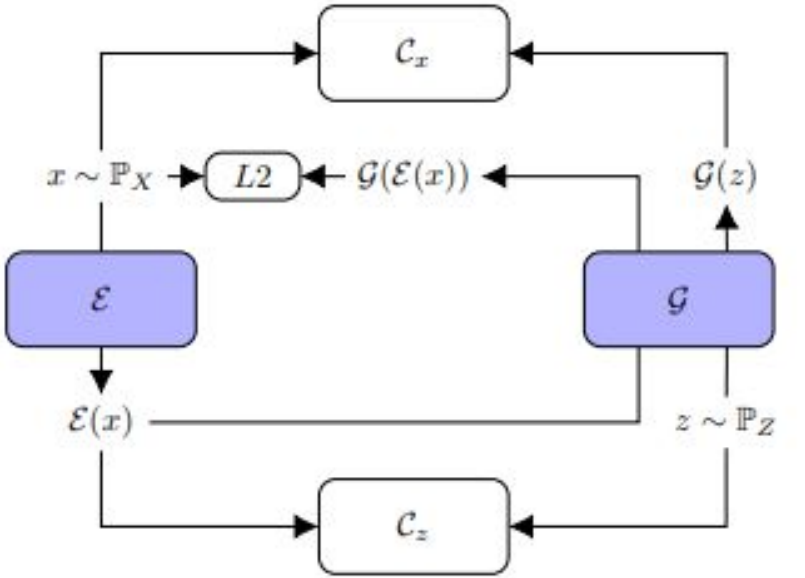
Accuracy on Test : 99.78%

2. Anomaly Detection using GAN :

2.a GAN Model :

- **Generator E** : Encoder that maps the time series sequences X into the latent space Z.
- **Generator G** : Decoder that transforms the latent space Z into the reconstructed time series G(Z) = $\hat{X}$.
- **Critic Cx** : Distinguish between real time series sequences from X and the generated time series sequences from G(z),
- **Critic Cz** : Measure the goodness of the mapping into the latent space by discriminating between Z and E(X).

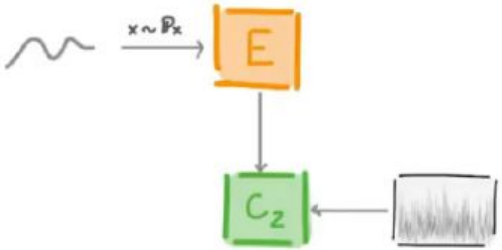
Model architecture of the Time anomaly detection GAN



2.a GAN Model : Error Evaluation (Wasserstein loss)

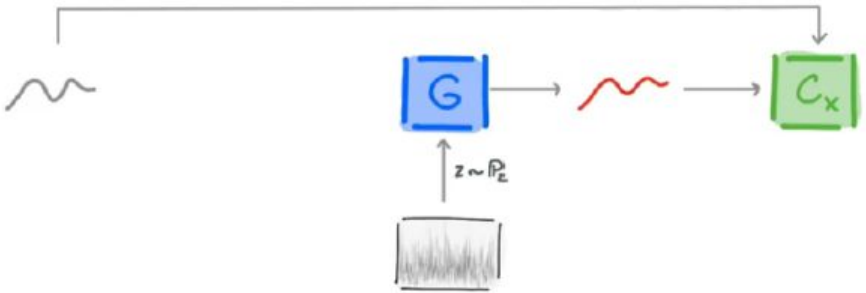
X → Z and its Critic Cz $V_Z(\mathcal{C}_z, \mathcal{E}) = \mathbb{E}_{z\sim\mathbb{P}_Z}[\mathcal{C}_z(z)] - \mathbb{E}_{x\sim\mathbb{P}_X}[\mathcal{C}_z(\mathcal{E}(x))]$

$$\min_{\mathcal{E}} \max_{\mathcal{C}_z \in \mathbf{C_z}} V_Z(\mathcal{C}_z, \mathcal{E})$$



Z → X and its Critic Cx $V_X(\mathcal{C}_x, \mathcal{G}) = \mathbb{E}_{x\sim\mathbb{P}_X}[\mathcal{C}_x(x)] - \mathbb{E}_{z\sim\mathbb{P}_Z}[\mathcal{C}_x(\mathcal{G}(z))]$

$$\min_{\mathcal{G}} \max_{\mathcal{C}_x \in \mathbf{C_x}} V_X(\mathcal{C}_x, \mathcal{G})$$



2.a GAN Model : Error Evaluation (Wasserstein loss)

Training the GAN with Wasserstein losses alone cannot ensure mapping x_i to a desired z_i that will be further mapped back to $\hat{x}_i$

$$x_i \rightarrow \mathcal{E}(x_i) \rightarrow \mathcal{G}(\mathcal{E}(x_i)) \approx \hat{x}_i$$

So we introduce Cycle Consistency Loss:

$$V_{L2}(\mathcal{E}, \mathcal{G}) = \mathbb{E}_{x \sim \mathbb{P}_X}[\|x - \mathcal{G}(\mathcal{E}(x))\|_2]$$

Epoch: 84/200, [**Dx loss:** [-0.1695949 -1.576806 1.3809068 0.00263042]] [**Dz loss:** [-2.445694 -1.9200627 -1.2011967 0.06755652]] [**G loss:** [-0.06447932 -1.4134738 1.2885365 0.00604579]]

Epoch: 85/200, [**Dx loss:** [-0.12871414 -1.7395961 1.5831734 0.00277086]] [**Dz loss:** [-2.3284328 -1.8004932 -1.0813236 0.05533842]] [**G loss:** [-0.4702531 -1.61338 1.0809712 0.00621555]]

2.b Data Pre-Processing :

- Unsupervised anomaly detection: simpy signal X (with timestamp) as input data.

- Need to do a few preprocessing steps :

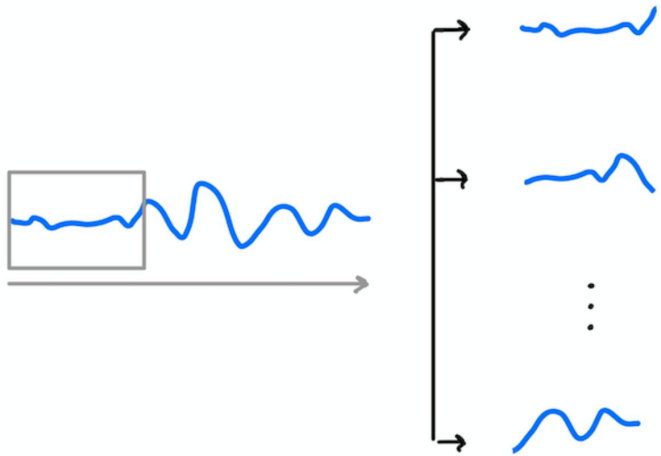
- **Data frequency :** adjust signal spacing to be of equal width across all times.
  There are two important parameters in this process:
  - **interval**: an integer that refers to the time span to compute aggregation of.
  - **method**: what aggregation method should be used to compute the value.
    By default set to the mean.
- **Data imputation :** Fill in missing values (by the mean value).
- **Data normalization:** Normalize the data between [-1, 1].
- **Slice the data to rolling window:** (see next slides)

exchange-2_cpc_results

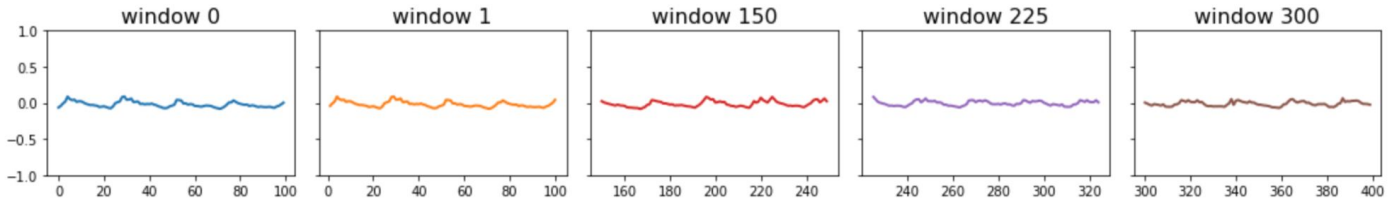
timestamp	value
1404165600	0.081964736
1404167400	0.098972236
1404169200	0.065313949
1404171000	0.070662834
1404172800	0.102490196
1404174600	0.123394649
1404176400	0.153045113
1404178200	0.148321513
1404180000	0.218257757
1404181800	0.226597938
1404183600	0.174045802
1404185400	0.159901961
1404187200	0.145626478
1404189000	0.143828571
1404190800	0.9
1404192600	0.97

2.b Data Pre-Processing : Slice the data to a rolling window

- Transform data into a sequence of “understable” data for the model.
- Creating samples from the signal (snapshots).
- The sliding window is defined by :
  - Its width
  - The step size
- Divide the signal into segments defined by the sliding window.

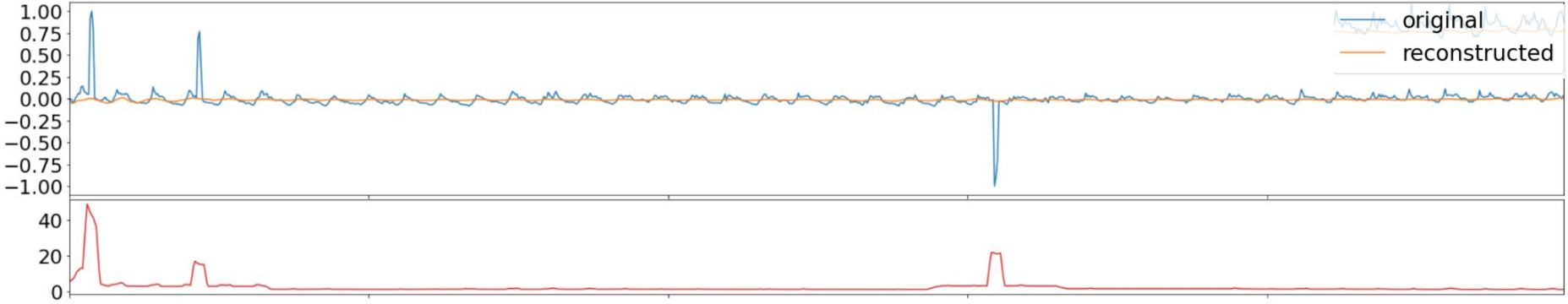


```
plot_rws(X_train)
```

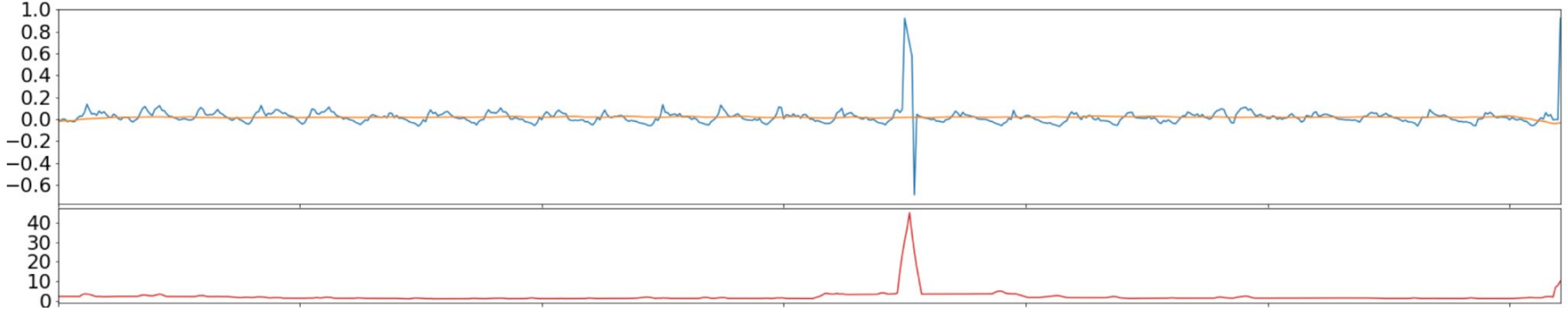


2.c Error Evaluation : Visualization

Original and reconstructed signal and the error curb on **training data**

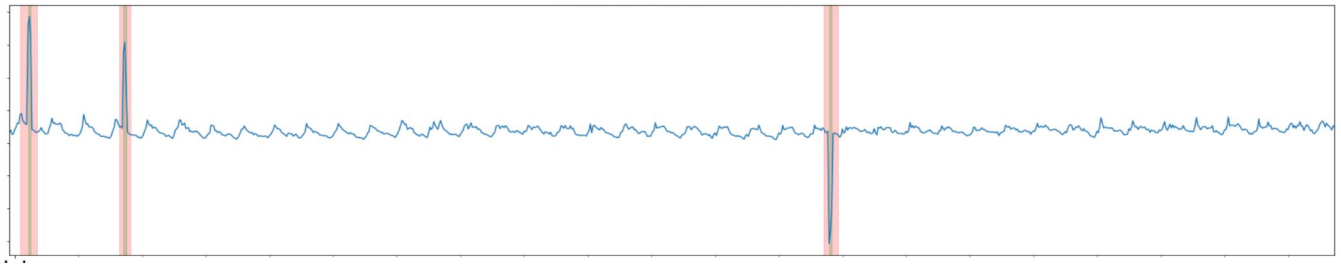


Original and reconstructed signal and the error curb on **testing data**

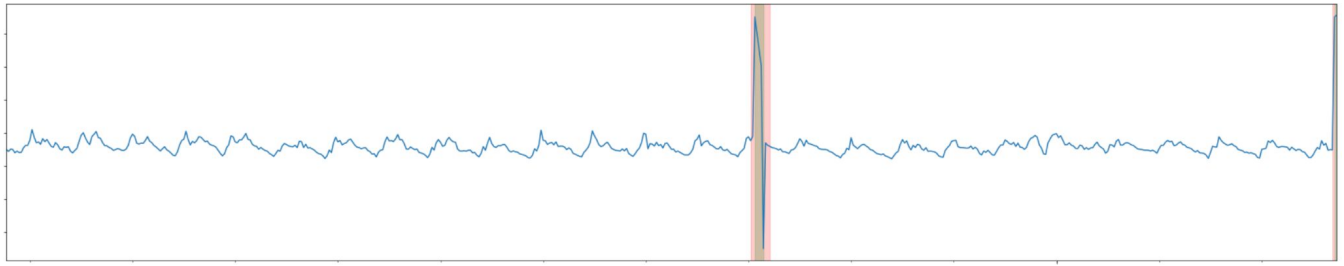


2.c Error Evaluation : Metrics

training



testing



model	accuracy (train/test)	f1-score (train/test)	recall (train/test)	precision (train/test)
tadgan	0.967/0.974	1.000 /1.000	0.967/0.974	1.000/1.000

3. Comparison LSTM and TadGAN

3. Results of the different metrics: accuracy, f1 score, recall and precision

Chosen metrics :

- The accuracy

$$Acc = \frac{sum(y_{hat} == y_{groundtruth})}{number\ of\ data}$$

- Sensitivity/recall : how good our model is at detecting the positives/anomalies. A test can cheat and maximize this by always returning “positive”.

$$S_e = \frac{TP}{TP + FN}$$

- Precision – how many of the positively classified were relevant. A test can cheat and maximize this by only returning positive on one result it’s most confident in.

$$P_e = \frac{TP}{TP + FP}$$

- The F1 score :

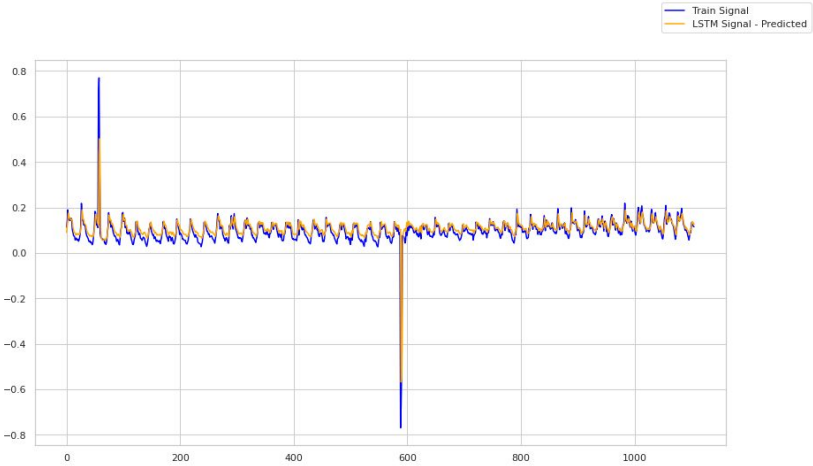
$$F_1 = 2.\frac{P_e.S_e}{P_e + S_e}$$

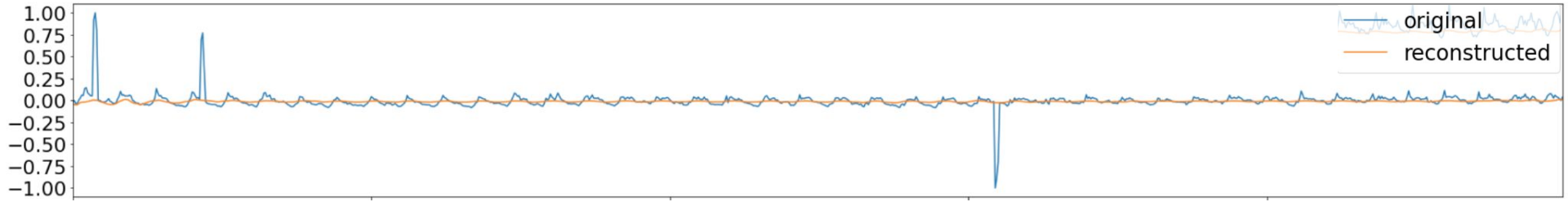
3. Results of the different metrics:
accuracy, f1 score, recall and precision

model	accuracy (train/test)	f1-score (train/test)	recall (train/test)	precision (train/test)
tadgan	0.967/0.974	1.000 /1.000	0.967/0.974	1.000/1.000
LSTM	0.997/0.998	0.798/0.907	0.830/1.000	0.770/0.830

3. Comparison of the models

Signal and predicted signal of the LSTM model and the TadGan





Interpretation

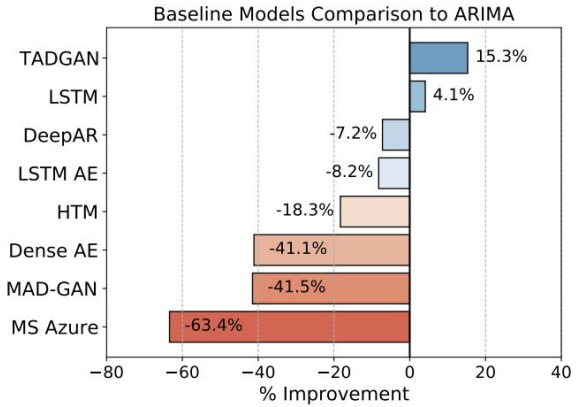


Fig. 3. Comparing average F1-Scores of baseline models across all datasets to ARIMA. The x-axis represents the percentage of improvement over the ARIMA score by each one of the baseline models.

How well do AutoEncoders perform? To view the superiority of GAN, we compare it to other reconstruction-based method such as LSTM AE, and Dense AE. One striking result is that the autoencoder alone does not perform well on point anomalies. We observe this as LSTM, AE, and Dense AE obtained an average F1 Score on A3 and A4 of 0.205 and 0.082 respectively, while TadGAN and MAD-GAN achieved a higher score of 0.643 and 0.527 respectively. One potential reason could be that AutoEncoders are optimizing L2 function and strictly attempt to fit the data, resulting in that anomalies get fitted as well. However, adversarial learning does not have this type of issue.

TadGAN v.s. MadGAN. Overall, TadGAN (0.7) outperformed Mad-GAN (0.219) significantly. This fully demonstrates the usage of forward cycle-consistency loss (Eq. 5) which prevents the contradiction between two *Generators* $\mathcal{E}$ and $\mathcal{G}$ and paves the most direct way to the optimal z_i that corresponds to the testing sample x_i. Mad-GAN uses

Alexander Geiger and Dongyu Liu and Sarah Alnegheimish and Alfredo Cuesta-Infante and Kalyan Veeramachaneni, "TadGAN: Time Series Anomaly Detection Using Generative Adversarial Networks" , 2020

Thanks for your attention

Area

This method captures the general shape of the orignal and reconstructed signal and then compares them together.

$$s_t = \frac{1}{2 * l} \left| \int_{t-l}^{t+l} x^t - \hat{x}^t \, dx \right|$$

Point

This method applies a point-to-point comparison between the original and reconstructed signal. It is considered a strict approach that does not allow for many mistakes.

$$s_t = \left| x^t - \hat{x}^t \right|$$

DTW

A more lenient method yet very effective is Dynamic Time Warping (DTW). It compares two signals together using any pair-wise distance measure but it allows for one signal to be lagging behind another.

$$s_t = W^* = \text{DTW}(X, \hat{X}) = \min_W \left[\frac{1}{K} \sqrt{\sum_{k=1}^{K} w_k} \right]$$