2021.08.26

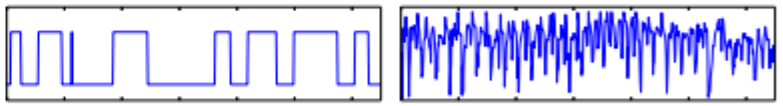
LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection

LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection

Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, Gautam Shroff
{MALHOTRA.PANKAJ, ANUSHA.RAMAKRISHNAN, GAURANGI.ANAND, LOVEKESH.VIG, PUNEET.A, GAUTAM.SHROFF}@TCS.COM
TCS Research, New Delhi, India

Abstract

Mechanical devices such as engines, vehicles, aircrafts, etc., are typically instrumented with numerous sensors to capture the behavior and health of the machine. However, there are often external factors or variables which are not captured by sensors leading to time-series which are inherently unpredictable. For instance, manual controls and/or unmonitored environmental



(a) Predictable (b) Unpredictable

Figure 1. Readings for a manual control sensor.

example, a laden machine behaves differently from an unladen machine. Further, the relevant information pertaining

1 / 논문 리뷰

Abstract

Mechanical devices such as engines, vehicles, aircrafts, etc., are typically instrumented with numerous sensors to capture the behavior and health of the machine. However, there are often external factors or variables which are not captured by sensors leading to time-series which are inherently unpredictable. For instance, manual controls and/or unmonitored environmental conditions or load may lead to inherently unpredictable time-series. Detecting anomalies in such scenarios becomes challenging using standard approaches based on mathematical models that rely on stationarity, or prediction models that utilize prediction errors to detect anomalies. We propose a Long Short Term Memory Networks based **Enc**oder-**Dec**oder scheme for **A**nomaly **D**etection (EncDec-AD) that learns to reconstruct 'normal' time-series behavior, and there-

1. 기계장치들은 일반적으로 기계동작이나 상태를 포착하기 위해 수많은 센서들을 통해 측정되어진다.
2. 하지만 예측하기 어려운 시계열을 발생시키는 외부요인과 변수들이 종종 존재한다.
3. 이러한 시나리오에서 이상치를 탐지하는 것은 수학적 모델의 일반적인 접근으로는 해결하기 어렵다.
4. 이상탐지에 대해 정상작동의 재구성을 학습하는 LSTM기반의 인코더-디코더 구조를 제안한다.

2 / 코드 리뷰

※ https://github.com/JoungheeKim님의 논문구현 코드를 따라 단계별 Pytorch 코드 리뷰와 모델 구성을 파악하였습니다.

1) Sensor Data

timestamp	sensor_00	sensor_01	sensor_02	sensor_03	sensor_04	...	sensor_51	machine_status
2018-04-01 0:00	2.465394	47.09201	53.2118	46.31076	634.375	...	201.3889	NORMAL
2018-04-01 0:01	2.465394	47.09201	53.2118	46.31076	634.375	...	201.3889	NORMAL
2018-04-01 0:02	2.444734	47.35243	53.2118	46.39757	638.8889	...	203.7037	NORMAL
2018-04-01 0:03	2.460474	47.09201	53.1684	46.39757	628.125	...	203.125	NORMAL
2018-04-01 0:04	2.445718	47.13541	53.2118	46.39757	636.4583	...	201.3889	NORMAL
2018-04-01 0:05	2.453588	47.09201	53.1684	46.39757	637.6157	...	201.6782	NORMAL
2018-04-01 0:06	2.455556	47.04861	53.1684	46.39757	633.3333	...	200.2315	NORMAL
2018-04-01 0:07	2.449653	47.13541	53.1684	46.39757	630.6713	...	201.0995	NORMAL
2018-04-01 0:08	2.463426	47.09201	53.1684	46.39757	631.9444	...	201.6782	NORMAL

- 코드리뷰를 위해 사용한 데이터는 Kaggle pump-seneor-data
- 펌프에 부착된 52개의 센서로부터 계측된 값들을 2018년 4월 ~ 2018년 8월 까지 분 단위로 수집한 데이터
- 수집 기간내에 총 7번의 시스템 오류가 존재
- machine_status열로 시스템의 오류상태를 알 수 있음

2) Make Dataset

pytorch의 Dataset을 상속받아 데이터를 구성

Class TagDataset(Dataset):

```python
if mean_df is not None and std_df is not None:
    sensor_columns = [item for item in df.columns if 'sensor_' in item]
    df[sensor_columns] = (df[sensor_columns]-mean_df)/std_df
```

① 센서데이터만 사용, 정규화

```python
## sensor 데이터만 사용하여 reconstruct에 활용
self.selected_column = [item for item in df.columns if 'sensor_' in item][:input_size]
self.var_data = torch.tensor(df[self.selected_column].values, dtype=torch.float)
```

② input_size(40)의 센서데이터만 이용, input_size의 tensor를 만듬

```python
## Dataset은 반드시 __len__ 함수를 만들어줘야함(데이터 길이)
## The __len__ function returns the number of samples in our dataset.
def __len__(self):
    return len(self.input_ids)
```

③ Pytorch Dataset을 상속받은 부분, 데이터 길이를 반환

```python
## torch 모듈은 __getitem__ 을 호출하여 학습할 데이터를 불러옴.
## The __getitem__ function loads and returns a sample from the dataset at the given index idx
def __getitem__(self, item):
    temp_input_ids = self.input_ids[item]
    input_values = self.var_data[temp_input_ids]
    return input_values
```

④ Item(index)에 해당하는 데이터(2번에서 만든 input_size의 tensor데이터)를 실제로 반환

2) Make Dataset

Dataset_Example :

timestamp	sensor_00	...	sensor_39	...	sensor_51
2018-04-01 0:00	2.465394	...	53.2118	...	201.3889
2018-04-01 0:01	2.465394	...	53.2118	...	201.3889
2018-04-01 0:02	2.444734	...	53.2118	...	203.7037
2018-04-01 0:03	2.460474	...	53.1684	...	203.125
2018-04-01 0:04	2.445718	...	53.2118	...	201.3889

Input_size를 40으로 했기 때문에 해당 컬럼만 사용할 예정

sensor_00	...	sensor_39
2.465394	...	53.2118
2.465394	...	53.2118
2.444734	...	53.2118
2.460474	...	53.1684
2.445718	...	53.2118

```
In [2]: normal_dataset1.__getitem__(0)
```

```
Out [2]: tensor([[ -6.9237, -17.0422, -14.1269, -21.9571,  -3.3343,  -3.7683,  -4.8385,
                   -4.9957,  -5.1882,  -5.0451,  -2.9945,  -2.8852,  -2.6036,  -0.9166,
                   -2.6529,  -2.6258,  -2.5788,  -4.5327,  -2.3386,  -2.8390,  -2.8014,
                   -2.3643,  -2.5578,  -2.4139,  -2.3218,  -2.5986,  -2.8804,  -2.3804,
                   -2.1543,  -2.5227,  -2.4582,  -2.5787,  -2.5842,  -2.4279,  -2.5681,
                   -1.6148,  -2.9698,  -4.5814,  -2.2589,  -3.1680],
                 [ -6.8767, -17.0480, -14.1274, -21.9571,  -3.3343,  -3.7680,  -4.8387,
                   -4.9918,  -5.1864,  -5.0435,  -2.9943,  -2.8850,  -2.6039,  -0.9179,
                   -2.6529,  -2.6258,  -2.5788,  -4.5206,  -2.3386,  -2.8390,  -2.8014,
                   -2.3643,  -2.5578,  -2.4139,  -2.3218,  -2.5986,  -2.8804,  -2.3804,
                   -2.1543,  -2.5227,  -2.4582,  -2.5787,  -2.5842,  -2.4279,  -2.5681,
                   -1.6148,  -2.9695,  -4.5811,  -2.2589,  -3.1677],
                 [ -6.9208, -17.0470, -14.1269, -21.9571,  -3.3343,  -3.7680,  -4.8377,
                   -4.9925,  -5.1864,  -5.0435,  -2.9939,  -2.8849,  -2.6037,  -0.9169,
                   -2.6529,  -2.6258,  -2.5788,  -4.3982,  -2.3386,  -2.8390,  -2.8014,
                   -2.3643,  -2.5578,  -2.4139,  -2.3218,  -2.5986,  -2.8804,  -2.3804,
                   -2.1543,  -2.5227,  -2.4582,  -2.5787,  -2.5842,  -2.4279,  -2.5681,
                   -1.6148,  -2.9694,  -4.5806,  -2.2589,  -3.1672]])
```

- window_size를 3으로 입력받아 3개의 row에 대해
센서40개(selected_column)에 해당하는 정규화된 값들이 반환

2) Make Dataset

Dataset_Example :

timestamp	sensor_00	...	sensor_39	...	sensor_51
2018-04-01 0:00	2.465394	...	53.2118	...	201.3889
2018-04-01 0:01	2.465394	...	53.2118	...	201.3889
2018-04-01 0:02	2.444734	...	53.2118	...	203.7037
2018-04-01 0:03	2.460474	...	53.1684	...	203.125
2018-04-01 0:04	2.445718	...	53.2118	...	201.3889

Input_size를 40으로 했기 때문에 해당 컬럼만 사용할 예정

sensor_00	...	sensor_39
2.465394	...	53.2118
2.465394	...	53.2118
2.444734	...	53.2118
2.460474	...	53.1684
2.445718	...	53.2118

```
In [2]: normal_dataset1.__getitem__(1)
```

```
Out [2]: tensor([[ -6.8767, -17.0480, -14.1274, -21.9571,  -3.3343,  -3.7680,  -4.8387,
                   -4.9918,  -5.1864,  -5.0435,  -2.9943,  -2.8850,  -2.6039,  -0.9179,
                   -2.6529,  -2.6258,  -2.5788,  -4.5206,  -2.3386,  -2.8390,  -2.8014,
                   -2.3643,  -2.5578,  -2.4139,  -2.3218,  -2.5986,  -2.8804,  -2.3804,
                   -2.1543,  -2.5227,  -2.4582,  -2.5787,  -2.5842,  -2.4279,  -2.5681,
                   -1.6148,  -2.9695,  -4.5811,  -2.2589,  -3.1677],
                 [ -6.9208, -17.0470, -14.1269, -21.9571,  -3.3343,  -3.7680,  -4.8377,
                   -4.9925,  -5.1864,  -5.0435,  -2.9939,  -2.8849,  -2.6037,  -0.9169,
                   -2.6529,  -2.6258,  -2.5788,  -4.3982,  -2.3386,  -2.8390,  -2.8014,
                   -2.3643,  -2.5578,  -2.4139,  -2.3218,  -2.5986,  -2.8804,  -2.3804,
                   -2.1543,  -2.5227,  -2.4582,  -2.5787,  -2.5842,  -2.4279,  -2.5681,
                   -1.6148,  -2.9694,  -4.5806,  -2.2589,  -3.1672],
                 [ -6.8973, -17.0480, -14.1274, -21.9571,  -3.3343,  -3.7678,  -4.8362,
                   -4.9933,  -5.1820,  -5.0417,  -2.9940,  -2.8848,  -2.6035,  -0.9170,
                   -2.6529,  -2.6258,  -2.5788,  -4.5226,  -2.3386,  -2.8390,  -2.8014,
                   -2.3643,  -2.5578,  -2.4139,  -2.3218,  -2.5986,  -2.8804,  -2.3804,
                   -2.1543,  -2.5227,  -2.4582,  -2.5787,  -2.5842,  -2.4279,  -2.5681,
                   -1.6148,  -2.9699,  -4.5806,  -2.2588,  -3.1668]])
```

- window의 개념으로 __getitem__(1)의 첫번째 값은 __getitem__(0)의 두번째가 됨을 볼 수 있다.

2) Make Dataset

TORCH.UTILS.DATA

Class DataLoder :

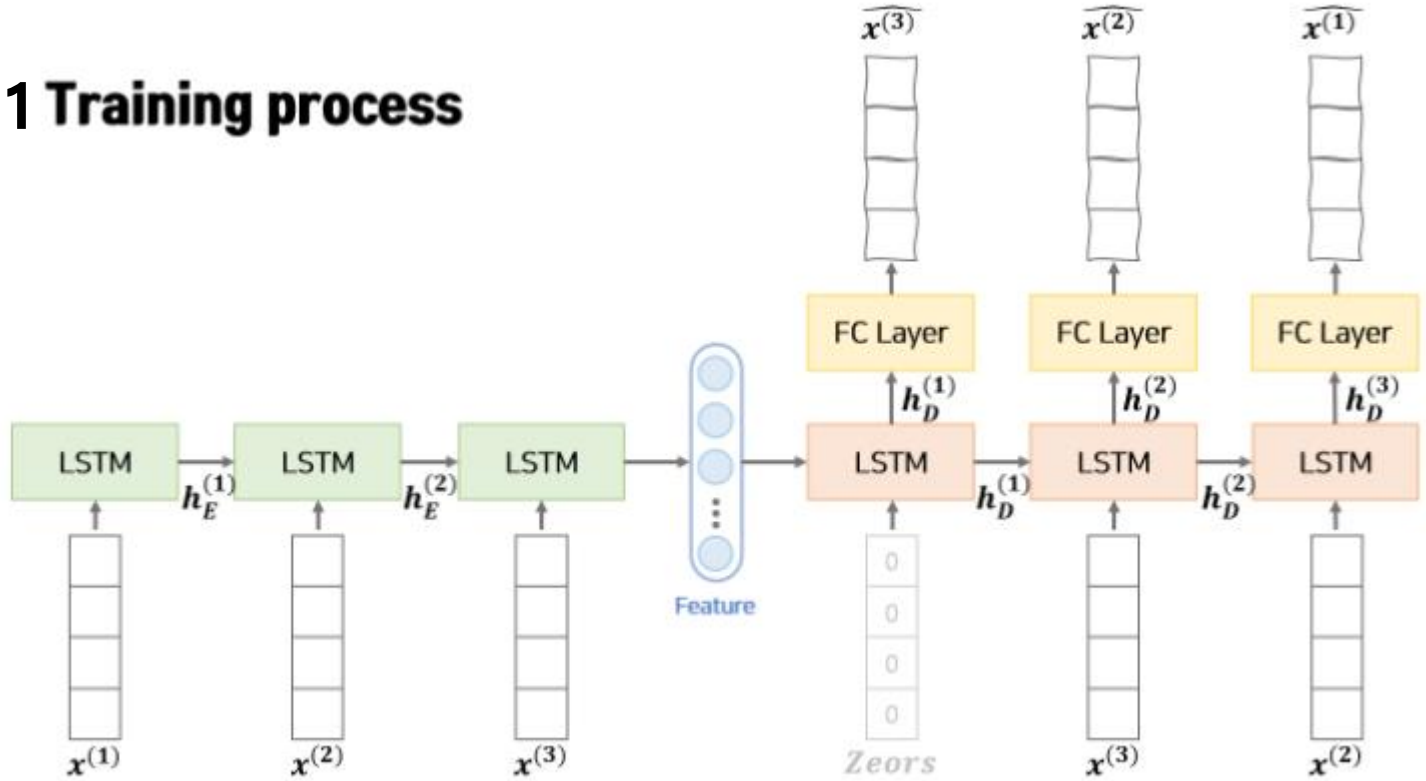
In [2]:
```
DataLoader(dataset, batch_size=1, shuffle=False, sampler=None,
           batch_sampler=None, num_workers=0, collate_fn=None,
           pin_memory=False, drop_last=False, timeout=0,
           worker_init_fn=None, *, prefetch_factor=2,
           persistent_workers=False)
```

- torch.utils.data를 이용해 Data Loader 형태로 변환
- Dataloader class는 batch기반의 딥러닝모델 학습을 위해서 mini batch를 만들어주는 역할
- 앞서 만들었던 dataset을 input으로 넣어주면 여러 옵션(데이터 묶기, 섞기, 알아서 병렬처리)을 통해 batch 생성
- "batch_size " 를 128로 두어 앞서 보았던 3개의 window를 가지는 tensor 128개가 하나의 집단이 되어 학습이 이루어진다.

3) Model Architecture

하이퍼파라미터 설명

1. "input_size": 40
   - 앞서 만든 dataset의 차원수인 40이 입력차원이 된다.
2. "latent_size": 10
   - 40차원의 입력벡터를 10차원으로 압축한다.
3. "output_size": 40
   - autoencoder기 때문에 input_size와 output_size의 크기는 같다.
4. "window_size" : 3
   - Window_size 파라미터를 3으로 두어 sequence Length를 설정

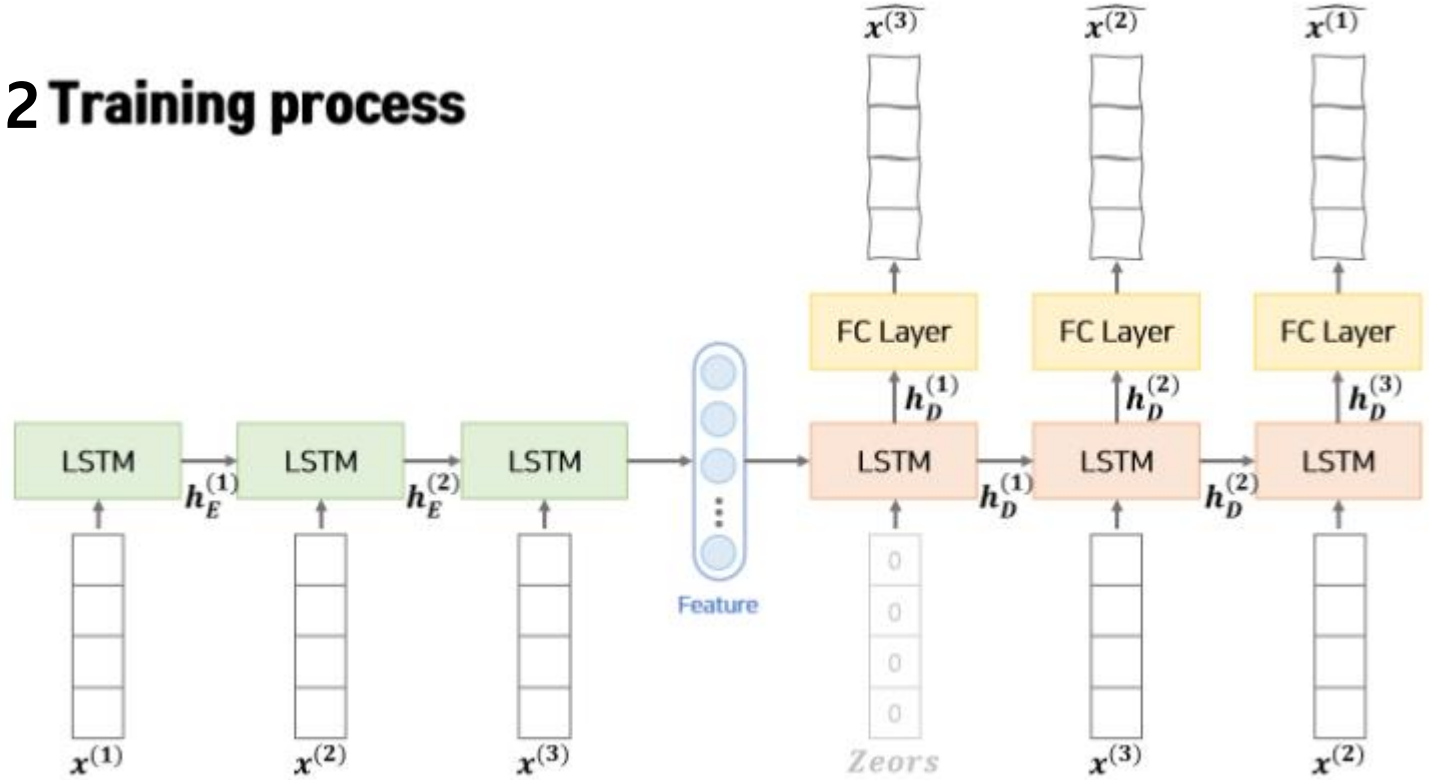


timestamp	sensor_00	...	sensor_39
2018-04-01 0:00	2.465394	...	53.2118
2018-04-01 0:01	2.465394	...	53.2118
2018-04-01 0:02	2.444734	...	53.2118
2018-04-01 0:03	2.460474	...	53.1684
2018-04-01 0:04	2.445718	...	53.2118

3) Model Architecture

인코더 설명

- Encoder는 입력으로 받은 x(t)와 이전 step에서 Encoder로부터 받은 hidden vector인 hE(t−1)을 활용하여 정보를 압축
- Encoder의 마지막 step에서 생성된 hE(t)는 feature vector로 부르며 Decoder의 초기 hidden vector로 활용

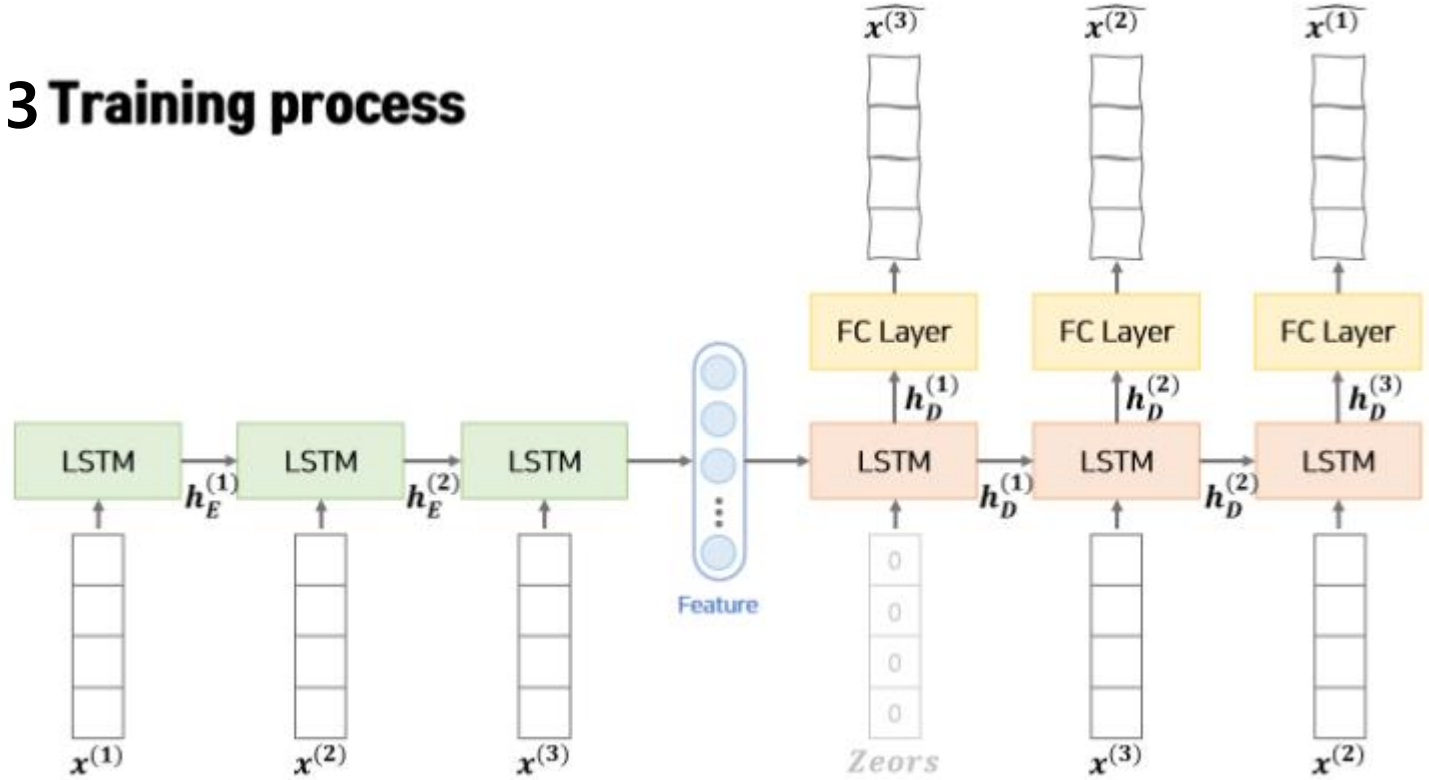


timestamp	sensor_00	...	sensor_39
2018-04-01 0:00	2.465394	...	53.2118
2018-04-01 0:01	2.465394	...	53.2118
2018-04-01 0:02	2.444734	...	53.2118
2018-04-01 0:03	2.460474	...	53.1684
2018-04-01 0:04	2.445718	...	53.2118

3) Model Architecture

디코더 설명

- Decoder는 입력으로 Encoder에서 생상한 feature를 받아 original 데이터를 역순으로 재구성
- decoder 데이터로 인코더의 은닉상태와 0으로 채워진 초기벡터가 입력 데이터로 사용
- Decoder의 매 step 입력으로 original 데이터 역순인 x(t+1)과 이전 step에서 Decoder로부터 받은 hidden vector인 hD(t−1)을 활용하여 정보를 압축하고 다음 step의 hidden vector인 hD(t)를 생성
- Auto-Encoder의 입력인 x(1),x(2),...,x(n) 와 출력인 x(1)^,x(2)^,...,x(n)^ 의 차이인 MSE(Mean Squared Error)를 최소화하는 방향으로 학습



timestamp	sensor_00	...	sensor_39
2018-04-01 0:00	2.465394	...	53.2118
2018-04-01 0:01	2.465394	...	53.2118
2018-04-01 0:02	2.444734	...	53.2118
2018-04-01 0:03	2.460474	...	53.1684
2018-04-01 0:04	2.445718	...	53.2118

4) Anomaly Score

비정상 점수 계산: 1. Reconstruction Error 구하기

get_loss_list : 테스트셋에 대해서 모든 행마다 predict값과 real값의 L1_loss를 담은 loss_list를 반환

$e^{(i)} = ||x^{(i)} - \hat{x^{(i)}}||$ $e^{(i)}$: i 지점의 Reconstruction Error

In [2]:

```
def get_loss_list(args, model, test_loader):
    test_iterator = tqdm(enumerate(test_loader), total=len(test_loader), desc="testing")
    loss_list = []

    with torch.no_grad():
        ## 검증하고자 하는 데이터의 각 row에 대해서
        for i, batch_data in test_iterator:

            batch_data = batch_data.to(args.device)
            predict_values = model(batch_data)

            ## MAE(Mean Absolute Error)로 계산
            ## forward함수의 출력이 [reconstruct_output, src]이므로 predict_values[0]이 예측값, [1]이 실제값
            loss = nn.functional.l1_loss(predict_values[0], predict_values[1], reduce=False)

            loss = loss.mean(dim=1).cpu().numpy()
            loss_list.append(loss)
    loss_list = np.concatenate(loss_list, axis=0)
    return loss_list
```

4) Anomaly Score

비정상 점수 계산: 2. Anomaly Score 구하기

predict값과 real값의 loss를 담은 loss_list의 평균과 표준편차를 통해 이상치 점수를 산출

In [2]:

```
## Reconstruction Error의 평균과 Covarinace 계산
mean = np.mean(loss_list, axis=0)
std = np.cov(loss_list.T)

class Anomaly_Calculator:

    def __call__(self, recons_error:np.array):
        x = (recons_error-self.mean)
        return np.matmul(np.matmul(x, self.std), x.T)


## 이상치 점수 계산하기
anomaly_scores = []
for temp_loss in tqdm(total_loss):
    temp_score = anomaly_calculator(temp_loss)
    anomaly_scores.append(temp_score)
```

비정상 점수 계산: 3. Threshold 찾기

Get_loss_list를 통해 얻은 loss_list에 대해서 위에서 정의한 anomaly score값을 계산
Anomaly score의 분포를 통해 정상과 이상치를 구분할 수 있는 threshold를 찾음

4) Anomaly Score

Example :

```
In [2]: Loss_list
```

```
Out[2]: array([[0.04841939, 1.106026  , 0.13421492, ..., 0.64909226, 0.08109307,
                0.49383393],
               [0.04548374, 1.0907255 , 0.16203715, ..., 0.59377766, 0.10367555,
                0.4685197 ],
               [0.03334466, 1.053698  , 0.18449809, ..., 0.6279655 , 0.1217952 ,
                0.51645947],
               ...,
               [0.26560387, 0.81461024, 0.44812092, ..., 0.09729228, 0.6041043 ,
                0.7180595 ],
               [0.26661506, 0.7991597 , 0.46541524, ..., 0.12063324, 0.5796008 ,
                0.7741154 ],
               [0.23873769, 0.7656827 , 0.5035777 , ..., 0.06630987, 0.5520674 ,
                0.7771657 ]], dtype=float32)
```

```
In [2]: print(len(normal_dataset2))
        print(len(loss_list))
        print(len(loss_list[0]))
```

```
Out[2]: 19620
        19620
        40
```

- get_loss_list를 통해 얻은 loss_list
- 테스트셋에 대해 모든 행마다 predict값과 real값의 차이가 담겨있음.
- 센서40개에 대해 reconstruction을 구하기 때문에 행마다 40개의 요소를 가지고 있음.

```
In [2]: print(len(mean))
        mean
```

```
Out[2]: 40
```

각 열에 대해 평균값을 가지기 때문에 40요소를 가짐

```
array([0.1124285 , 0.9433947 , 0.27921733, 0.48984355, 0.14115635,
       0.13264062, 0.13627566, 0.16457261, 0.18576843, 0.1621946 ,
       0.20577076, 0.6282747 , 0.25500405, 0.23507975, 0.05727583,
       0.05133757, 0.19506602, 0.2191343 , 0.03903371, 0.04116544,
       0.06328755, 0.05340189, 0.24777192, 0.04243483, 0.10736197,
       0.12453719, 0.16001491, 0.20859778, 0.11175483, 0.22855373,
       0.19089158, 0.2903897 , 0.11069734, 0.37517166, 0.13214566,
       0.24636285, 0.315132  , 0.37032217, 0.26478842, 0.4671727 ],
      dtype=float32)
```

```
In [2]: print(len(std))
        std
```

```
Out[2]: 40
```

```
array([[ 1.36531981e-02, -1.69235749e-03, -3.47682769e-05, ...,
         2.34861267e-03,  9.75462970e-03,  1.37457281e-02],
       [-1.69235749e-03,  3.31854328e-02, -1.30182881e-02, ...,
         1.12400443e-03, -2.39473876e-03, -9.20807850e-04],
       [-3.47682769e-05, -1.30182881e-02,  3.55921812e-02, ...,
         2.08166514e-03, -1.09678613e-03,  4.96228538e-03],
       ...,
       [ 2.34861267e-03,  1.12400443e-03,  2.08166514e-03, ...,
         7.88582561e-02,  7.48638088e-03,  4.74519059e-02],
       [ 9.75462970e-03, -2.39473876e-03, -1.09678613e-03, ...,
         7.48638088e-03,  6.42404776e-02,  3.75732162e-02],
       [ 1.37457281e-02, -9.20807850e-04,  4.96228538e-03, ...,
         4.74519059e-02,  3.75732162e-02,  1.54343455e-01]])
```

4) Anomaly Score

Example :

```
return np.matmul(np.matmul(x, self.std), x.T)
```

loss_list[0]

```
array([0.05713118, 1.0743984 , 0.14493991, 0.45254168, 0.05341689,
       0.19082153, 0.05956356, 0.07983622, 0.13749012, 0.07656614,
       0.29213893, 0.3196945 , 0.31011567, 0.1626014 , 0.03636823,
       0.03590527, 0.19451152, 0.24518563, 0.01797611, 0.01489301,
       0.04884903, 0.07329591, 0.24536379, 0.0164234 , 0.02174884,
       0.13282506, 0.1678089 , 0.12191073, 0.07838083, 0.1502966 ,
       0.03968389, 0.2275219 , 0.04913597, 0.11209228, 0.58022946,
       0.23724782, 0.10477284, 0.6342156 , 0.07148065, 0.52043587],
      dtype=float32)
```

×

std

```
array([[ 1.36531981e-02, -1.69235749e-03, -3.47682769e-05, ...,
         2.34861267e-03,  9.75462970e-03,  1.37457281e-02],
       [-1.69235749e-03,  3.31854328e-02, -1.30182881e-02, ...,
         1.12400443e-03, -2.39473876e-03, -9.20807850e-04],
       [-3.47682769e-05, -1.30182881e-02,  3.55921812e-02, ...,
         2.08166514e-03, -1.09678613e-03,  4.96228538e-03],
       ...,
       [ 2.34861267e-03,  1.12400443e-03,  2.08166514e-03, ...,
         7.88582561e-02,  7.48638088e-03,  4.74519059e-02],
       [ 9.75462970e-03, -2.39473876e-03, -1.09678613e-03, ...,
         7.48638088e-03,  6.42404776e-02,  3.75732162e-02],
       [ 1.37457281e-02, -9.20807850e-04,  4.96228538e-03, ...,
         4.74519059e-02,  3.75732162e-02,  1.54343455e-01]])
```

=

첫번째 행에 대해 40개 센서의 reconstruction error와 std의 벡터 곱

```
array([ 0.0212276 ,  0.02502773, -0.00542879,  0.02143034,  0.02242229,
        0.00890567,  0.00284957,  0.00673597,  0.00652451,  0.01124235,
        0.01308363, -0.0069304 ,  0.00791578,  0.01295564,  0.00376368,
        0.00355723,  0.00580922,  0.00685868,  0.00609671,  0.00603678,
        0.00433213,  0.00475891,  0.0027714 ,  0.00560073,  0.00845432,
        0.00675787,  0.0120653 ,  0.00883632,  0.00395503,  0.00695962,
        0.00324605,  0.02937878,  0.01747451,  0.00337277,  0.02199235,
        0.02132908,  0.01745883,  0.08394575,  0.04126819,  0.12337069])
```

loss_list[0].T)

```
array([ 0.0212276 ,  0.02502773, -0.00542879,  0.02143034,  0.02242229,
        0.00890567,  0.00284957,  0.00673597,  0.00652451,  0.01124235,
        0.01308363, -0.0069304 ,  0.00791578,  0.01295564,  0.00376368,
        0.00355723,  0.00580922,  0.00685868,  0.00609671,  0.00603678,
        0.00433213,  0.00475891,  0.0027714 ,  0.00560073,  0.00845432,
        0.00675787,  0.0120653 ,  0.00883632,  0.00395503,  0.00695962,
        0.00324605,  0.02937878,  0.01747451,  0.00337277,  0.02199235,
        0.02132908,  0.01745883,  0.08394575,  0.04126819,  0.12337069])
```

×

```
array([0.05713118, 1.0743984 , 0.14493991, 0.45254168, 0.05341689,
       0.19082153, 0.05956356, 0.07983622, 0.13749012, 0.07656614,
       0.29213893, 0.3196945 , 0.31011567, 0.1626014 , 0.03636823,
       0.03590527, 0.19451152, 0.24518563, 0.01797611, 0.01489301,
       0.04884903, 0.07329591, 0.24536379, 0.0164234 , 0.02174884,
       0.13282506, 0.1678089 , 0.12191073, 0.07838083, 0.1502966 ,
       0.03968389, 0.2275219 , 0.04913597, 0.11209228, 0.58022946,
       0.23724782, 0.10477284, 0.6342156 , 0.07148065, 0.52043587],
      dtype=float32)
```

loss_list[0]를 단순히 전치시킨 행렬

=

첫번째 행의 이상치 점수가 반환

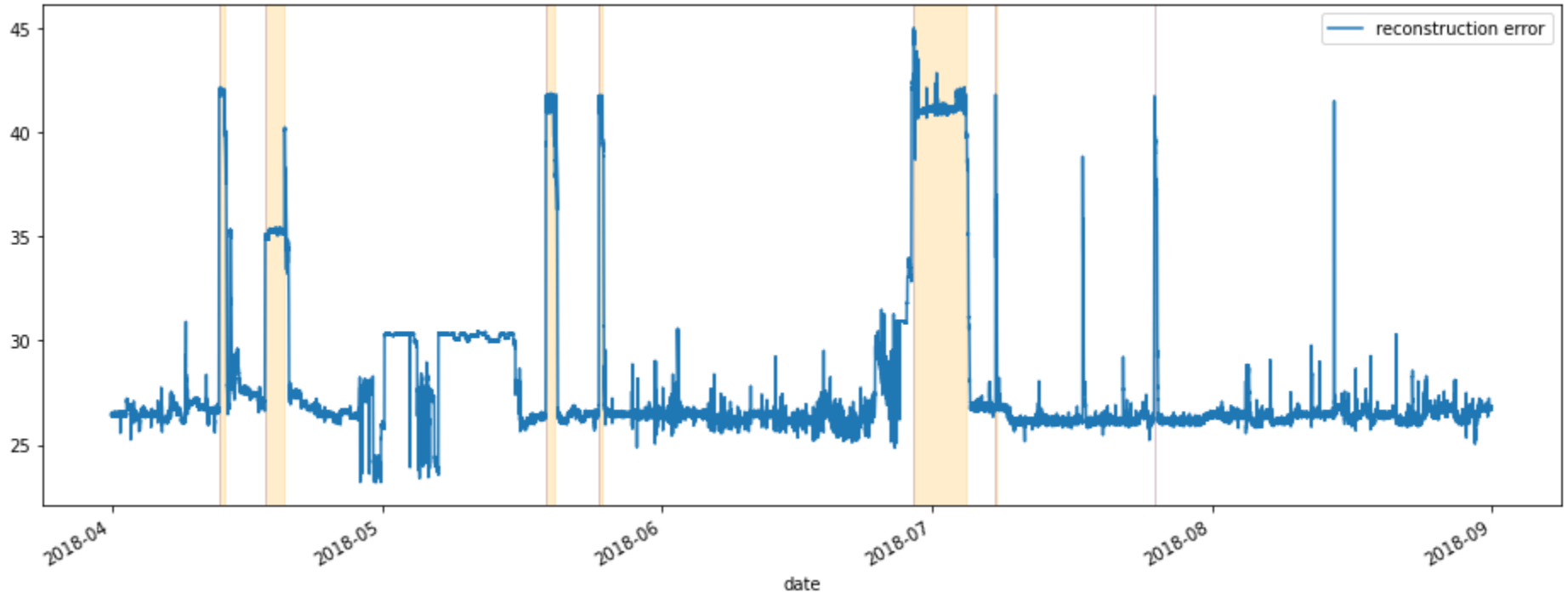
```
0.20678783818440993
```

5) Visualize Result

- 앞에서 정상 데이터 만을 이용해 학습을 시켰다면, 이제 모든 데이터들에 대해 모델을 통과시켜 이상치 점수 계산
- 정상과 비정상 데이터의 Reconstruction error, abnormal score를 비교해보면서 학습이 잘 이루어졌는지 파악
- 다음은 Reconstruction error에 대한 시각화이며 기존 7번의 이상치를 모두 탐지한 것을 알 수 있음

timestamp	Machine_status	score	Recons_error
2018-04-01 0:00	NORMAL	6.856698	26.48537
2018-04-01 0:01	NORMAL	6.838026	26.46287
…	…	…	…
2018-08-31 23:59	NORMAL	7.177563	26.80827

Reconstruction Error Graph



5) Visualize Result

- 앞에서 정상 데이터 만을 이용해 학습을 시켰다면, 이제 모든 데이터들에 대해 모델을 통과시켜 이상치 점수 계산
- 정상과 비정상 데이터의 Reconstruction error, abnormal score를 비교해보면서 학습이 잘 이루어졌는지 파악
- 다음은 abnormal score에 대한 시각화

timestamp	Machine_status	score	Recons_error
2018-04-01 0:00	NORMAL	6.856698	26.48537
2018-04-01 0:01	NORMAL	6.838026	26.46287
…	…	…	…
2018-08-31 23:59	NORMAL	7.177563	26.80827

Abnormal Score Graph