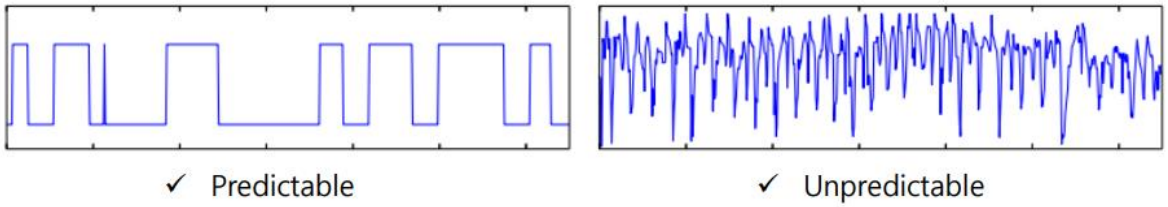
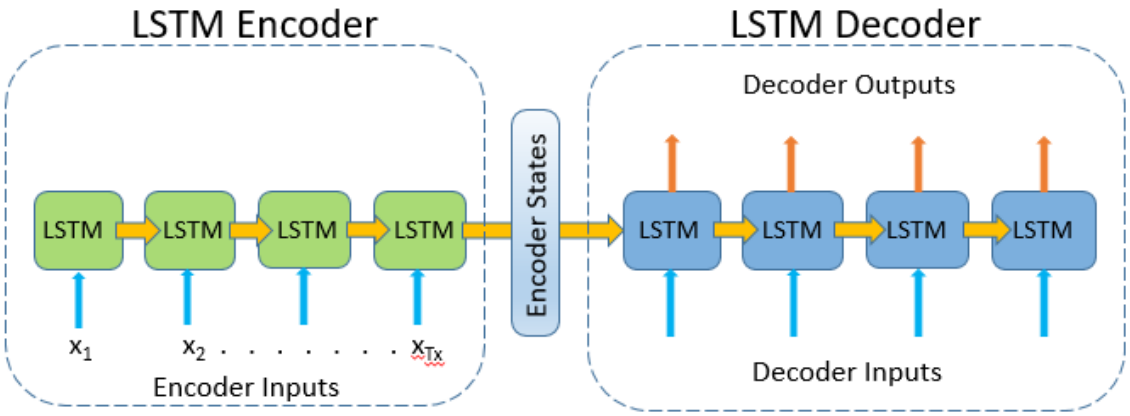
LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection

2020020534 김정원

- 본 논문은 Multi-sensor 데이터의 정상 데이터를 학습하여 Reconstruction Error를 통해서 Anomaly detection을 할 수 있는 LSTM based Encoder-Decoder scheme for Anomaly Detection 방법을 제안하였다. 또한 Predictable data 와 Unpredictable data 모두 적용 가능하다고 말하고 있다.



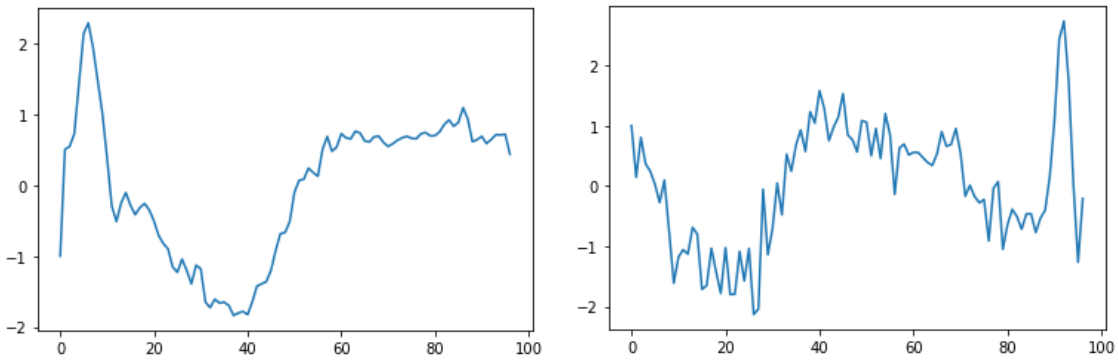


- 논문에서 제안하는 LSTM Encoder-Decoder의 구조는 위와 같이 이루어져 있다. Input data인 $(x_1, x_2, \dots, x_T)$를 LSTM Encoder를 통해서 Encoder States로 변환(Latent space의 vector)하여 LSTM Decoder를 통해 $(x_1', x_2', \dots, x_T')$를 복원한다. 이 때 Reconstruction Error인 $\sum_{i=1}^{T} \|X_i - X_i'\|^2$을 minimize 하도록 학습을 진행한다. 즉 정상 데이터에 대해서 잘 복원을 하는 방향으로 학습을 진행한다.

Data & Library Load

```python
import pandas as pd
import matplotlib.pyplot as plt
import os
import numpy as np
import sklearn
import collections
import torch.nn.functional as F
import torch
import torch.nn as nn
import seaborn as sns

from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA

is_cuda = torch.cuda.is_available()

if is_cuda:
    device = torch.device("cuda")
    print("GPU is available")
else:
    device = torch.device("cpu")
    print("GPU not available, CPU used")


colnames = ["label"] + list(np.arange(1,97))

data = pd.read_csv("C:/Users/KJW/Downloads/ECG200/ECG200_TRAIN.txt",sep = "  " ,names = colnames)
test_data = pd.read_csv("C:/Users/KJW/Downloads/ECG200/ECG200_TEST.txt",sep = "  " ,names = colnames)
```

- 본 논문의 구현을 위해 사용한 데이터인 ECG200는 heartbeat의 electrical activity를 record하였다. Train data의 개수는 100개 Test data의 개수도 100개로 각 데이터마다 Normal/Ischemia의 두개의 Class가 존재함. 그리고 모든 데이터의 길이는 96으로 동일하다.



- 왼쪽 데이터는 Normal Class의 예시이고, 오른쪽 데이터는 Ischemia Class의 예시이다.

Data preprocessing

```
data["label"] = data["label"].astype(int)
test_data["label"] = test_data["label"].astype(int)

normal_train_data = data[data.label == 1].reset_index(drop = True).drop(["label"] , axis = 1)
abnormal_data = data[data.label == -1].reset_index(drop = True).drop(["label"] , axis = 1)
normal_test_data = test_data[test_data.label == 1].reset_index(drop = True).drop(["label"] , axis = 1)
abnormal_test_data = test_data[test_data.label == -1].reset_index(drop = True).drop(["label"] , axis = 1)

def create_dataset(df):

  sequences = df.astype(np.float32).to_numpy().tolist()
  dataset = [torch.tensor(s).unsqueeze(1).float() for s in sequences]
  n_seq, seq_len, n_features = torch.stack(dataset).shape

  return dataset, seq_len, n_features

train_dataset, seq_len, n_features = create_dataset(normal_train_data)
Anomaly_dataset, seq_len , n_features = create_dataset(abnormal_data)
normal_testset , seq_len , n_features = create_dataset(normal_test_data)
abnormal_testset , seq_len , n_features = create_dataset(abnormal_test_data)
```

- 본 논문의 구현은 Normal data만을 사용하여 학습을 진행하기 때문에 Train data에서 Normal(label = 1)인 데이터만을 학습에 사용하였다.

Encoder

```
class Encoder(nn.Module):

  def __init__(self, seq_len, n_features, embedding_dim=64):

    super(Encoder, self).__init__()

    self.seq_len, self.n_features = seq_len, n_features
    self.embedding_dim, self.hidden_dim = embedding_dim, 2 * embedding_dim
    self.rnn1 = nn.LSTM(
      input_size=n_features,
      hidden_size=self.hidden_dim,
      num_layers=1,
      batch_first=True
    )
    self.rnn2 = nn.LSTM(
      input_size=self.hidden_dim,
      hidden_size=embedding_dim,
      num_layers=1,
      batch_first=True
    )

  def forward(self, x):

    x = x.reshape((1, self.seq_len, self.n_features))
    x, (_, _) = self.rnn1(x)
    x, (hidden_n, _) = self.rnn2(x)

    return hidden_n.reshape((self.n_features, self.embedding_dim))
```

Decoder

```
class Decoder(nn.Module):
  def __init__(self, seq_len, input_dim=64, n_features=1):
    super(Decoder, self).__init__()

    self.seq_len, self.input_dim = seq_len, input_dim
    self.hidden_dim, self.n_features = 2 * input_dim, n_features
    self.rnn1 = nn.LSTM(
      input_size=input_dim,
      hidden_size=input_dim,
      num_layers=1,
      batch_first=True
    )

    self.rnn2 = nn.LSTM(
      input_size=input_dim,
      hidden_size=self.hidden_dim,
      num_layers=1,
      batch_first=True
    )

    self.output_layer = nn.Linear(self.hidden_dim, n_features)

  def forward(self, x):

    x = x.repeat(self.seq_len, self.n_features)
    x = x.reshape((self.n_features, self.seq_len, self.input_dim))
    x, (hidden_n, cell_n) = self.rnn1(x)
    x, (hidden_n, cell_n) = self.rnn2(x)
    x = x.reshape((self.seq_len, self.hidden_dim))

    return self.output_layer(x)
```

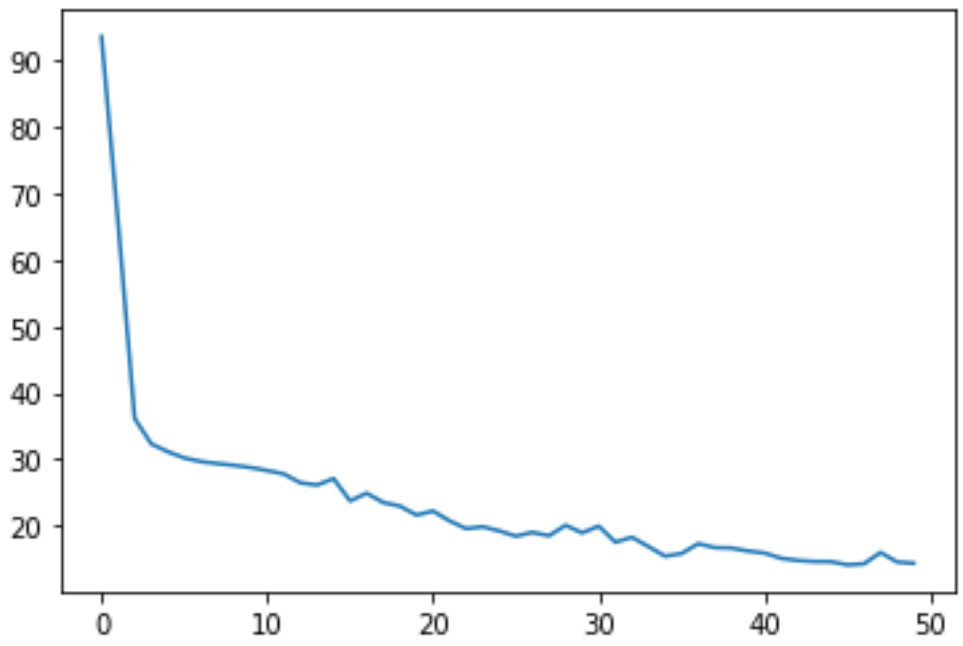
LSTM_EncDec

```
class LSTM_EncDec(nn.Module):

  def __init__(self, seq_len, n_features, embedding_dim=64):
    super(LSTM_EncDec, self).__init__()
    self.encoder = Encoder(seq_len, n_features, embedding_dim).to(device)
    self.decoder = Decoder(seq_len, embedding_dim, n_features).to(device)

  def forward(self, x):
    x = self.encoder(x)
    x = self.decoder(x)

    return x
```

Train

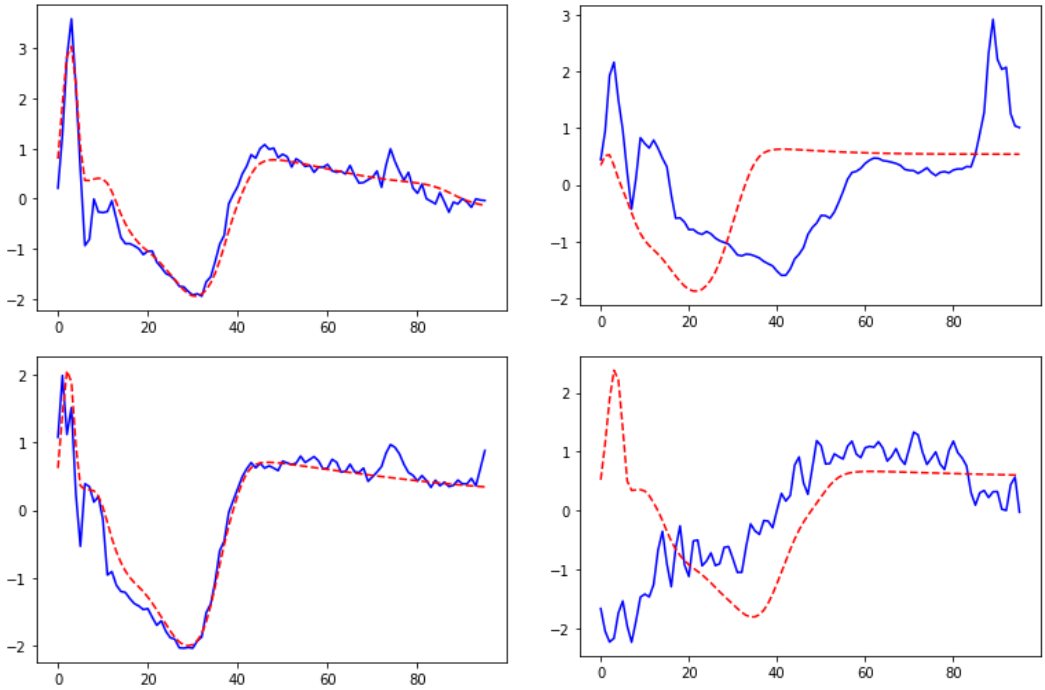
```
model = LSTM_EncDec(seq_len, n_features, 16)
model = model.to(device)
n_epochs = 50

optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

criterion = nn.MSELoss(reduction='sum').to(device)
history = []

for epoch in range(1, n_epochs+1):

    model = model.train()
    train_losses = []

    for seq_true in train_dataset:

        optimizer.zero_grad()

        seq_true = seq_true.to(device)
        seq_pred = model(seq_true)

        loss = criterion(seq_pred , seq_true)
        loss.backward()
        optimizer.step()
        train_losses.append(loss.item())

    train_loss = np.mean(train_losses)
    history.append(train_loss)

    print(f'Epoch {epoch}: train_loss {train_loss}')
```

- 모델의 학습을 위한 LSTM의 hidden size는 16, Epoch는 50회 수행하였다.



- 위 그래프는 학습시에 Epoch에 따른 Loss의 변화 그래프이다.

Test



- 위 그림은 Test data의 Normal/Ischemia Class에 대한 reconstruction 결과이다. 파란색 선이 실제 데이터이고 빨간색 점선이 model을 통해 복원된 데이터이다. 왼쪽은 Normal Class이고, 오른쪽은 Ischemia Class이다.