```
import torch
import torch.nn as nn
import torch.optim as optim
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from torch.utils.data import DataLoader, TensorDataset

received = pd.read_csv("Received_data_set.csv")
transmitted = pd.read_csv("Transmitted_data_set.csv")

received.columns = ["Received_Signal"]
transmitted.columns = ["Transmitted_Signal"]

mean_X, std_X = received["Received_Signal"].mean(),
received["Received_Signal"].std()
X = (received["Received_Signal"] - mean_X) / std_X
X = X.values.reshape(-1, 1)
Y = (received["Received_Signal"] -
transmitted["Transmitted_Signal"]).values.reshape(-1, 1)

indices = np.arange(X.shape[0])
np.random.shuffle(indices)
X, Y = X[indices], Y[indices]

sample = X.shape[0]
size_t = int(sample * 0.5)
size_v = int(sample * 0.2)

X_t, Y_t = X[:size_t], Y[:size_t]
X_v, Y_v = X[size_t:size_t + size_v], Y[size_t:size_t + size_v]
X_test, Y_test = X[size_t + size_v:], Y[size_t + size_v:]

device = torch.device("cpu")

X_t_tensor, Y_t_tensor = torch.FloatTensor(X_t).to(device),
torch.FloatTensor(Y_t).to(device)
X_v_tensor, Y_v_tensor = torch.FloatTensor(X_v).to(device),
torch.FloatTensor(Y_v).to(device)
X_test_tensor, Y_test_tensor = torch.FloatTensor(X_test).to(device),
torch.FloatTensor(Y_test).to(device)

class NoisePredictorMLP(nn.Module):
    def __init__(self):
        super(NoisePredictorMLP, self).__init__()
        self.fc1 = nn.Linear(1, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, 16)
```

```
        self.fc4 = nn.Linear(16, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = self.fc4(x)
        return x

model = NoisePredictorMLP().to(device)

loss_function = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

num_epochs = 100
train_losses = []
val_losses = []

train_dataset = TensorDataset(X_t_tensor, Y_t_tensor)
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)

for epoch in range(num_epochs):
    model.train()
    epoch_loss = 0.0

    for batch_X, batch_Y in train_loader:
        batch_X = batch_X.to(device)
        batch_Y = batch_Y.to(device)

        optimizer.zero_grad()
        outputs = model(batch_X)
        loss = loss_function(outputs, batch_Y)
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()

    avg_train_loss = epoch_loss / len(train_loader)
    val_loss = loss_function(model(X_v_tensor), Y_v_tensor).item()

    train_losses.append(avg_train_loss)
    val_losses.append(val_loss)

    if epoch % 10 == 0 or epoch == num_epochs:
        print(f"Epoch [{epoch}/{num_epochs}], Train Loss:
{avg_train_loss:.4f}, Val Loss: {val_loss:.4f}")

model.eval()
with torch.no_grad():
```

```
    predictions = model(X_test_tensor).cpu().numpy()

test_results = pd.DataFrame({
    "Actual Noise": Y_test.flatten(),
    "Predicted Noise": predictions.flatten()
})

test_results_20 = test_results.head(20)
print("\n20 개 샘플")
print(test_results_20)

mae_20 = np.mean(np.abs(test_results_20["Actual Noise"] -
test_results_20["Predicted Noise"])) # 원래 노이즈 - 예상 노이즈의 평균
print(f"\n 평균 오차 (MAE): {mae_20:.4f}")

plt.figure(figsize=(8, 5))
plt.plot(range(num_epochs), train_losses, label='Train Loss')
plt.plot(range(num_epochs), val_losses, label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Trend')
plt.legend()
plt.show()
```

received와 transmitted를 직접 대응시킬 경우 생각보다 loss가 매우컸다. 데이터들이 1열로 단순하기 때문에 어떤 상관관계를 분석하기 어렵기 때문이라고 추청. 아이디어를 수정하여 y에 보낸 값과 받은 값의 차이, 즉 노이즈를 대응. 보낸 값에 대한 노이즈를 예측하는 이 모델을 통과시켜 깨끗한 신호로 복원 가능.

결과: 첫번째 데이터셋

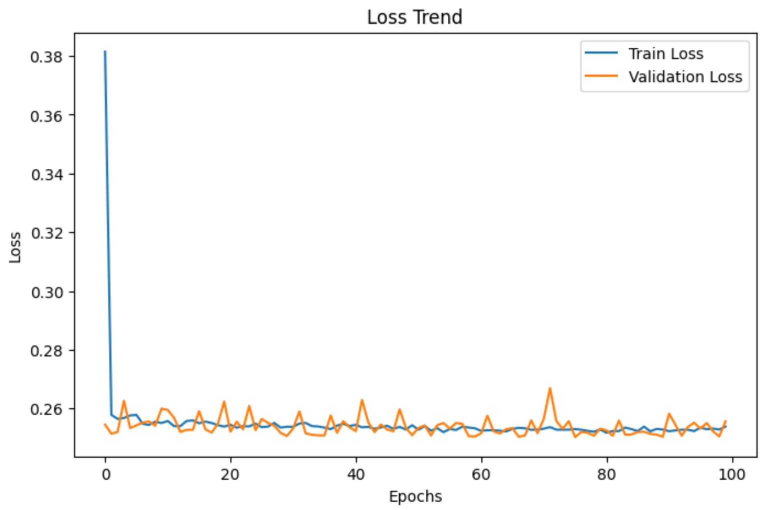
```
Epoch [0/100], Train Loss: 0.3815, Val Loss: 0.2545
Epoch [10/100], Train Loss: 0.2557, Val Loss: 0.2595
Epoch [20/100], Train Loss: 0.2544, Val Loss: 0.2521
Epoch [30/100], Train Loss: 0.2537, Val Loss: 0.2532
Epoch [40/100], Train Loss: 0.2544, Val Loss: 0.2523
Epoch [50/100], Train Loss: 0.2528, Val Loss: 0.2534
Epoch [60/100], Train Loss: 0.2524, Val Loss: 0.2515
Epoch [70/100], Train Loss: 0.2531, Val Loss: 0.2565
Epoch [80/100], Train Loss: 0.2517, Val Loss: 0.2526
Epoch [90/100], Train Loss: 0.2522, Val Loss: 0.2582
```

-> 큰 loss값

```
20개 샘플
    Actual Noise  Predicted Noise
0      -1.198660        -0.672723
1       0.088624        -0.406842
2      -0.727190        -0.229257
3      -1.609000        -2.099042
4       0.152770        -0.338279
5       0.055854        -0.439568
6      -0.837060        -0.327377
7      -0.273540        -0.737704
8      -0.328640         0.216280
9       1.991600         1.636667
10      0.700000         1.315316
11     -0.632600        -1.113947
12     -2.522900        -3.013796
13      1.306700         1.982606
14      2.482400         2.173417
15      0.135180        -0.357136
16     -2.896600        -3.381291
17      0.195830        -0.294086
18     -2.646900        -3.137294
19     -2.101300        -1.601965

평균 오차 (MAE): 0.4949
```

-> 랜덤하게 고른 20개의 샘플에 대한 평균 오차는 0.4949로 매우 큰 값을 보였다.



-> loss 그래프도 유의미한 변화를 보여주진 않음

두번째 데이터셋: 동일모델, 데이터셋만 바꿈

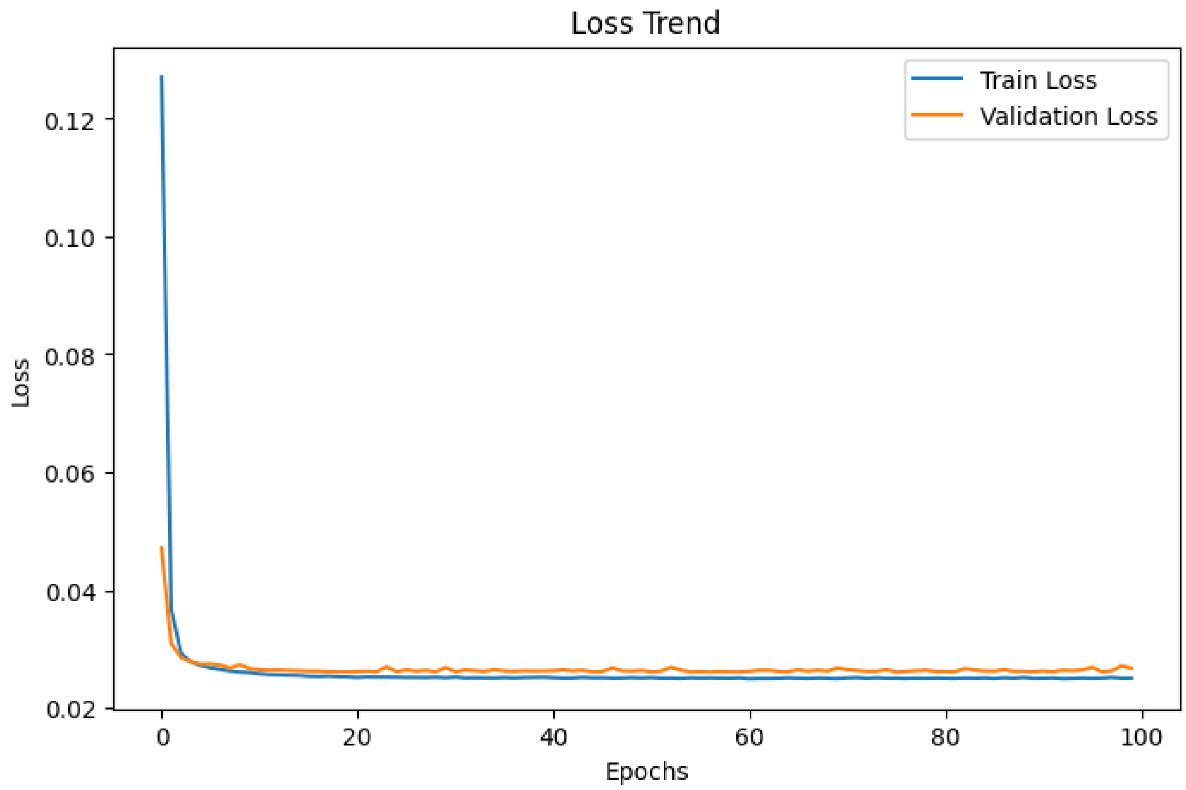
```
Epoch [0/100], Train Loss: 0.1271, Val Loss: 0.0471
Epoch [10/100], Train Loss: 0.0258, Val Loss: 0.0264
Epoch [20/100], Train Loss: 0.0251, Val Loss: 0.0261
Epoch [30/100], Train Loss: 0.0252, Val Loss: 0.0261
Epoch [40/100], Train Loss: 0.0251, Val Loss: 0.0263
Epoch [50/100], Train Loss: 0.0251, Val Loss: 0.0261
Epoch [60/100], Train Loss: 0.0249, Val Loss: 0.0262
Epoch [70/100], Train Loss: 0.0250, Val Loss: 0.0265
Epoch [80/100], Train Loss: 0.0250, Val Loss: 0.0261
Epoch [90/100], Train Loss: 0.0250, Val Loss: 0.0262
```

-> 작아진 loss

```
20개 샘플
    Actual Noise  Predicted Noise
0      -0.288680        -0.343667
1      -1.806000        -1.805401
2      -1.914600        -1.913348
3      -1.970400        -1.969398
4      -1.394600        -1.393303
5      -1.088300        -1.086229
6      -1.807900        -1.807295
7      -1.133100        -1.130680
8      -0.964820        -0.964007
9      -0.528700        -0.487327
10     -0.014070         0.020942
11     -1.058400        -1.056563
12     -0.305820        -0.273122
13     -0.434660        -0.403950
14     -0.992921        -0.491740
15      0.172500         0.207954
16      0.345100         0.386642
17     -1.007700        -1.006417
18     -1.868000        -1.866572
19     -0.076420        -0.040813

평균 오차(MAE): 0.0412
```

-> 20개 샘플에 대한 평균 오차는 0.0412로 작아졌으며, actual noise와 predicted noise의 차이가 크지 않음을 확인 가능



두번째 데이터셋에 대한 손실그래프, 특정 반복을 넘어가면 training 데이터에 과적합되어 validation 데이터들에 대해서 손실이 조금씩 증가하는 현상을 관찰 가능하다.

개선점: 손실 값이 0.002대에서 수렴하는 것을 조금 더 낮출 방법을 모색해야 한다. MLP의 뉴런 수를 늘려서 더 깊은 모델을 만드는 방법, dropout을 추가하는 방법을 시도해봄직 하다. 혹은 시계열모델인 RNN, LSTM을 적용해보아도 좋을 듯하다.