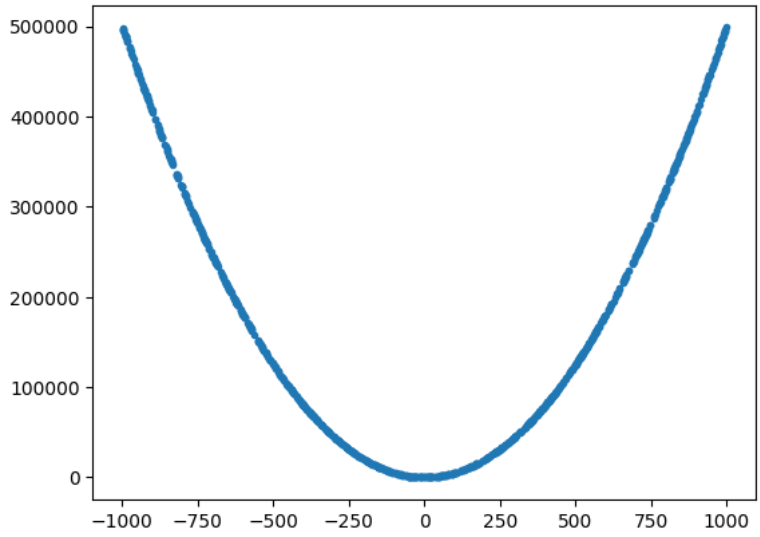
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```

from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_california_housing
from sklearn.preprocessing import StandardScaler
from torchvision import datasets, transforms

import copy
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import tqdm


```

```
N, D_in, D_out = 1000, 2, 1


# m =1
X2 = []
y2 = []
for j in range(N):
    X2.append([])
    #y.append([])
    for i in range(D_out):
        X2[-1].append( 1 )
        X2[-1].append( np.random.uniform(low=-1000.0, high=1000.0, size=None) )
        y2.append( (X2[-1][-1]**2) / (2*X2[-1][-2])  )


X2 = torch.Tensor(X2)
y2 = torch.Tensor(y2)


# p =1
X3 = []
y3 = []
for j in range(N):
    X3.append([])
    #y.append([])
    for i in range(D_out):
        X3[-1].append( np.random.uniform(low=0.0, high=10.0, size=None) )
        X3[-1].append( 1 )
        y3.append( (X3[-1][-1]**2) / (2*X3[-1][-2])  )


X3 = torch.Tensor(X3)
y3 = torch.Tensor(y3)

# X2_test = torch.tensor(X2, dtype=torch.float32)
# y2_test = torch.tensor(y2, dtype=torch.float32).reshape(-1, 1)
# X3_test = torch.tensor(X3, dtype=torch.float32)
# y3_test = torch.tensor(y3, dtype=torch.float32).reshape(-1, 1)
```

```
plt.scatter(X2[:,1] ,y2, marker='.', )
plt.show()
plt.scatter(X3[:,0] ,y3, marker='.', )
plt.show()

```




```
1
2 X = torch.load('/content/drive/MyDrive/0_318lab/SCMP_ML/KEdataX_1000_1690897080.8460803.pt')
3 y = torch.load('/content/drive/MyDrive/0_318lab/SCMP_ML/KEdataY_1000_1690897080.8489494.pt')
4 X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, shuffle=False)
5 # X2_train, X2_test, y2_train, y2_test = train_test_split(X, y, train_size=0.7, shuffle=True)
6 # X3_train, X3_test, y3_train, y3_test = train_test_split(X, y, train_size=0.7, shuffle=True)
7
8 # Convert to 2D PyTorch tensors
9 #X_train = torch.tensor(X_train, dtype=torch.float32)
10 #y_train = torch.tensor(y_train, dtype=torch.float32).reshape(-1, 1)
11 #X_test = torch.tensor(X_test, dtype=torch.float32)
12 #y_test = torch.tensor(y_test, dtype=torch.float32).reshape(-1, 1)
13
14
15
```

```
1
```

```
tensor([[ 147.4935, -818.5190],
        [ 625.2579,   44.7736],
        [ 806.2130,  130.8175],
        ...,
        [ 514.6038,  791.7943],
        [ 112.5630, -573.6776],
        [ 942.1508, -727.9680]])
```

```
1 len(before_loss)
2 #len(before_loss_twos)
```

```
200
```

```
1 torch.arange(0, len(X_train), batch_size)
```

```
tensor([  0, 500])
```

```
1 len(X_train)
```

```
700
```

```
1 optim
```

```
<module 'torch.optim' from '/usr/local/lib/python3.10/dist-packages/torch/optim/__init__.py'>
```

```
help(optim)
```

```
Help on package torch.optim in torch:

NAME
    torch.optim

DESCRIPTION
    :mod:`torch.optim` is a package implementing various optimization algorithms.
    Most commonly used methods are already supported, and the interface is general
    enough, so that more sophisticated ones can also be easily integrated in the
    future.

PACKAGE CONTENTS
    _functional
    _multi_tensor (package)
    adadelta
    adagrad
    adam
    adamax
    adamw
    asgd
    lbfgs
    lr_scheduler
    nadam
    optimizer
    radam
    rmsprop
    rprop
    sgd
    sparse_adam
    swa_utils

FILE
    /usr/local/lib/python3.10/dist-packages/torch/optim/__init__.py
```

```
exec('print(dim_list)')
```

```
[2, 16, 8, 1]
```

```
1
```

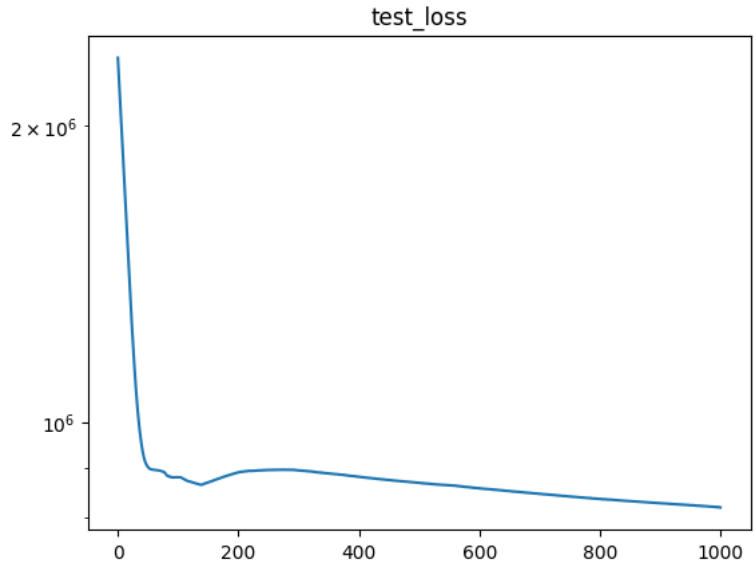
```
1
```

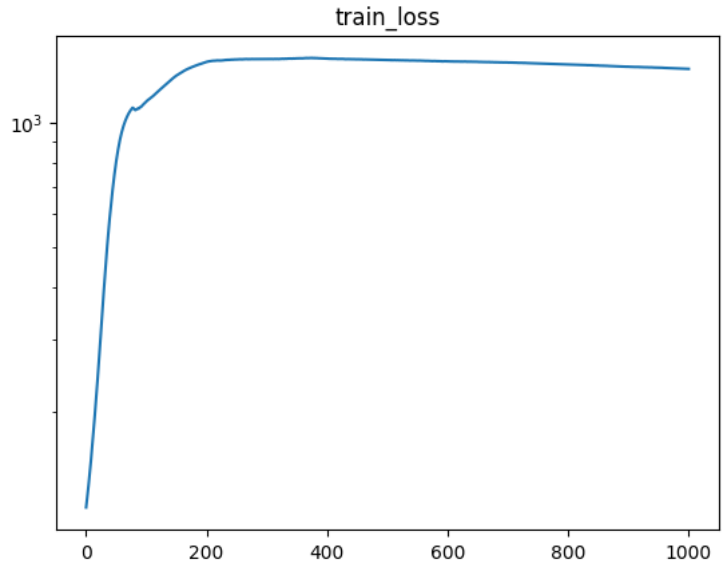
```
#default

"""
Bsize: 500 learningRate: 0.03 RMSE: 111.31 1000 loss: 515.4253401318177

"""
# for learningRate in range(10): #


dim_list = [2, 16, 8, 1]

#If data is less complex and is having fewer dimensions or features then neural networks with 1 to 2 hidden layers would work.
# If data is having large dimensions or features then to get an optimum solution, 3 to 5 hidden layers can be used.


# for Bsize in [500]:
for opt in [
            'Adadelta',
            'Adagrad',
            'Adam',
            'AdamW',
            'SparseAdam',
            'Adamax',
            'ASGD',
            'LBFGS',
            'NAdam',
            'RAdam',
            'RMSprop',
            'Rprop',
            'SGD'
            ]:
    if opt == 'Adam':# --------------delete later----------------------------------------------------------------------------------------------------------------------------------------------------------
        continue# --------------delete later----------------------------------------------------------------------------------------------------------------------------------------------------------
    for Bsize in [ 10, 50, 100, 500]:
        # print('Bsize:', Bsize, end =' ')
        #before_loss_twos =  torch.ones(Bsize, dtype=torch.float32)*2
        for learningRate in [ 1e-1, 5e-1, 1e-2, 5e-2, 1e-3, 5e-3, 1e-4, 5e-4, 1e-5, 5e-5, 1e-6, 5e-6, 1e-7, 5e-7 ]:
```

```
            # print('learningRate:', learningRate)
            # Define the model

            make_model = 'model = nn.Sequential('
            for layer_num in range( len(dim_list) ):
                if layer_num == len(dim_list) - 1:
                    make_model = make_model + 'nn.Linear(dim_list[' + str(dim_list[i]) + '], dim_list[' + str(dim_list[i+1]) + ']) )'
                else:
                    make_model = make_model + 'nn.Linear(dim_list[' + str(dim_list[i]) + '], dim_list[' + str(dim_list[i+1]) + ']), nn.LeakyReLU(

            #make_model = make_model +

            # model = nn.Sequential(
            #     nn.Linear(dim_list[0], dim_list[1]),
            #     nn.LeakyReLU(),
            #     nn.Linear(dim_list[1], dim_list[2]),
            #     nn.LeakyReLU(),
            #     nn.Linear(dim_list[2], dim_list[3])
            #     # nn.Linear(2, 3),
            #     # nn.LeakyReLU(),
            #     # nn.Linear(3, 2),
            #     # nn.LeakyReLU(),
            #     # nn.Linear(2, 1)
            # )

            # loss function and optimizer
            loss_fn = nn.MSELoss()  # mean square error
            # optimizer = optim.Adam(model.parameters(), lr=learningRate  )  # 10 loss: 5157042688.0
            exec('optimizer = optim.' + opt + '(model.parameters(), lr=learningRate  )')
            n_epochs = 1000+1   # number of epochs to run
            batch_size = Bsize  # size of each batch
            batch_start = torch.arange(0, len(X_train), batch_size)

            # Hold the best model
            best_mse = np.inf   # init to infinity
            best_weights = None
            history = []
            history_train = []
            for epoch in range(n_epochs):
                model.train()
                with tqdm.tqdm(batch_start, unit="batch", mininterval=0, disable=True) as bar:
                    bar.set_description(f"Epoch {epoch}")
                    for start in bar:
                        # take a batch
                        X_batch = X_train[start:start+batch_size]
                        y_batch = y_train[start:start+batch_size]
                        # forward pass
                        y_pred = model(X_batch)

                        before_loss = y_pred/y_batch + y_batch/y_pred
                        before_loss_twos =  torch.ones(len(y_pred), dtype=torch.float32)*2


                        loss = loss_fn(before_loss, before_loss_twos)
                        # backward pass
                        optimizer.zero_grad()
                        loss.backward()
                        # update weights
                        optimizer.step()
                        # print progress
                        bar.set_postfix(mse=float(loss))

                loss = float(loss)

                history_train.append(loss)
                # evaluate accuracy at end of each epoch
                model.eval()
                y_pred = model(X_test)

                before_loss = y_pred/y_test + y_test/y_pred
                before_loss_twos =  torch.ones(len(y_pred), dtype=torch.float32)*2
                mse = loss_fn(before_loss, before_loss_twos)

                mse = float(mse)

                history.append(mse)
                if mse < best_mse:
                    best_mse = mse
                    best_weights = copy.deepcopy(model.state_dict())

                # if epoch % 100 == 0:
                #     print('epoch: %5d' % epoch, 'test_loss: %5d' % np.sqrt(mse), 'train_loss: %5d' % np.sqrt(loss))
```

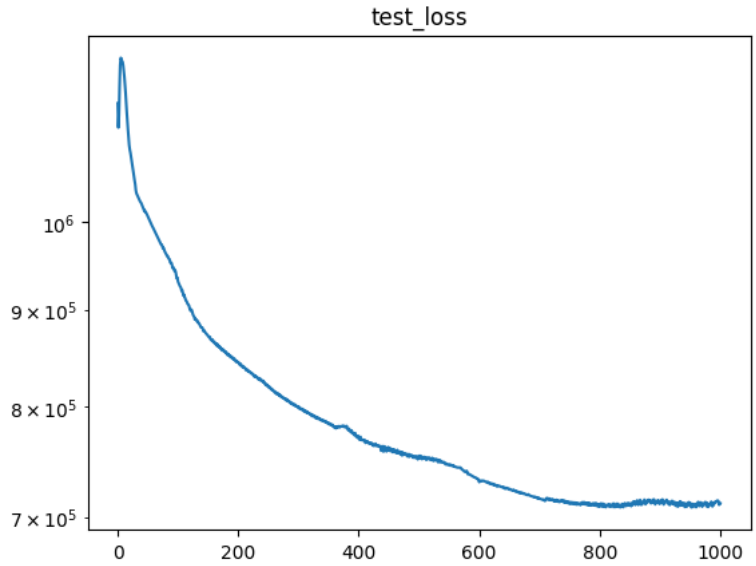
```
            # restore model and return best accuracy
            model.load_state_dict(best_weights)
            # print("MSE: %.2f" % best_mse)
            # print("RMSE: %.2f" % np.sqrt(best_mse))
            print('optimizer:', opt, ', n_of_data:',N, ', Bsize:', Bsize ,', learningRate:', learningRate , ", minimum_RMSE: %.2f" % np.sqrt(best
            # print(y_pred[:10])
            # print(y_test[:10])
            plt.plot(history)
            plt.yscale('log')
            plt.title('test_loss')
            plt.show()

            plt.plot(history_train)
            plt.title('train_loss')
            plt.yscale('log')
            plt.show()
print('all done')
# ------------------------------------------------------------------------------------------------------------
# ------------------------------------------------------------------------------------------------------------
```

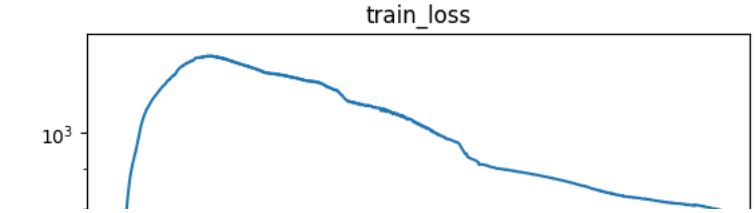
```
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/loss.py:536: UserWarning: Using a target size (torch.Size([10])) that is different t
  return F.mse_loss(input, target, reduction=self.reduction)
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/loss.py:536: UserWarning: Using a target size (torch.Size([300])) that is different t
  return F.mse_loss(input, target, reduction=self.reduction)
optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 0.1 , minimum_RMSE: 905.62 , epoch:  1000 , test_loss:   905 , train_loss:
```

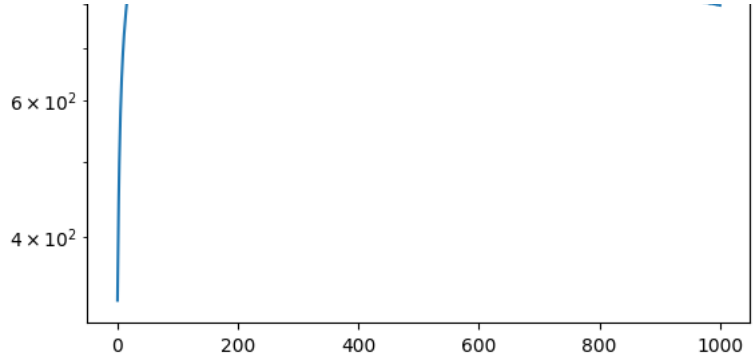




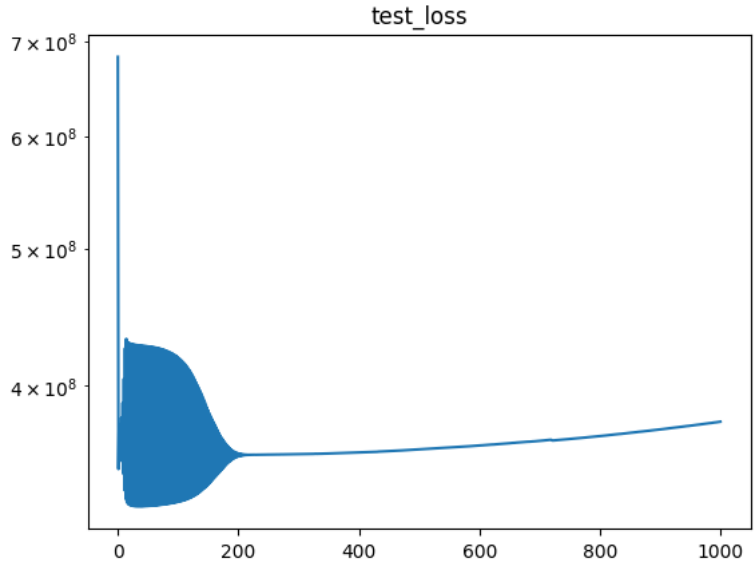

```
optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 0.5 , minimum_RMSE: 841.62 , epoch:  1000 , test_loss:   843 , train_loss:
```

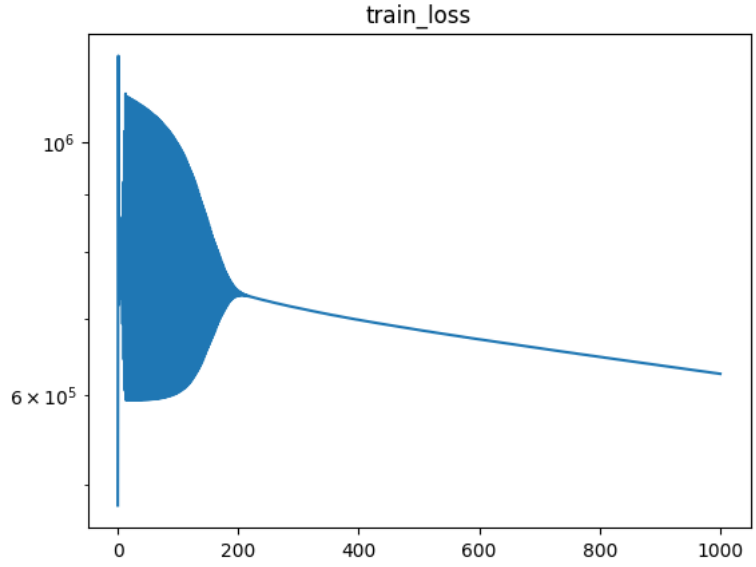




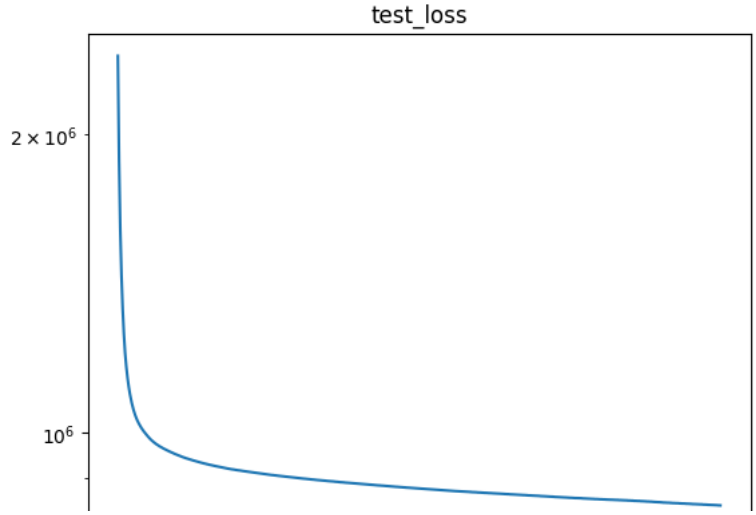


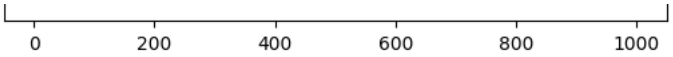
optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 0.01 , minimum_RMSE: 18135.77 , epoch: 1000 , test_loss: 19426 , train_loss

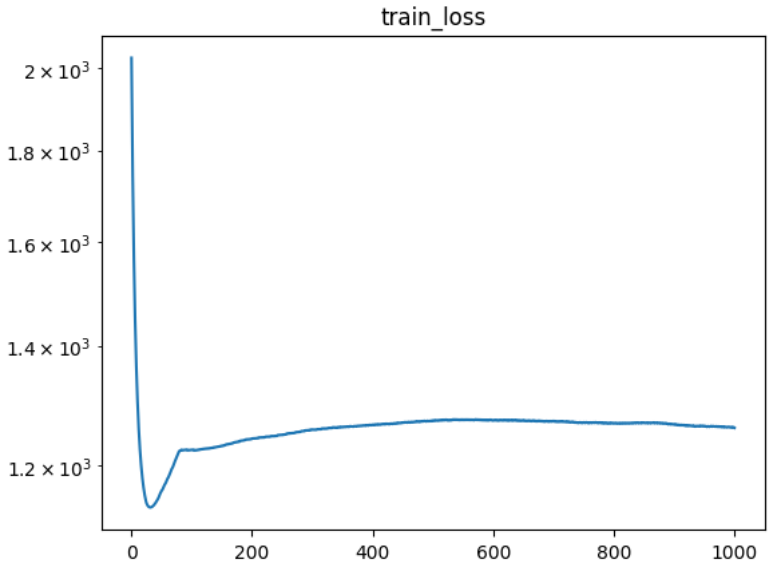




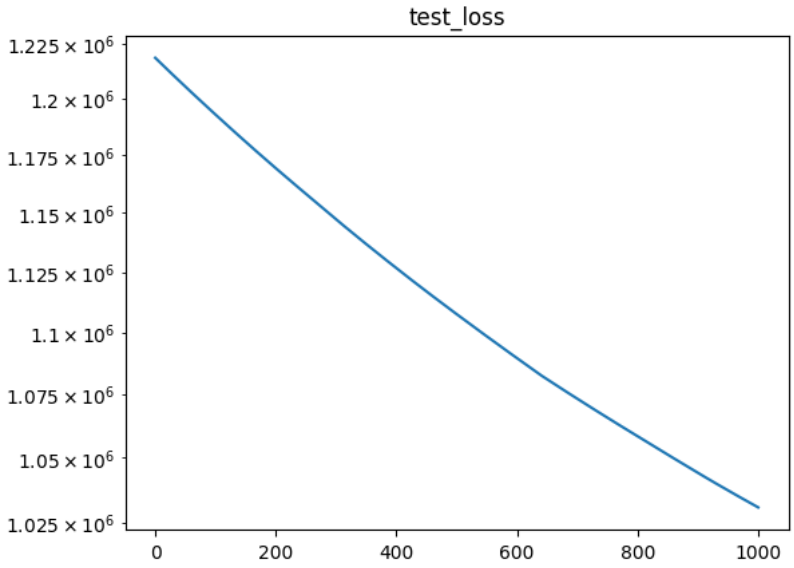
optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 0.05 , minimum_RMSE: 919.39 , epoch: 1000 , test_loss: 919 , train_loss:

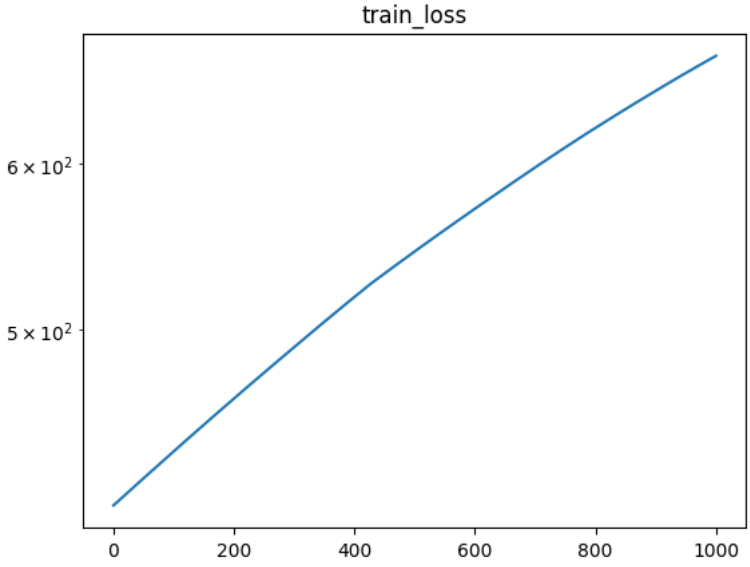




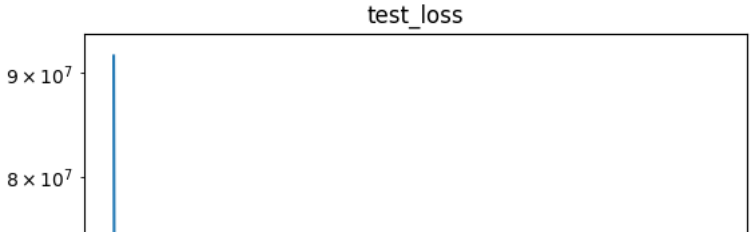


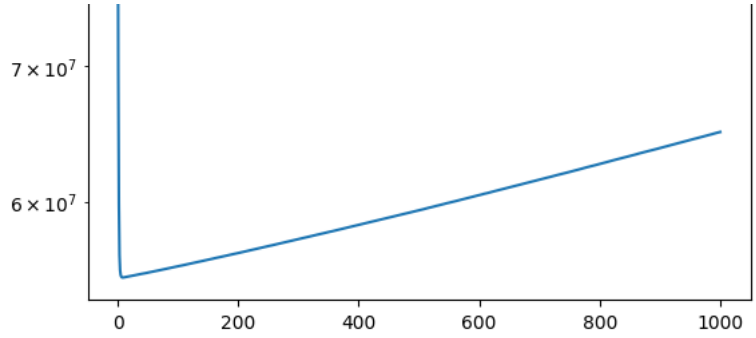
optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 0.001 , minimum_RMSE: 1015.24 , epoch: 1000 , test_loss: 1015 , train_loss

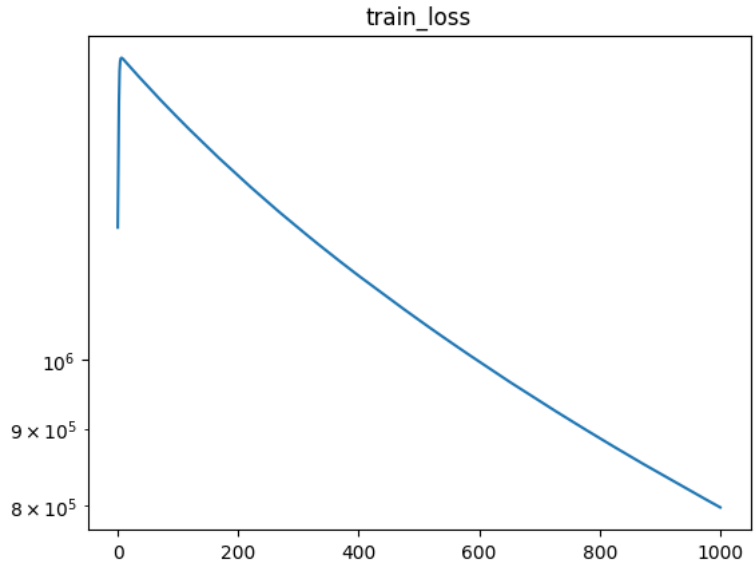




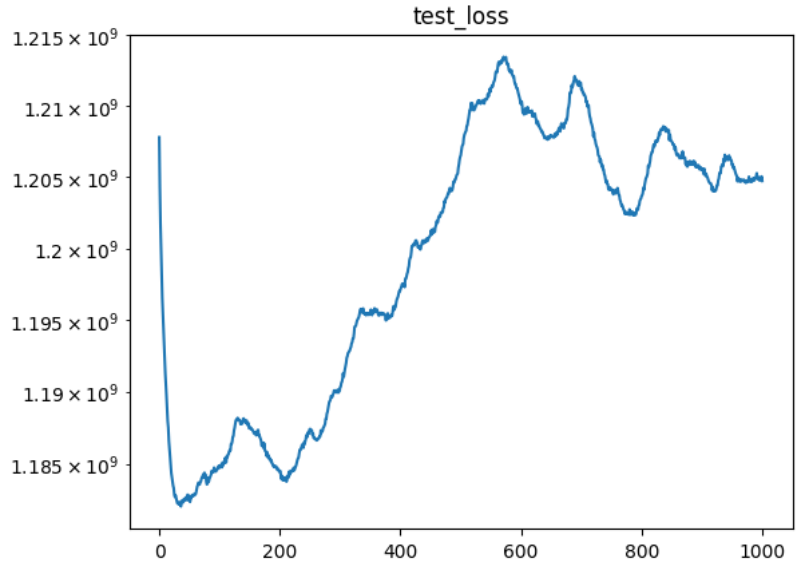
optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 0.005 , minimum_RMSE: 7418.12 , epoch: 1000 , test_loss: 8059 , train_loss

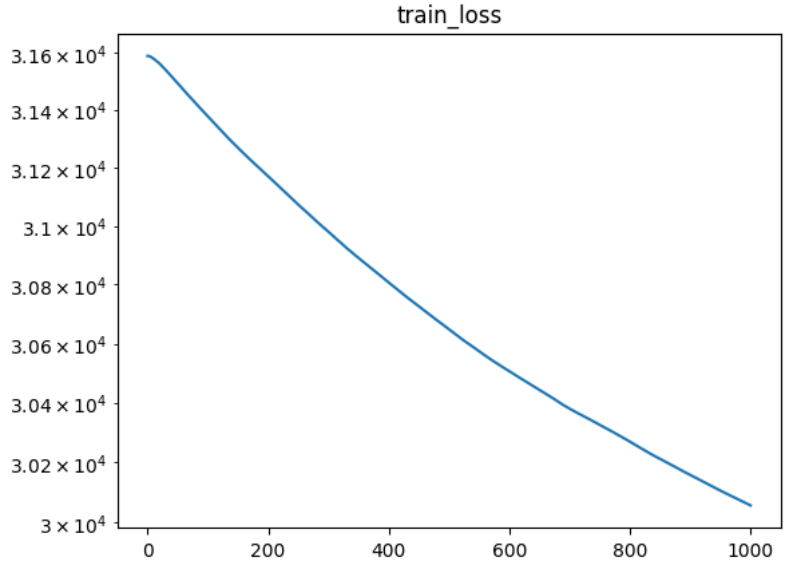




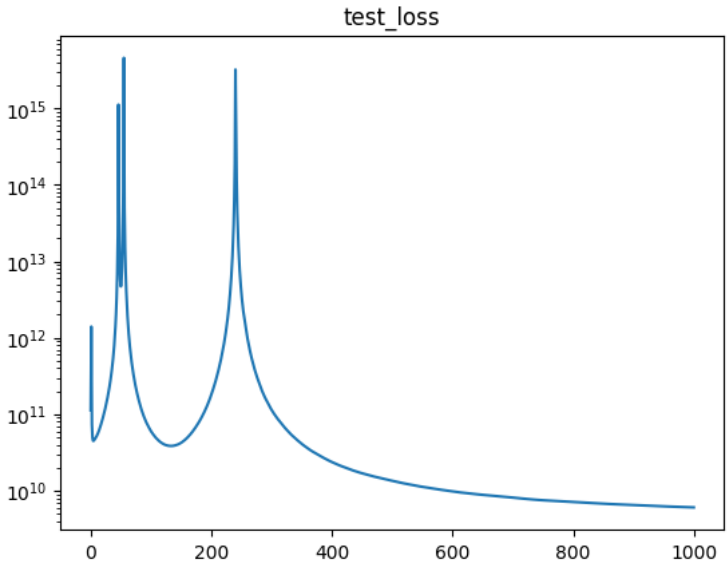


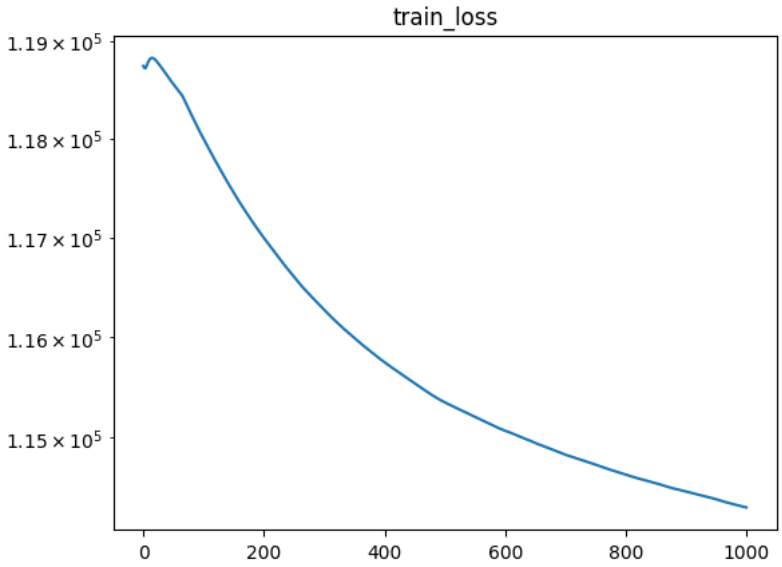
optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 0.0001 , minimum_RMSE: 34381.68 , epoch: 1000 , test_loss: 34709 , train_l



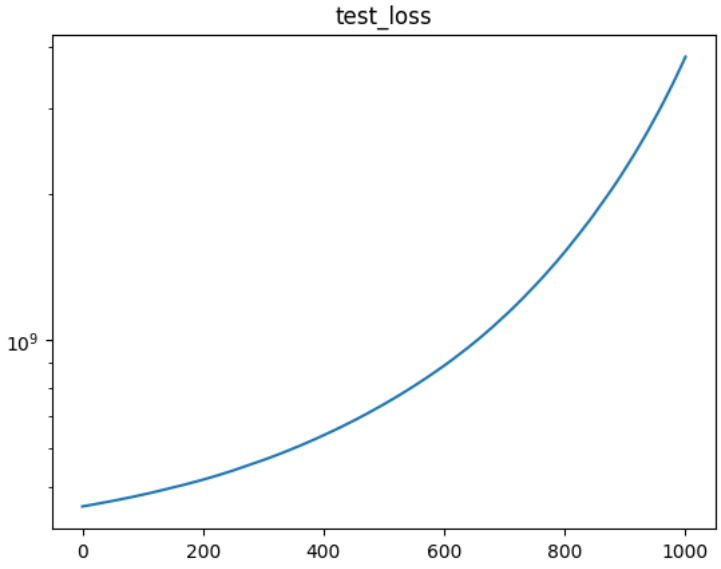


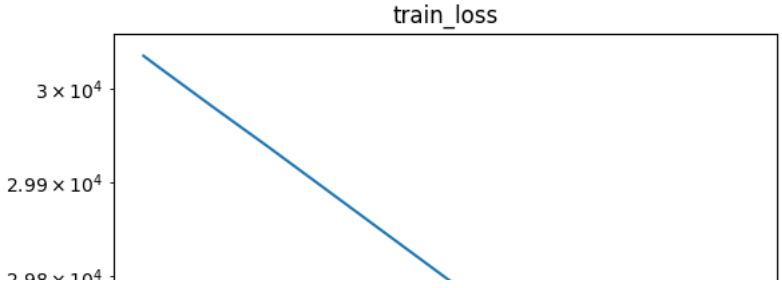
```
optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 0.0005 , minimum_RMSE: 77847.40 , epoch:  1000 , test_loss: 77847 , train_lo
```

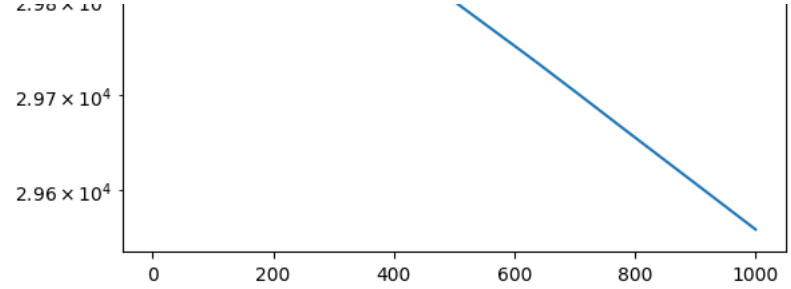




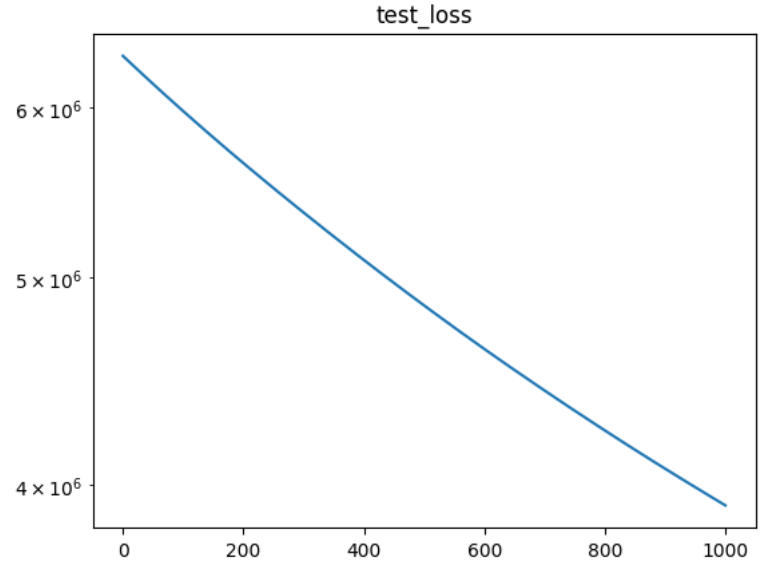

```
optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-05 , minimum_RMSE: 21361.47 , epoch:  1000 , test_loss: 61851 , train_los
```

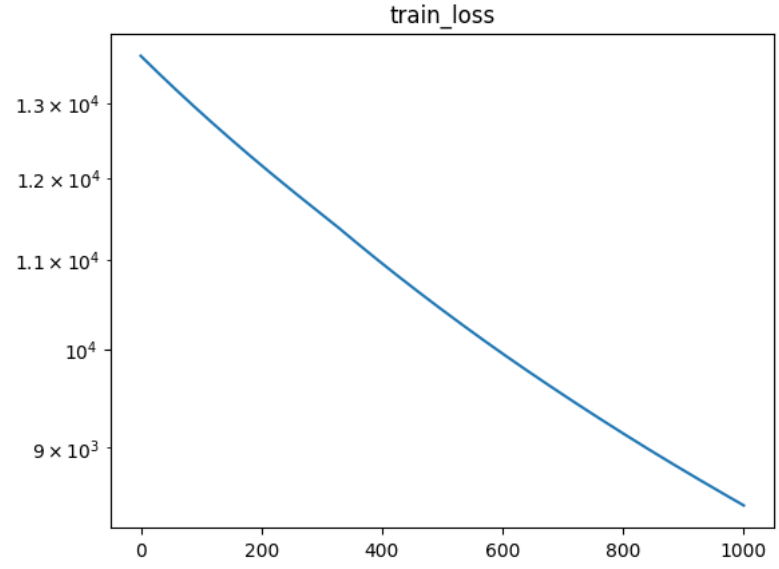




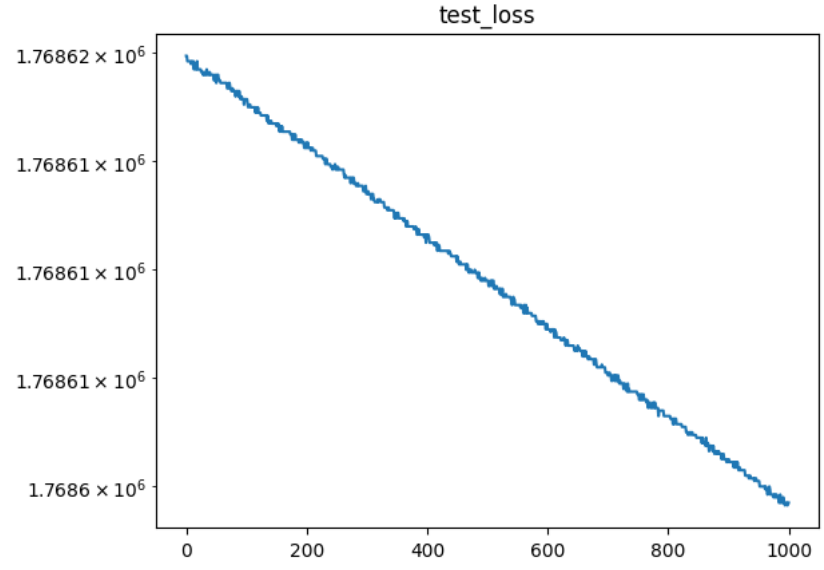


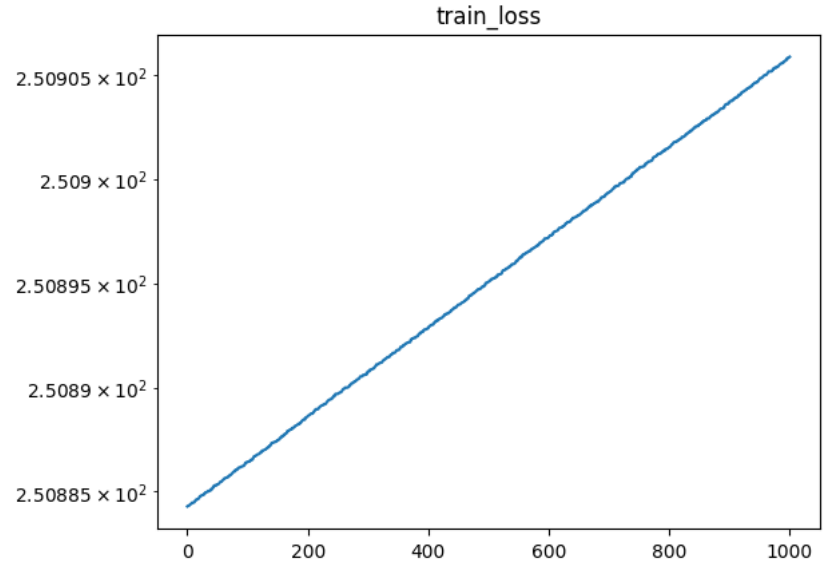
optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 5e-05 , minimum_RMSE: 1978.10 , epoch: 1000 , test_loss: 1978 , train_loss



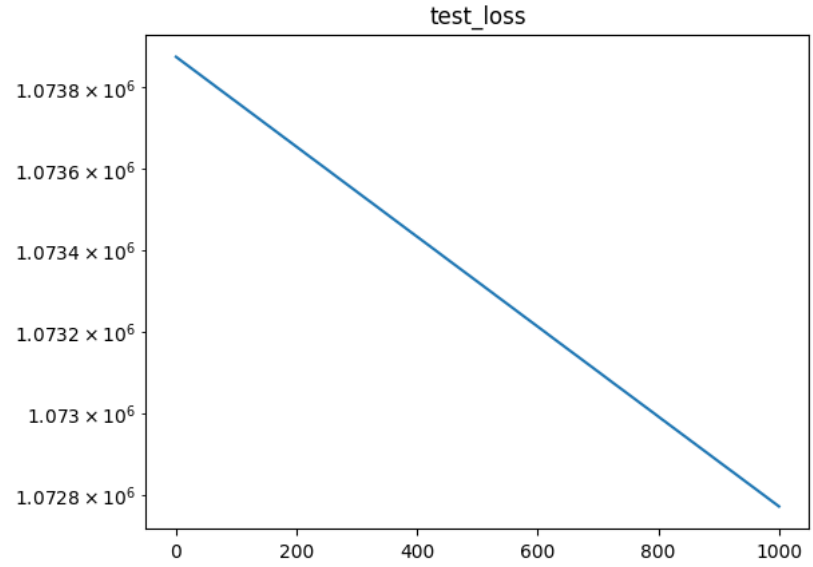


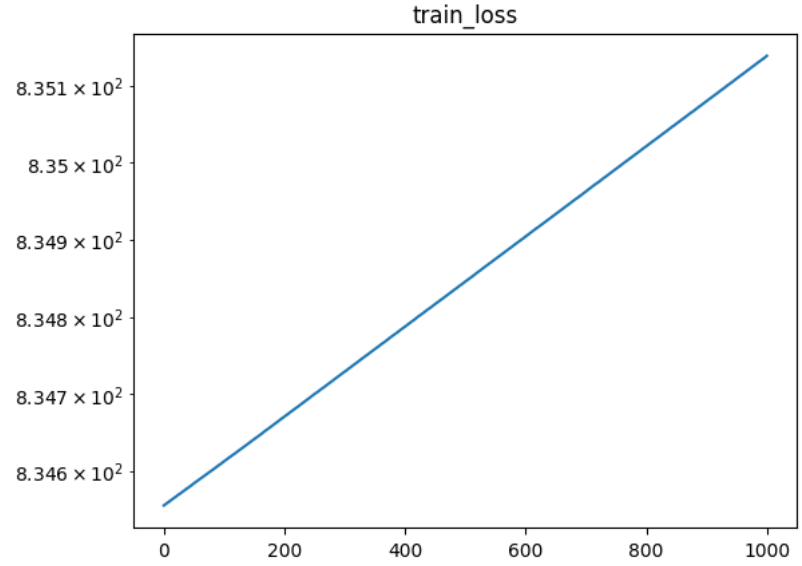
optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-06 , minimum_RMSE: 1329.89 , epoch: 1000 , test_loss: 1329 , train_loss



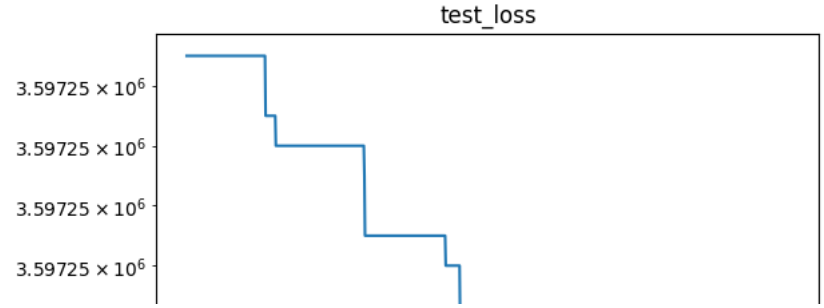


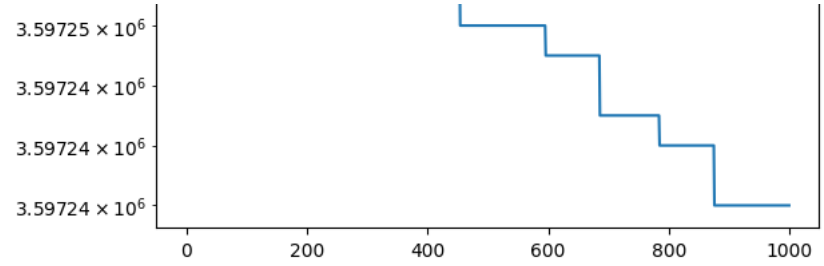
optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 5e-06 , minimum_RMSE: 1035.75 , epoch: 1000 , test_loss: 1035 , train_loss

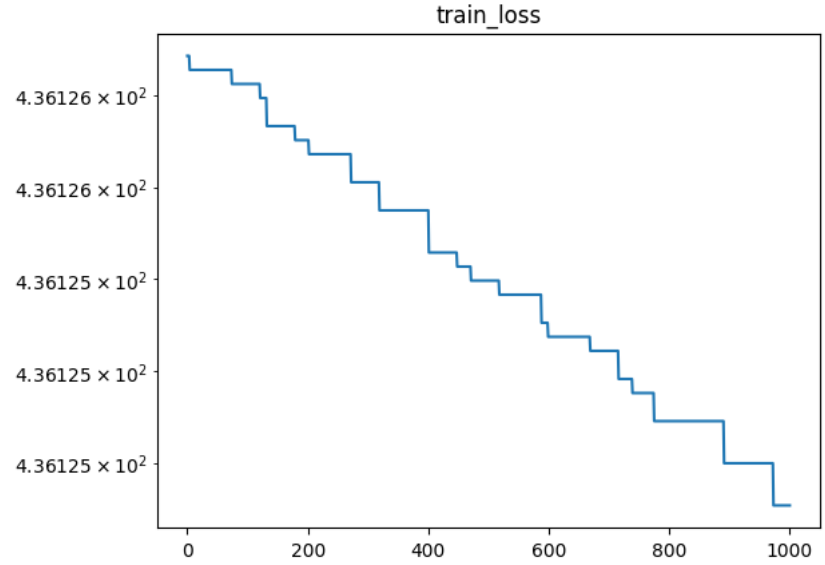




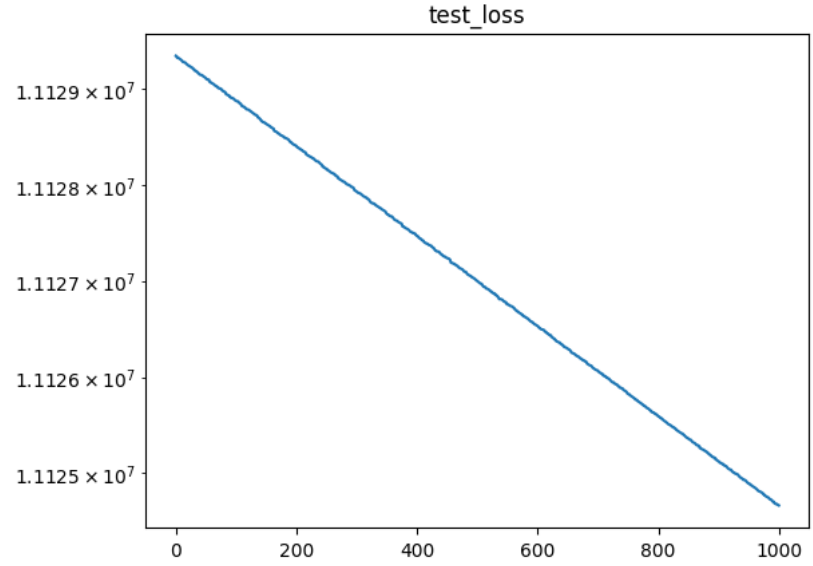
optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-07 , minimum_RMSE: 1896.64 , epoch: 1000 , test_loss: 1896 , train_loss

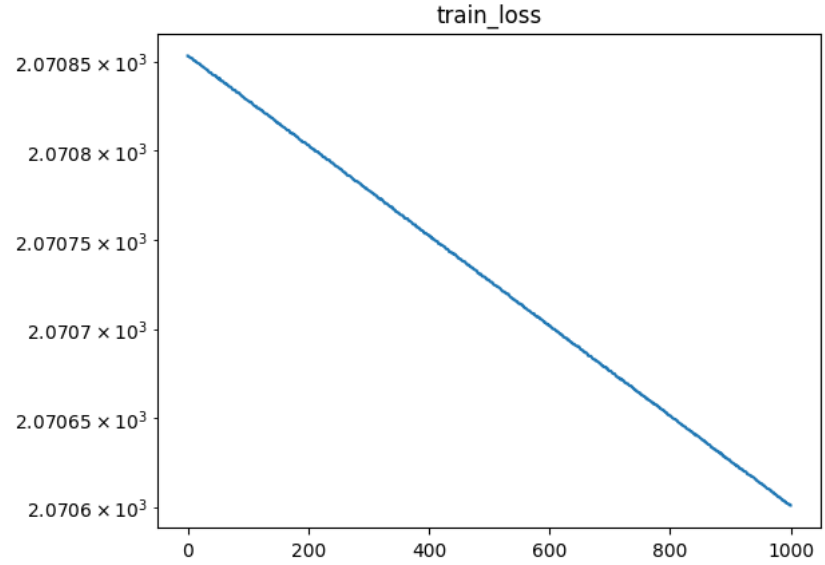






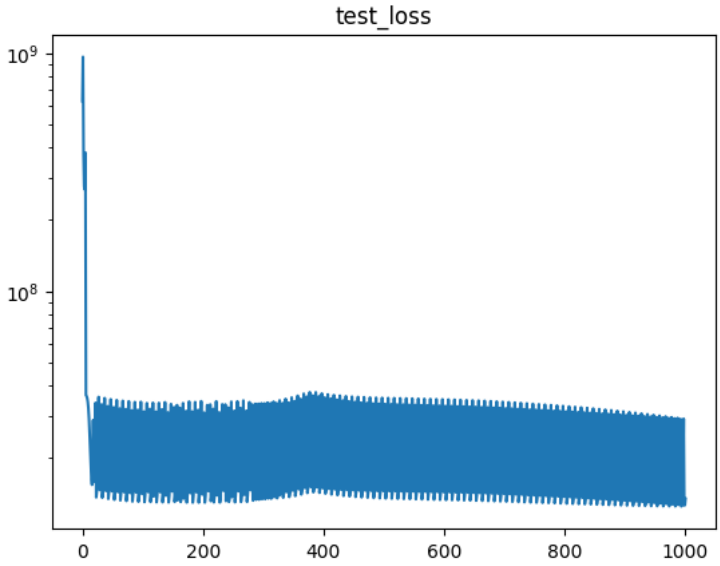
optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 5e-07 , minimum_RMSE: 3335.37 , epoch: 1000 , test_loss: 3335 , train_loss

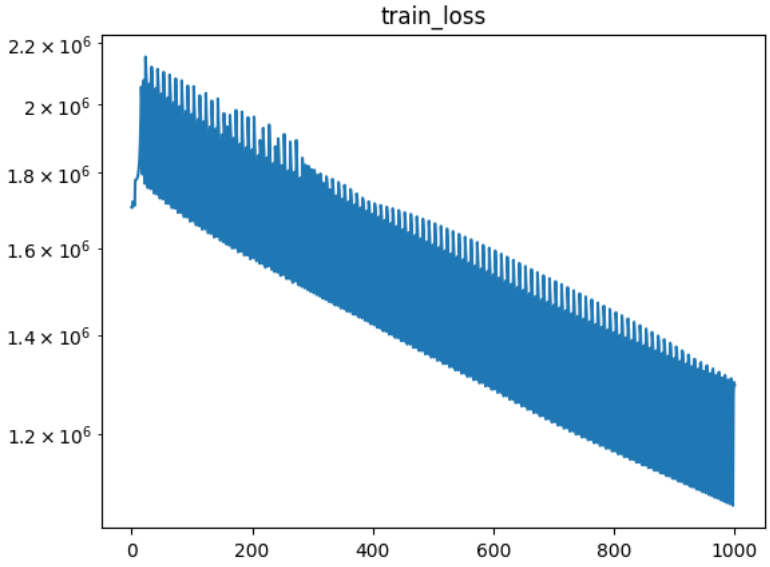




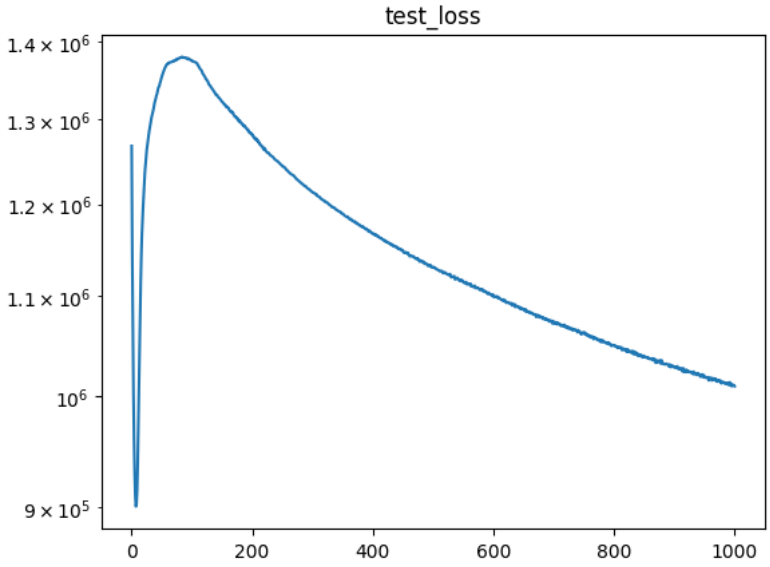
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/loss.py:536: UserWarning: Using a target size (torch.Size([50])) that is different to
return F.mse_loss(input, target, reduction=self.reduction)

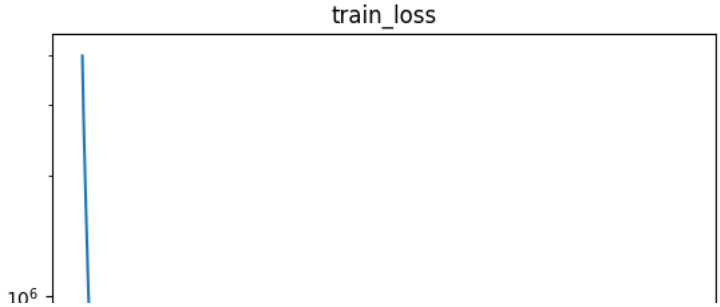
optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 0.1 , minimum_RMSE: 3529.56 , epoch: 1000 , test_loss: 3663 , train_loss:

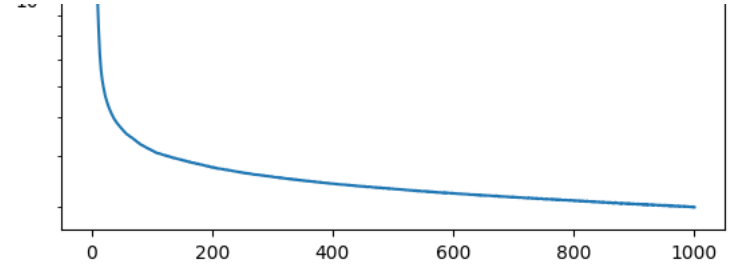




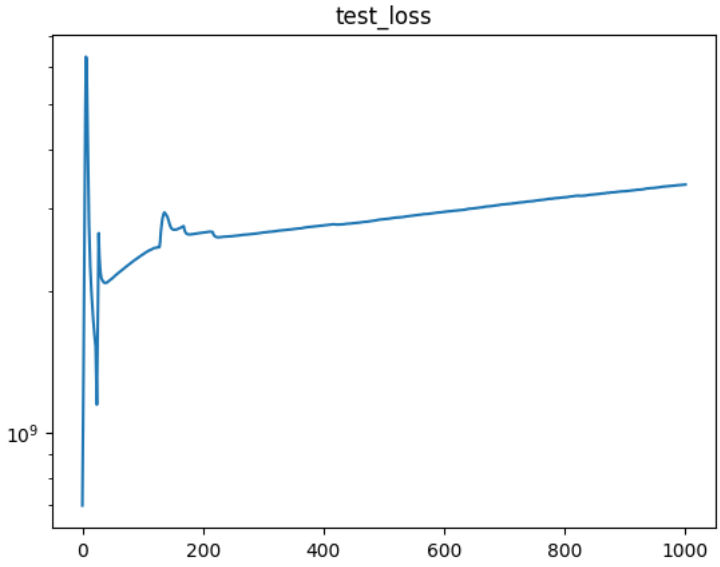
optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 0.5 , minimum_RMSE: 949.11 , epoch: 1000 , test_loss: 1004 , train_loss:

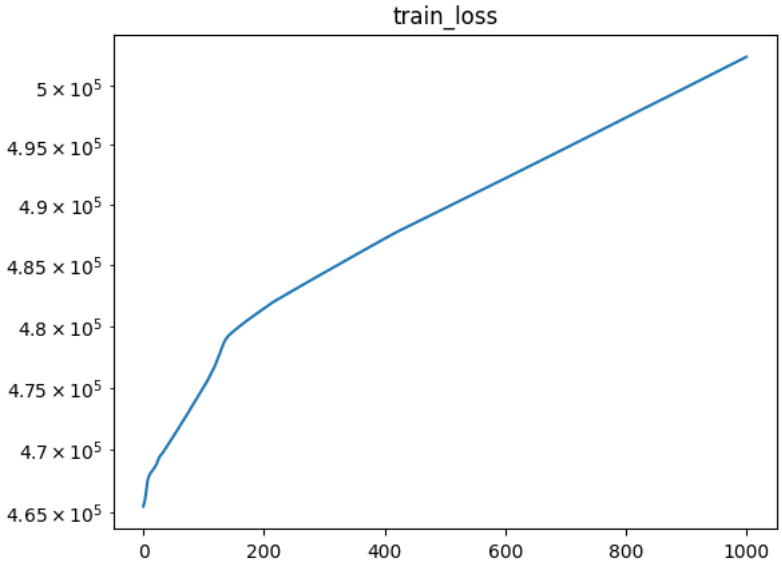




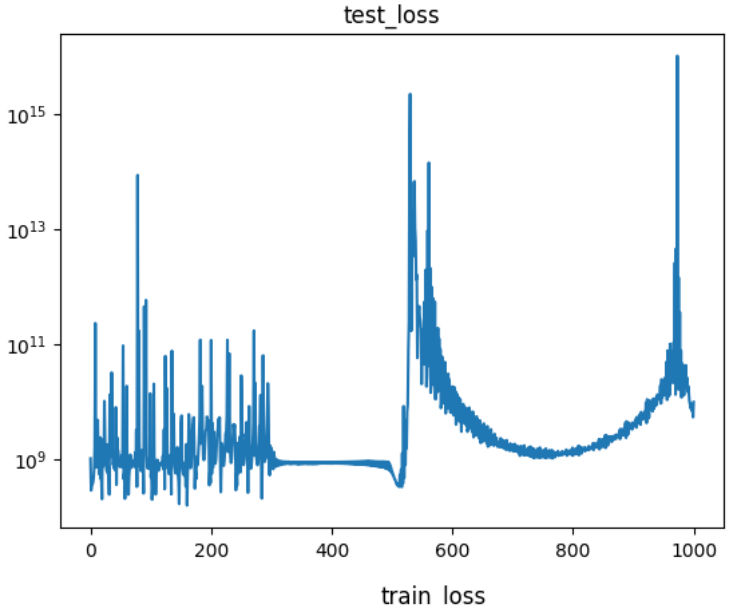


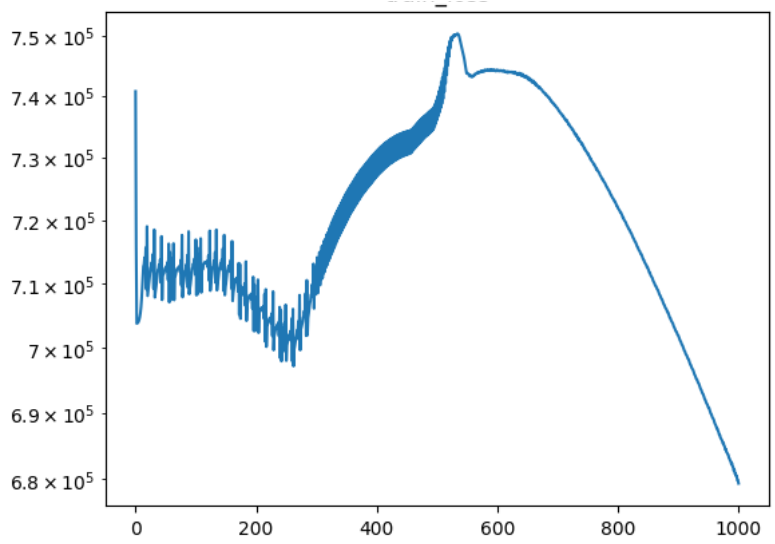
optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 0.01 , minimum_RMSE: 26443.31 , epoch: 1000 , test_loss: 58099 , train_loss



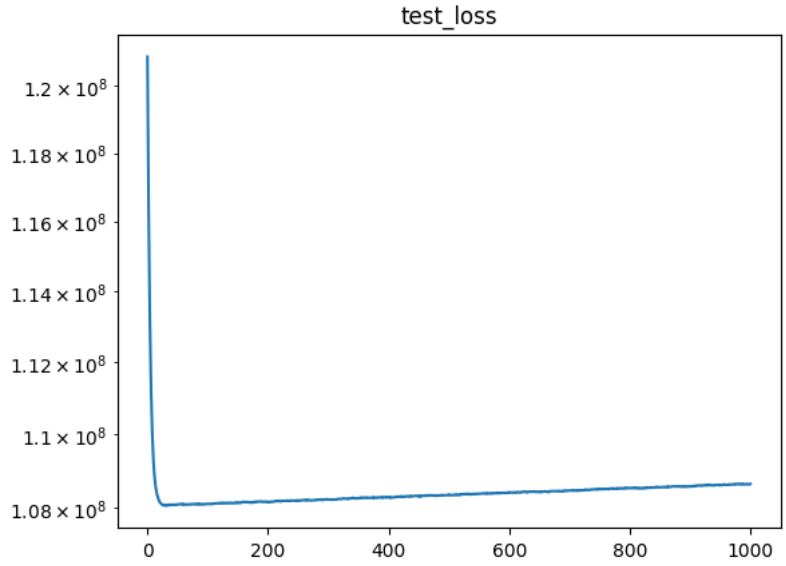


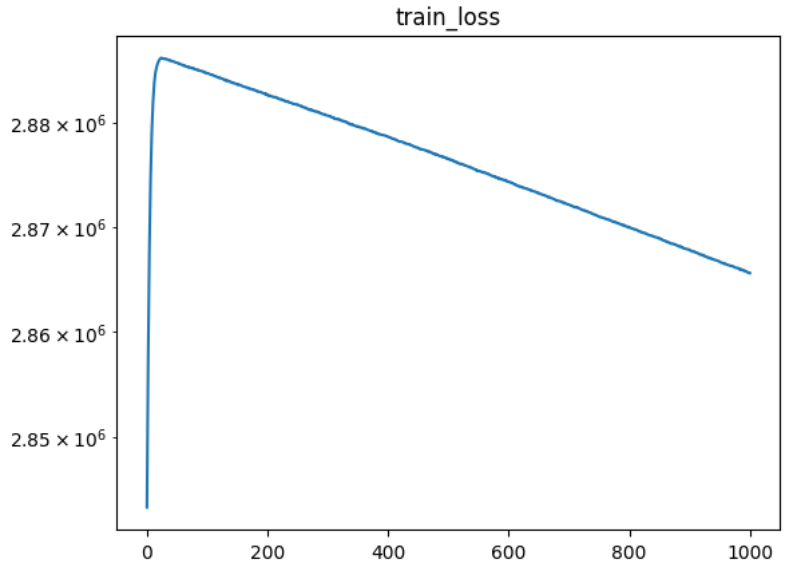
optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 0.05 , minimum_RMSE: 12397.49 , epoch: 1000 , test_loss: 98089 , train_loss



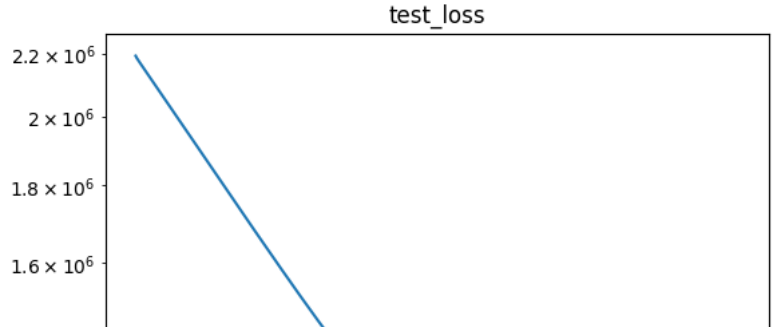


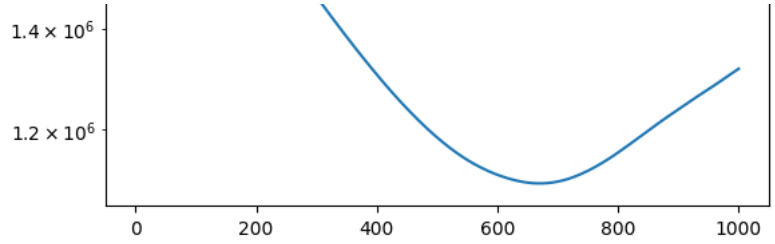
optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 0.001 , minimum_RMSE: 10395.05 , epoch: 1000 , test_loss: 10423 , train_los

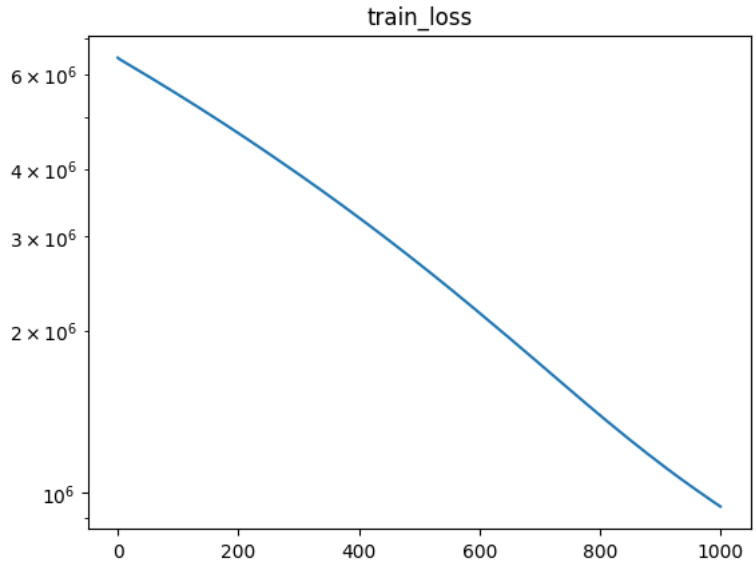




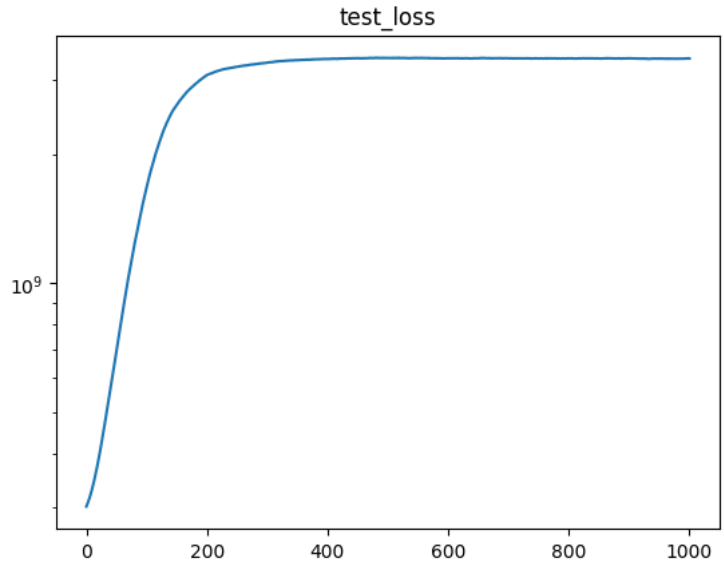
optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 0.005 , minimum_RMSE: 1051.56 , epoch: 1000 , test_loss: 1147 , train_loss

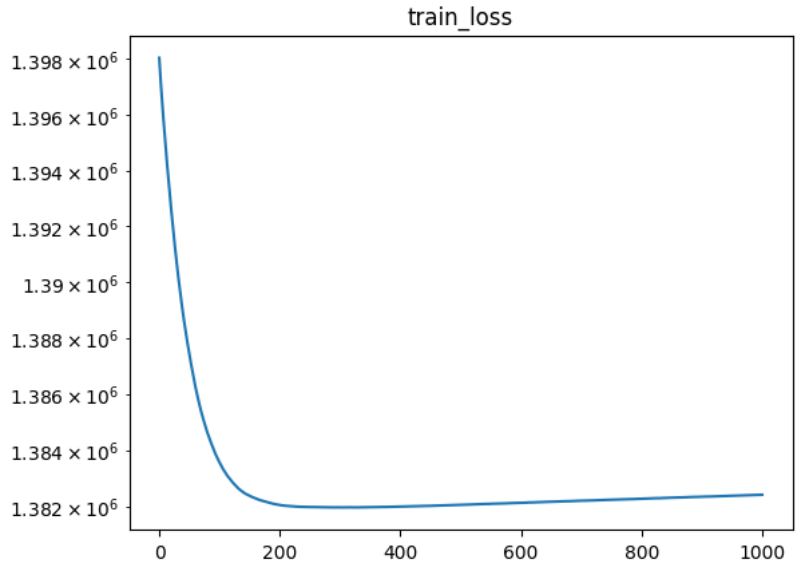






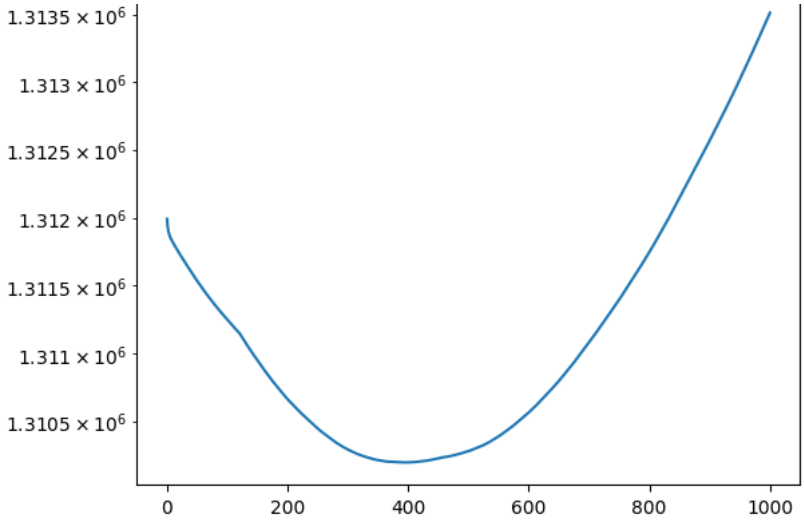
optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 0.0001 , minimum_RMSE: 17334.21 , epoch: 1000 , test_loss: 57912 , train_l

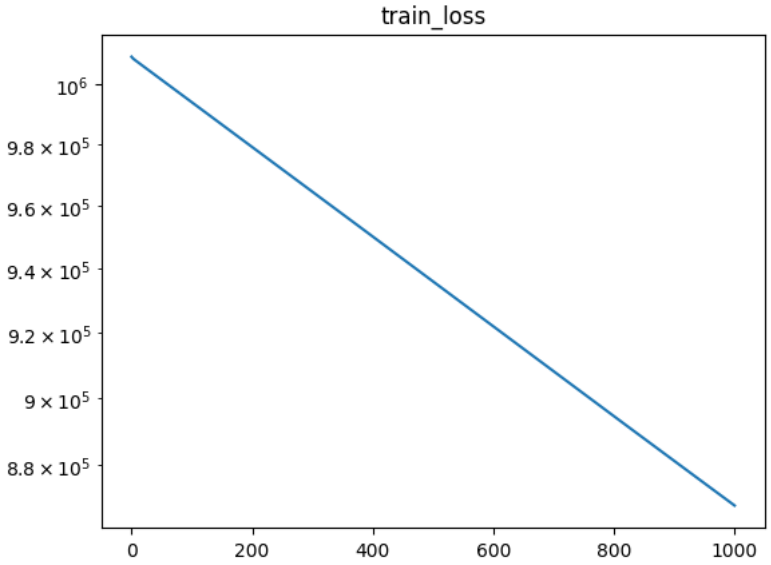




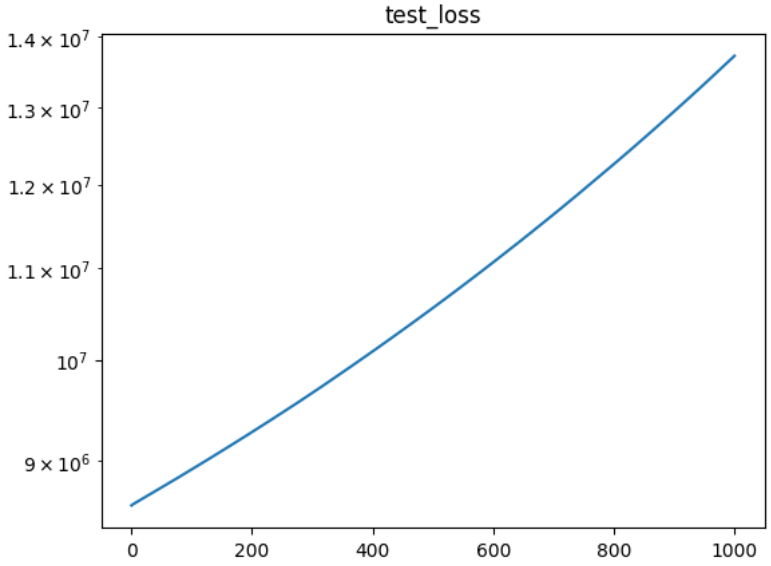
optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 0.0005 , minimum_RMSE: 1144.64 , epoch: 1000 , test_loss: 1146 , train_los

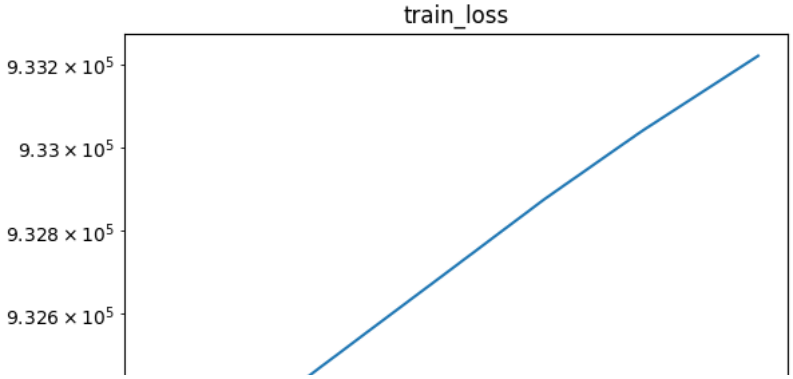
test_loss

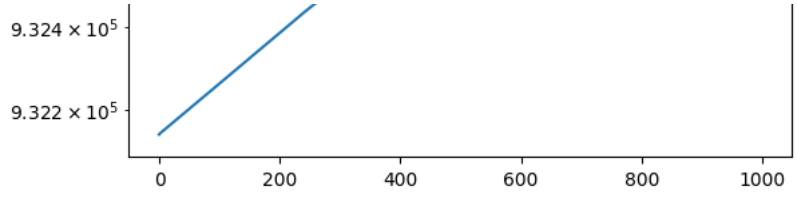




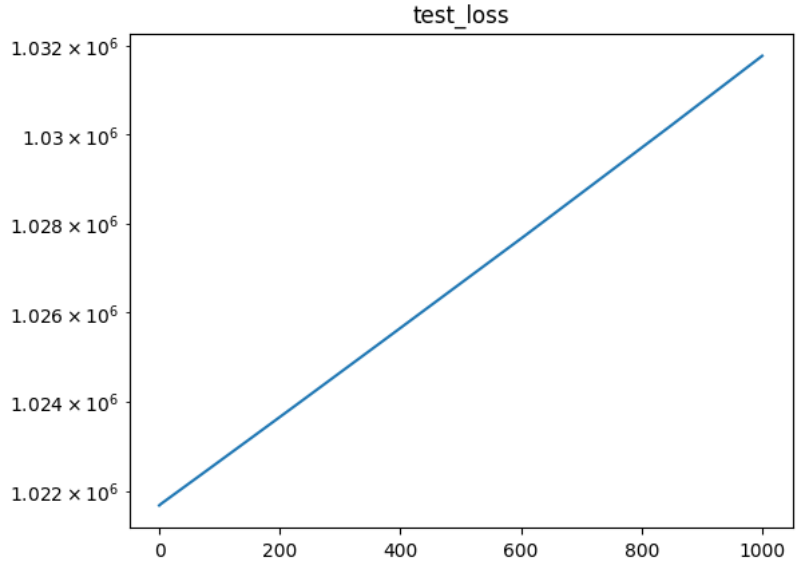
optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-05 , minimum_RMSE: 2931.33 , epoch: 1000 , test_loss: 3704 , train_loss

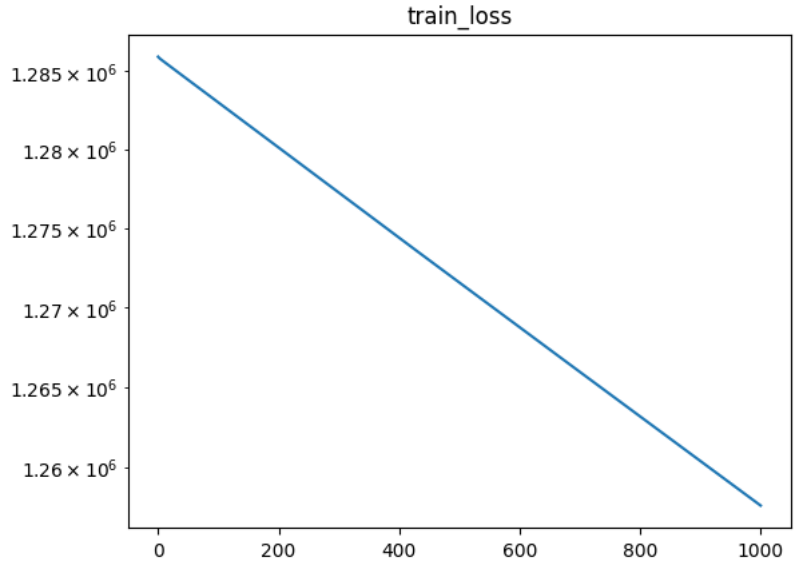




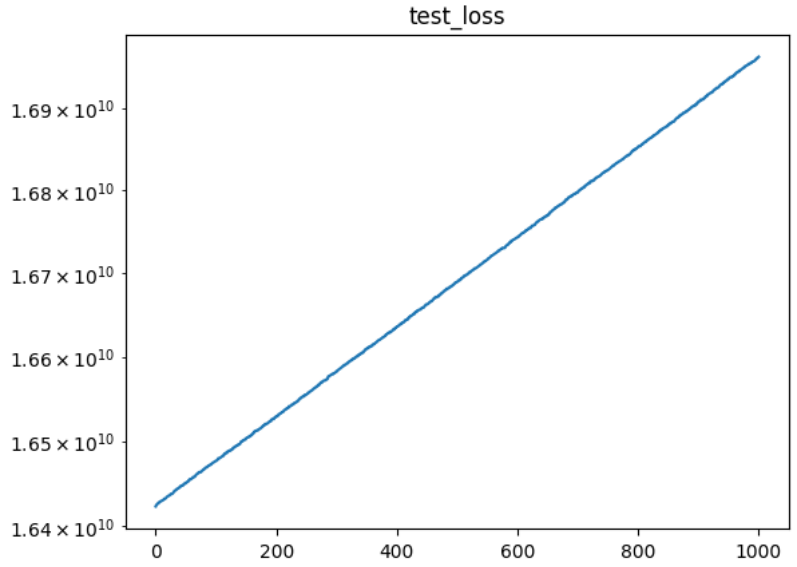


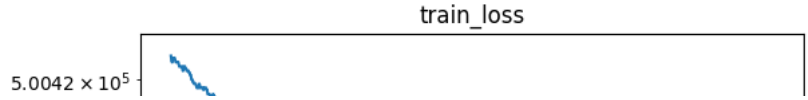
optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 5e-05 , minimum_RMSE: 1010.78 , epoch: 1000 , test_loss: 1015 , train_loss

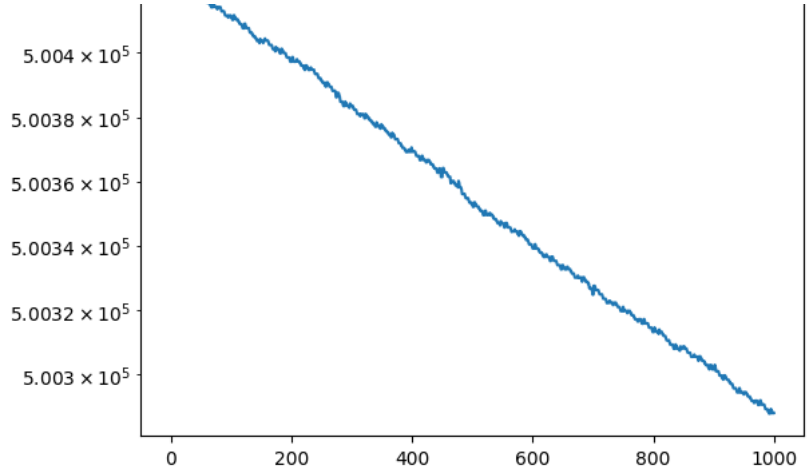




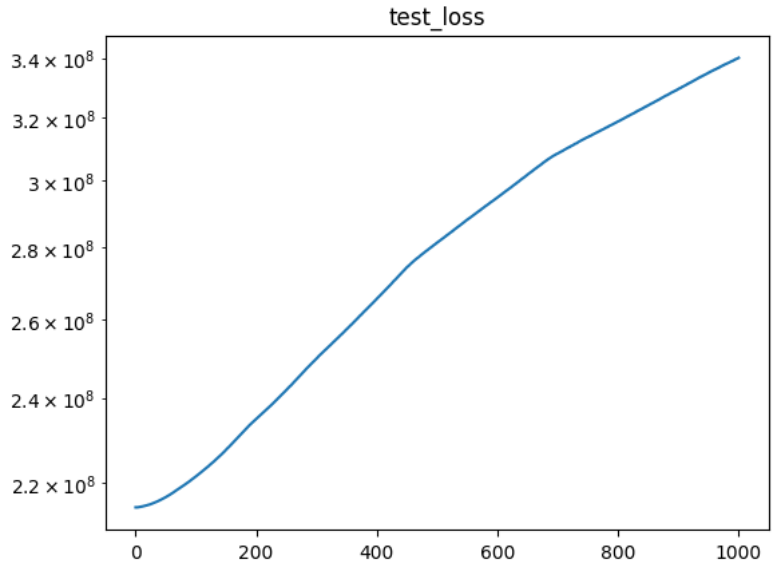
optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-06 , minimum_RMSE: 128152.29 , epoch: 1000 , test_loss: 130238 , train_

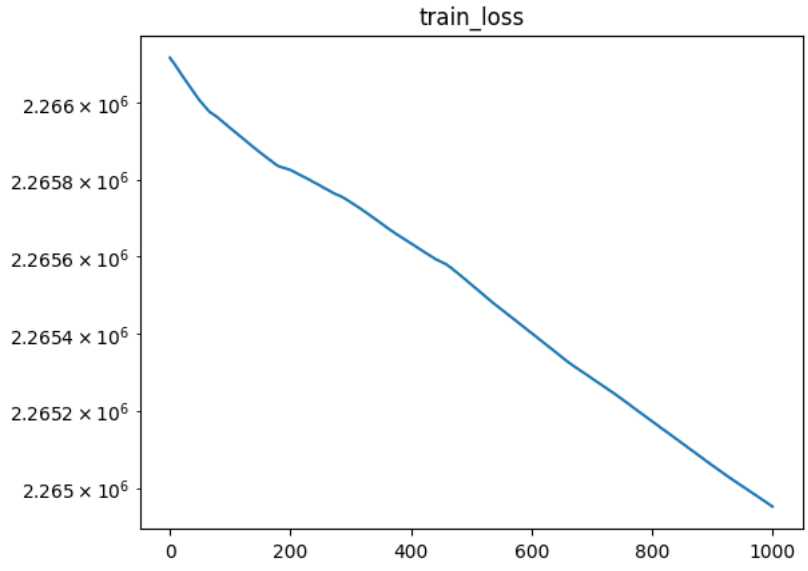




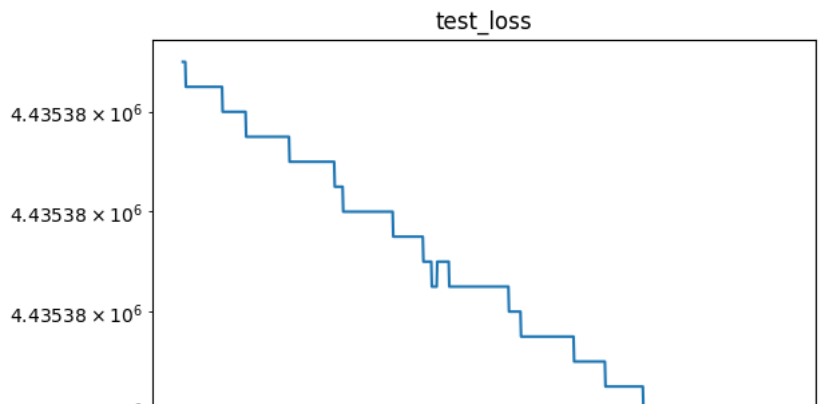


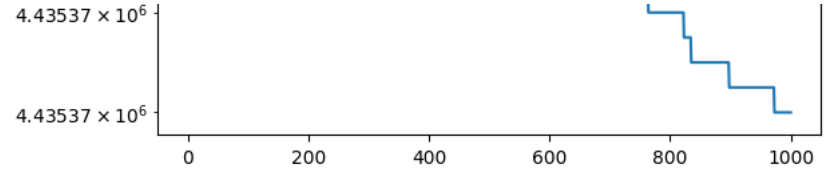
optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 5e-06 , minimum_RMSE: 14647.91 , epoch: 1000 , test_loss: 18445 , train_los

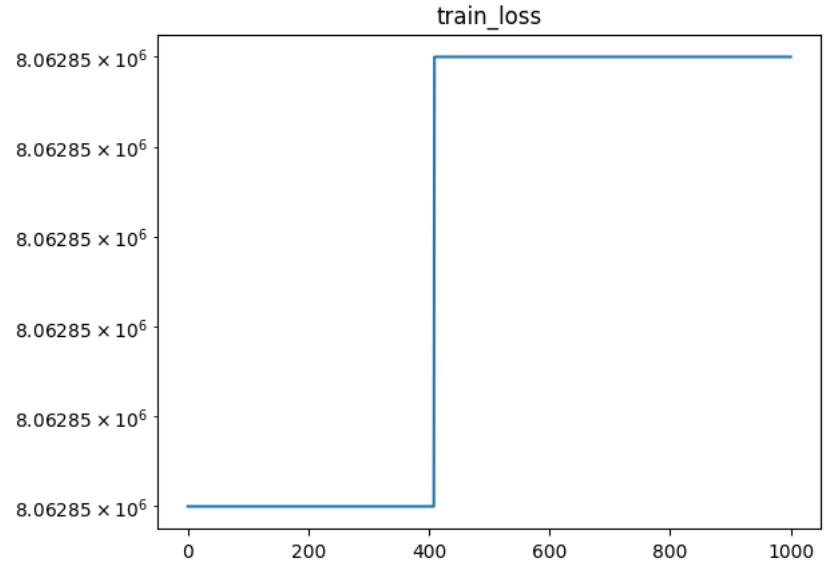




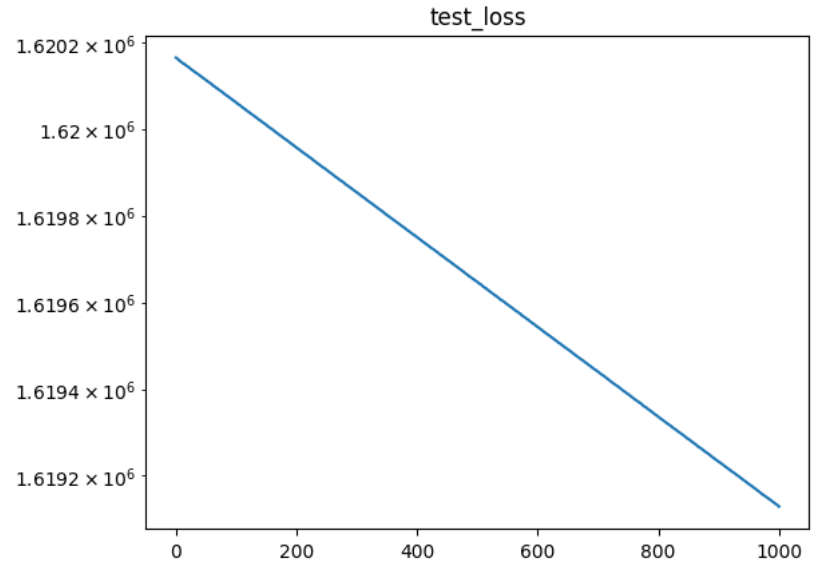
optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-07 , minimum_RMSE: 2106.03 , epoch: 1000 , test_loss: 2106 , train_loss

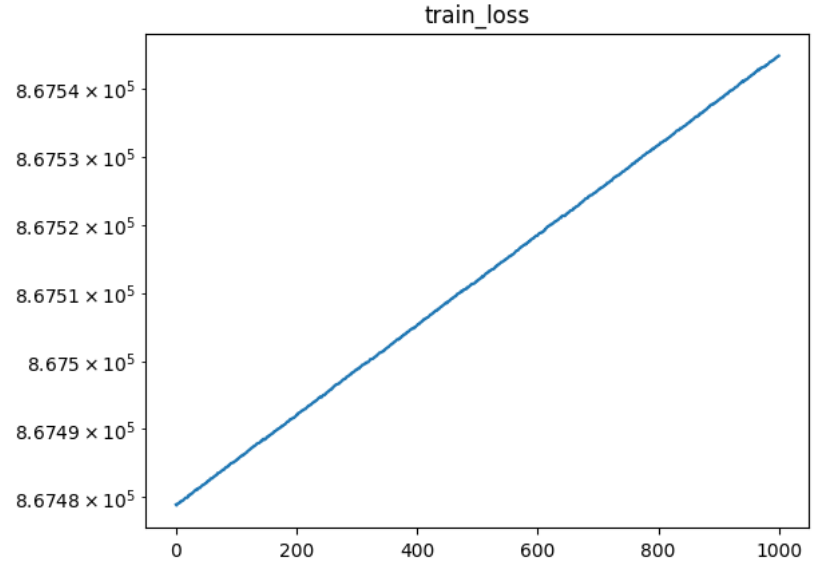




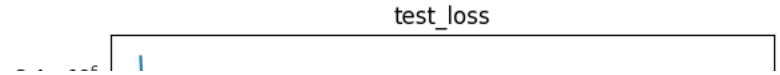


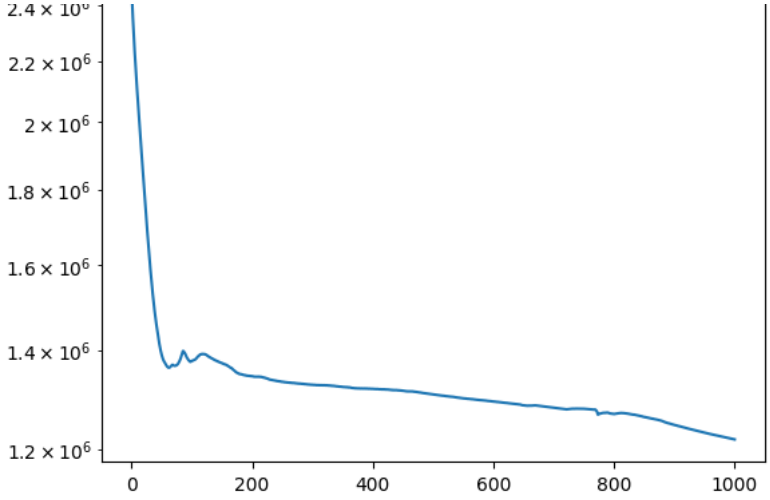

```
optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 5e-07 , minimum_RMSE: 1272.45 , epoch:  1000 , test_loss:  1272 , train_loss
```

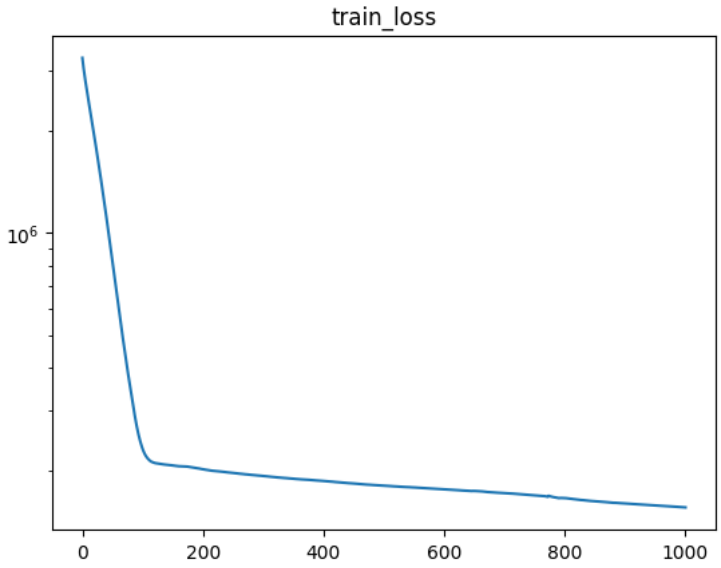




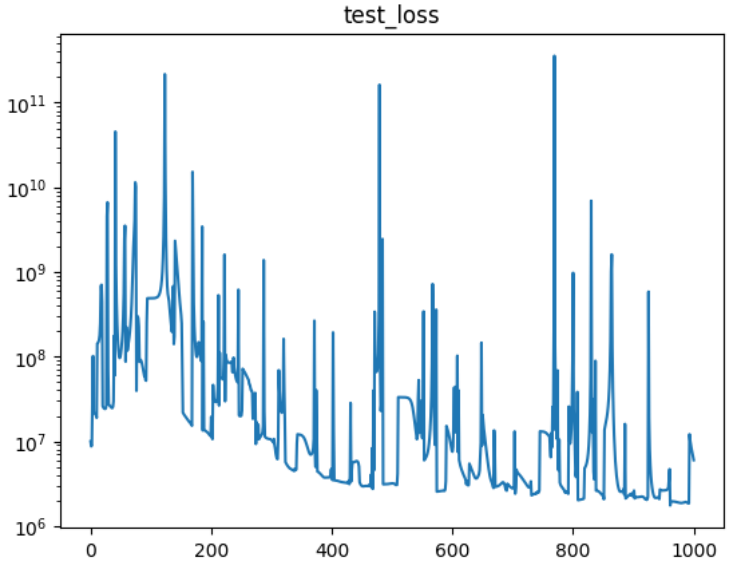

```
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/loss.py:536: UserWarning: Using a target size (torch.Size([100])) that is different t
  return F.mse_loss(input, target, reduction=self.reduction)
optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 0.1 , minimum_RMSE: 1103.77 , epoch:  1000 , test_loss:  1103 , train_loss
```

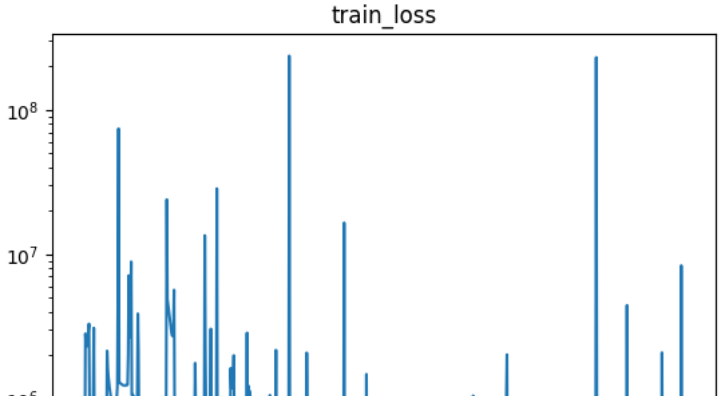


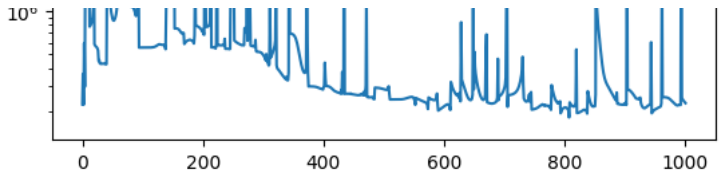




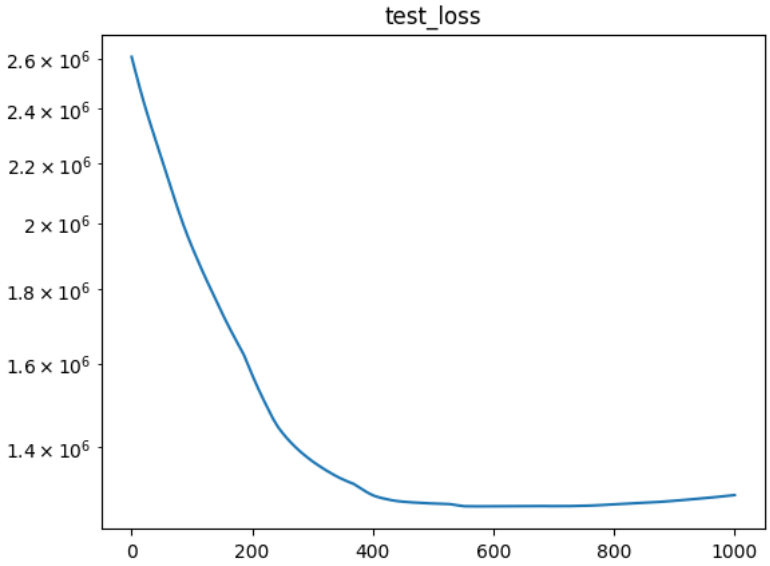
optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 0.5 , minimum_RMSE: 1323.23 , epoch: 1000 , test_loss: 2450 , train_loss:

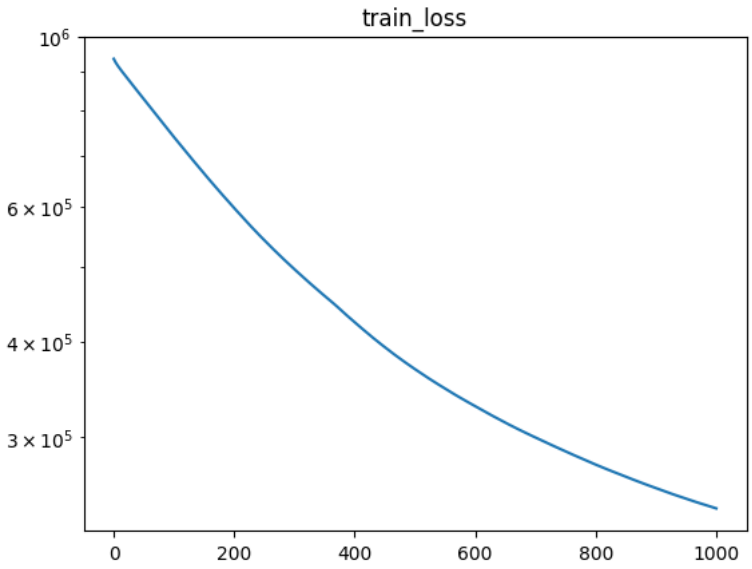




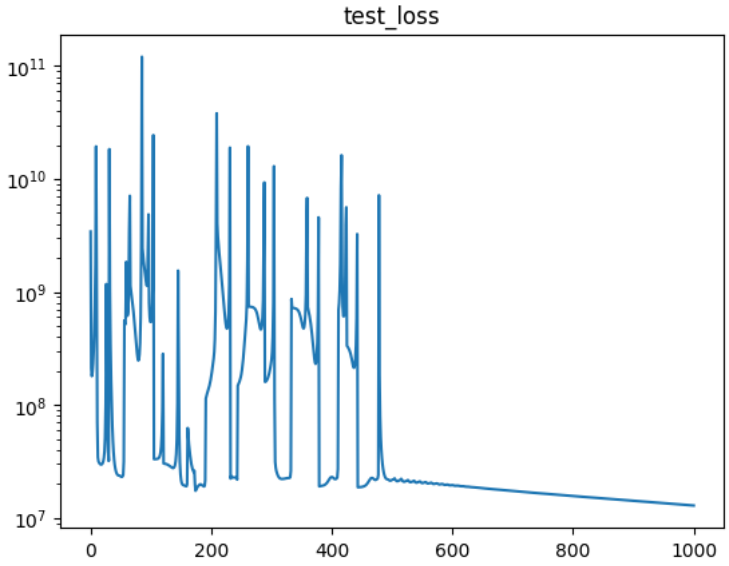


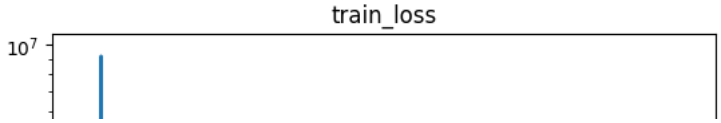
optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 0.01 , minimum_RMSE: 1128.82 , epoch: 1000 , test_loss: 1139 , train_loss

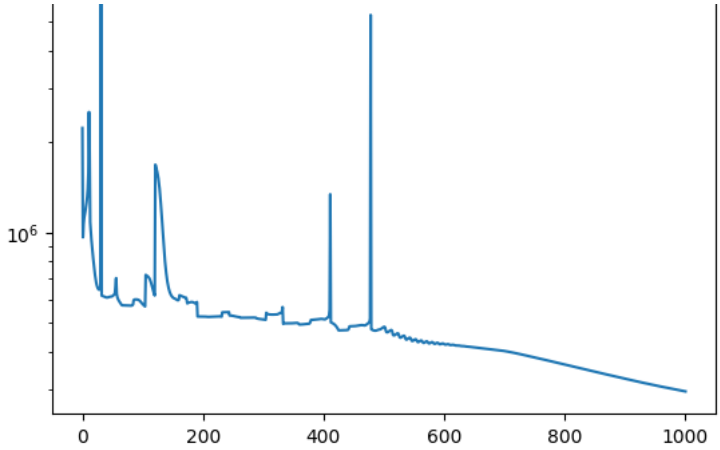




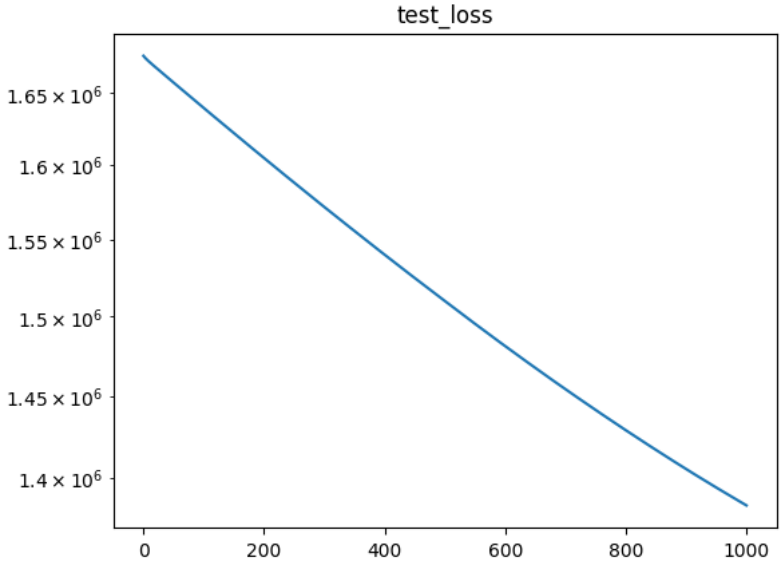
optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 0.05 , minimum_RMSE: 3591.47 , epoch: 1000 , test_loss: 3591 , train_loss

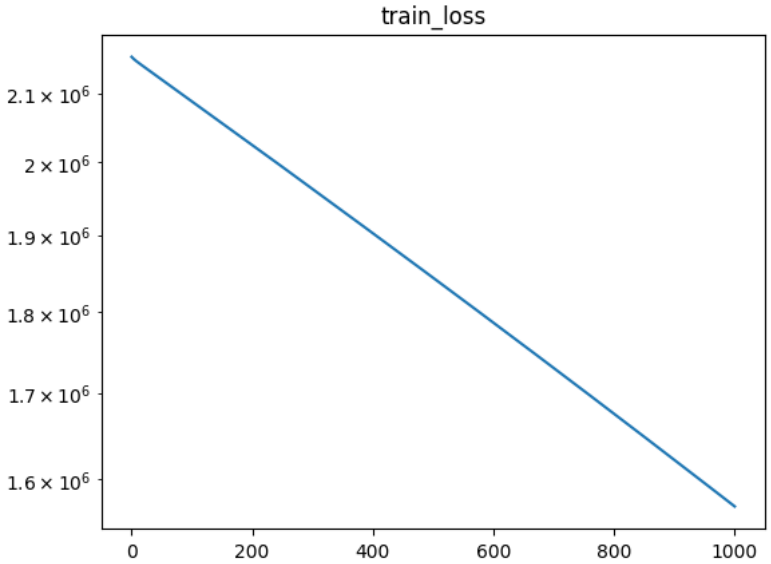




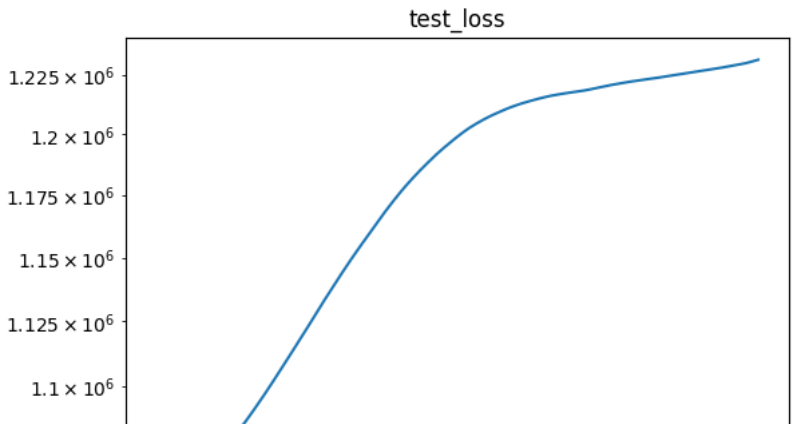


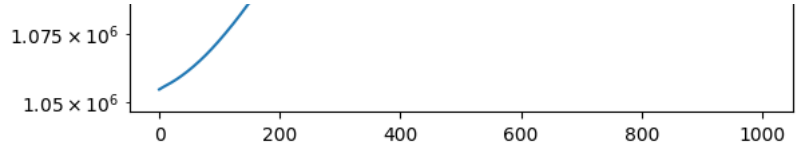
optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 0.001 , minimum_RMSE: 1176.34 , epoch: 1000 , test_loss: 1176 , train_los

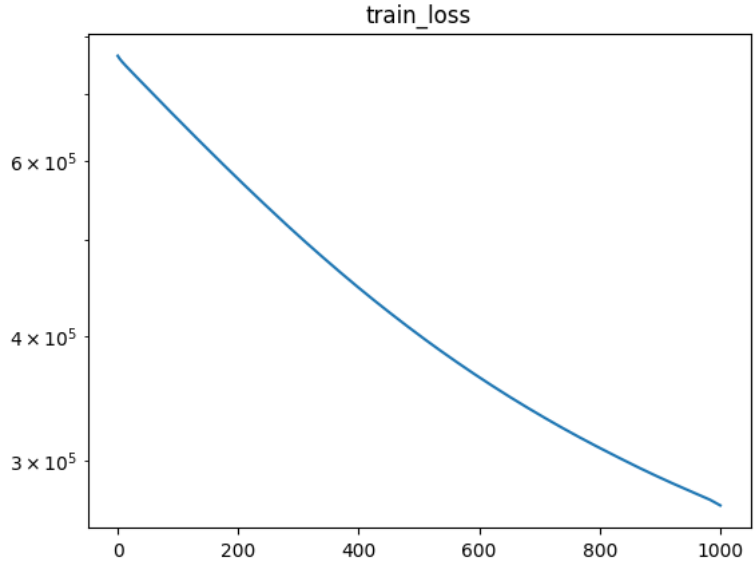




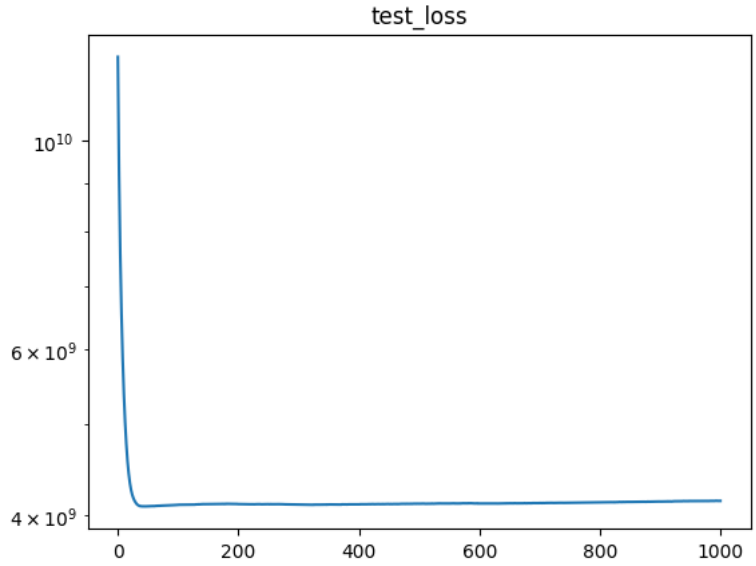
optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 0.005 , minimum_RMSE: 1026.84 , epoch: 1000 , test_loss: 1109 , train_los

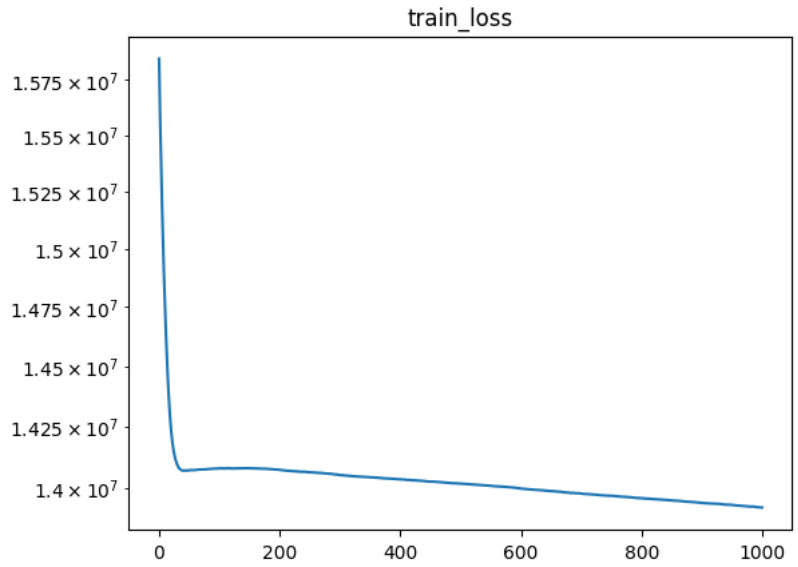




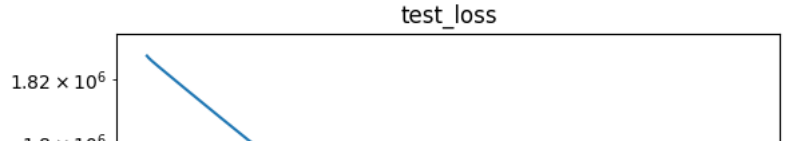


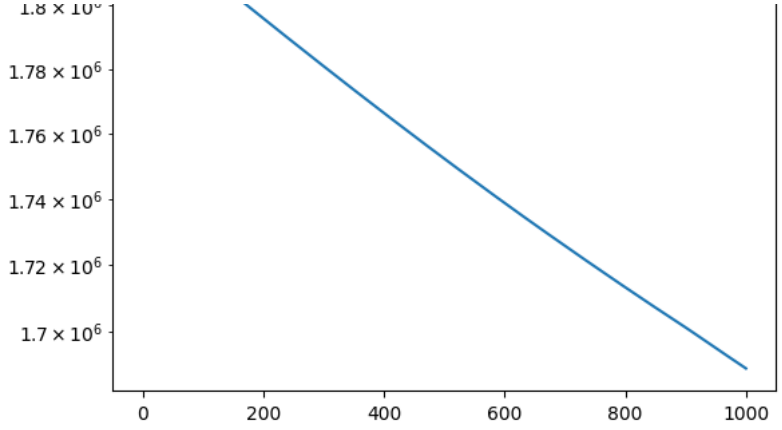
optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 0.0001 , minimum_RMSE: 63942.14 , epoch: 1000 , test_loss: 64376 , train_

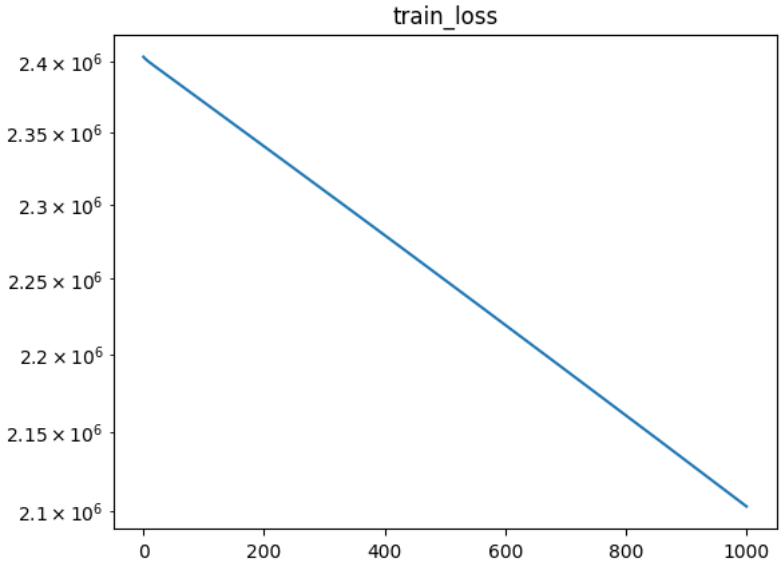




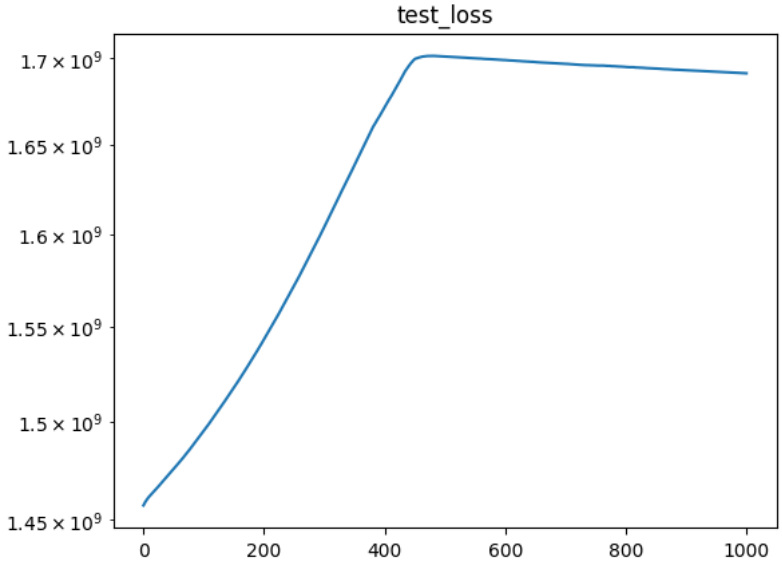
optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 0.0005 , minimum_RMSE: 1299.61 , epoch: 1000 , test_loss: 1299 , train_l

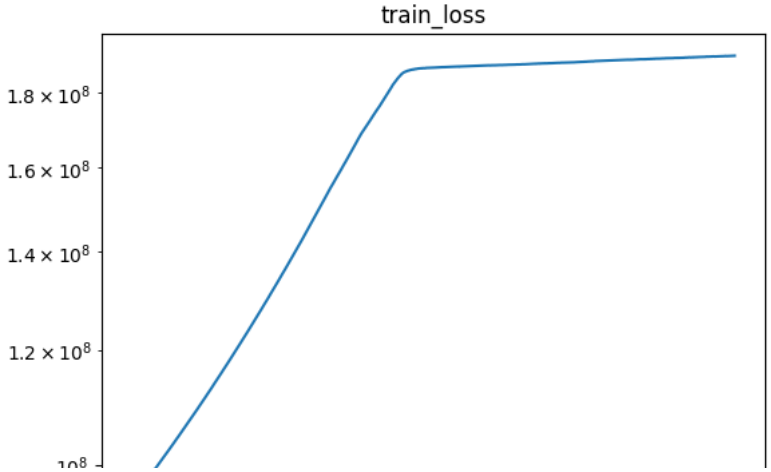


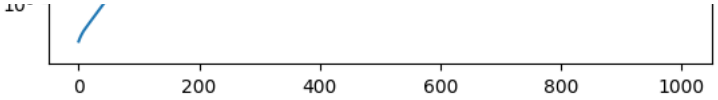




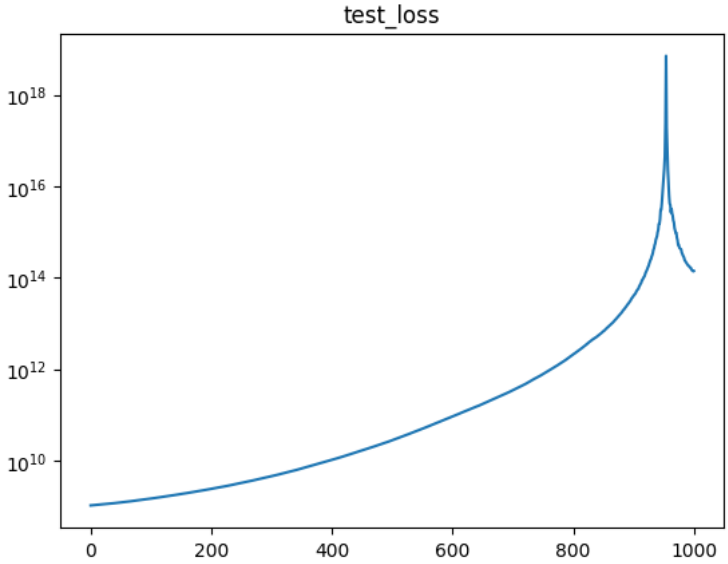
optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-05 , minimum_RMSE: 38174.19 , epoch: 1000 , test_loss: 41124 , train_lo

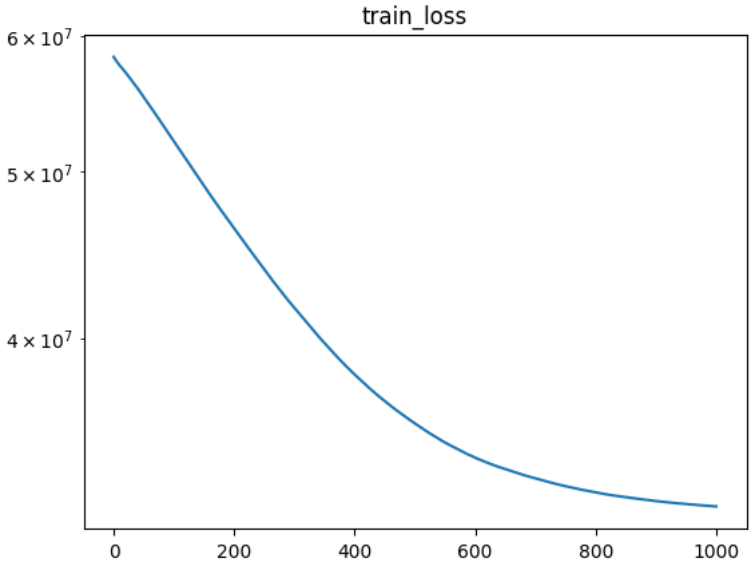




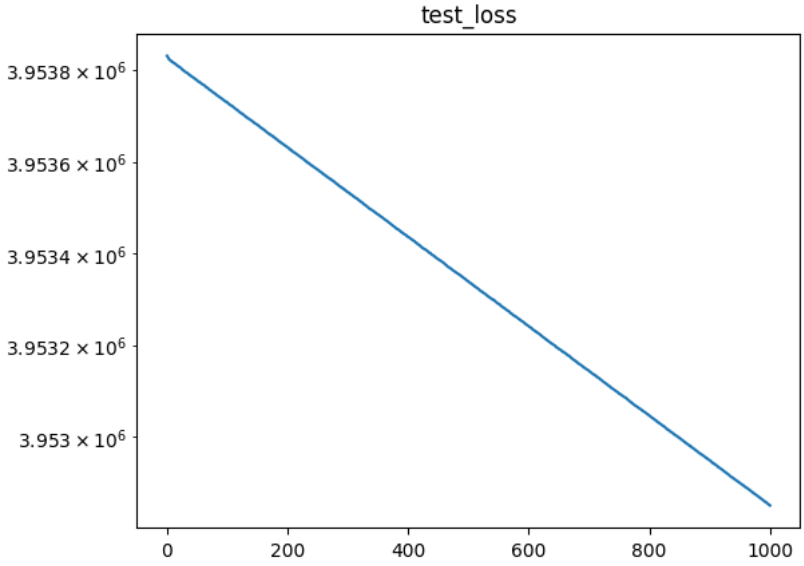


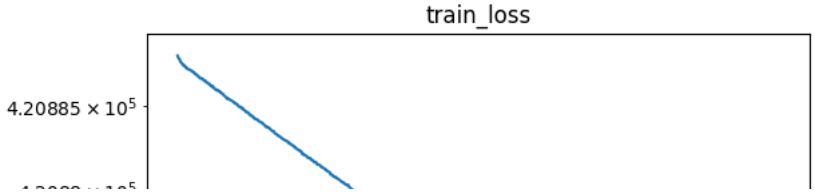
optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-05 , minimum_RMSE: 32569.47 , epoch: 1000 , test_loss: 11910895 , train

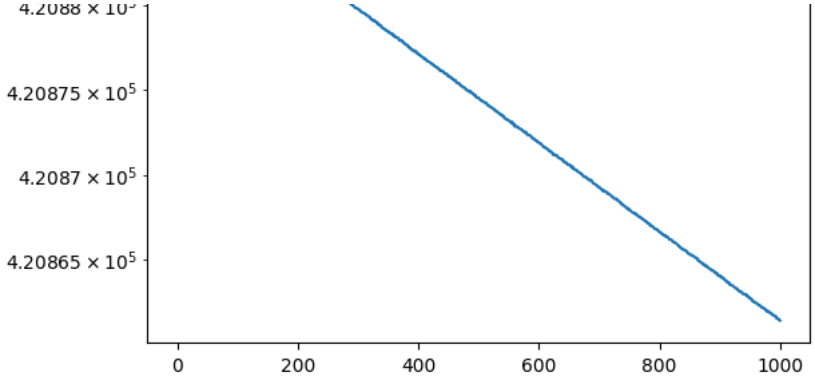




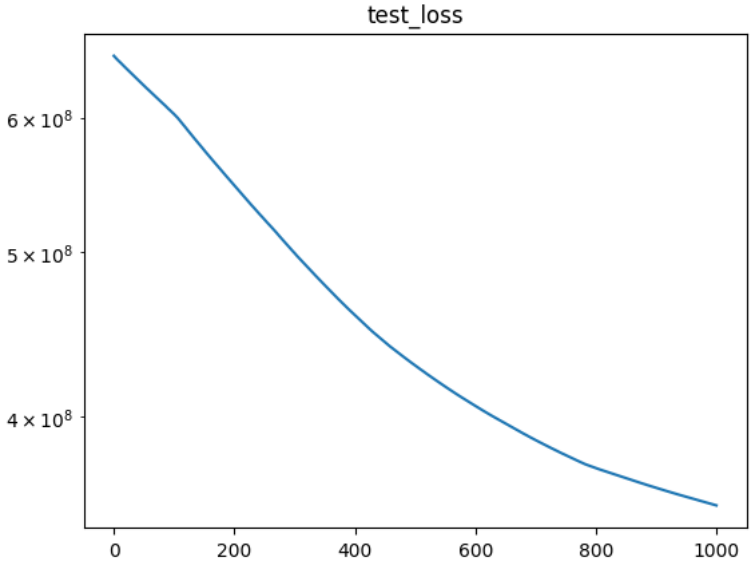
optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-06 , minimum_RMSE: 1988.18 , epoch: 1000 , test_loss: 1988 , train_los

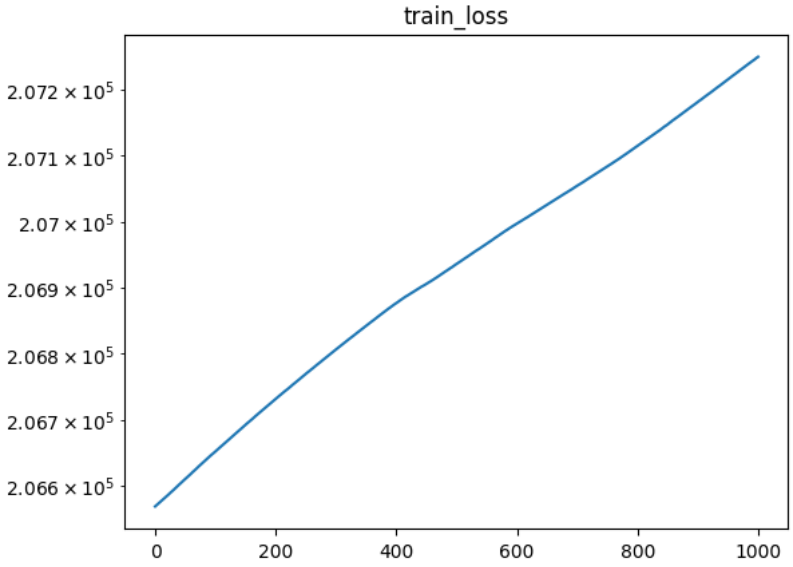




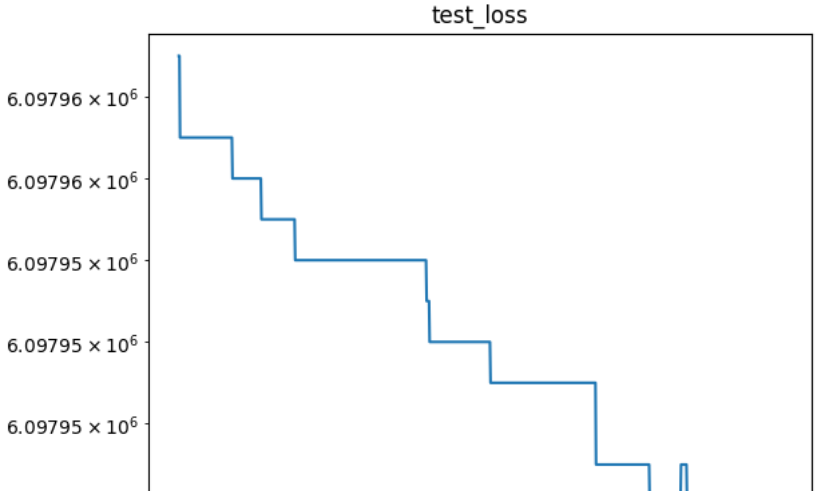


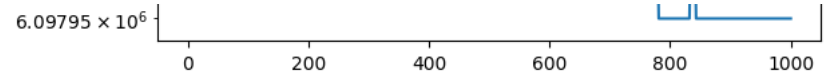
optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-06 , minimum_RMSE: 18833.19 , epoch: 1000 , test_loss: 18833 , train_lo

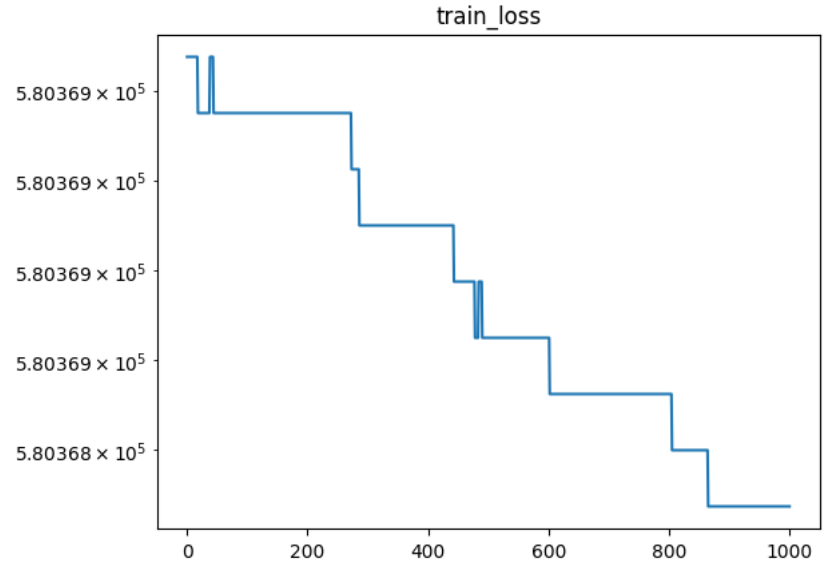




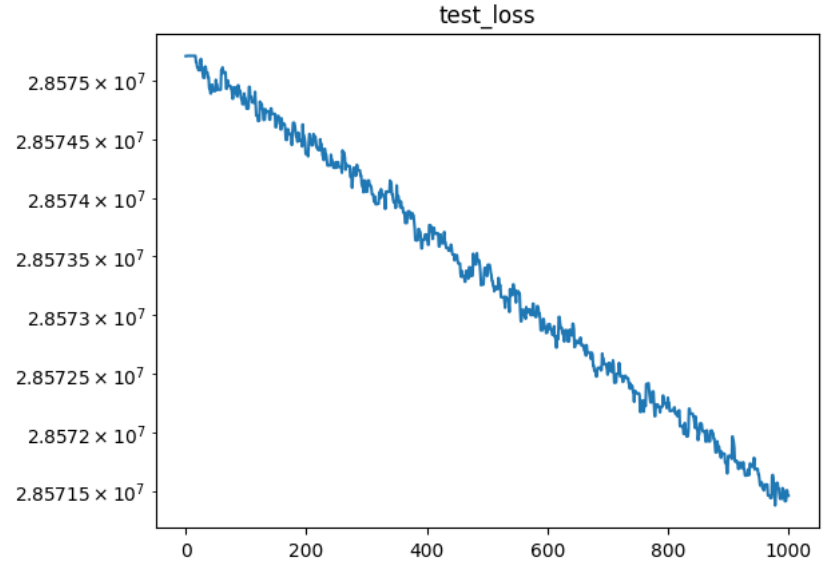
optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-07 , minimum_RMSE: 2469.40 , epoch: 1000 , test_loss: 2469 , train_los

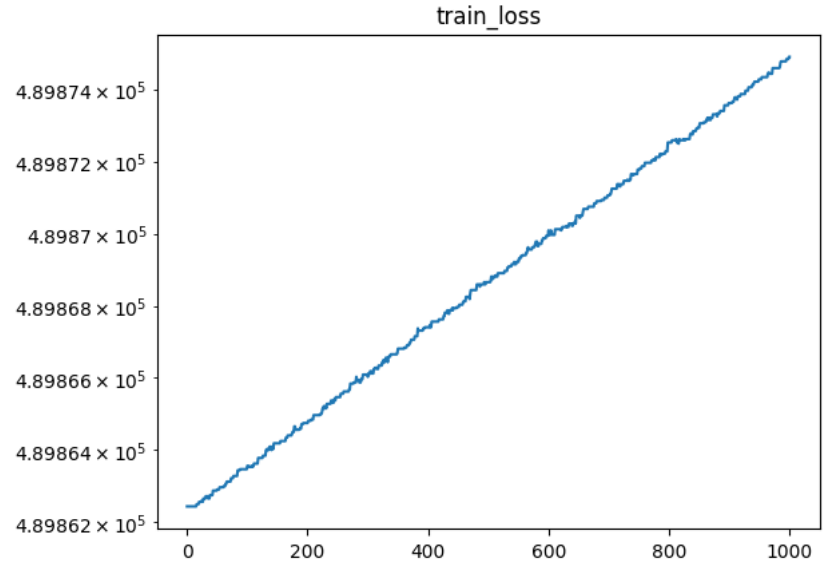




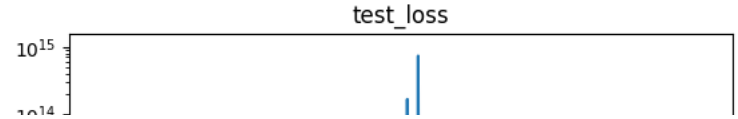


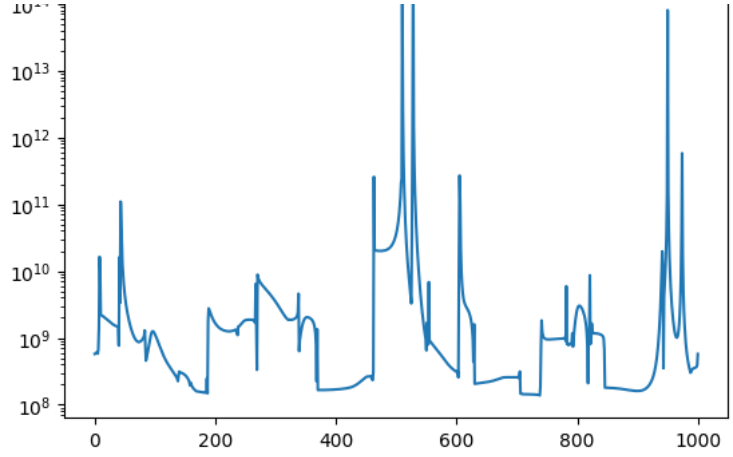
optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-07 , minimum_RMSE: 5345.22 , epoch: 1000 , test_loss: 5345 , train_los

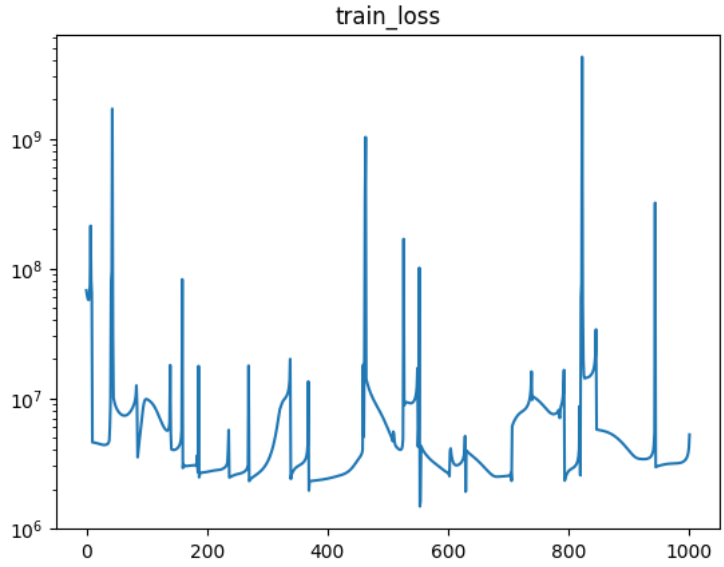




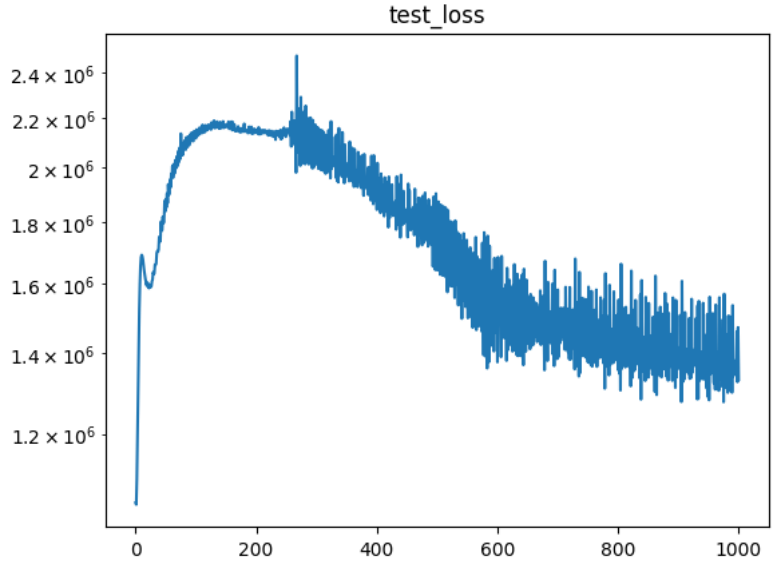
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/loss.py:536: UserWarning: Using a target size (torch.Size([500])) that is different t
return F.mse_loss(input, target, reduction=self.reduction)
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/loss.py:536: UserWarning: Using a target size (torch.Size([200])) that is different t
return F.mse_loss(input, target, reduction=self.reduction)
optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 0.1 , minimum_RMSE: 11752.05 , epoch: 1000 , test_loss: 23996 , train_loss

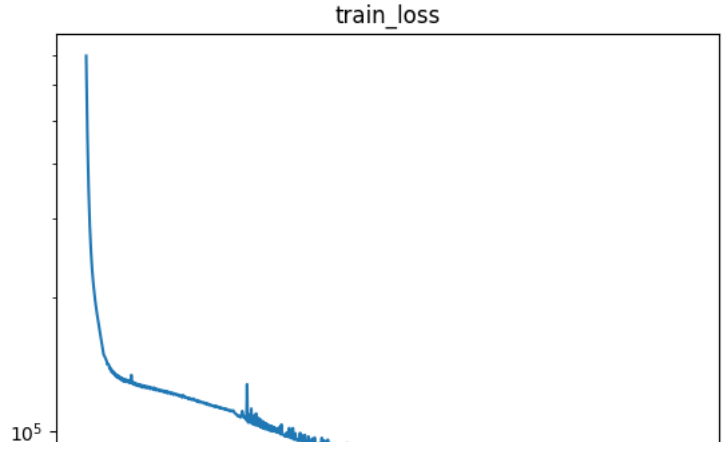


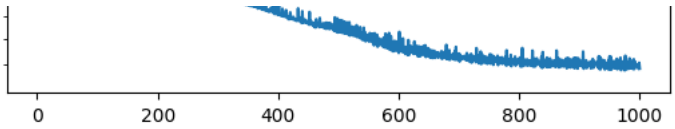




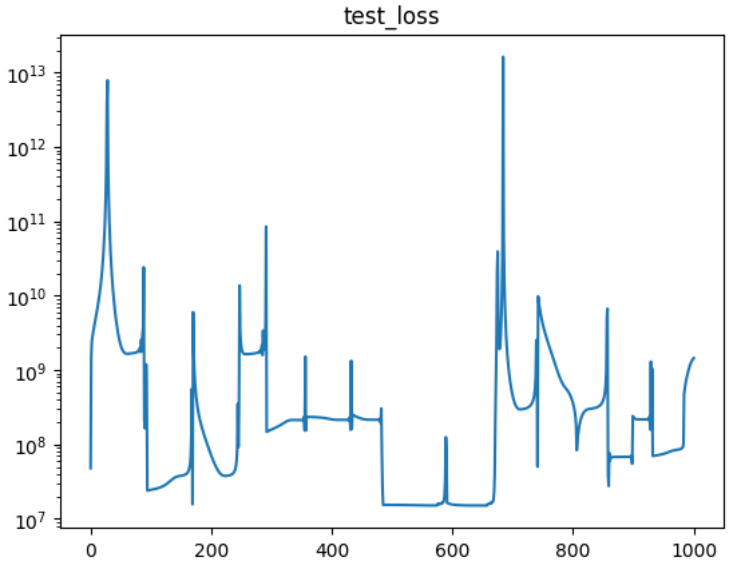
optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 0.5 , minimum_RMSE: 1023.78 , epoch: 1000 , test_loss: 1154 , train_loss

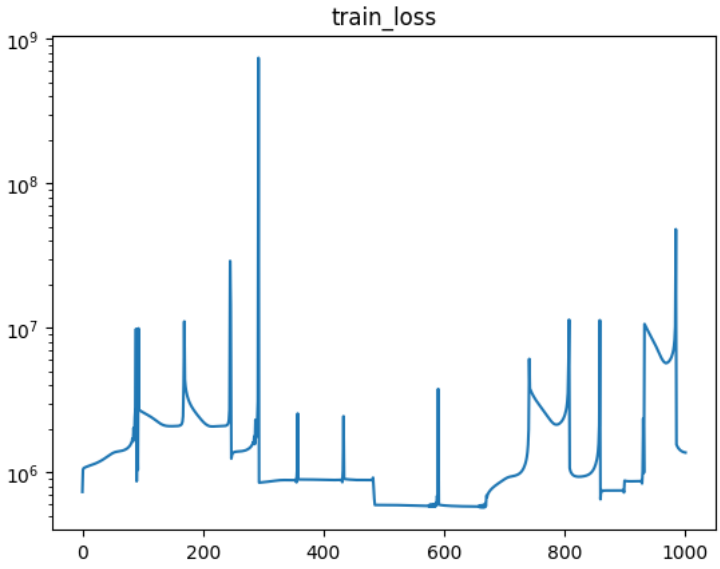




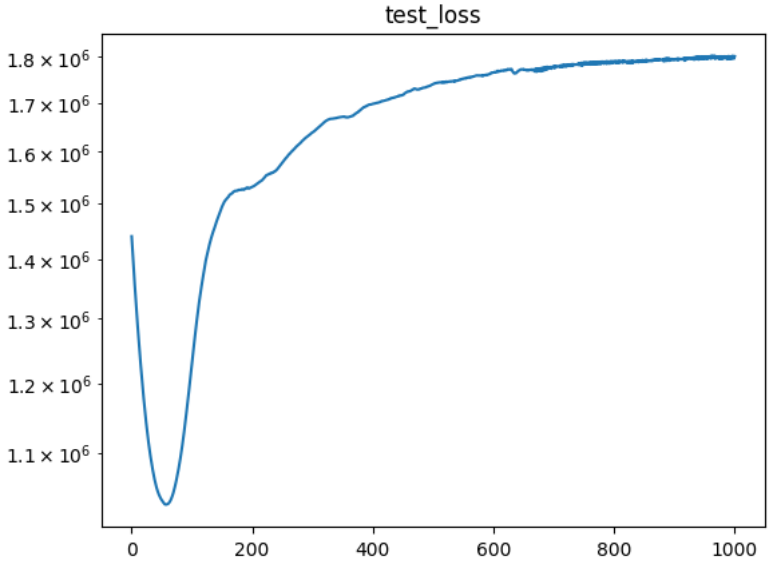


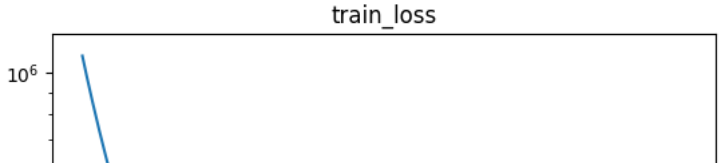
optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 0.01 , minimum_RMSE: 3885.09 , epoch: 1000 , test_loss: 38045 , train_loss

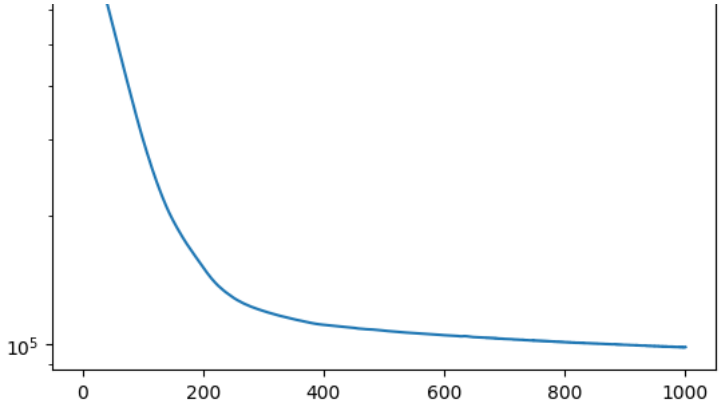




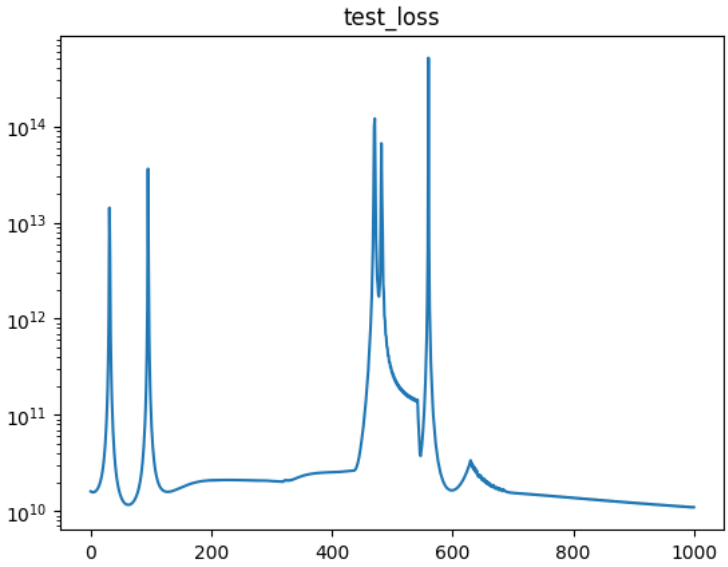
optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 0.05 , minimum_RMSE: 1015.88 , epoch: 1000 , test_loss: 1341 , train_loss

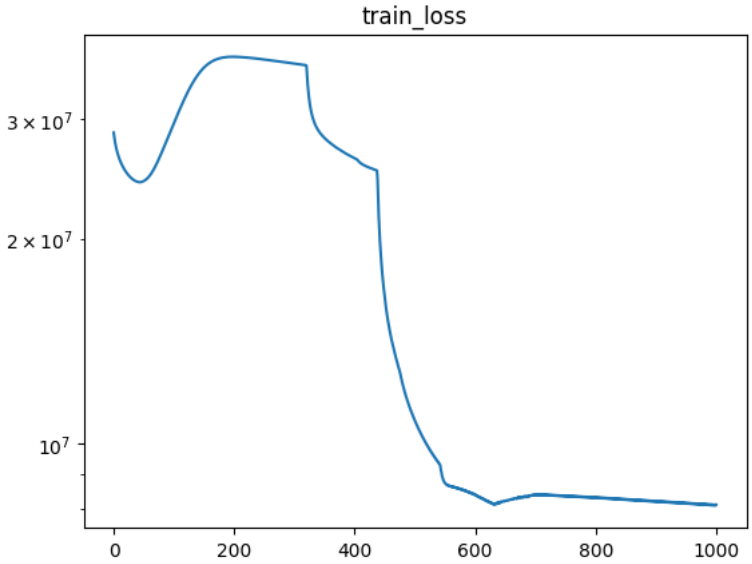




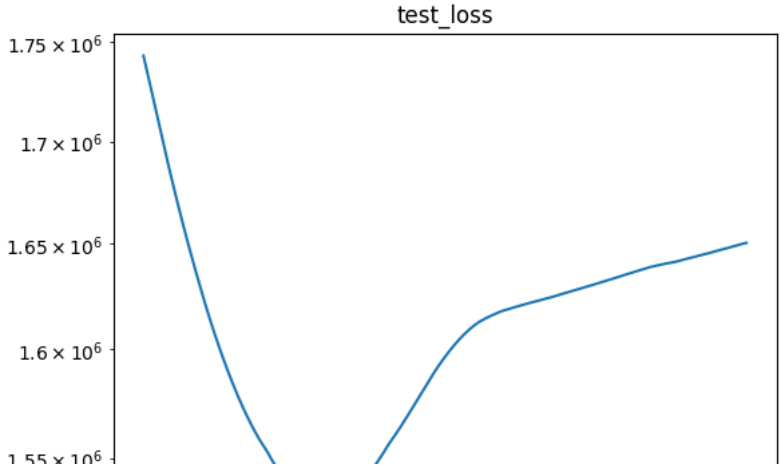


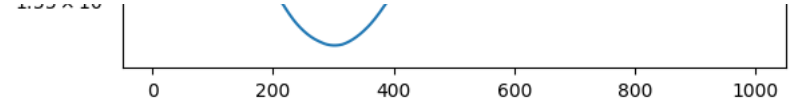
optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 0.001 , minimum_RMSE: 104543.28 , epoch: 1000 , test_loss: 104775 , train_

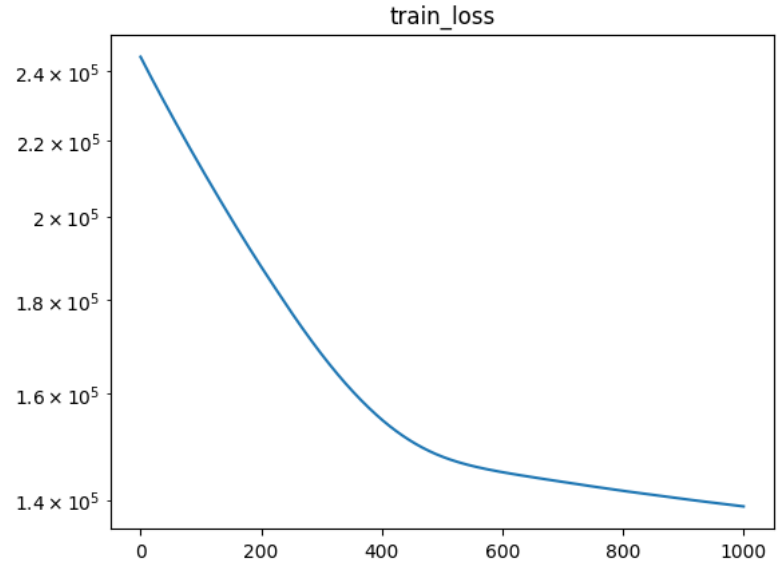




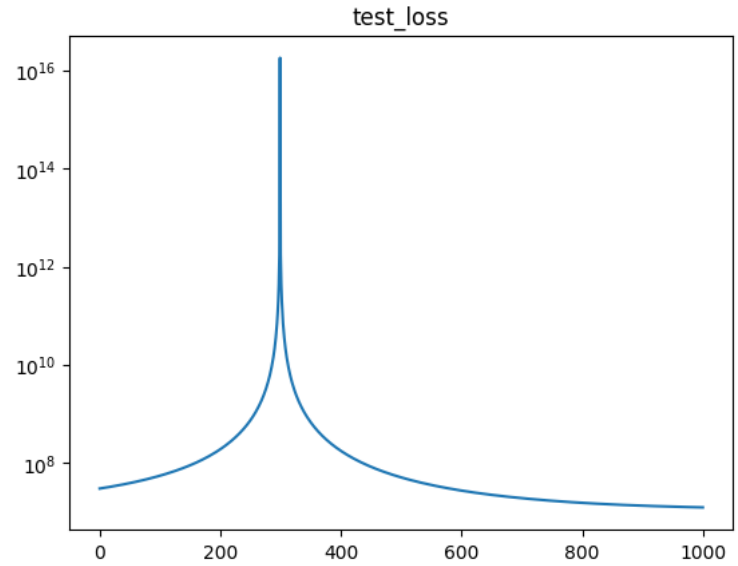
optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 0.005 , minimum_RMSE: 1236.63 , epoch: 1000 , test_loss: 1284 , train_los

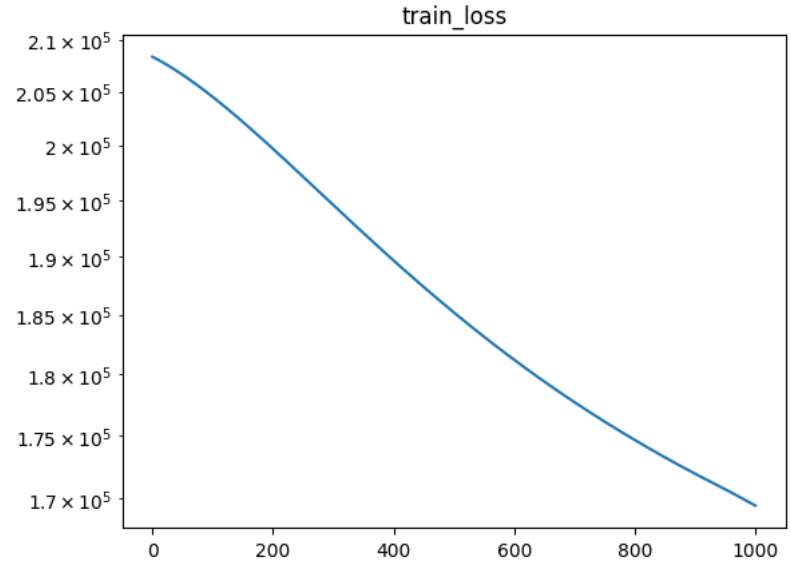




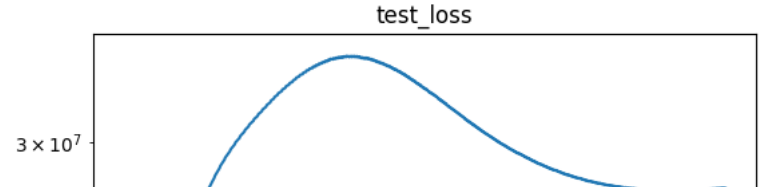


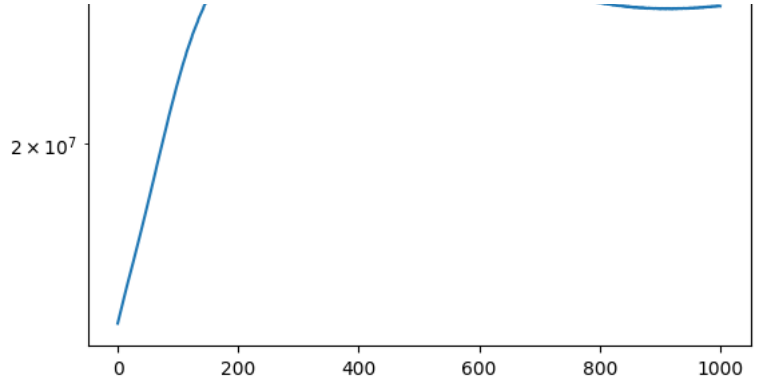
optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 0.0001 , minimum_RMSE: 3527.47 , epoch: 1000 , test_loss: 3527 , train_l

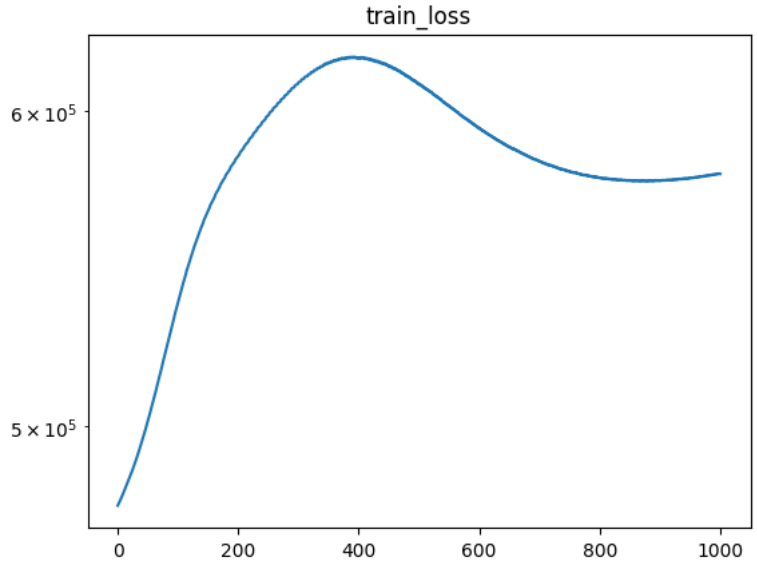




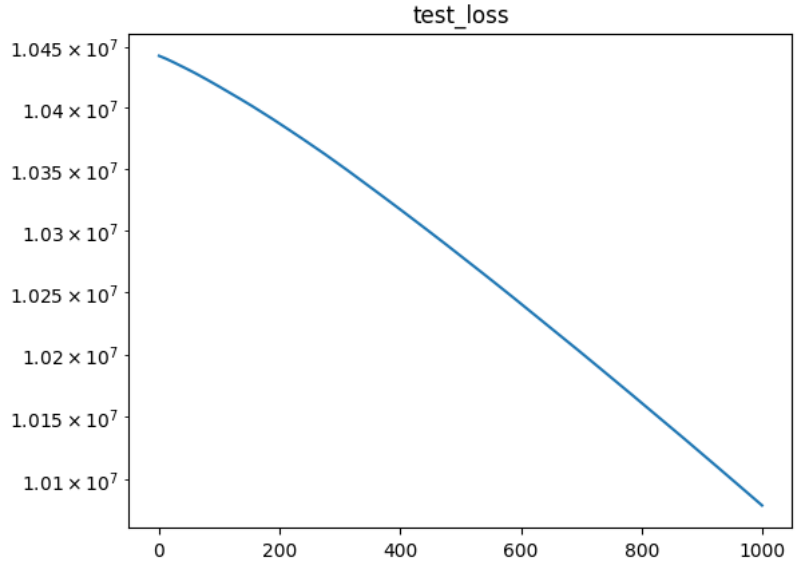
optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 0.0005 , minimum_RMSE: 3667.14 , epoch: 1000 , test_loss: 5205 , train_l

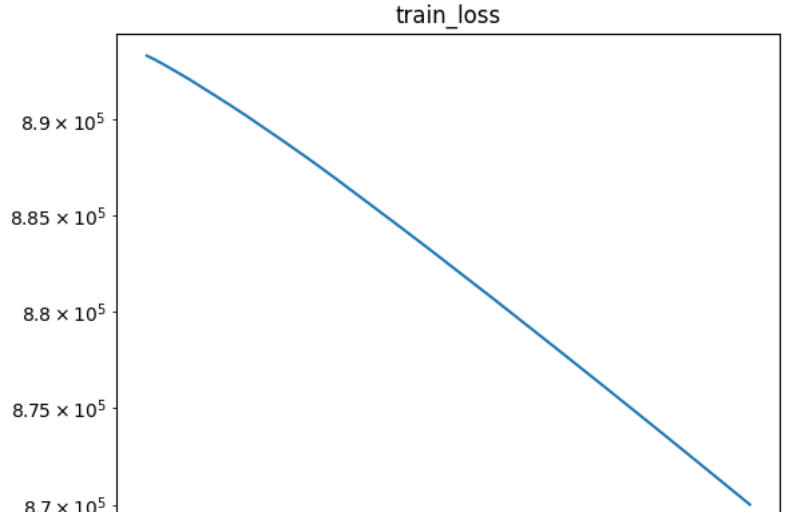




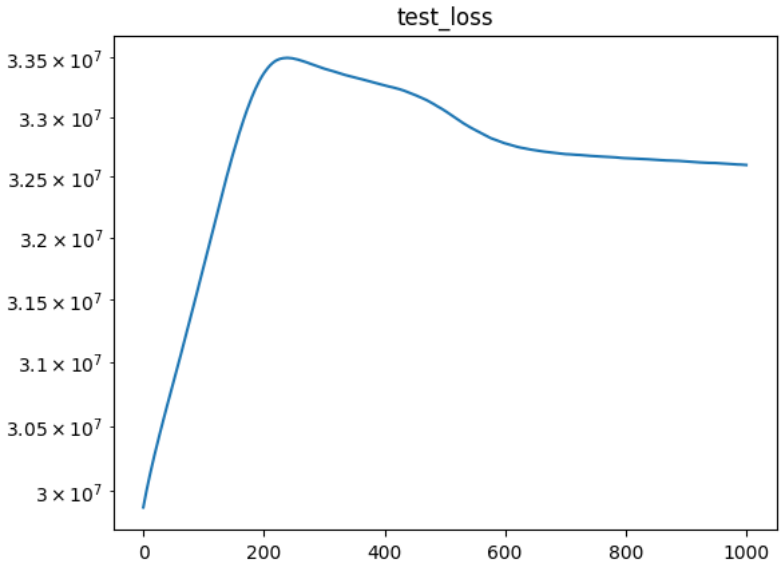


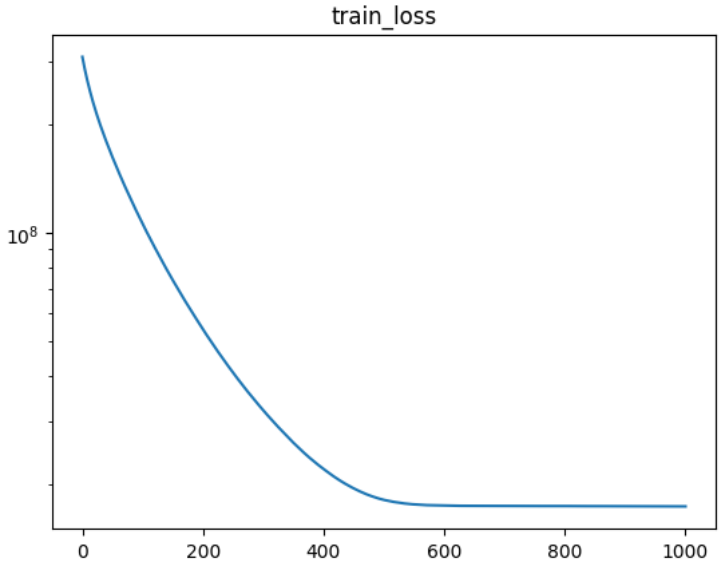
optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-05 , minimum_RMSE: 3174.81 , epoch: 1000 , test_loss: 3174 , train_los



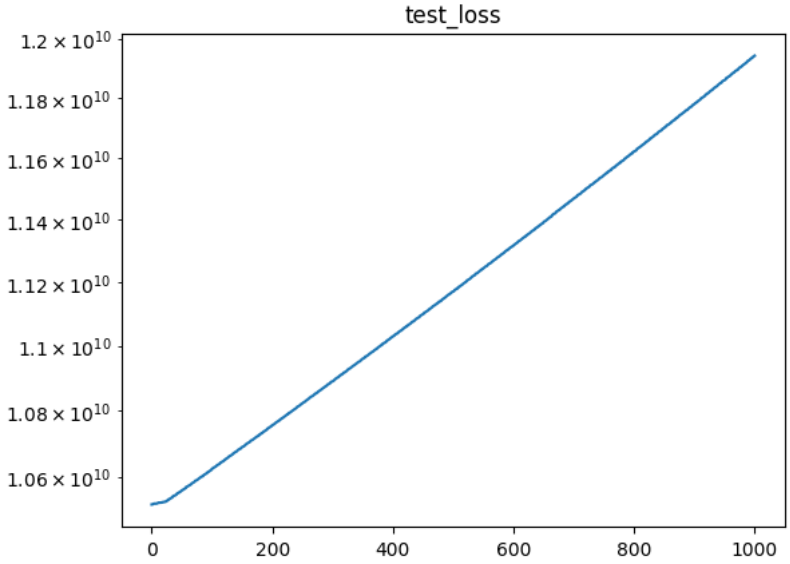


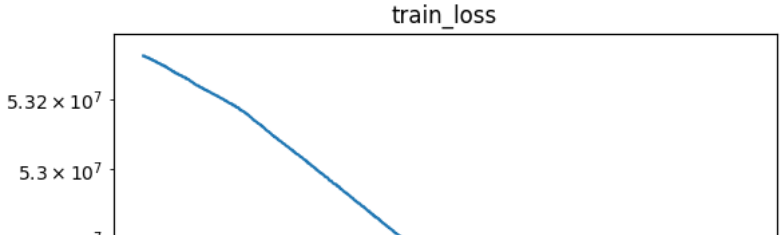
optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 5e-05 , minimum_RMSE: 5465.84 , epoch: 1000 , test_loss: 5709 , train_los

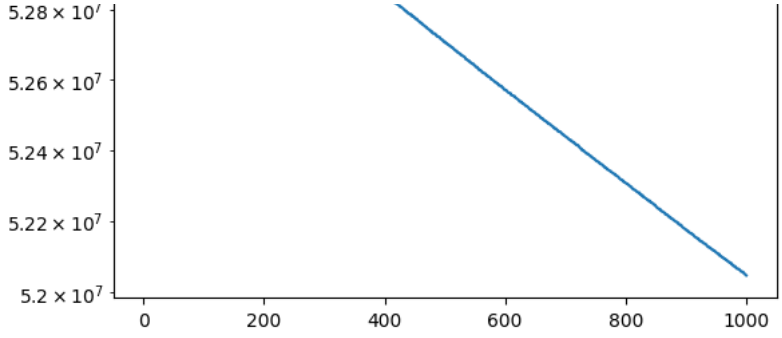




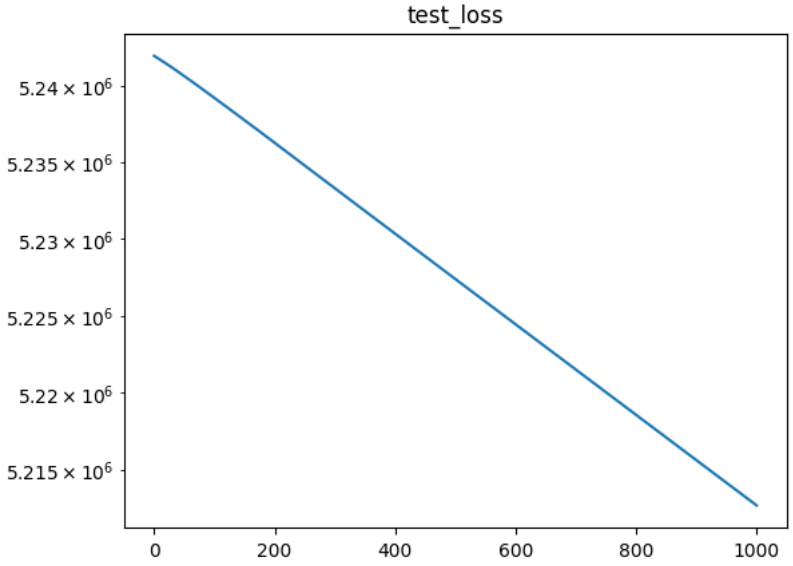
optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-06 , minimum_RMSE: 102550.92 , epoch: 1000 , test_loss: 109282 , train_

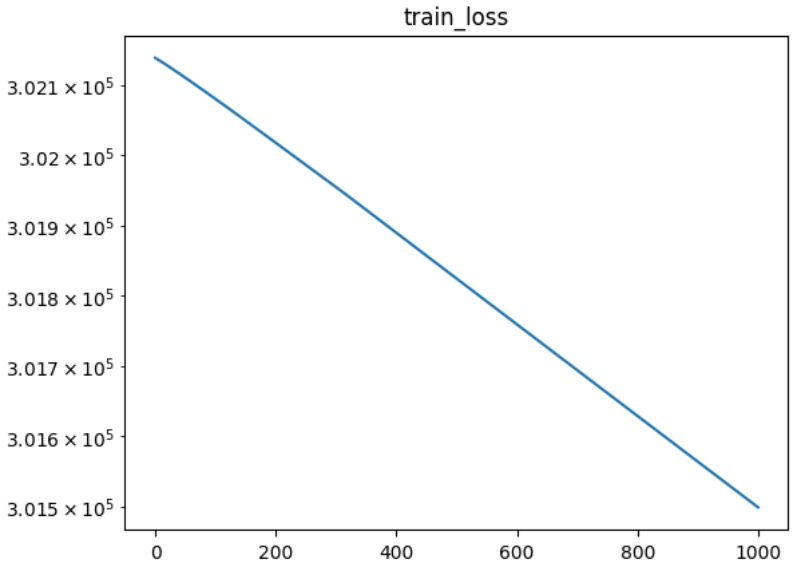




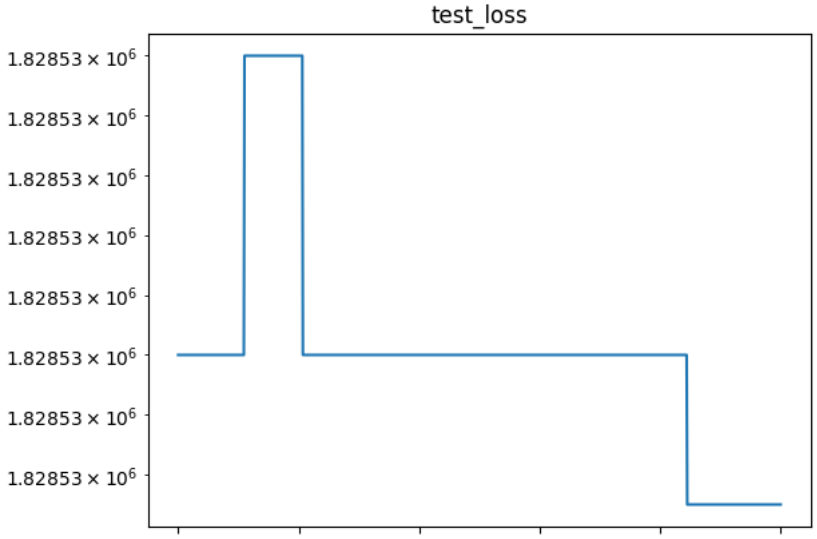


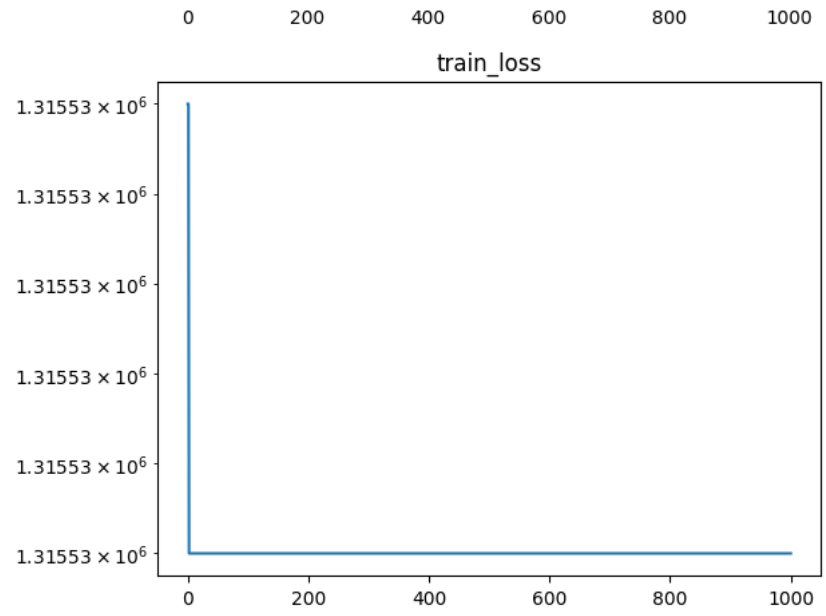
optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 5e-06 , minimum_RMSE: 2283.13 , epoch: 1000 , test_loss: 2283 , train_los



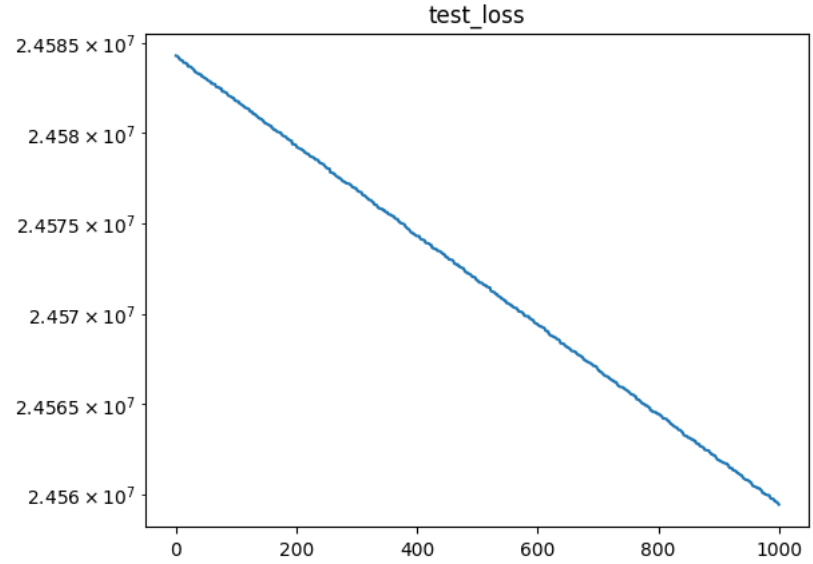


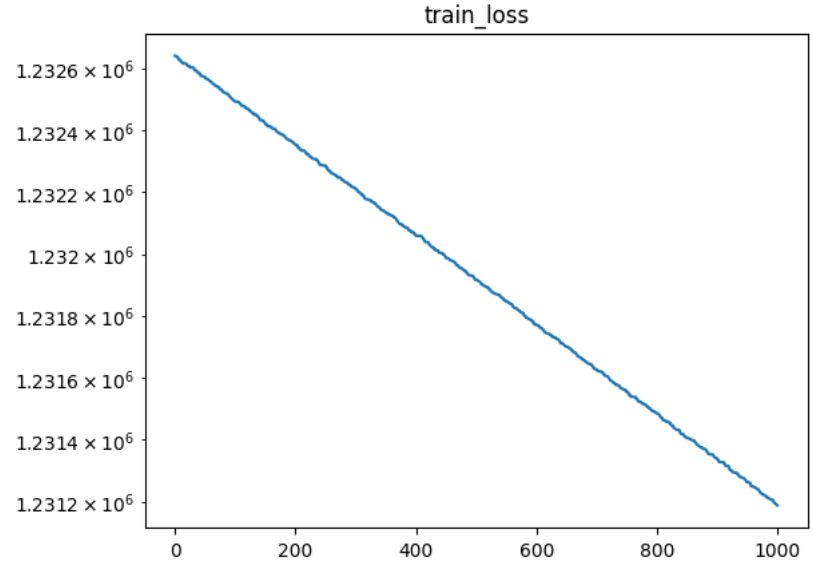
optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-07 , minimum_RMSE: 1352.23 , epoch: 1000 , test_loss: 1352 , train_los



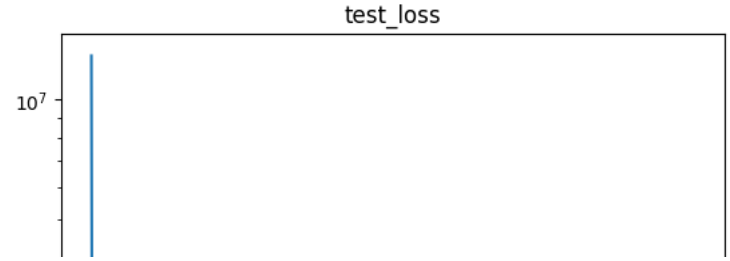


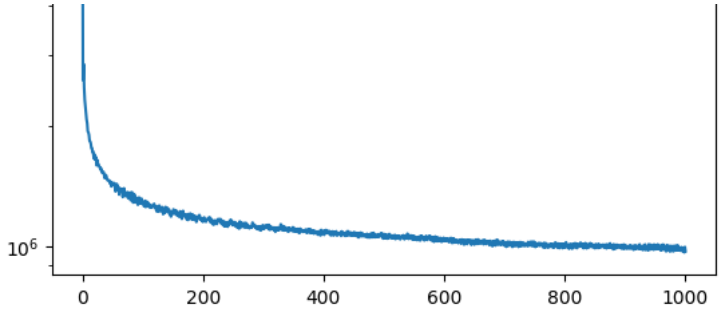
optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 5e-07 , minimum_RMSE: 4955.75 , epoch: 1000 , test_loss: 4955 , train_los

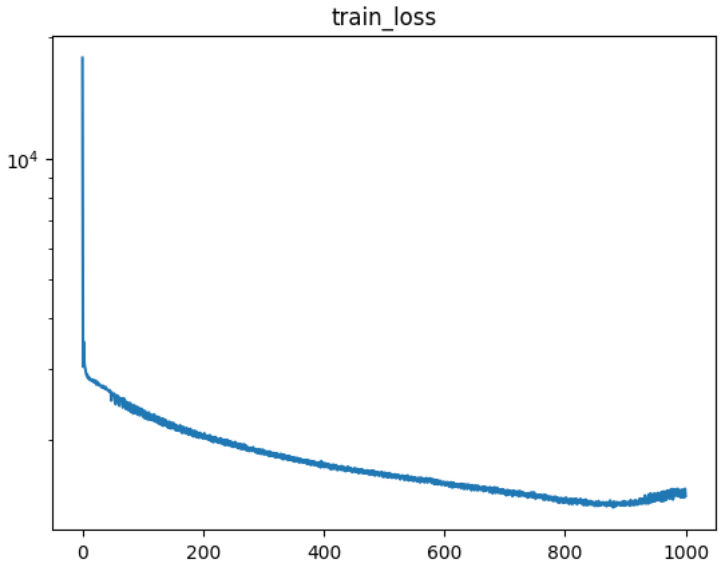




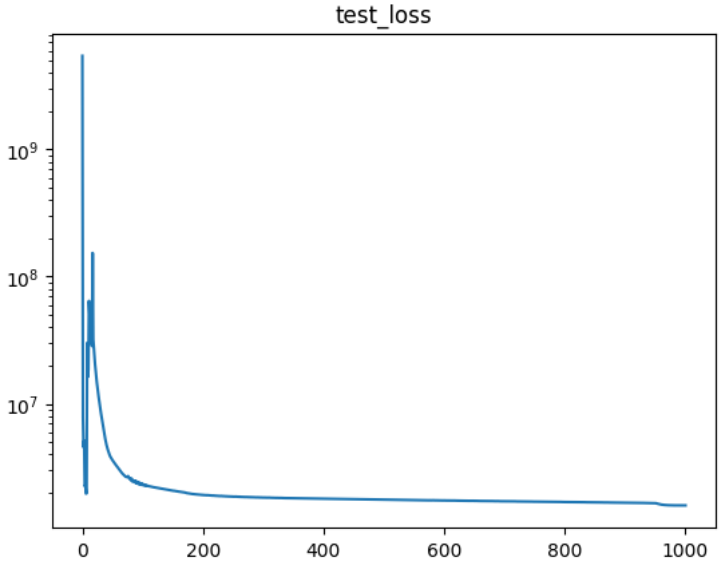
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 0.1 , minimum_RMSE: 982.28 , epoch: 1000 , test_loss: 985 , train_loss:

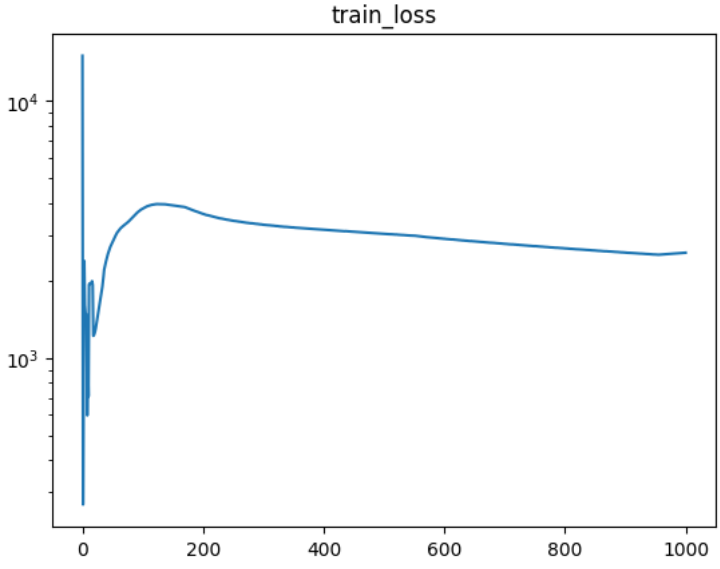






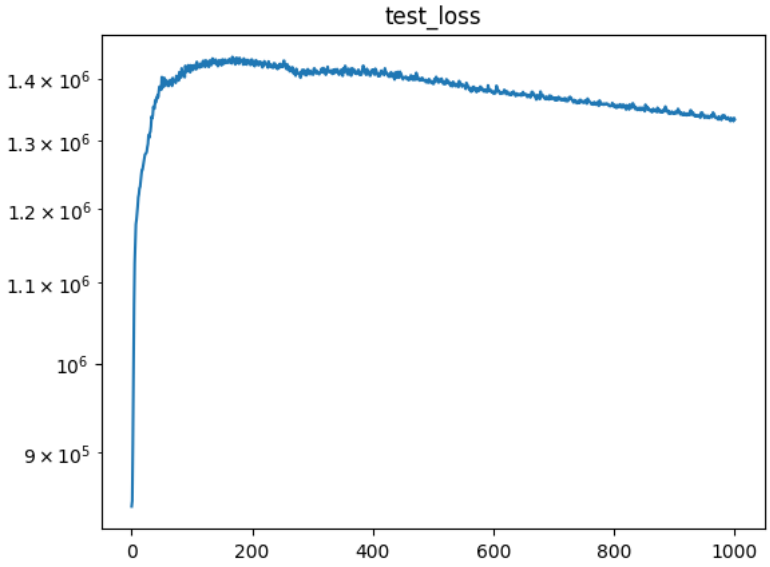
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 0.5 , minimum_RMSE: 1259.95 , epoch: 1000 , test_loss: 1260 , train_loss:

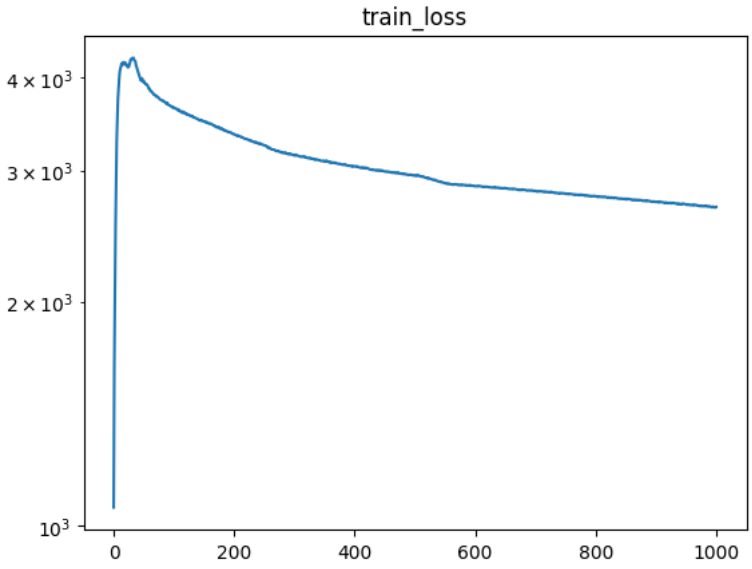




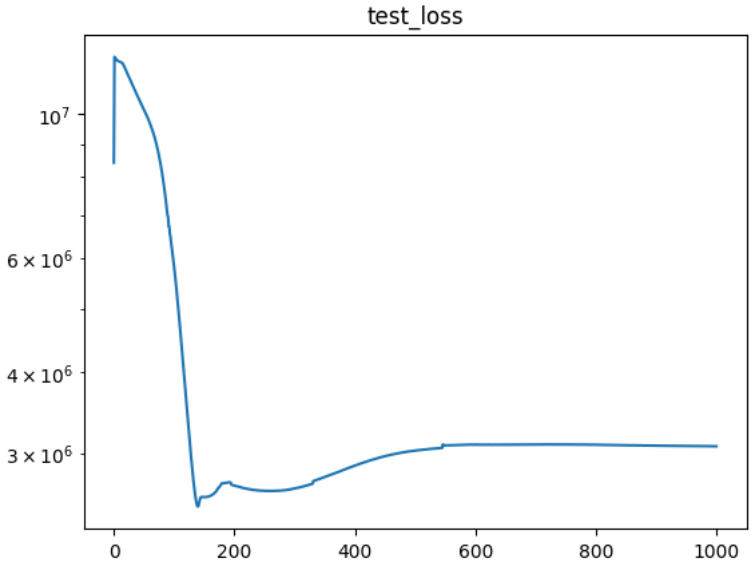
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 0.01 , minimum_RMSE: 919.13 , epoch: 1000 , test_loss: 1154 , train_loss:

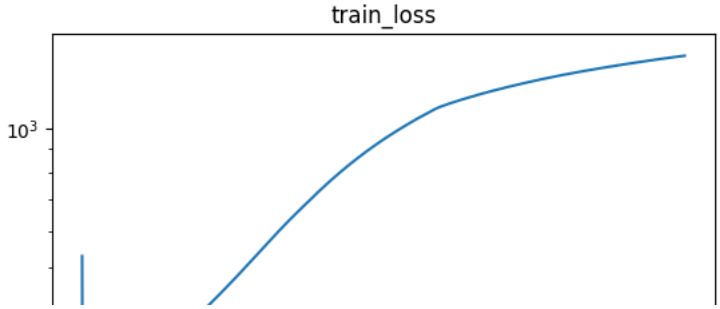
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 0.01 , minimum_RMSE: 919.13 , epoch: 1000 , test_loss: 1754 , train_loss:

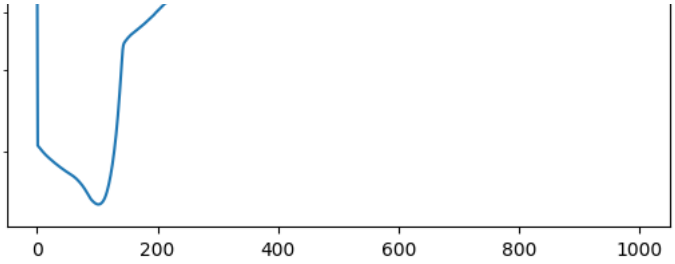




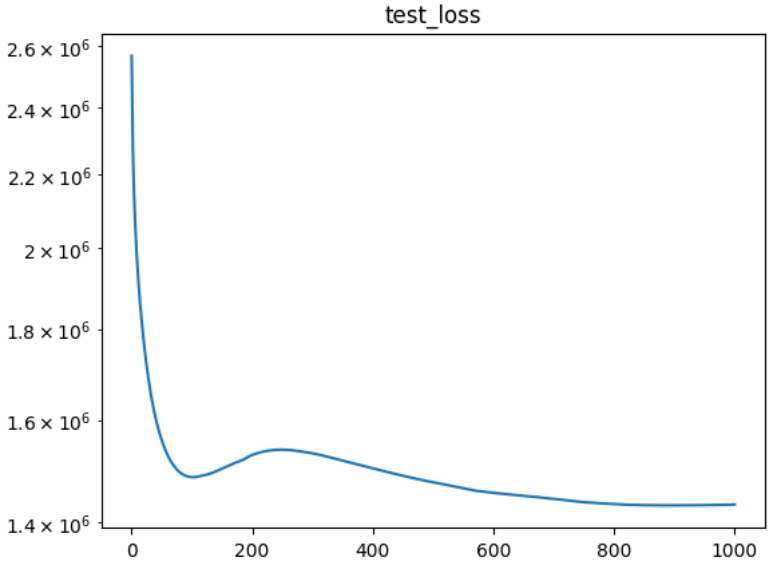
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 0.05 , minimum_RMSE: 1576.89 , epoch: 1000 , test_loss: 1754 , train_loss:

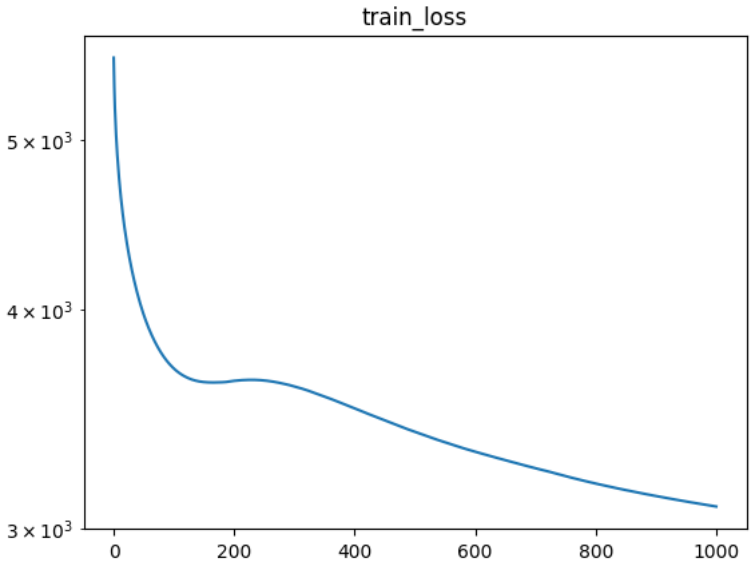




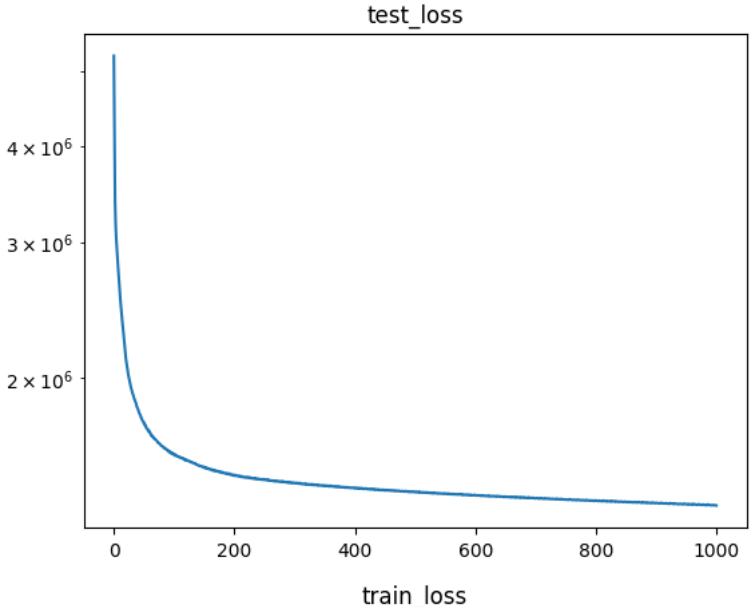


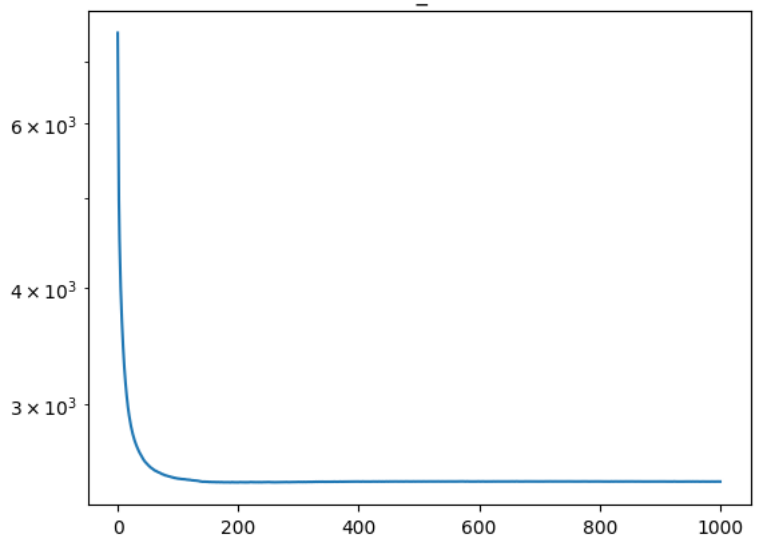
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 0.001 , minimum_RMSE: 1196.63 , epoch: 1000 , test_loss: 1197 , train_loss



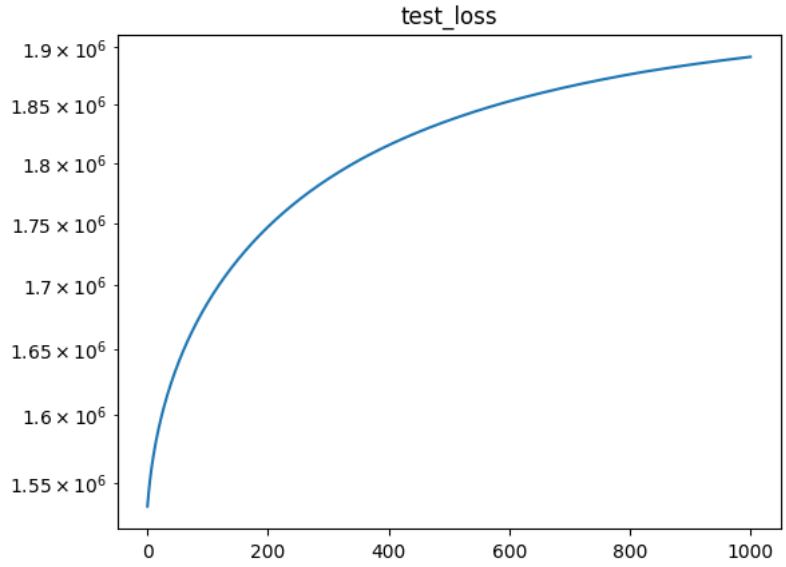


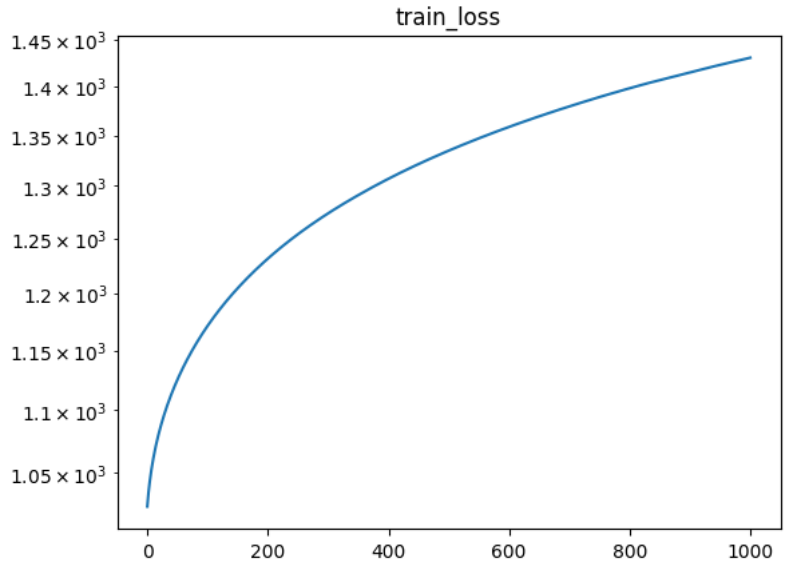
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 0.005 , minimum_RMSE: 1168.44 , epoch: 1000 , test_loss: 1168 , train_loss



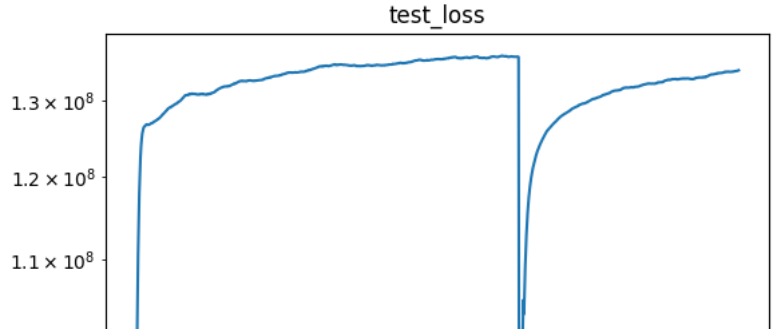


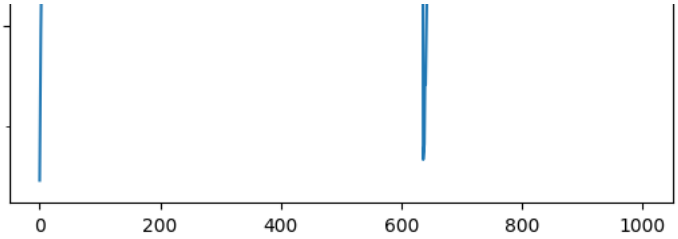
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 0.0001 , minimum_RMSE: 1238.01 , epoch: 1000 , test_loss: 1375 , train_loss

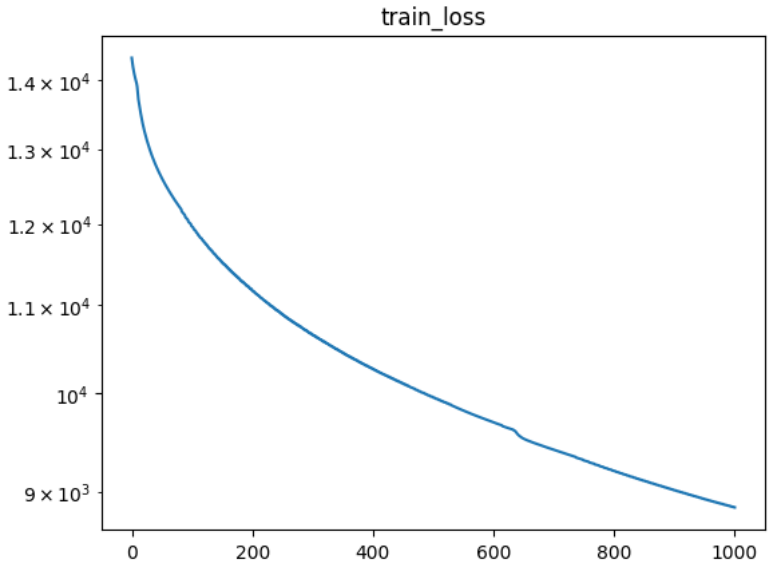




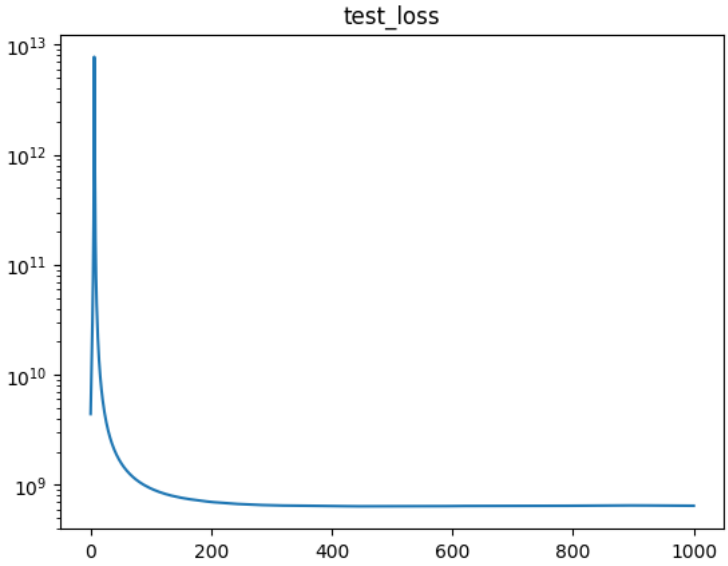
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 0.0005 , minimum_RMSE: 9221.62 , epoch: 1000 , test_loss: 11580 , train_loss

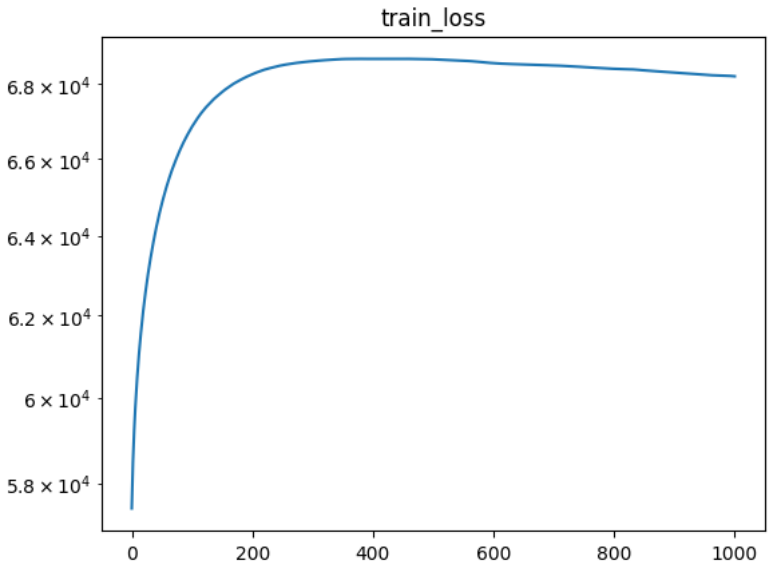




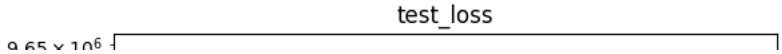


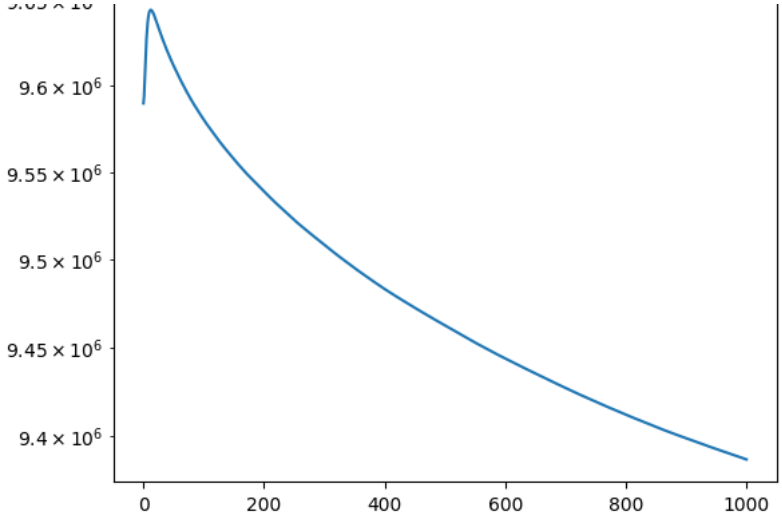
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-05 , minimum_RMSE: 25226.23 , epoch: 1000 , test_loss: 25400 , train_loss

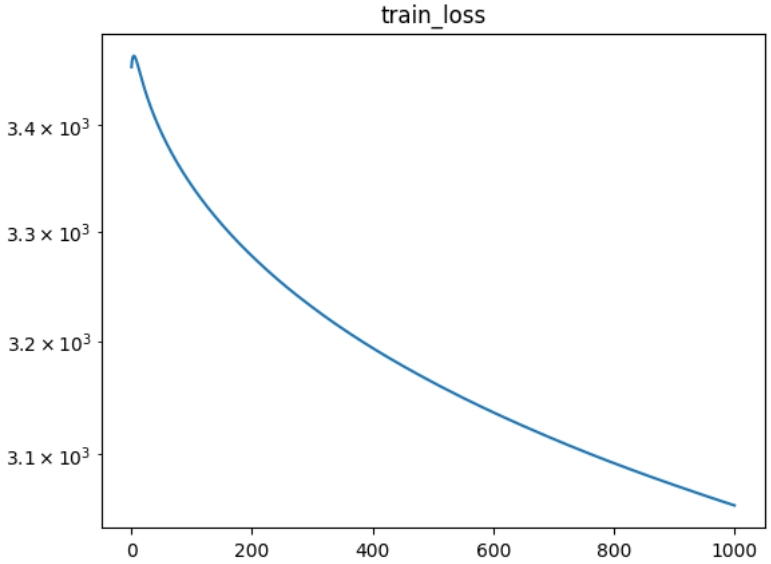




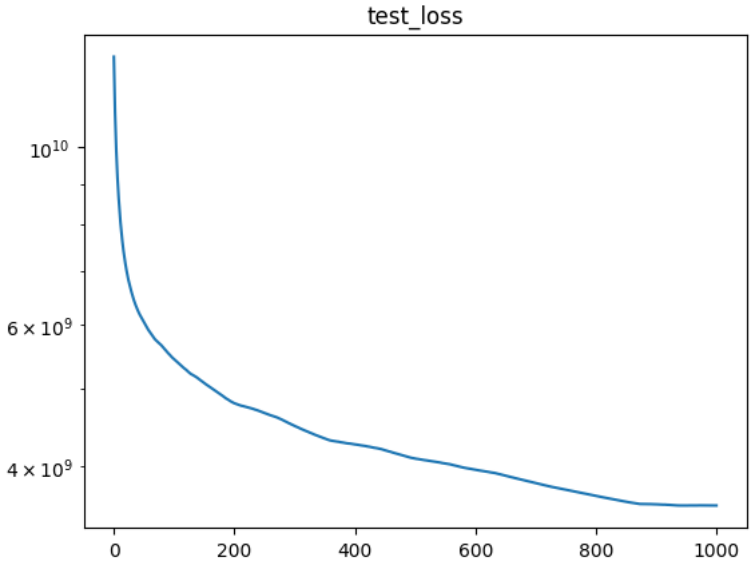
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 5e-05 , minimum_RMSE: 3063.73 , epoch: 1000 , test_loss: 3063 , train_loss

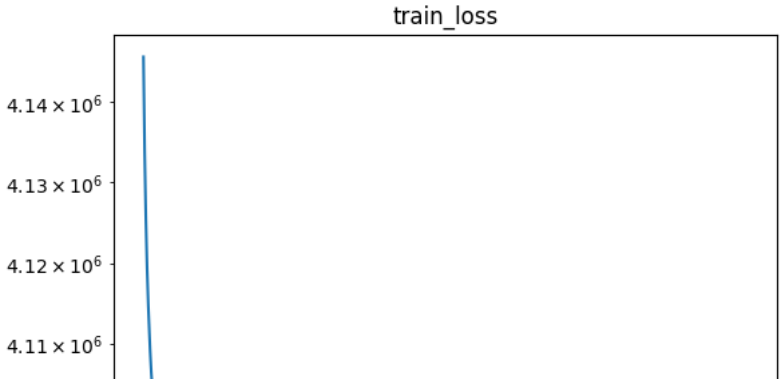


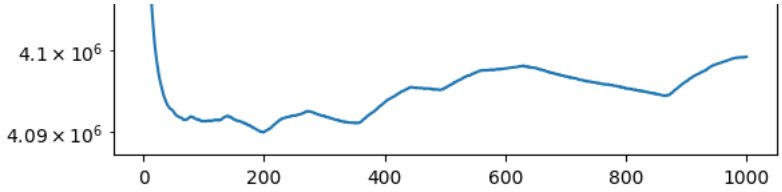




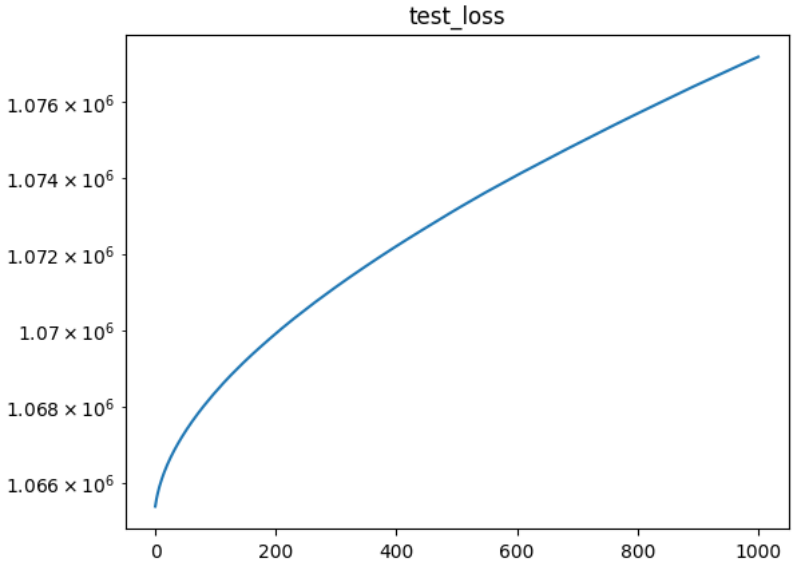
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-06 , minimum_RMSE: 59797.25 , epoch: 1000 , test_loss: 59797 , train_loss

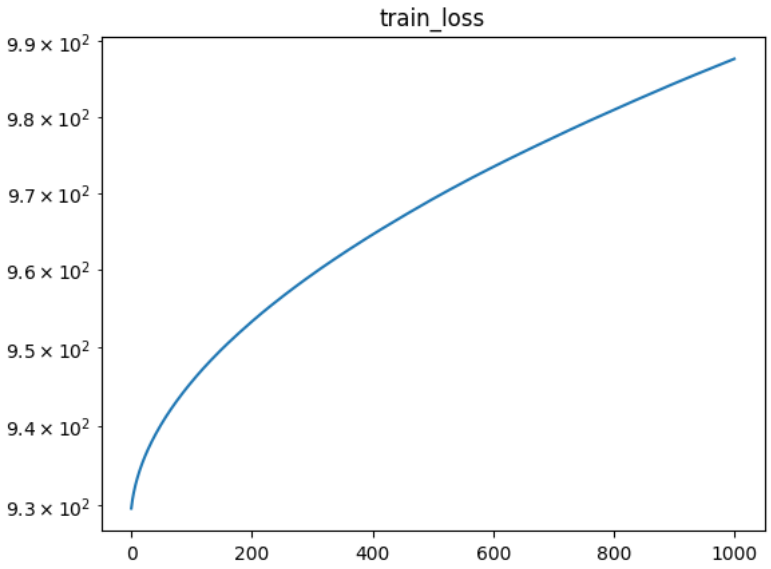




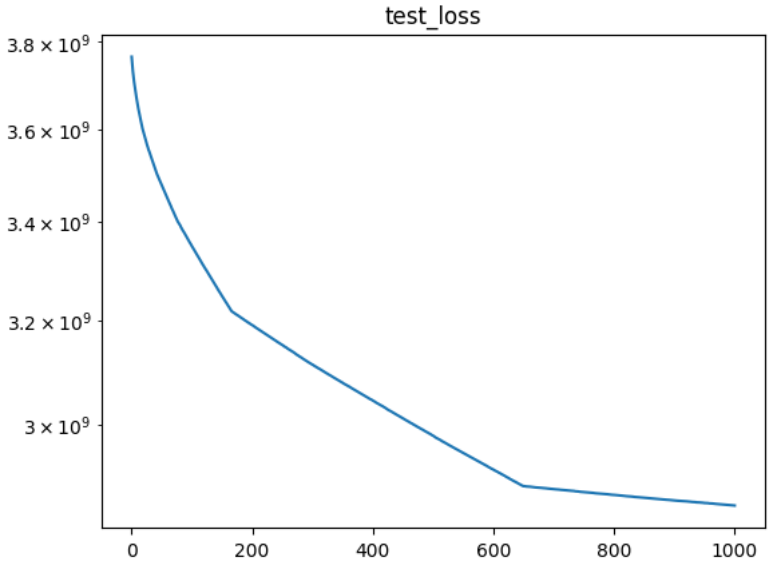


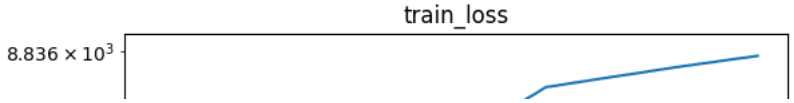
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 5e-06 , minimum_RMSE: 1032.17 , epoch: 1000 , test_loss: 1037 , train_loss

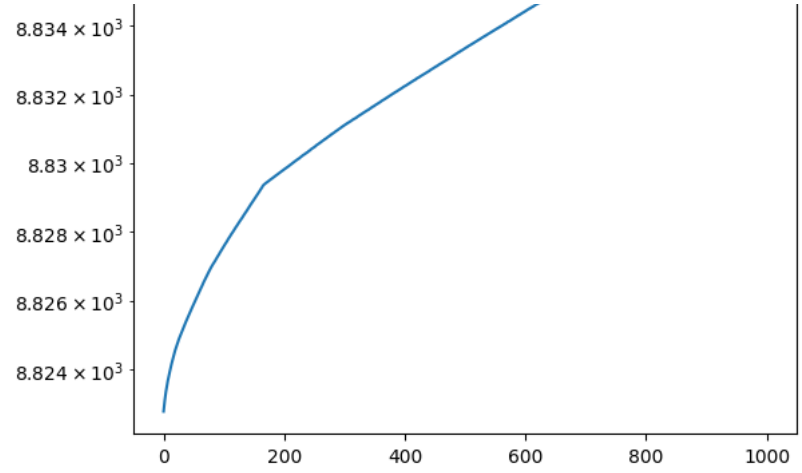




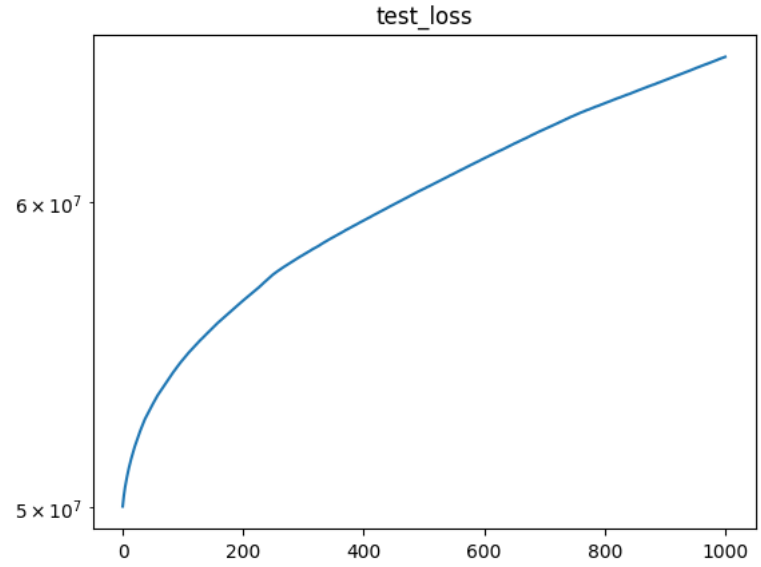
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-07 , minimum_RMSE: 53423.01 , epoch: 1000 , test_loss: 53423 , train_loss

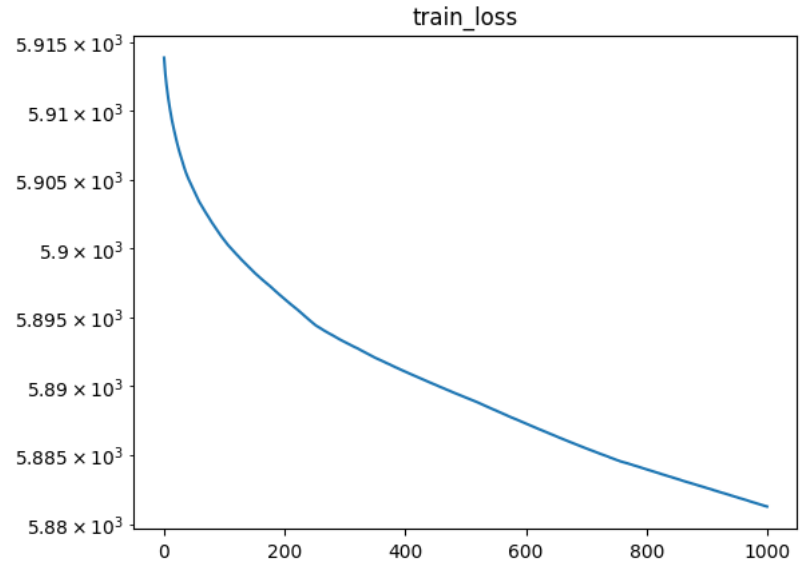




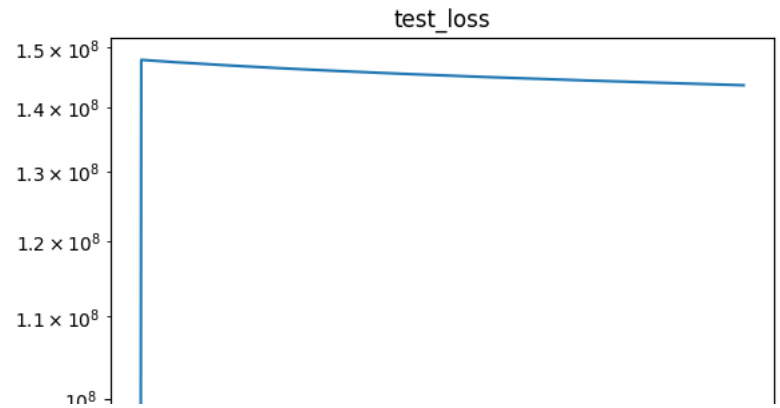


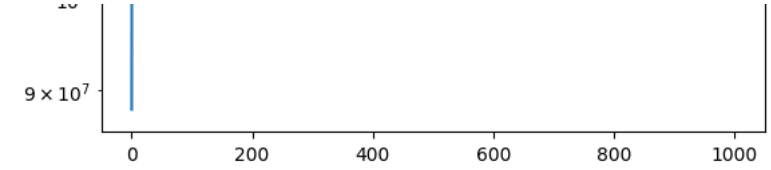
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 5e-07 , minimum_RMSE: 7073.55 , epoch: 1000 , test_loss: 8091 , train_loss:

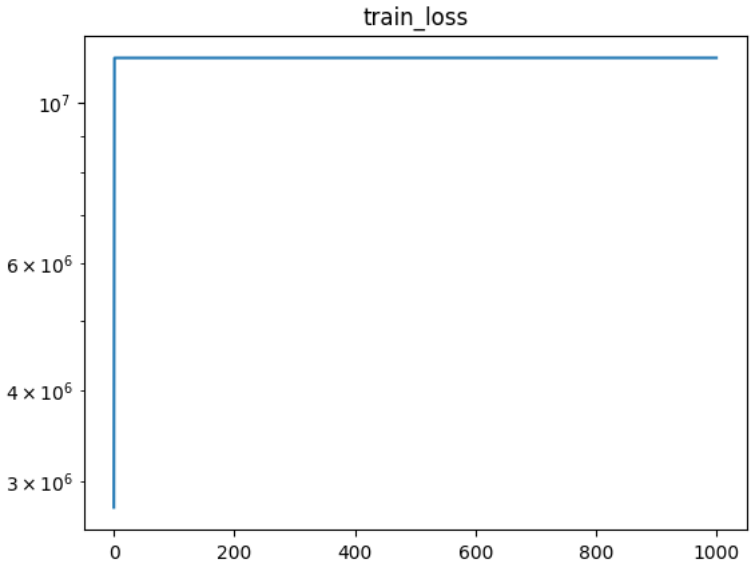




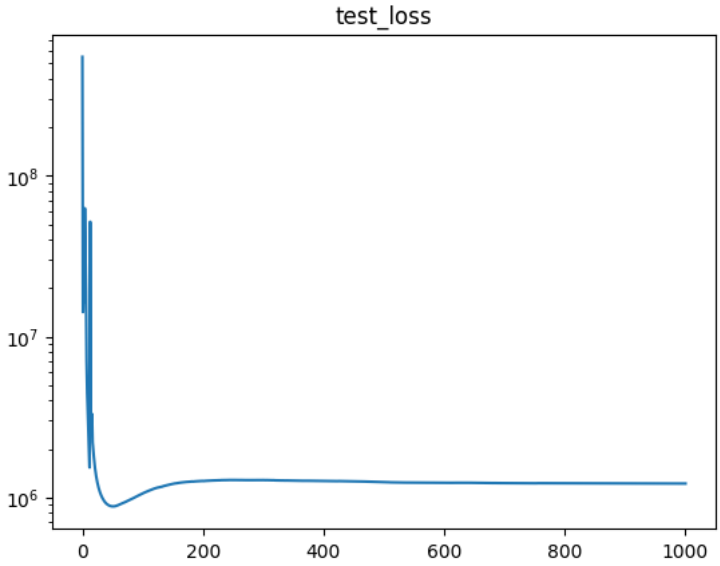
optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 0.1 , minimum_RMSE: 9382.73 , epoch: 1000 , test_loss: 11982 , train_loss:

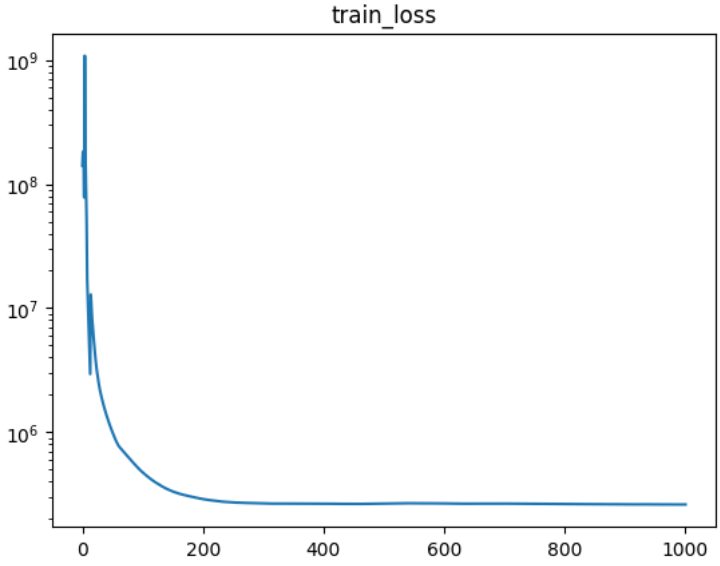




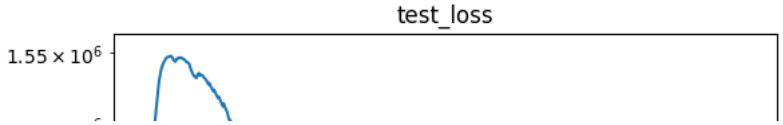


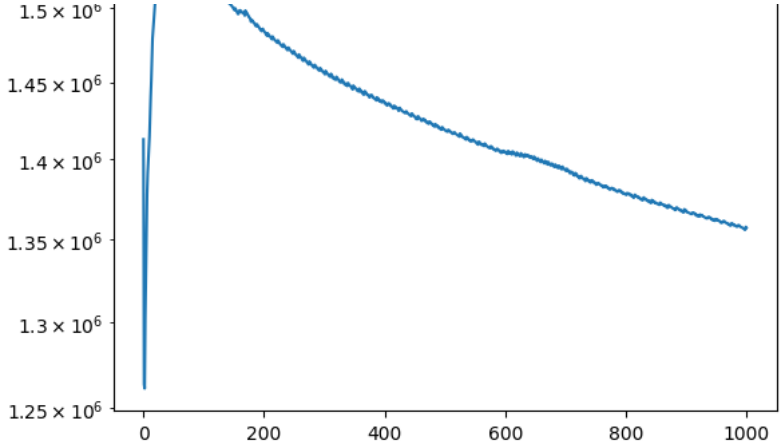
optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 0.5 , minimum_RMSE: 937.01 , epoch: 1000 , test_loss: 1103 , train_loss:

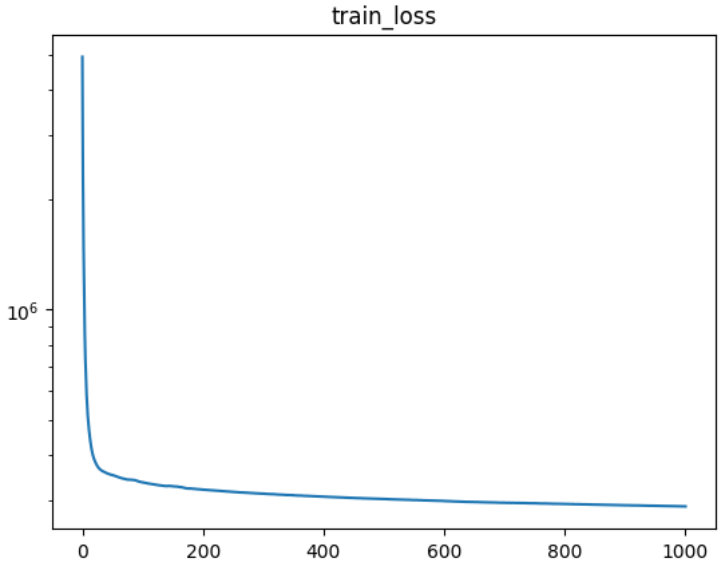




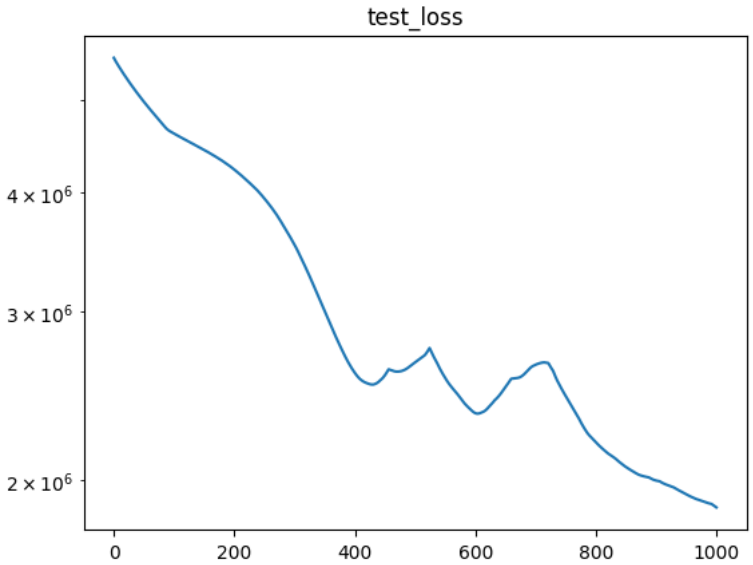
optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 0.01 , minimum_RMSE: 1123.16 , epoch: 1000 , test_loss: 1165 , train_loss:

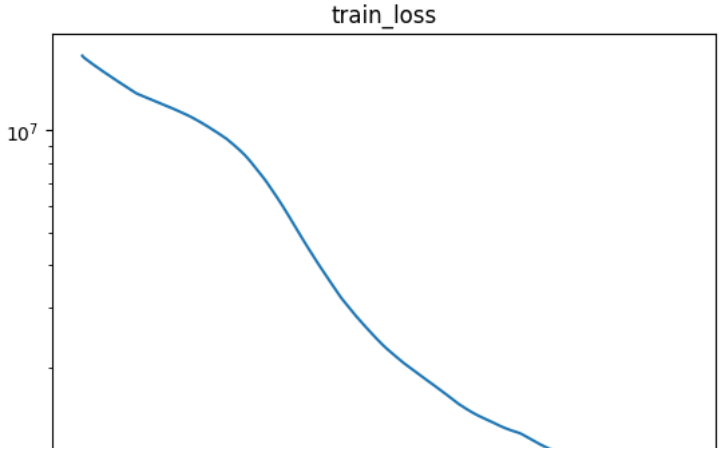


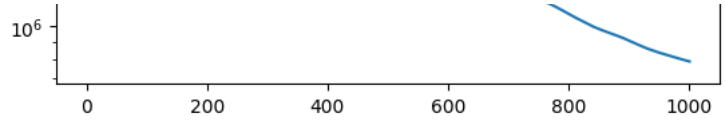




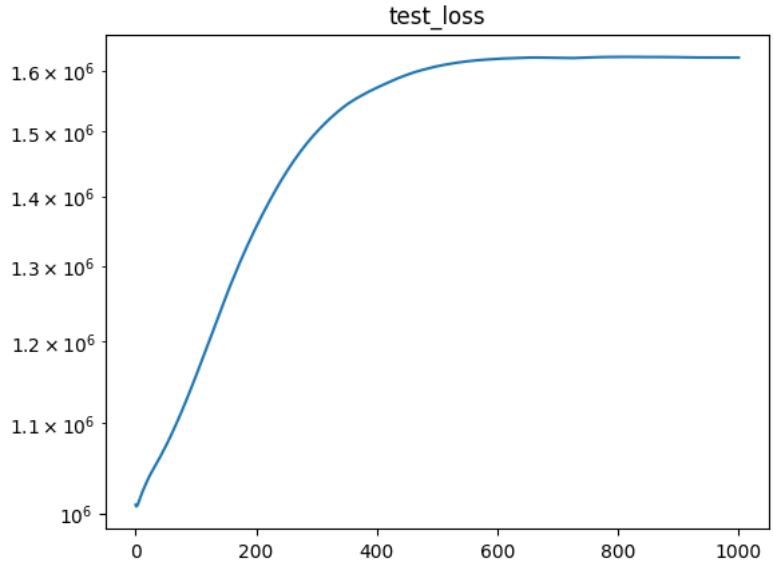
optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 0.05 , minimum_RMSE: 1368.86 , epoch: 1000 , test_loss: 1368 , train_loss:

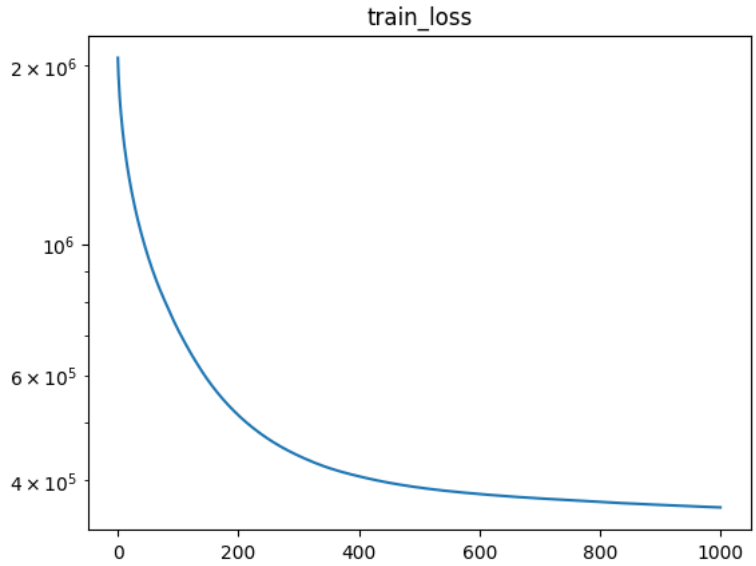




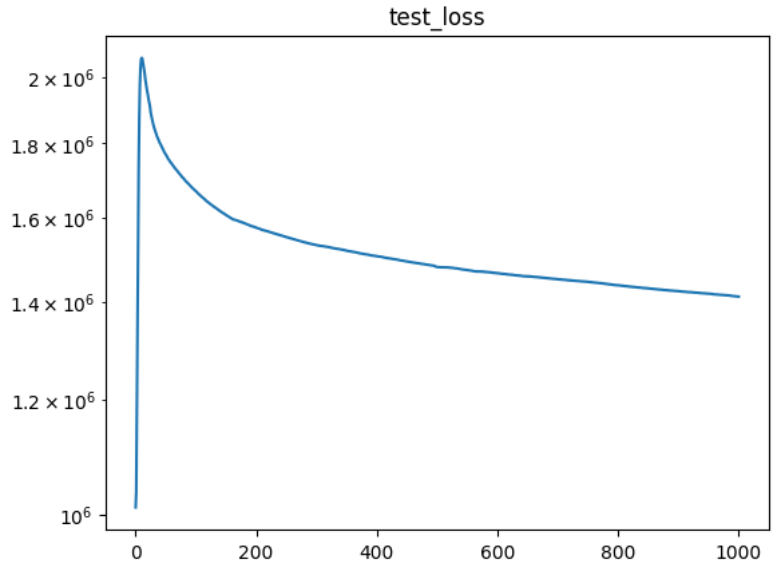


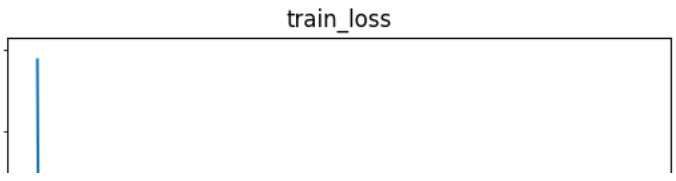
optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 0.001 , minimum_RMSE: 1003.61 , epoch: 1000 , test_loss: 1273 , train_loss:

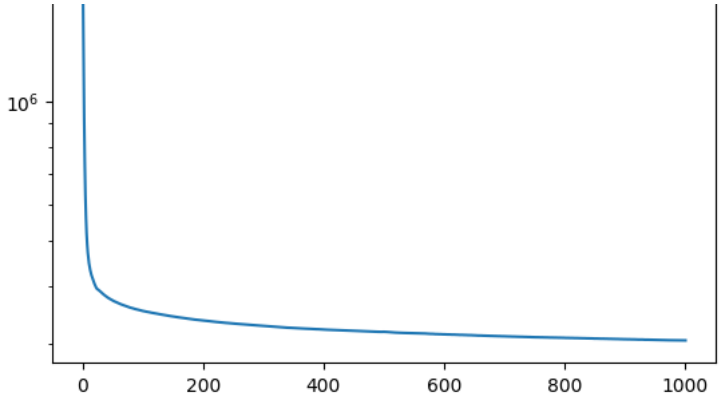




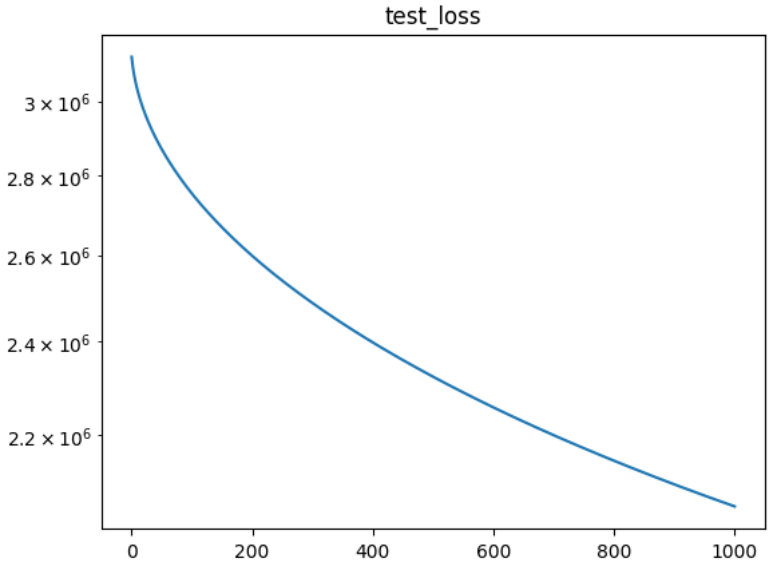
optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 0.005 , minimum_RMSE: 1005.49 , epoch: 1000 , test_loss: 1188 , train_loss:

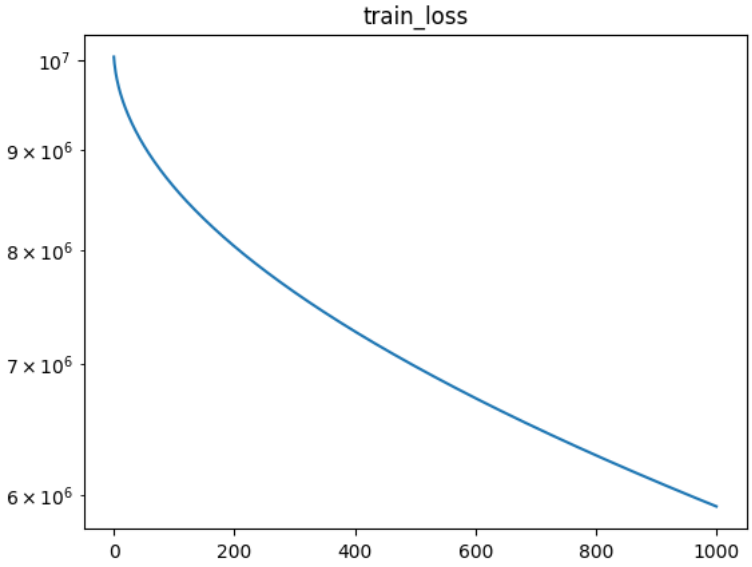




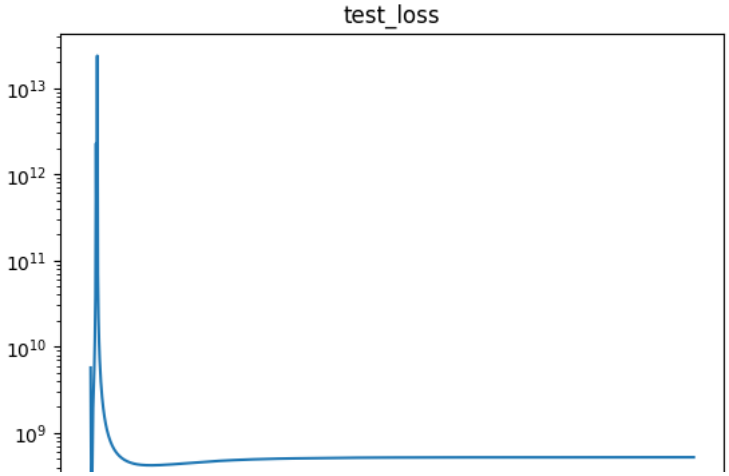


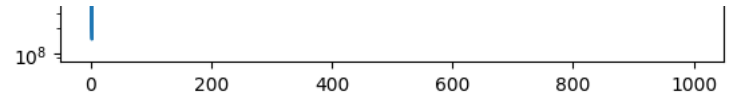
optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 0.0001 , minimum_RMSE: 1435.02 , epoch: 1000 , test_loss: 1435 , train_loss

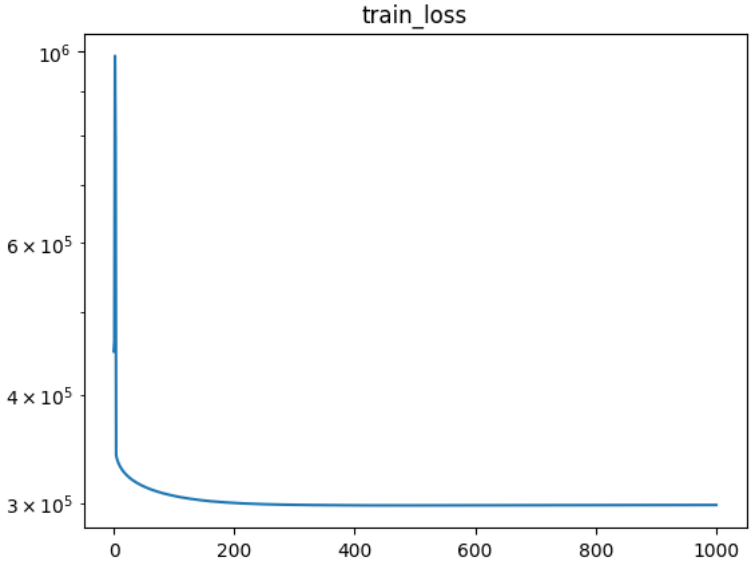




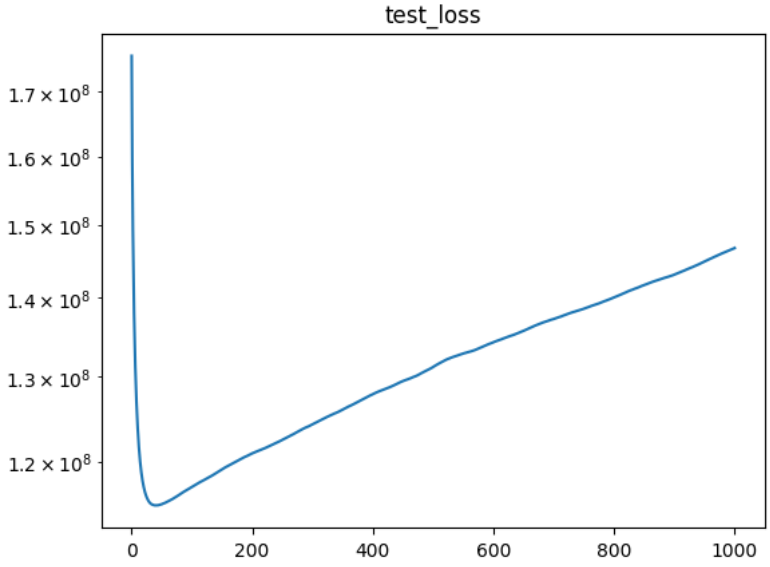
optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 0.0005 , minimum_RMSE: 12155.32 , epoch: 1000 , test_loss: 22842 , train_los

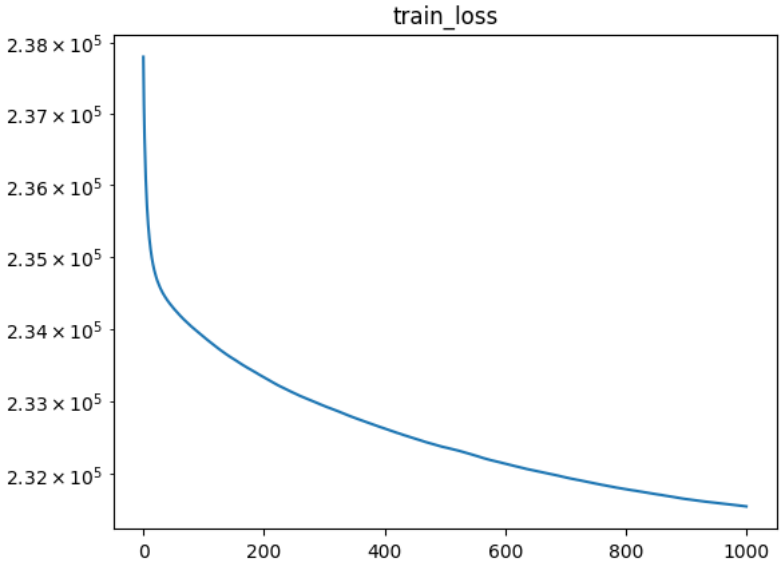




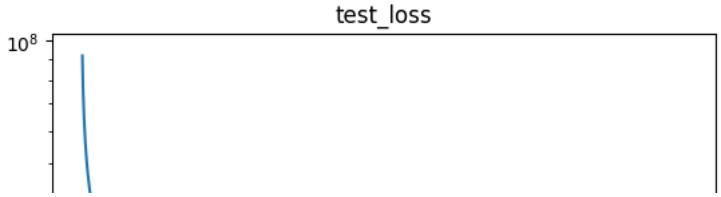


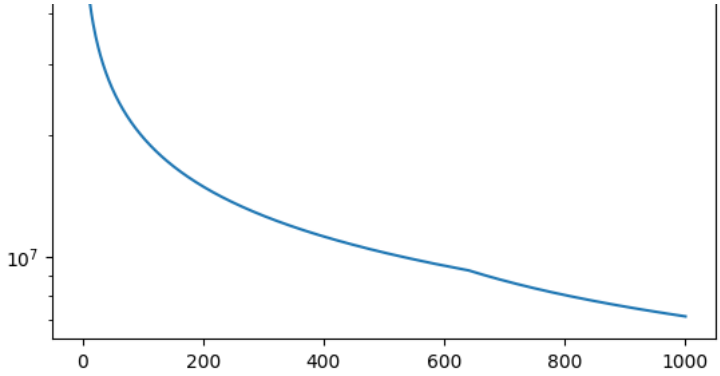
optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-05 , minimum_RMSE: 10731.15 , epoch: 1000 , test_loss: 12114 , train_loss

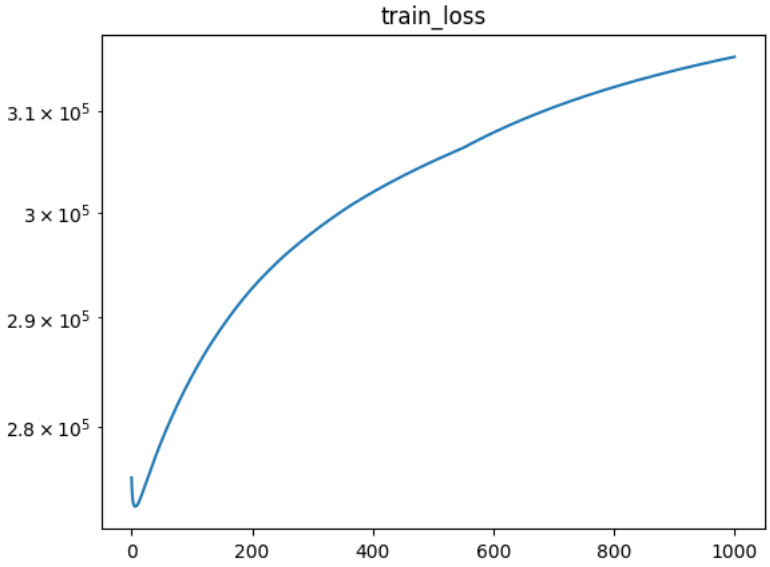




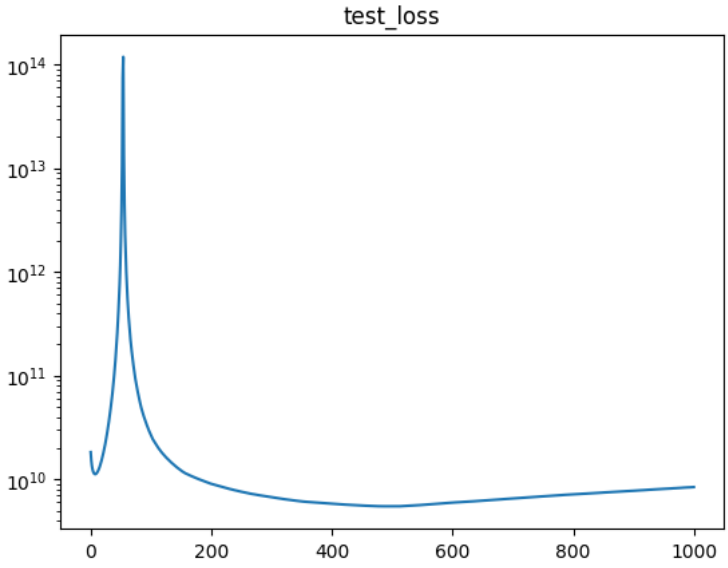
optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 5e-05 , minimum_RMSE: 2668.87 , epoch: 1000 , test_loss: 2668 , train_loss

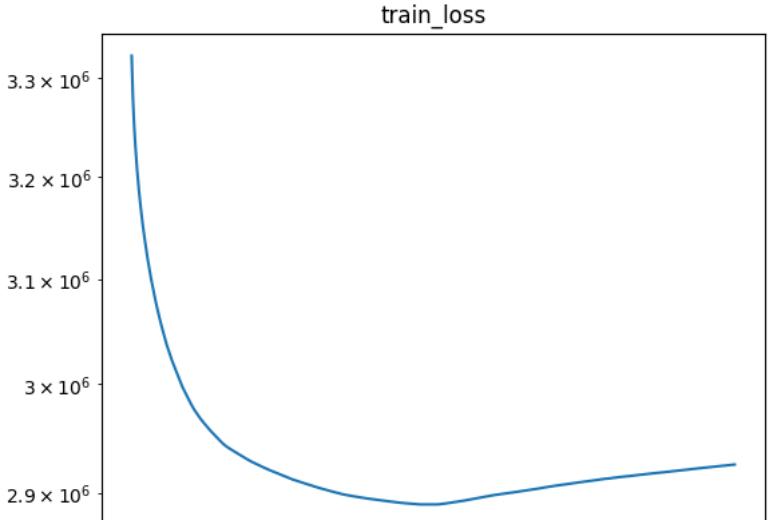




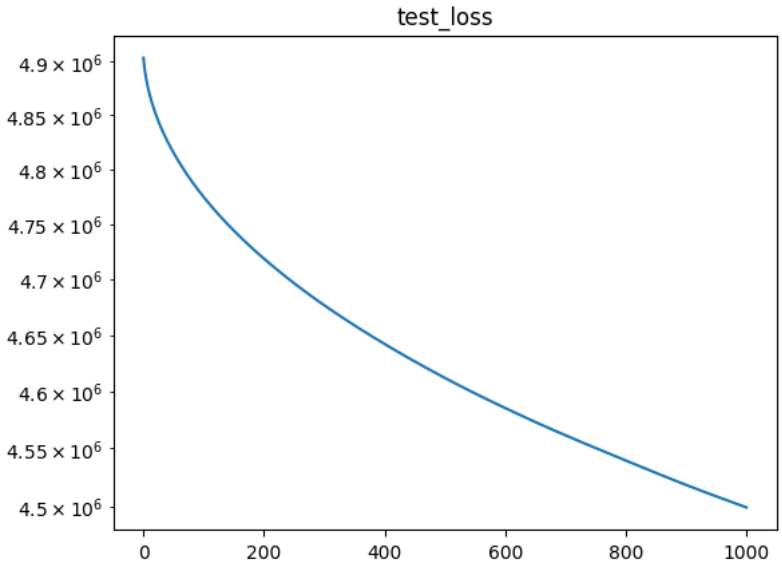


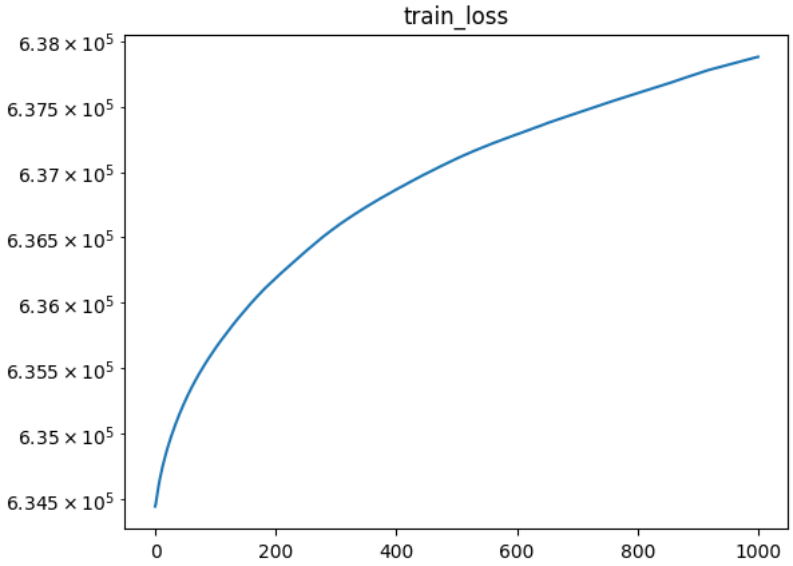
optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-06 , minimum_RMSE: 74035.47 , epoch: 1000 , test_loss: 91762 , train_loss



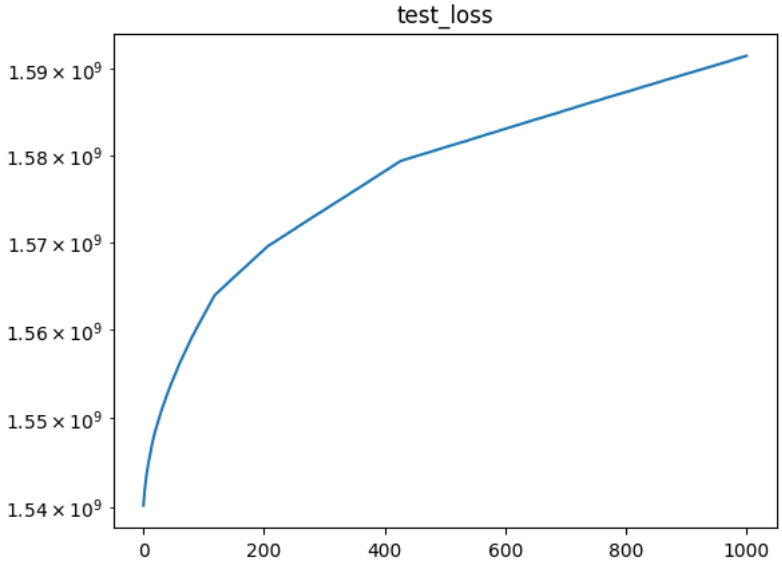


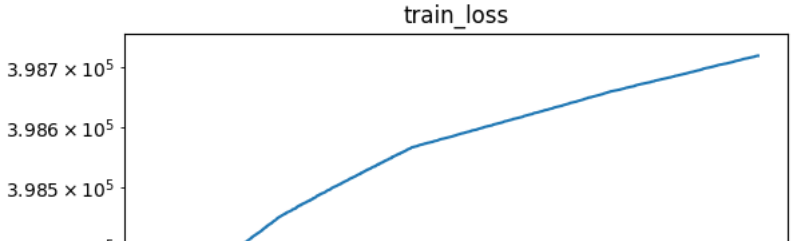
optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 5e-06 , minimum_RMSE: 2121.09 , epoch: 1000 , test_loss: 2121 , train_loss

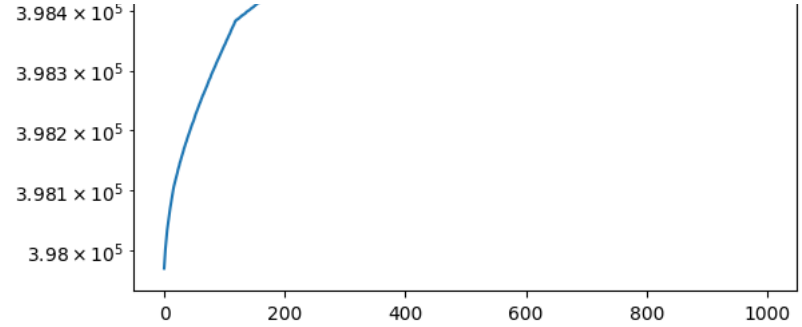




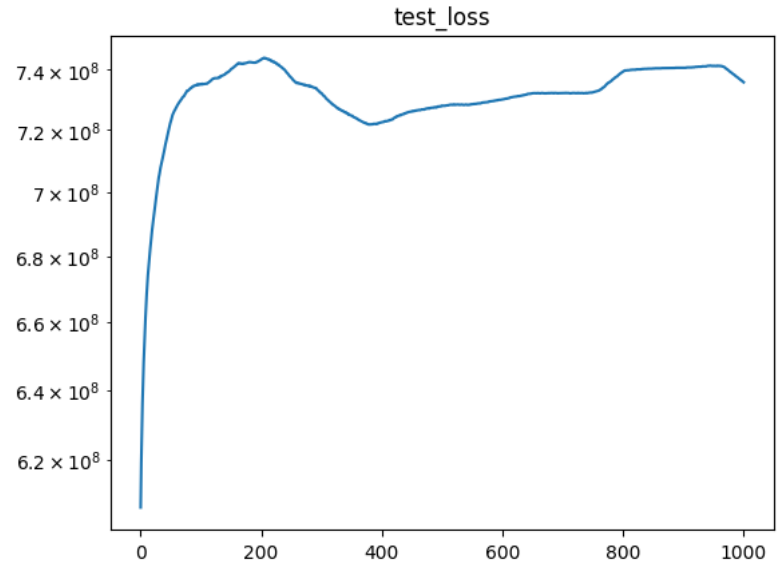
optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-07 , minimum_RMSE: 39245.16 , epoch: 1000 , test_loss: 39893 , train_loss

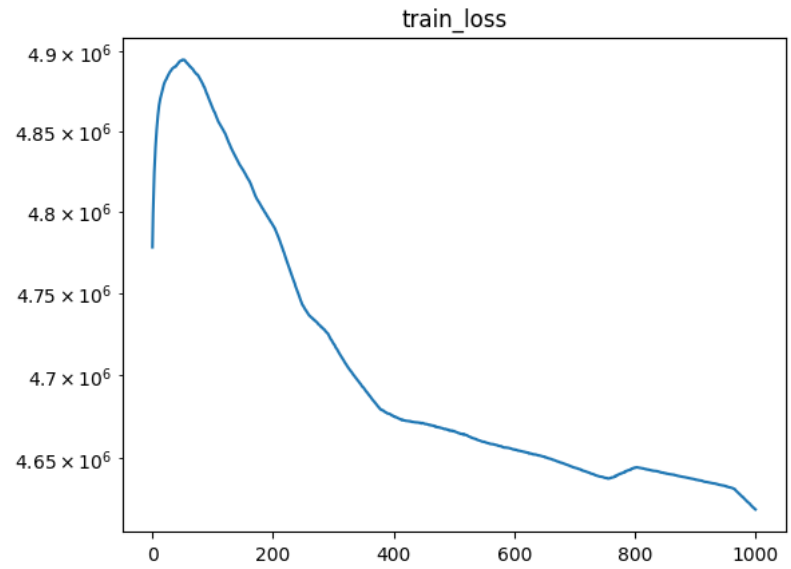




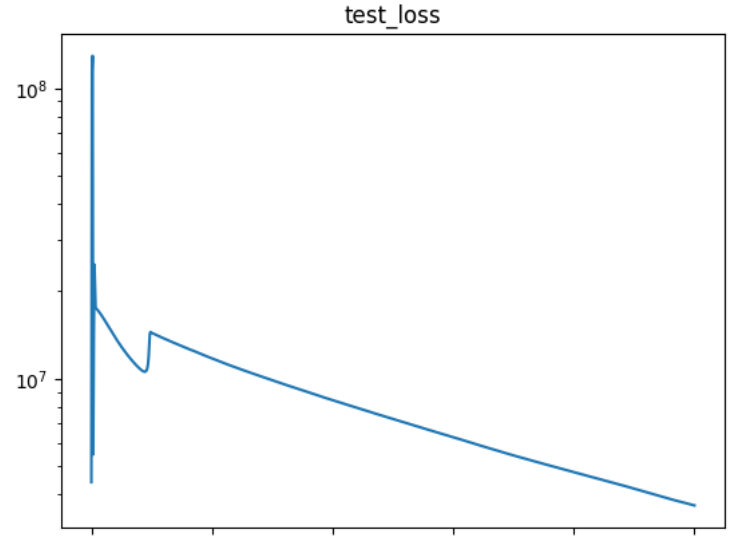


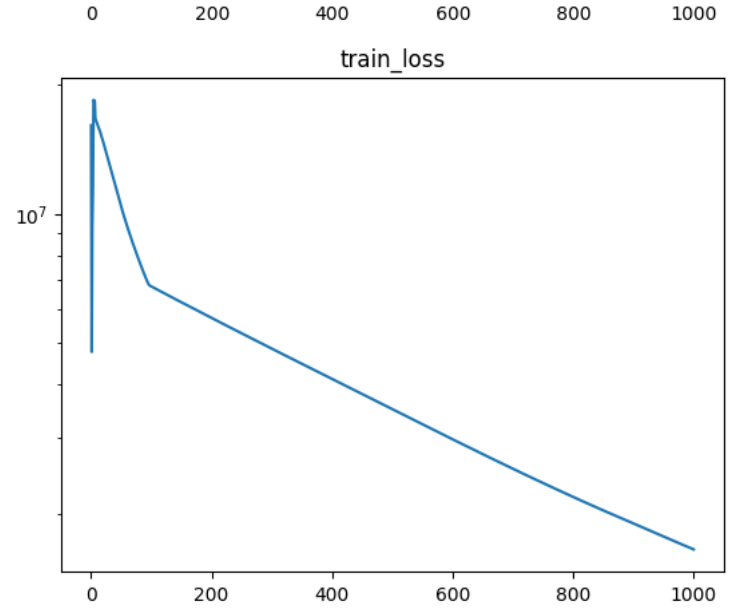
optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 5e-07 , minimum_RMSE: 24632.42 , epoch: 1000 , test_loss: 27124 , train_loss



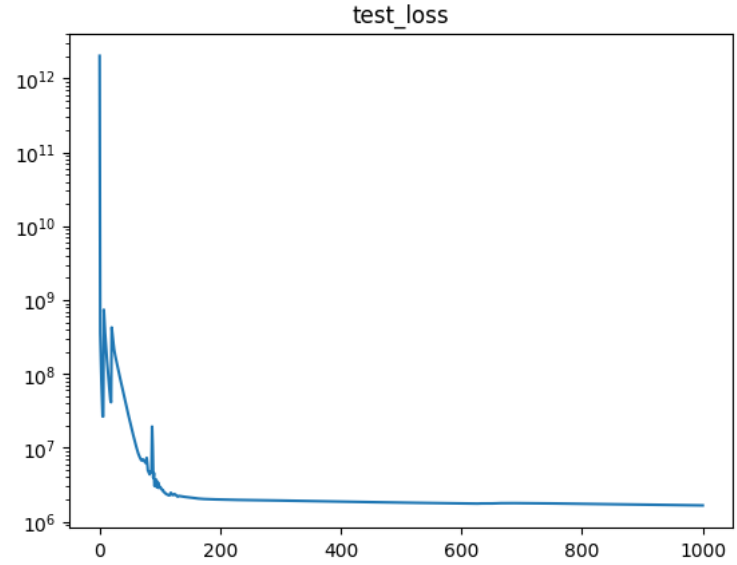


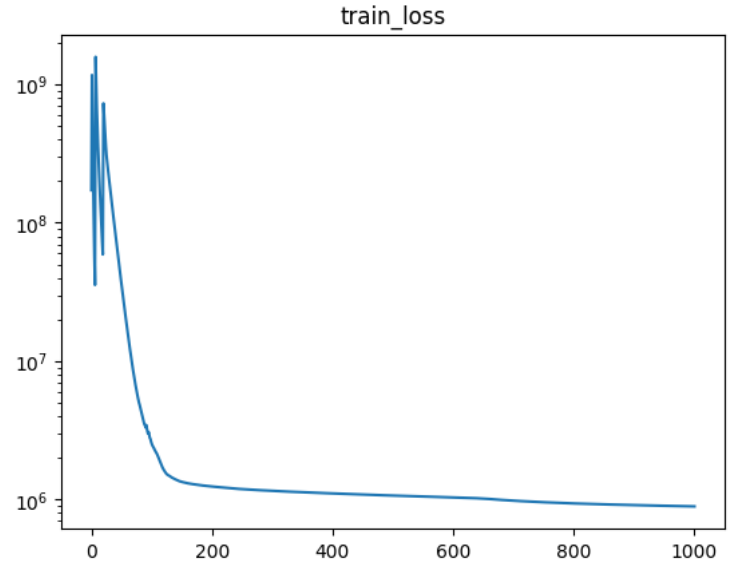
optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 0.1 , minimum_RMSE: 1912.32 , epoch: 1000 , test_loss: 1912 , train_loss:



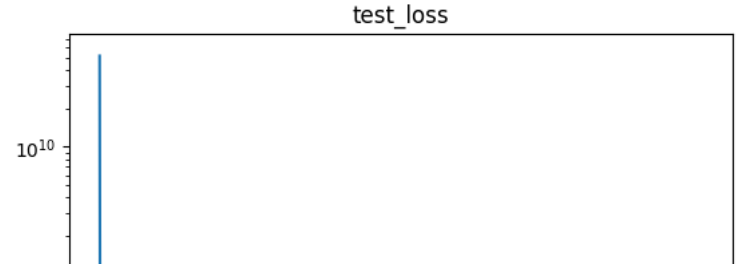


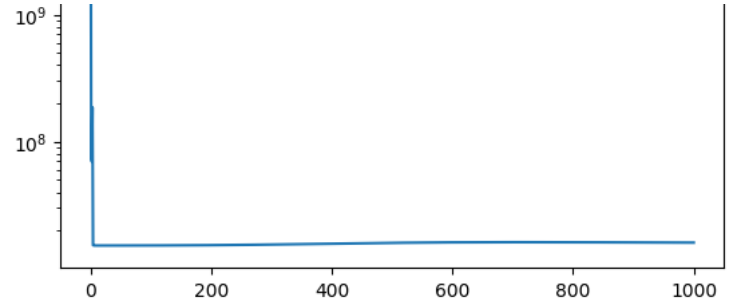
optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 0.5 , minimum_RMSE: 1285.88 , epoch: 1000 , test_loss: 1285 , train_loss:

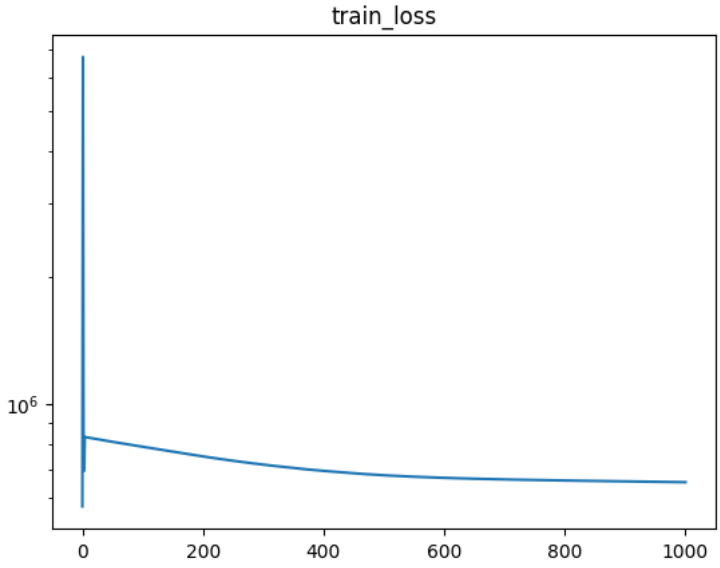




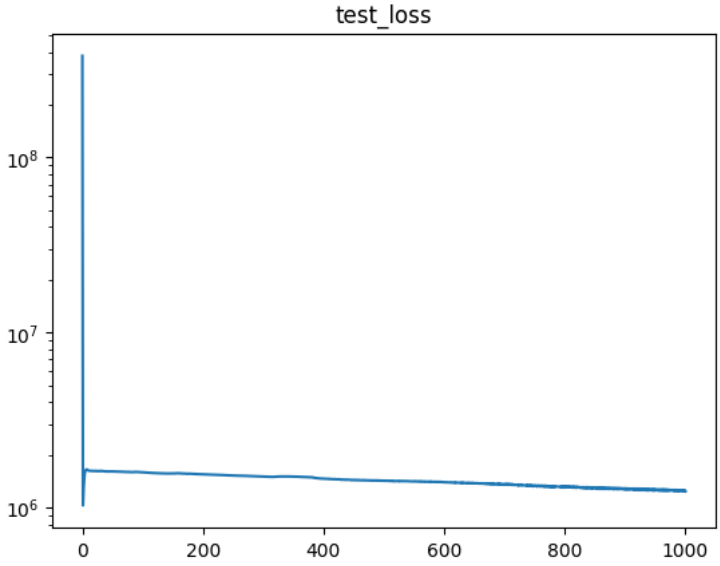
optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 0.01 , minimum_RMSE: 3899.51 , epoch: 1000 , test_loss: 4003 , train_loss

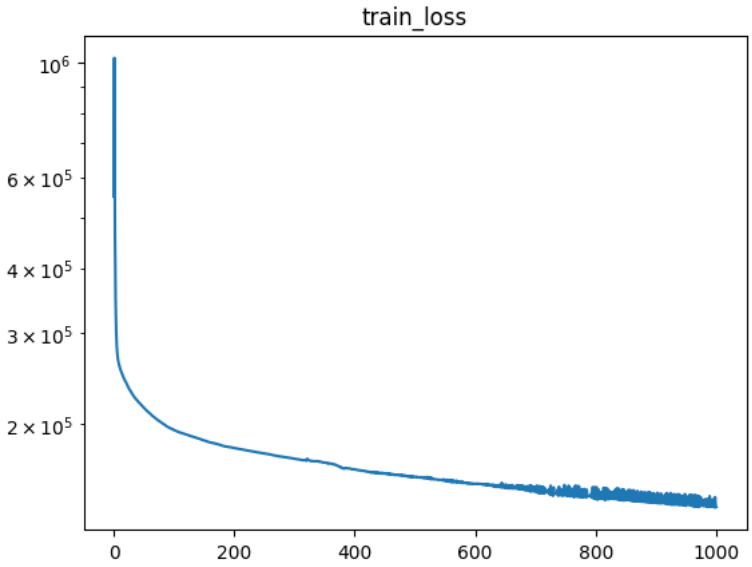




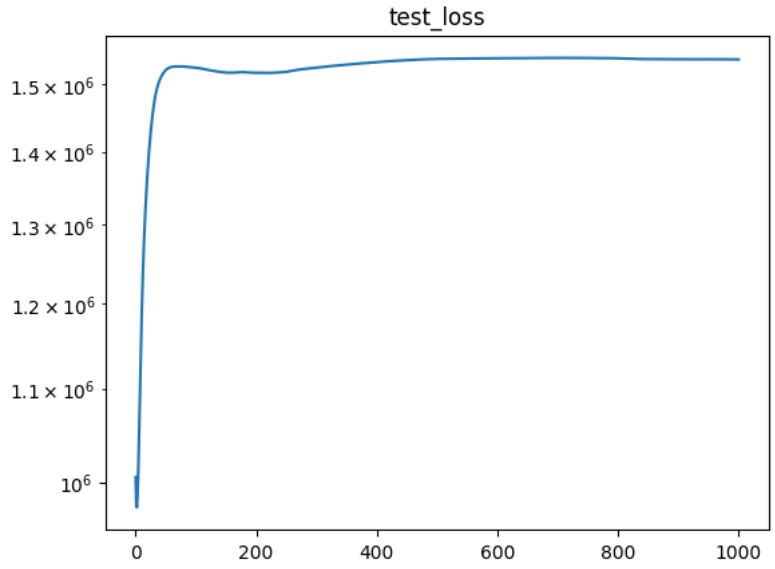


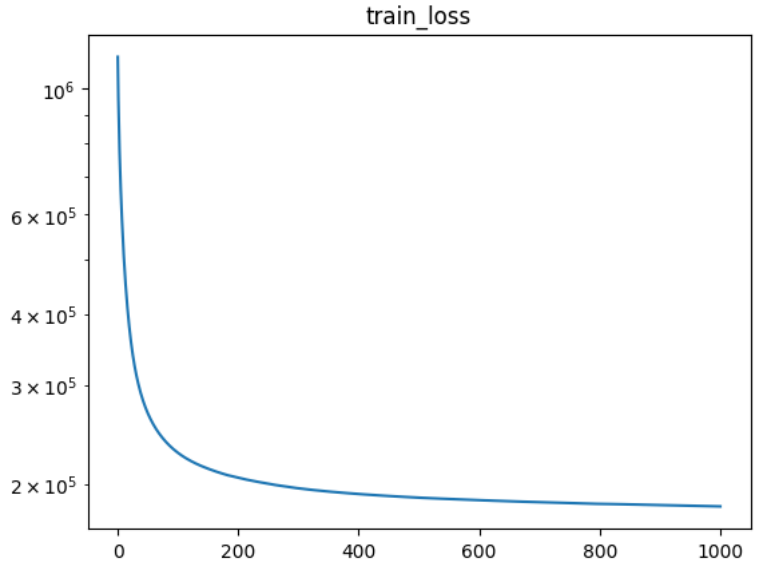
optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 0.05 , minimum_RMSE: 1012.57 , epoch: 1000 , test_loss: 1111 , train_loss



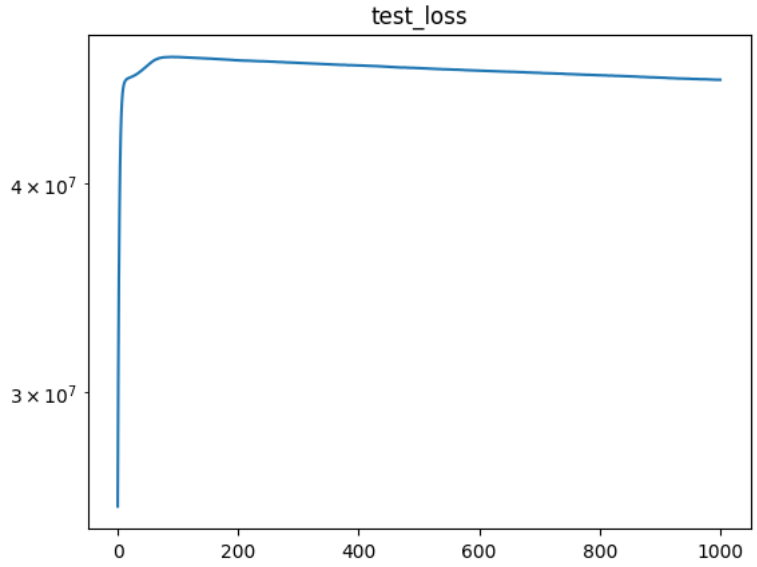


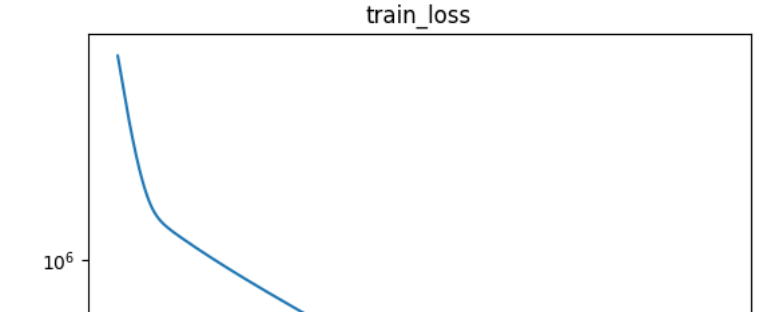
optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 0.001 , minimum_RMSE: 987.49 , epoch: 1000 , test_loss: 1240 , train_loss

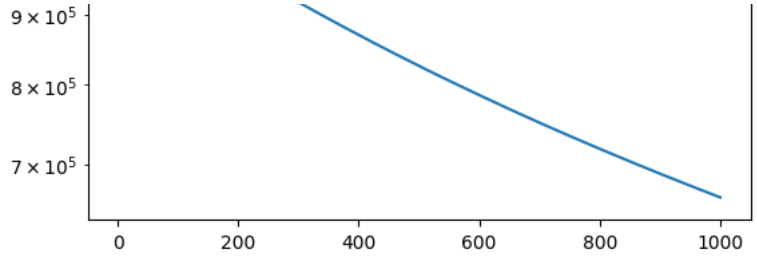




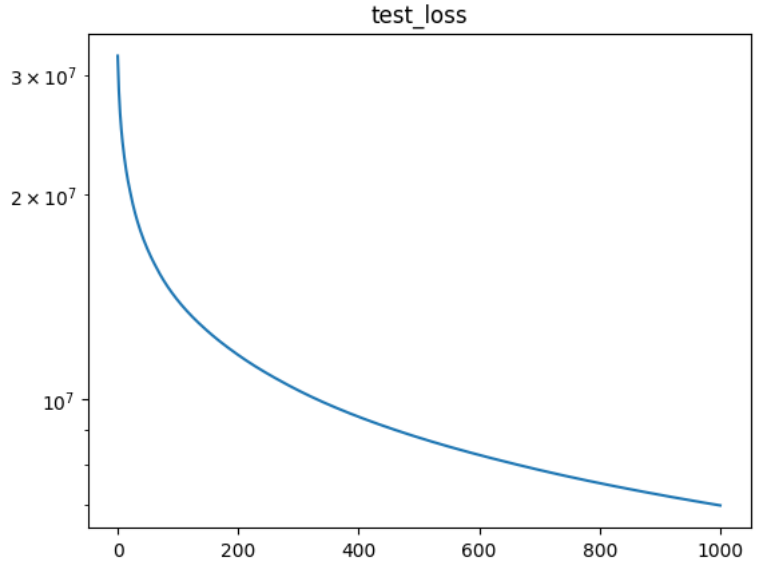
optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 0.005 , minimum_RMSE: 5063.07 , epoch: 1000 , test_loss: 6792 , train_loss

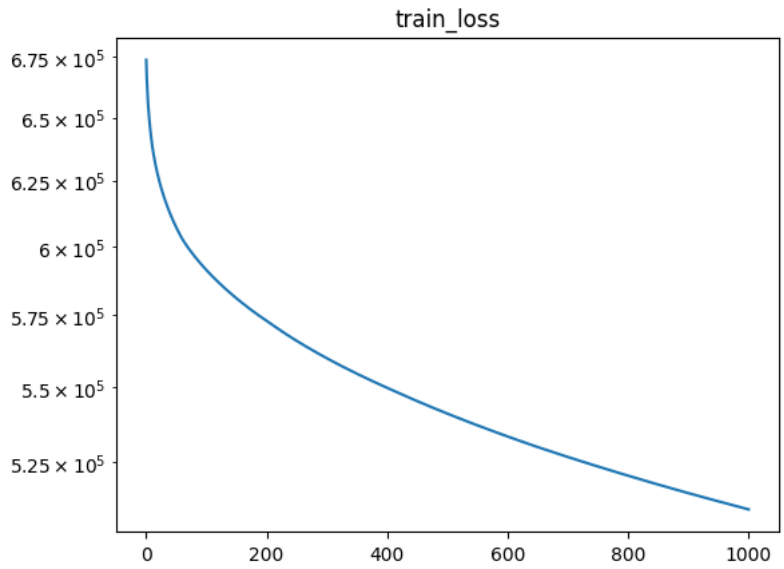




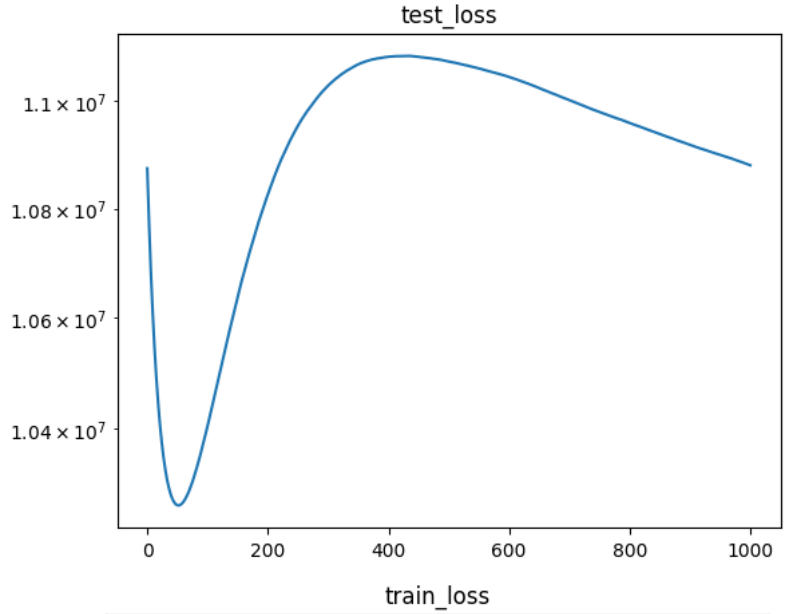


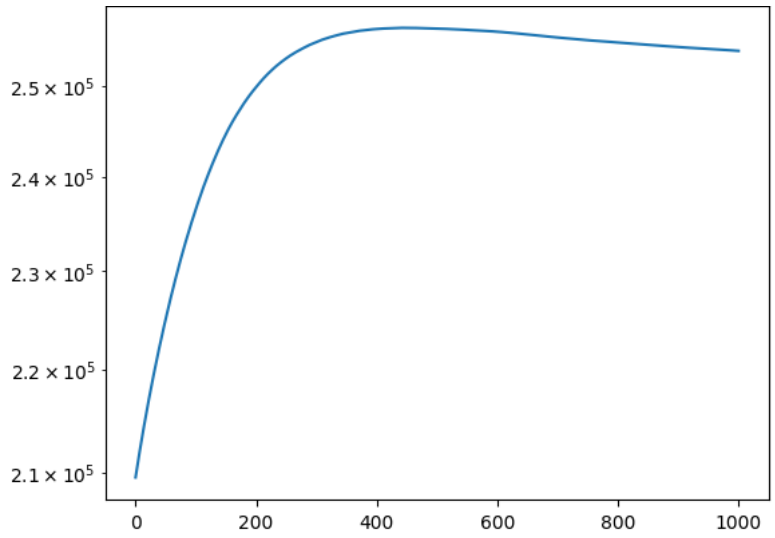
optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 0.0001 , minimum_RMSE: 2641.75 , epoch: 1000 , test_loss: 2641 , train_los



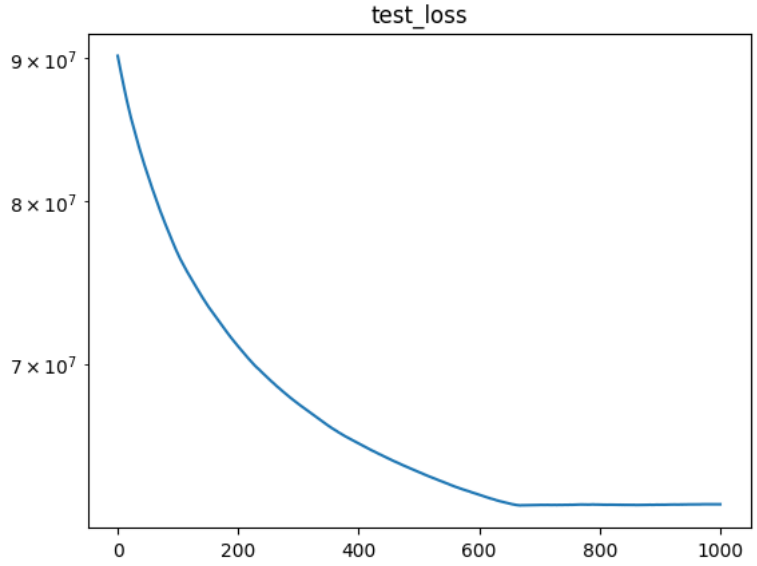


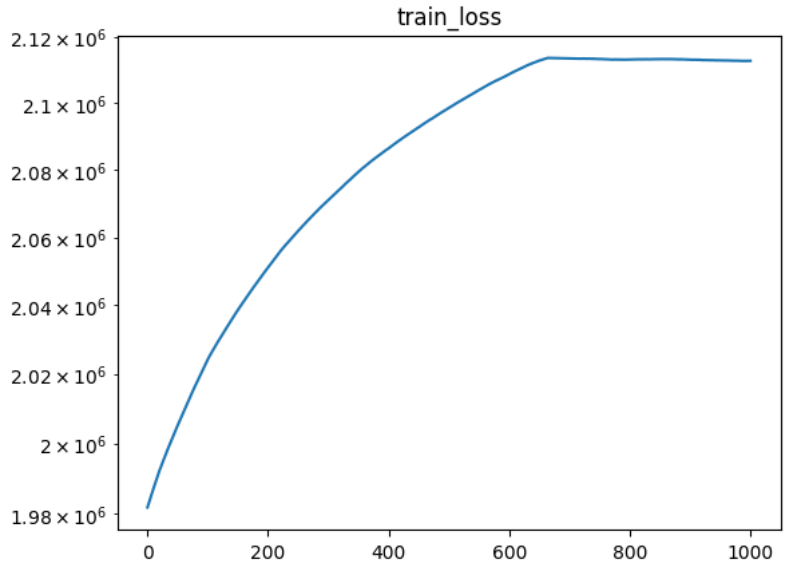
optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 0.0005 , minimum_RMSE: 3203.92 , epoch: 1000 , test_loss: 3298 , train_los



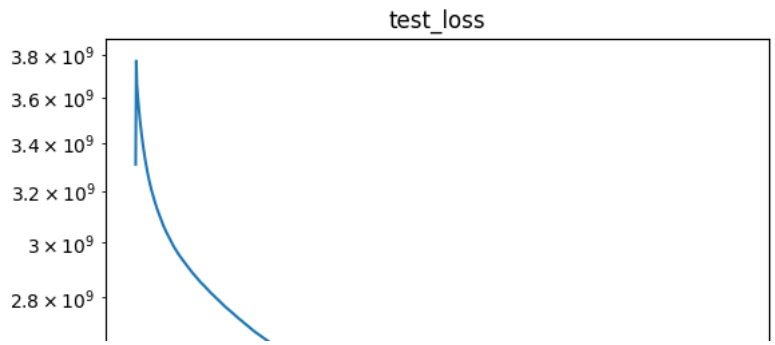


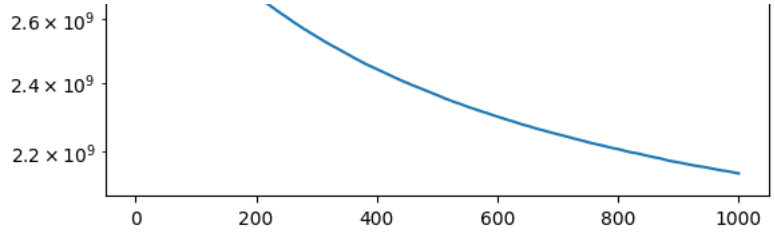
optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-05 , minimum_RMSE: 7893.49 , epoch: 1000 , test_loss: 7897 , train_loss

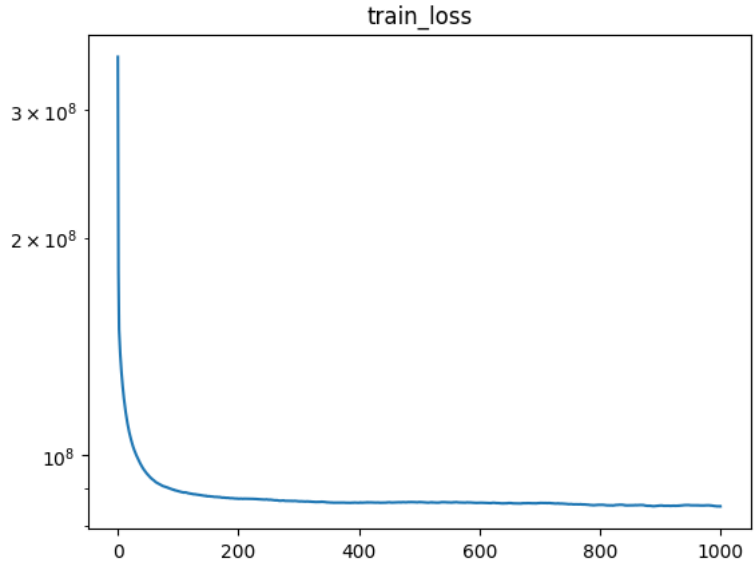




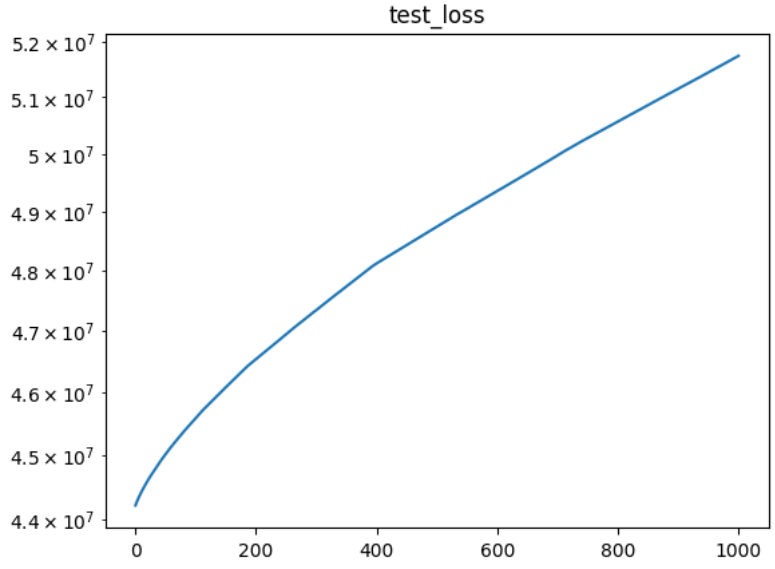
optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-05 , minimum_RMSE: 46276.57 , epoch: 1000 , test_loss: 46276 , train_los

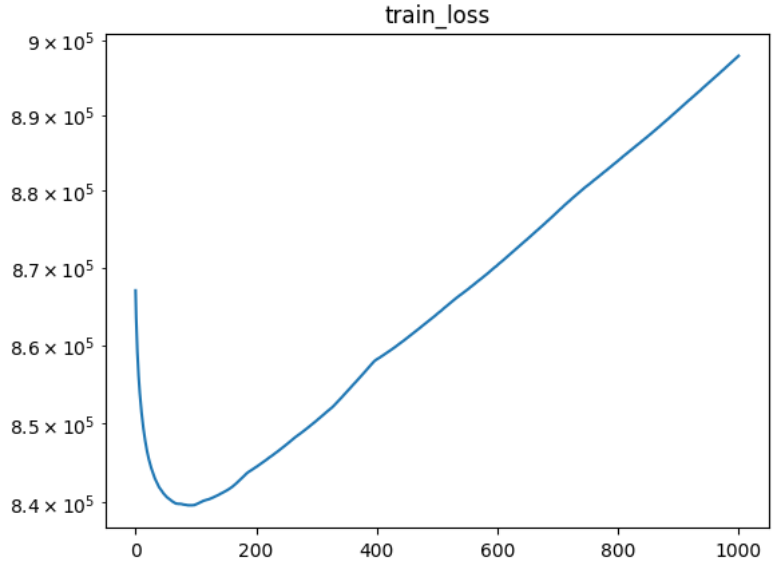




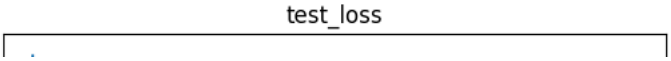


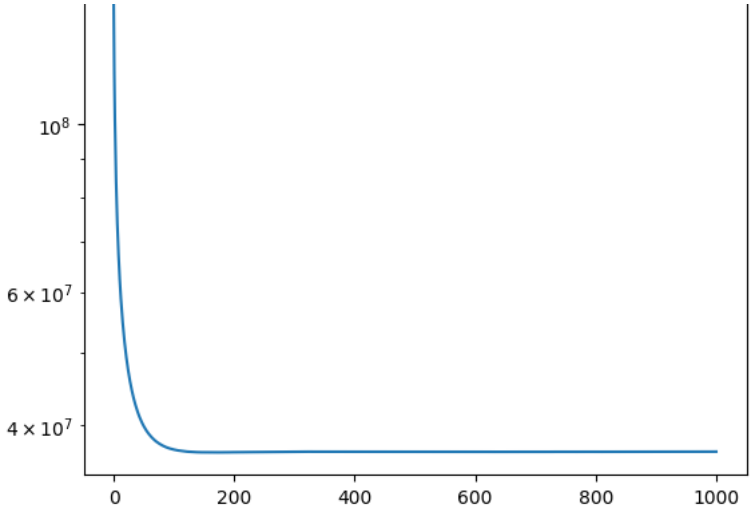
optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-06 , minimum_RMSE: 6649.63 , epoch: 1000 , test_loss: 7192 , train_loss

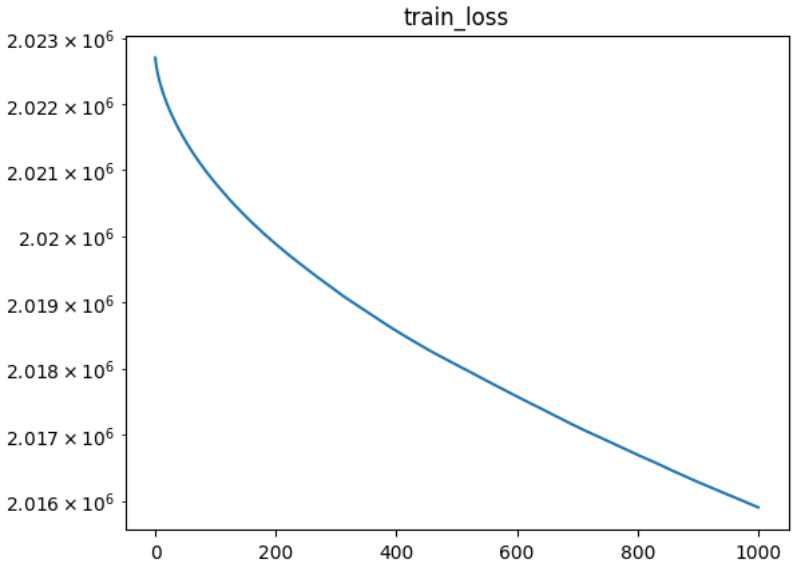




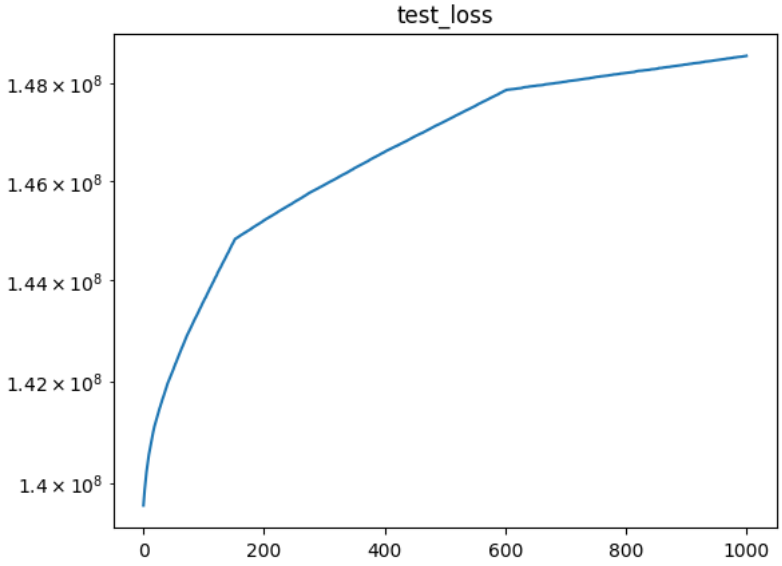
optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-06 , minimum_RMSE: 6070.40 , epoch: 1000 , test_loss: 6077 , train_loss

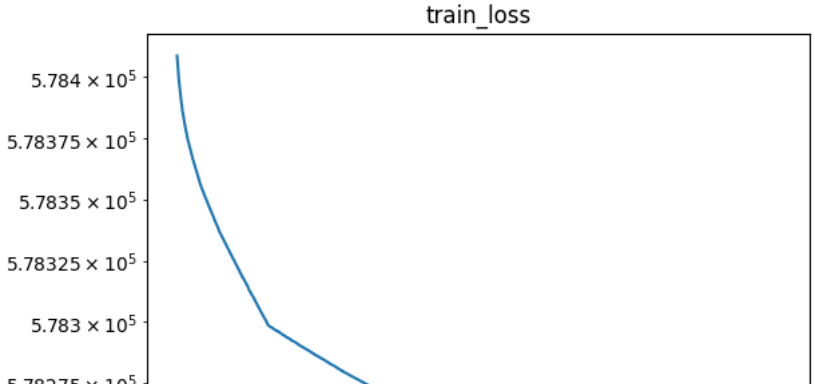


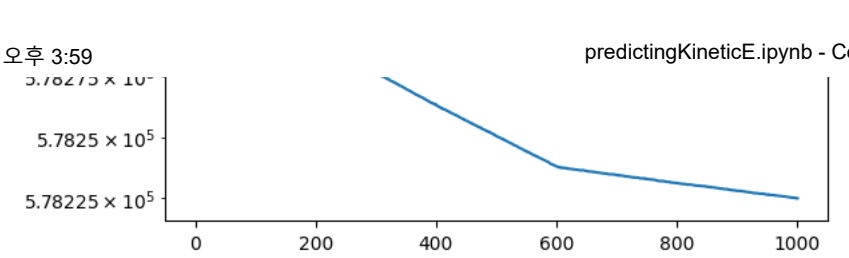




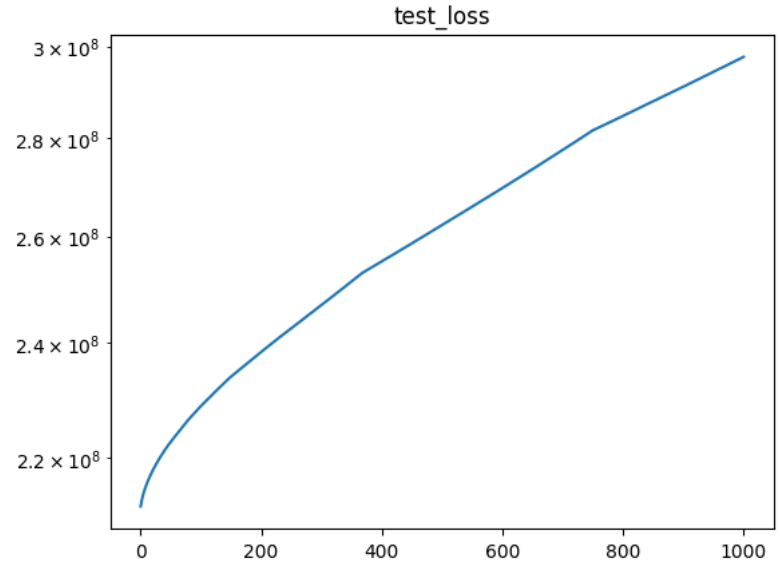
optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-07 , minimum_RMSE: 11813.93 , epoch: 1000 , test_loss: 12188 , train_los

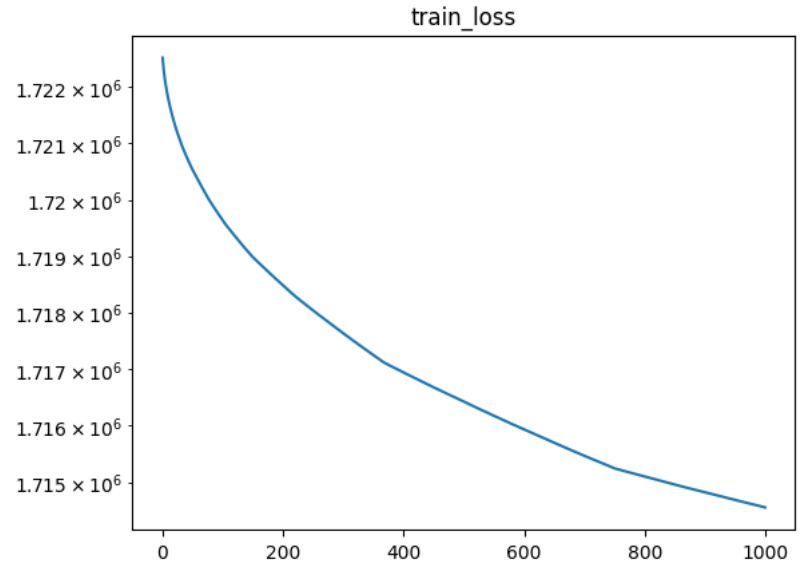




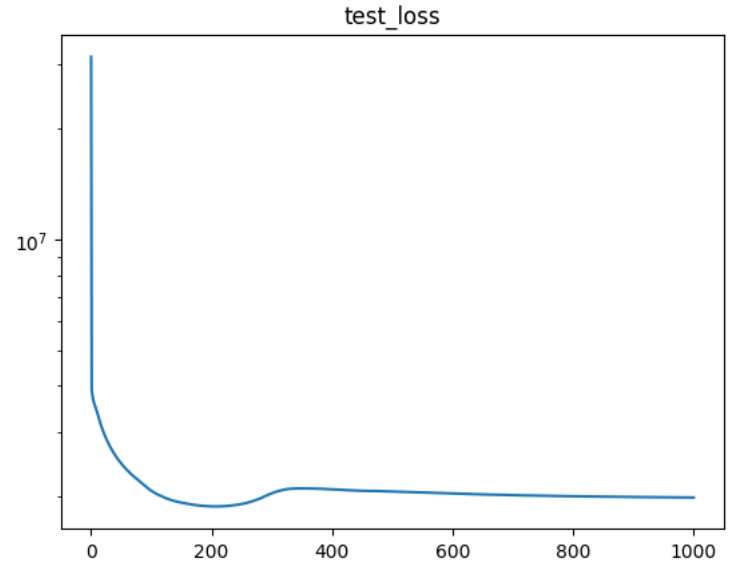


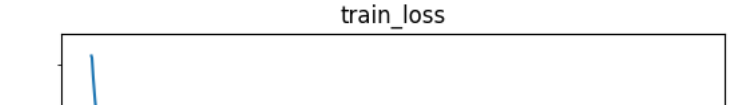
optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-07 , minimum_RMSE: 14565.80 , epoch: 1000 , test_loss: 17251 , train_los

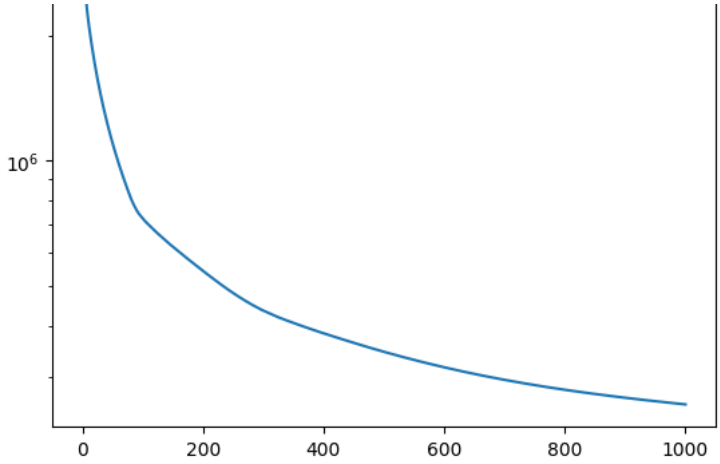




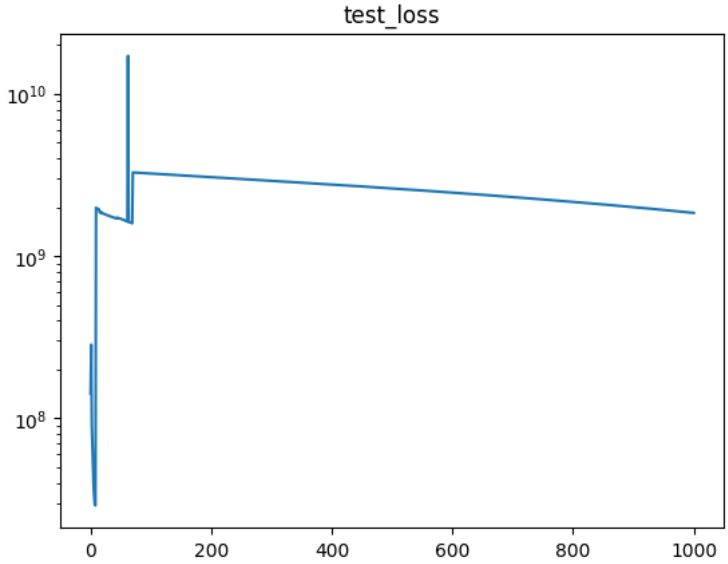
optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 0.1 , minimum_RMSE: 1371.31 , epoch: 1000 , test_loss: 1409 , train_loss:

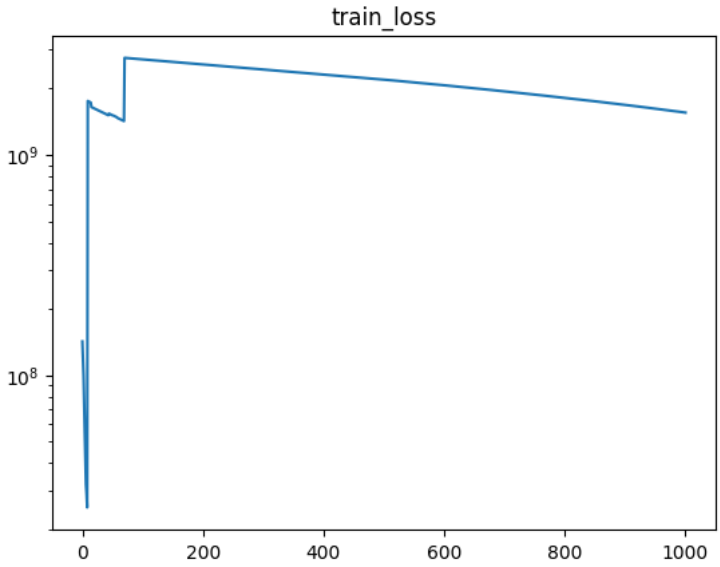




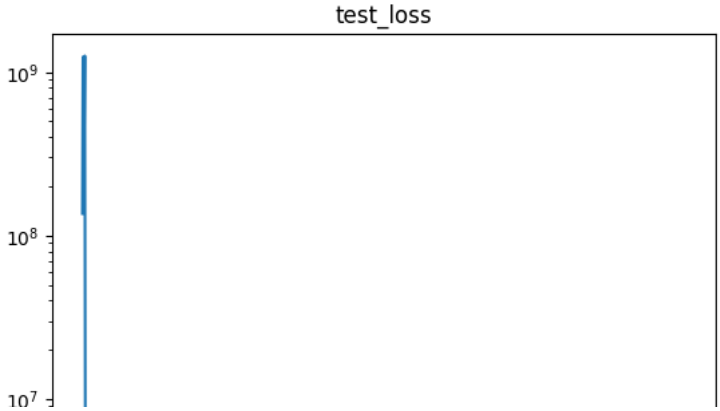


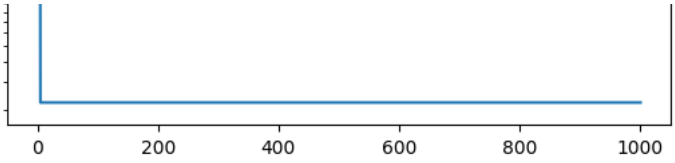
optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 0.5 , minimum_RMSE: 5386.56 , epoch: 1000 , test_loss: 42957 , train_loss:

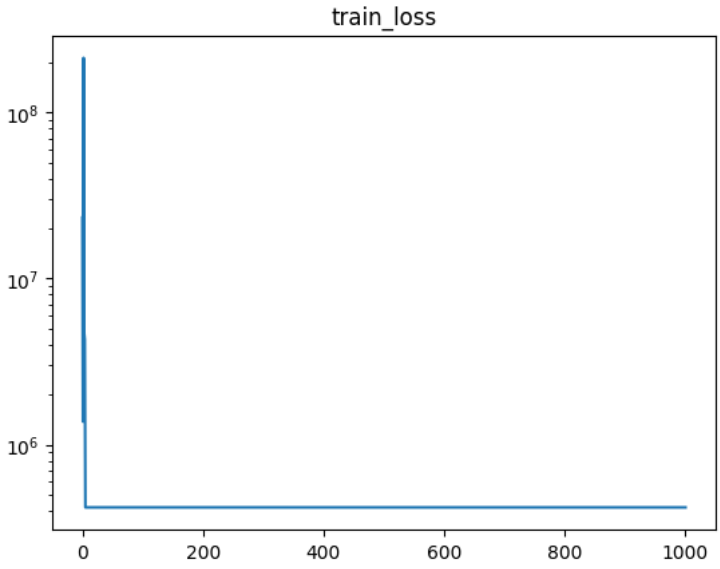




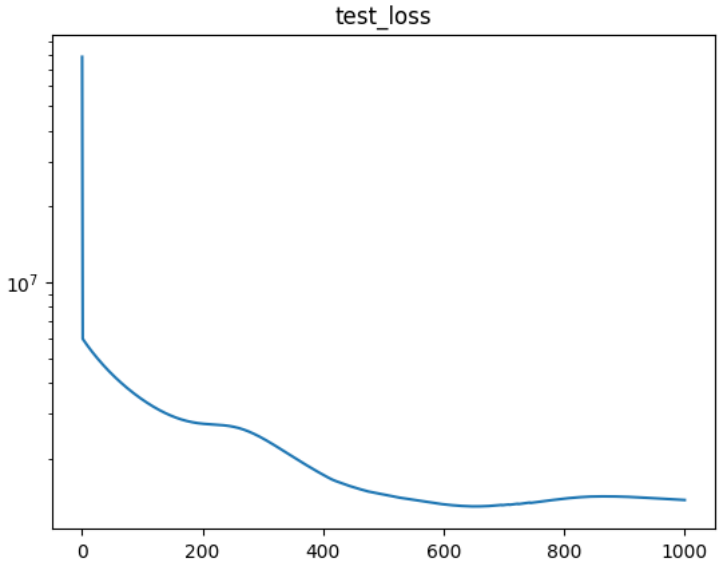
optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 0.01 , minimum_RMSE: 1493.77 , epoch: 1000 , test_loss: 1493 , train_loss

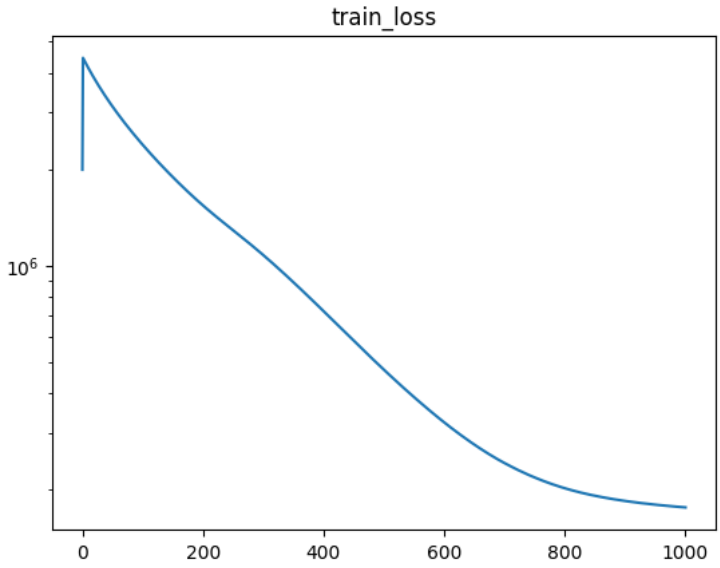




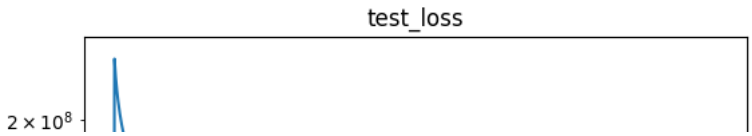


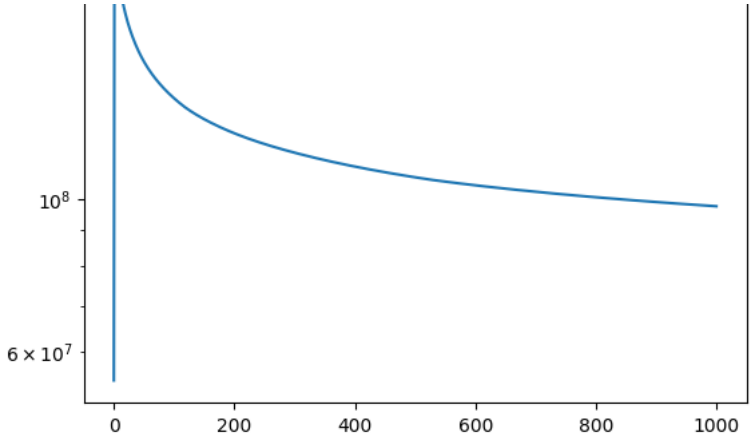
optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 0.05 , minimum_RMSE: 1135.50 , epoch: 1000 , test_loss: 1169 , train_loss

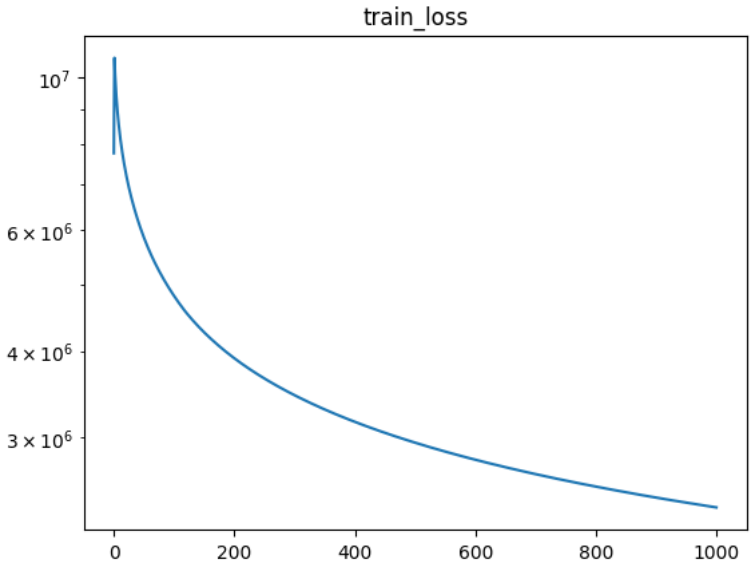




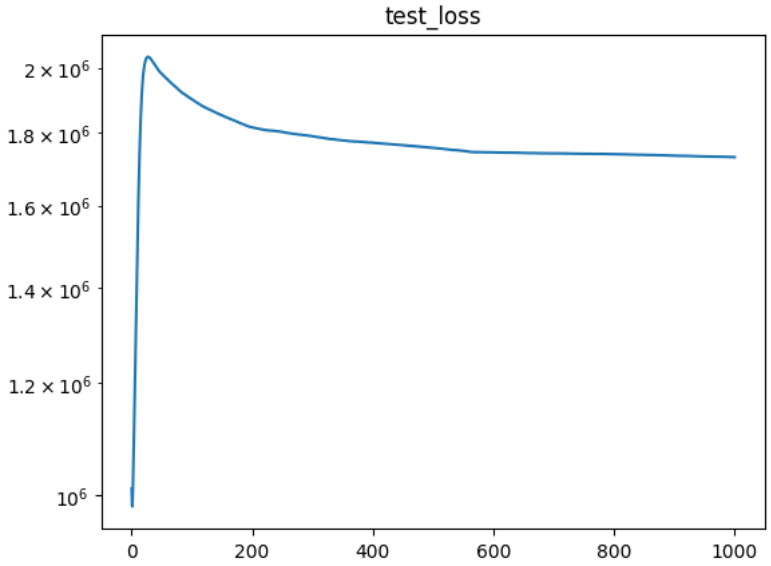
optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 0.001 , minimum_RMSE: 7383.28 , epoch: 1000 , test_loss: 9875 , train_loss

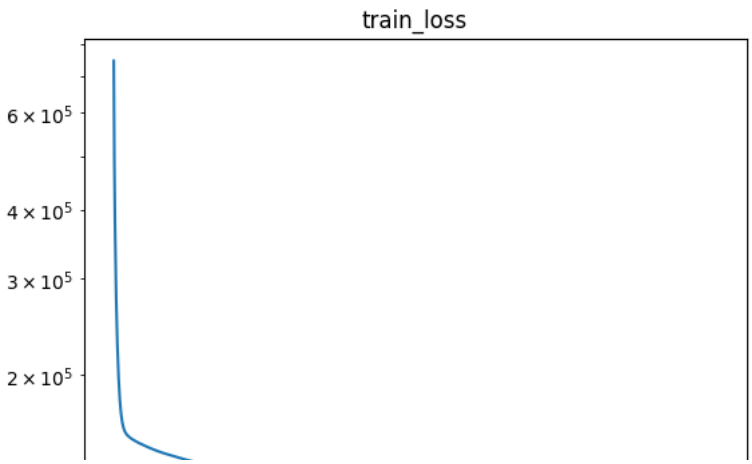


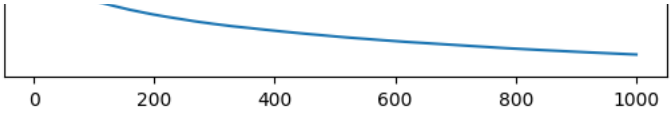




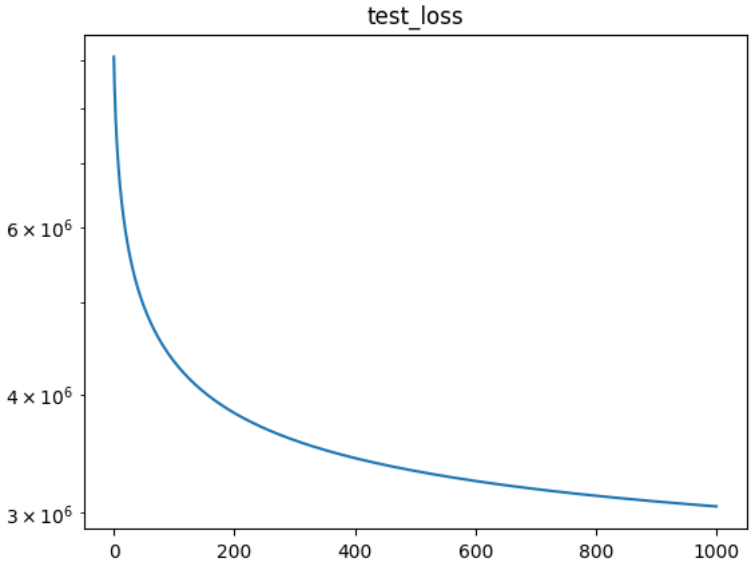
optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 0.005 , minimum_RMSE: 990.98 , epoch: 1000 , test_loss: 1315 , train_loss:

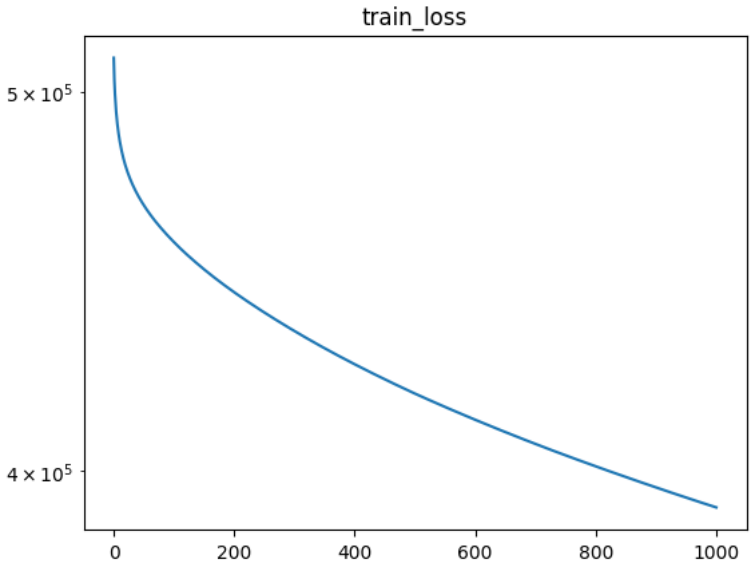




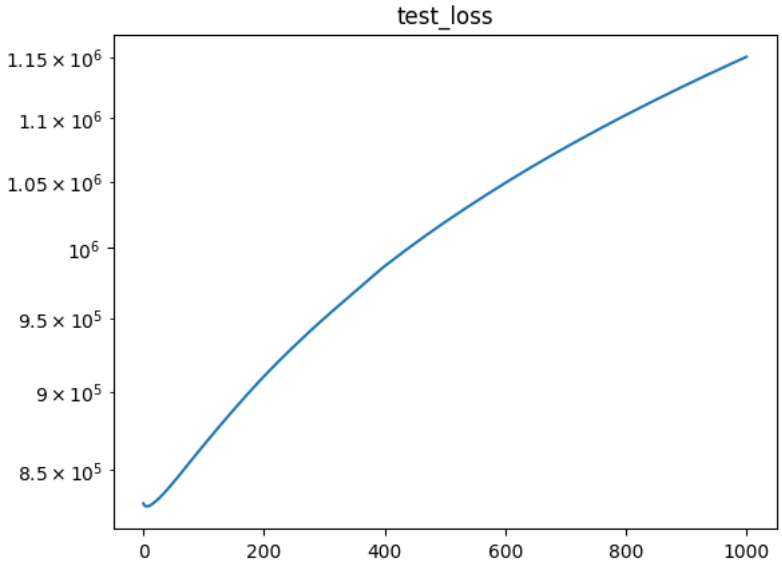


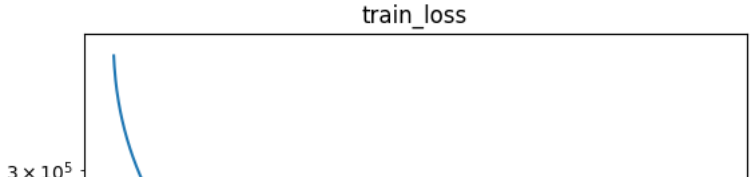
optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 0.0001 , minimum_RMSE: 1745.68 , epoch: 1000 , test_loss: 1745 , train_los

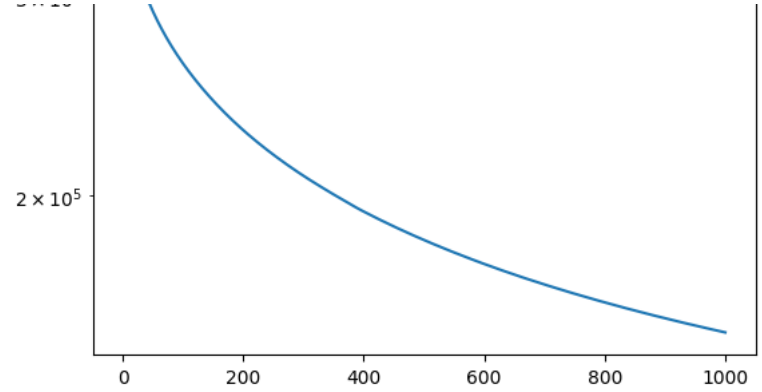




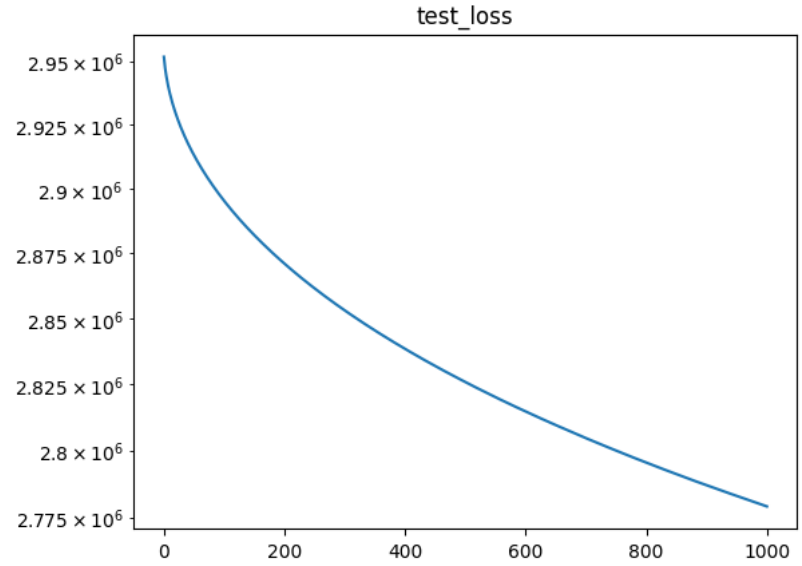
optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 0.0005 , minimum_RMSE: 909.75 , epoch: 1000 , test_loss: 1072 , train_loss

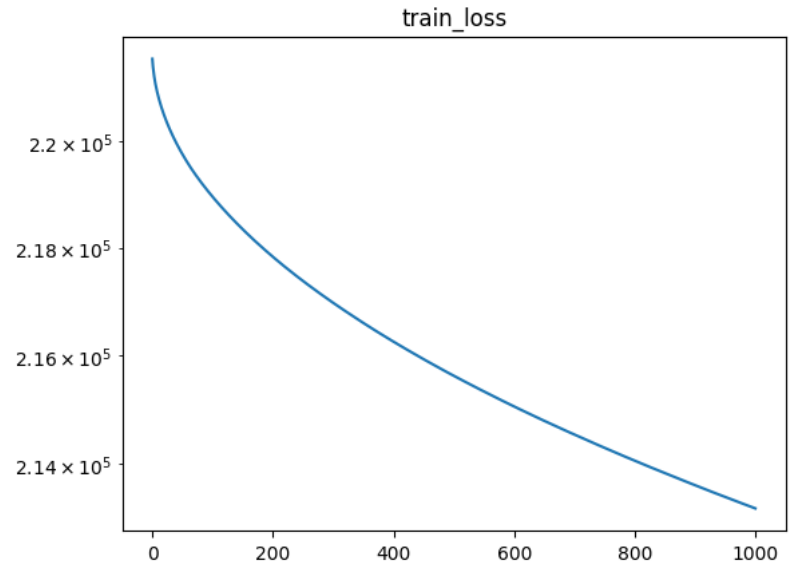




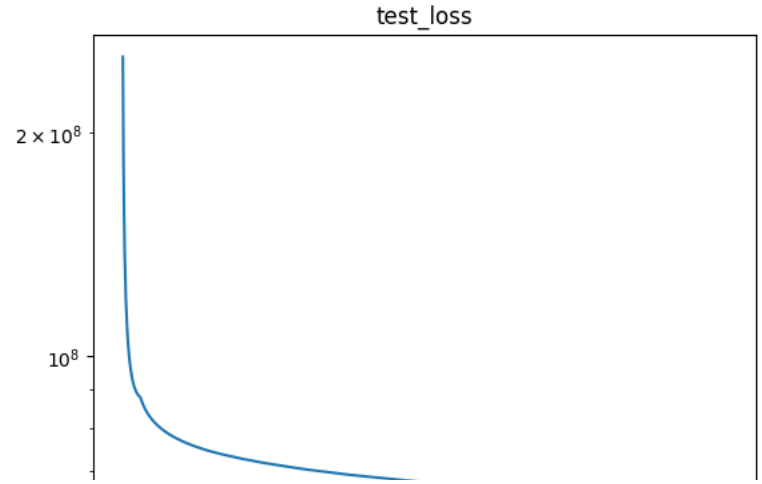


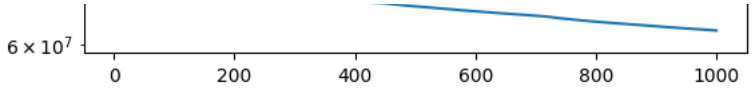
optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-05 , minimum_RMSE: 1667.08 , epoch: 1000 , test_loss: 1667 , train_loss

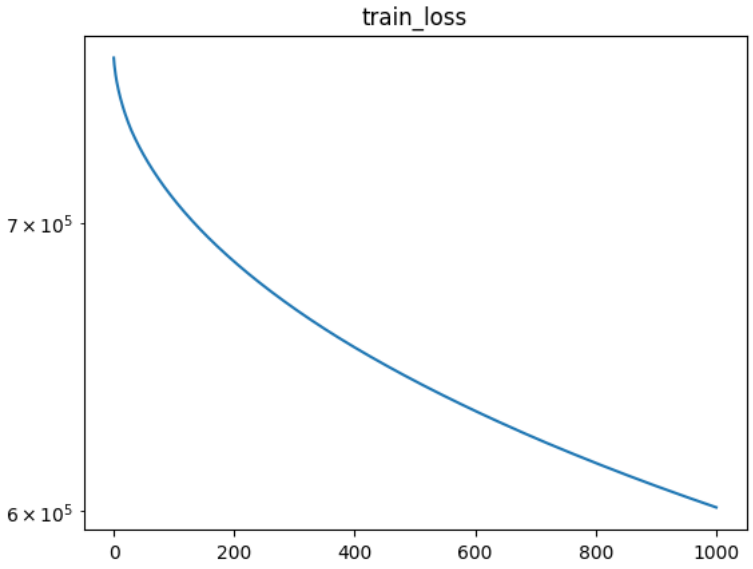




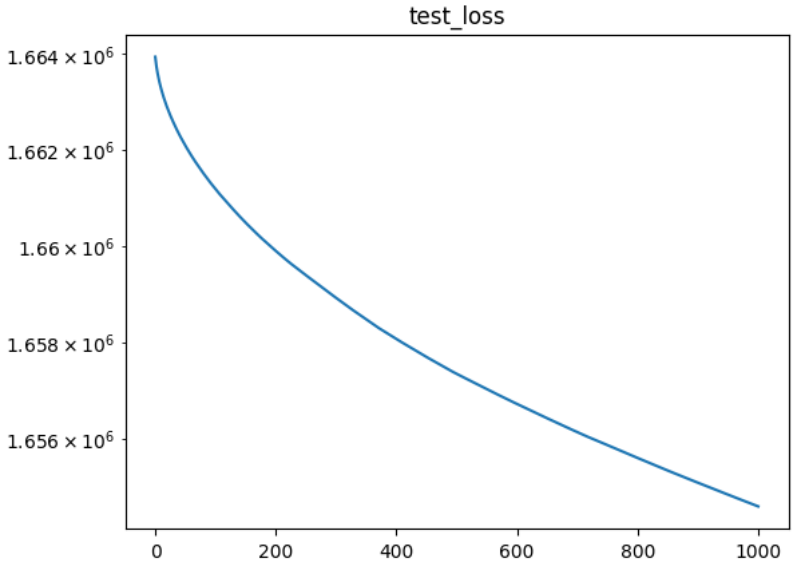
optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 5e-05 , minimum_RMSE: 7916.31 , epoch: 1000 , test_loss: 7916 , train_loss

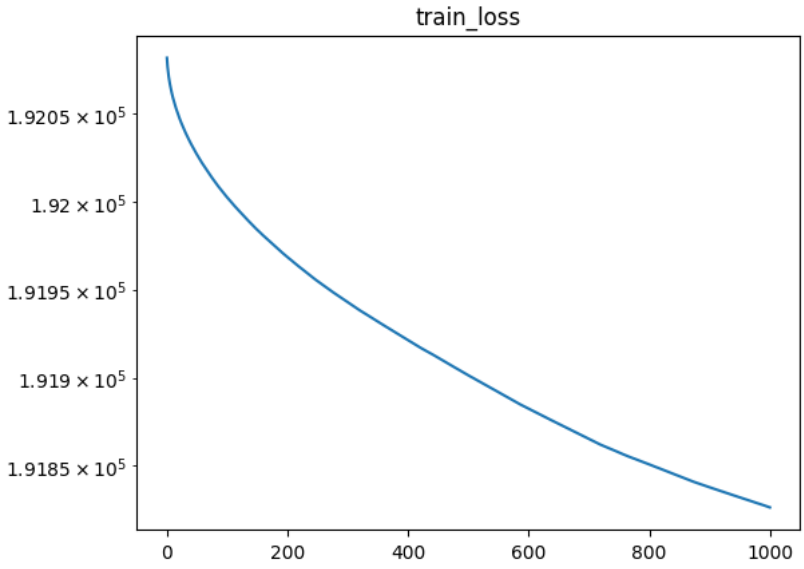




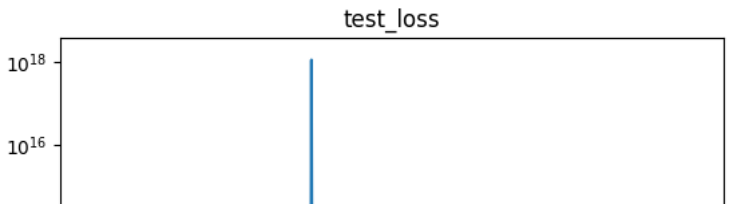


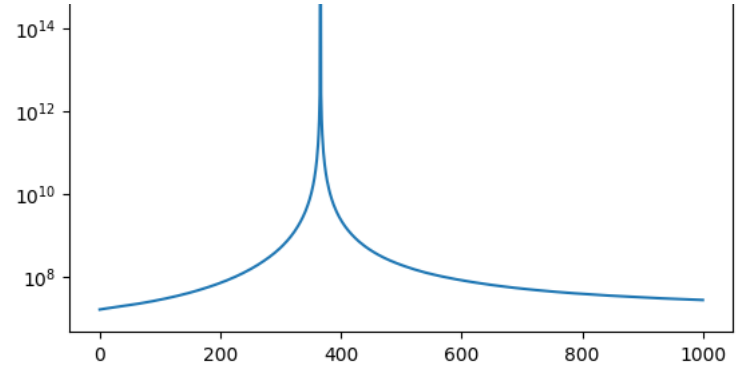
optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-06 , minimum_RMSE: 1286.31 , epoch: 1000 , test_loss: 1286 , train_loss

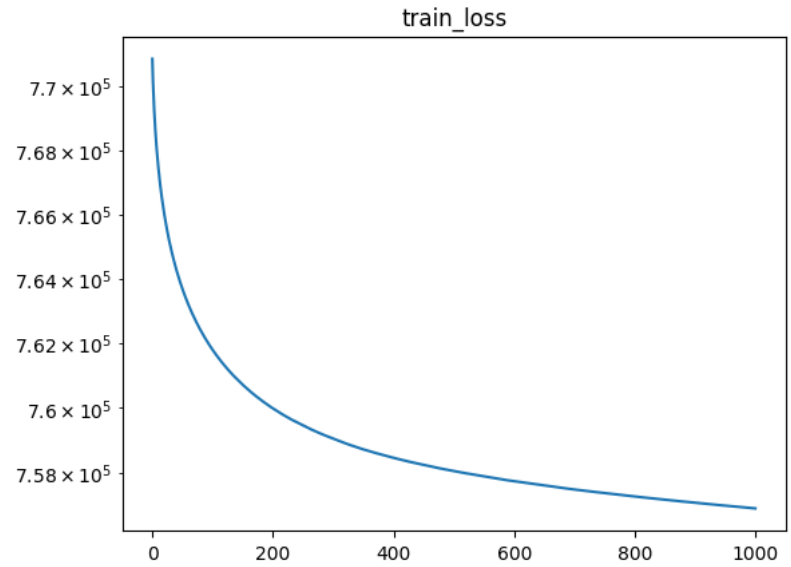




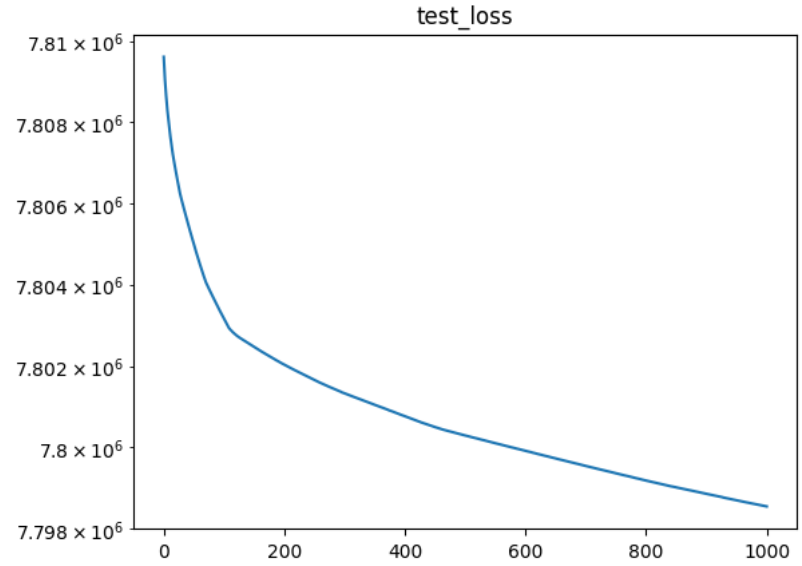
optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 5e-06 , minimum_RMSE: 4030.78 , epoch: 1000 , test_loss: 5249 , train_loss

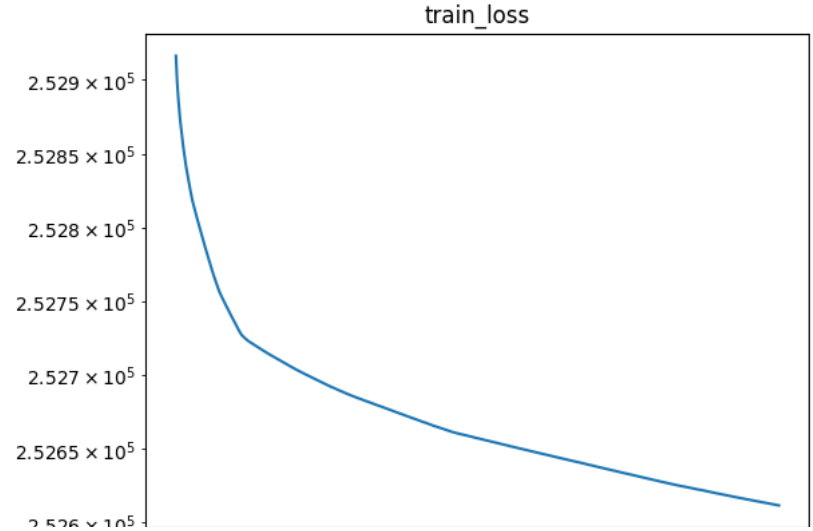


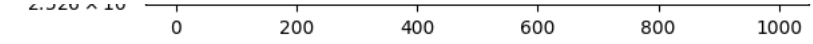




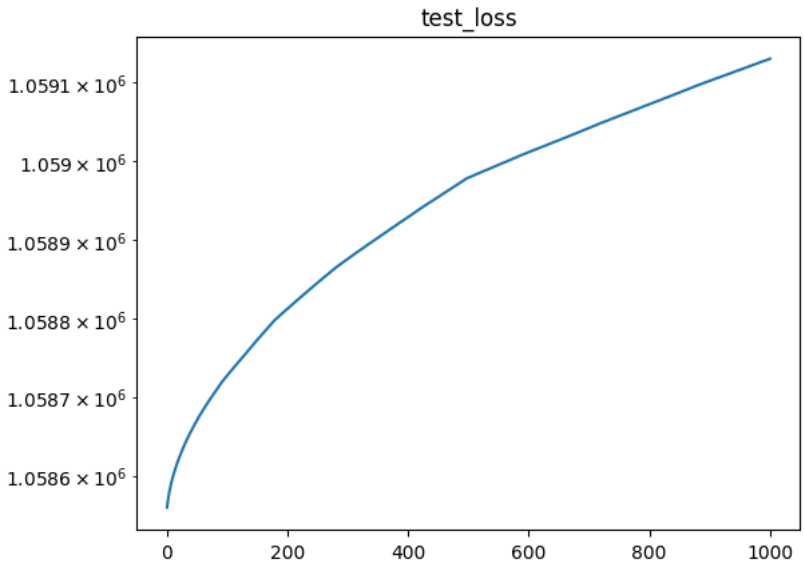
optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-07 , minimum_RMSE: 2792.59 , epoch: 1000 , test_loss: 2792 , train_loss

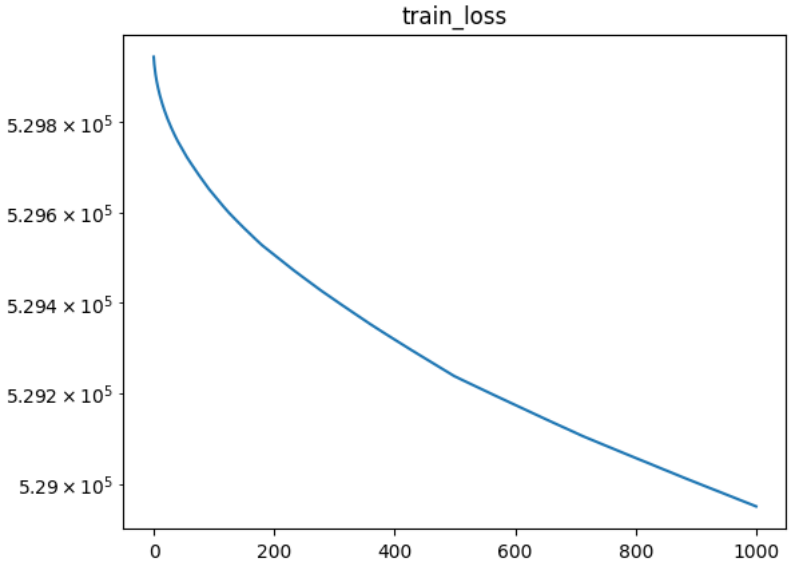




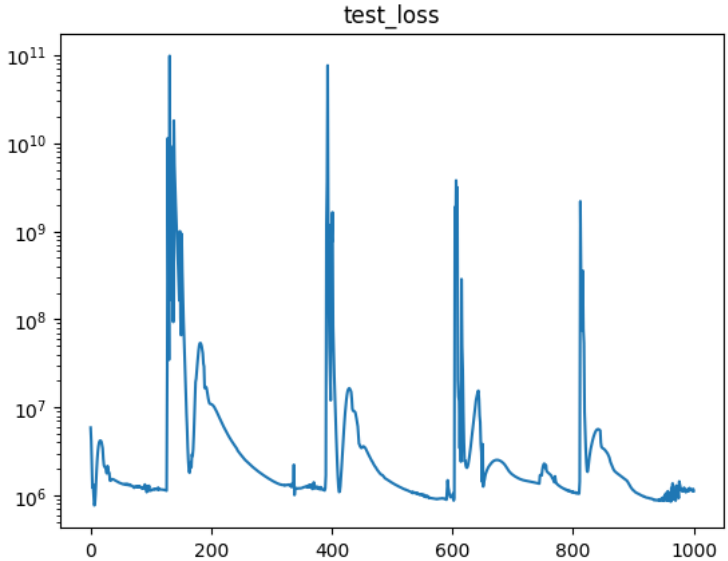


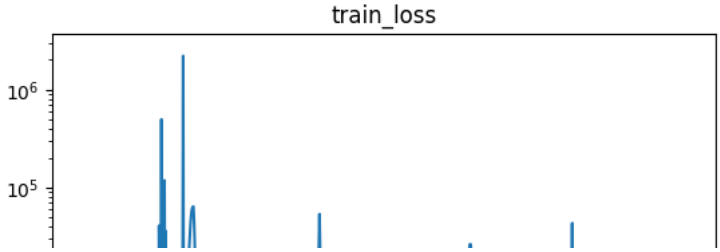
optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 5e-07 , minimum_RMSE: 1028.86 , epoch: 1000 , test_loss: 1029 , train_loss

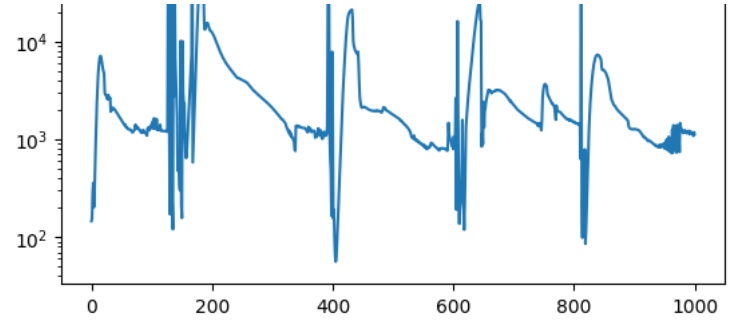




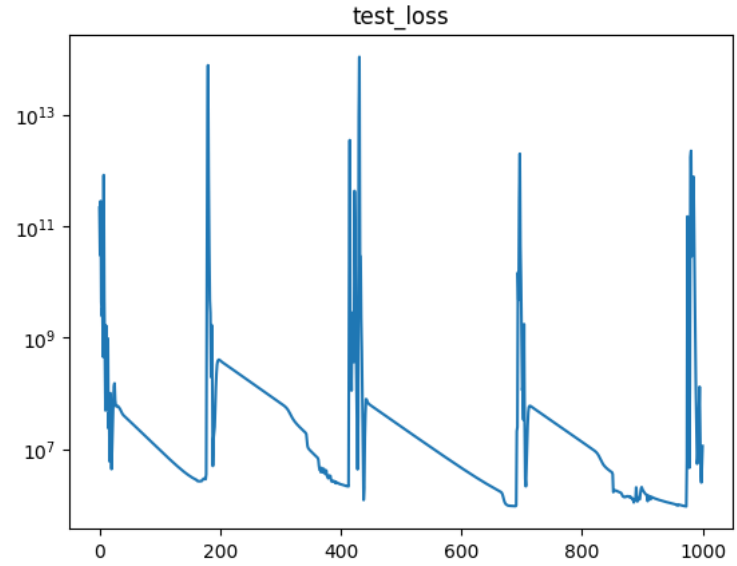
optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 0.1 , minimum_RMSE: 879.14 , epoch: 1000 , test_loss: 1061 , train_loss:

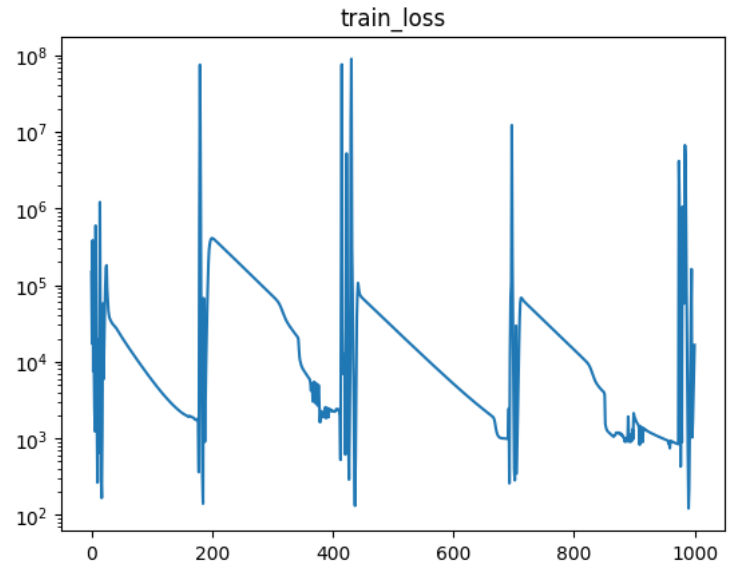




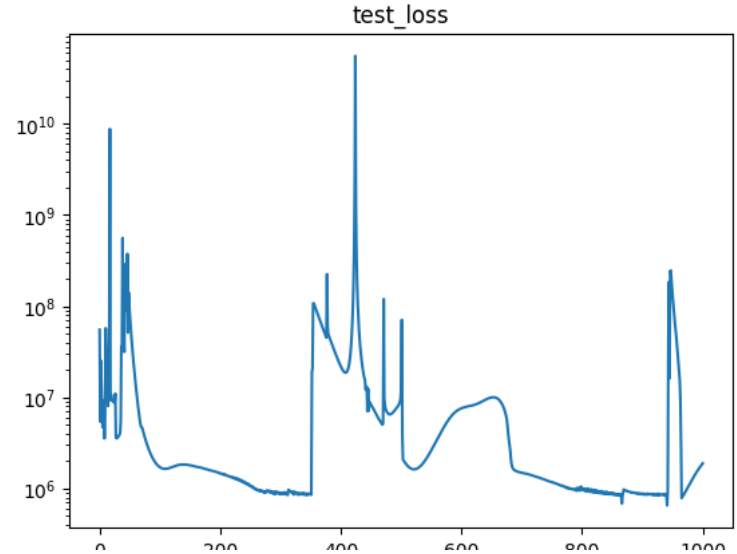


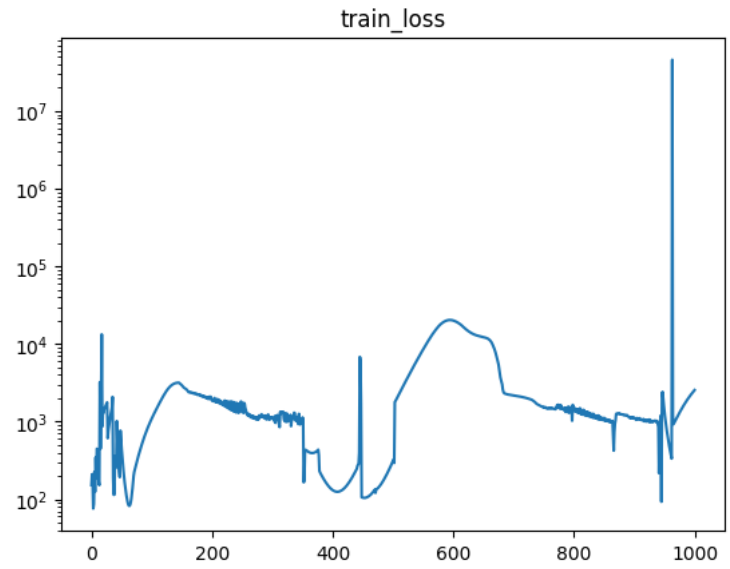
optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 0.5 , minimum_RMSE: 964.70 , epoch: 1000 , test_loss: 3362 , train_loss: 12



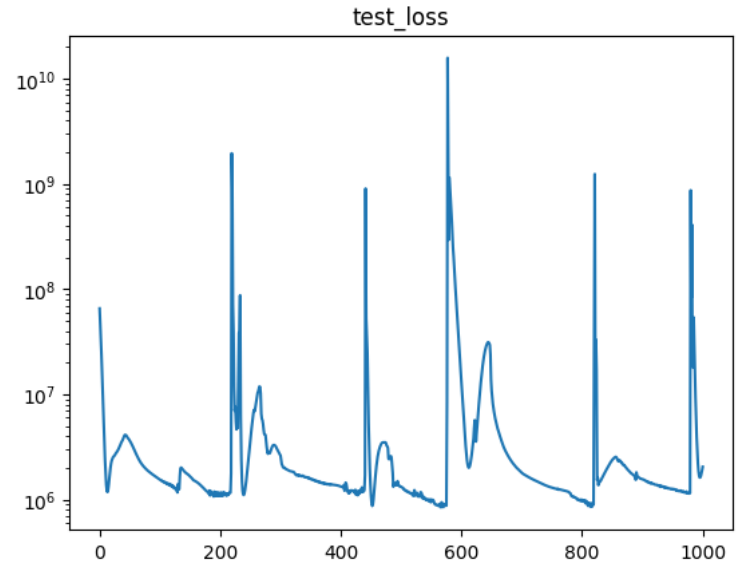


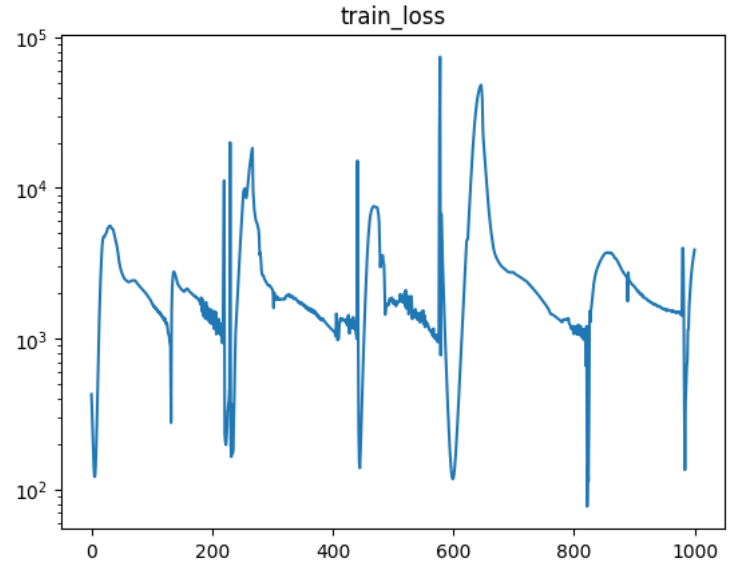
optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 0.01 , minimum_RMSE: 808.65 , epoch: 1000 , test_loss: 1374 , train_loss:



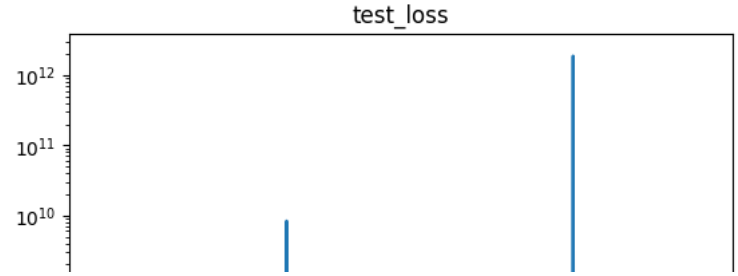


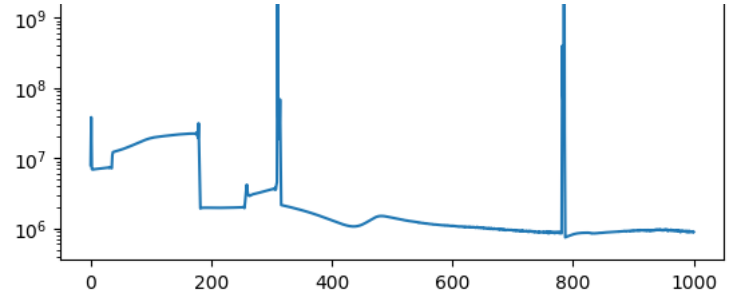
optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 0.05 , minimum_RMSE: 917.83 , epoch: 1000 , test_loss: 1430 , train_loss:

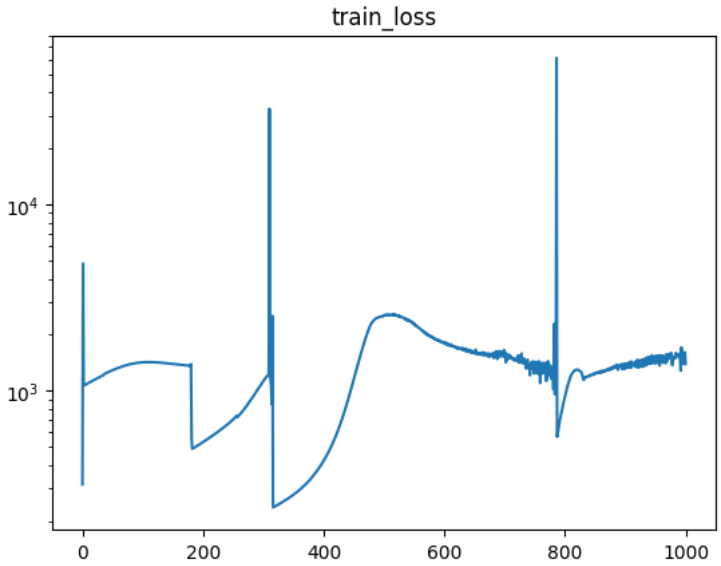




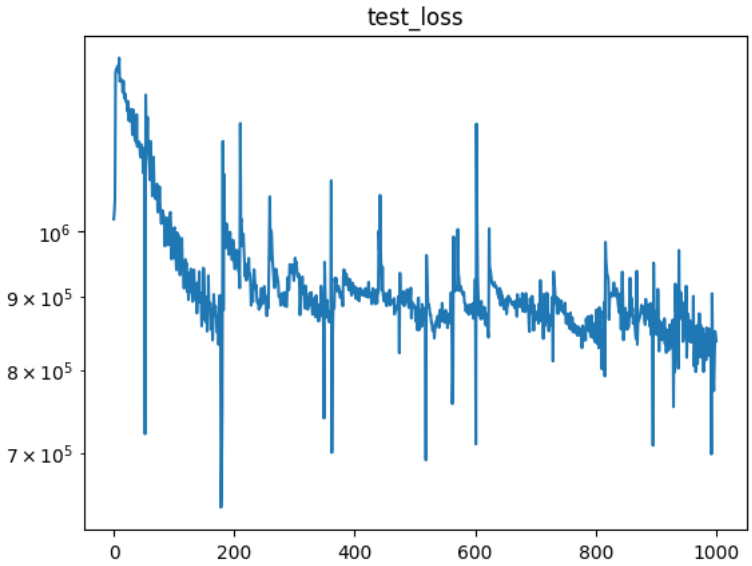
optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 0.001 , minimum_RMSE: 862.54 , epoch: 1000 , test_loss: 943 , train_loss:

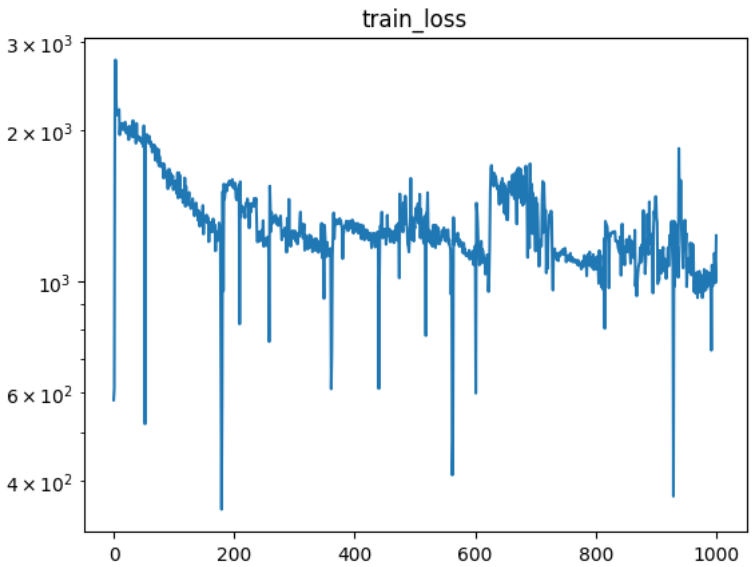




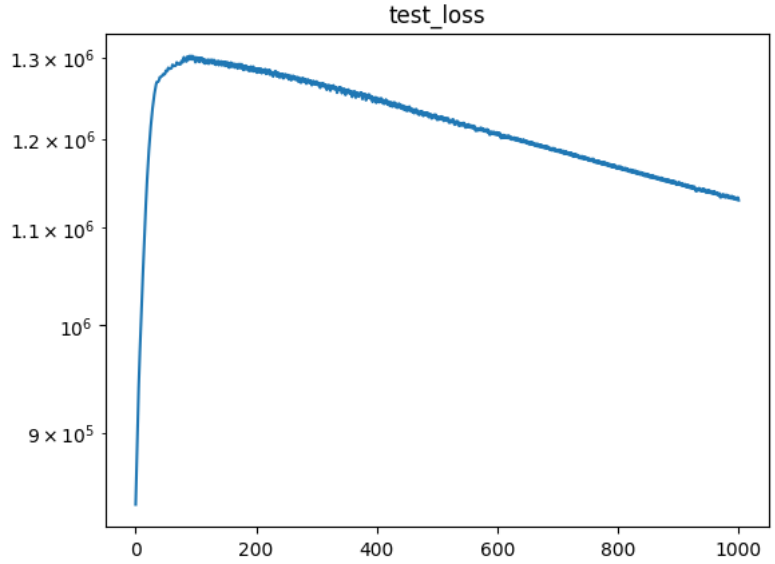


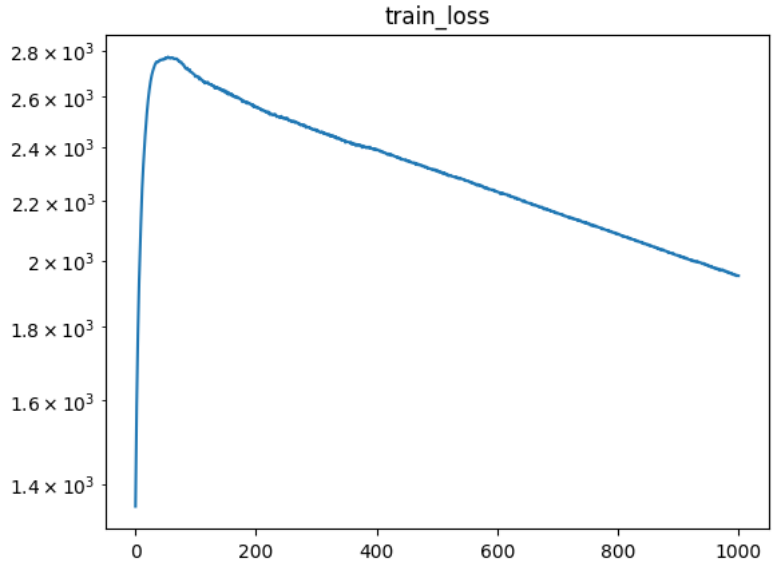
optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 0.005 , minimum_RMSE: 800.60 , epoch: 1000 , test_loss: 915 , train_loss:



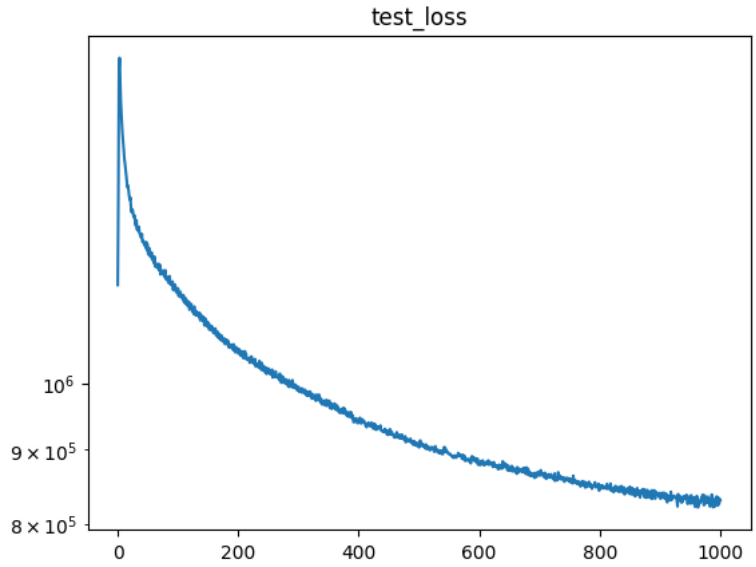


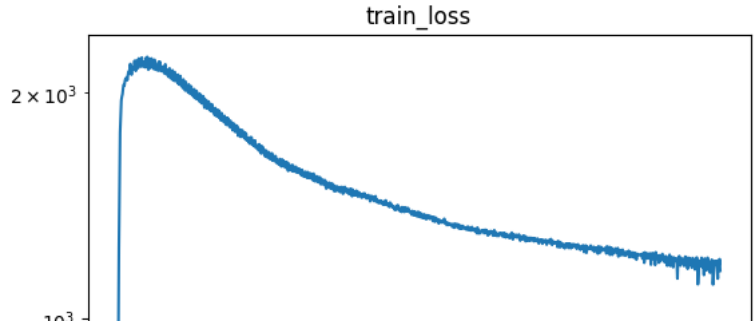
optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 0.0001 , minimum_RMSE: 916.01 , epoch: 1000 , test_loss: 1063 , train_loss:

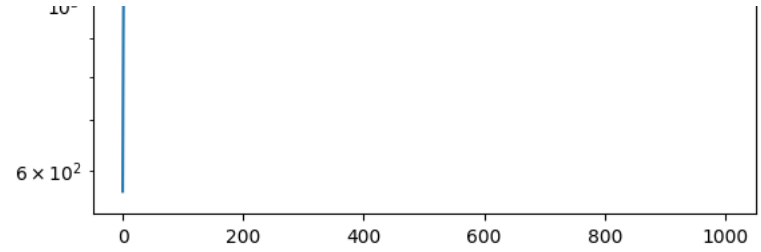




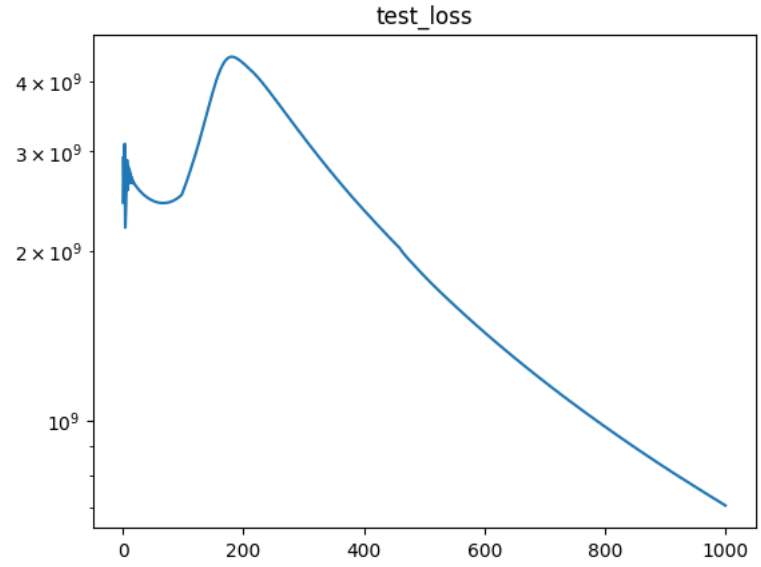
optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 0.0005 , minimum_RMSE: 906.31 , epoch: 1000 , test_loss: 911 , train_loss:

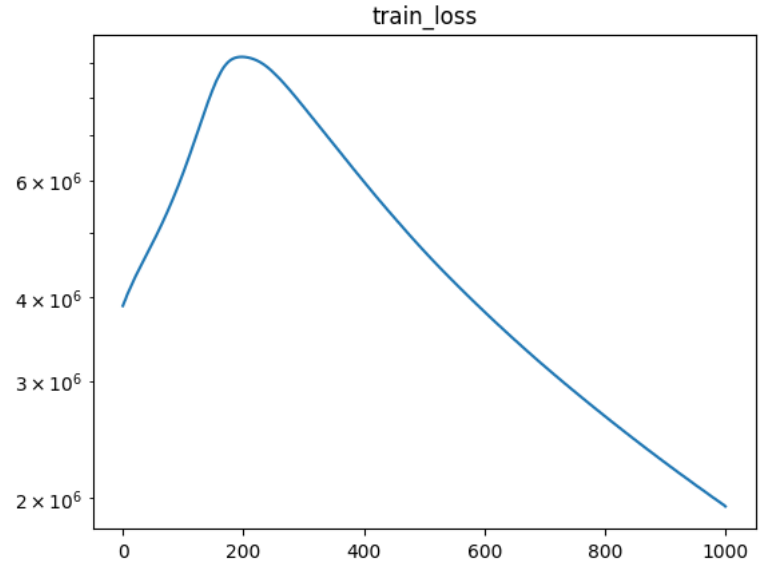




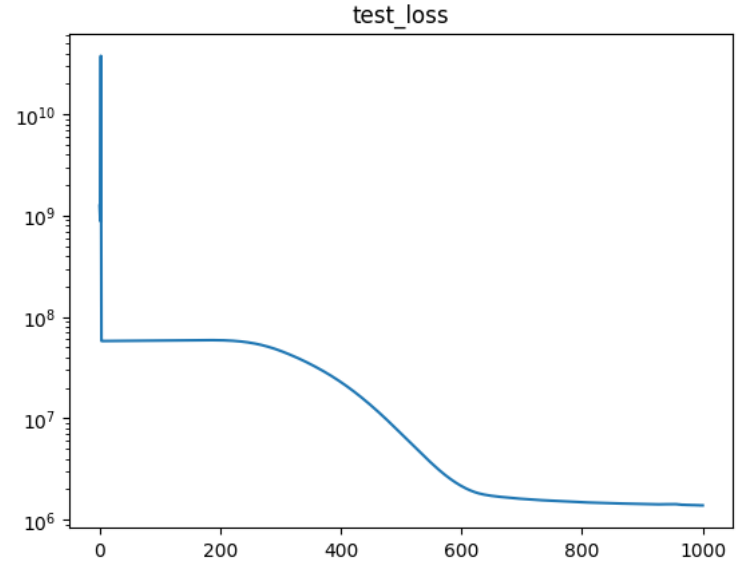


optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-05 , minimum_RMSE: 26597.27 , epoch: 1000 , test_loss: 26597 , train_loss:

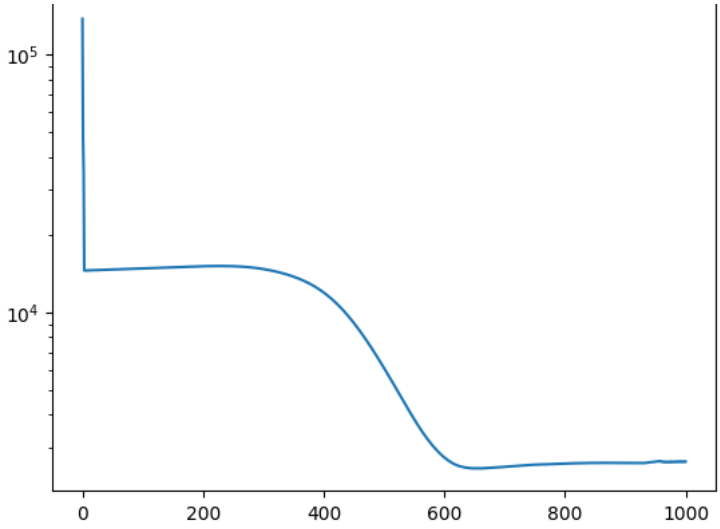




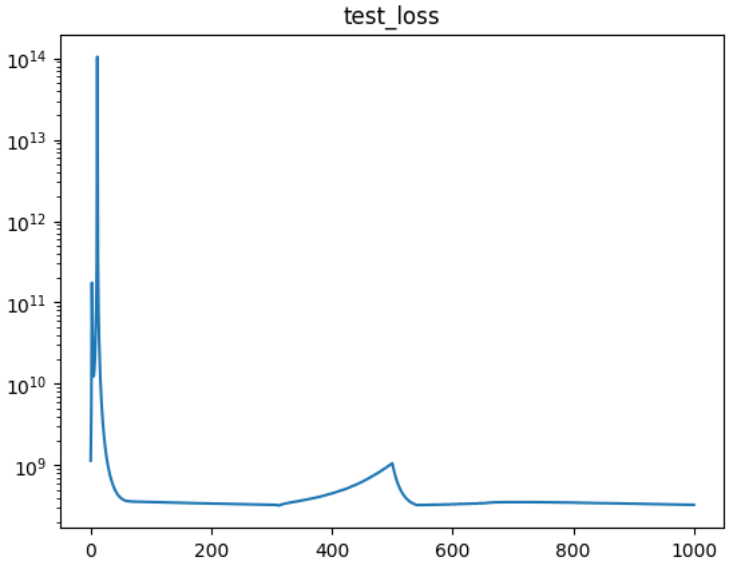
optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 5e-05 , minimum_RMSE: 1175.99 , epoch: 1000 , test_loss: 1176 , train_loss:

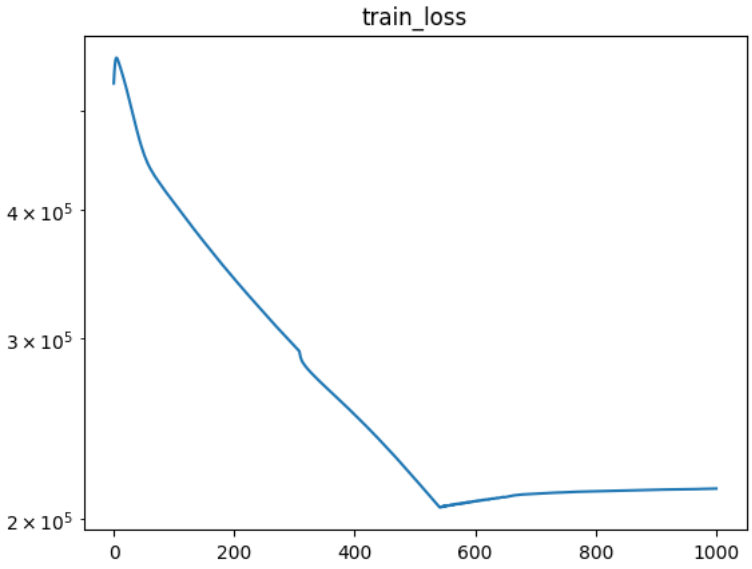


train_loss

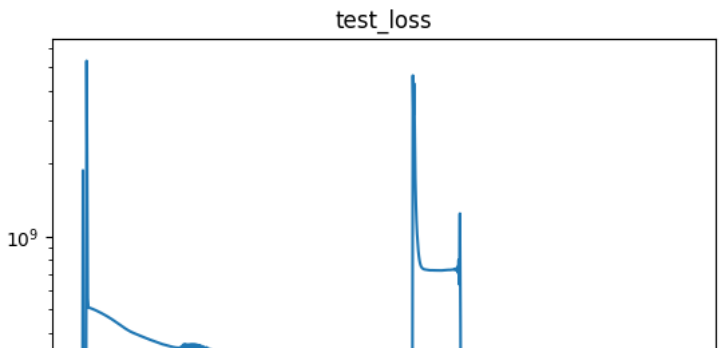


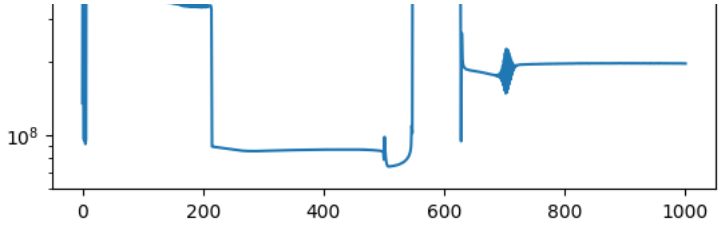
optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-06 , minimum_RMSE: 17902.22 , epoch: 1000 , test_loss: 18081 , train_loss:

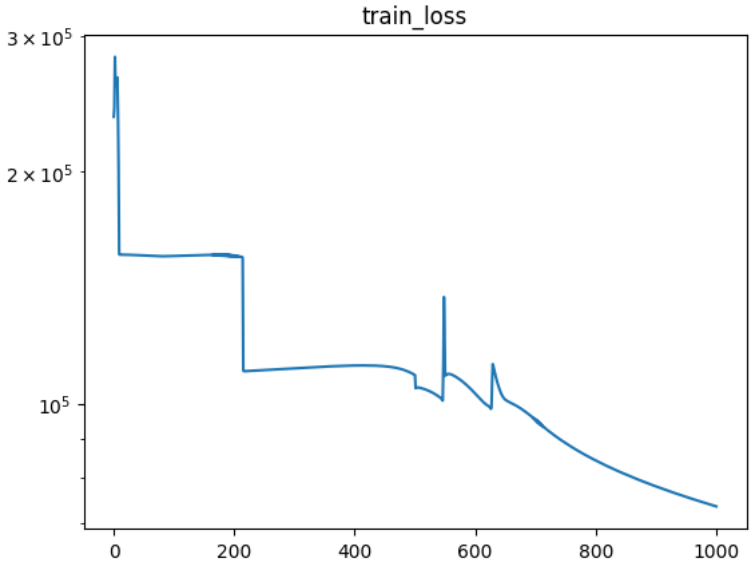




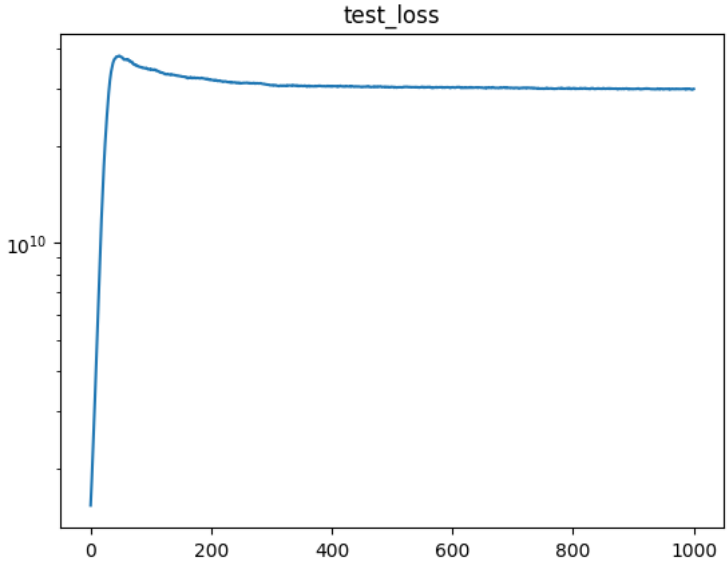
optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 5e-06 , minimum_RMSE: 8599.91 , epoch: 1000 , test_loss: 14039 , train_loss:

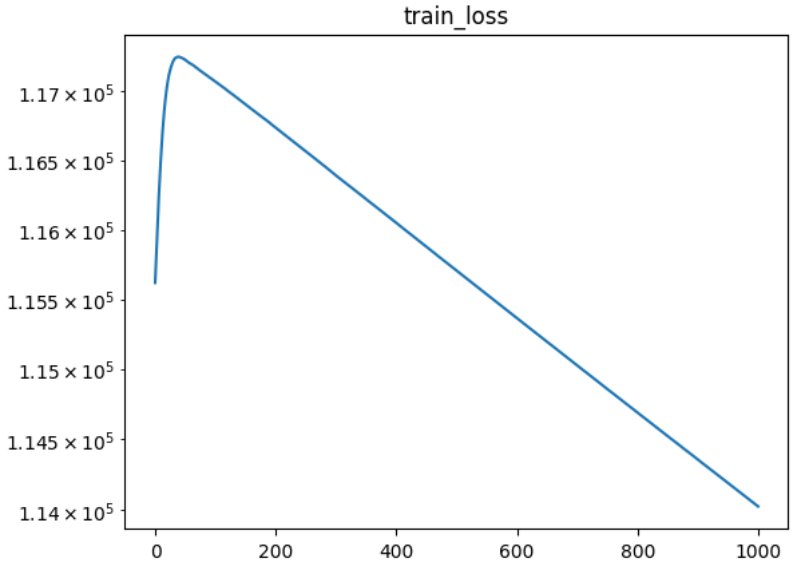




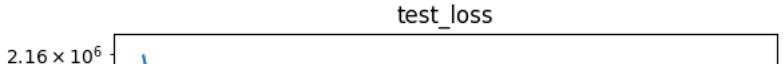


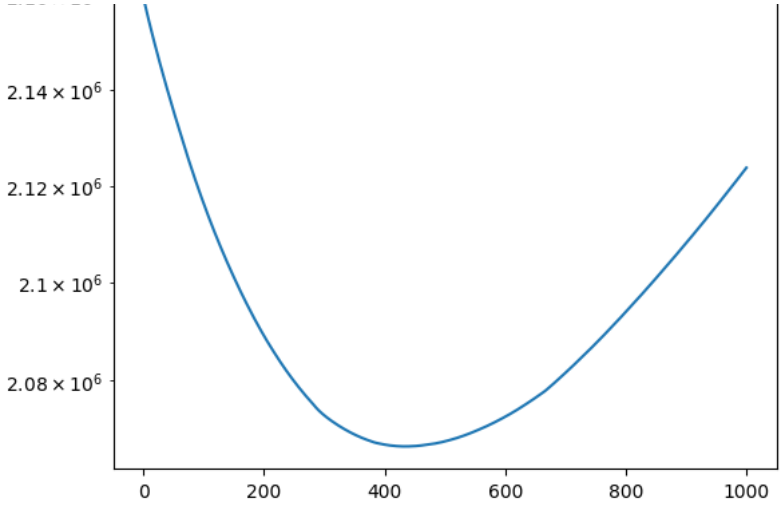
optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-07 , minimum_RMSE: 39123.97 , epoch: 1000 , test_loss: 173274 , train_loss:

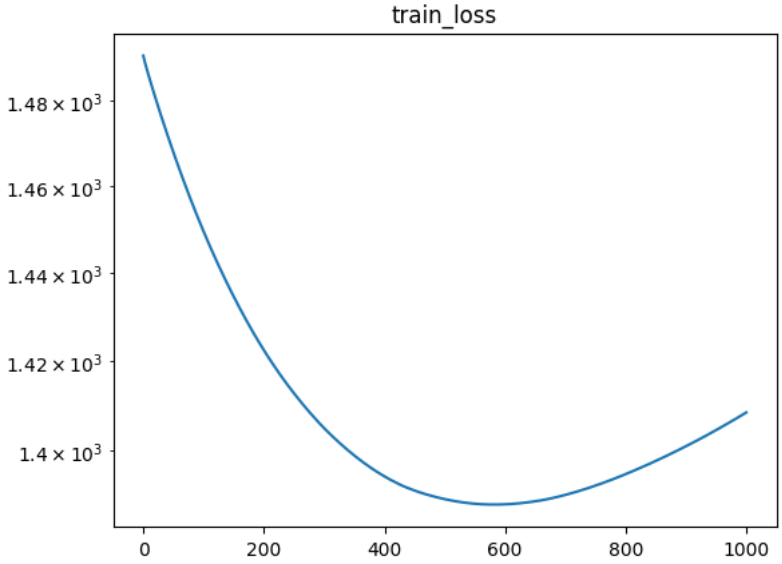




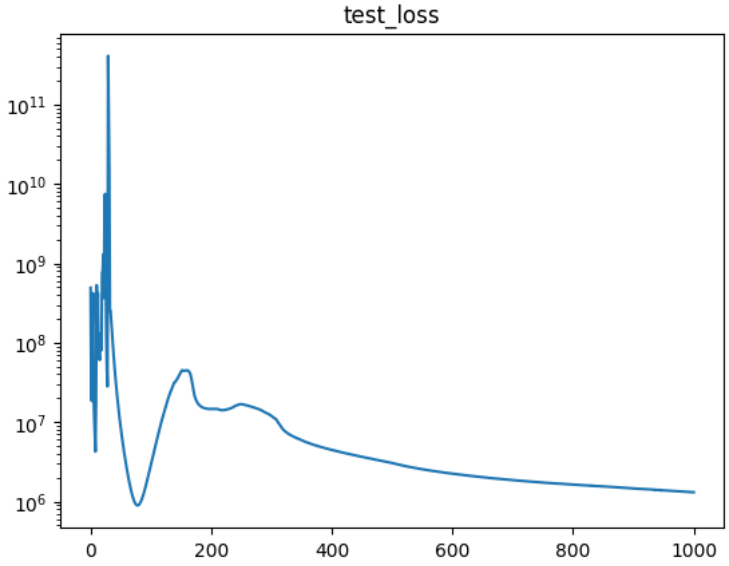
optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 5e-07 , minimum_RMSE: 1437.57 , epoch: 1000 , test_loss: 1457 , train_loss:

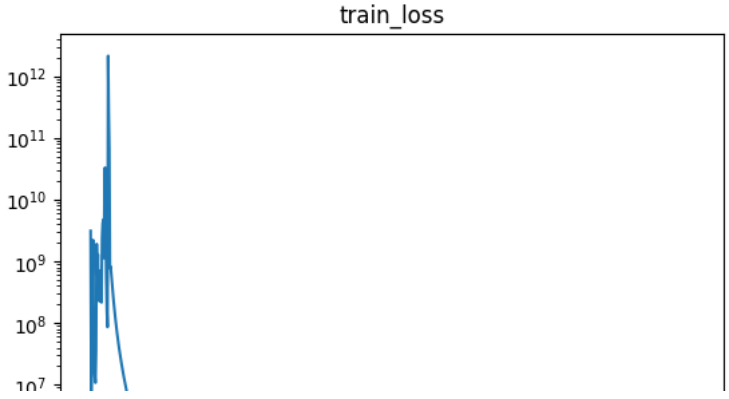


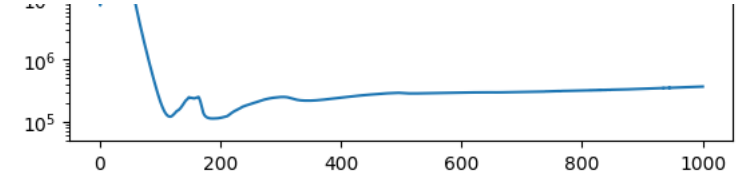




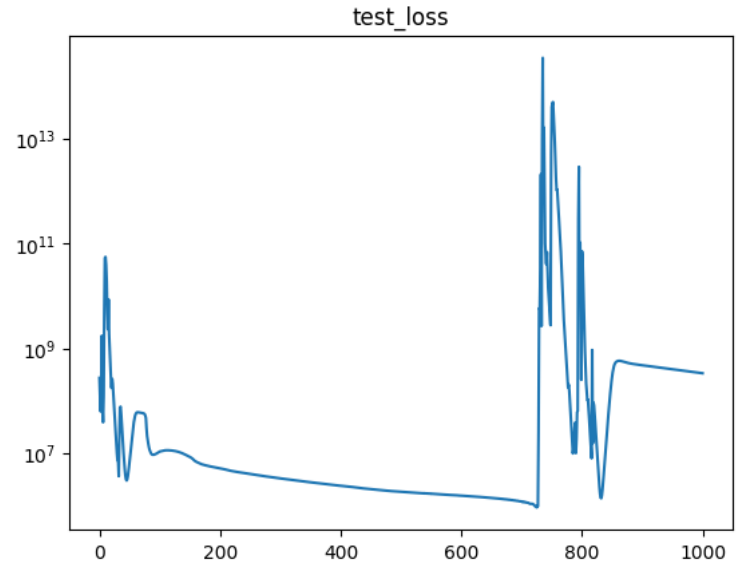
optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 0.1 , minimum_RMSE: 943.94 , epoch: 1000 , test_loss: 1142 , train_loss: 6(

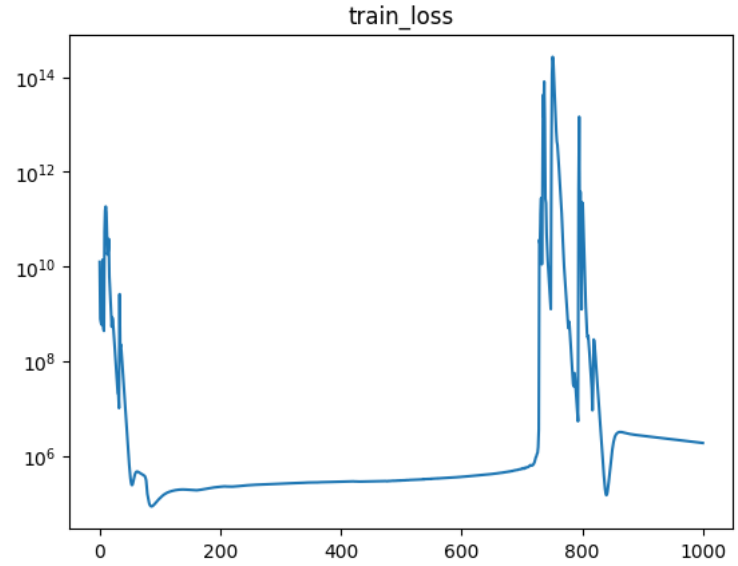




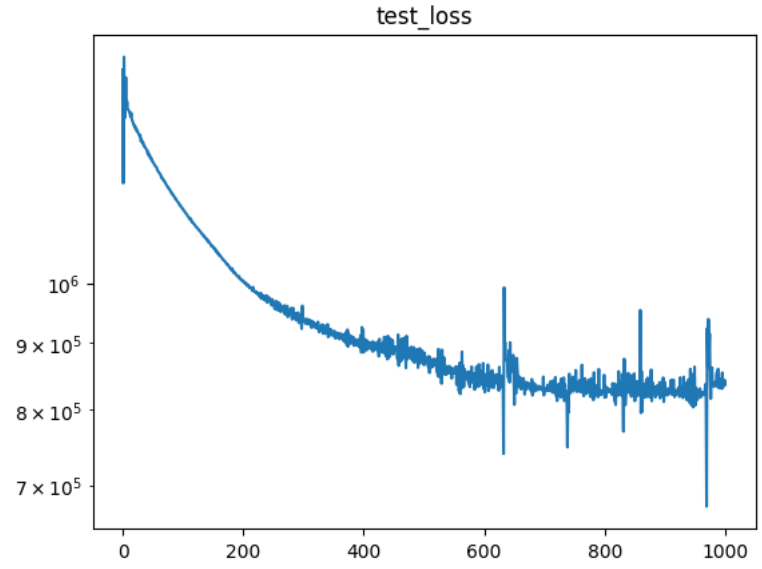


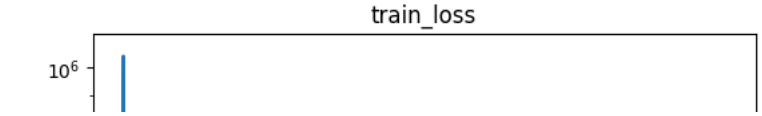
optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 0.5 , minimum_RMSE: 959.74 , epoch: 1000 , test_loss: 18281 , train_loss: 137

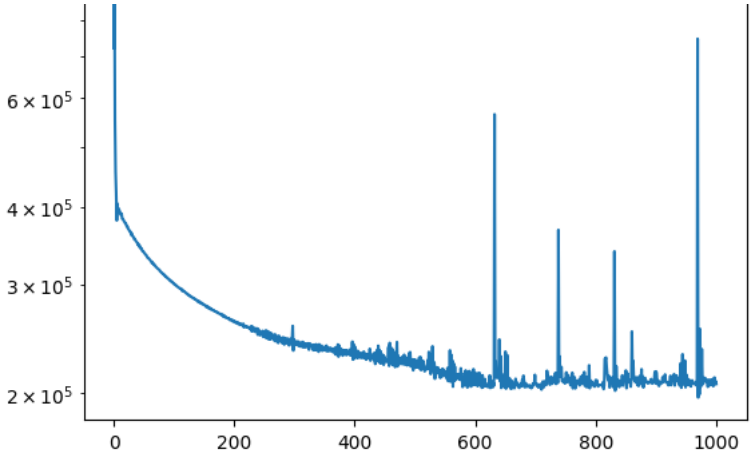




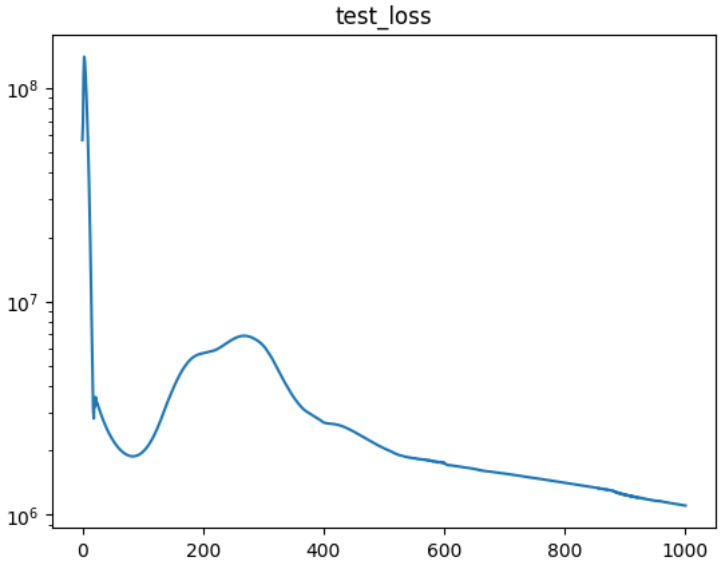
optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 0.01 , minimum_RMSE: 821.53 , epoch: 1000 , test_loss: 914 , train_loss: 4

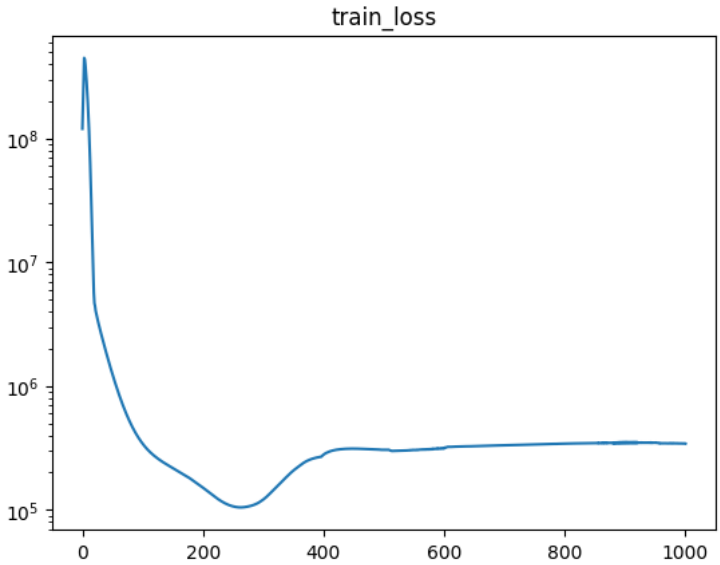




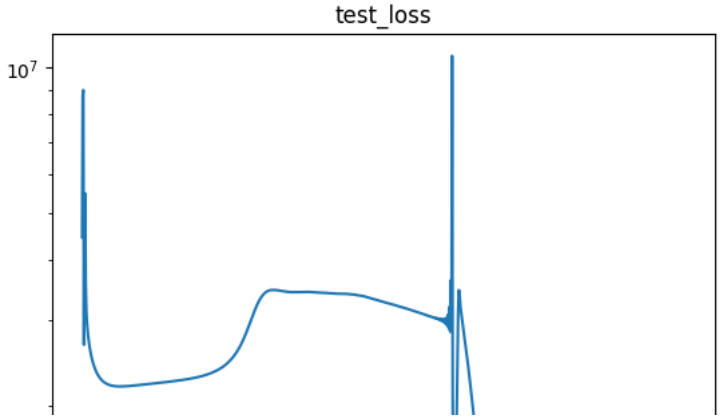


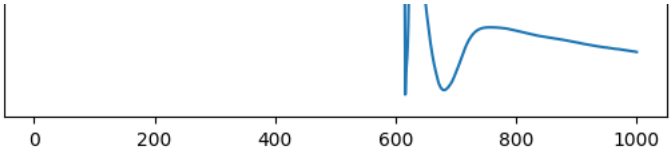
optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 0.05 , minimum_RMSE: 1050.42 , epoch: 1000 , test_loss: 1050 , train_loss:

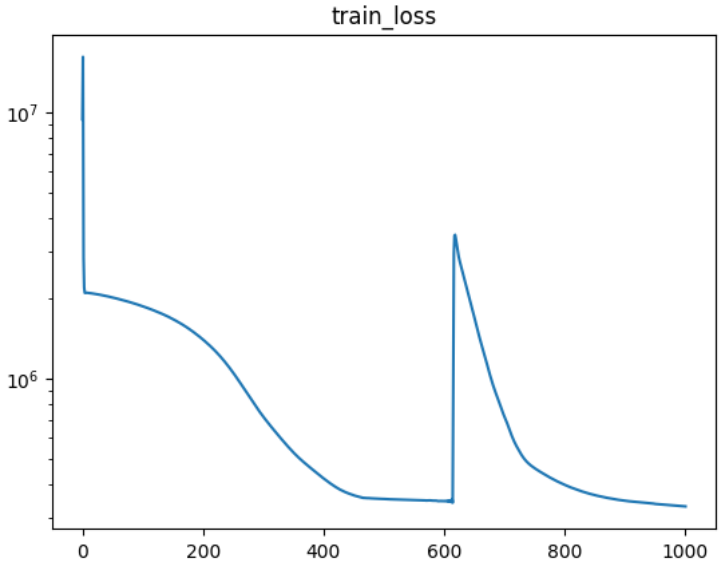




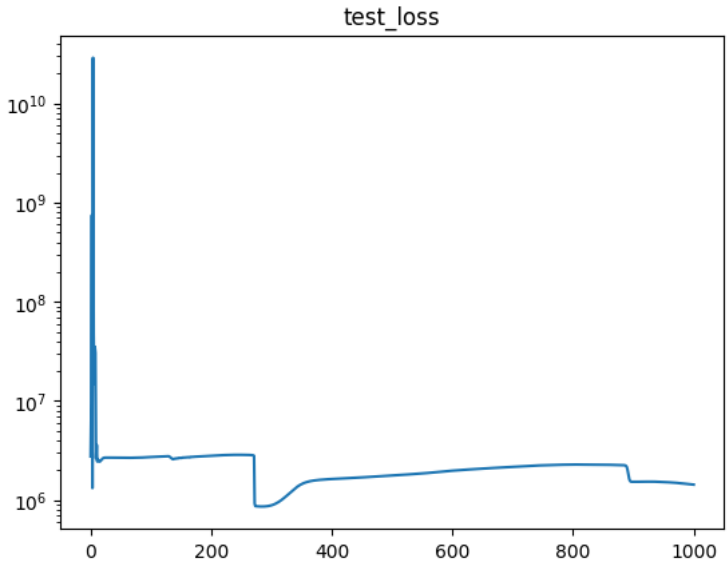
optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 0.001 , minimum_RMSE: 1114.49 , epoch: 1000 , test_loss: 1233 , train_loss:

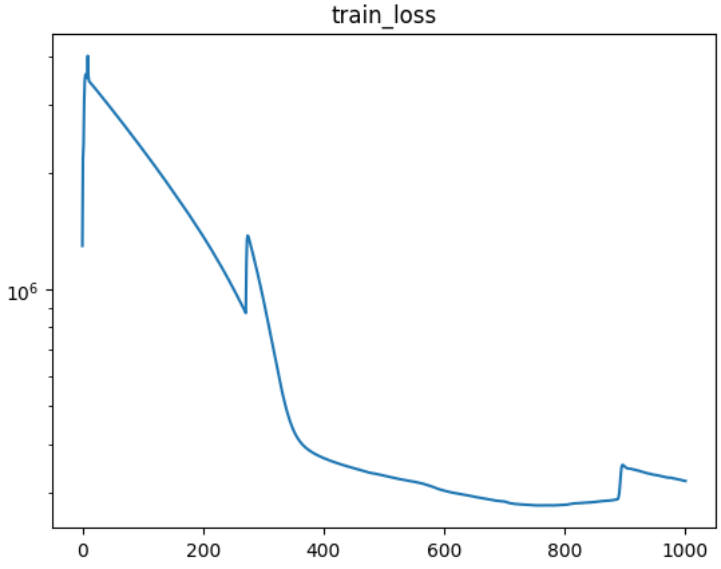




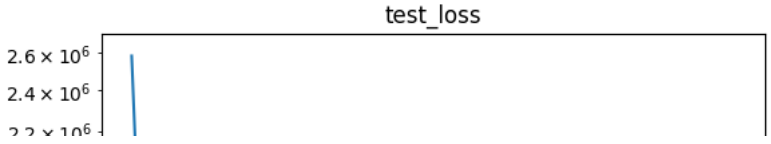


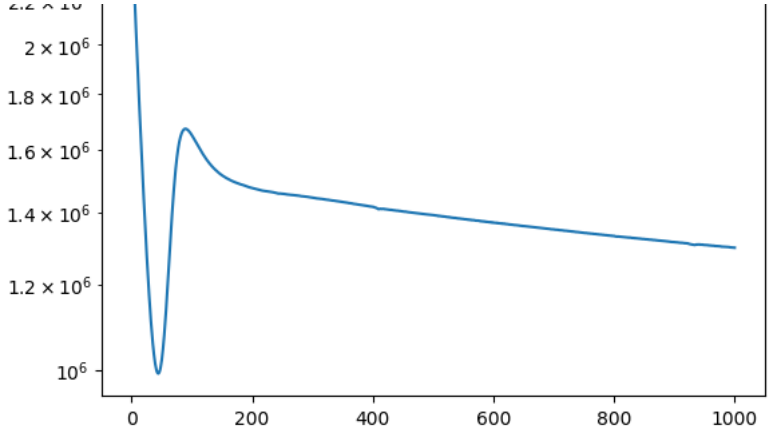
optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 0.005 , minimum_RMSE: 926.32 , epoch: 1000 , test_loss: 1194 , train_loss:

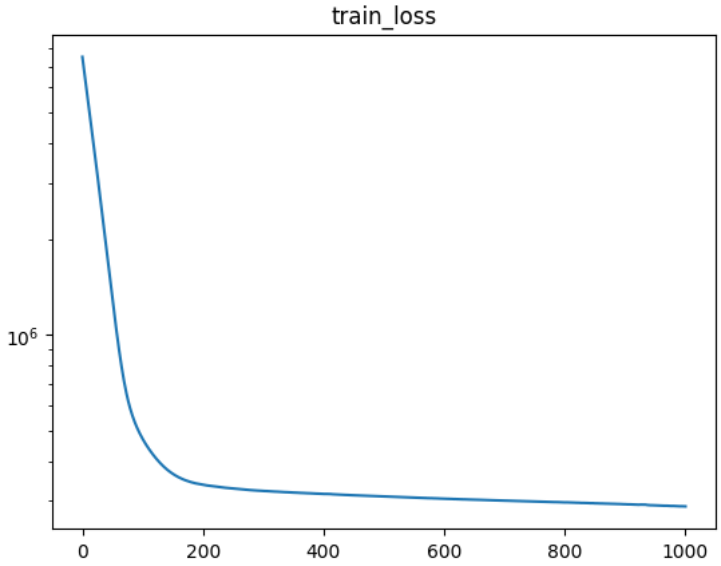




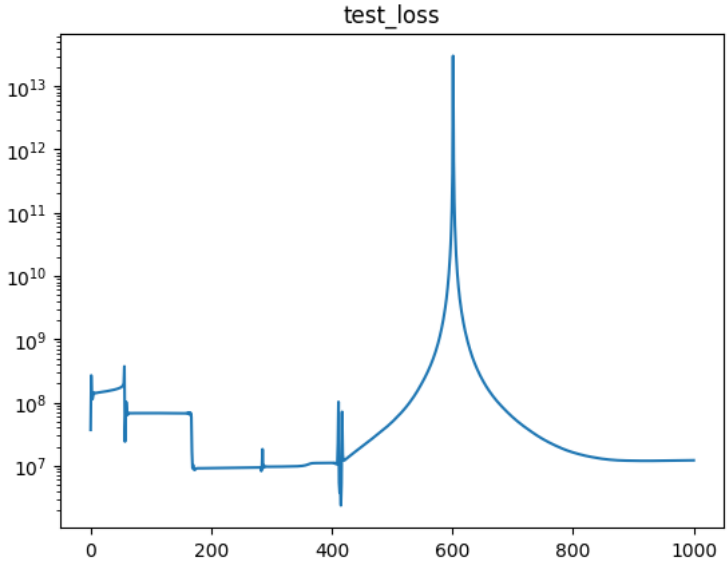
optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 0.0001 , minimum_RMSE: 996.41 , epoch: 1000 , test_loss: 1139 , train_loss:

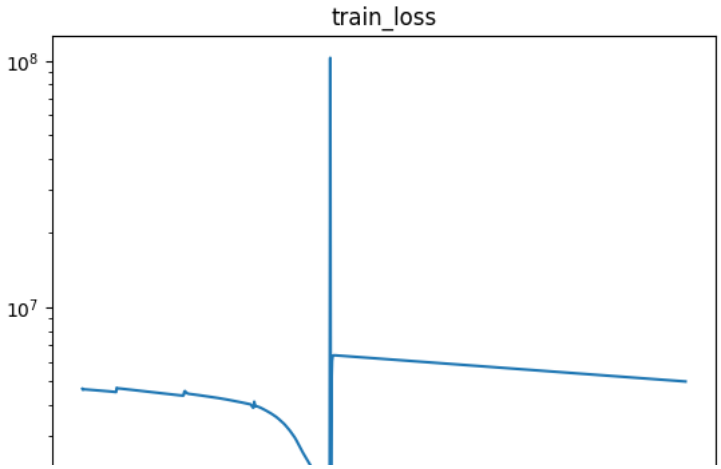


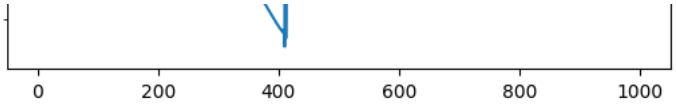




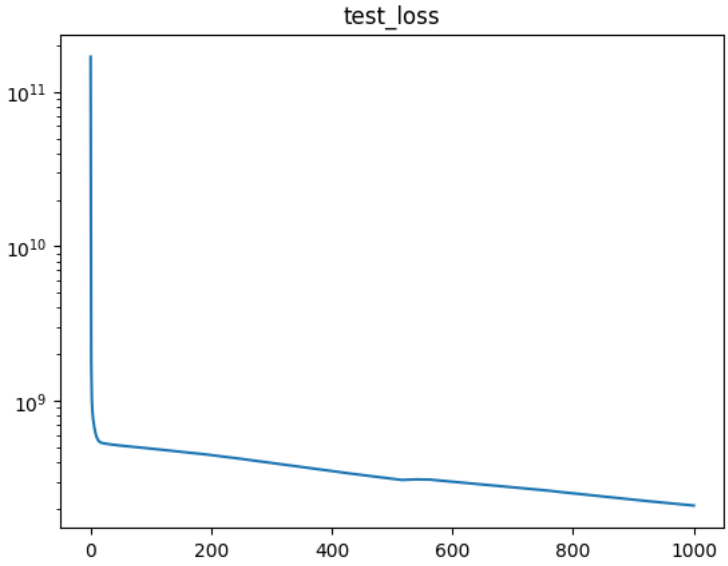
optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 0.0005 , minimum_RMSE: 1538.68 , epoch: 1000 , test_loss: 3495 , train_loss:

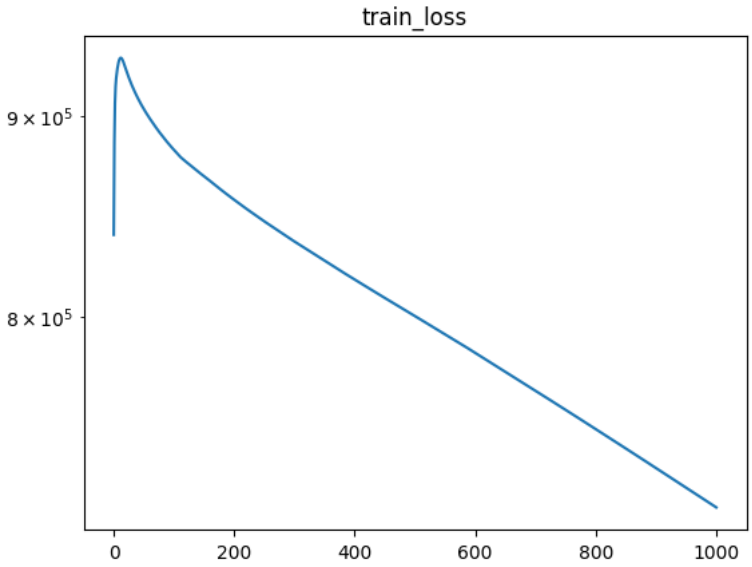




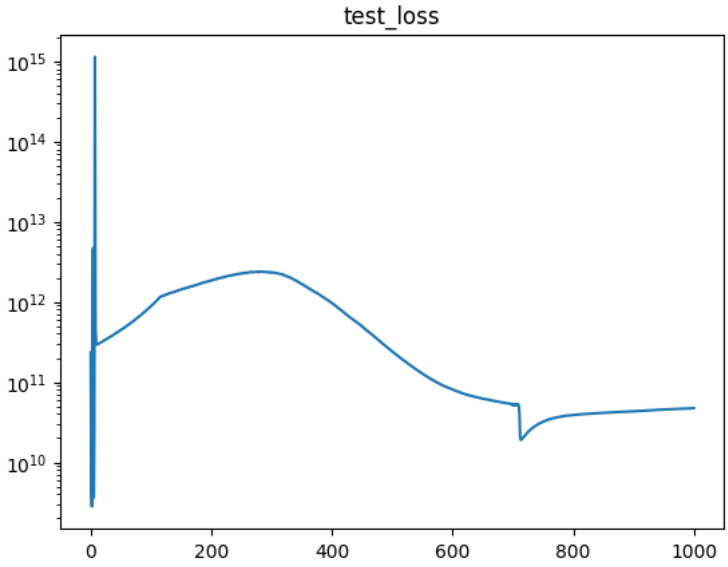


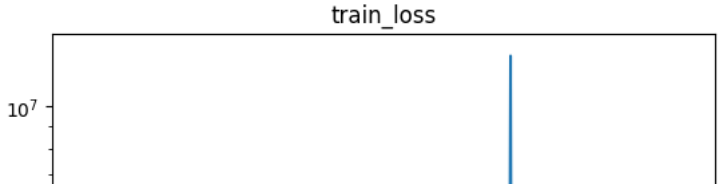
optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-05 , minimum_RMSE: 14503.24 , epoch: 1000 , test_loss: 14503 , train_loss:

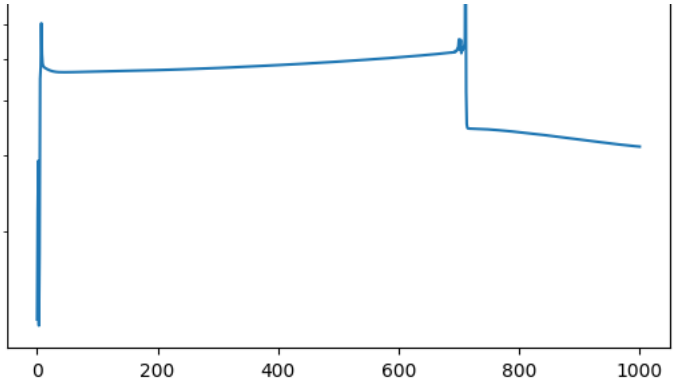




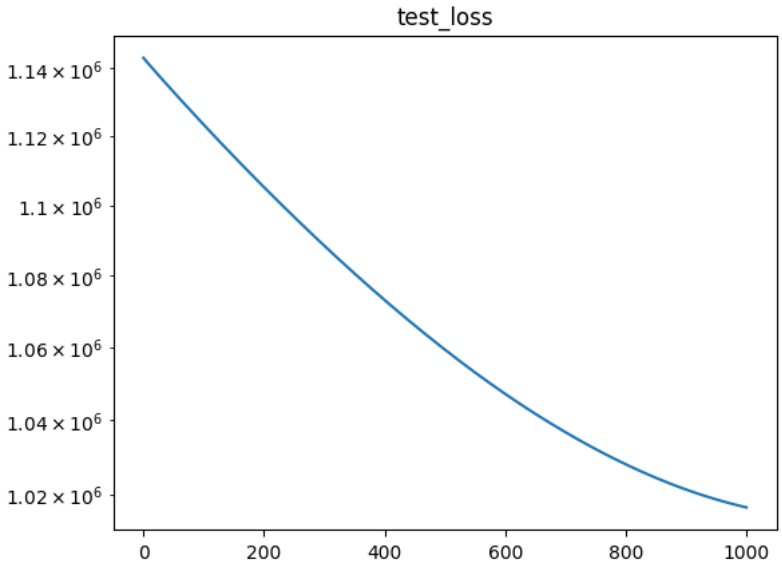
optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 5e-05 , minimum_RMSE: 53109.38 , epoch: 1000 , test_loss: 217581 , train_loss:

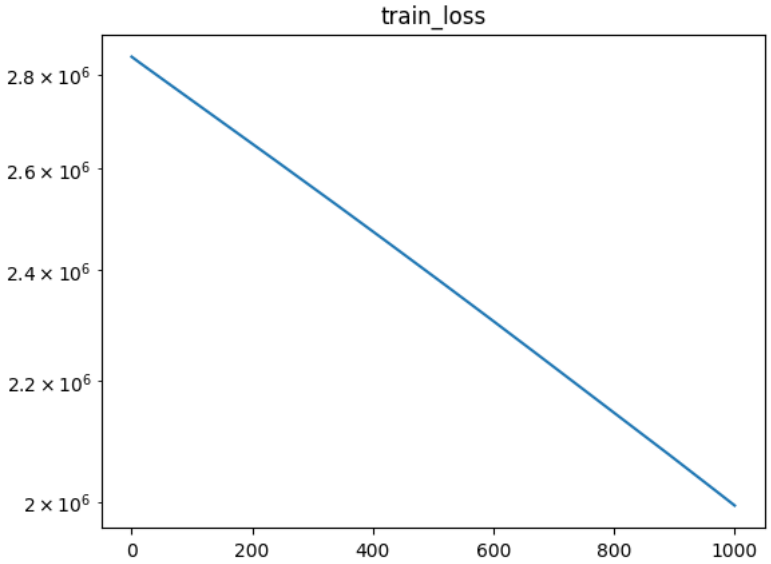




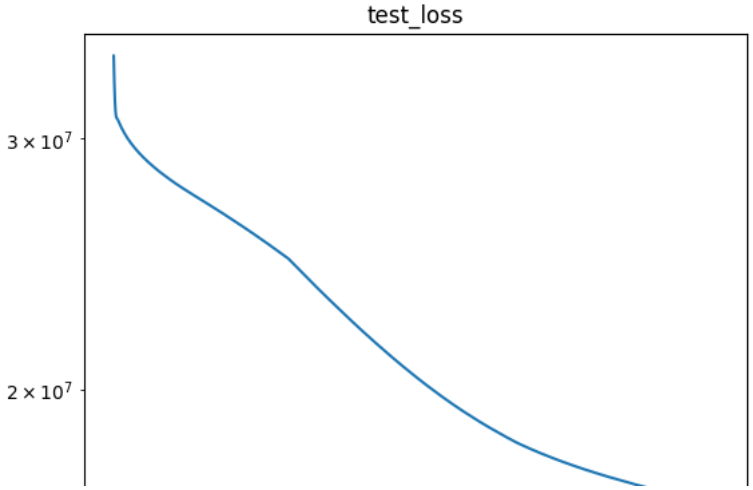


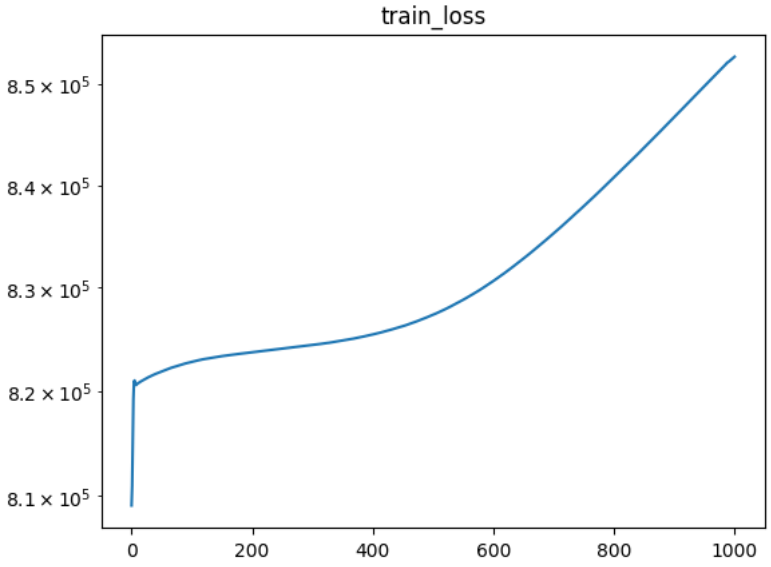
optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-06 , minimum_RMSE: 1008.28 , epoch: 1000 , test_loss: 1008 , train_loss:



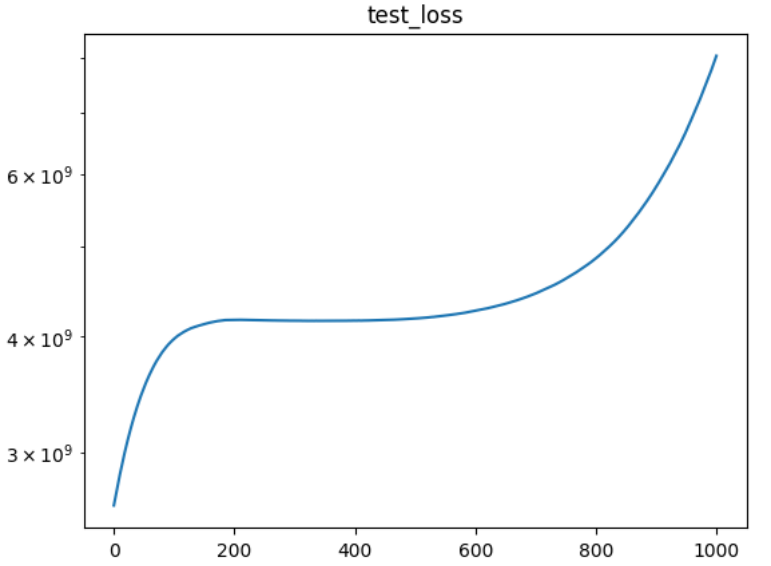


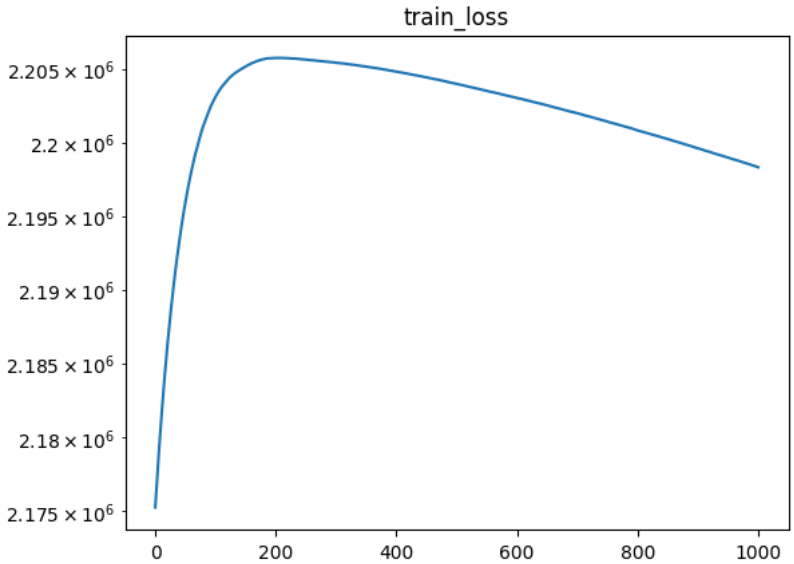
optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 5e-06 , minimum_RMSE: 4071.18 , epoch: 1000 , test_loss: 4071 , train_loss:



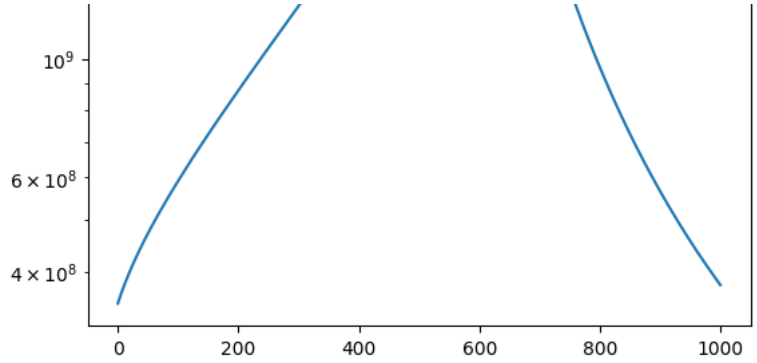


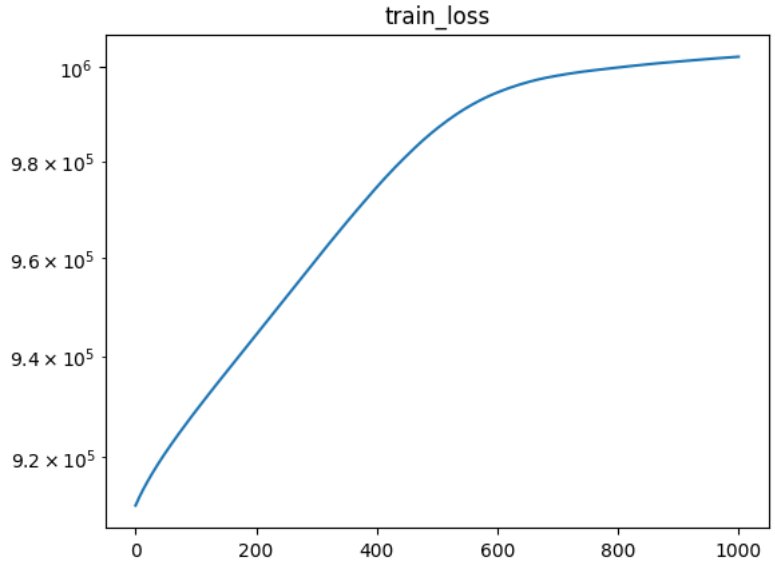
optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-07 , minimum_RMSE: 51265.68 , epoch: 1000 , test_loss: 89735 , train_loss:



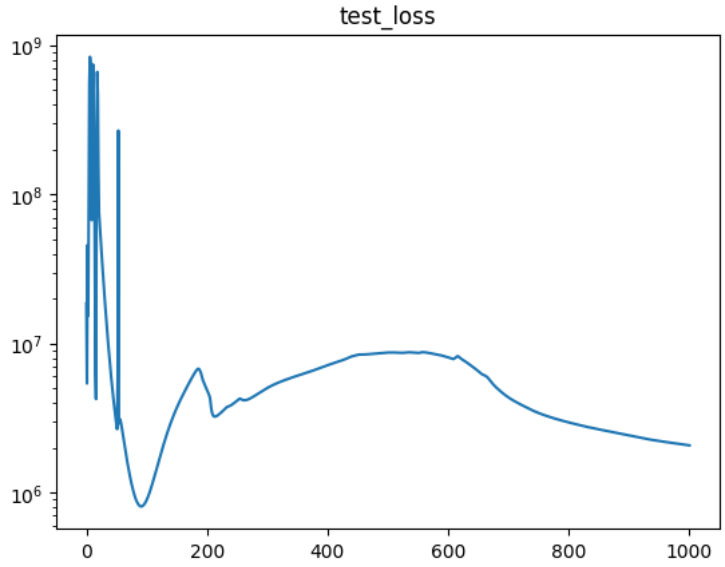


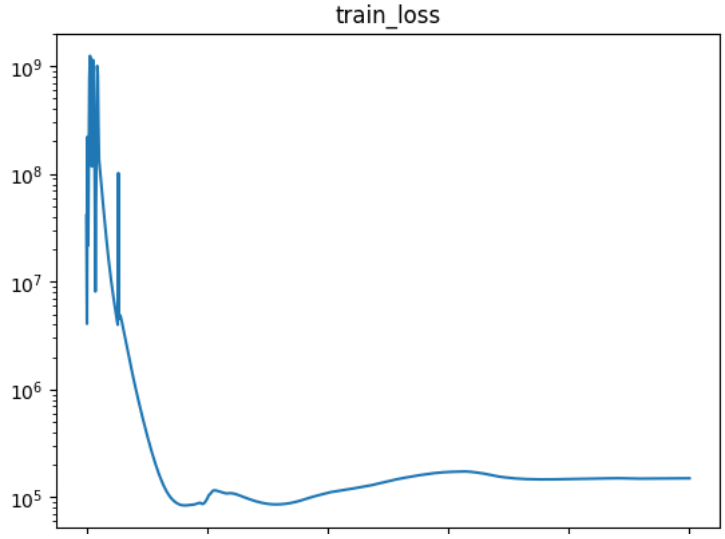
optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 5e-07 , minimum_RMSE: 18669.13 , epoch: 1000 , test_loss: 19426 , train_loss:



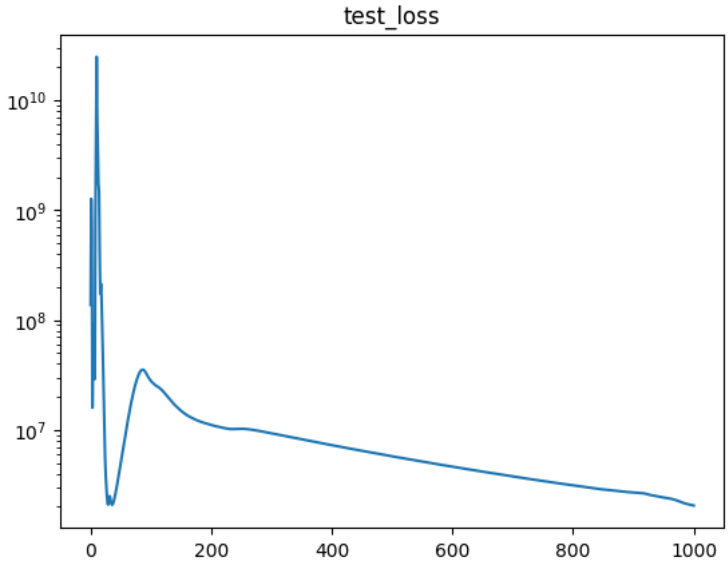


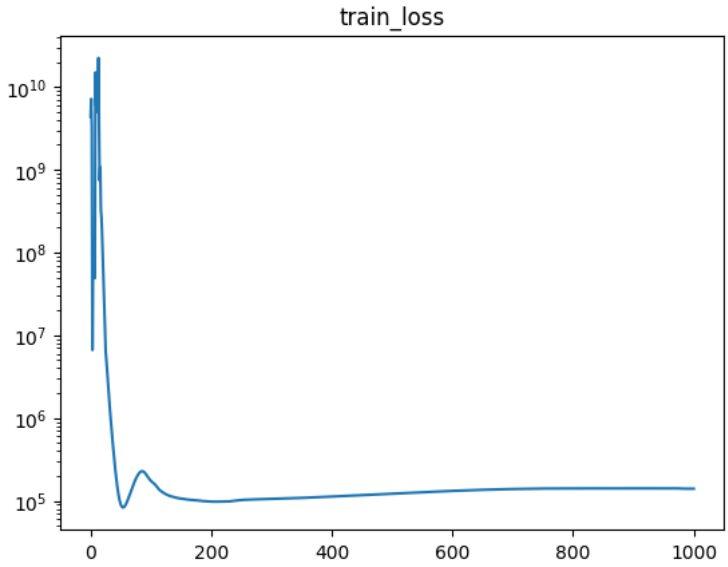
optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 0.1 , minimum_RMSE: 899.54 , epoch: 1000 , test_loss: 1440 , train_loss: 3



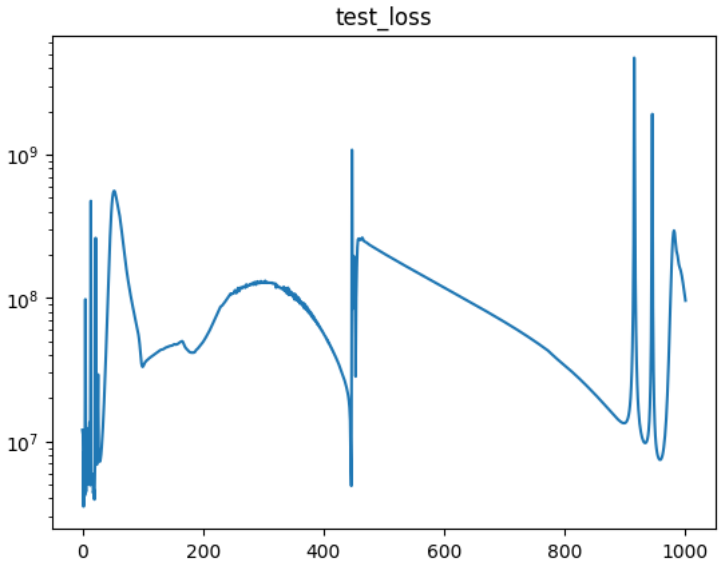


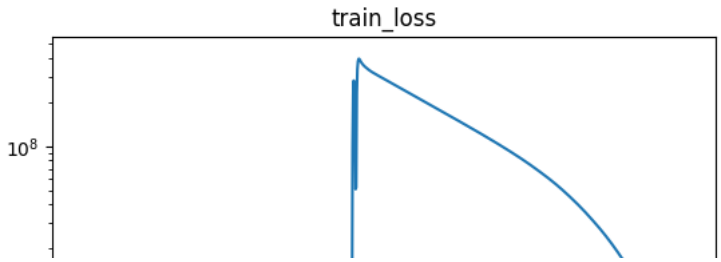
optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 0.5 , minimum_RMSE: 1434.30 , epoch: 1000 , test_loss: 1434 , train_loss:

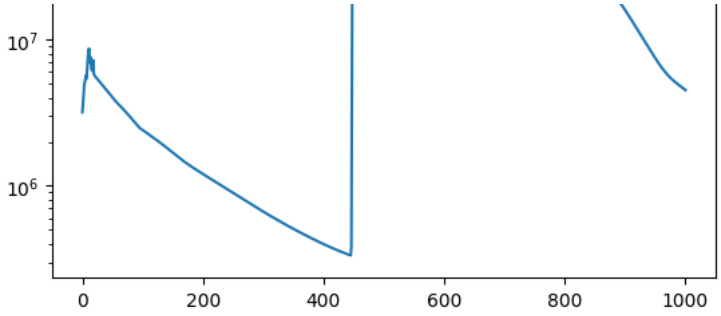




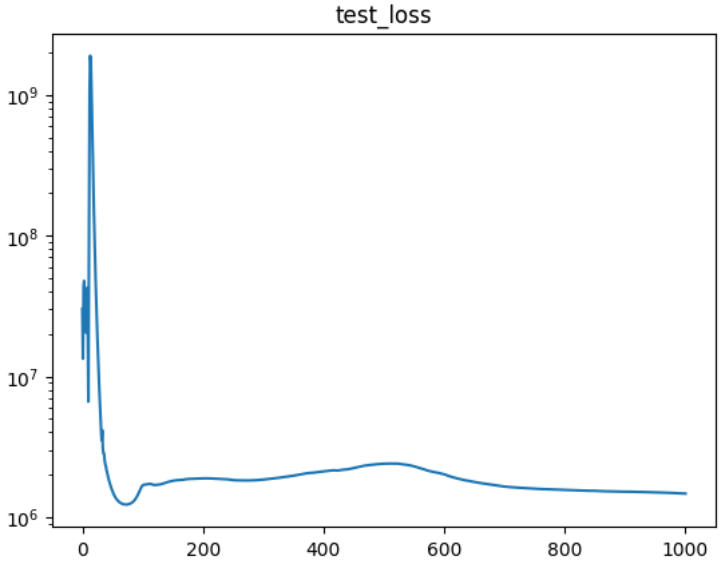
optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 0.01 , minimum_RMSE: 1874.76 , epoch: 1000 , test_loss: 9789 , train_loss:

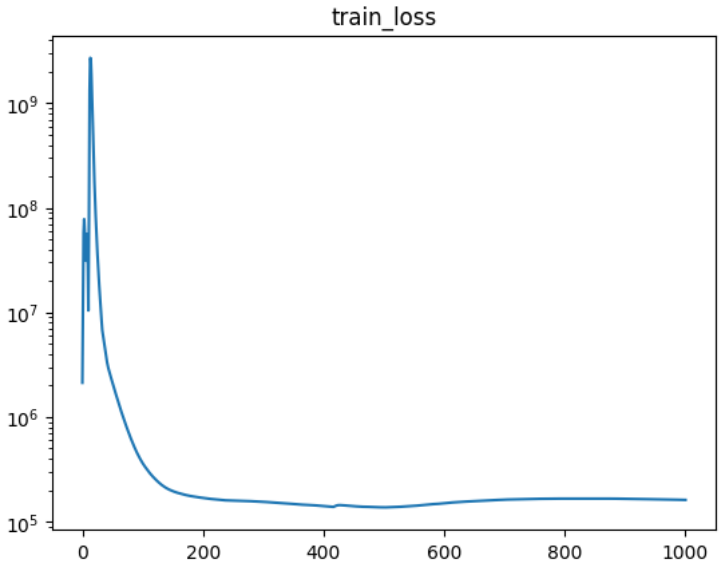




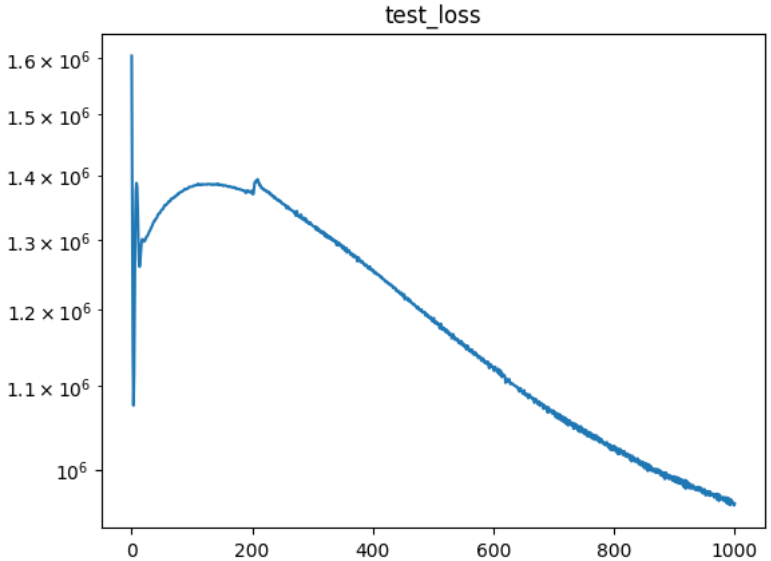


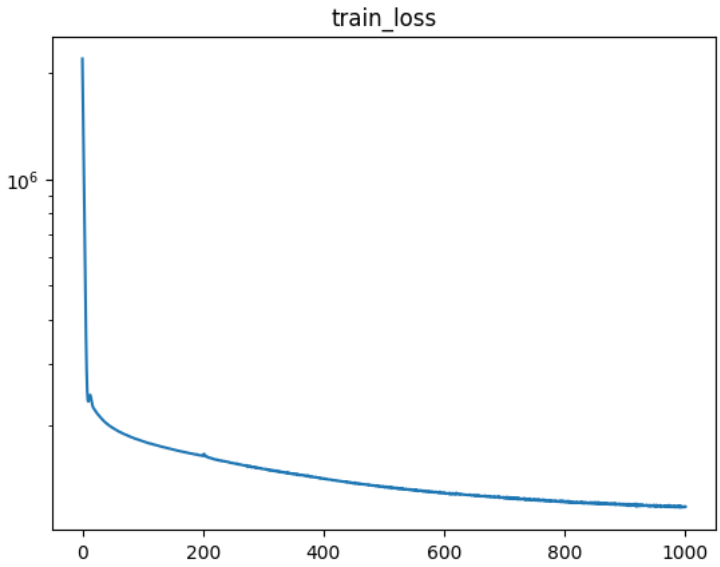
optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 0.05 , minimum_RMSE: 1107.96 , epoch: 1000 , test_loss: 1211 , train_loss:



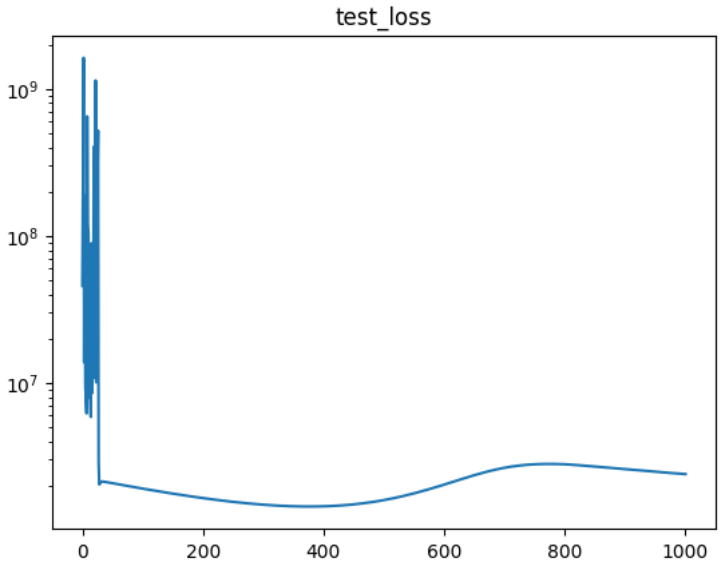


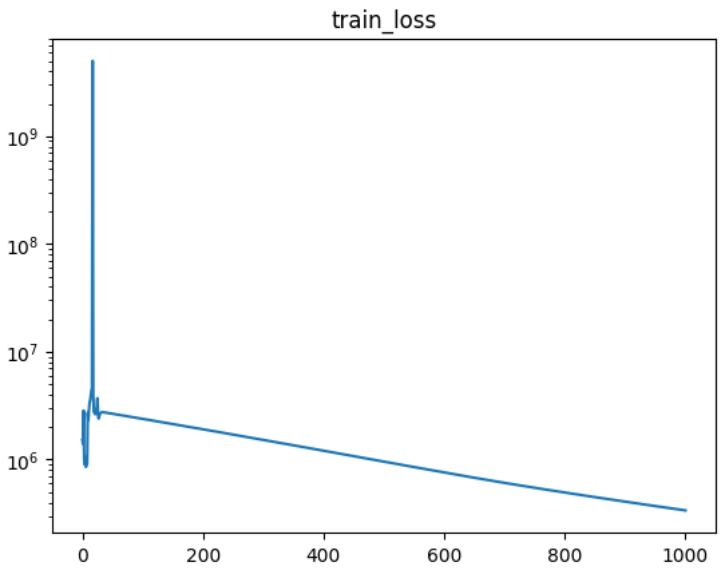
optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 0.001 , minimum_RMSE: 979.86 , epoch: 1000 , test_loss: 980 , train_loss:



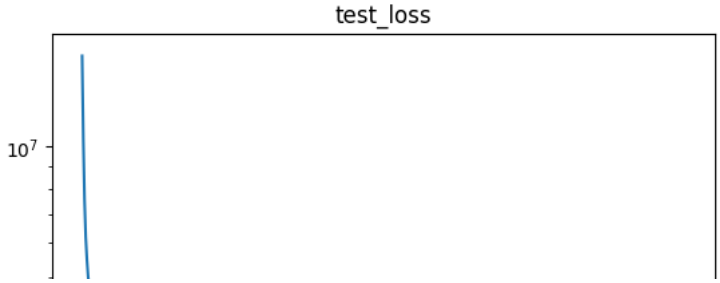


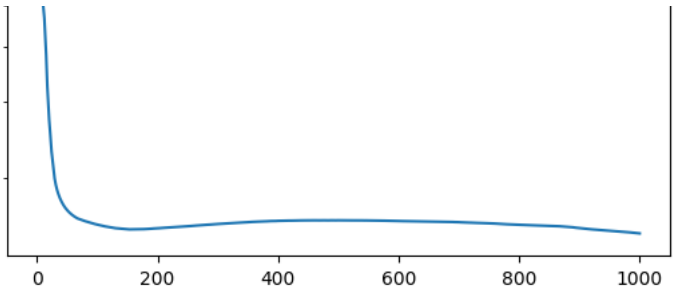
optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 0.005 , minimum_RMSE: 1201.15 , epoch: 1000 , test_loss: 1548 , train_loss:

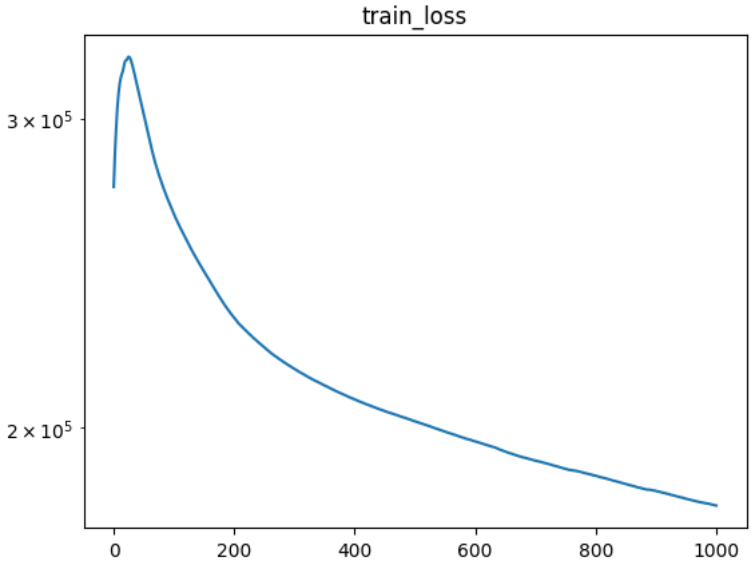




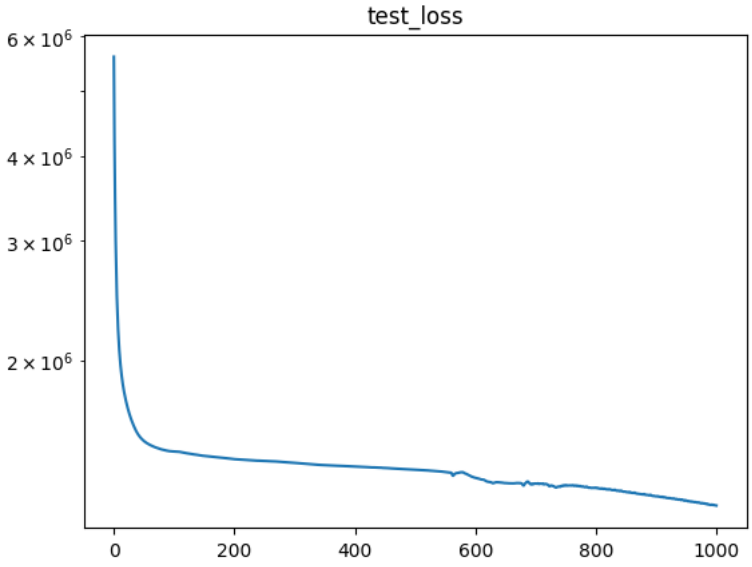
optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 0.0001 , minimum_RMSE: 1221.03 , epoch: 1000 , test_loss: 1221 , train_loss

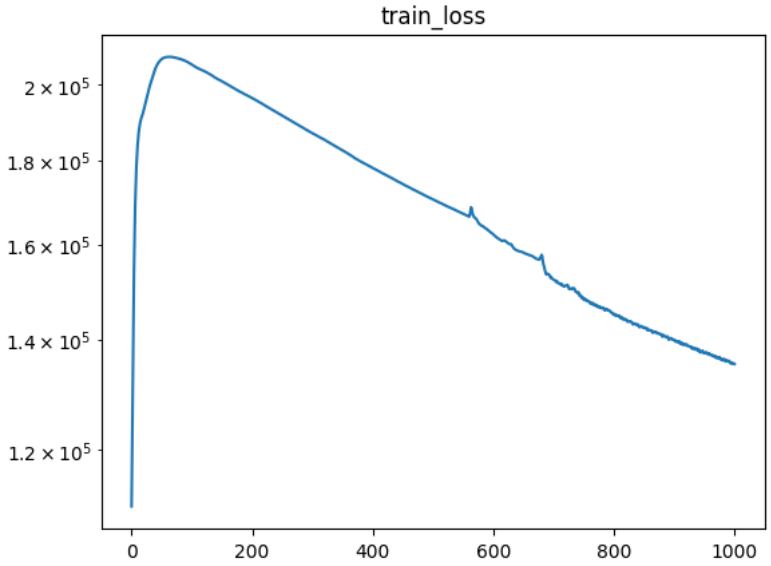




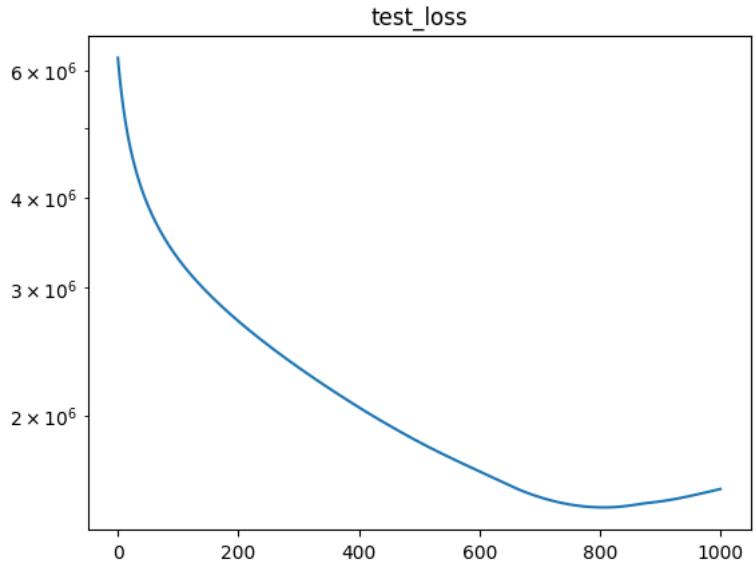


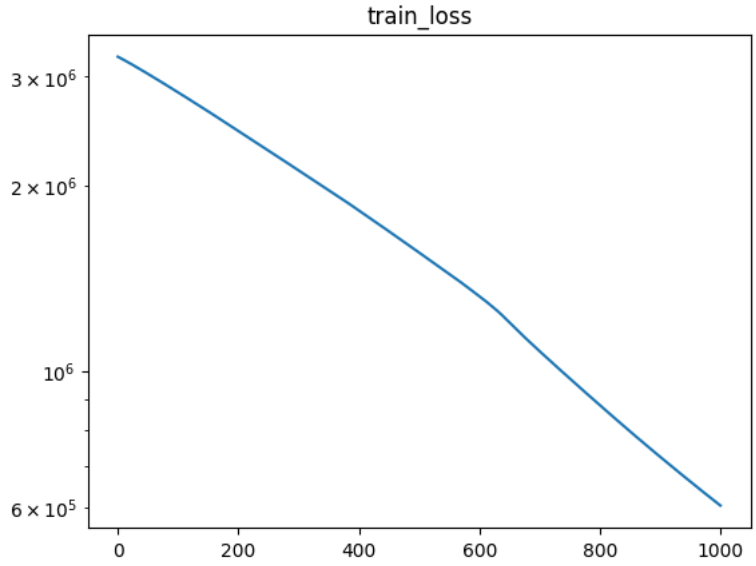
optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 0.0005 , minimum_RMSE: 1104.69 , epoch: 1000 , test_loss: 1104 , train_loss:



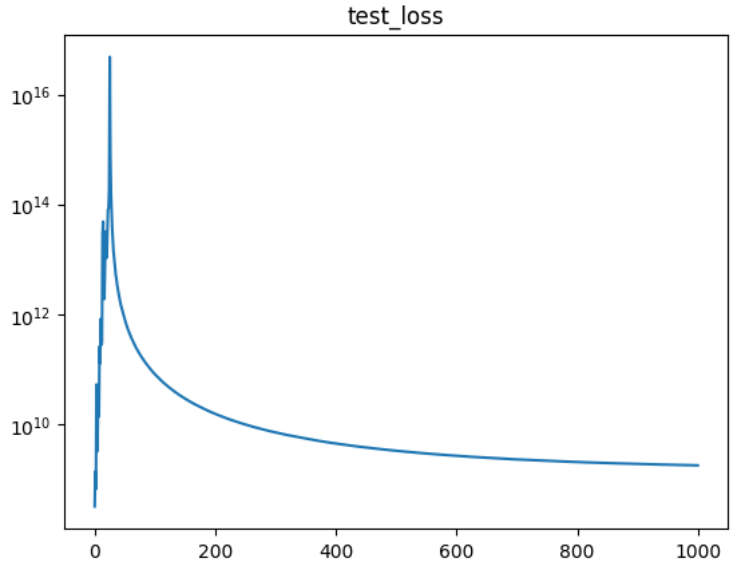


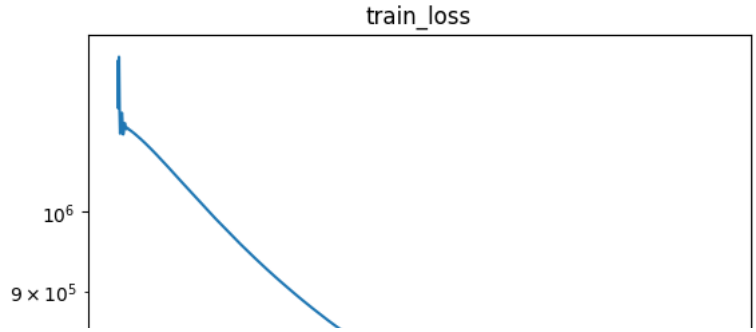
optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-05 , minimum_RMSE: 1221.30 , epoch: 1000 , test_loss: 1257 , train_loss:

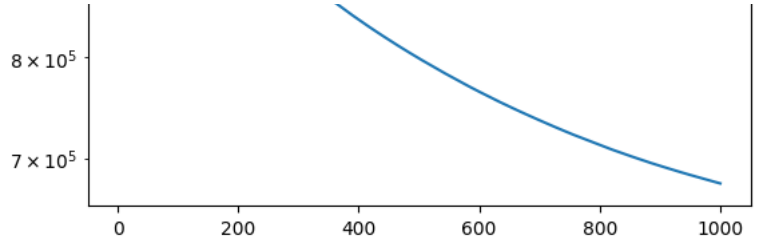




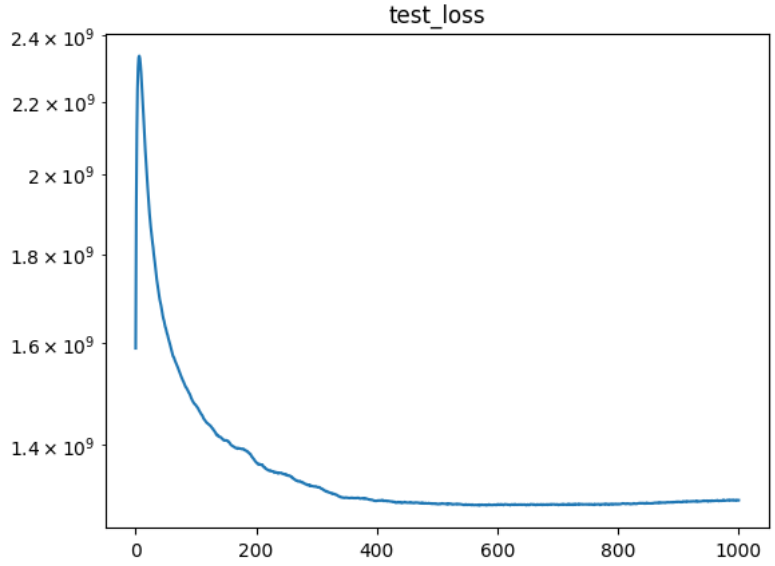
optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-05 , minimum_RMSE: 17972.77 , epoch: 1000 , test_loss: 42466 , train_loss

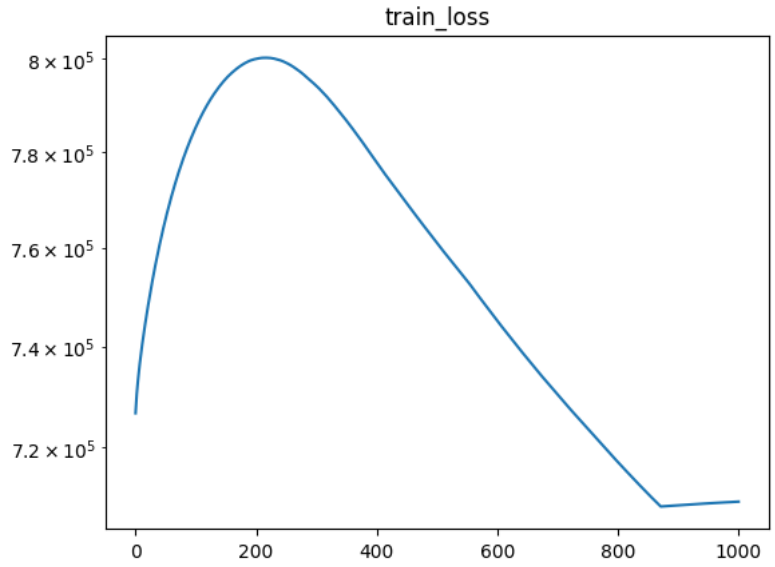




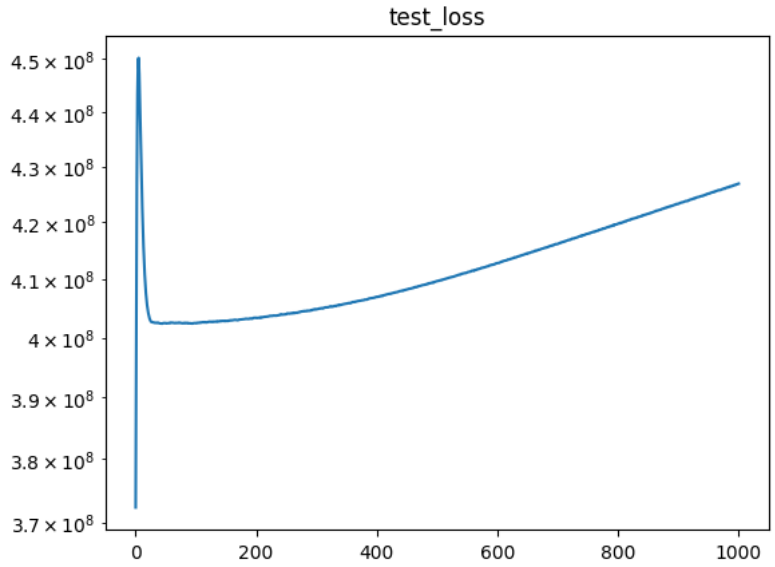


optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-06 , minimum_RMSE: 35950.72 , epoch: 1000 , test_loss: 36074 , train_loss

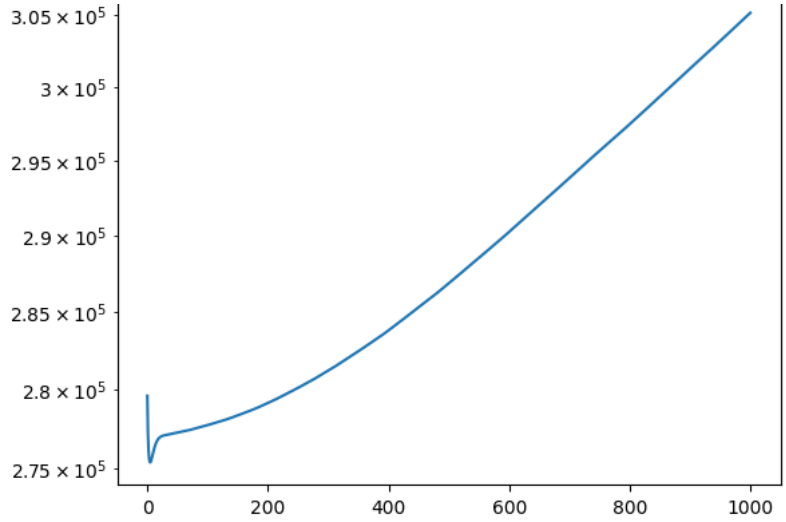




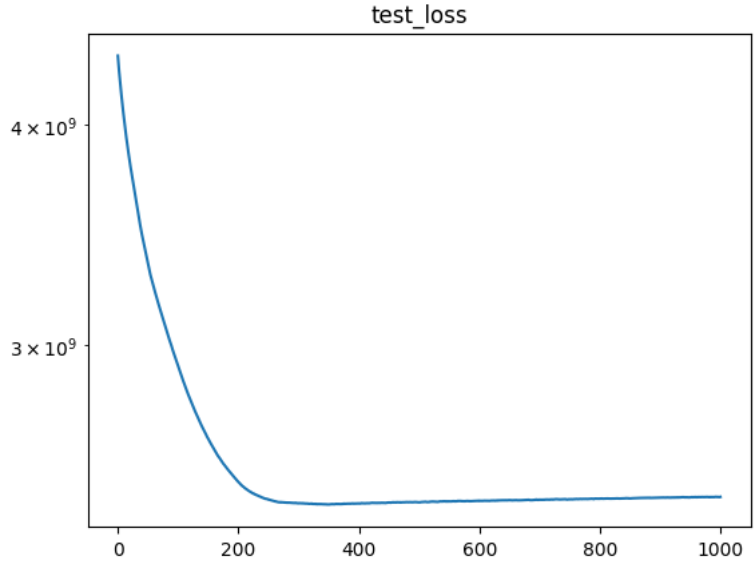
optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-06 , minimum_RMSE: 19295.79 , epoch: 1000 , test_loss: 20662 , train_loss

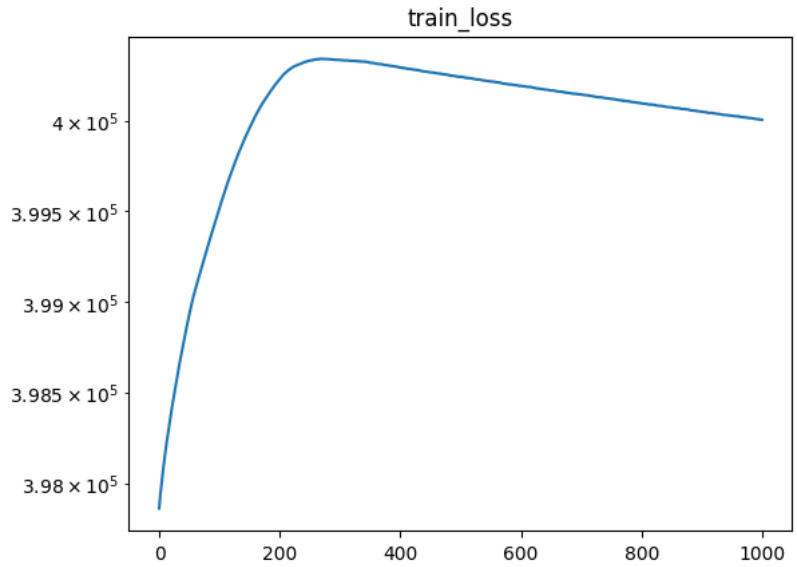


train_loss

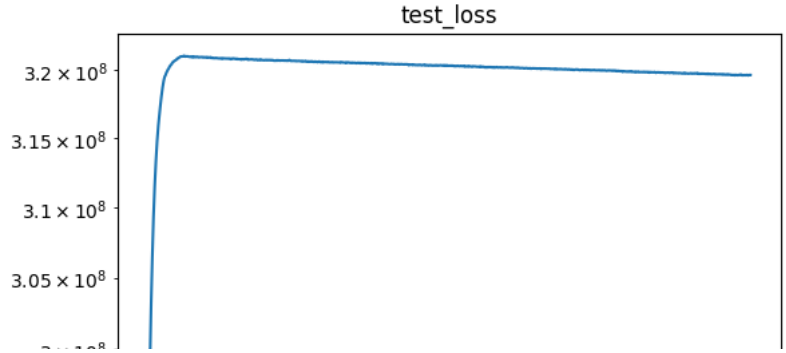


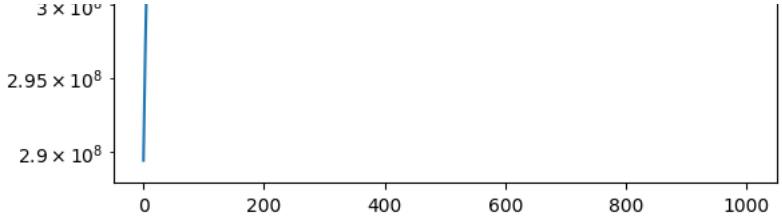
optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-07 , minimum_RMSE: 49366.19 , epoch: 1000 , test_loss: 49612 , train_loss

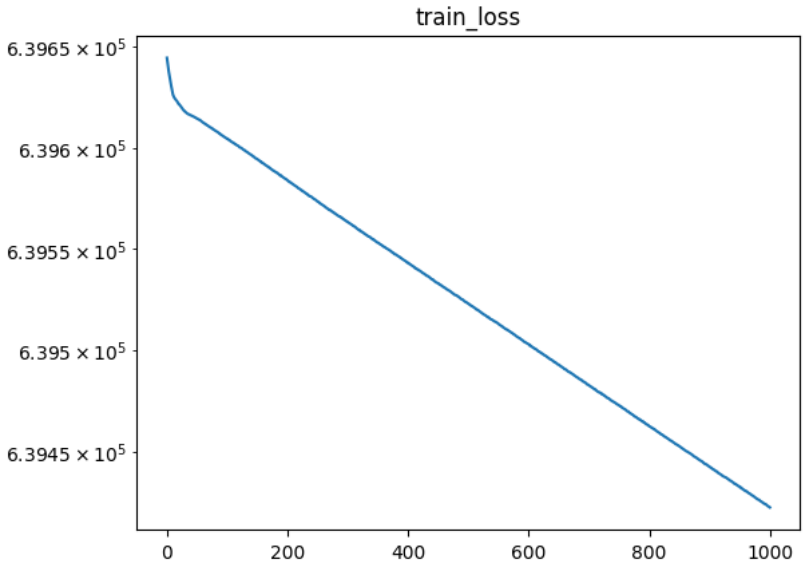




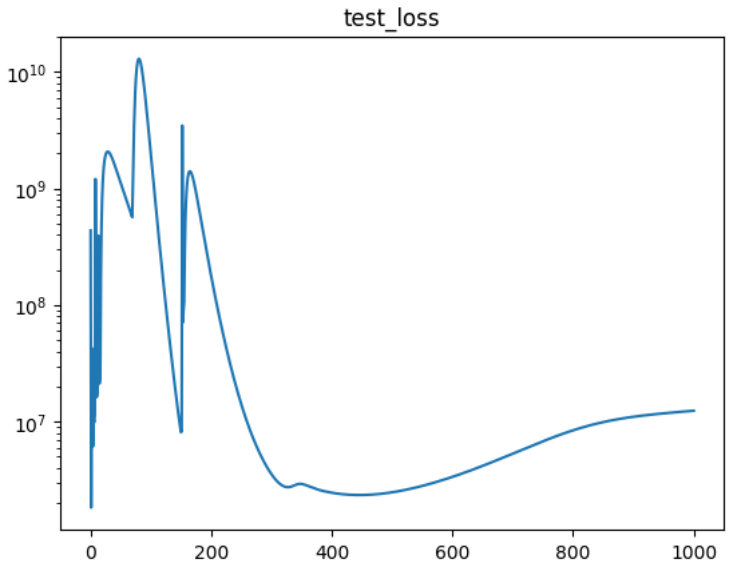
optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-07 , minimum_RMSE: 17013.55 , epoch: 1000 , test_loss: 17878 , train_loss

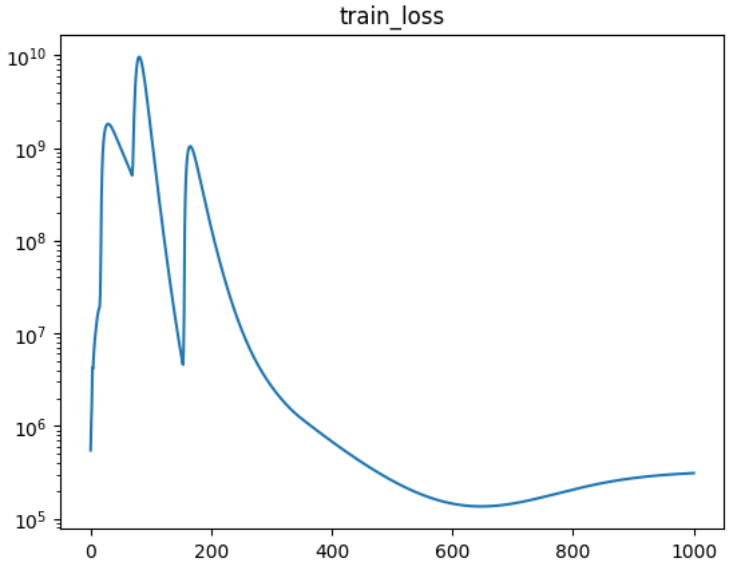




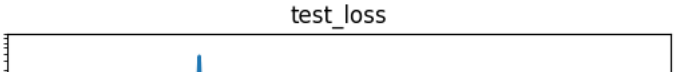


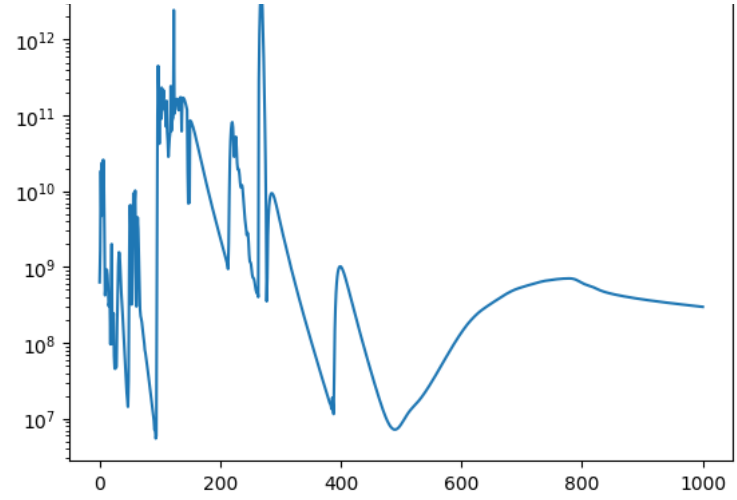
optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 0.1 , minimum_RMSE: 1354.85 , epoch: 1000 , test_loss: 3524 , train_loss:



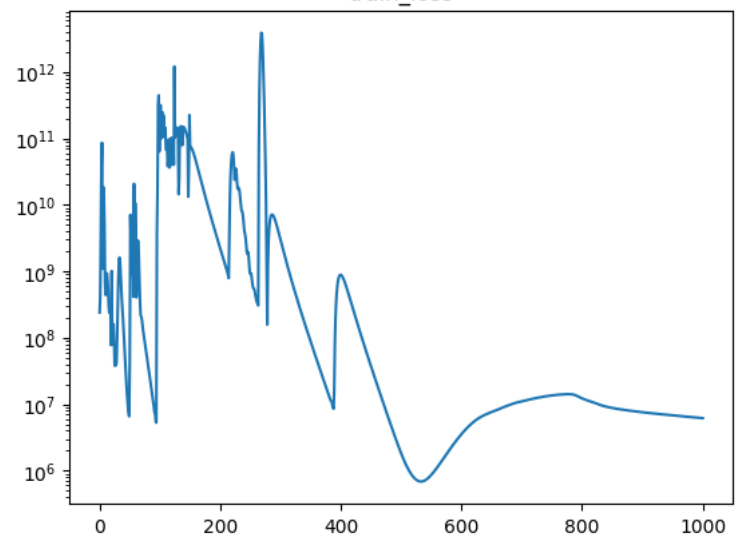


optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 0.5 , minimum_RMSE: 2343.98 , epoch: 1000 , test_loss: 17288 , train_loss:



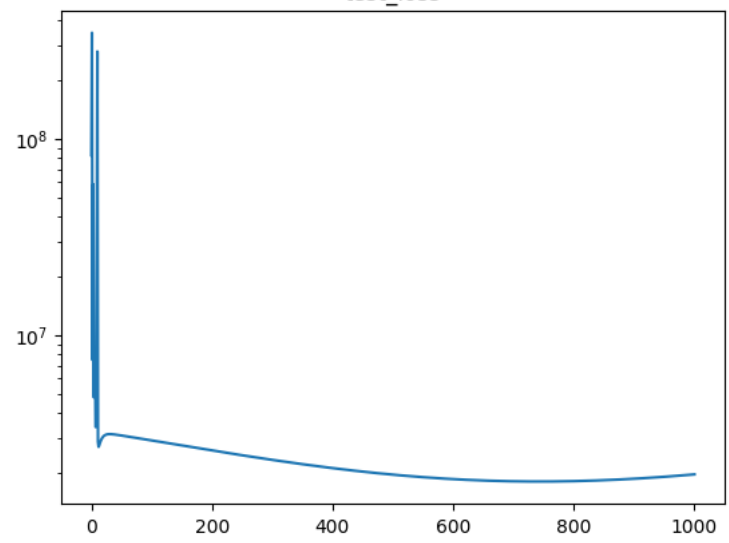


train_loss

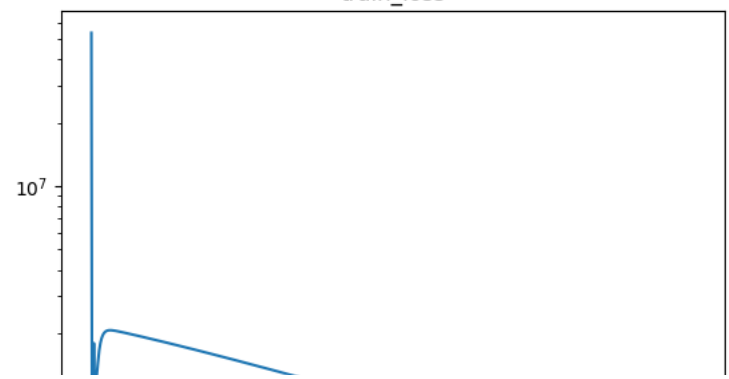


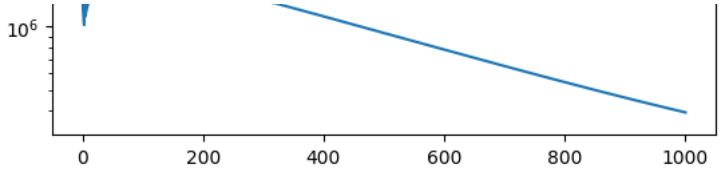
optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 0.01 , minimum_RMSE: 1339.04 , epoch: 1000 , test_loss: 1396 , train_loss:

test_loss

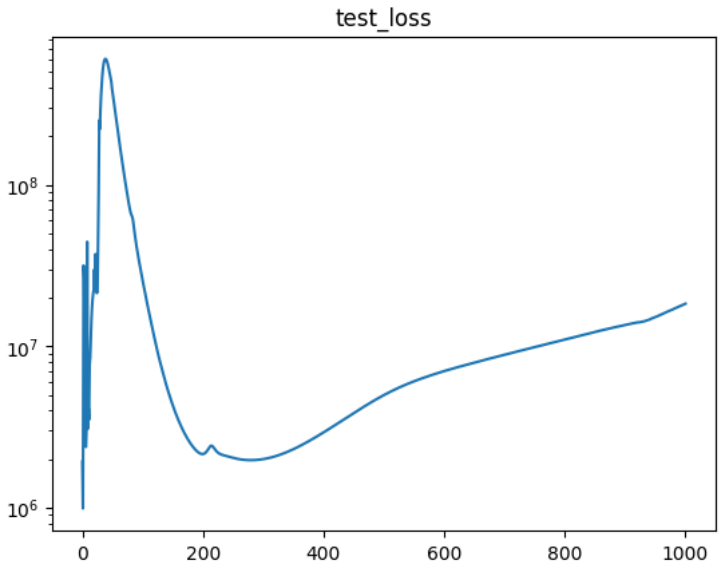


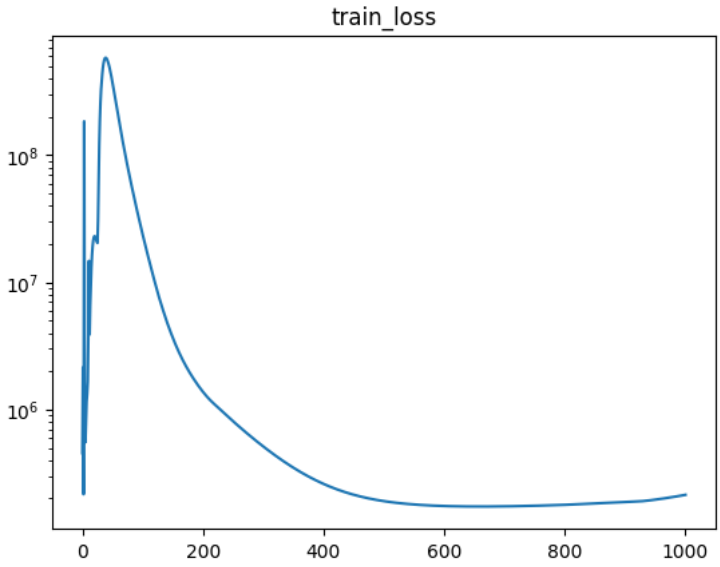
train_loss



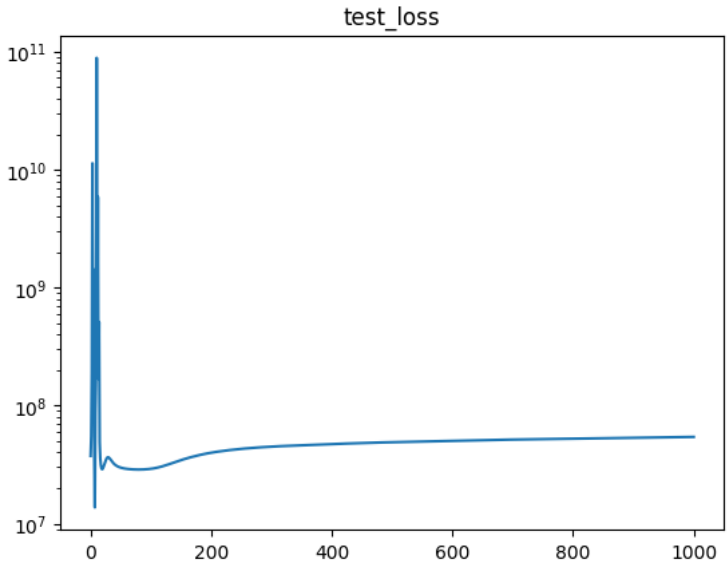


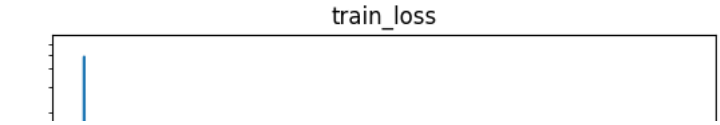
optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 0.05 , minimum_RMSE: 993.79 , epoch: 1000 , test_loss: 4280 , train_loss:

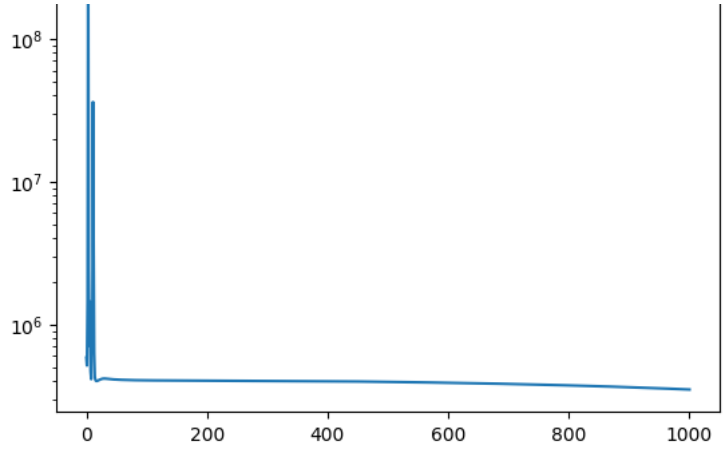




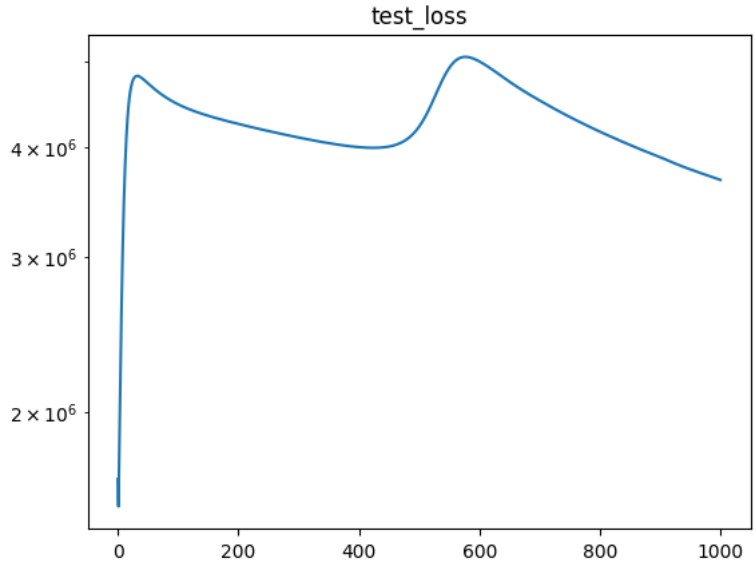
optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 0.001 , minimum_RMSE: 3701.65 , epoch: 1000 , test_loss: 7364 , train_loss:

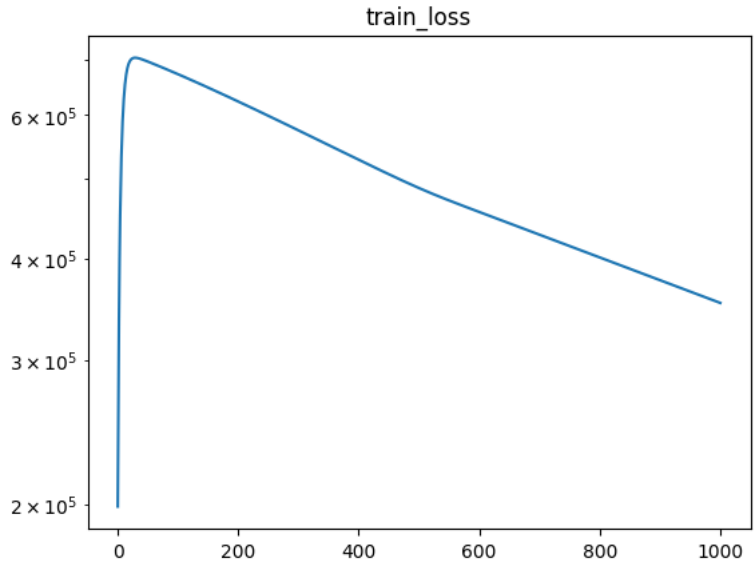




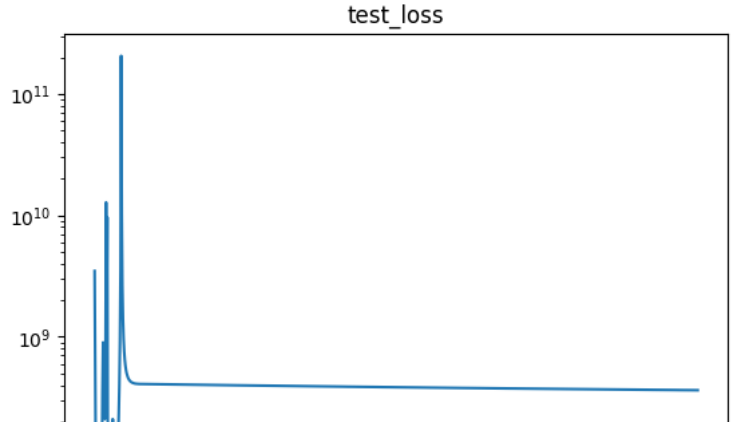


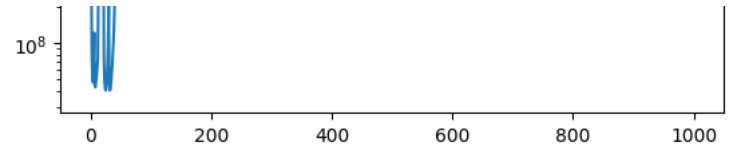
optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 0.005 , minimum_RMSE: 1250.16 , epoch: 1000 , test_loss: 1915 , train_loss:

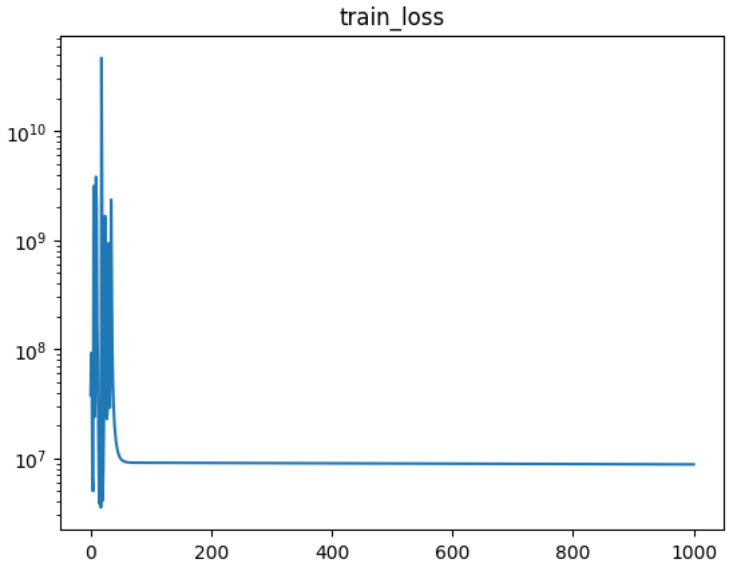




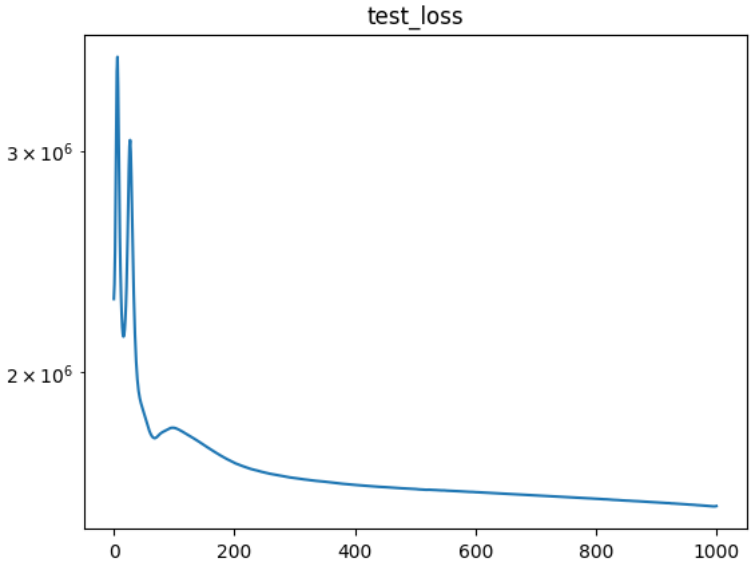
optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 0.0001 , minimum_RMSE: 6387.49 , epoch: 1000 , test_loss: 19039 , train_loss

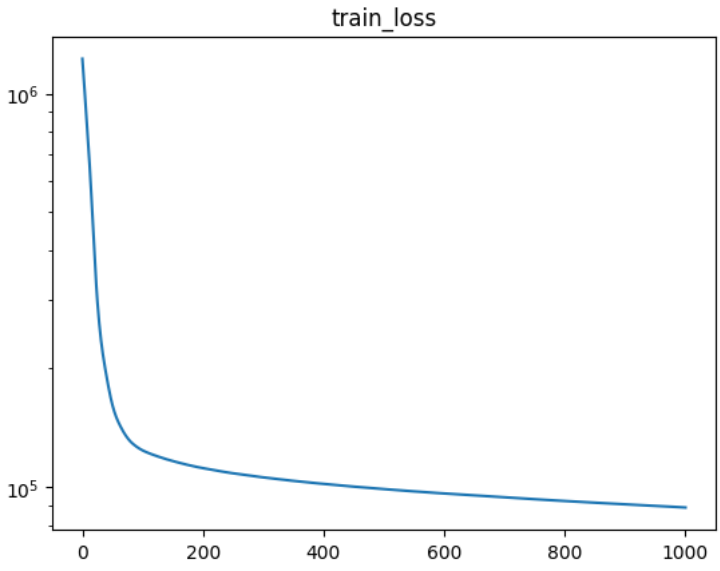




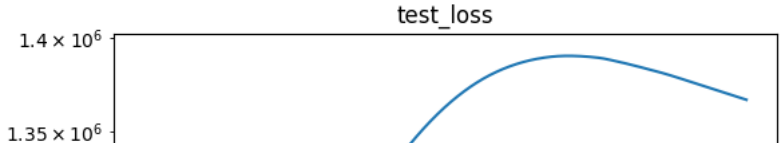


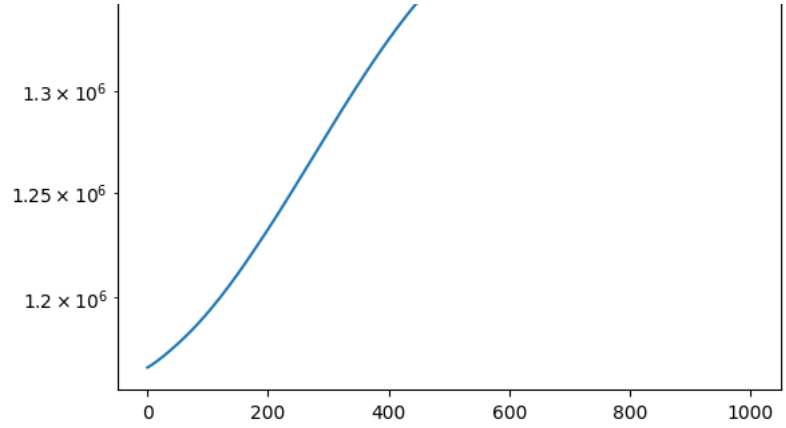
optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 0.0005 , minimum_RMSE: 1250.50 , epoch: 1000 , test_loss: 1250 , train_loss

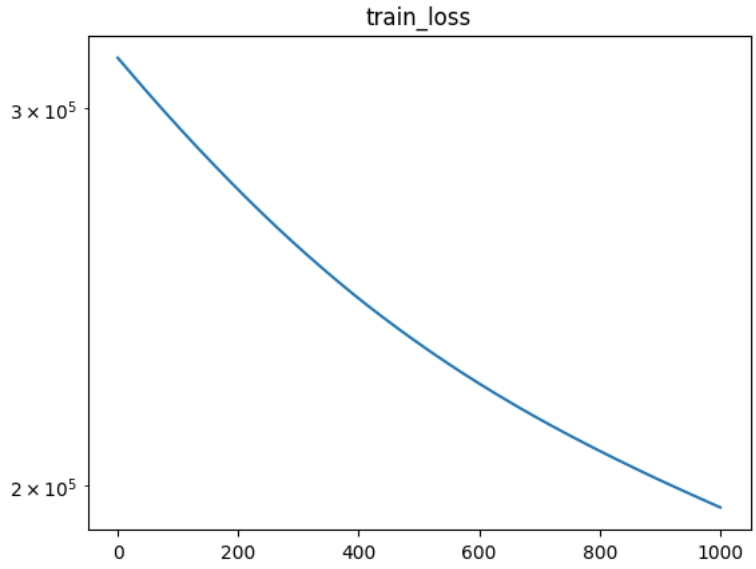




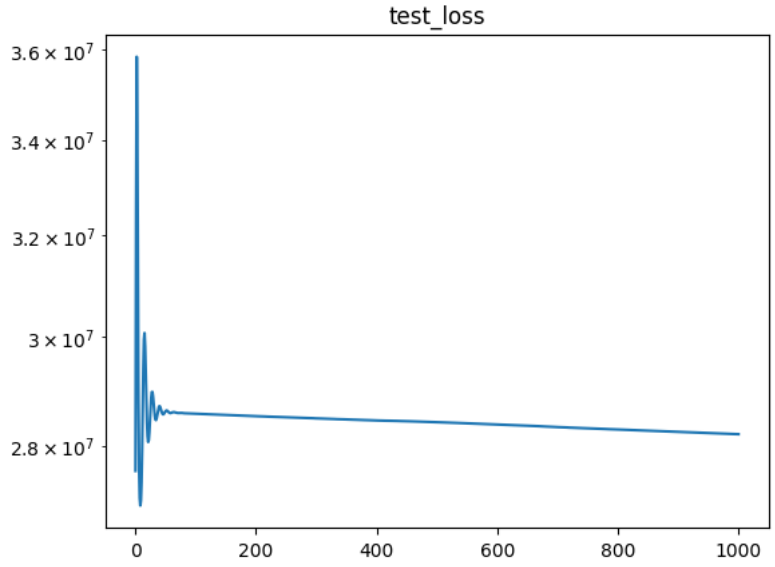
optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-05 , minimum_RMSE: 1080.77 , epoch: 1000 , test_loss: 1169 , train_loss:

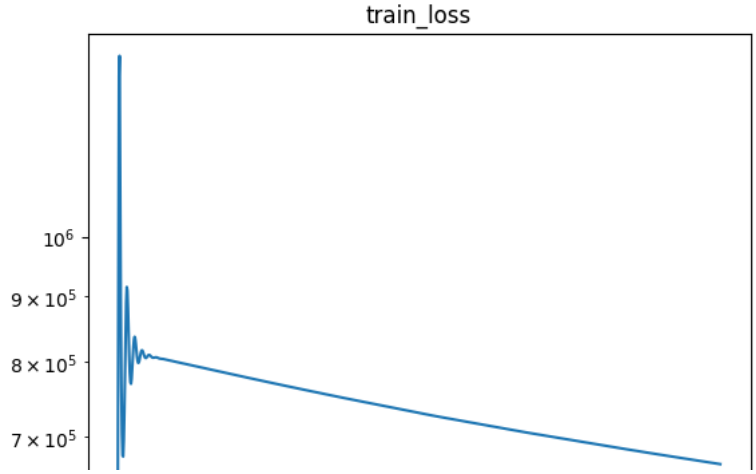


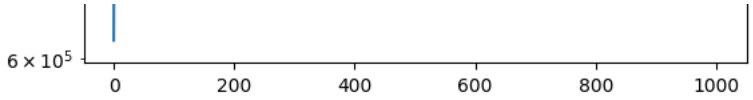




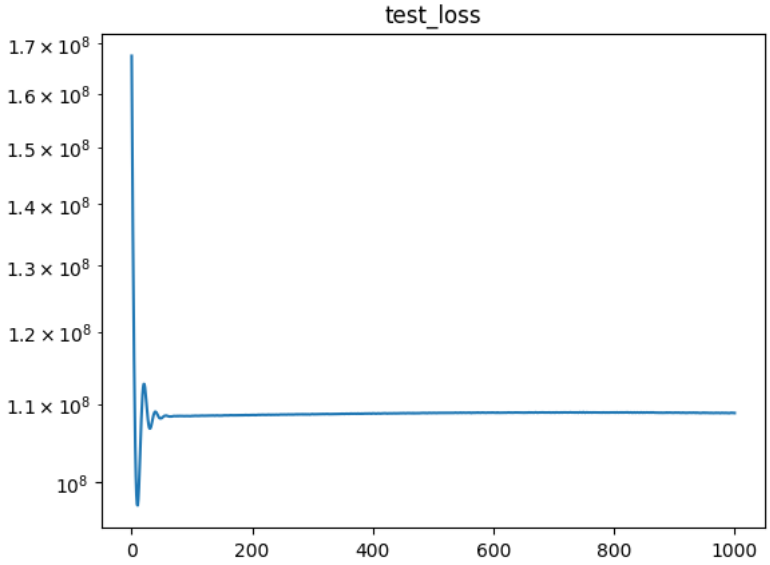
optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 5e-05 , minimum_RMSE: 5192.13 , epoch: 1000 , test_loss: 5310 , train_loss:

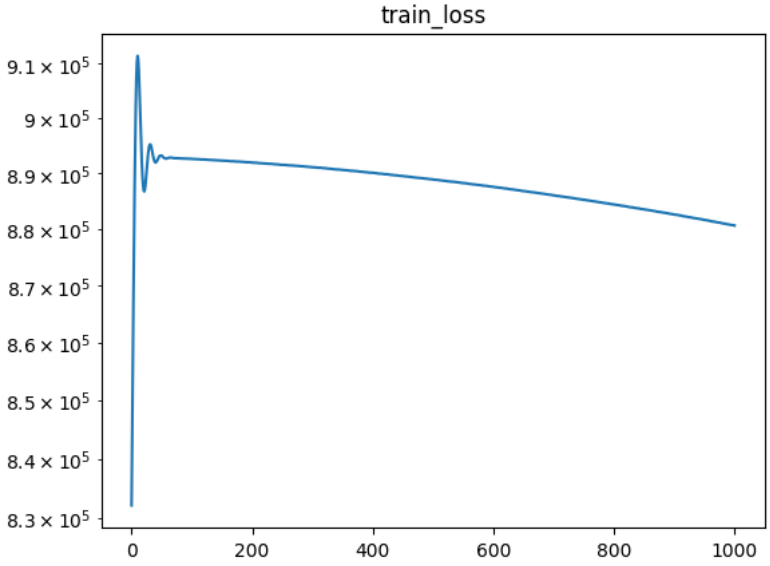




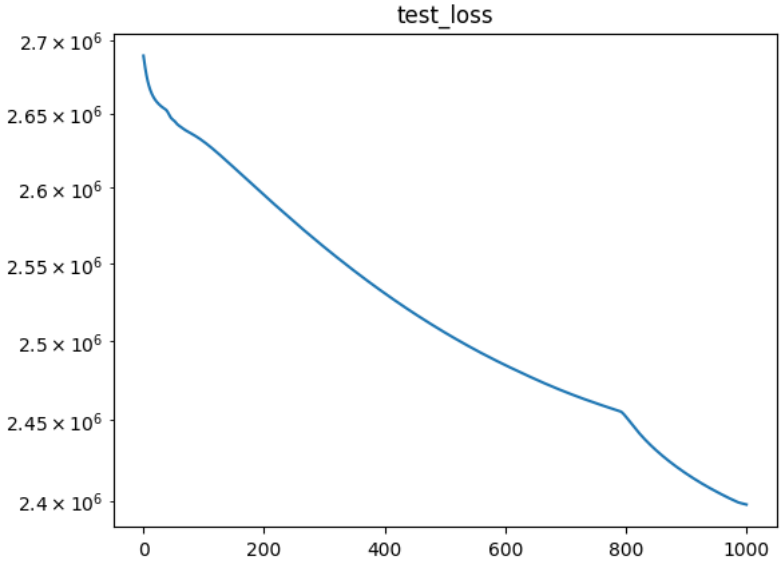


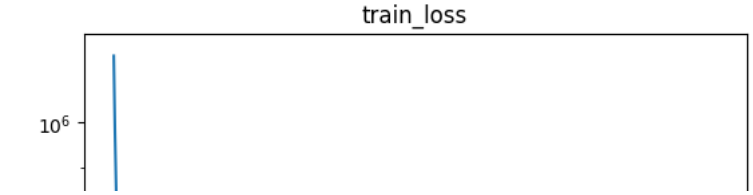
optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-06 , minimum_RMSE: 9862.93 , epoch: 1000 , test_loss: 10429 , train_loss:

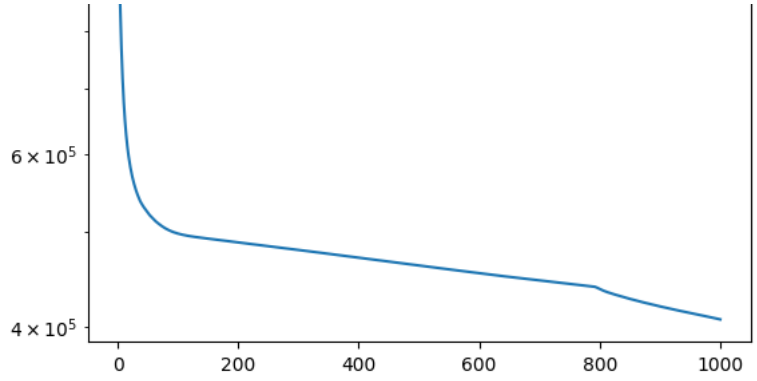




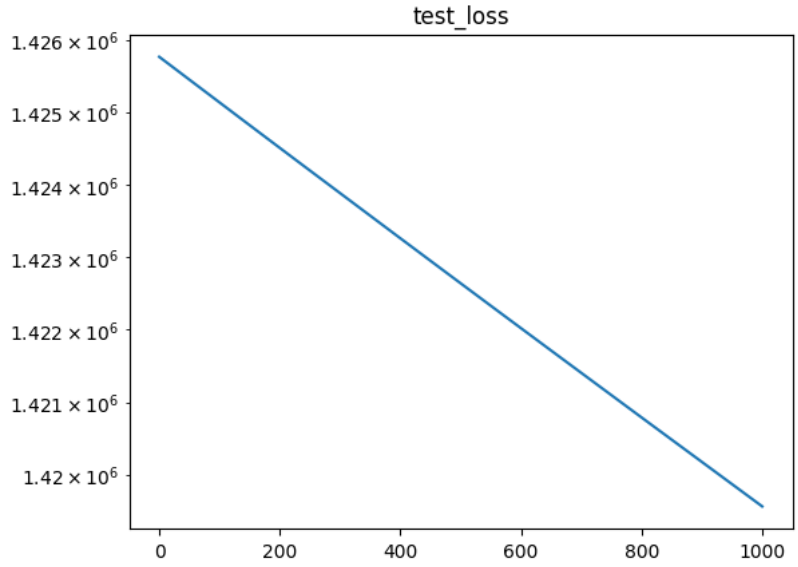
optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 5e-06 , minimum_RMSE: 1548.41 , epoch: 1000 , test_loss: 1548 , train_loss:

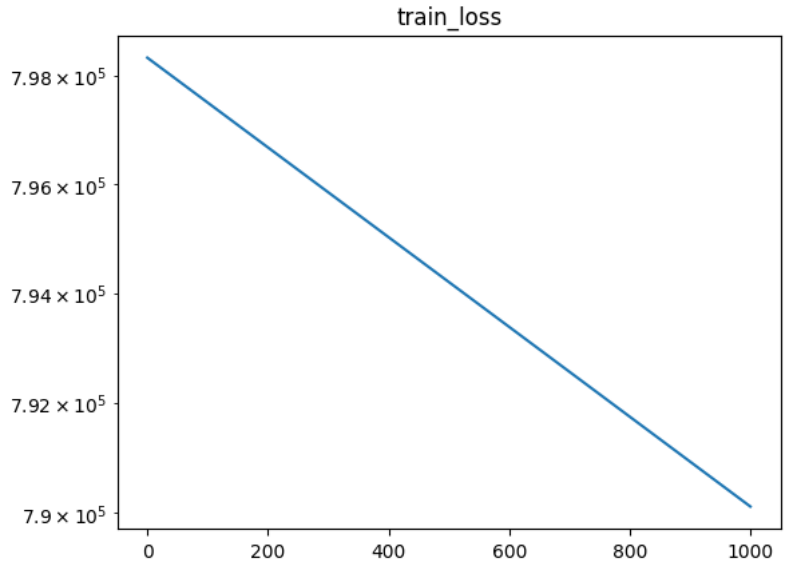




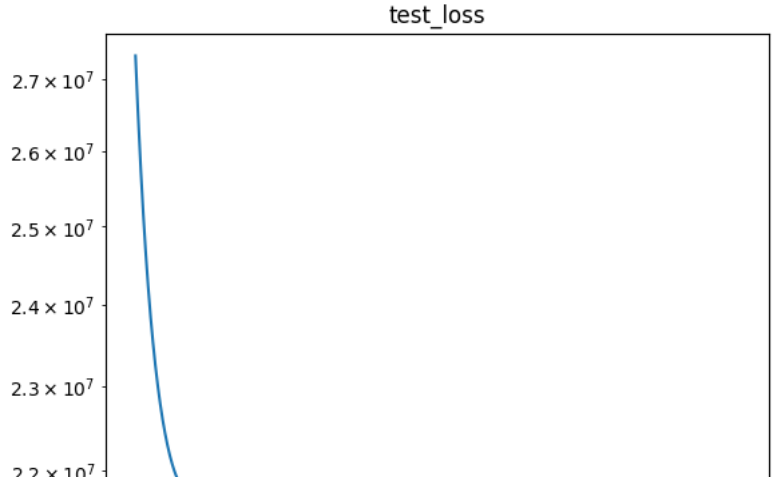


optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-07 , minimum_RMSE: 1191.46 , epoch: 1000 , test_loss: 1191 , train_loss:



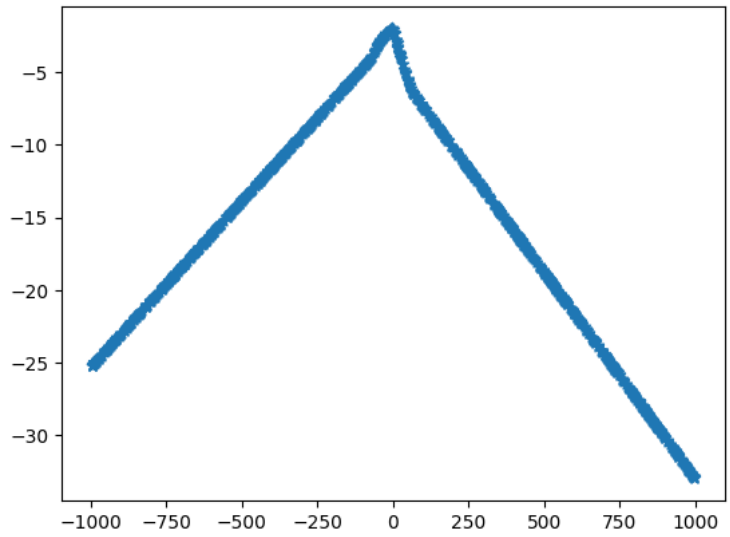


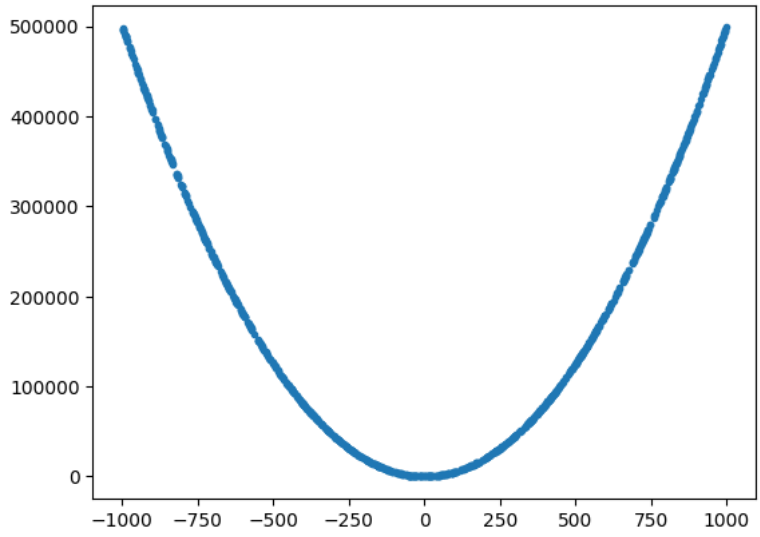
optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 5e-07 , minimum_RMSE: 4649.74 , epoch: 1000 , test_loss: 4649 , train_loss:



```
1
```

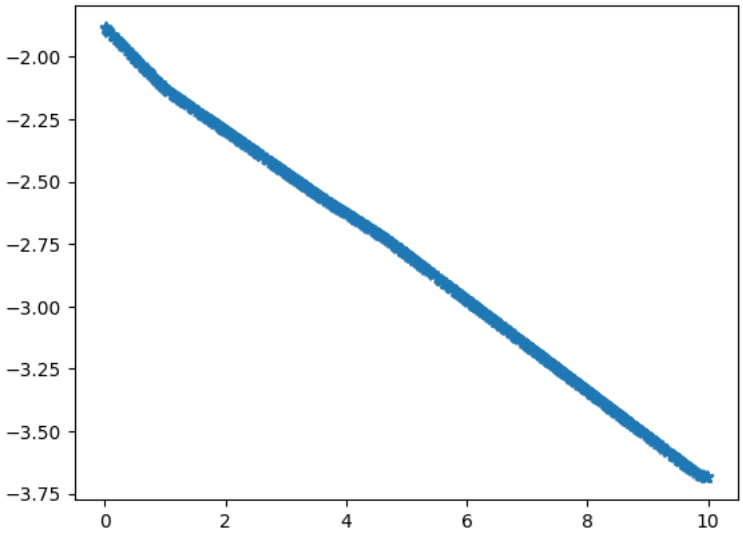
```
model.eval()
y_pred = model(X2)
plt.scatter(X2[:,1].cpu().detach().numpy(),y_pred.cpu().detach().numpy(), marker='*')
# plt.scatter(X2[:,1],y_pred, marker='*')
plt.show()


plt.scatter(X2[:,1] ,y2, marker='.', )
plt.show()


print('all done')
```






```
all done
```

```
model.eval()
y_pred = model(X3)
plt.scatter(X3[:,0].cpu().detach().numpy(),y_pred.cpu().detach().numpy(), marker='*')
# plt.scatter(X3[:,1],y_pred, marker='*')
plt.show()


plt.scatter(X3[:,0] ,y3, marker='.', )
plt.show()


print('all done')
```




```
plt.plot(history)
plt.yscale('log')
plt.title('test_loss')
plt.show()

plt.plot(history_train)
plt.title('train_loss')
plt.yscale('log')
plt.show()
```