

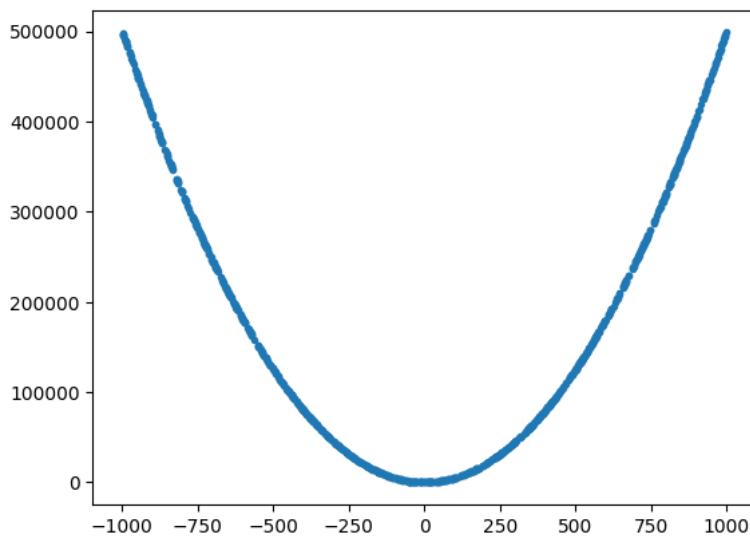
```
1 from google.colab import drive
2 drive.mount('/content/drive')

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

1
2 from sklearn.model_selection import train_test_split
3 from sklearn.datasets import fetch_california_housing
4 from sklearn.preprocessing import StandardScaler
5 from torchvision import datasets, transforms
6
7 import copy
8 import numpy as np
9 import matplotlib.pyplot as plt
10 import numpy as np
11 import pandas as pd
12 import torch
13 import torch.nn as nn
14 import torch.nn.functional as F
15 import torch.optim as optim
16 import tqdm
17
18
19

1 N, D_in, D_out = 1000, 2, 1
2
3
4 # m =1
5 X2 = []
6 y2 = []
7 for j in range(N):
8     X2.append([])
9     #y.append([])
10    for i in range(D_out):
11        X2[-1].append( 1 )
12        X2[-1].append( np.random.uniform(low=-1000.0, high=1000.0, size=None) )
13        y2.append( (X2[-1][-1]**2) / (2*X2[-1][-2]) )
14
15
16 X2 = torch.Tensor(X2)
17 y2 = torch.Tensor(y2)
18
19
20 # p =1
21 X3 = []
22 y3 = []
23 for j in range(N):
24     X3.append([])
25     #y.append([])
26     for i in range(D_out):
27         X3[-1].append( np.random.uniform(low=0.0, high=10.0, size=None) )
28         X3[-1].append( 1 )
29         y3.append( (X3[-1][-1]**2) / (2*X3[-1][-2]) )
30
31
32 X3 = torch.Tensor(X3)
33 y3 = torch.Tensor(y3)
34
35 # X2_test = torch.tensor(X2, dtype=torch.float32)
36 # y2_test = torch.tensor(y2, dtype=torch.float32).reshape(-1, 1)
37 # X3_test = torch.tensor(X3, dtype=torch.float32)
38 # y3_test = torch.tensor(y3, dtype=torch.float32).reshape(-1, 1)

39
40 plt.scatter(X2[:,1] ,y2, marker='.', )
41 plt.show()
42 plt.scatter(X3[:,0] ,y3, marker='.', )
43 plt.show()
44
45
```



```

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'>

```

```
1 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

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

[2, 16, 8, 1]

1

1

```
1 #default
2 """
3 Bsize: 500 learningRate: 0.03 RMSE: 111.31 1000 loss: 515.4253401318177
4 """
5 """
6 # for learningRate in range(10): #
7
8
9
10 dim_list = [2, 16, 8, 1]
11
12 #If data is less complex and is having fewer dimensions or features then neural networks with 1 to 2 hidden layers would work.
13 # If data is having large dimensions or features then to get an optimum solution, 3 to 5 hidden layers can be used.
14
15
16 # for Bsize in [500]:
17 for opt in [
18     'Adadelta',
19     'Adagrad',
20     'Adam',
21     'AdamW',
22     'SparseAdam',
23     'Adamax',
24     'ASGD',
25     'LBFGS',
26     'NAdam',
27     'RAdam',
28     'RMSprop',
29     'Rprop',
30     'SGD'
31 ]:
32     if opt == 'Adam':# -----delete later-----
33         continue# -----delete later-----
34     for Bsize in [ 10, 50, 100, 500]:
35         # print('Bsize:', Bsize, end=' ')
36         #before_loss_twos = torch.ones(Bsize, dtype=torch.float32)*2
37         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 ]:
```

```

38         # print('learningRate:', learningRate)
39         # Define the model
40
41         make_model = 'model = nn.Sequential('
42         for layer_num in range( len(dim_list) ):
43             if layer_num == len(dim_list) - 1:
44                 make_model = make_model + 'nn.Linear(dim_list[' + str(dim_list[i]) + '], dim_list[' + str(dim_list[i+1]) + '])'
45             else:
46                 make_model = make_model + 'nn.Linear(dim_list[' + str(dim_list[i]) + '], dim_list[' + str(dim_list[i+1]) + ']), nn.LeakyReLU('
47
48         #make_model = make_model +
49
50         # model = nn.Sequential(
51         #     nn.Linear(dim_list[0], dim_list[1]),
52         #     nn.LeakyReLU(),
53         #     nn.Linear(dim_list[1], dim_list[2]),
54         #     nn.LeakyReLU(),
55         #     nn.Linear(dim_list[2], dim_list[3])
56         #     # nn.Linear(2, 3),
57         #     # nn.LeakyReLU(),
58         #     # nn.Linear(3, 2),
59         #     # nn.LeakyReLU(),
60         #     # nn.Linear(2, 1)
61     # )
62
63
64         # loss function and optimizer
65         loss_fn = nn.MSELoss() # mean square error
66         # optimizer = optim.Adam(model.parameters(), lr=learningRate ) # 10 loss: 5157042688.0
67         exec('optimizer = optim.' + opt + '(model.parameters(), lr=learningRate )')
68         n_epochs = 1000+1 # number of epochs to run
69         batch_size = Bsize # size of each batch
70         batch_start = torch.arange(0, len(X_train), batch_size)
71
72         # Hold the best model
73         best_mse = np.inf # init to infinity
74         best_weights = None
75         history = []
76         history_train = []
77         for epoch in range(n_epochs):
78             model.train()
79             with tqdm.tqdm(batch_start, unit="batch", mininterval=0, disable=True) as bar:
80                 bar.set_description(f"Epoch {epoch}")
81                 for start in bar:
82                     # take a batch
83                     X_batch = X_train[start:start+batch_size]
84                     y_batch = y_train[start:start+batch_size]
85                     # forward pass
86                     y_pred = model(X_batch)
87
88                     before_loss = y_pred/y_batch + y_batch/y_pred
89                     before_loss_twos = torch.ones(len(y_pred), dtype=torch.float32)*2
90
91                     loss = loss_fn(before_loss, before_loss_twos)
92                     # backward pass
93                     optimizer.zero_grad()
94                     loss.backward()
95                     # update weights
96                     optimizer.step()
97                     # print progress
98                     bar.set_postfix(mse=float(loss))
99
100                    loss = float(loss)
101
102                    history_train.append(loss)
103                    # evaluate accuracy at end of each epoch
104                    model.eval()
105                    y_pred = model(X_test)
106
107                    before_loss = y_pred/y_test + y_test/y_pred
108                    before_loss_twos = torch.ones(len(y_pred), dtype=torch.float32)*2
109                    mse = loss_fn(before_loss, before_loss_twos)
110
111                    mse = float(mse)
112
113                    history.append(mse)
114                    if mse < best_mse:
115                        best_mse = mse
116                        best_weights = copy.deepcopy(model.state_dict())
117
118                    # if epoch % 100 == 0:
119                    #     print('epoch: %d' % epoch, 'test_loss: %d' % np.sqrt(mse), 'train_loss: %d' % np.sqrt(loss))
120

```

```
121      # restore model and return best accuracy
122      model.load_state_dict(best_weights)
123      # print("MSE: %.2f" % best_mse)
124      # print("RMSE: %.2f" % np.sqrt(best_mse))
125      print('optimizer:', opt, ', n_of_data:', N, ', Bsize:', Bsize, ', learningRate:', learningRate, ", minimum_RMSE: %.2f" % np.sqrt(best
126      # print(y_pred[:10])
127      # print(y_test[:10])
128      plt.plot(history)
129      plt.yscale('log')
130      plt.title('test_loss')
131      plt.show()
132
133      plt.plot(history_train)
134      plt.title('train_loss')
135      plt.yscale('log')
136      plt.show()
137      print('all done')
138      # -----
139      # -----
```

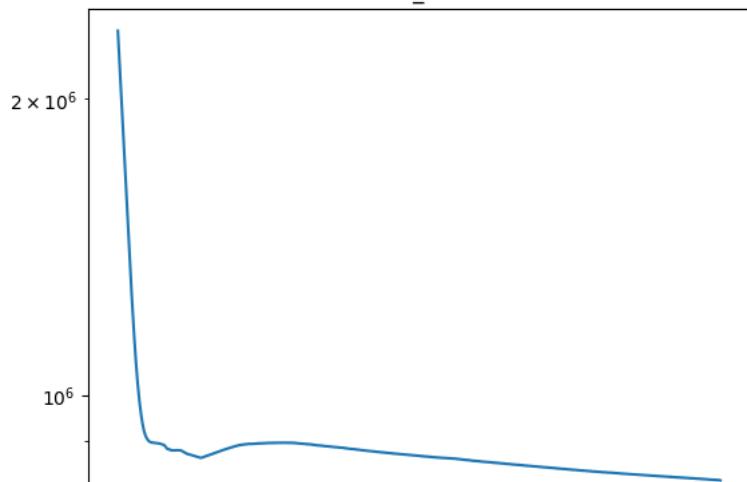


23. 8. 2. 오후 3:59

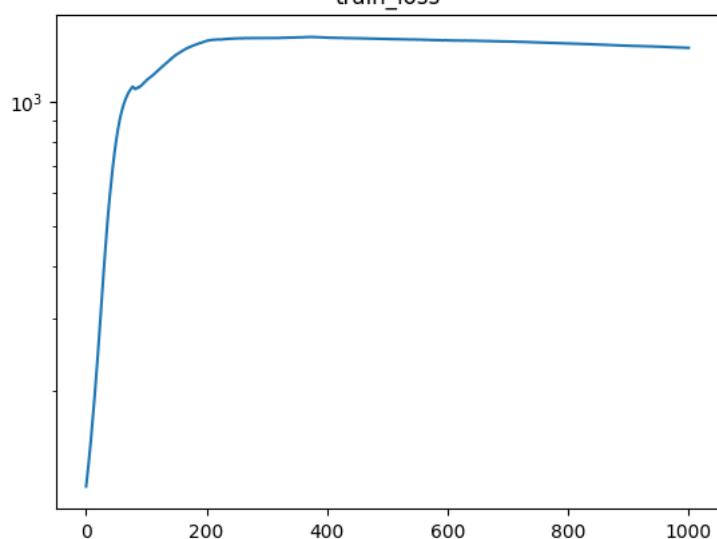
predictingKineticE.ipynb - Colaboratory

```
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/loss.py:500: UserWarning: Using a target size (torch.Size([300])) that is different from the input size (torch.Size([10])) - returning a loss scalar instead of a vector. This behaviour will change in a future PyTorch release.
  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 from the input size (torch.Size([10])) - returning a loss scalar instead of a vector. This behaviour will change in a future PyTorch release.
  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:
```

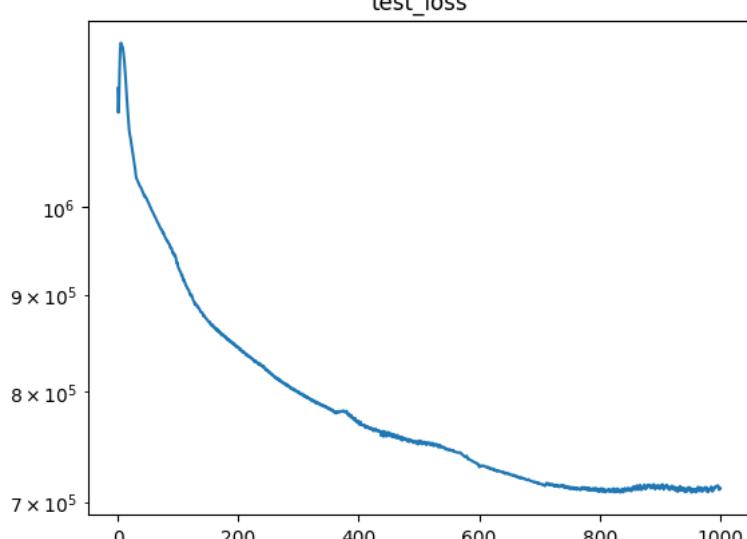
test_loss



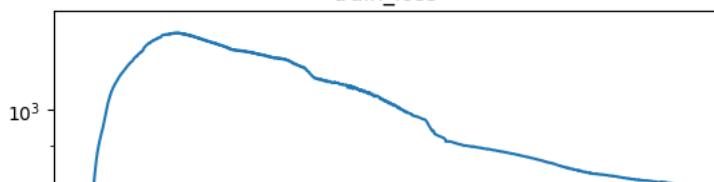
train_loss

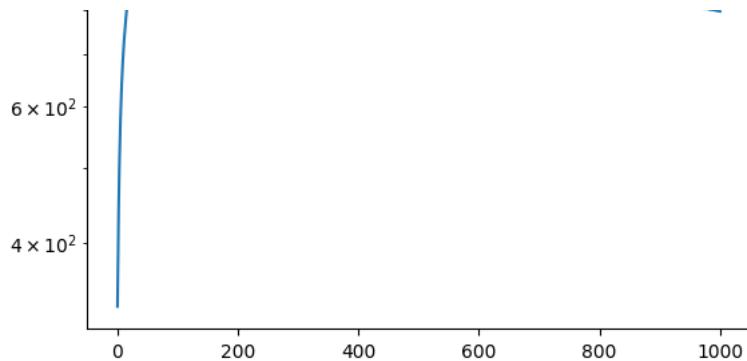


test_loss

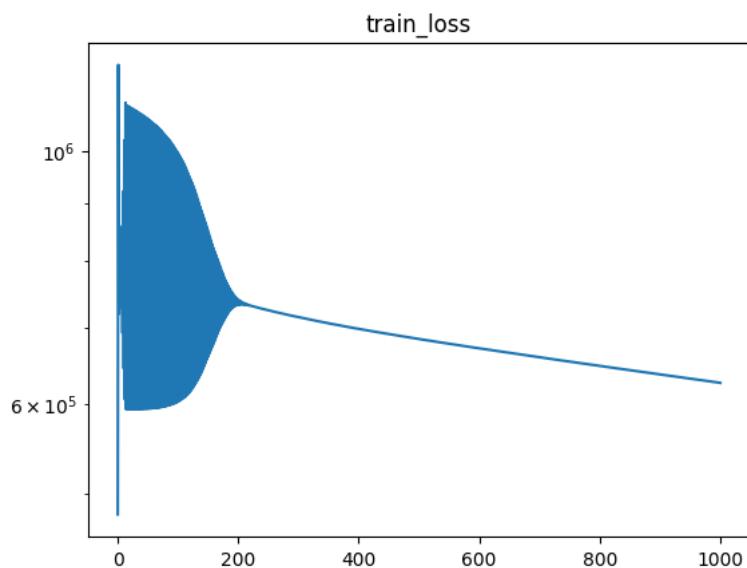
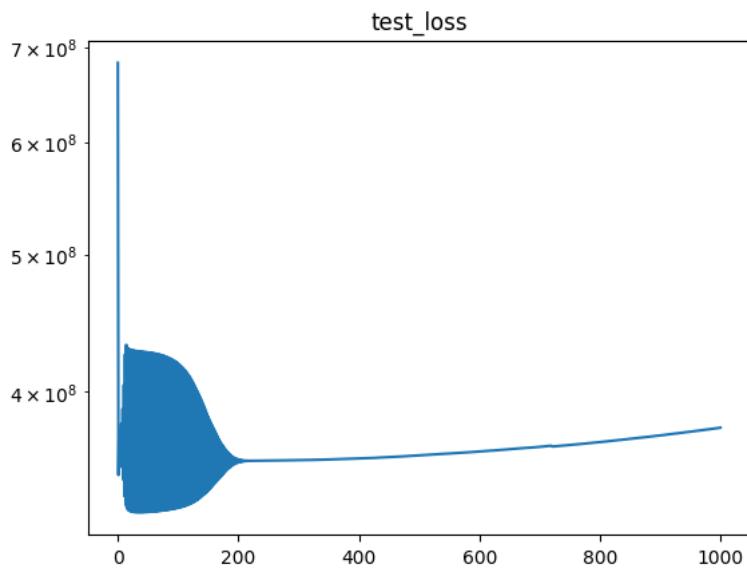


train_loss

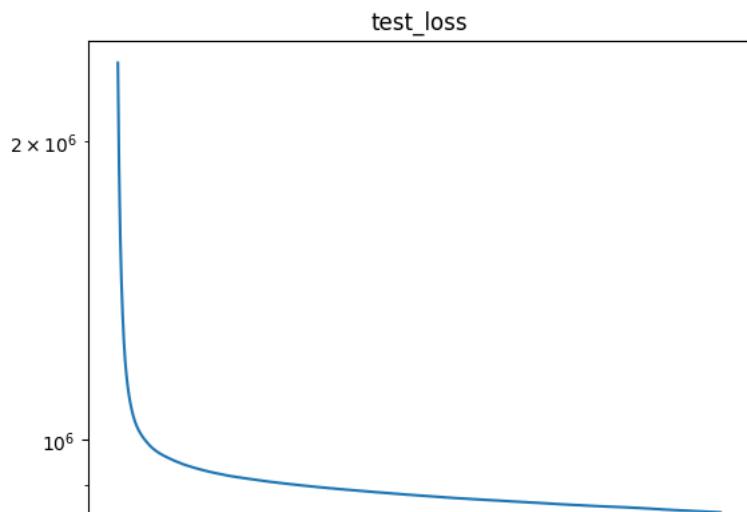


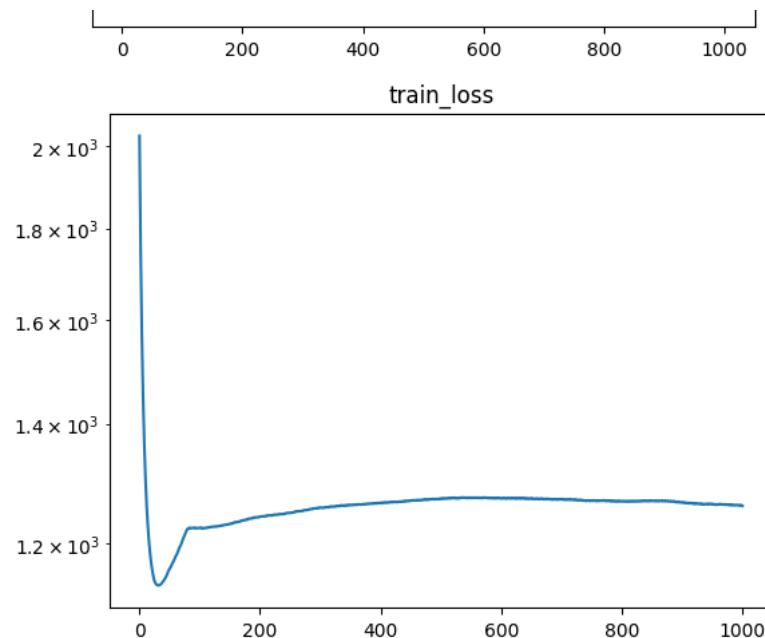


optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 0.01 , minimum_RMSE: 18135.77 , epoch: 1000 , test_loss: 19426 , train_loss:

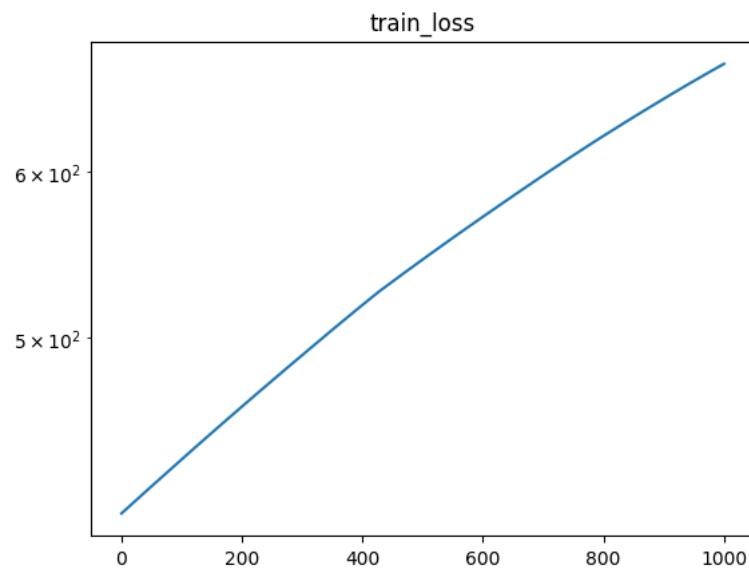
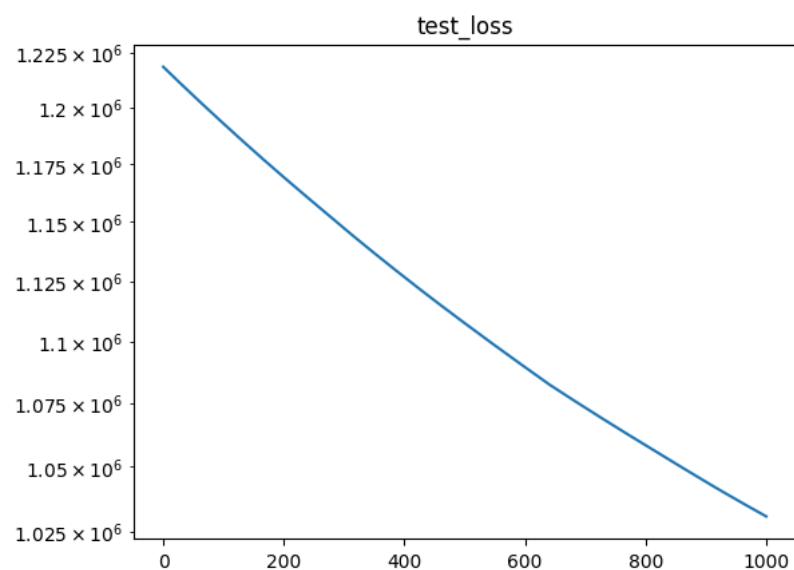


optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 0.05 , minimum_RMSE: 919.39 , epoch: 1000 , test_loss: 919 , train_loss:

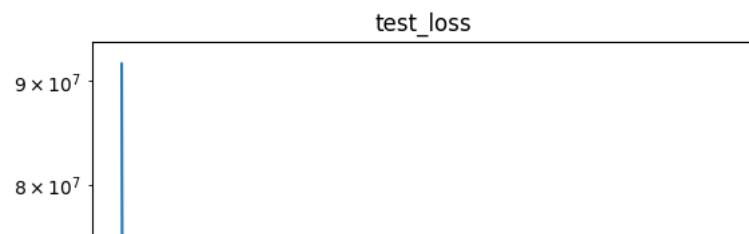


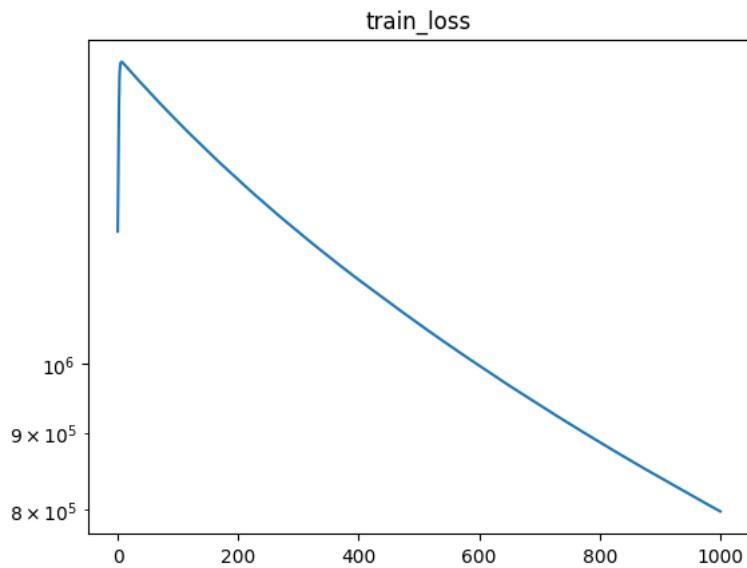
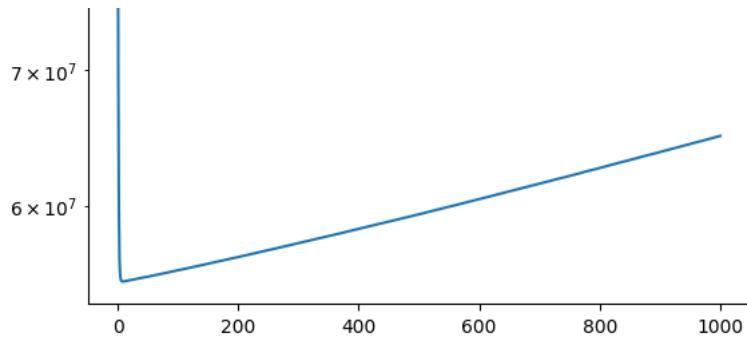


optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 0.001 , minimum_RMSE: 1015.24 , epoch: 1000 , test_loss: 1015 , train_loss:

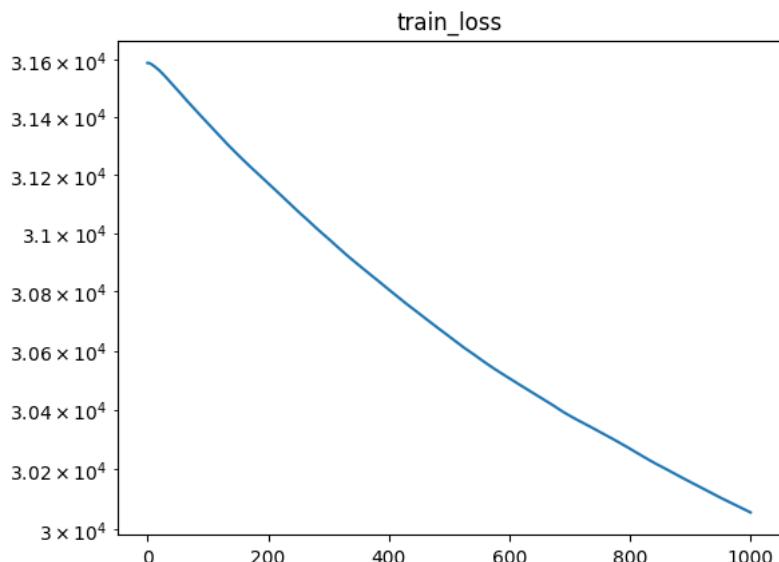
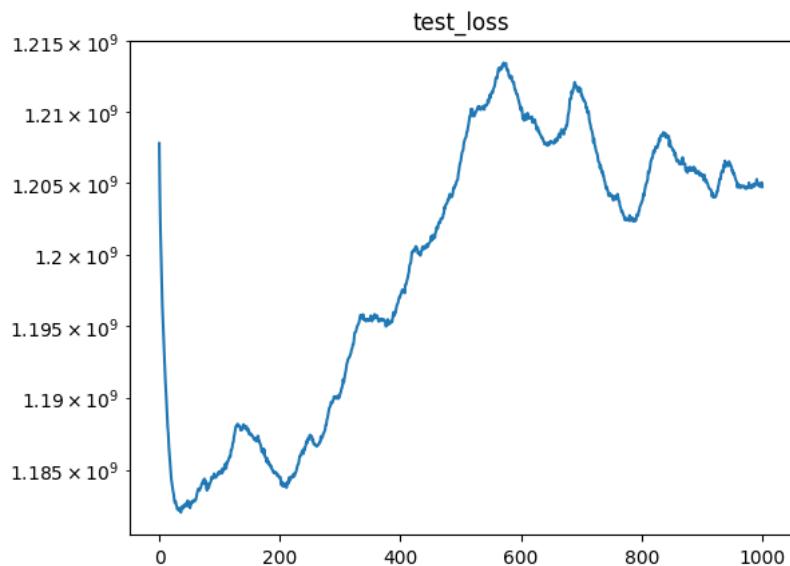


optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 0.005 , minimum_RMSE: 7418.12 , epoch: 1000 , test_loss: 8059 , train_loss:

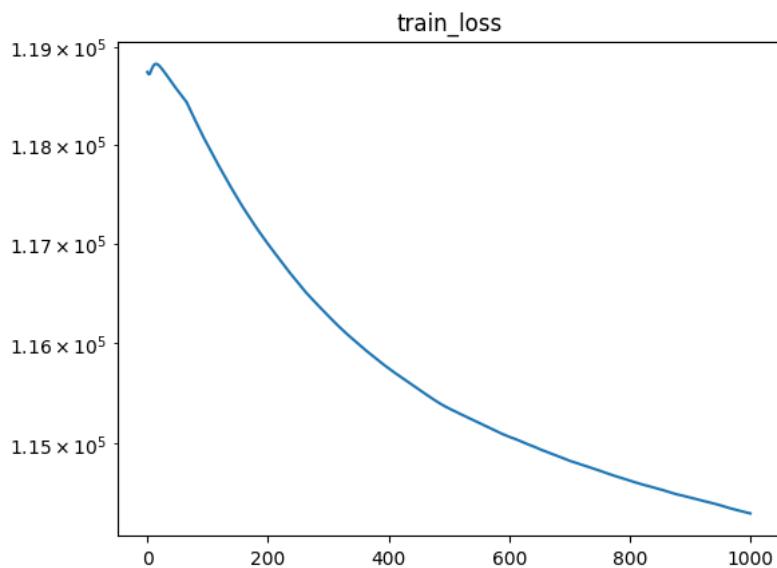
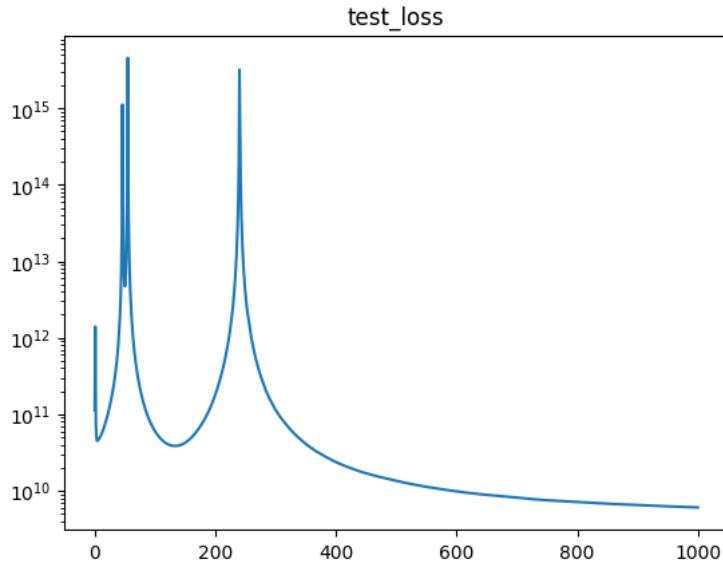




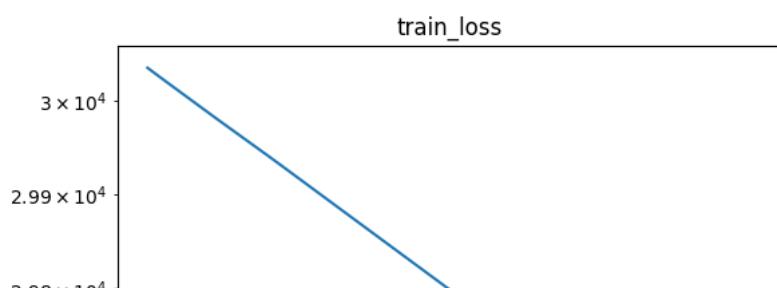
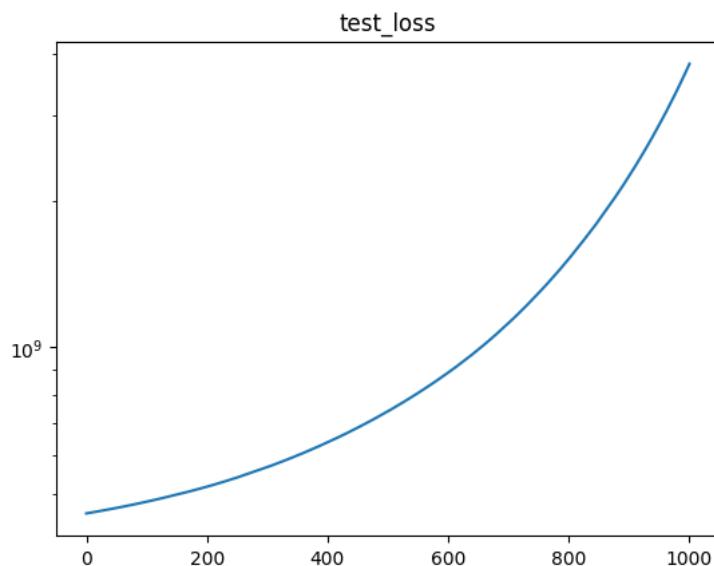
optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 0.0001 , minimum_RMSE: 34381.68 , epoch: 1000 , test_loss: 34709 , train_lo



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

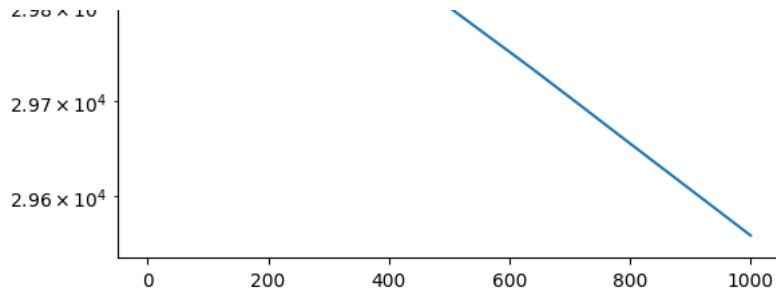


optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-05 , minimum_RMSE: 21361.47 , epoch: 1000 , test_loss: 61851 , train_loss:

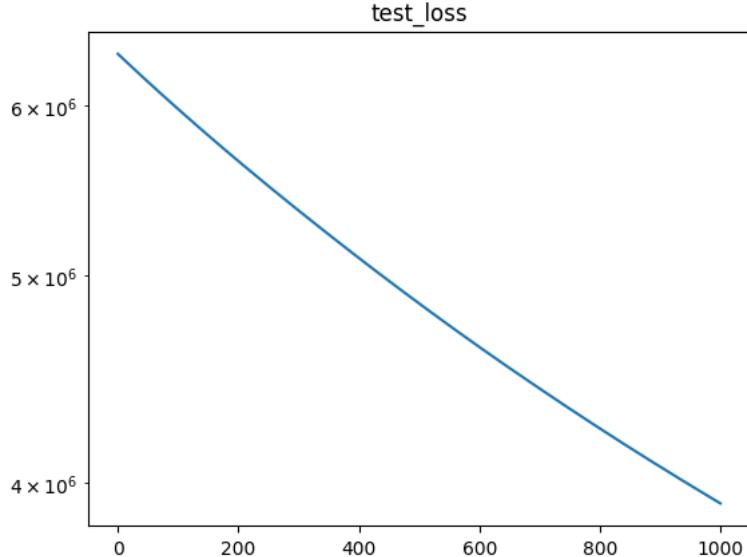


23. 8. 2. 오후 3:59

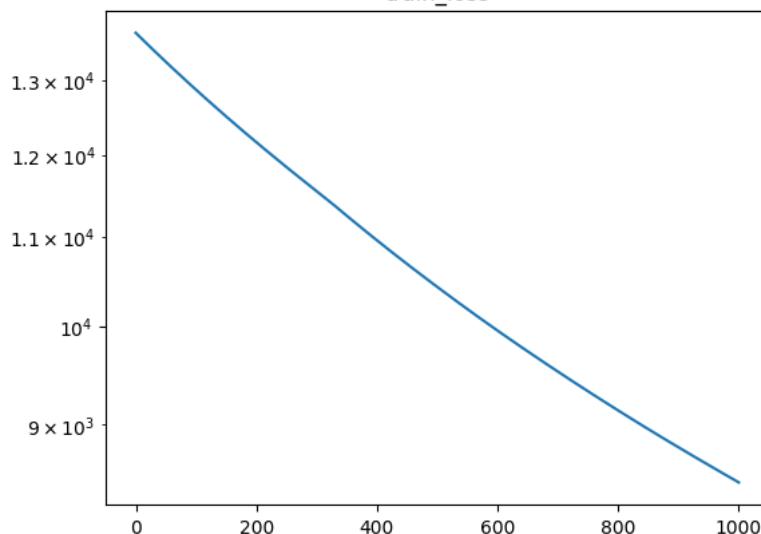
predictingKineticE.ipynb - Colaboratory



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

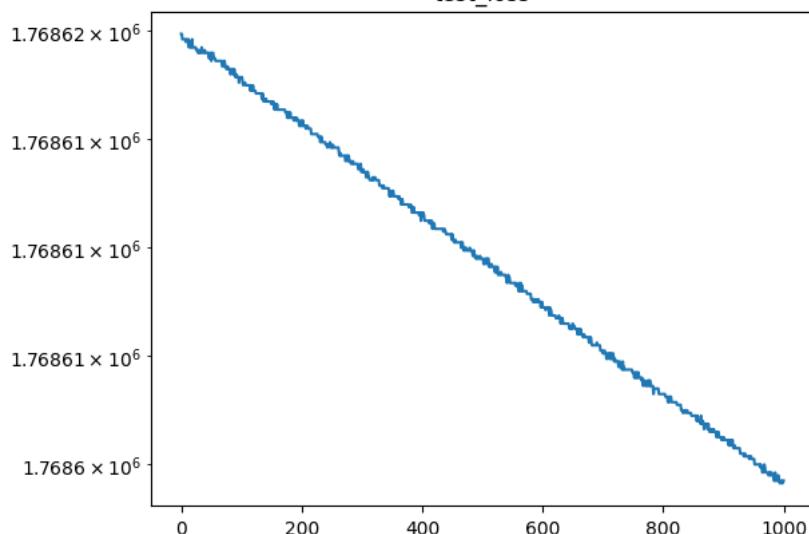


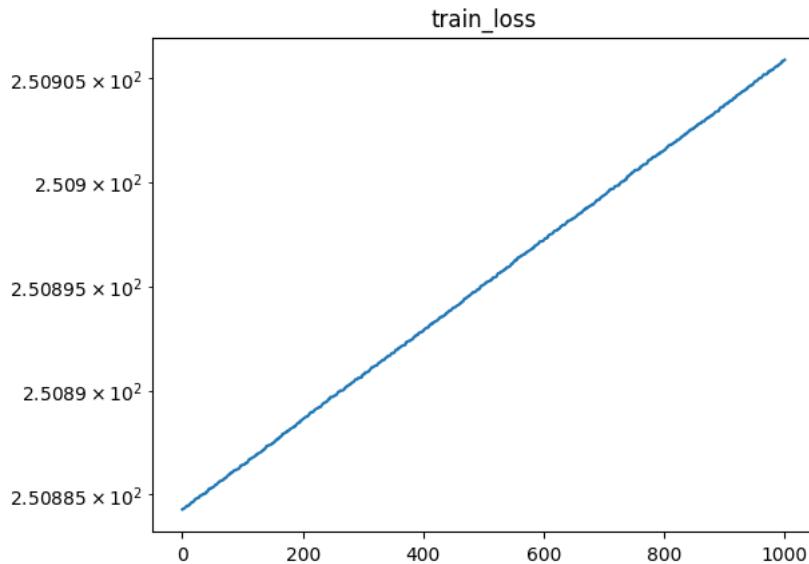
train_loss



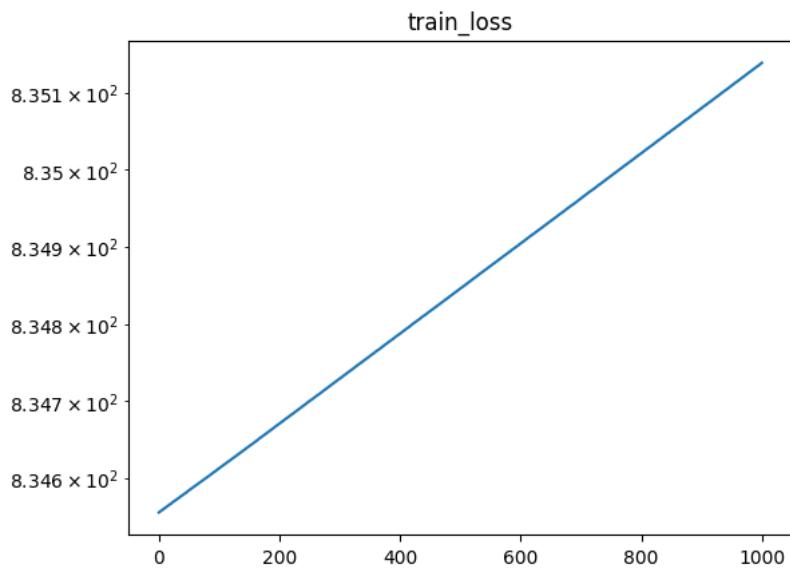
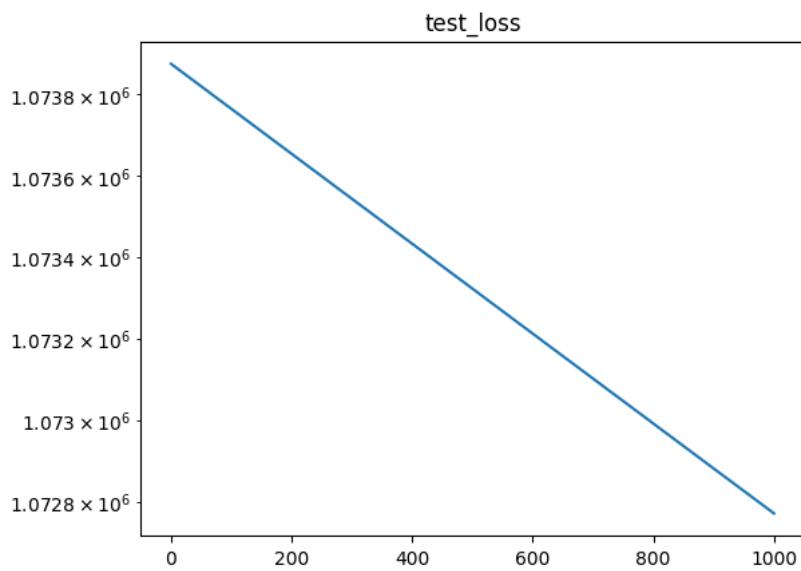
optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-06 , minimum_RMSE: 1329.89 , epoch: 1000 , test_loss: 1329 , train_loss:

test_loss

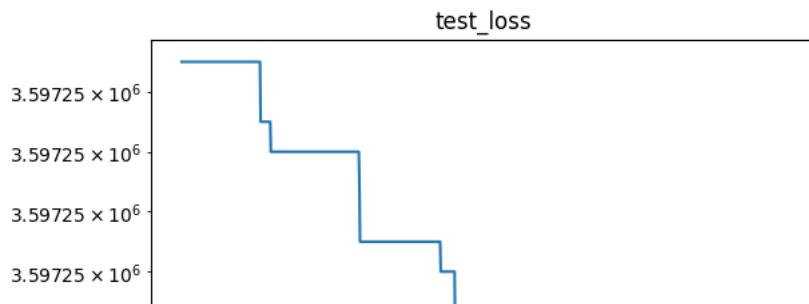


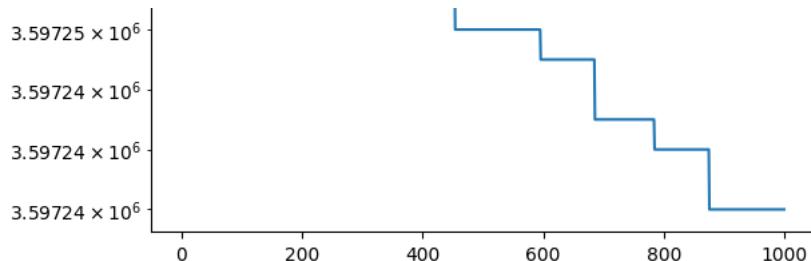


optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 5e-06 , minimum_RMSE: 1035.75 , epoch: 1000 , test_loss: 1035 , train_loss:

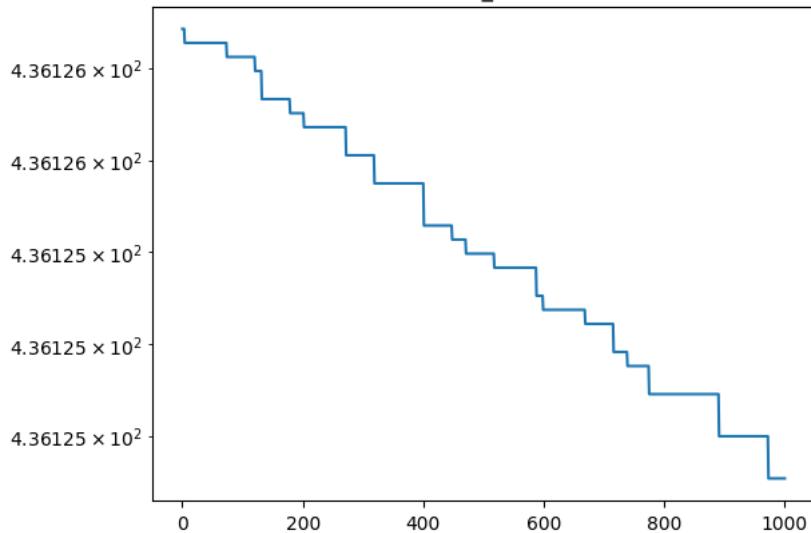


optimizer: Adadelta , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-07 , minimum_RMSE: 1896.64 , epoch: 1000 , test_loss: 1896 , train_loss:

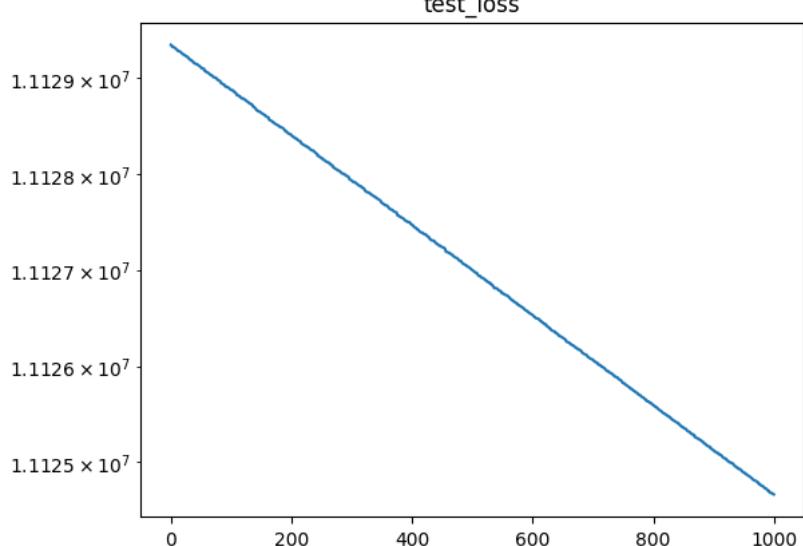




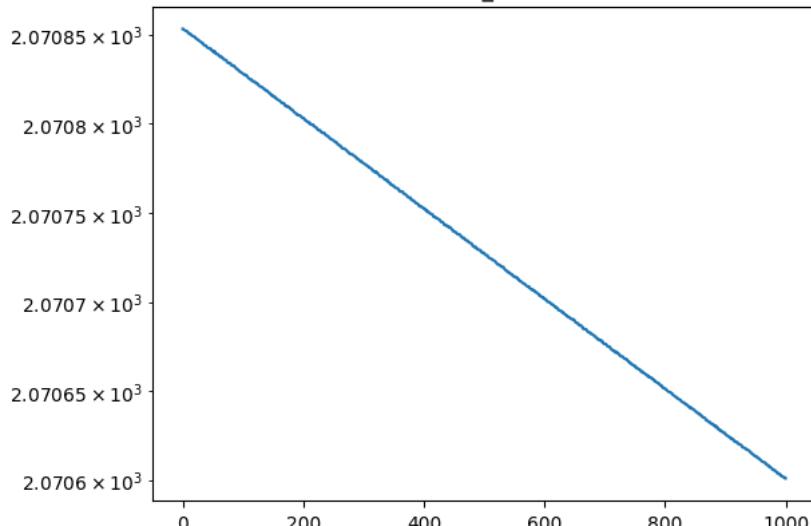
train_loss



test_loss



train_loss

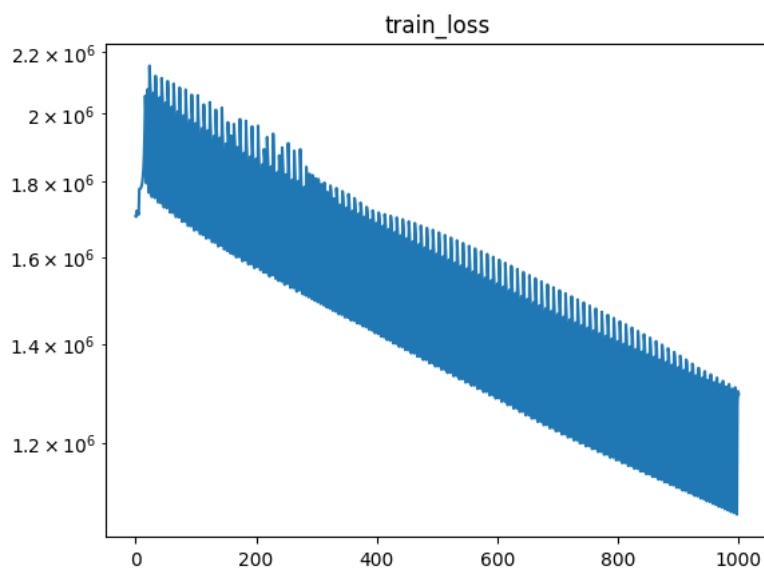
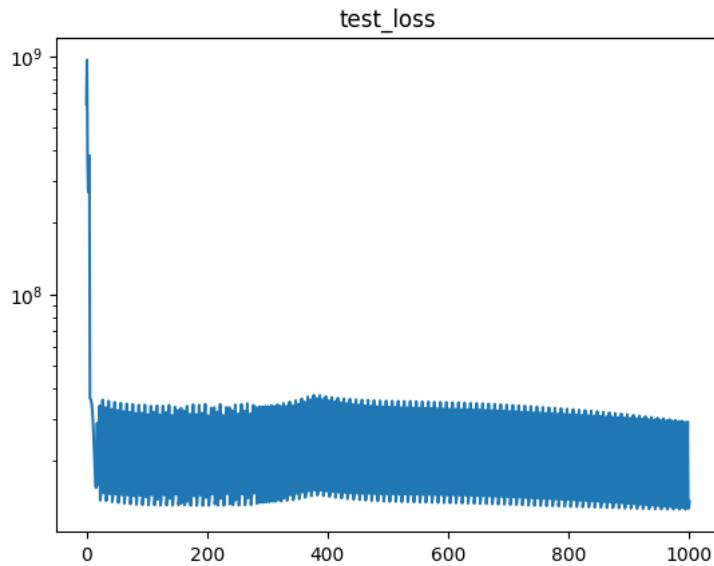


/usr/local/lib/python3.10/dist-packages/torch/nn/modules/loss.py:536: UserWarning: Using a target size (torch.Size([50])) that is different to return F.mse_loss(input, target, reduction=self.reduction)

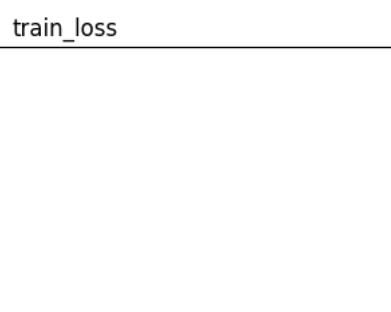
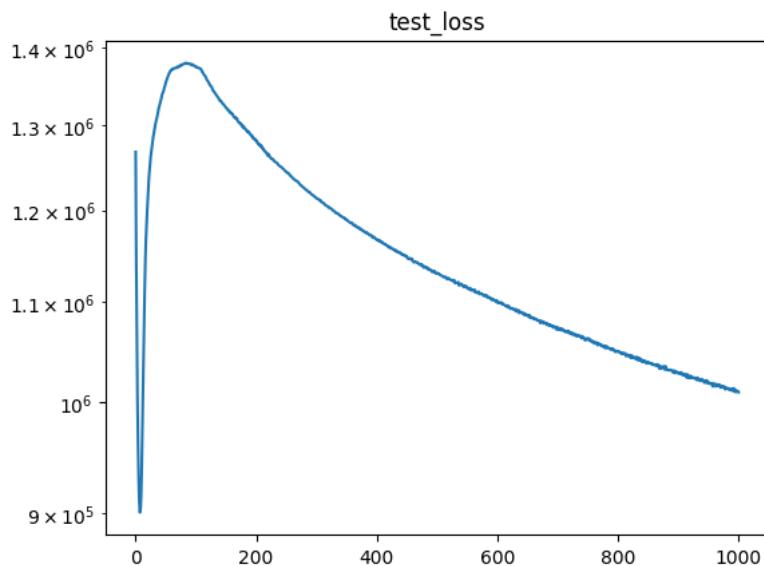
23. 8. 2. 오후 3:59

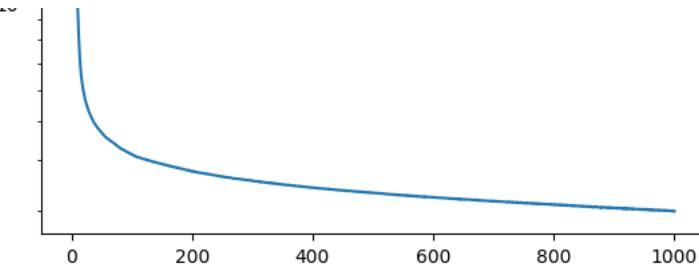
predictingKineticE.ipynb - Colaboratory

optimizer: Adadelta , n_of_data: 1000 , bsize: 50 , learningRate: 0.1 , minimum_RMSE: 3529.55 , epoch: 1000 , test_loss: 3003 , train_loss:



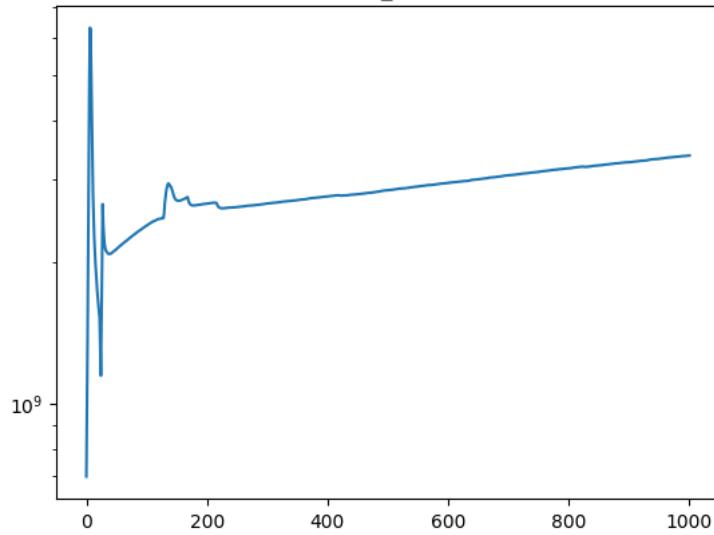
optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 0.5 , minimum_RMSE: 949.11 , epoch: 1000 , test_loss: 1004 , train_loss:



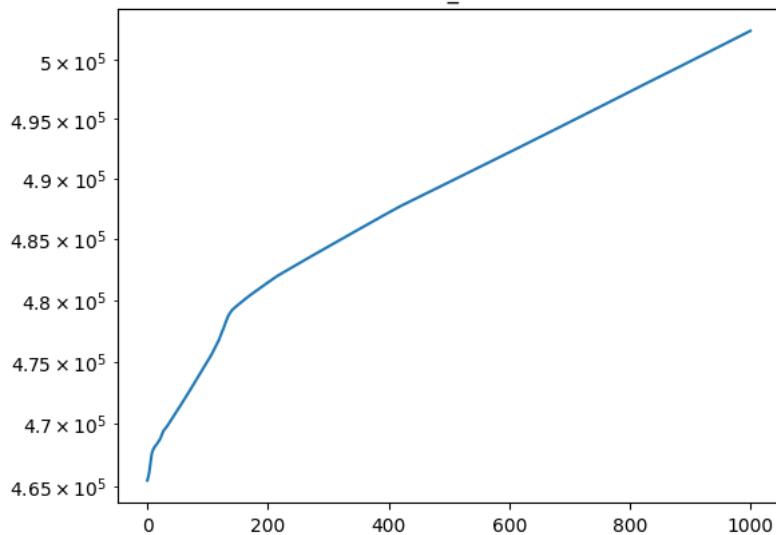


optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 0.01 , minimum_RMSE: 26443.31 , epoch: 1000 , test_loss: 58099 , train_loss:

test_loss

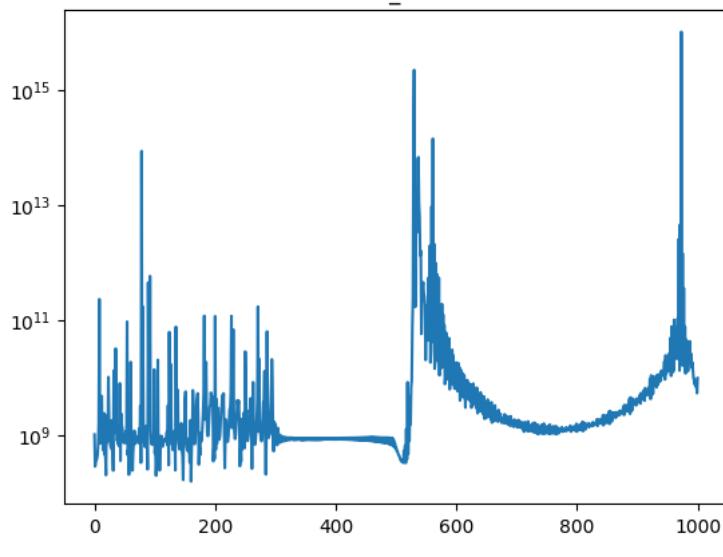


train_loss

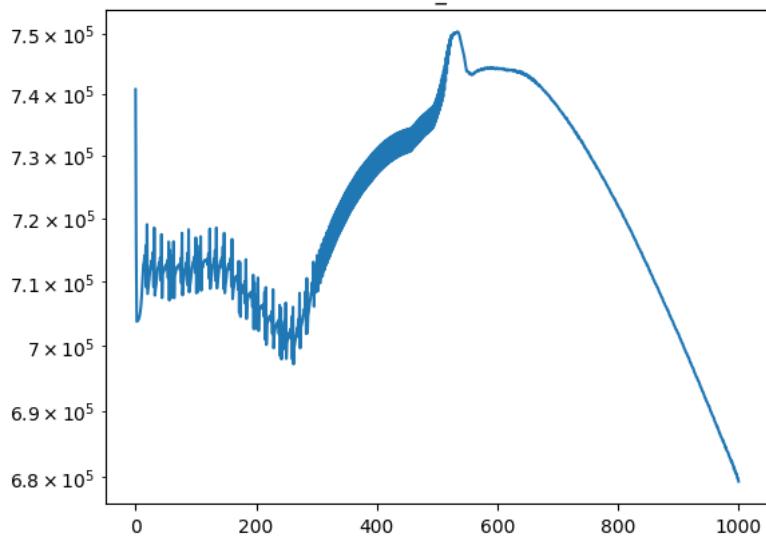


optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 0.05 , minimum_RMSE: 12397.49 , epoch: 1000 , test_loss: 98089 , train_loss:

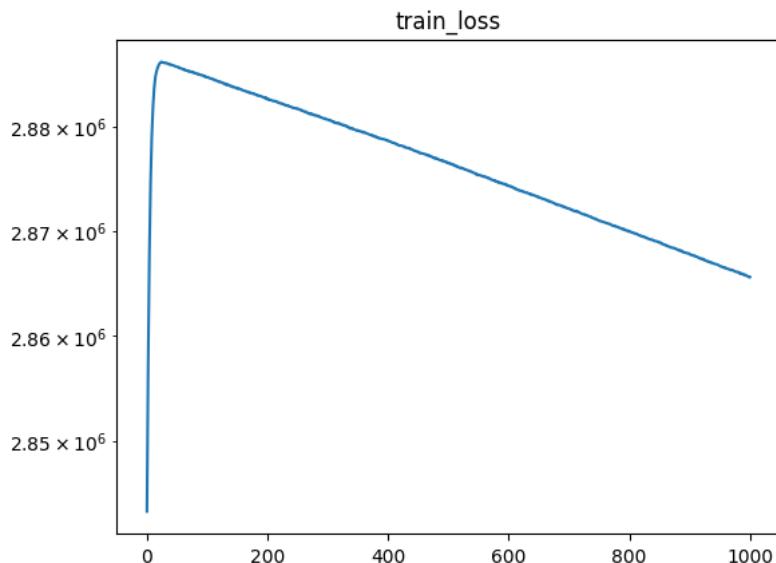
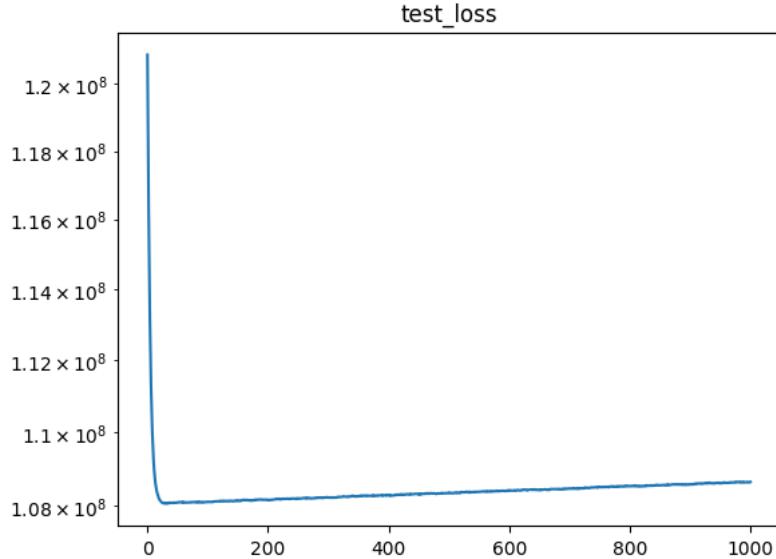
test_loss



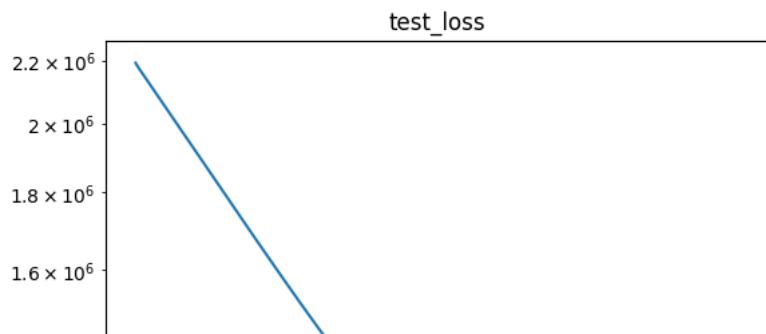
train loss

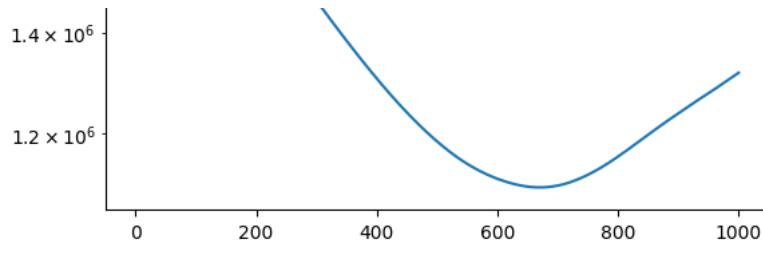


optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 0.001 , minimum_RMSE: 10395.05 , epoch: 1000 , test_loss: 10423 , train_loss:

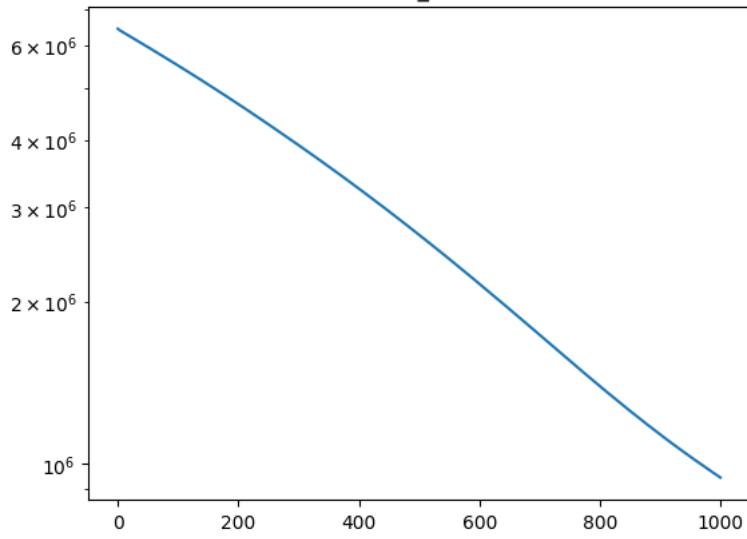


optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 0.005 , minimum_RMSE: 1051.56 , epoch: 1000 , test_loss: 1147 , train_loss:



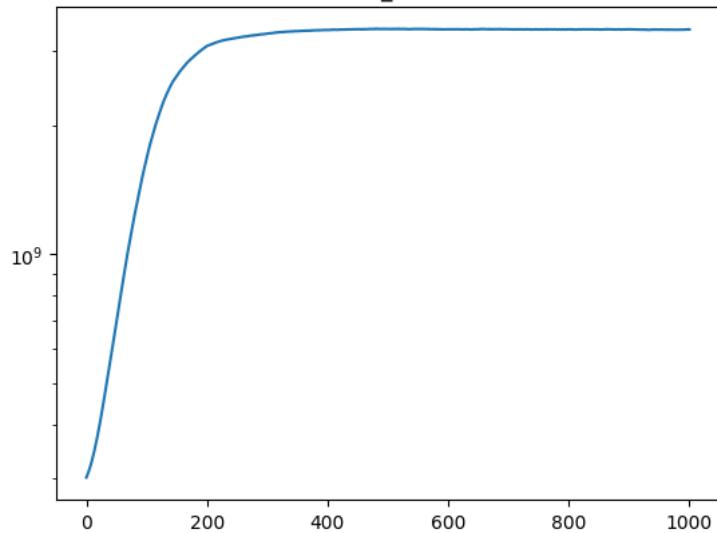


train_loss

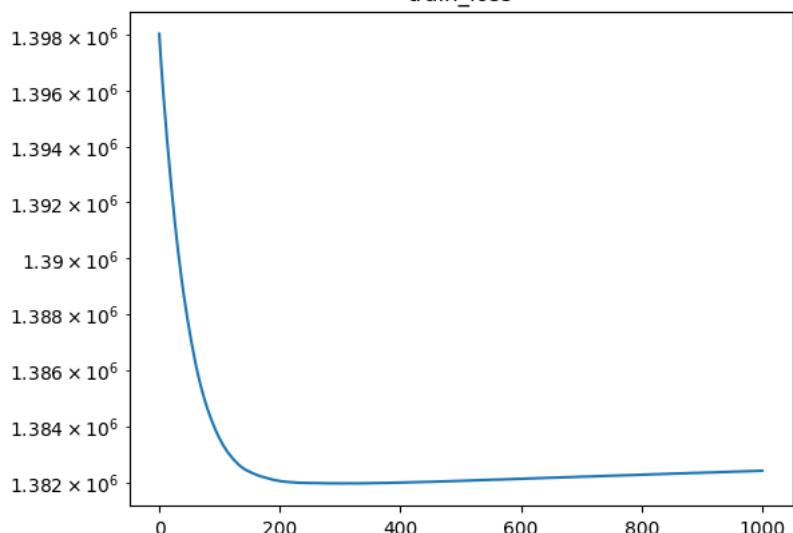


optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 0.0001 , minimum_RMSE: 17334.21 , epoch: 1000 , test_loss: 57912 , train_lo

test_loss

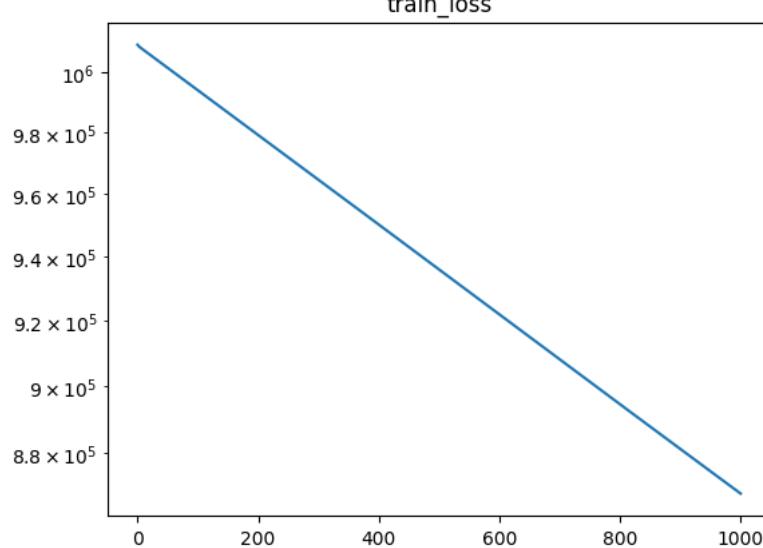
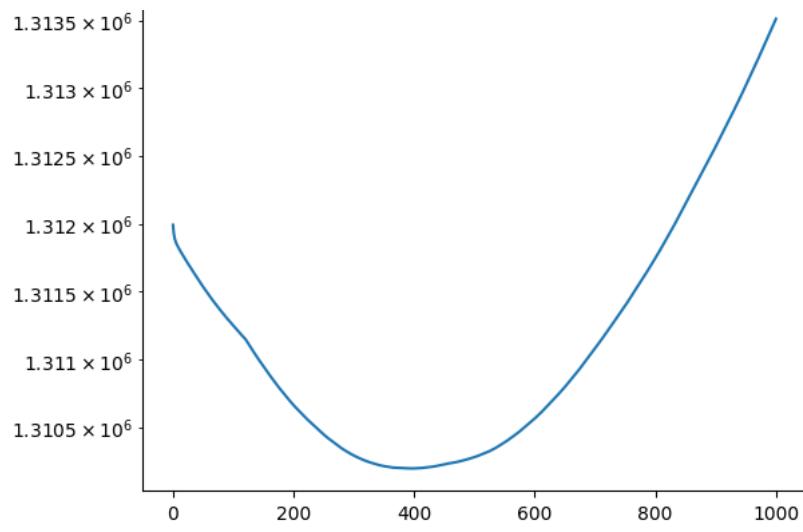


train_loss

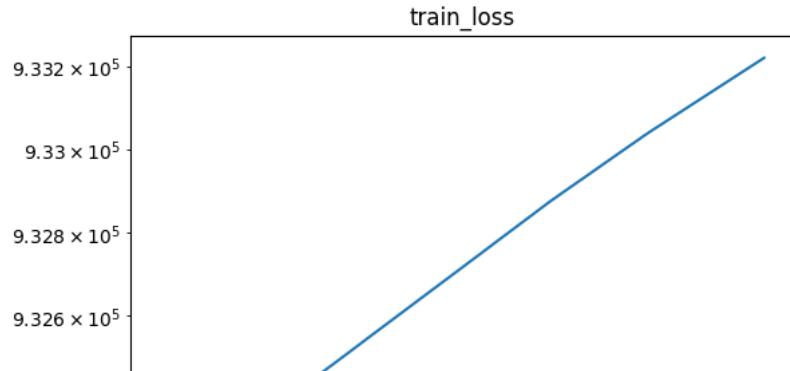
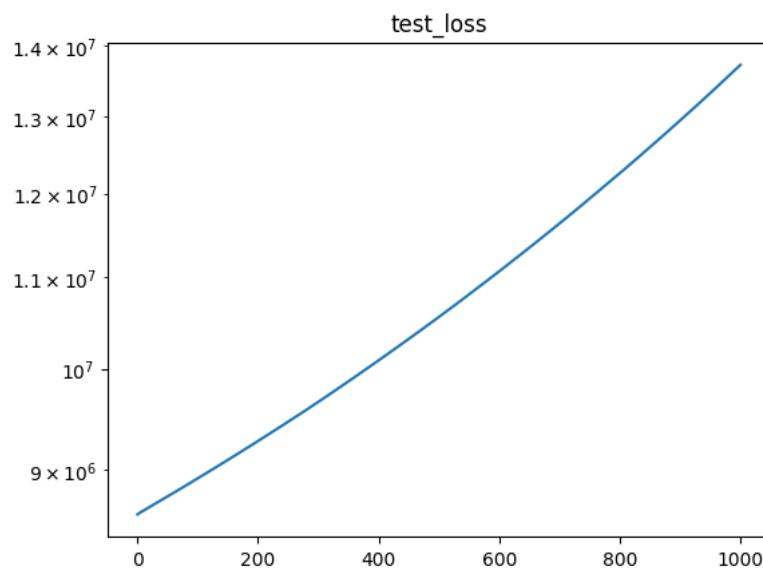


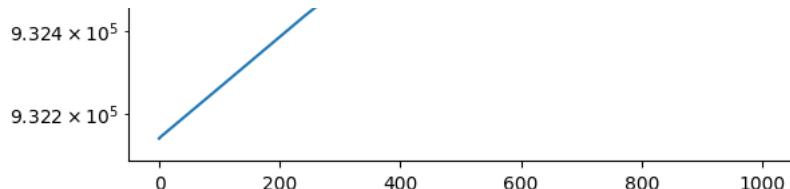
optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 0.0005 , minimum_RMSE: 1144.64 , epoch: 1000 , test_loss: 1146 , train_lo

test_loss

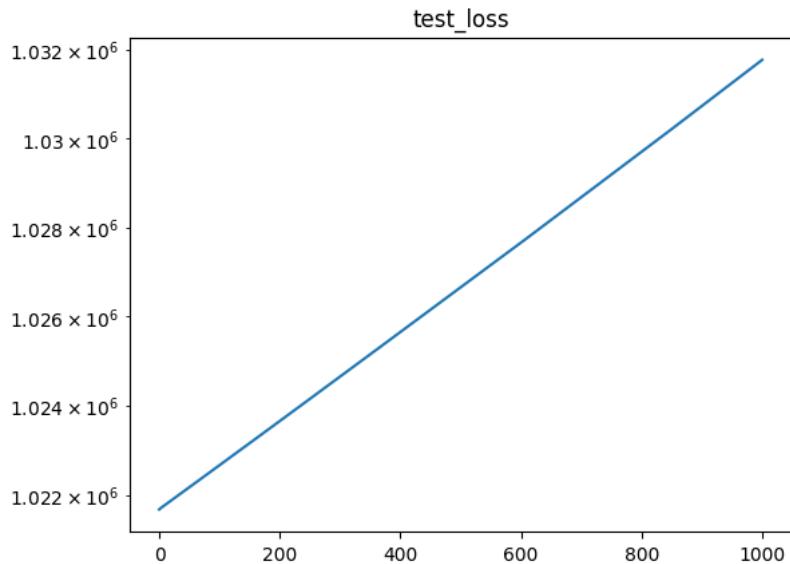


optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-05 , minimum_RMSE: 2931.33 , epoch: 1000 , test_loss: 3704 , train_loss:

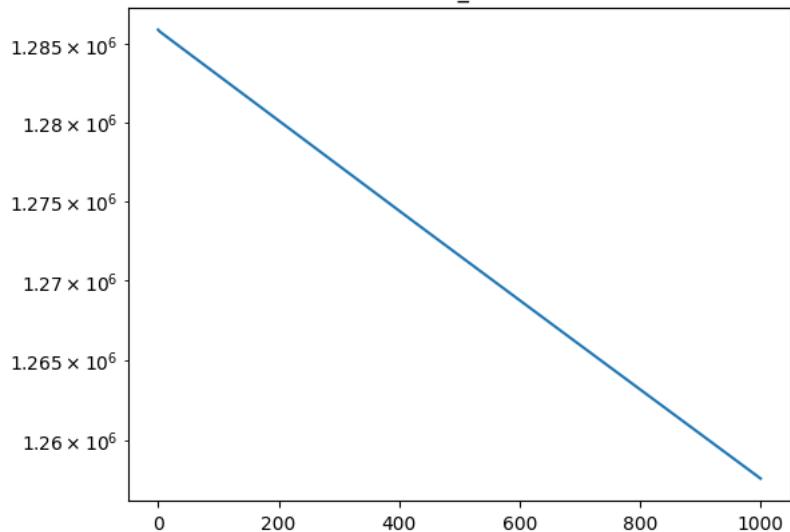




optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 5e-05 , minimum_RMSE: 1010.78 , epoch: 1000 , test_loss: 1015 , train_loss:

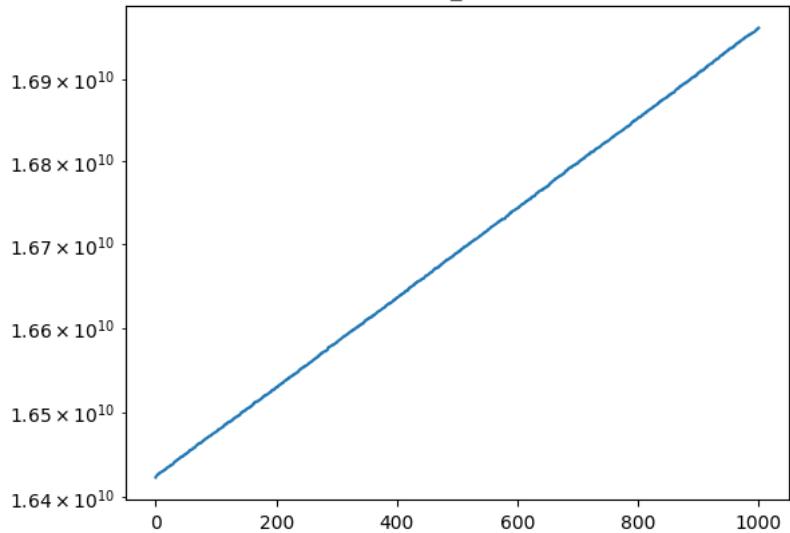


train_loss

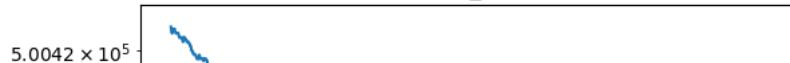


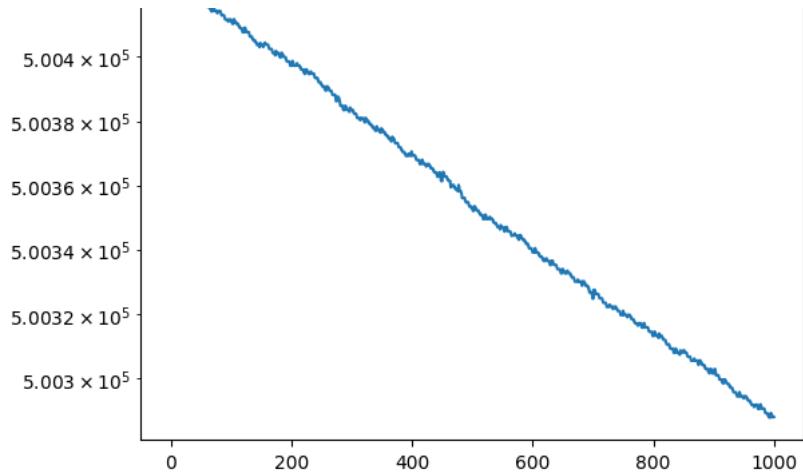
optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-06 , minimum_RMSE: 128152.29 , epoch: 1000 , test_loss: 130238 , train_

test_loss

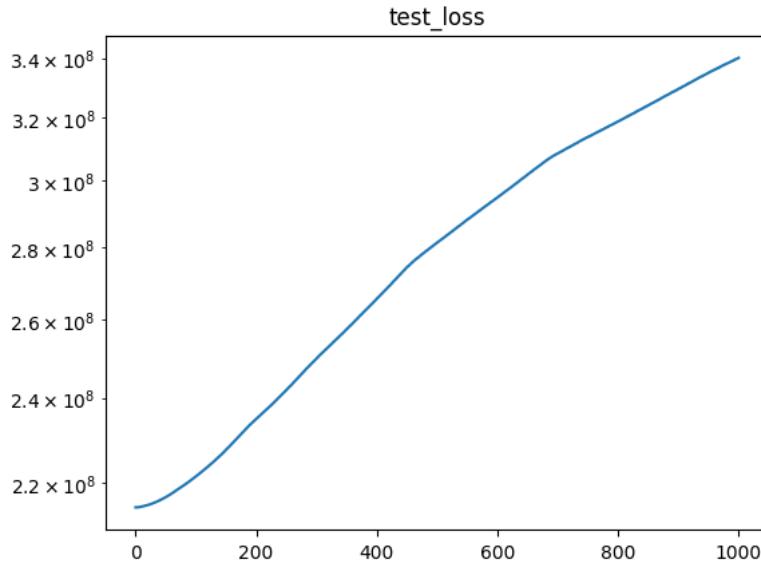


train_loss

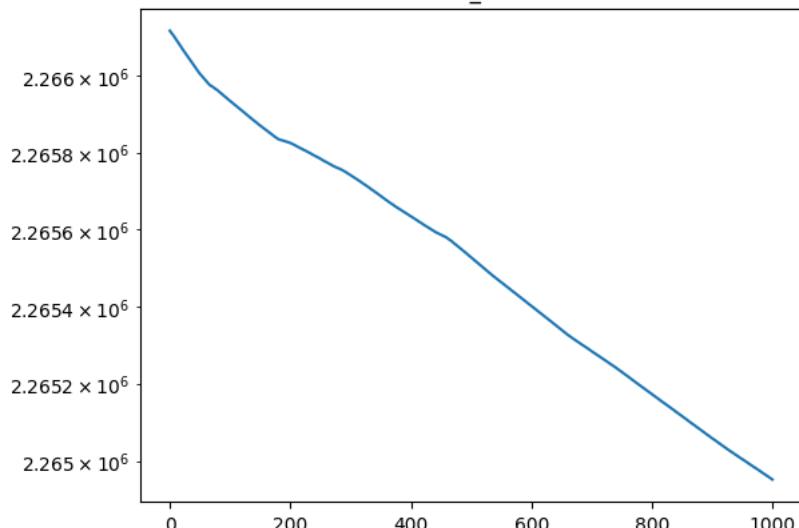




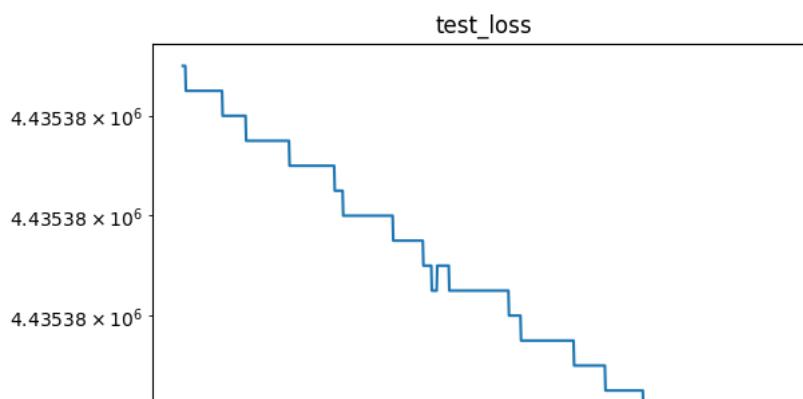
optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 5e-06 , minimum_RMSE: 14647.91 , epoch: 1000 , test_loss: 18445 , train_loss:



train_loss

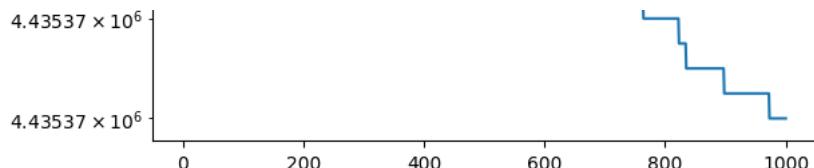


optimizer: Adadelta , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-07 , minimum_RMSE: 2106.03 , epoch: 1000 , test_loss: 2106 , train_loss:

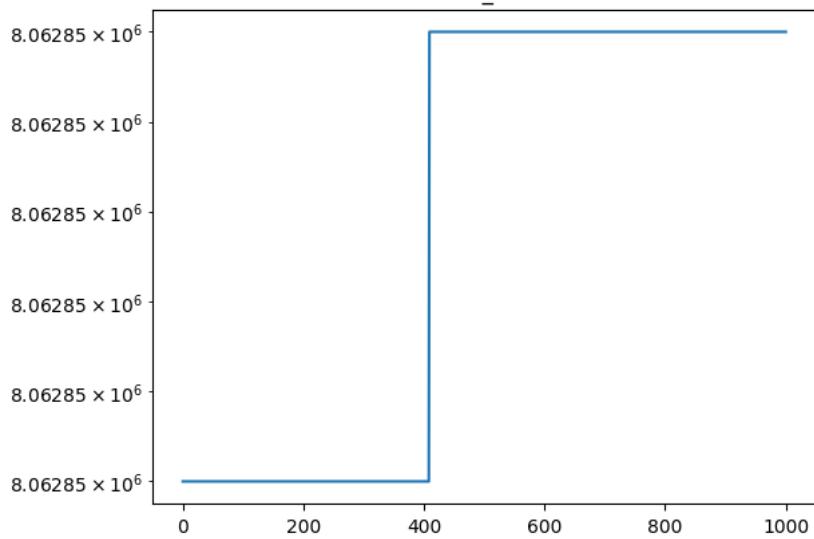


23. 8. 2. 오후 3:59

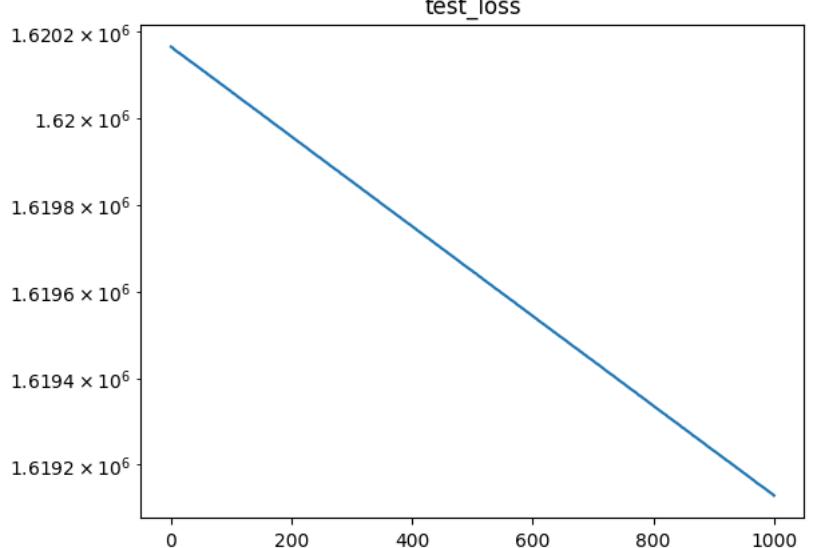
predictingKineticE.ipynb - Colaboratory



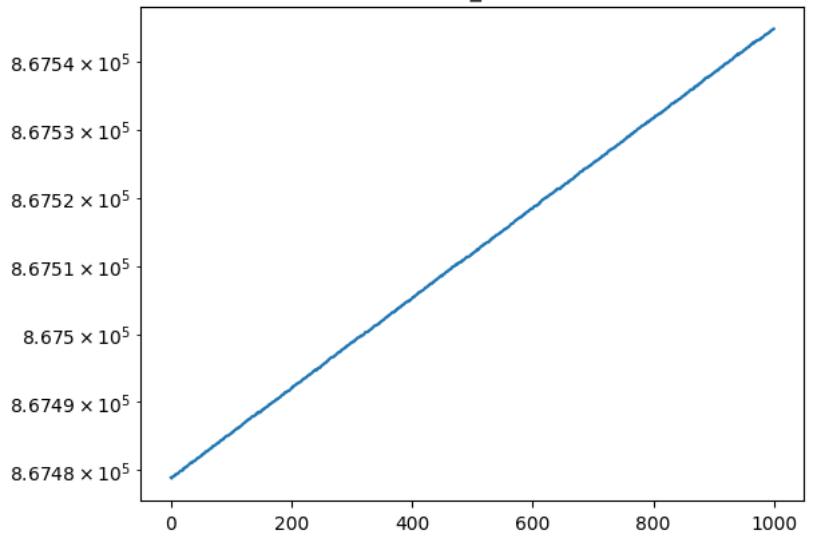
train_loss



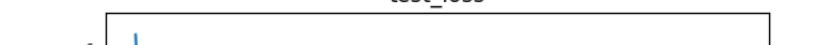
test_loss



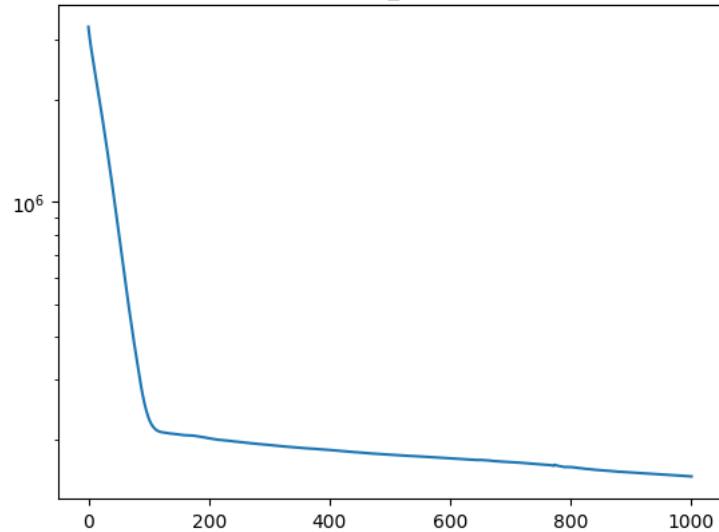
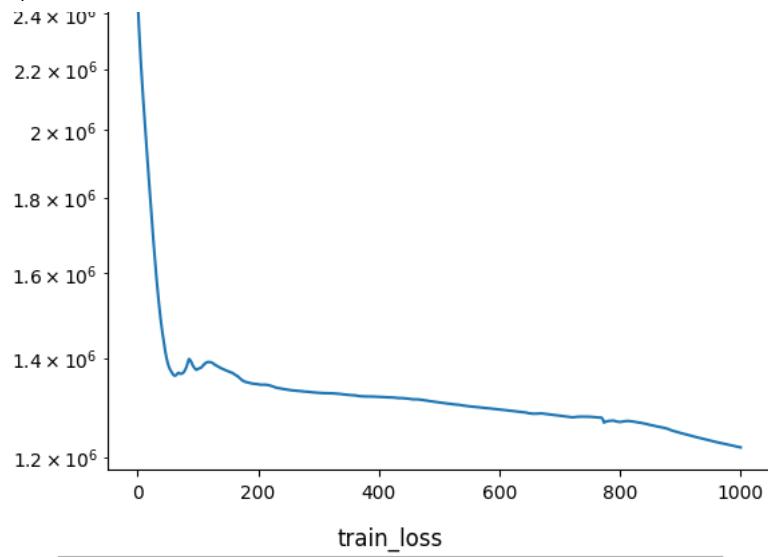
train_loss



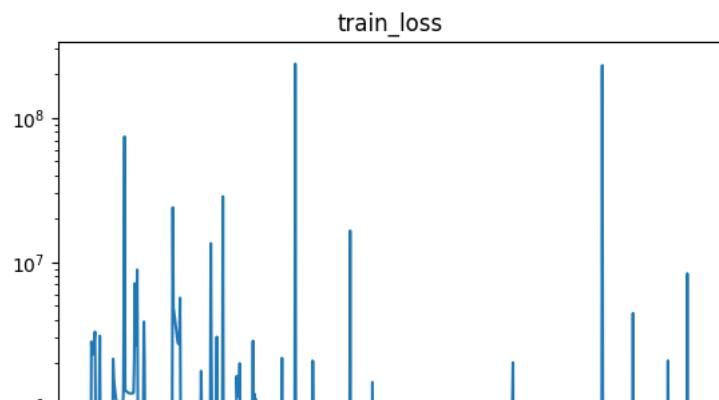
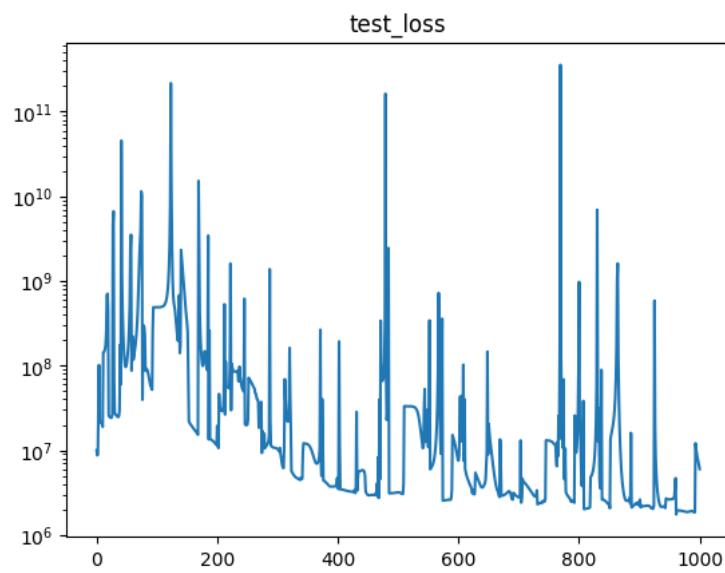
test_loss

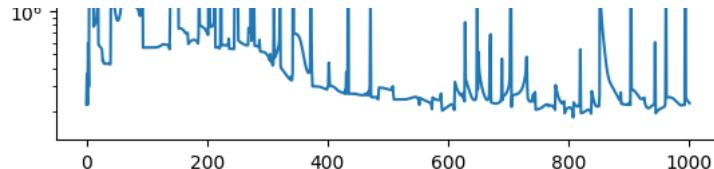


<https://colab.research.google.com/drive/1I3CviS3XInlrYcrL4wCiJy8xqjdbgUaV#scrollTo=KX3k7kqt45E0>



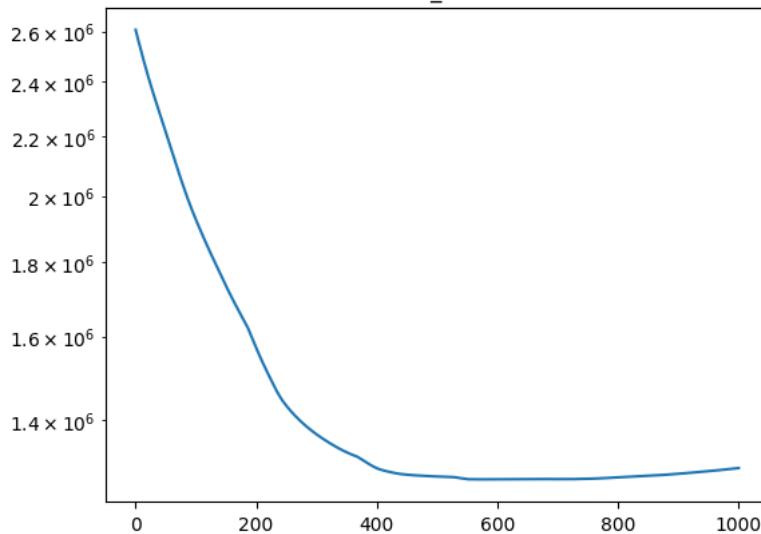
optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 0.5 , minimum_RMSE: 1323.23 , epoch: 1000 , test_loss: 2450 , train_loss:



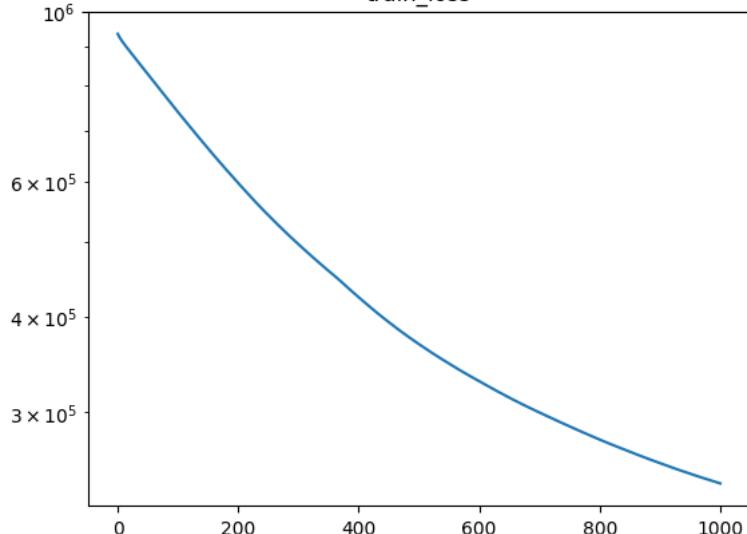


optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 0.01 , minimum_RMSE: 1128.82 , epoch: 1000 , test_loss: 1139 , train_loss:

test_loss

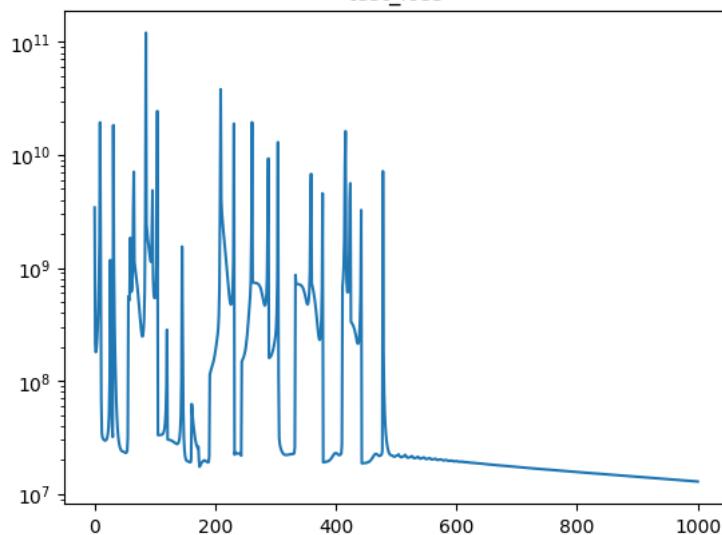


train_loss



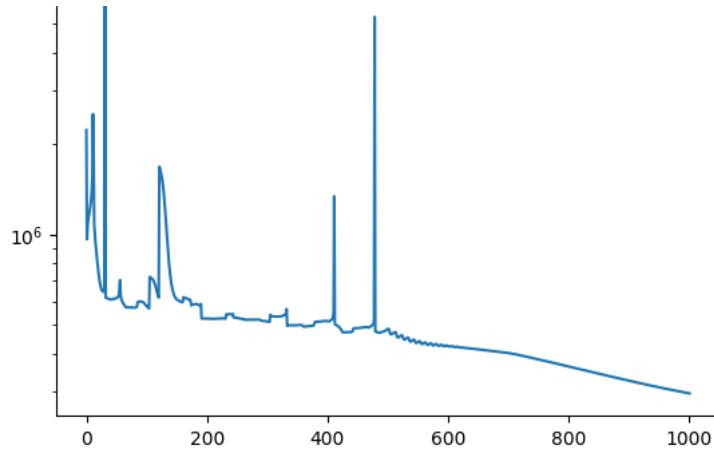
optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 0.05 , minimum_RMSE: 3591.47 , epoch: 1000 , test_loss: 3591 , train_loss:

test_loss

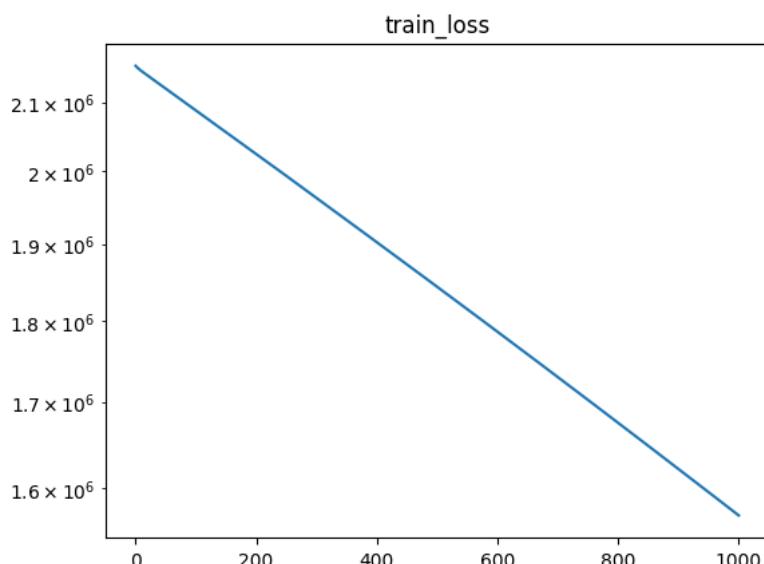
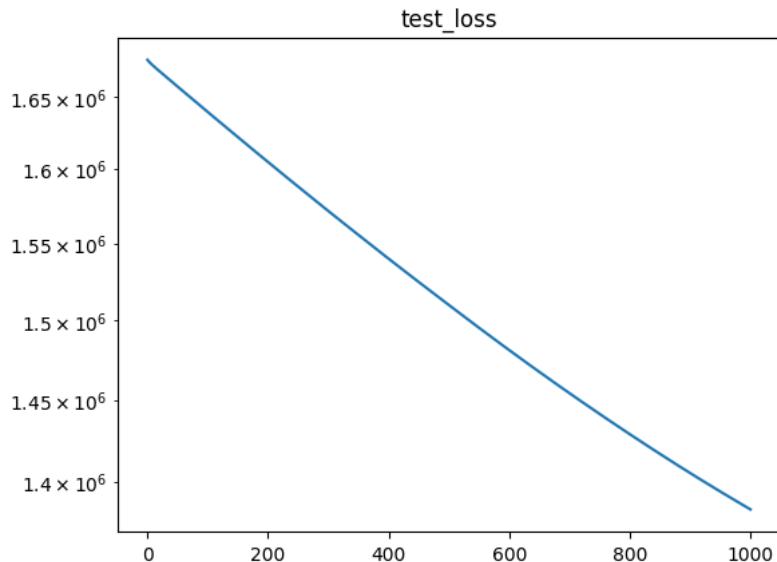


train_loss

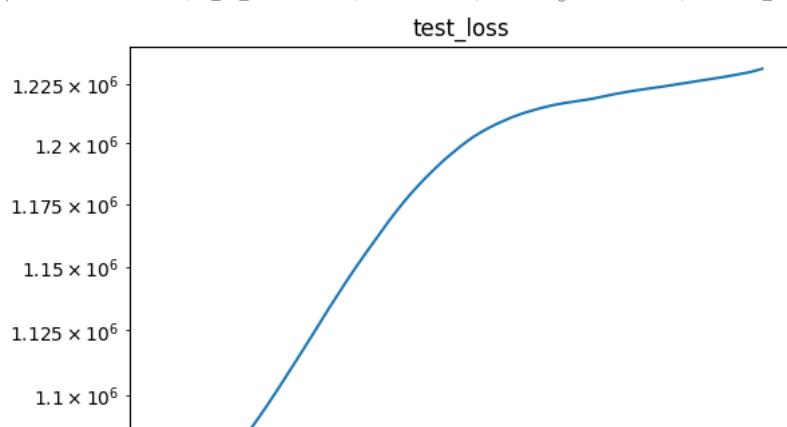


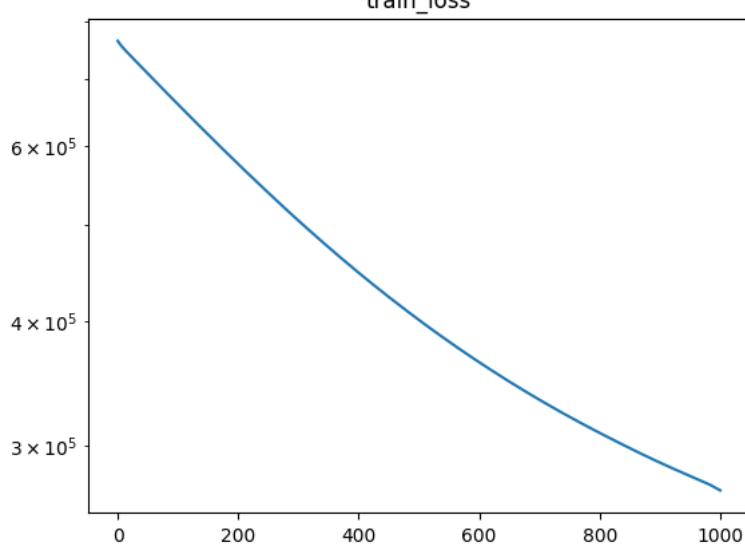
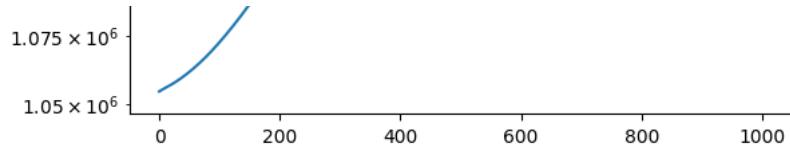


optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 0.001 , minimum_RMSE: 1176.34 , epoch: 1000 , test_loss: 1176 , train_loss:

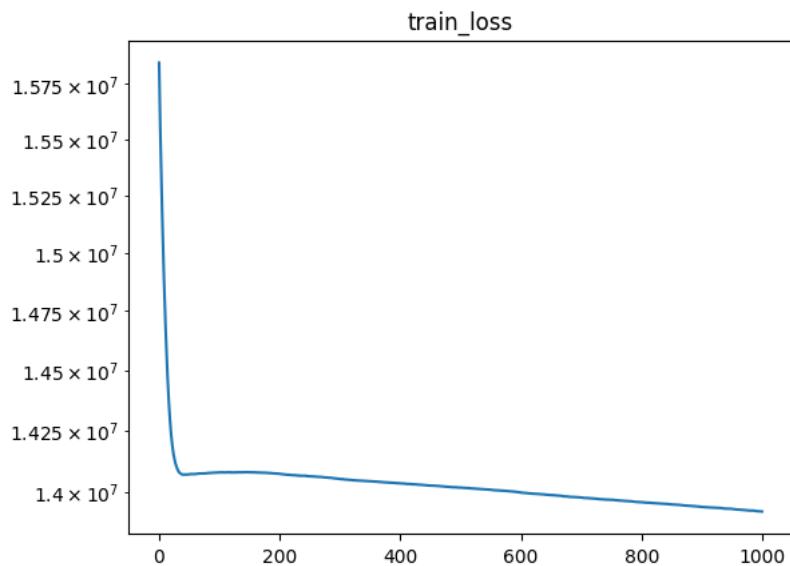
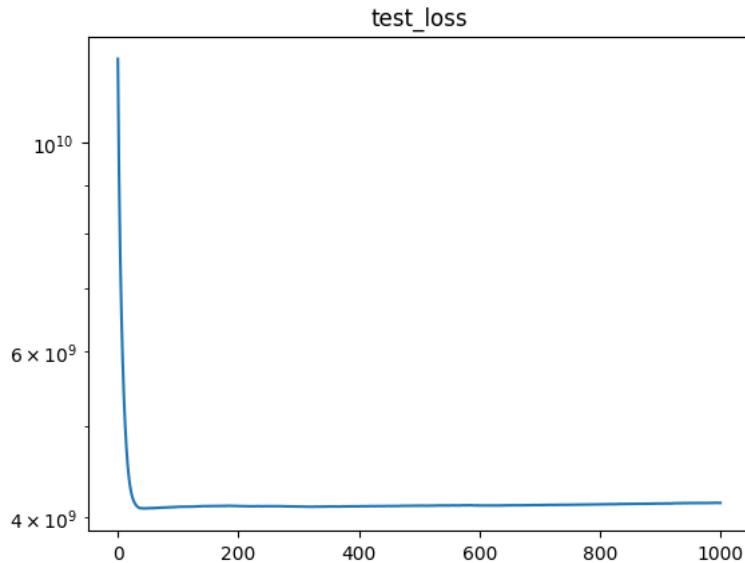


optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 0.005 , minimum_RMSE: 1026.84 , epoch: 1000 , test_loss: 1109 , train_loss:

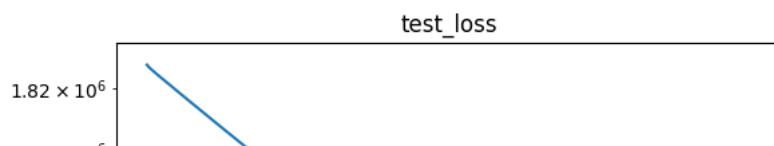


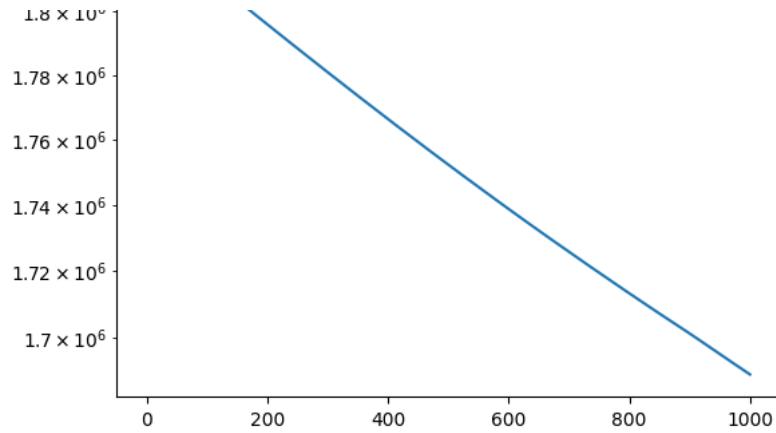


optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 0.0001 , minimum_RMSE: 63942.14 , epoch: 1000 , test_loss: 64376 , train_

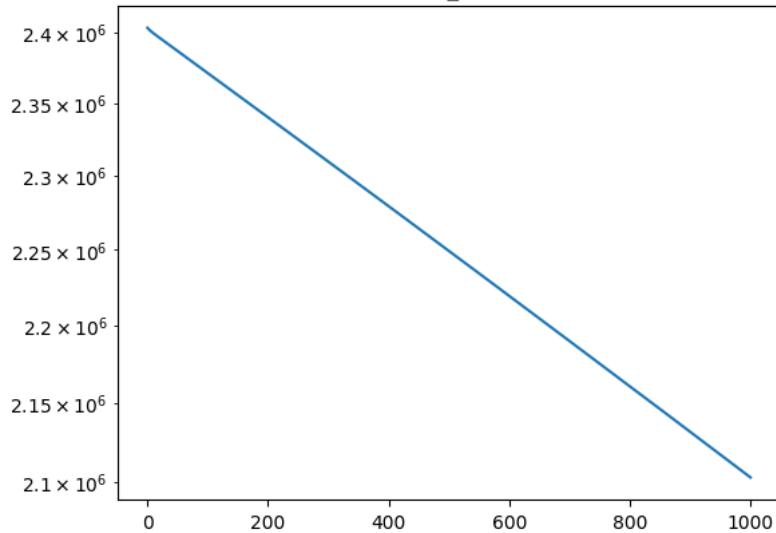


optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 0.0005 , minimum_RMSE: 1299.61 , epoch: 1000 , test_loss: 1299 , train_



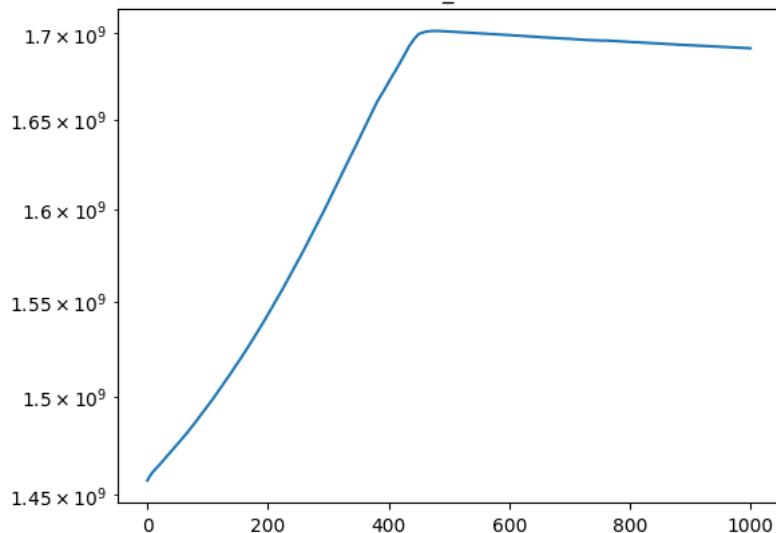


train_loss

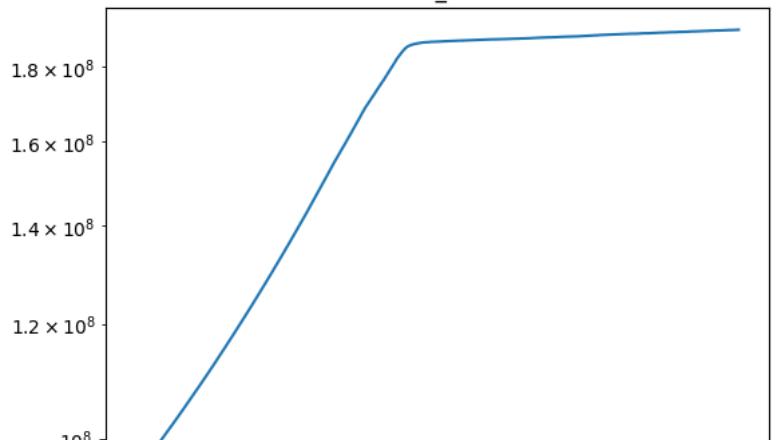


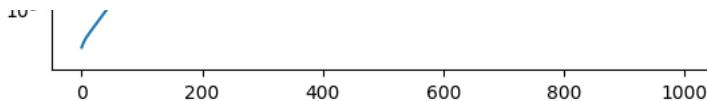
optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-05 , minimum_RMSE: 38174.19 , epoch: 1000 , test_loss: 41124 , train_lo

test_loss

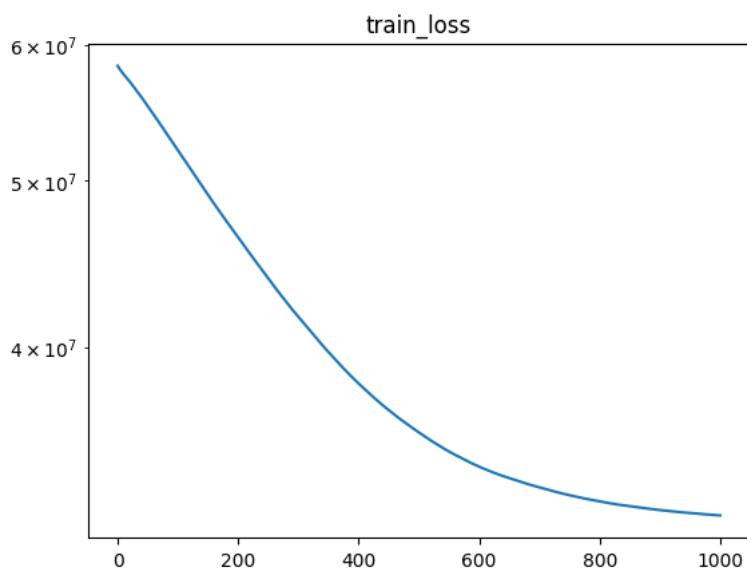
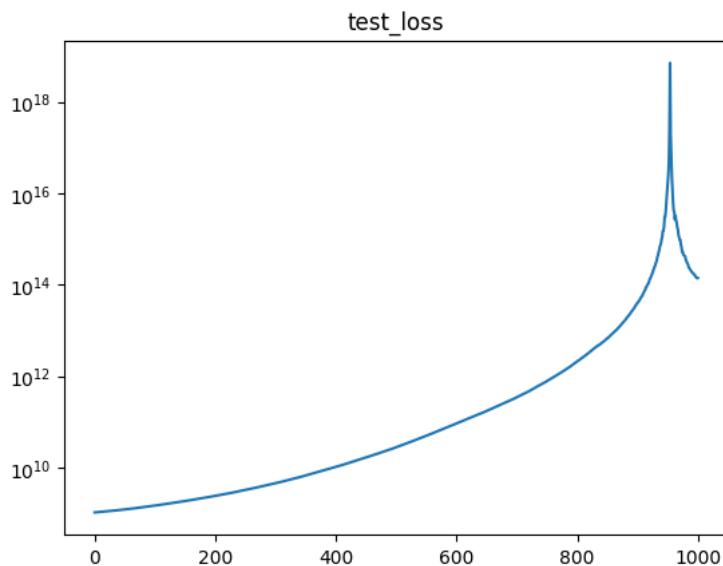


train_loss

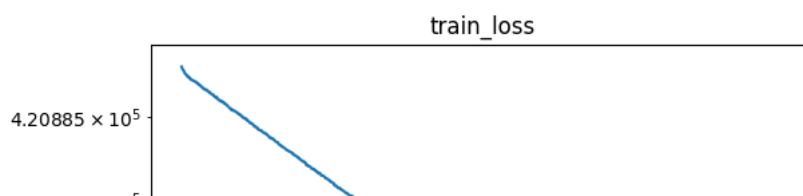
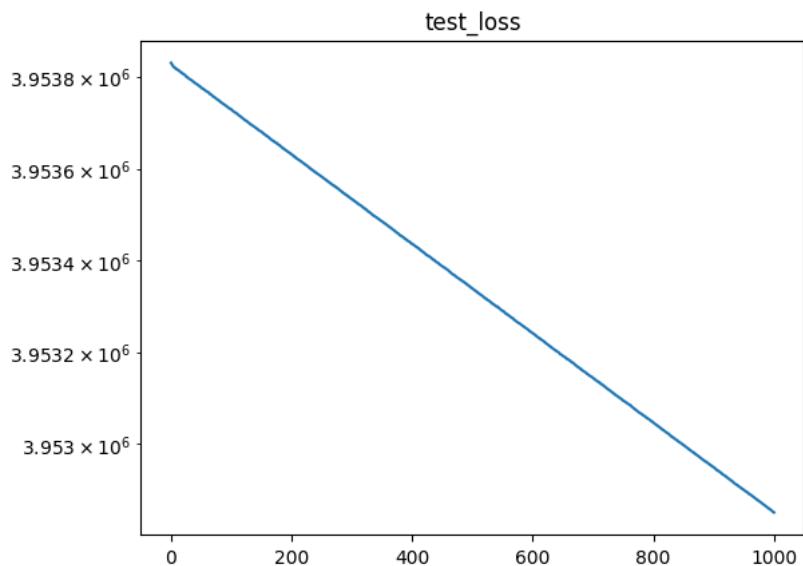


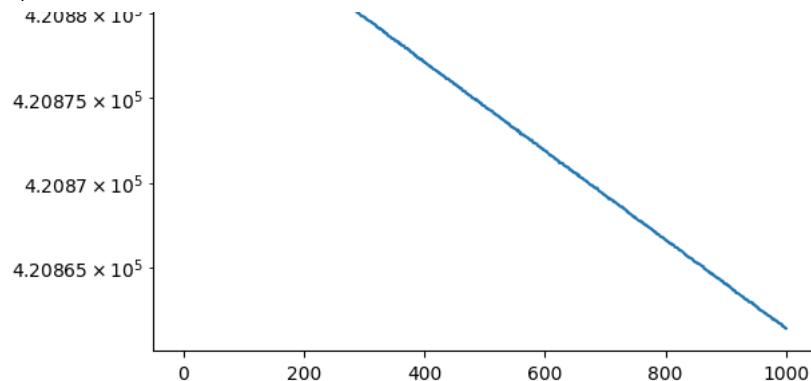


optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-05 , minimum_RMSE: 32569.47 , epoch: 1000 , test_loss: 11910895 , train



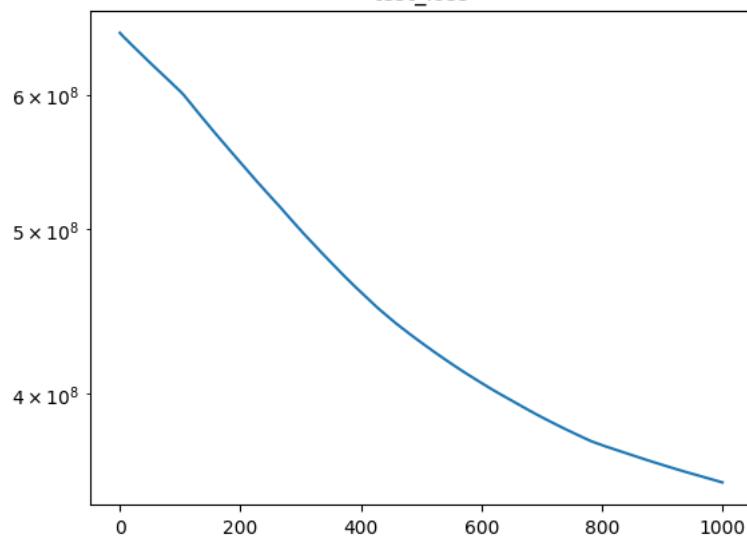
optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-06 , minimum_RMSE: 1988.18 , epoch: 1000 , test_loss: 1988 , train



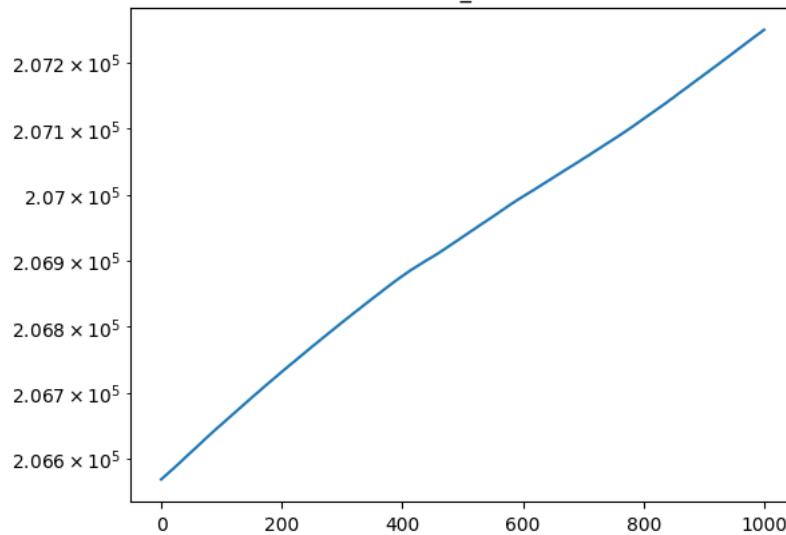


optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-06 , minimum_RMSE: 18833.19 , epoch: 1000 , test_loss: 18833 , train_lo

test_loss

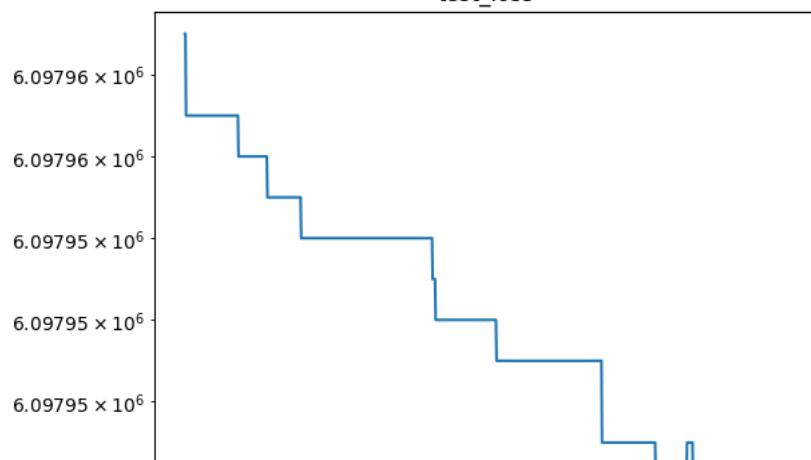


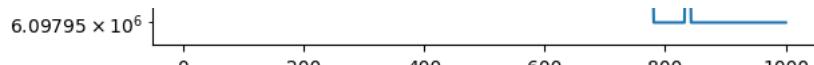
train_loss



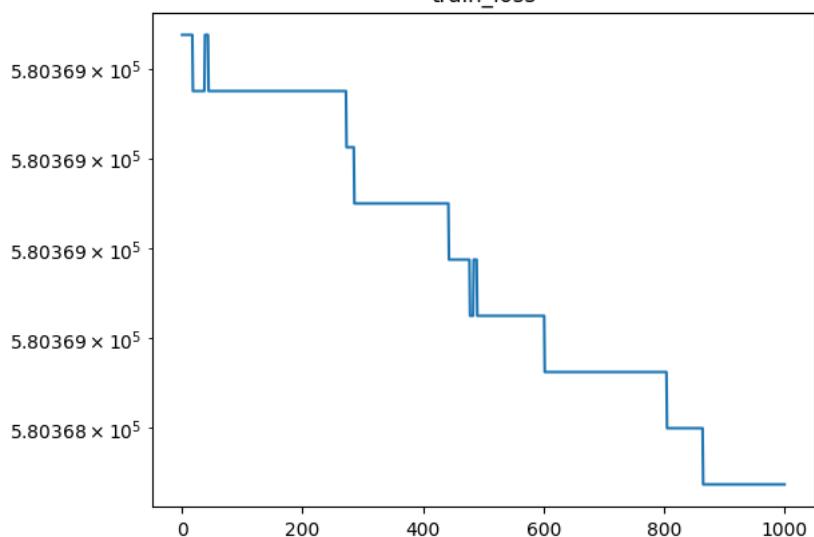
optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-07 , minimum_RMSE: 2469.40 , epoch: 1000 , test_loss: 2469 , train_lo

test_loss



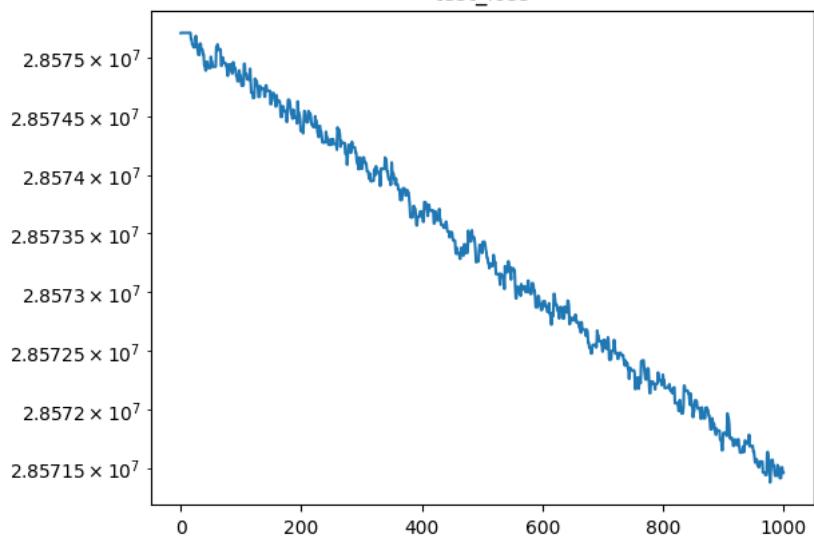


train_loss

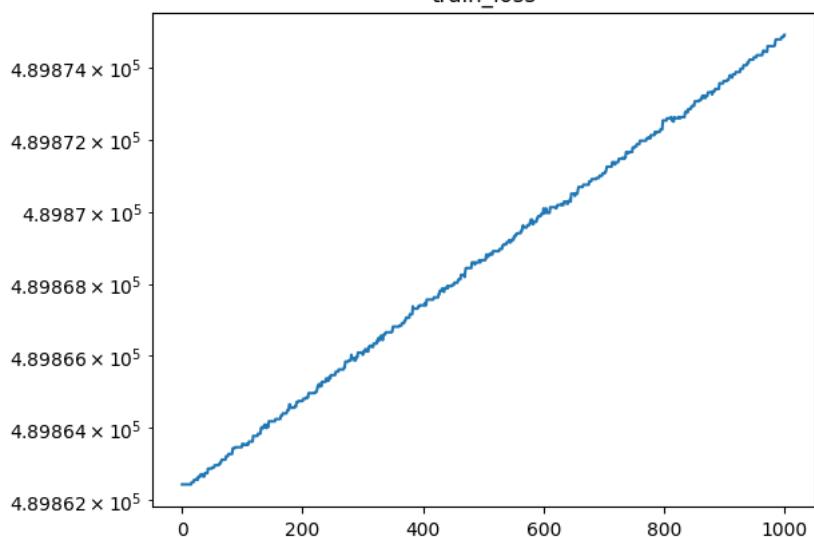


optimizer: Adadelta , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-07 , minimum_RMSE: 5345.22 , epoch: 1000 , test_loss: 5345 , train_loss:

test_loss



train_loss



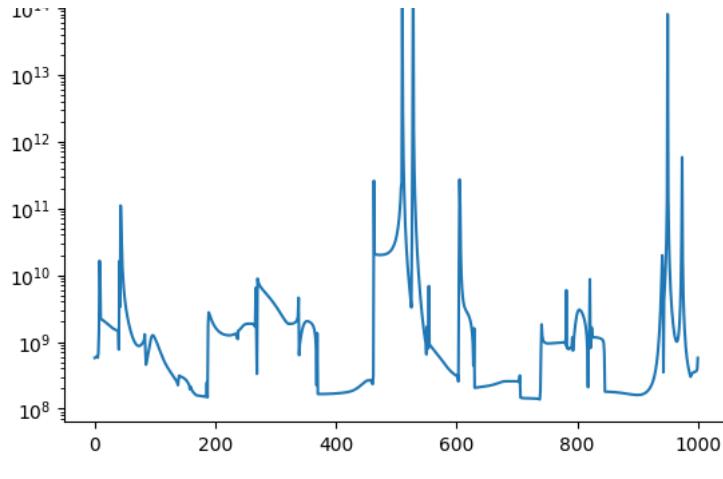
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/loss.py:536: UserWarning: Using a target size (torch.Size([500])) that is different from input size (torch.Size([1000]))
return F.mse_loss(input, target, reduction=self.reduction)

/usr/local/lib/python3.10/dist-packages/torch/nn/modules/loss.py:536: UserWarning: Using a target size (torch.Size([200])) that is different from input size (torch.Size([1000]))
return F.mse_loss(input, target, reduction=self.reduction)

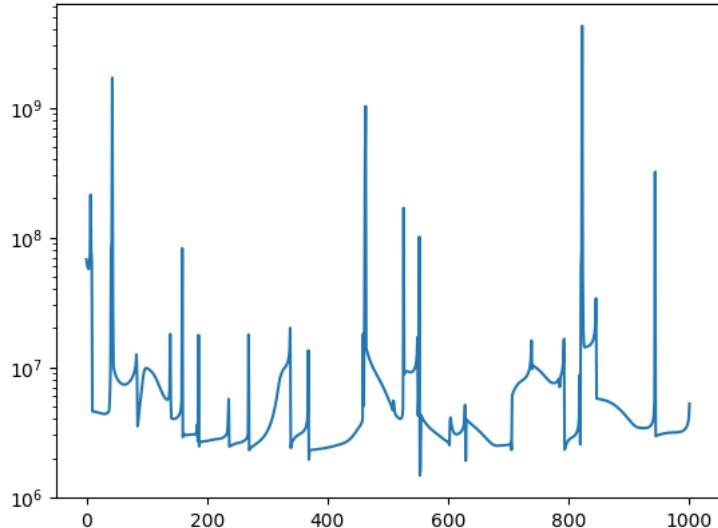
optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 0.1 , minimum_RMSE: 11752.05 , epoch: 1000 , test_loss: 23996 , train_loss:

test_loss

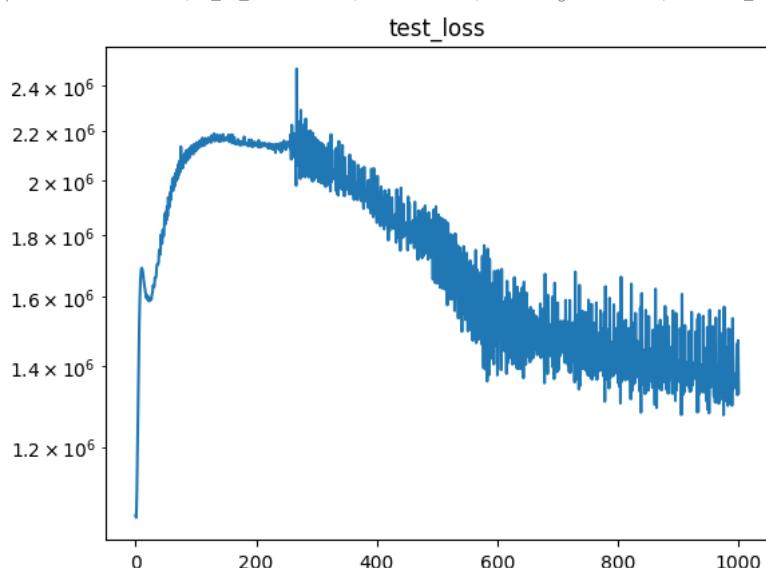




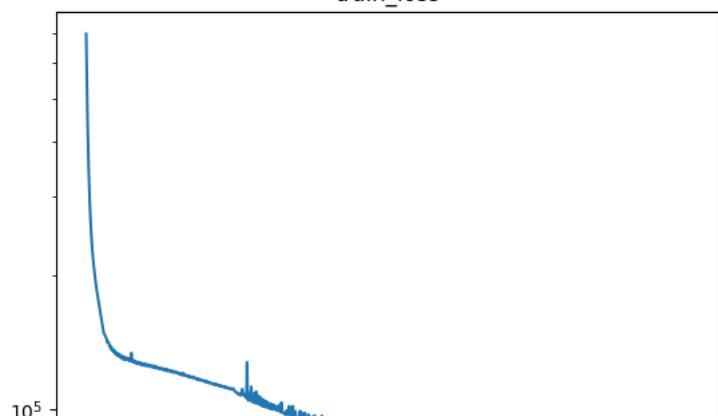
train_loss

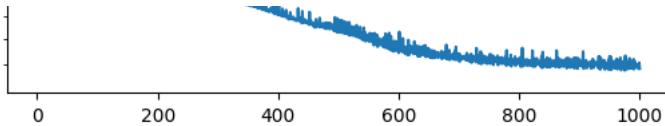


test_loss



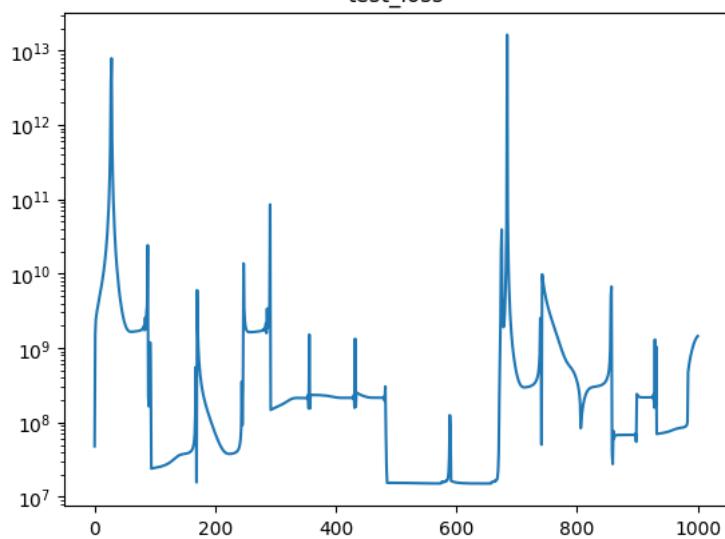
train_loss



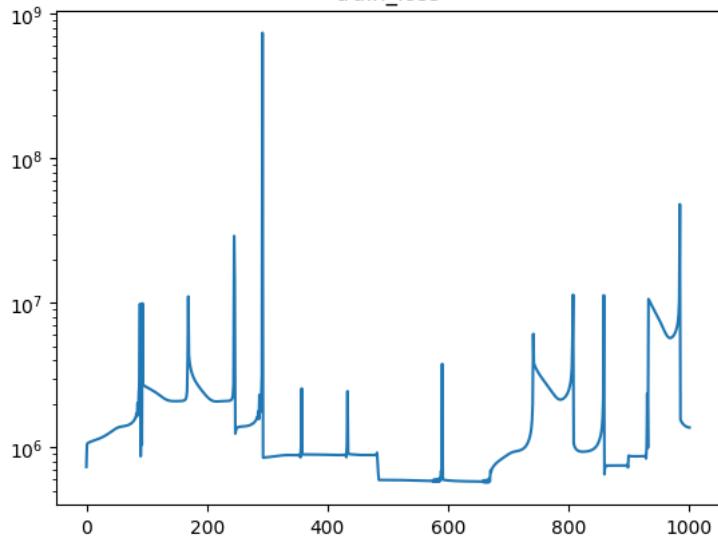


optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 0.01 , minimum_RMSE: 3885.09 , epoch: 1000 , test_loss: 38045 , train_loss:

test_loss

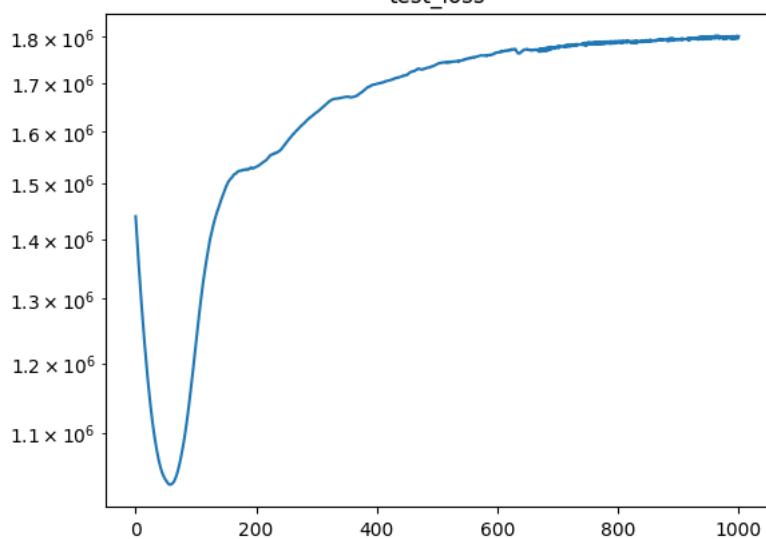


train_loss

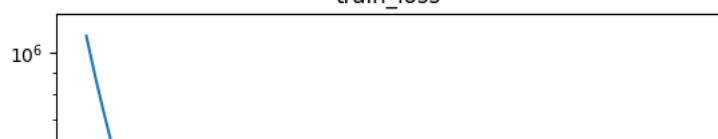


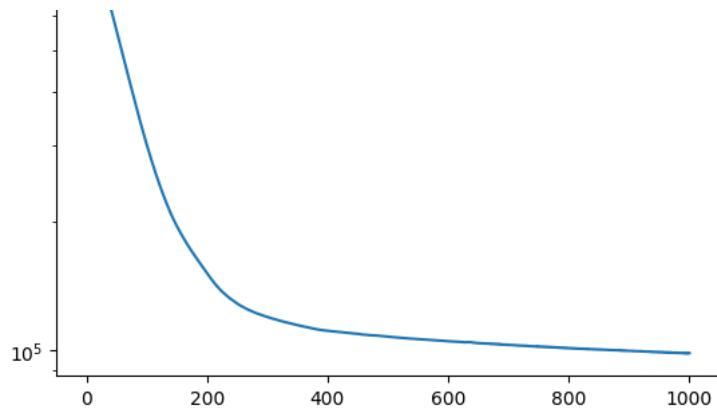
optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 0.05 , minimum_RMSE: 1015.88 , epoch: 1000 , test_loss: 1341 , train_loss:

test_loss

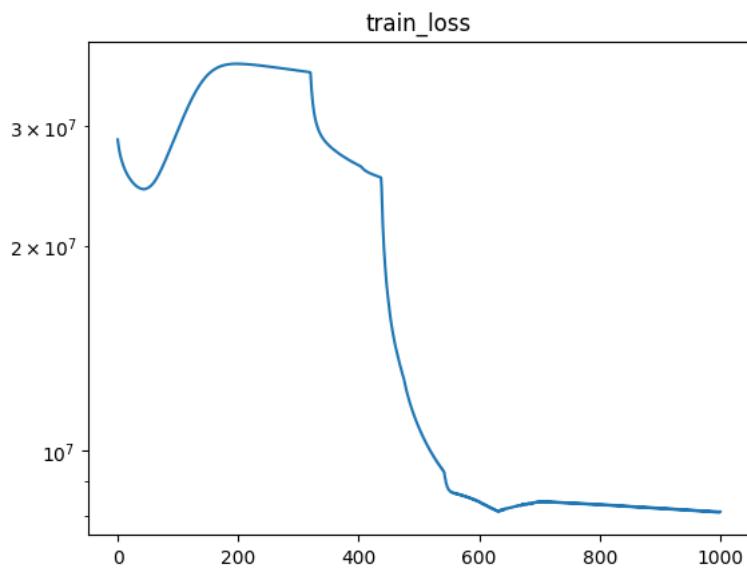
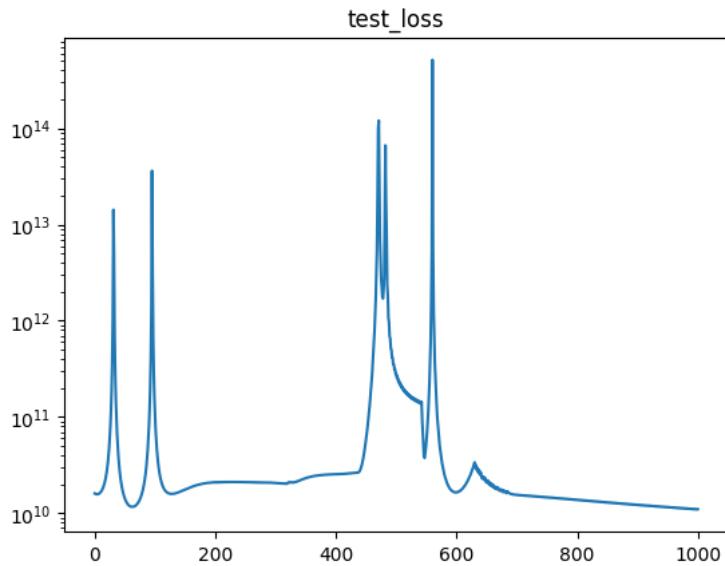


train_loss

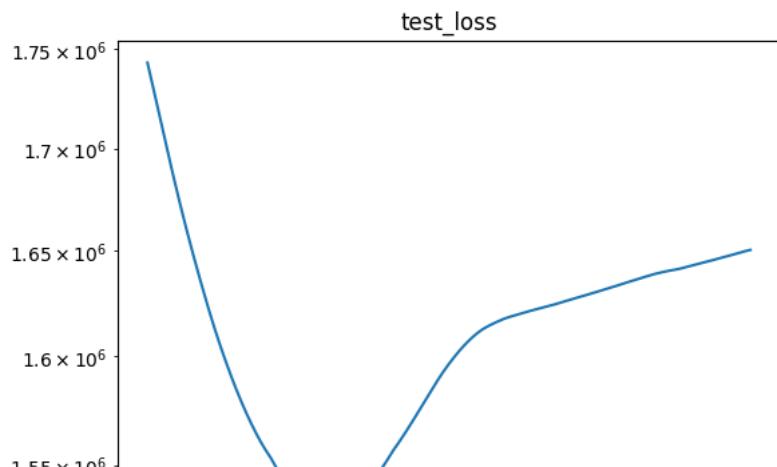


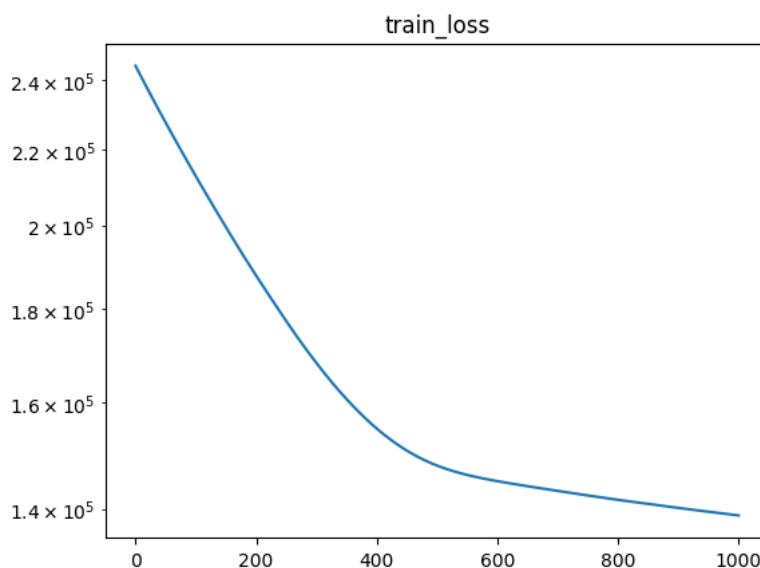
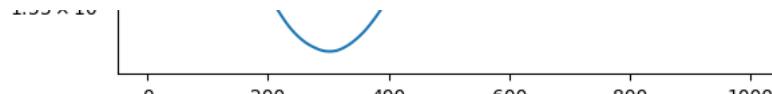


optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 0.001 , minimum_RMSE: 104543.28 , epoch: 1000 , test_loss: 104775 , train_

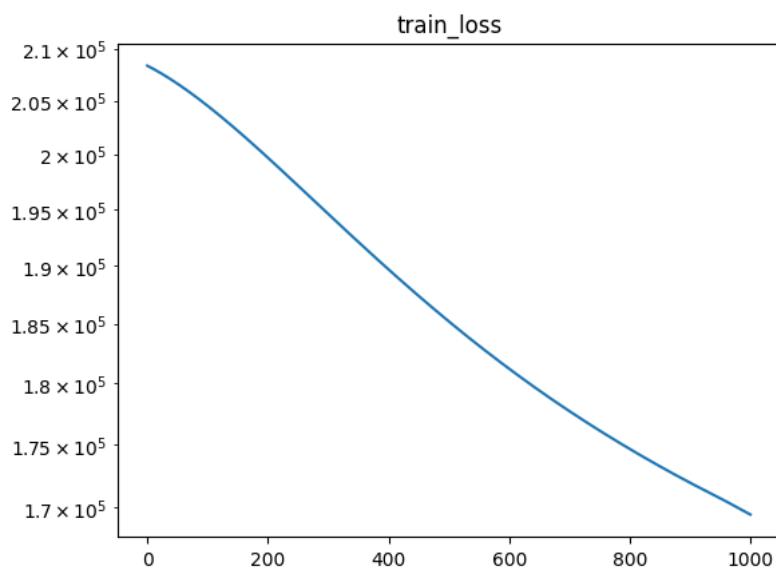
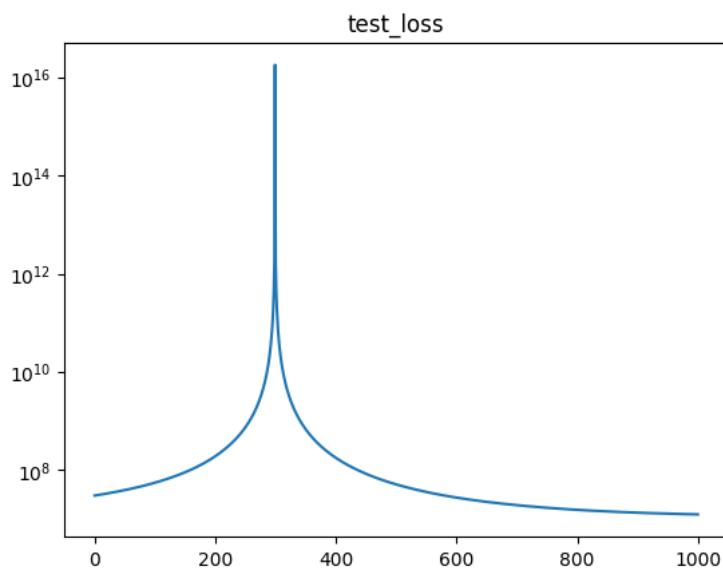


optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 0.005 , minimum_RMSE: 1236.63 , epoch: 1000 , test_loss: 1284 , train_

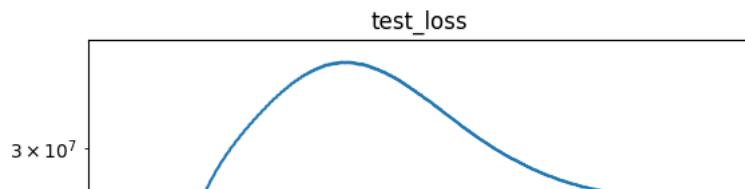


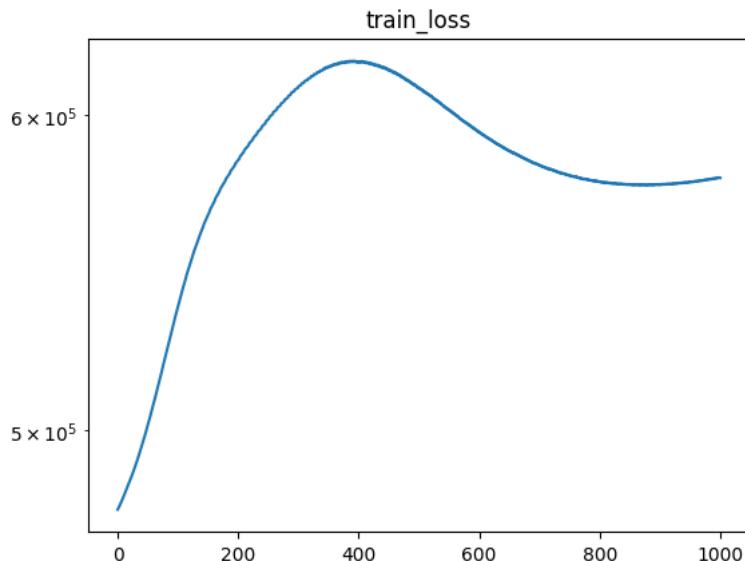
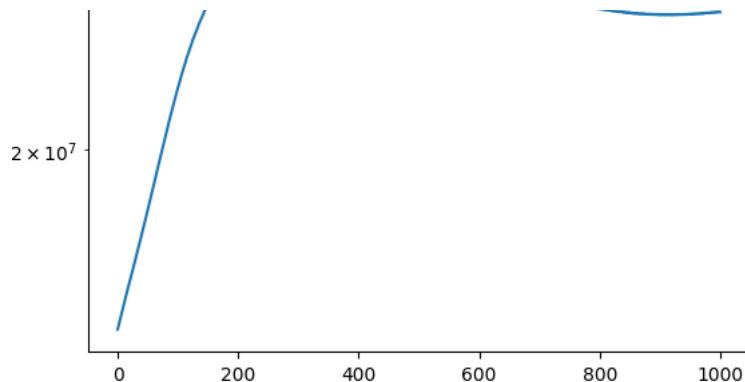


optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 0.0001 , minimum_RMSE: 3527.47 , epoch: 1000 , test_loss: 3527 , train_lo

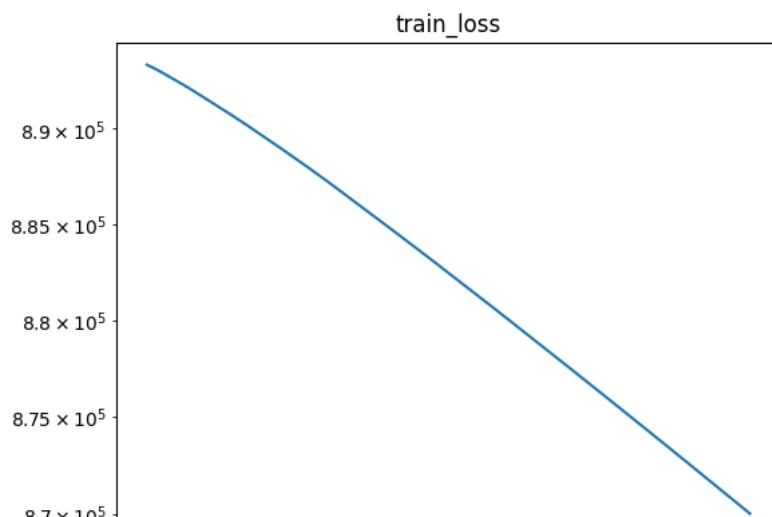
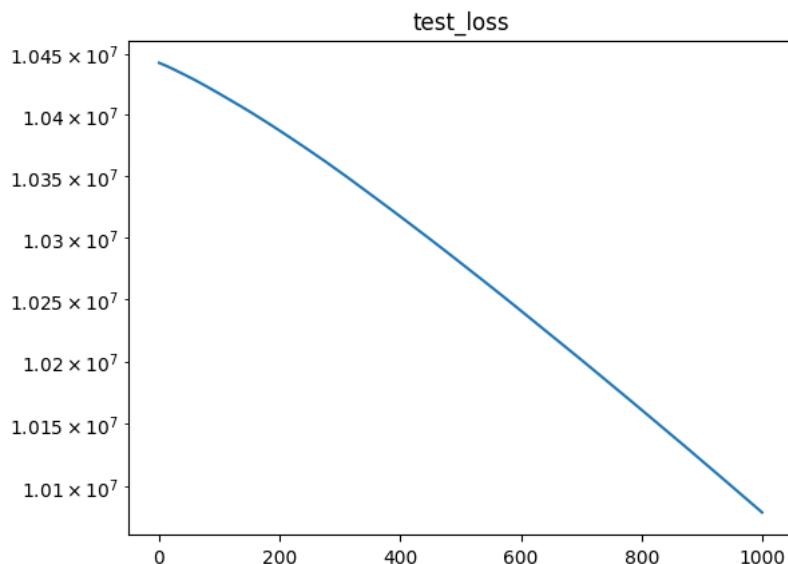


optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 0.0005 , minimum_RMSE: 3667.14 , epoch: 1000 , test_loss: 5205 , train_lo

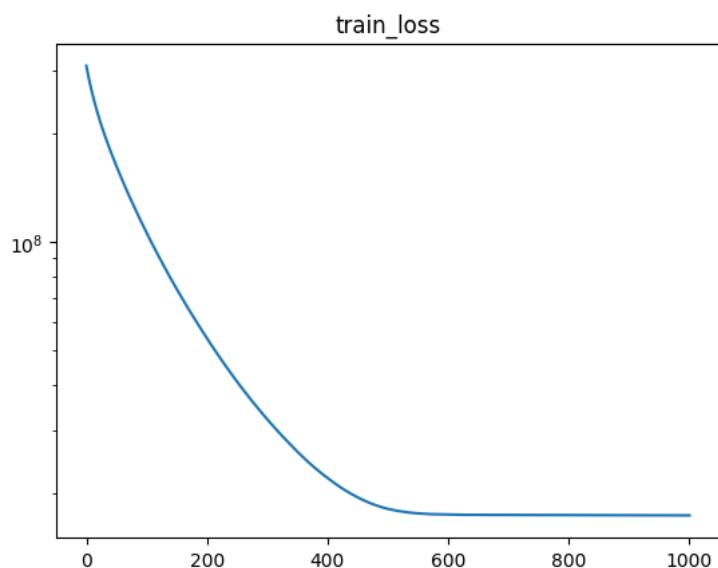
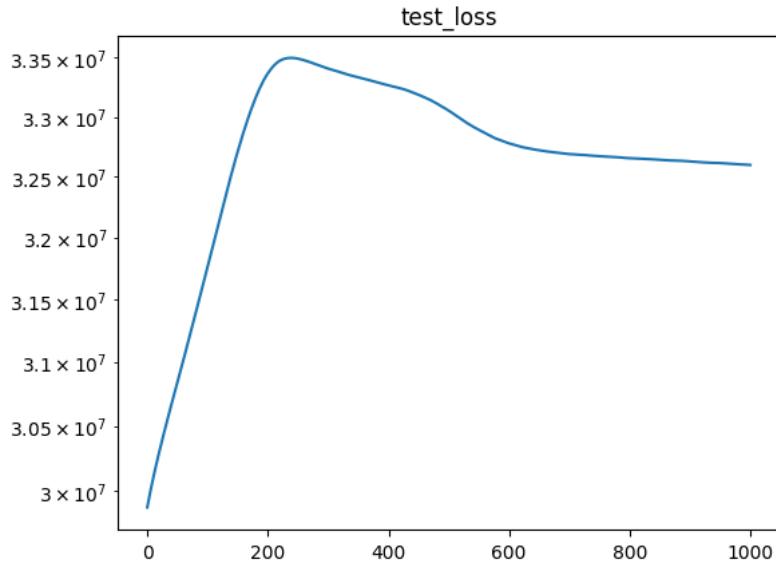




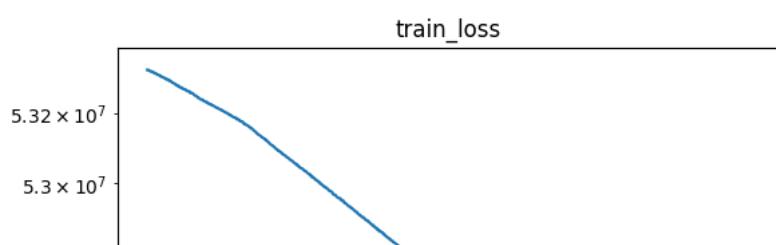
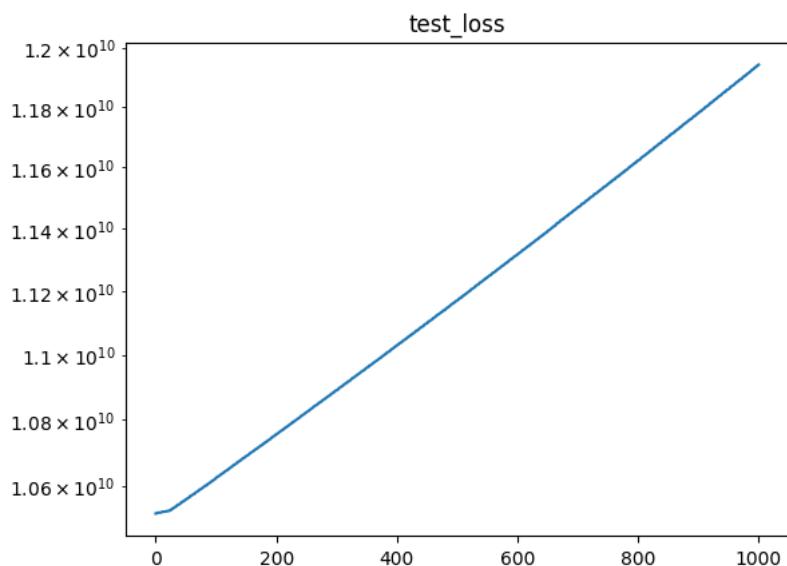
optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-05 , minimum_RMSE: 3174.81 , epoch: 1000 , test_loss: 3174 , train_loss:



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

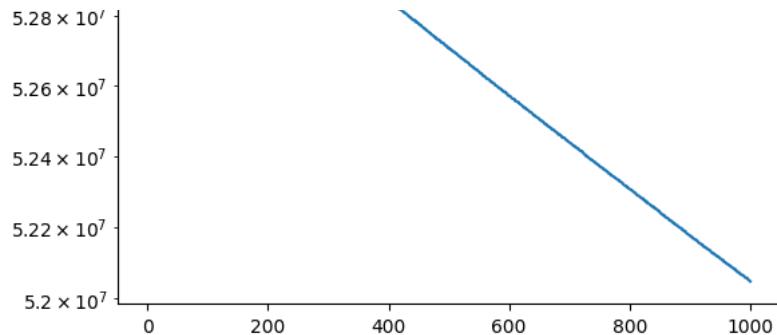


optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-06 , minimum_RMSE: 102550.92 , epoch: 1000 , test_loss: 109282 , train_loss: 109282

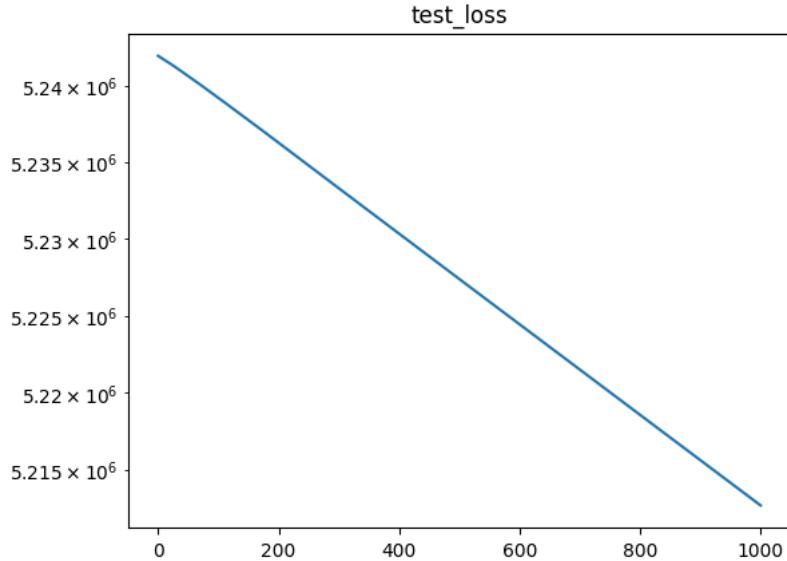


23. 8. 2. 오후 3:59

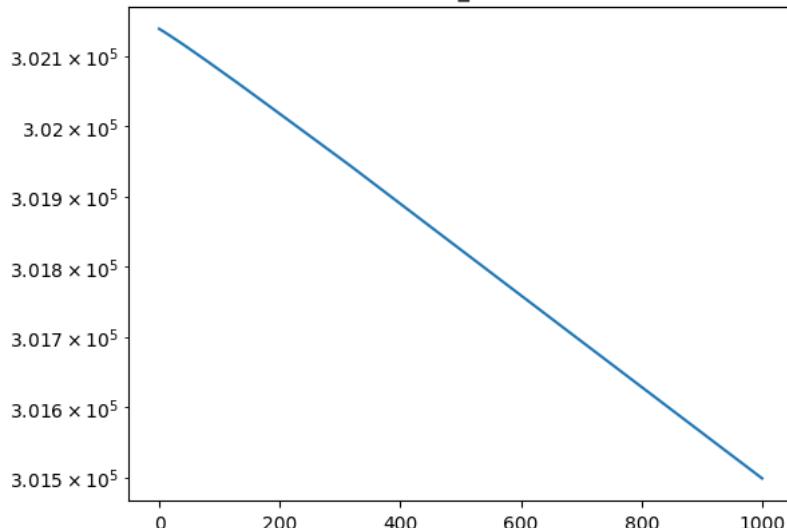
predictingKineticE.ipynb - Colaboratory



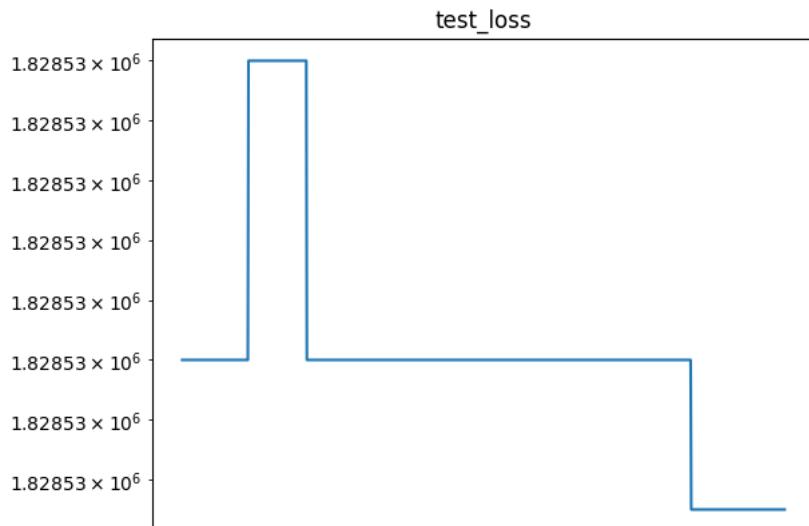
optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 5e-06 , minimum_RMSE: 2283.13 , epoch: 1000 , test_loss: 2283 , train_loss:

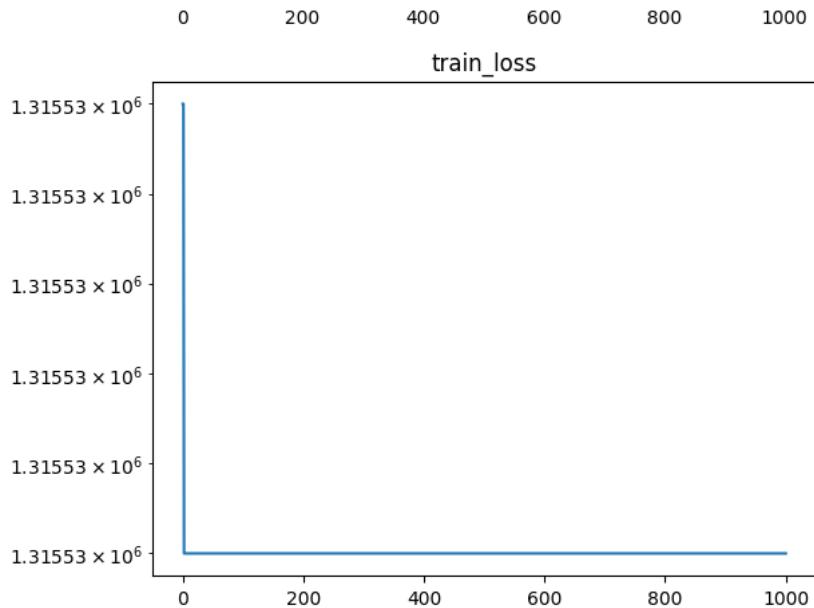


train_loss

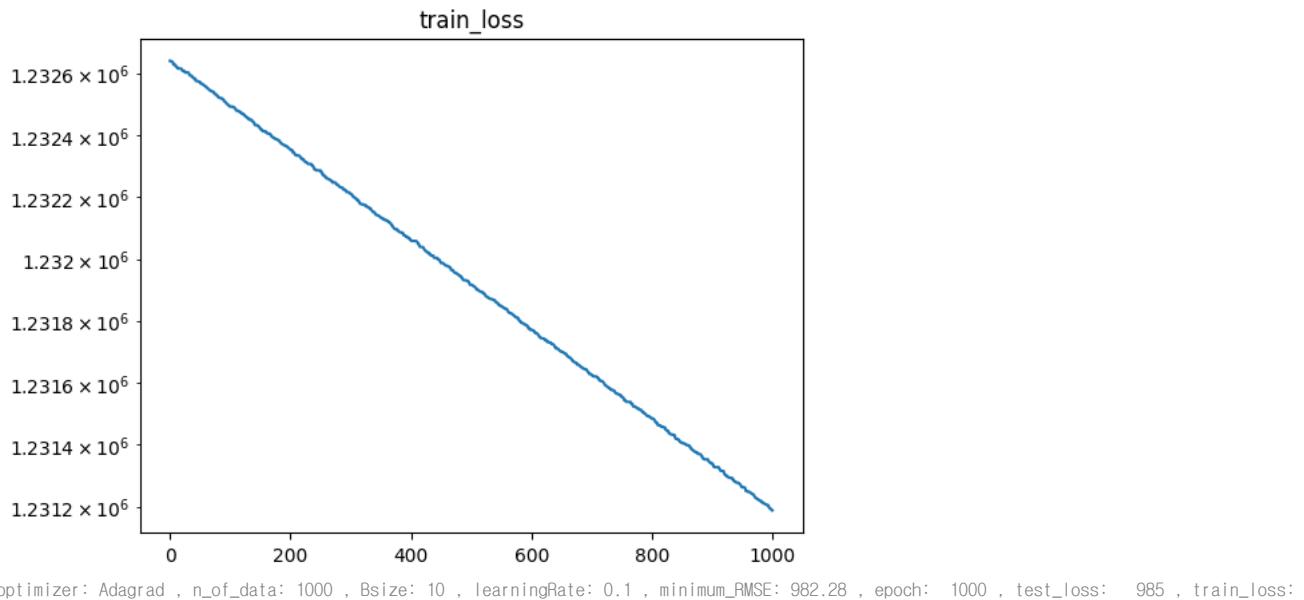
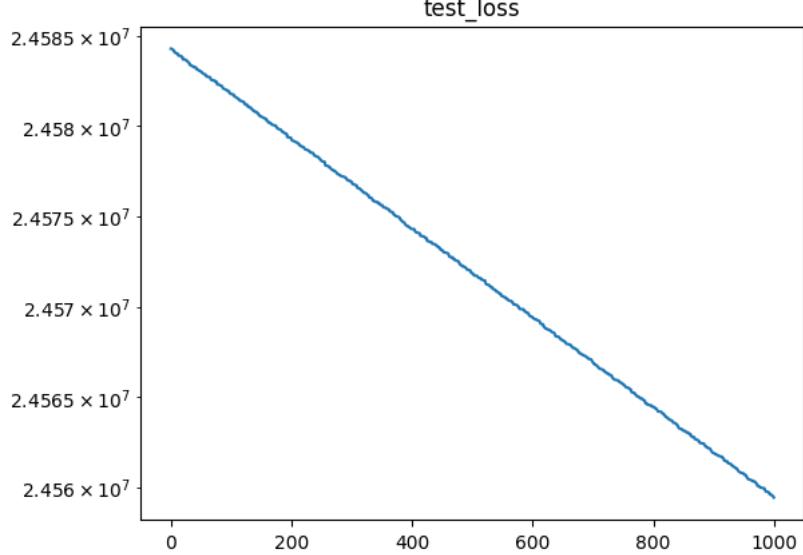


optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-07 , minimum_RMSE: 1352.23 , epoch: 1000 , test_loss: 1352 , train_loss:

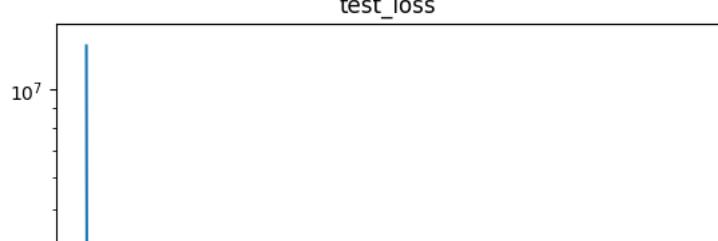


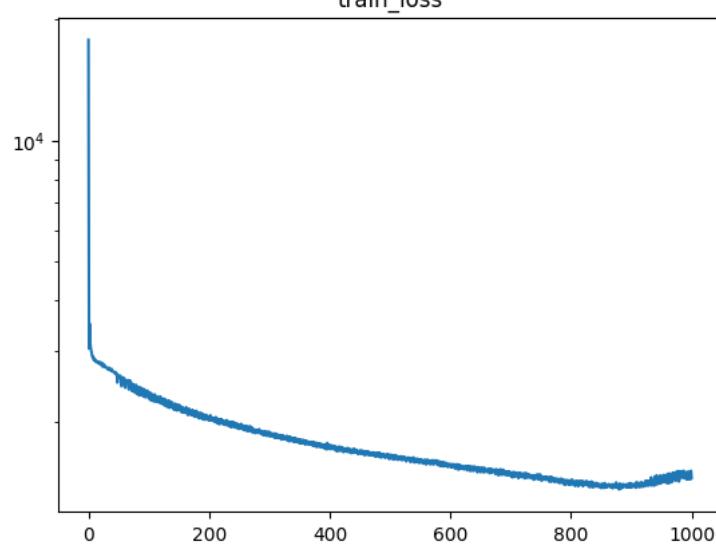
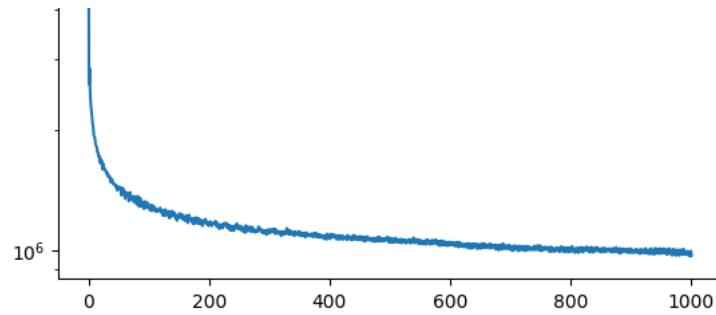


optimizer: Adadelta , n_of_data: 1000 , Bsize: 500 , learningRate: 5e-07 , minimum_RMSE: 4955.75 , epoch: 1000 , test_loss: 4955 , train_loss:



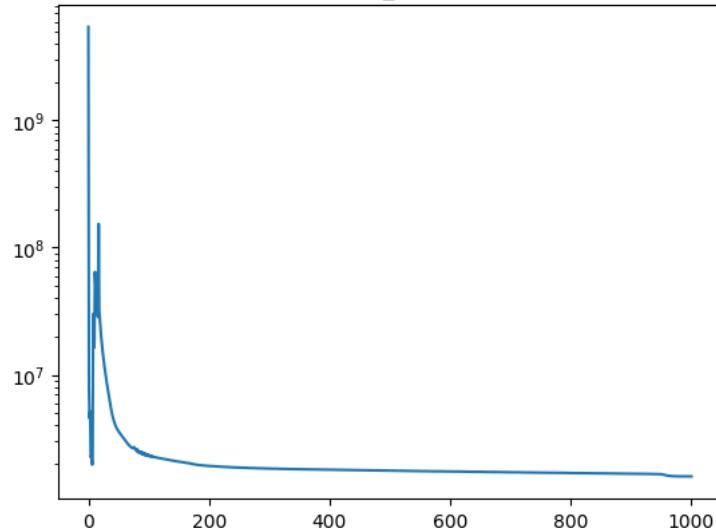
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 0.1 , minimum_RMSE: 982.28 , epoch: 1000 , test_loss: 985 , train_loss:



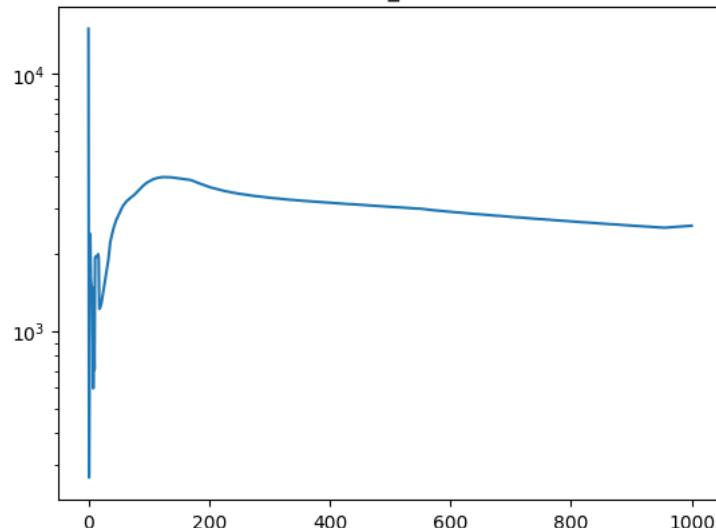


optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 0.5 , minimum_RMSE: 1259.95 , epoch: 1000 , test_loss: 1260 , train_loss:

test_loss



train_loss

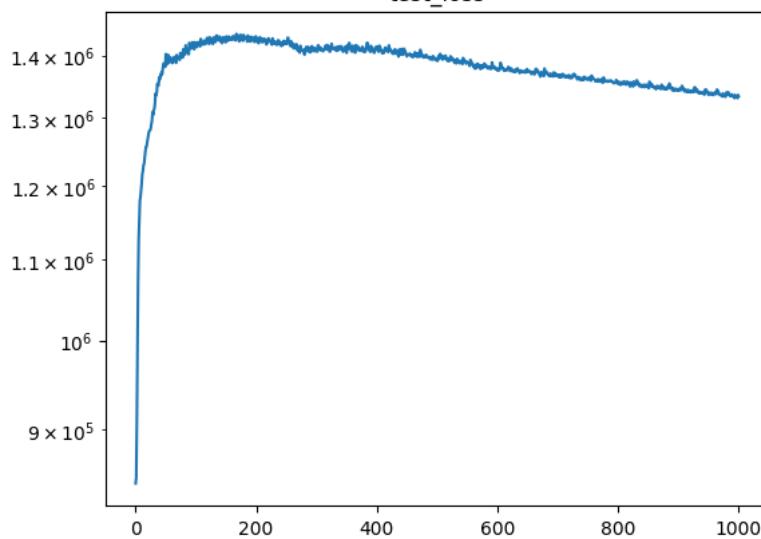


23. 8. 2. 오후 3:59

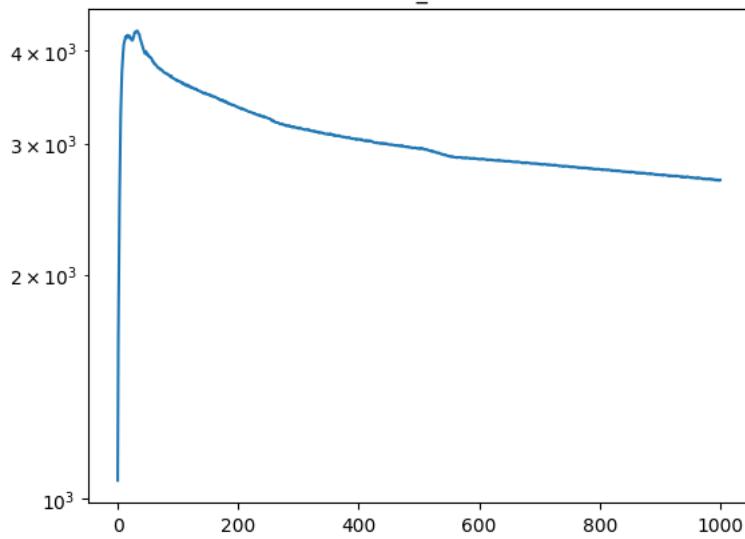
predictingKineticE.ipynb - Colaboratory

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

test_loss

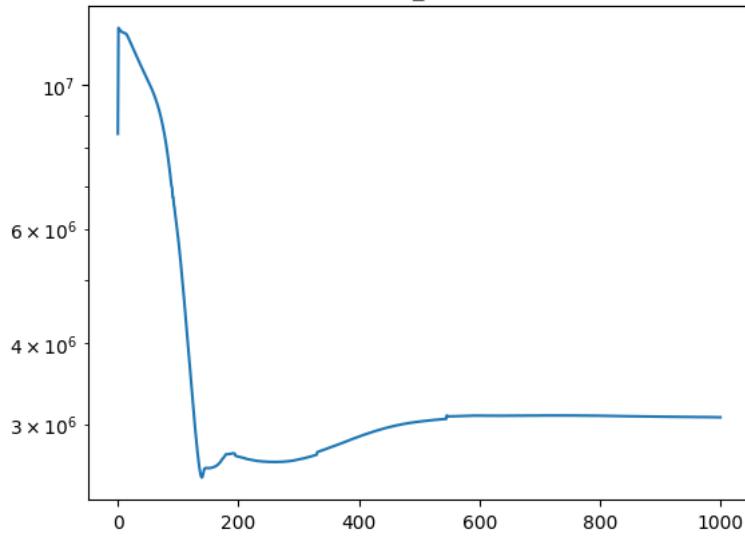


train_loss

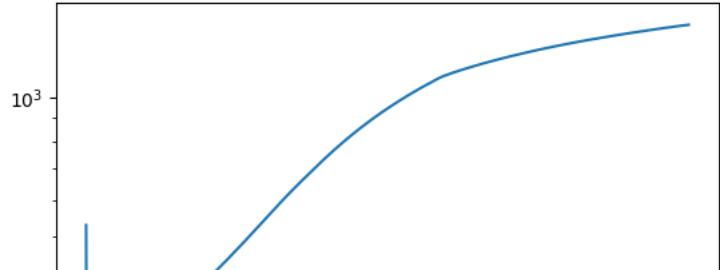


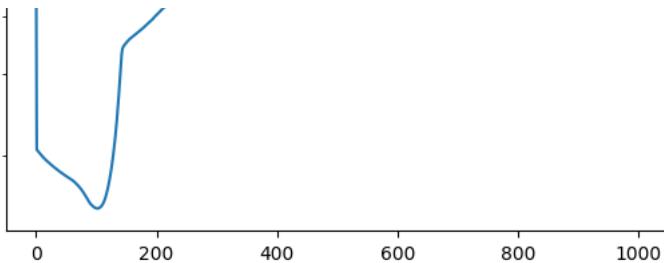
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 0.05 , minimum_RMSE: 1576.89 , epoch: 1000 , test_loss: 1754 , train_loss:

test_loss

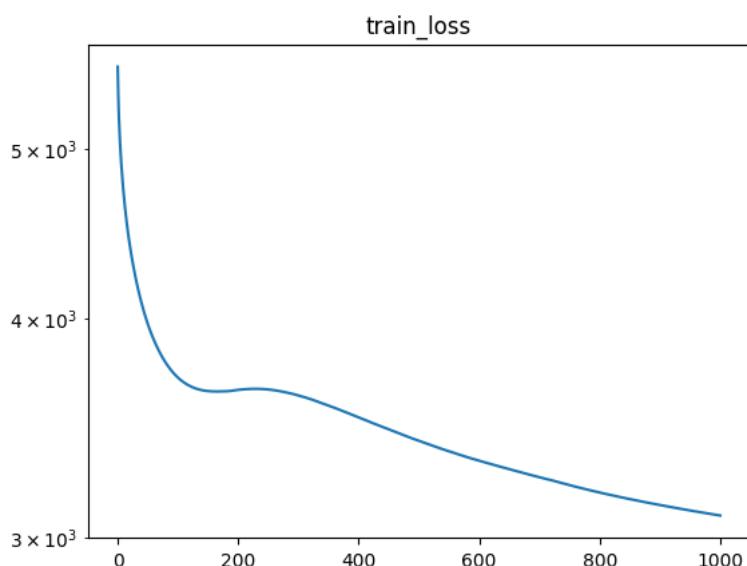
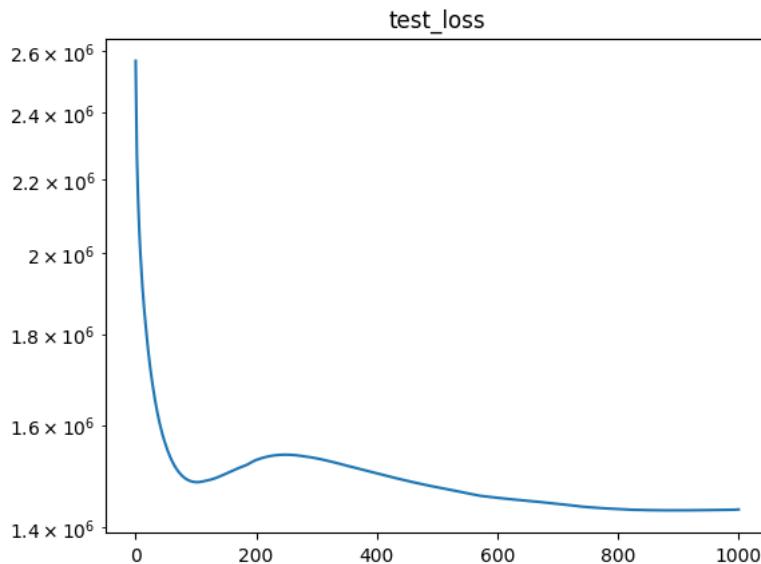


train_loss

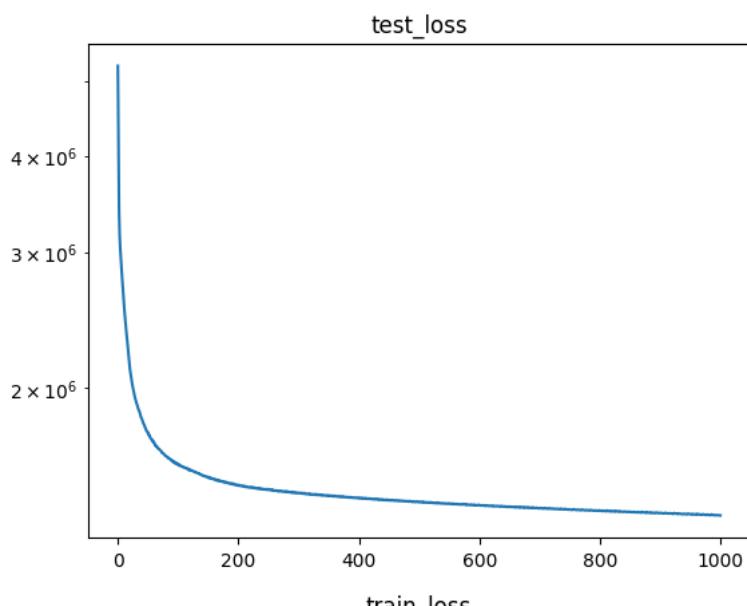




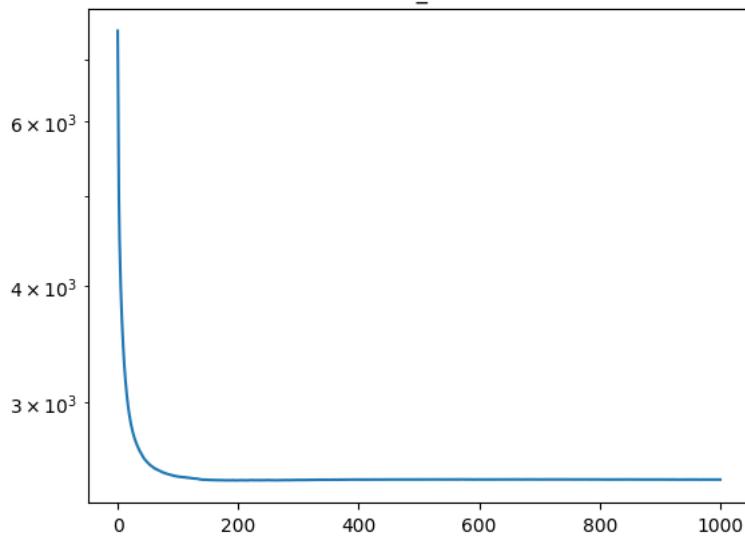
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 0.001 , minimum_RMSE: 1196.63 , epoch: 1000 , test_loss: 1197 , train_loss



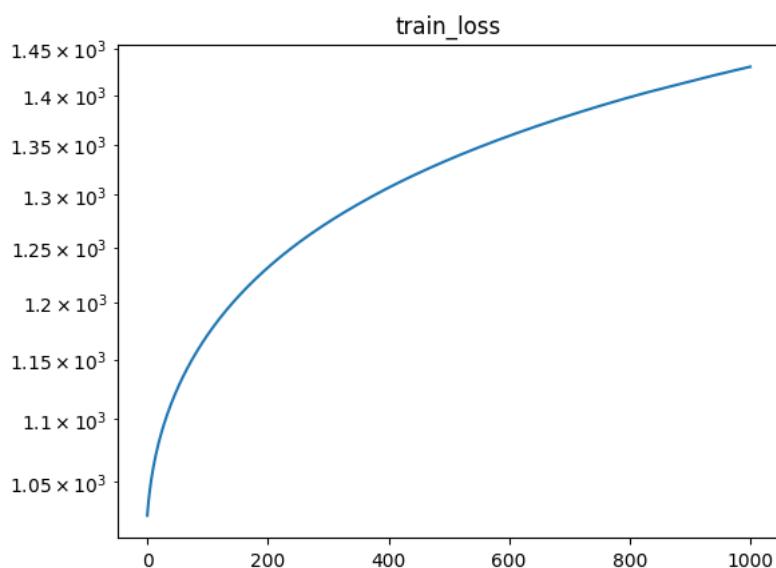
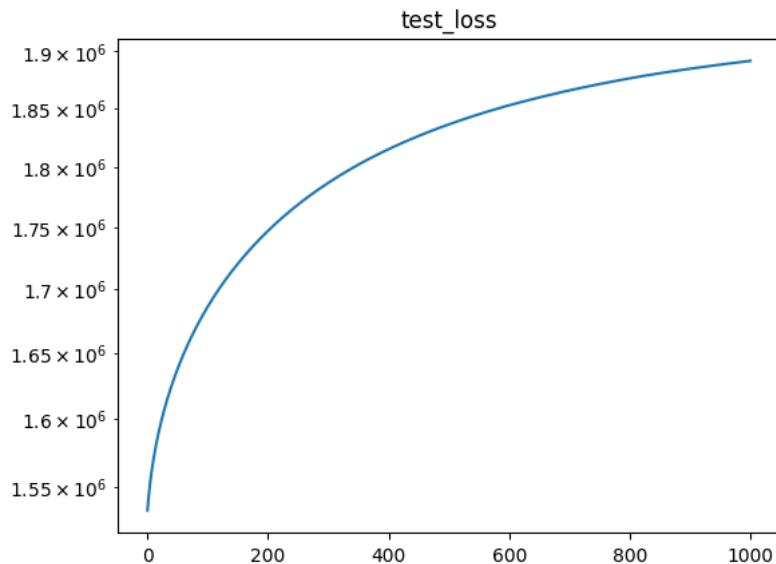
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 0.005 , minimum_RMSE: 1168.44 , epoch: 1000 , test_loss: 1168 , train_loss



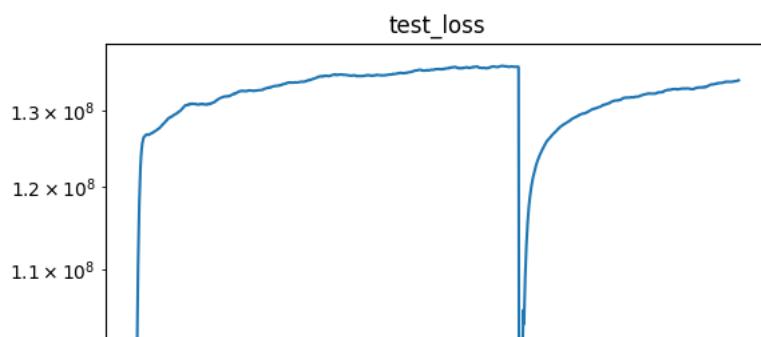
train loss

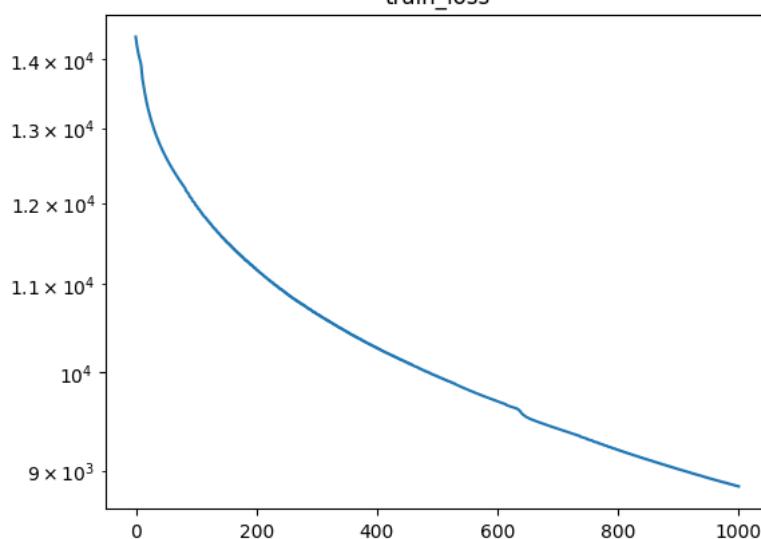
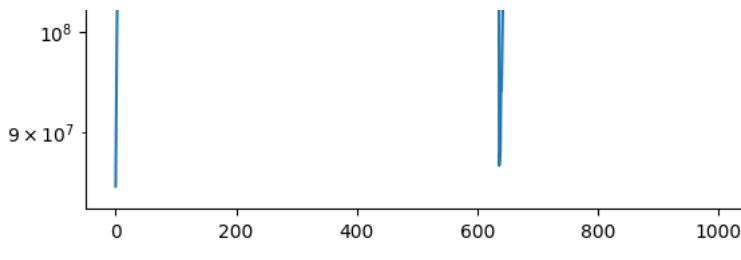


optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 0.0001 , minimum_RMSE: 1238.01 , epoch: 1000 , test_loss: 1375 , train_loss:

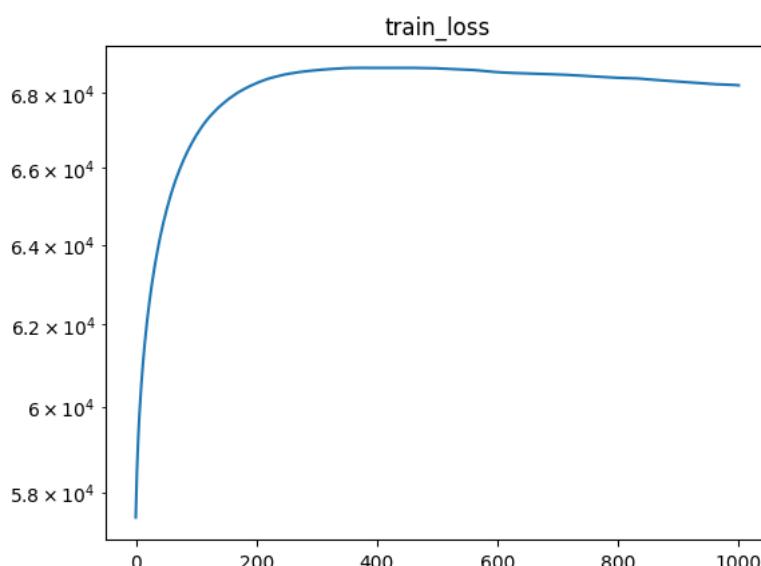
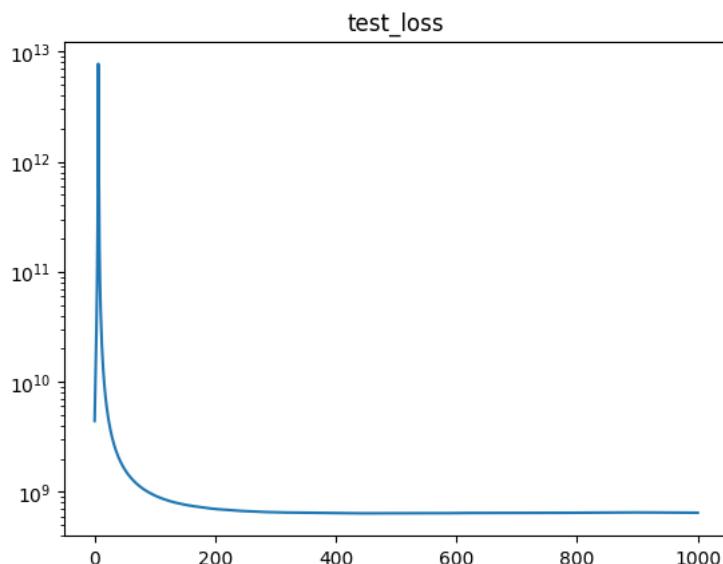


optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 0.0005 , minimum_RMSE: 9221.62 , epoch: 1000 , test_loss: 11580 , train_loss:





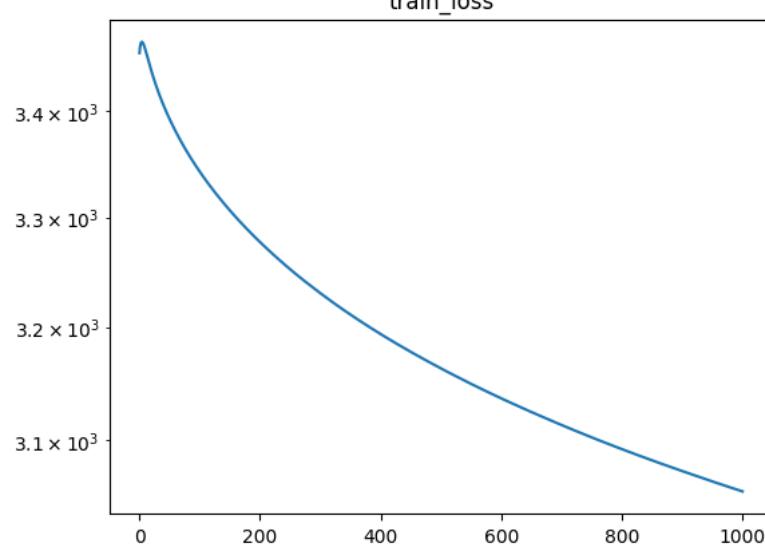
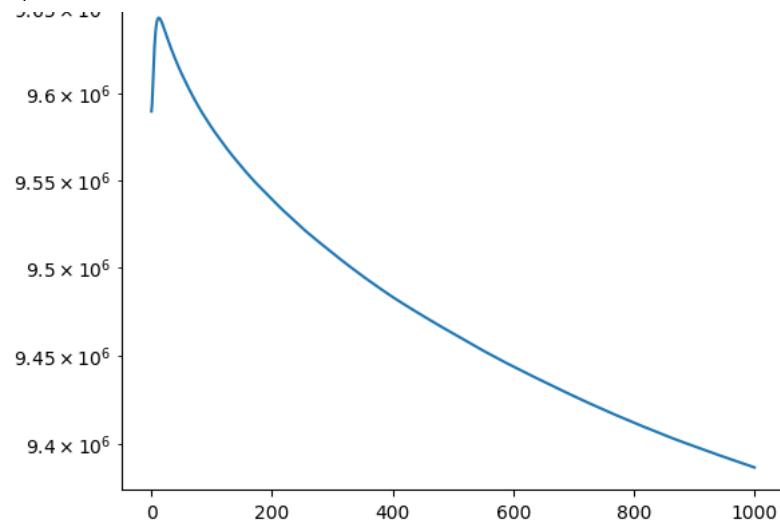
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-05 , minimum_RMSE: 25226.23 , epoch: 1000 , test_loss: 25400 , train_loss:



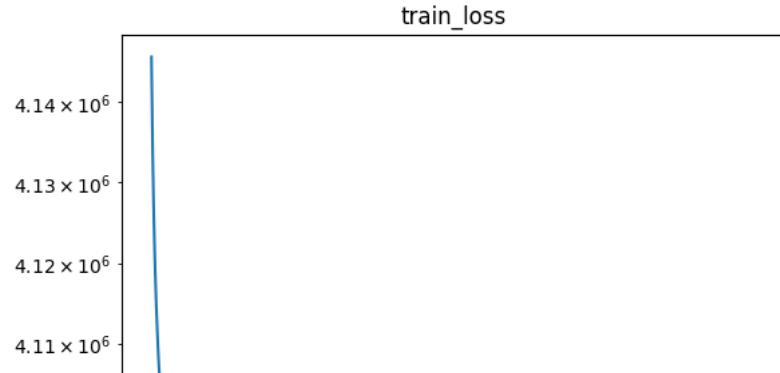
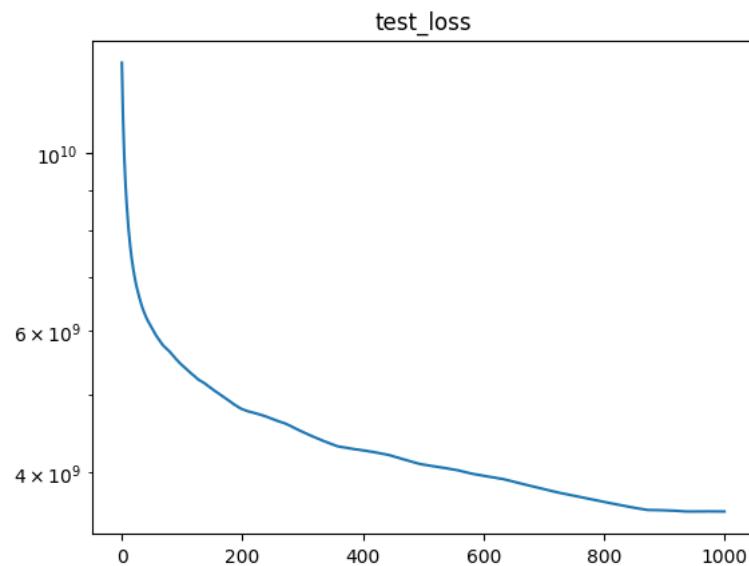
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 5e-05 , minimum_RMSE: 3063.73 , epoch: 1000 , test_loss: 3063 , train_loss:

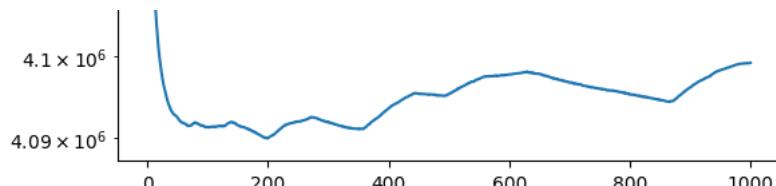
test_loss

6.65 x 10⁶



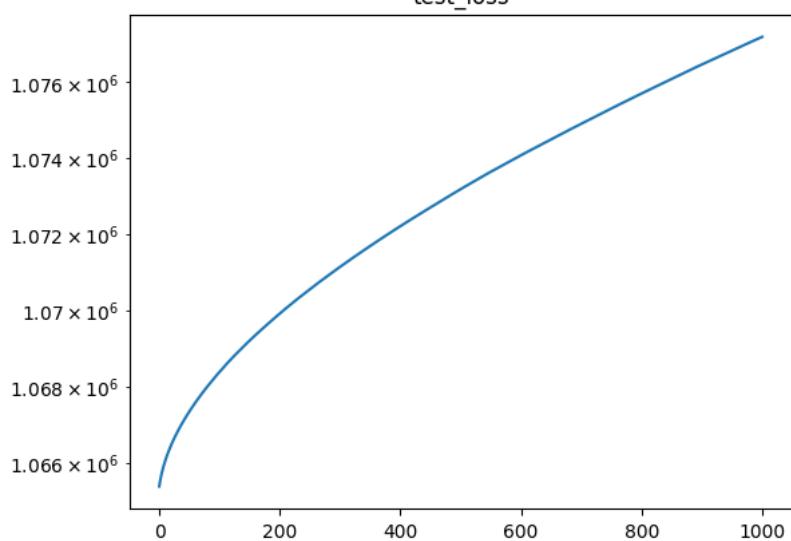
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-06 , minimum_RMSE: 59797.25 , epoch: 1000 , test_loss: 59797 , train_loss:



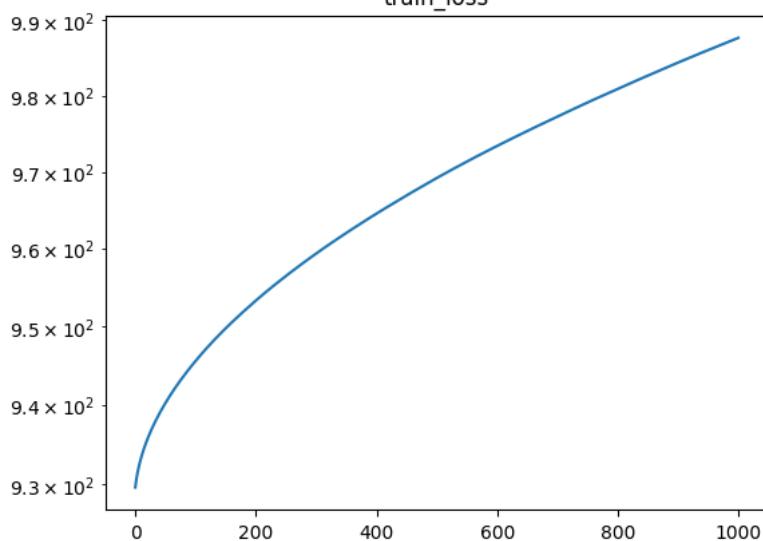


optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 5e-06 , minimum_RMSE: 1032.17 , epoch: 1000 , test_loss: 1037 , train_loss:

test_loss

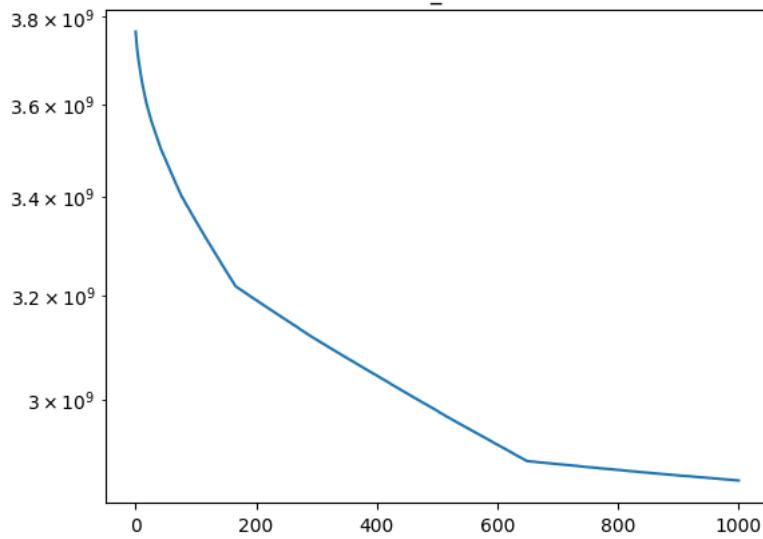


train_loss

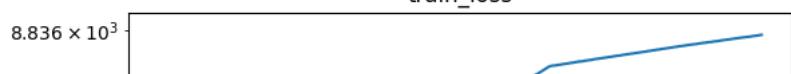


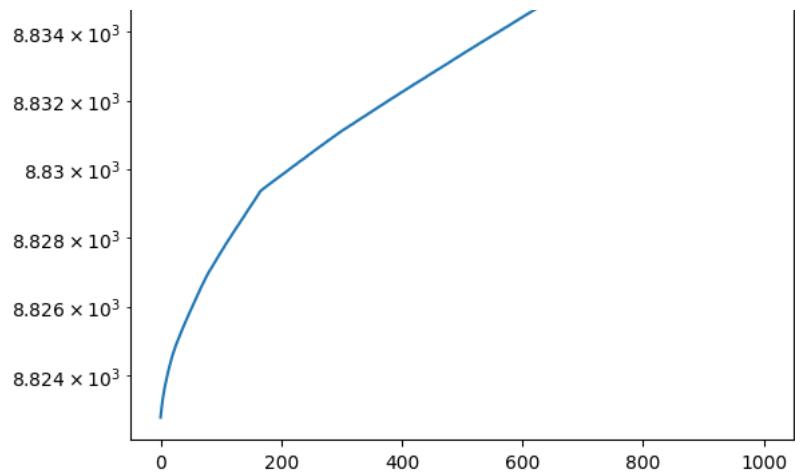
optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-07 , minimum_RMSE: 53423.01 , epoch: 1000 , test_loss: 53423 , train_loss:

test_loss



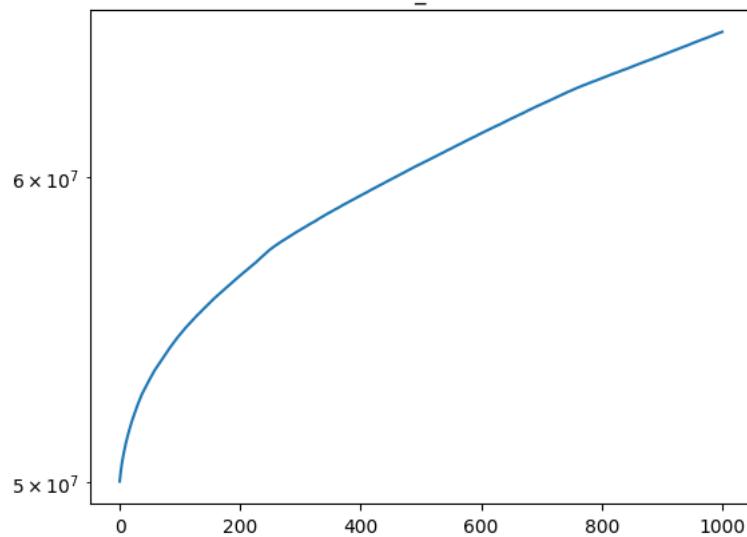
train_loss



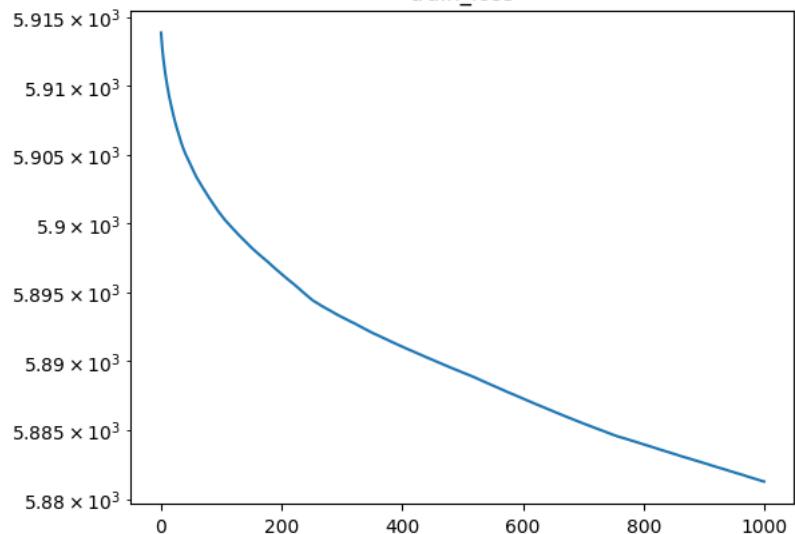


optimizer: Adagrad , n_of_data: 1000 , Bsize: 10 , learningRate: 5e-07 , minimum_RMSE: 7073.55 , epoch: 1000 , test_loss: 8091 , train_loss:

test_loss

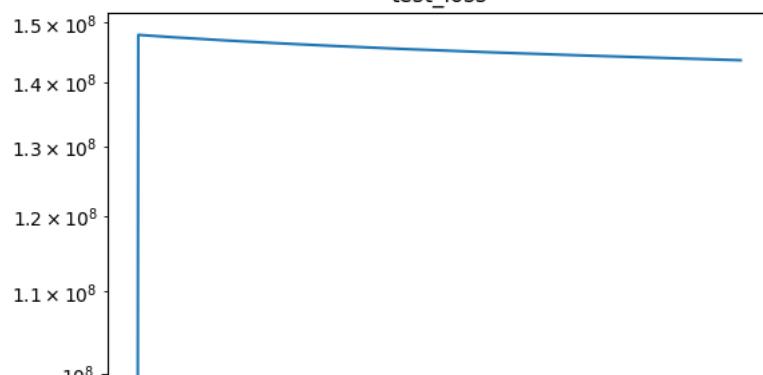


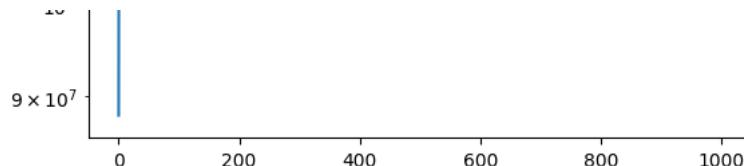
train_loss



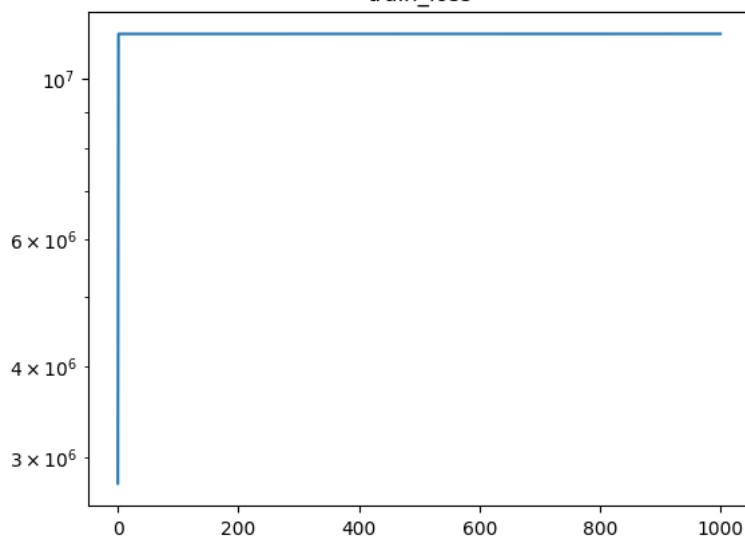
optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 0.1 , minimum_RMSE: 9382.73 , epoch: 1000 , test_loss: 11982 , train_loss:

test_loss



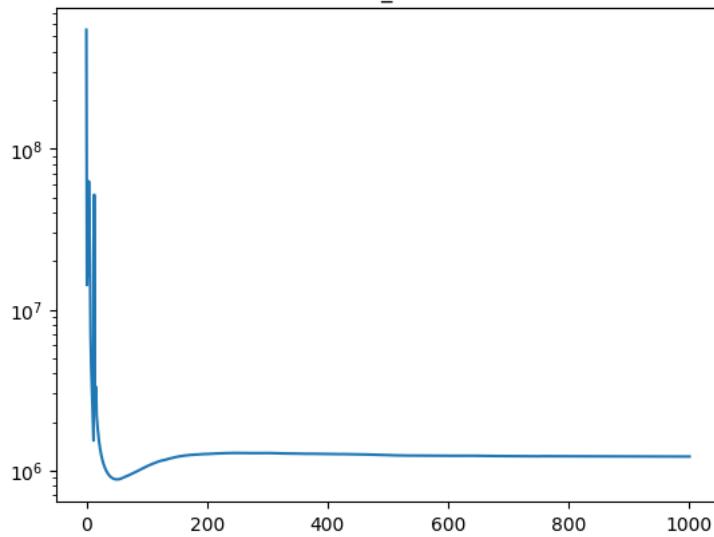


train_loss

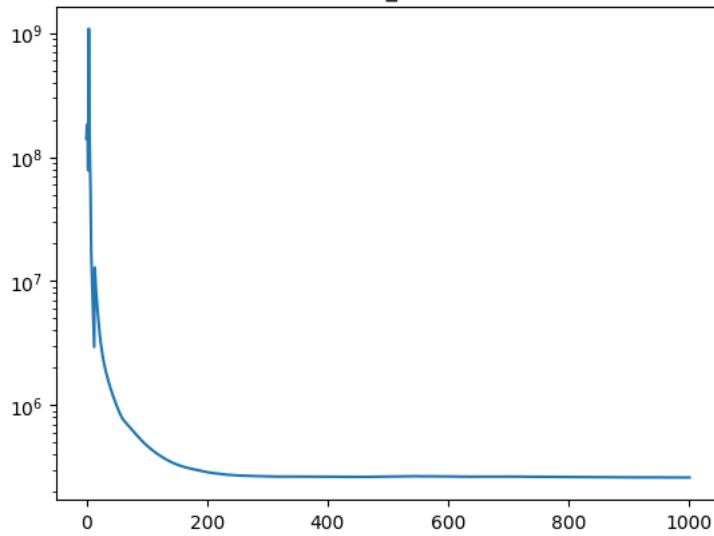


optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 0.5 , minimum_RMSE: 937.01 , epoch: 1000 , test_loss: 1103 , train_loss:

test_loss

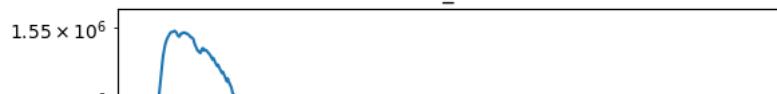


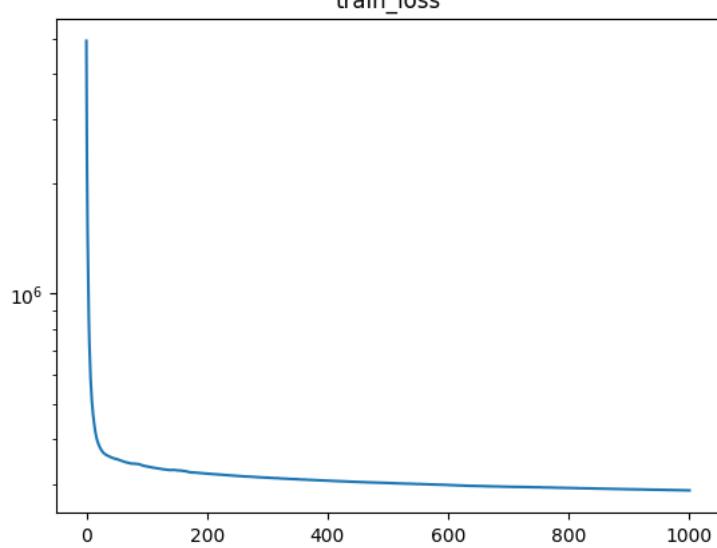
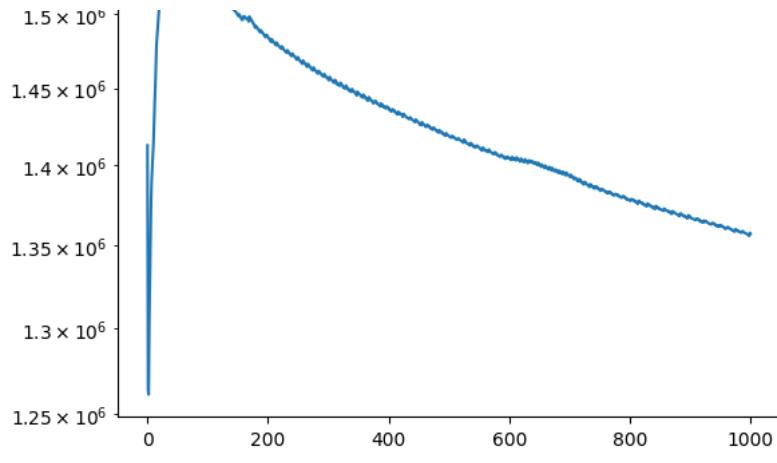
train_loss



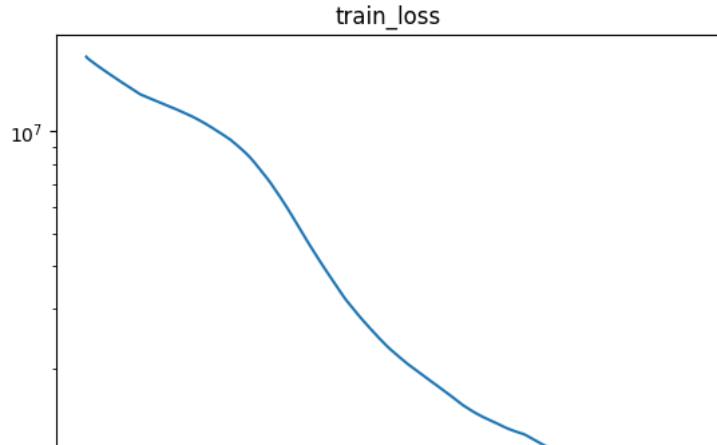
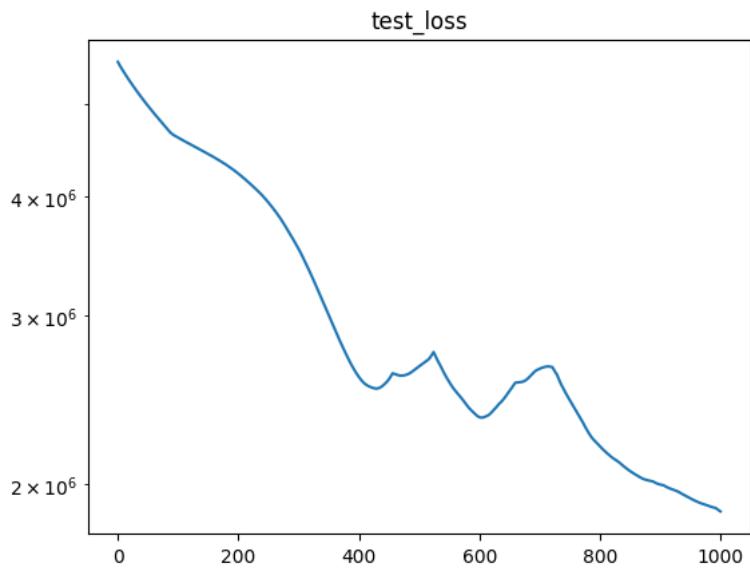
optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 0.01 , minimum_RMSE: 1123.16 , epoch: 1000 , test_loss: 1165 , train_loss:

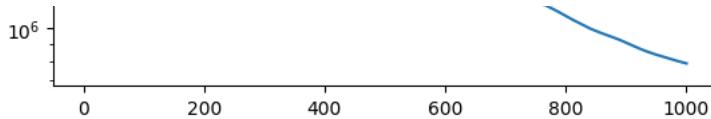
test_loss





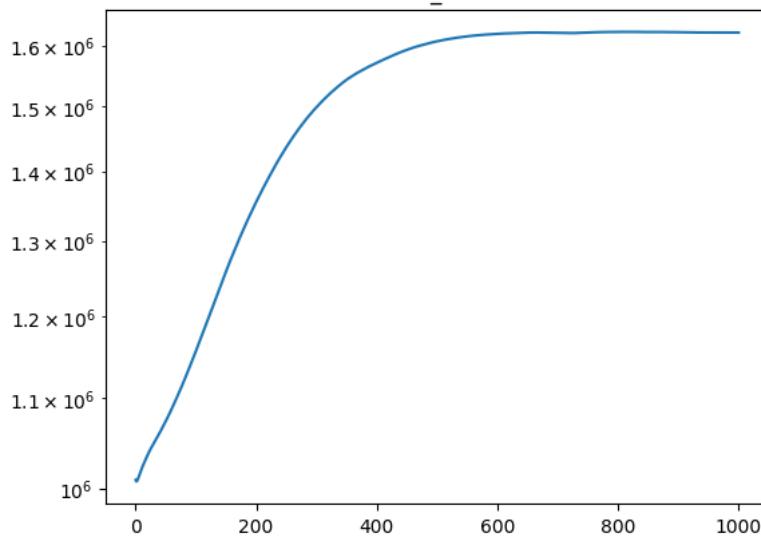
optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 0.05 , minimum_RMSE: 1368.86 , epoch: 1000 , test_loss: 1368 , train_loss:



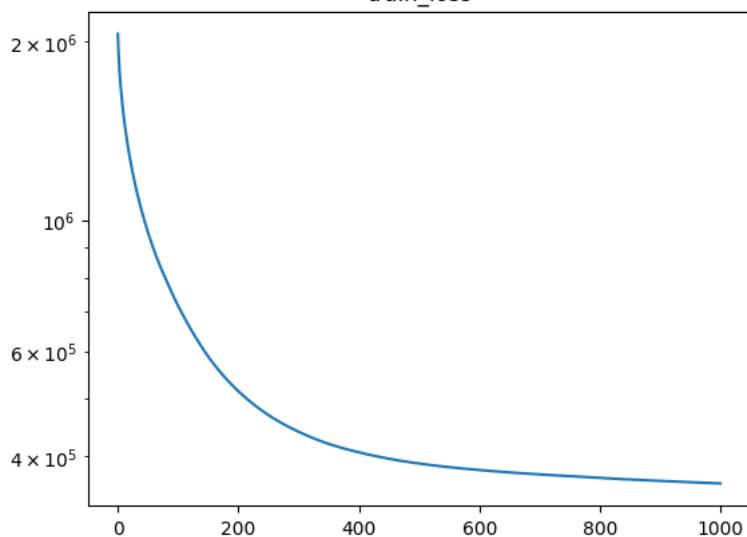


optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 0.001 , minimum_RMSE: 1003.61 , epoch: 1000 , test_loss: 1273 , train_loss

test_loss

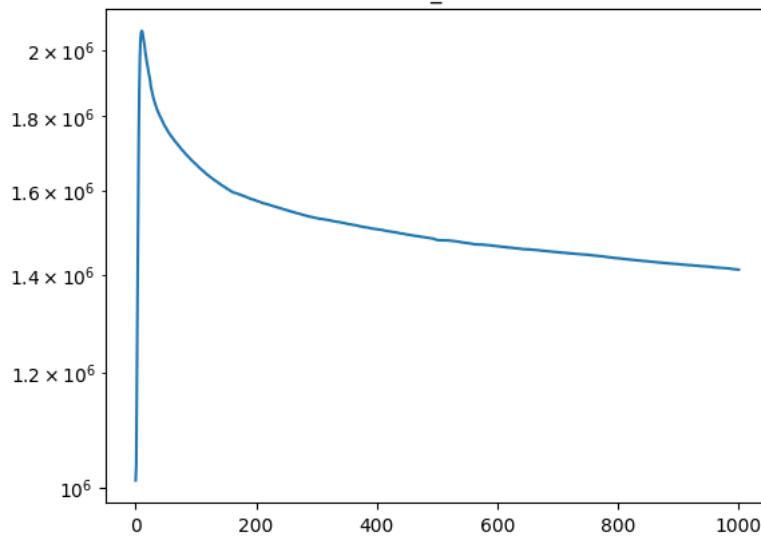


train_loss



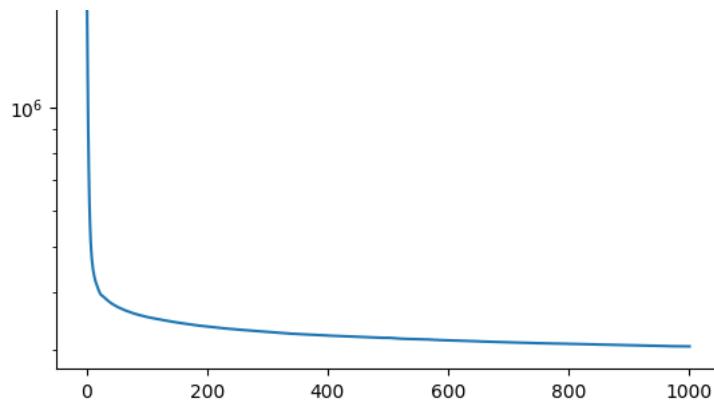
optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 0.005 , minimum_RMSE: 1005.49 , epoch: 1000 , test_loss: 1188 , train_loss

test_loss



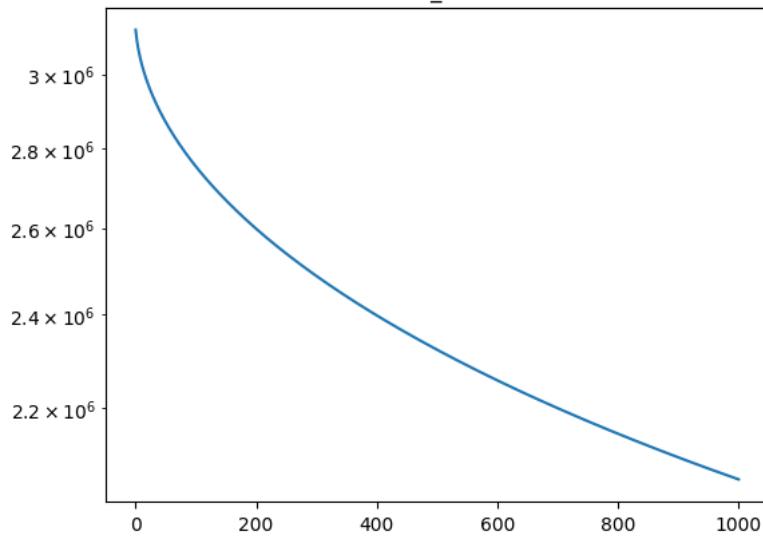
train_loss



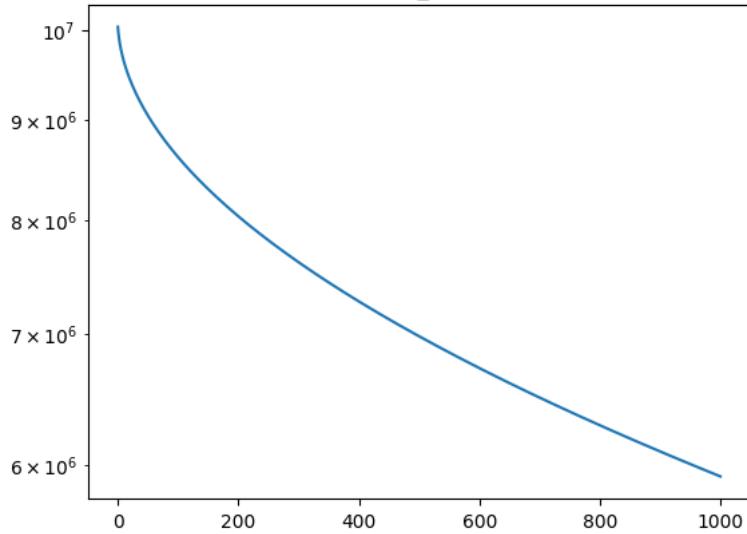


optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 0.0001 , minimum_RMSE: 1435.02 , epoch: 1000 , test_loss: 1435 , train_loss:

test_loss

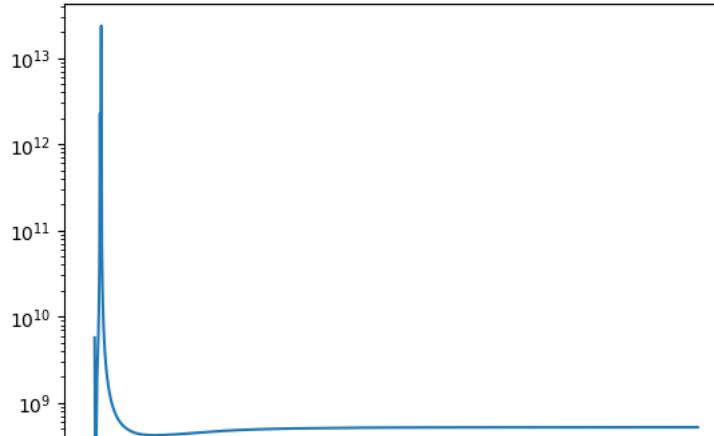


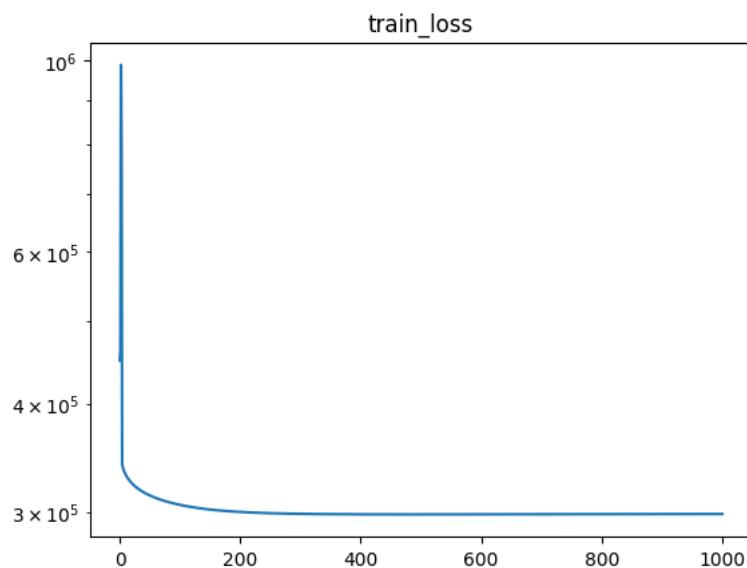
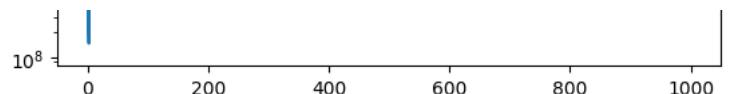
train_loss



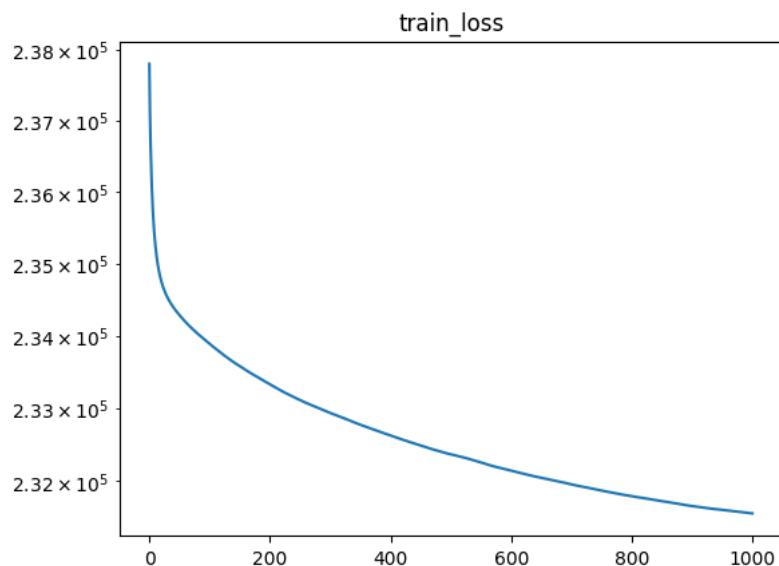
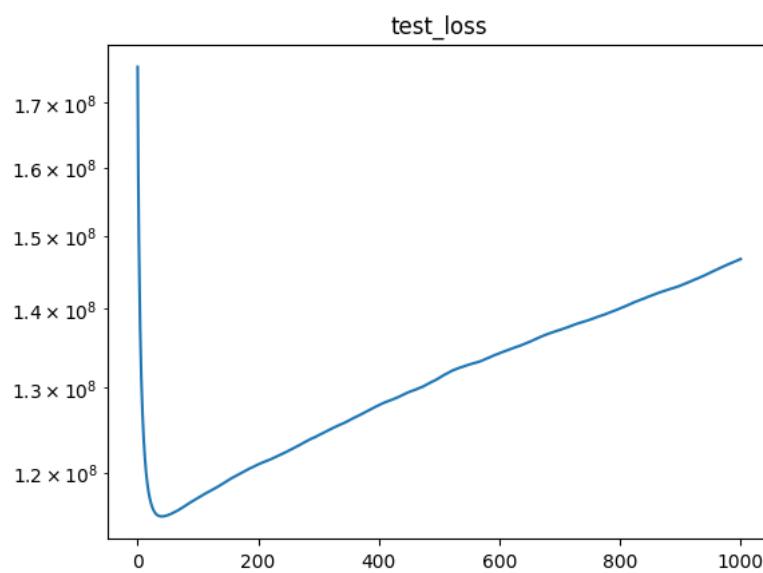
optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 0.0005 , minimum_RMSE: 12155.32 , epoch: 1000 , test_loss: 22842 , train_loss:

test_loss

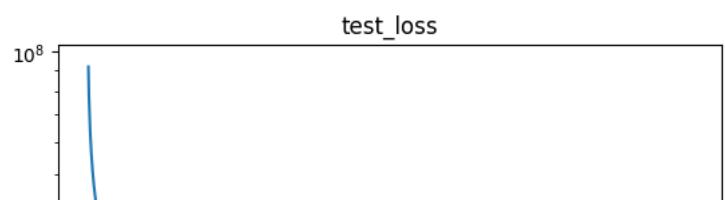


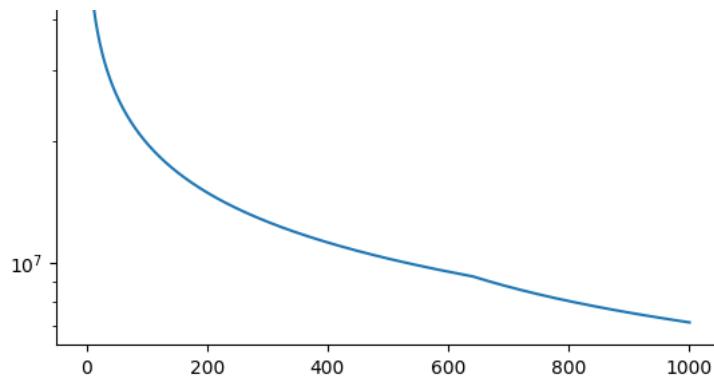


optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-05 , minimum_RMSE: 10731.15 , epoch: 1000 , test_loss: 12114 , train_loss:

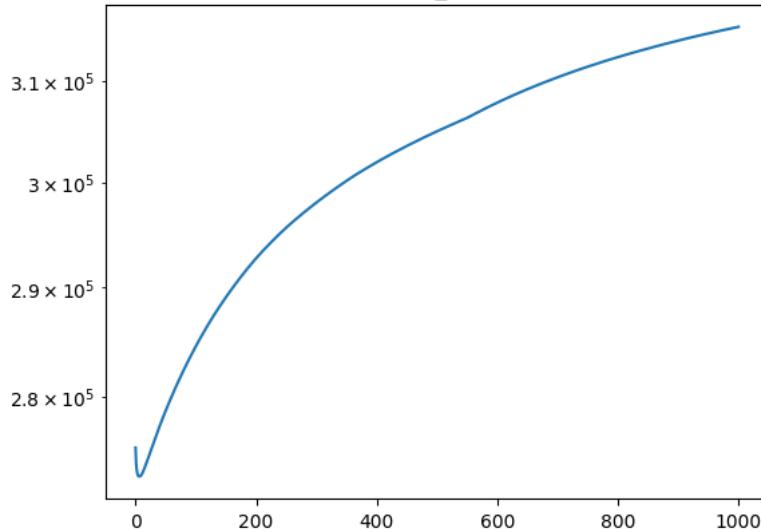


optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 5e-05 , minimum_RMSE: 2668.87 , epoch: 1000 , test_loss: 2668 , train_loss:

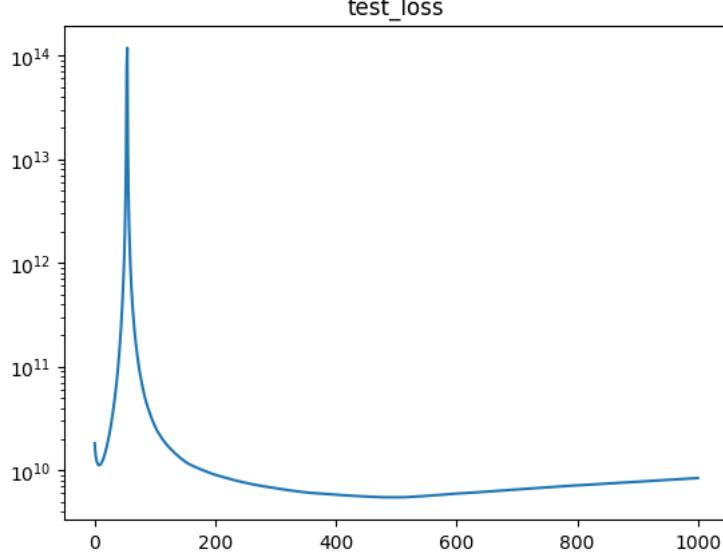




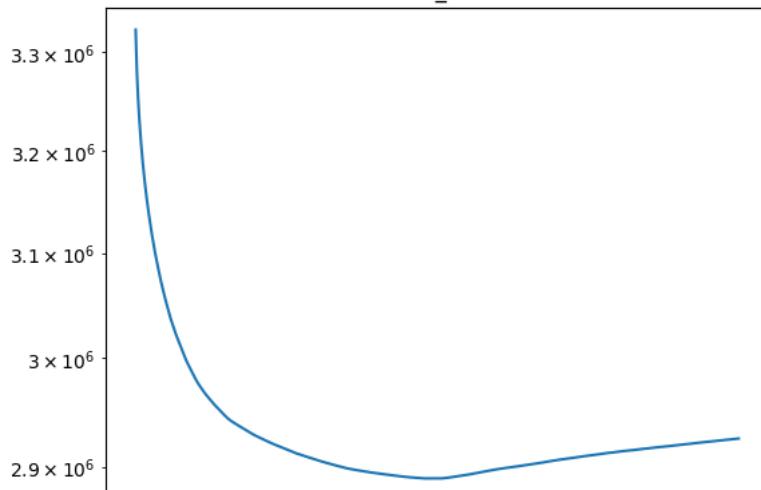
train_loss



train_loss

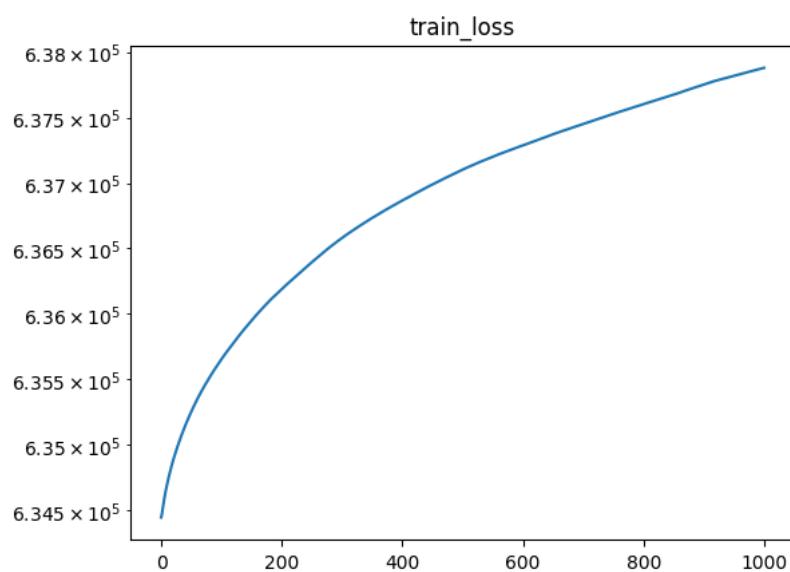
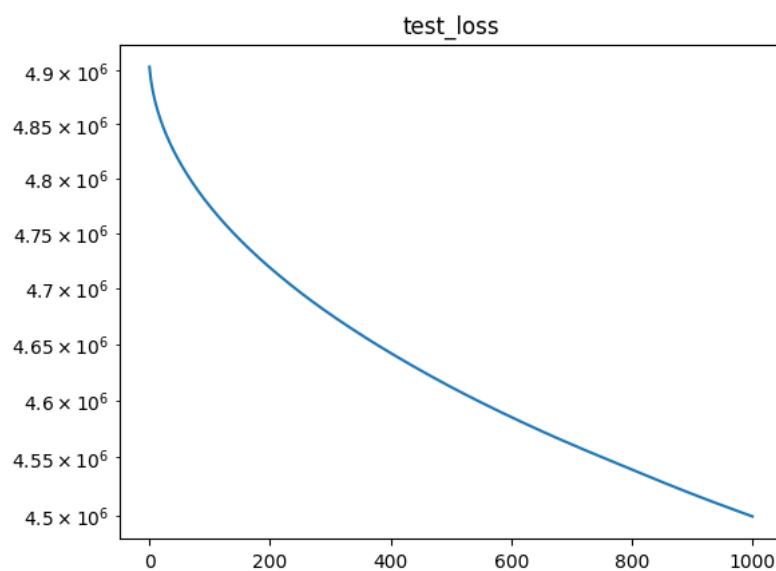


test_loss

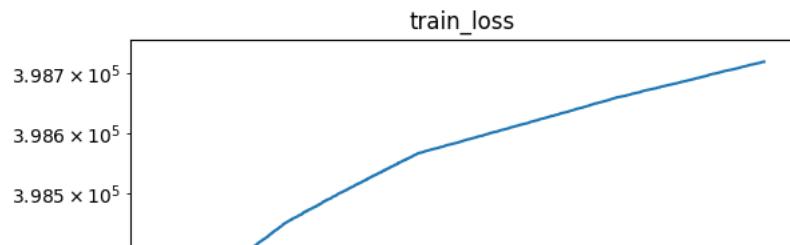
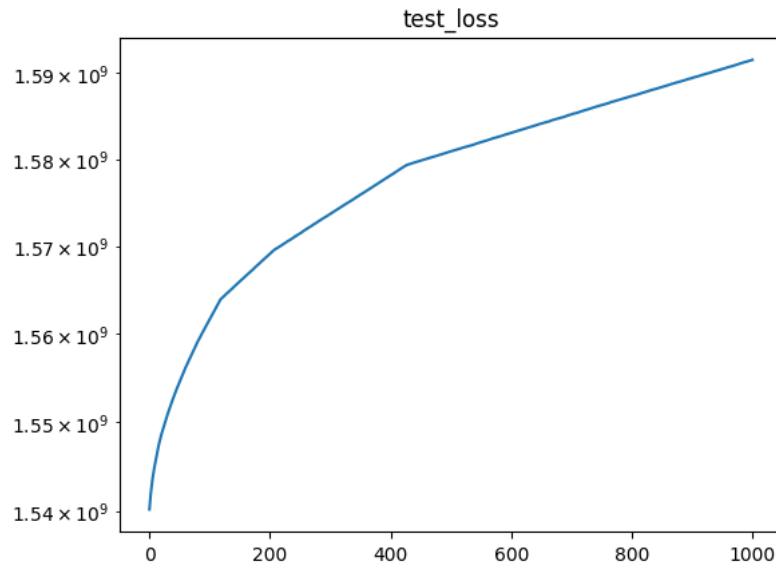


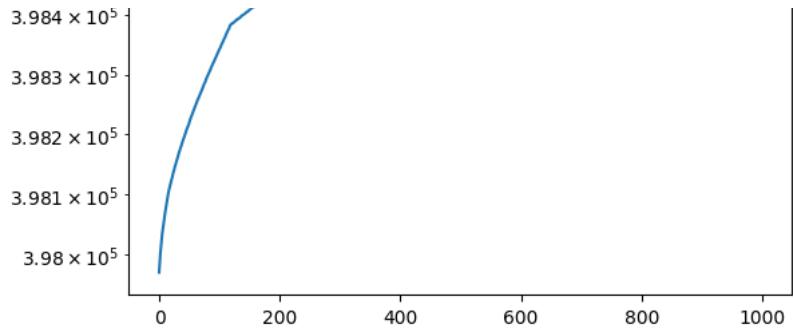
train_loss

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

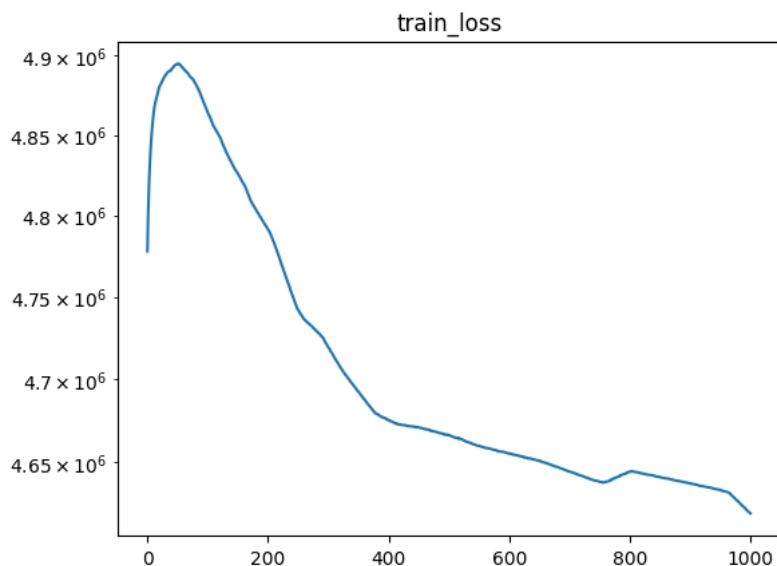
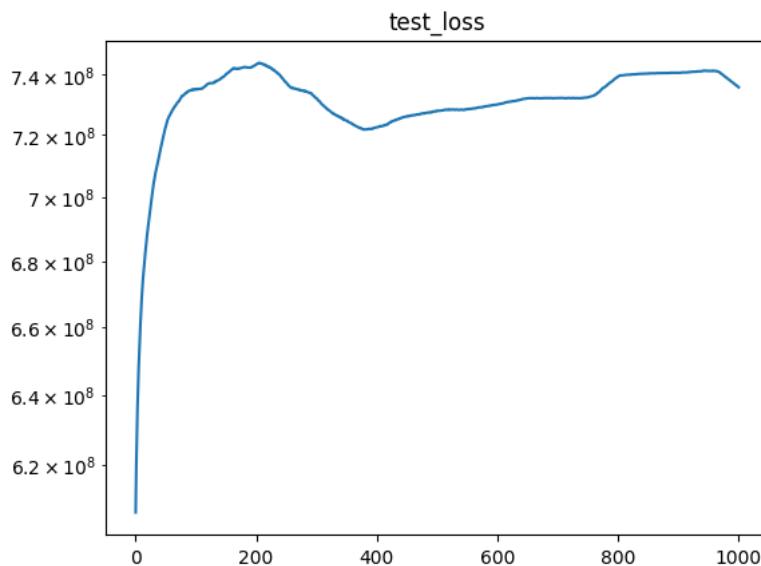


optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-07 , minimum_RMSE: 39245.16 , epoch: 1000 , test_loss: 39893 , train_loss:

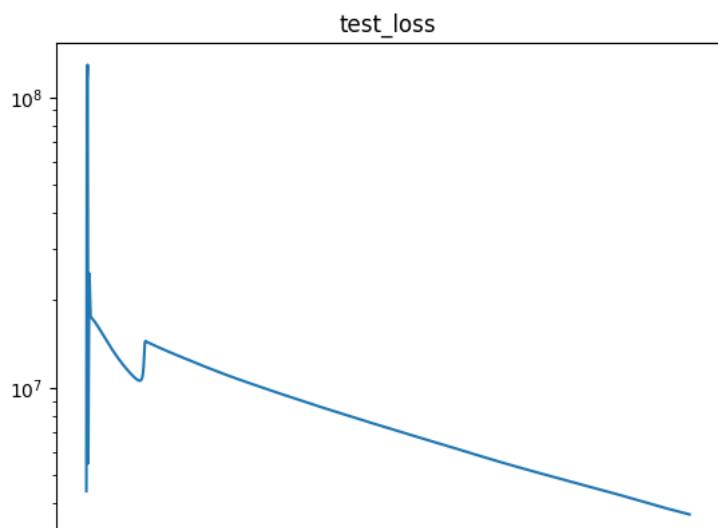


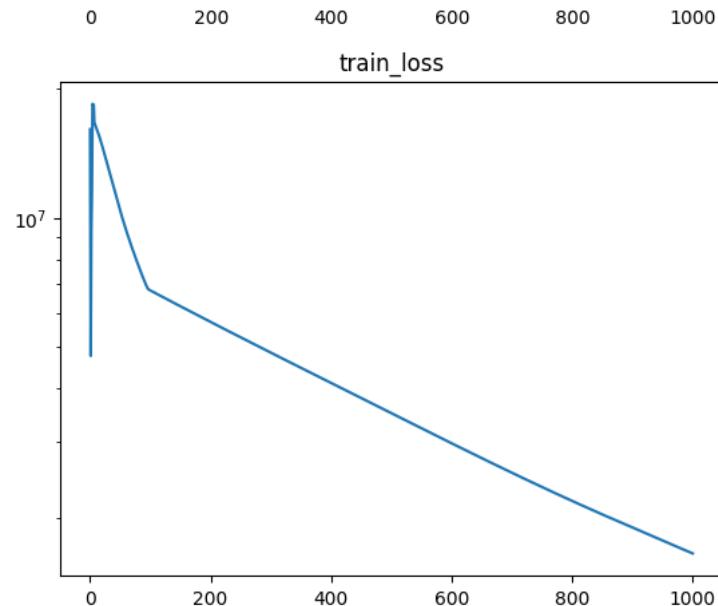


optimizer: Adagrad , n_of_data: 1000 , Bsize: 50 , learningRate: 5e-07 , minimum_RMSE: 24632.42 , epoch: 1000 , test_loss: 27124 , train_loss:

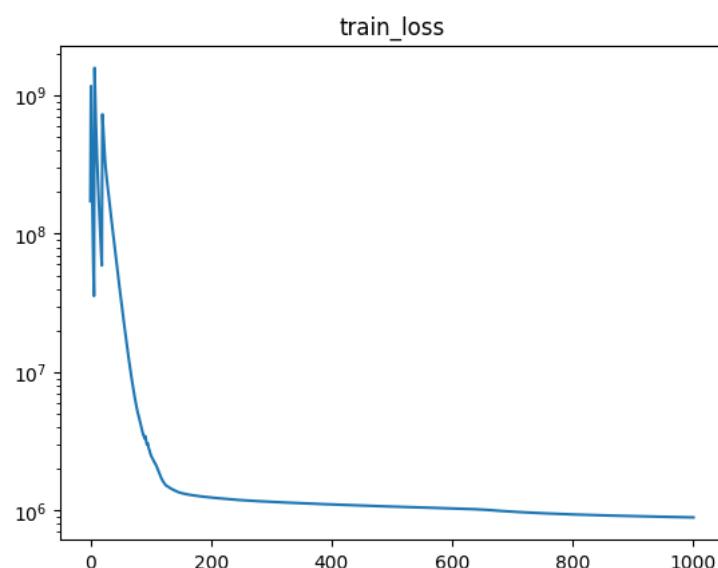
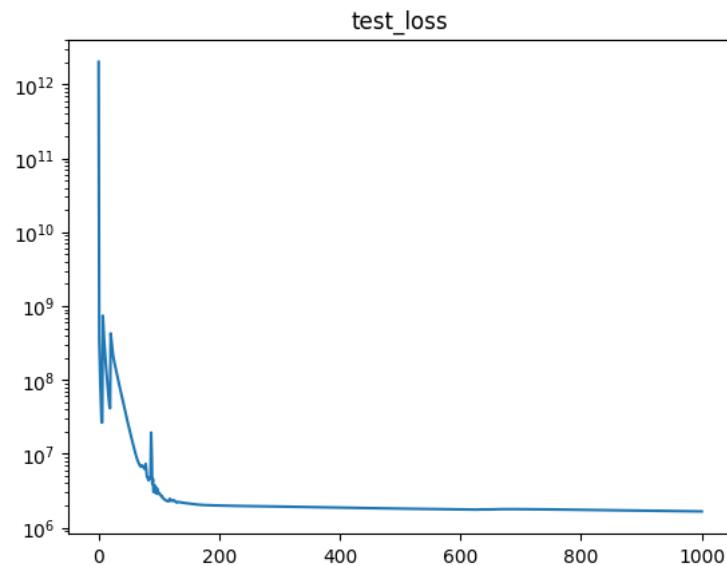


optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 0.1 , minimum_RMSE: 1912.32 , epoch: 1000 , test_loss: 1912 , train_loss:

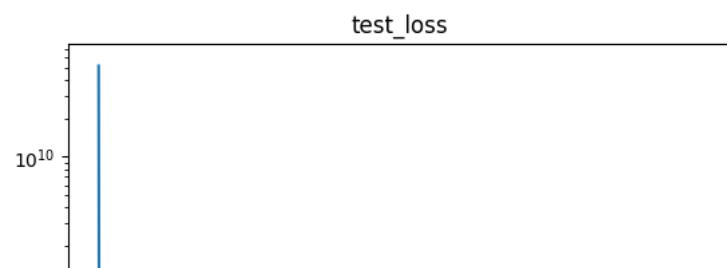


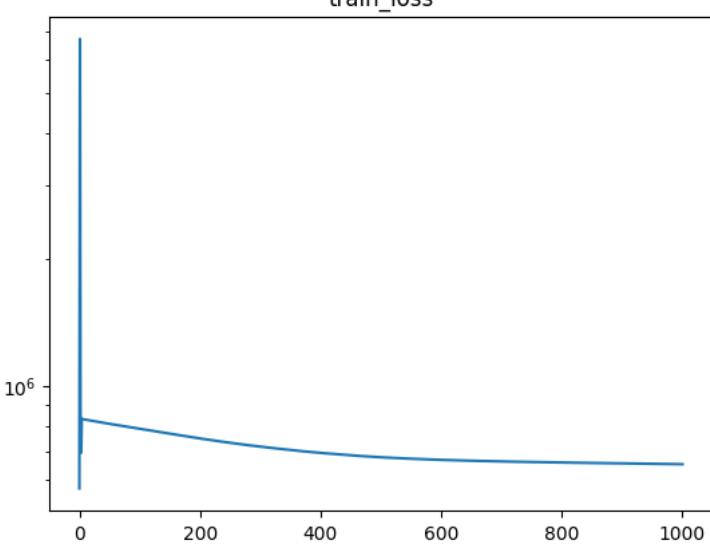
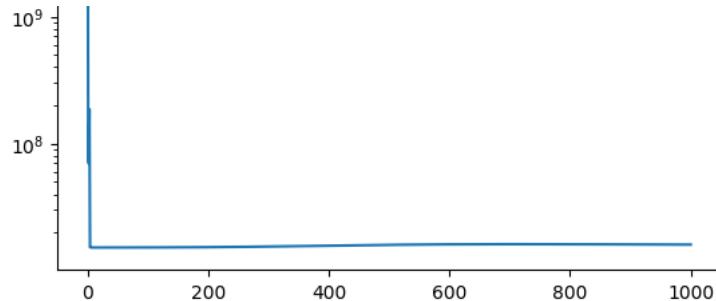


optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 0.5 , minimum_RMSE: 1285.88 , epoch: 1000 , test_loss: 1285 , train_loss:

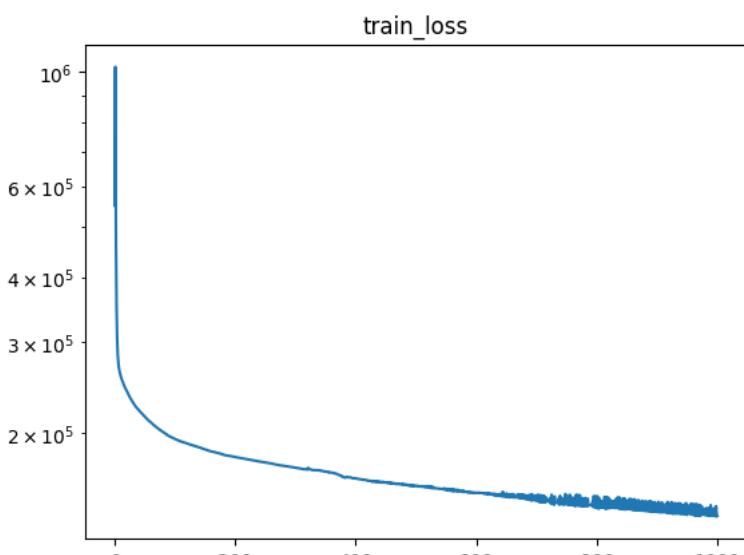
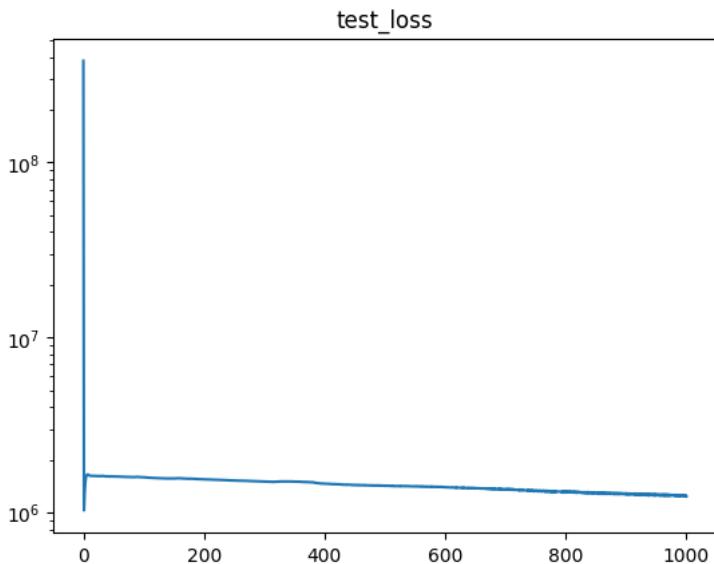


optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 0.01 , minimum_RMSE: 3899.51 , epoch: 1000 , test_loss: 4003 , train_loss:

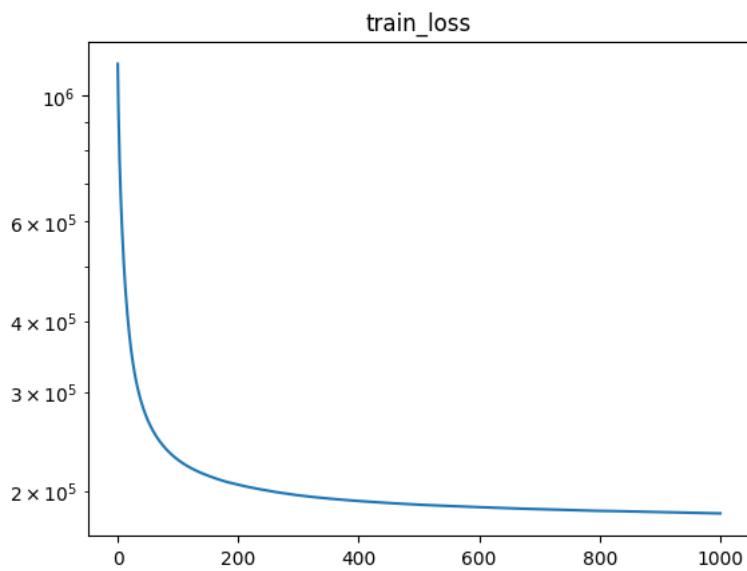
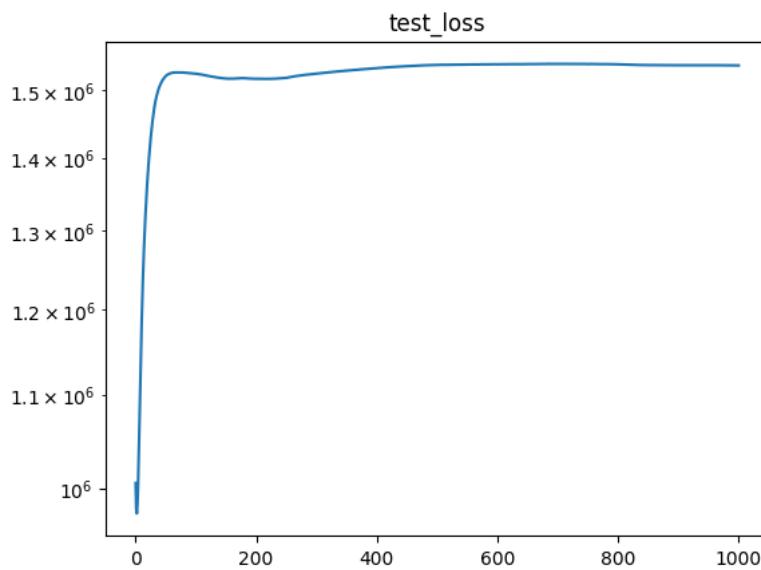




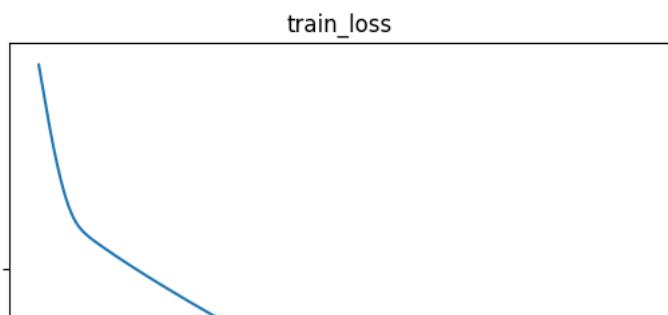
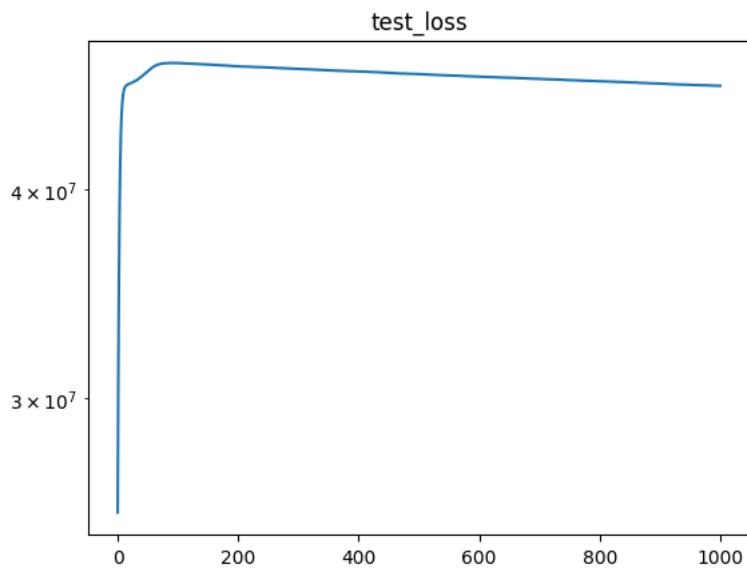
optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 0.05 , minimum_RMSE: 1012.57 , epoch: 1000 , test_loss: 1111 , train_loss

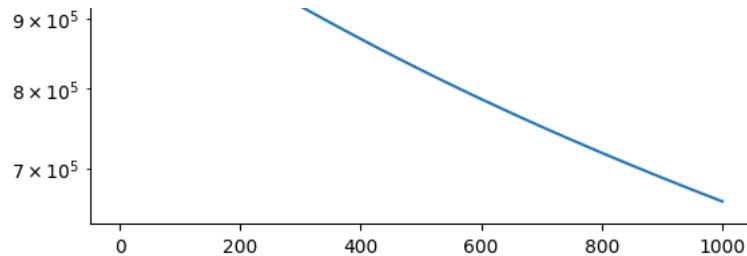


optimizer: Adadrad . n of data: 1000 . Bsize: 100 . learningRate: 0.001 . minimum RMSE: 987.49 . epoch: 1000 . test loss: 1240 . train loss

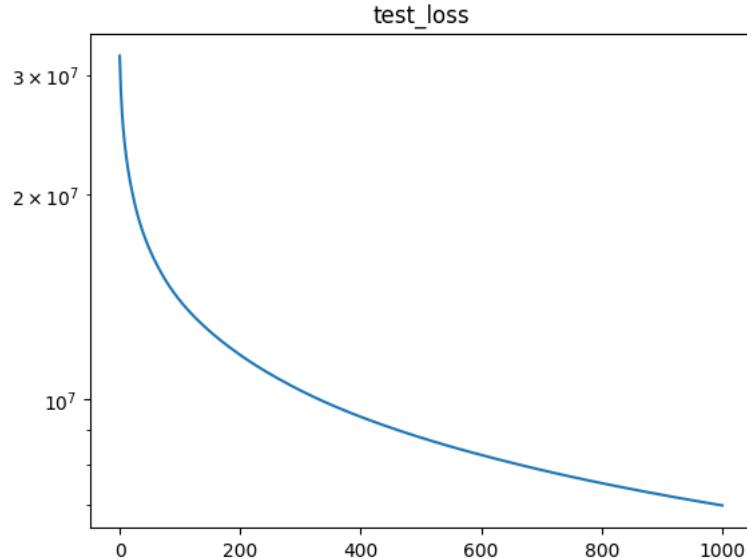


optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 0.005 , minimum_RMSE: 5063.07 , epoch: 1000 , test_loss: 6792 , train_loss:

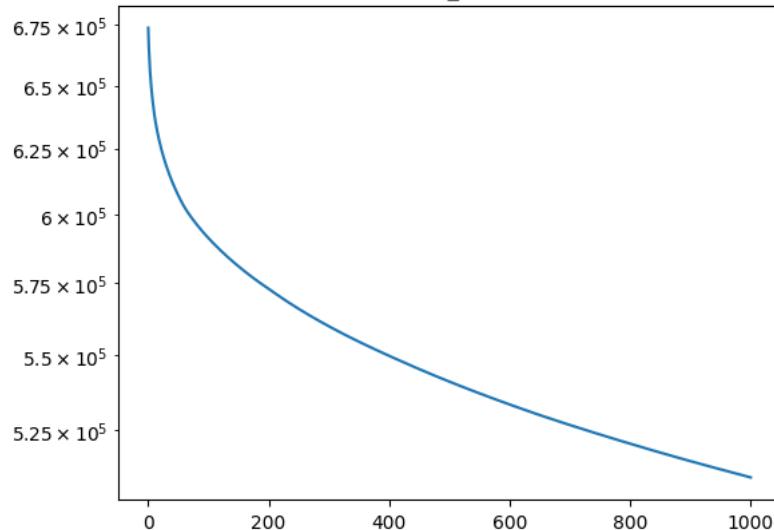




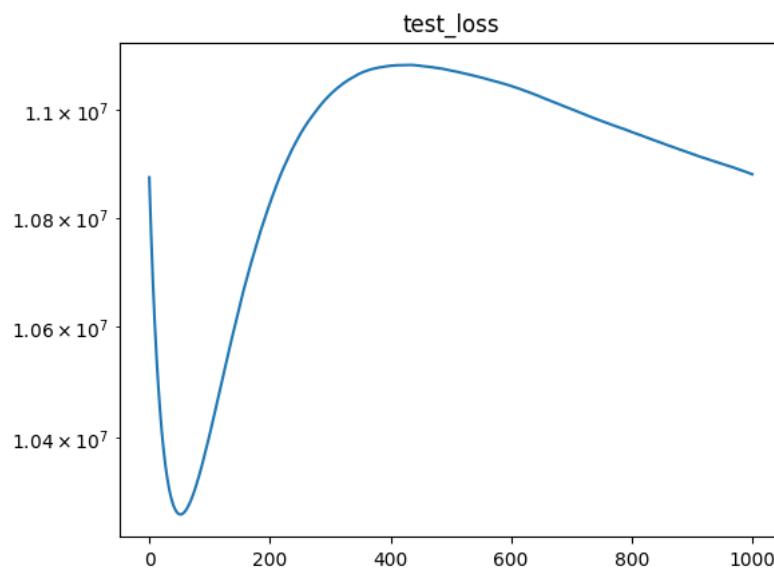
optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 0.0001 , minimum_RMSE: 2641.75 , epoch: 1000 , test_loss: 2641 , train_loss:



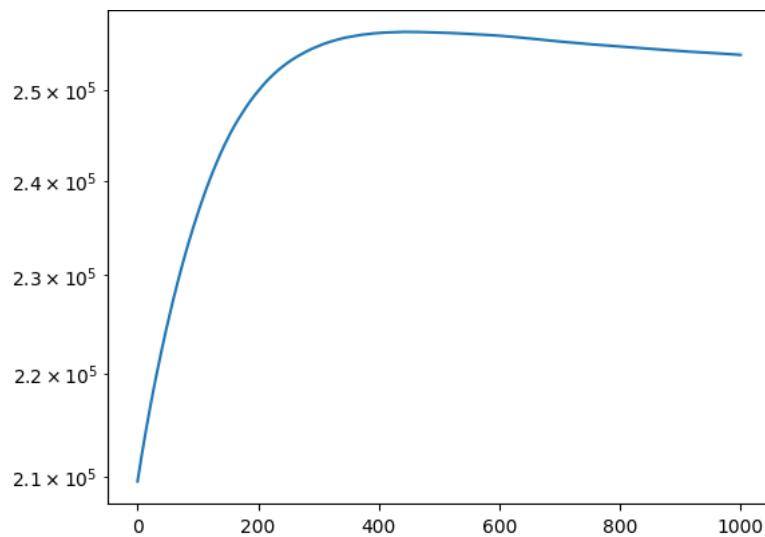
train_loss



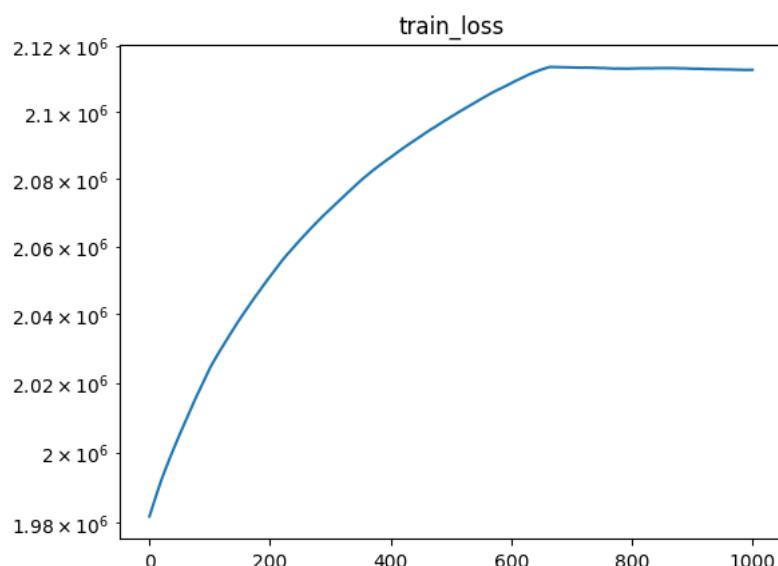
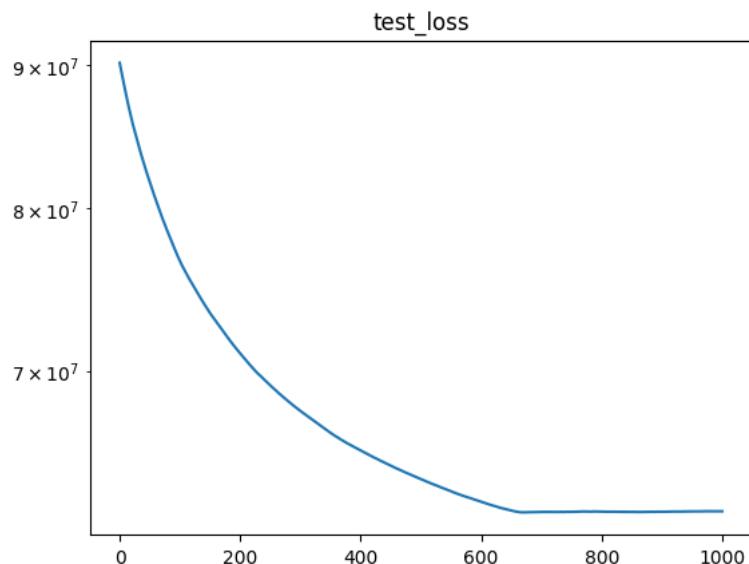
optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 0.0005 , minimum_RMSE: 3203.92 , epoch: 1000 , test_loss: 3298 , train_loss:



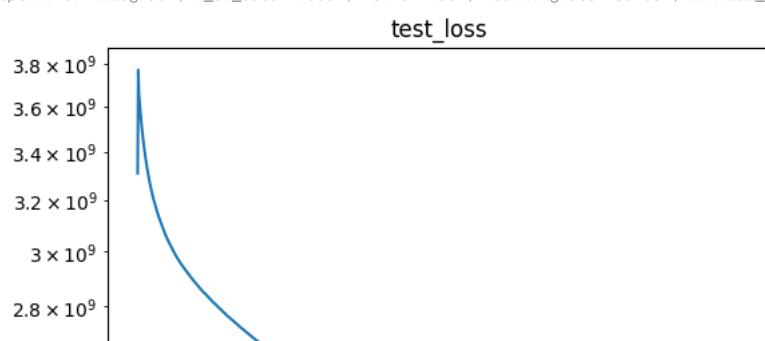
train_loss

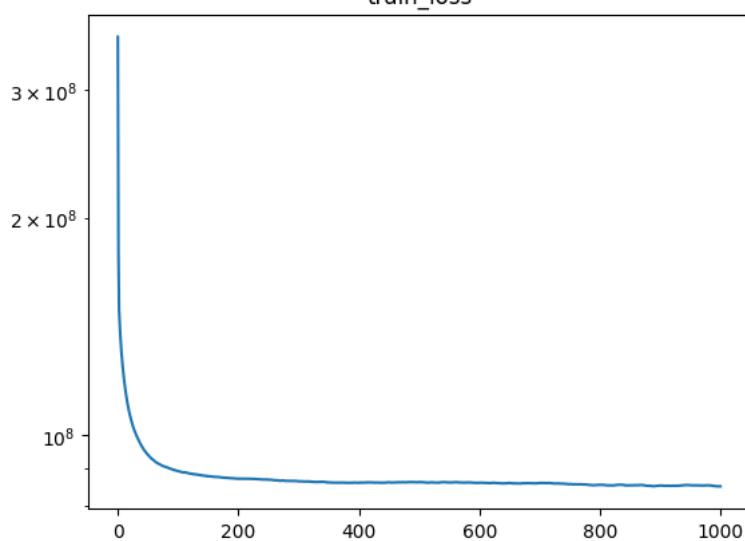
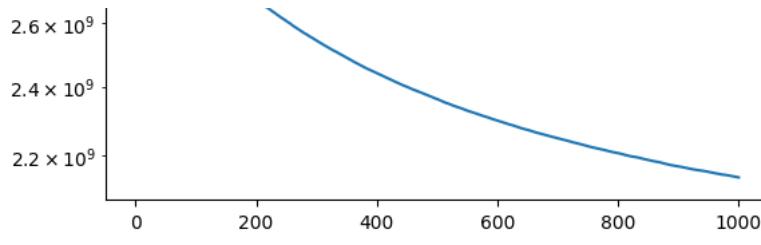


optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-05 , minimum_RMSE: 7893.49 , epoch: 1000 , test_loss: 7897 , train_loss:

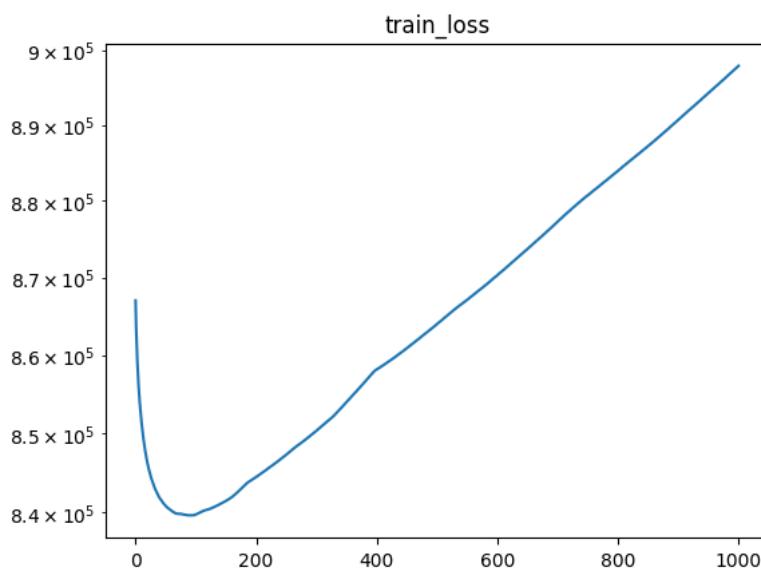
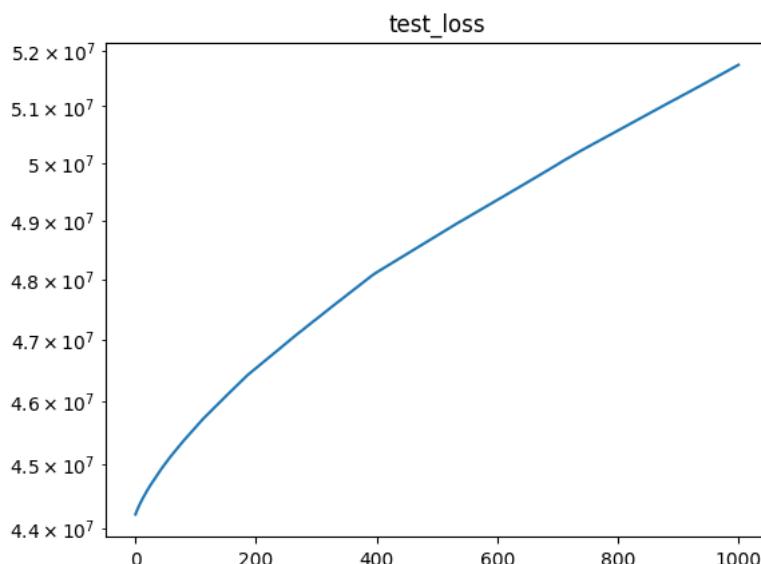


optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-05 , minimum_RMSE: 46276.57 , epoch: 1000 , test_loss: 46276 , train_loss:



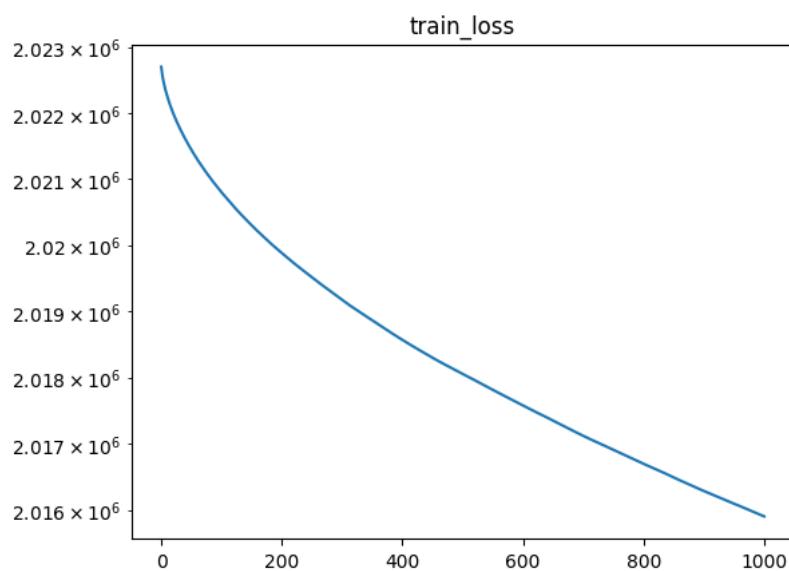
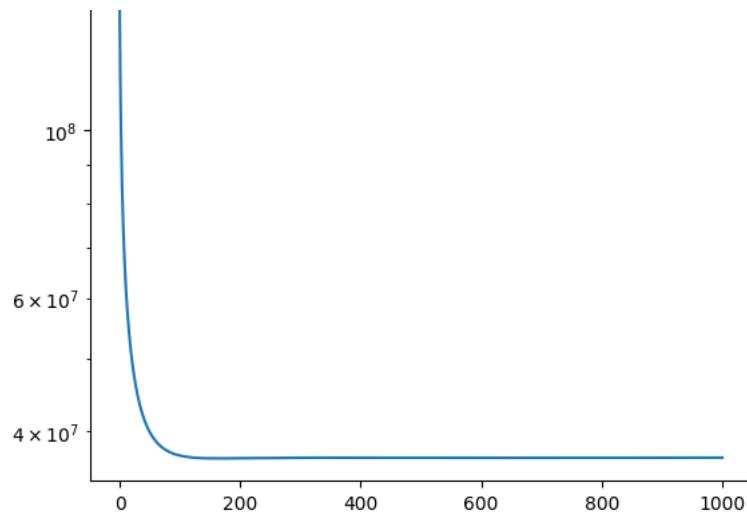


optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-06 , minimum_RMSE: 6649.63 , epoch: 1000 , test_loss: 7192 , train_loss:

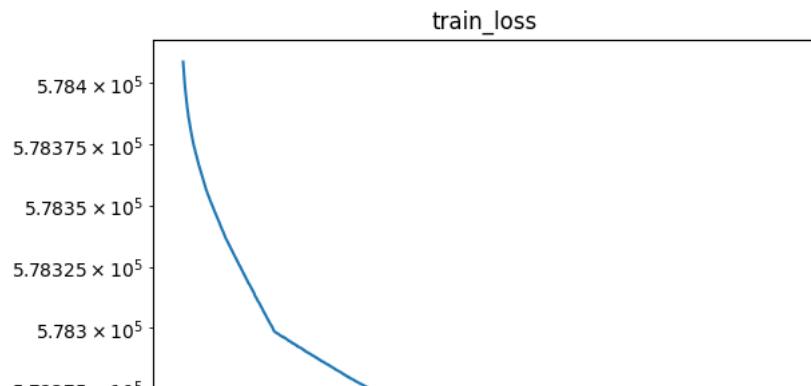
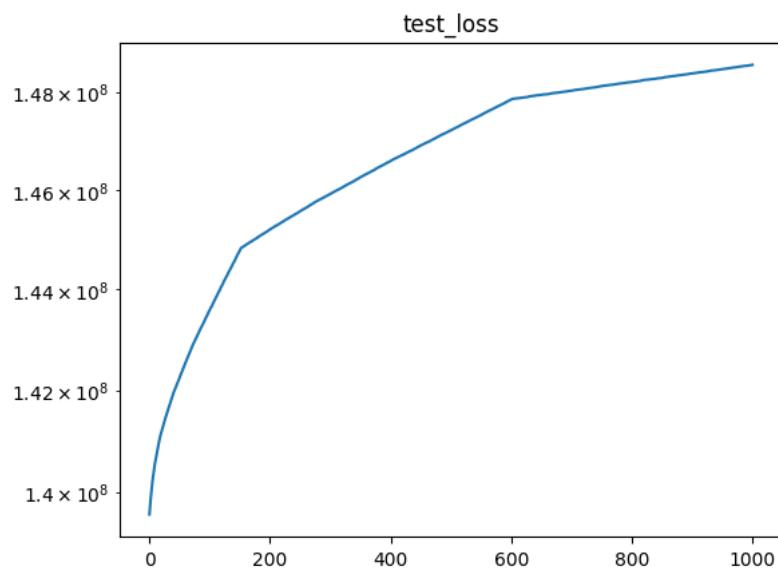


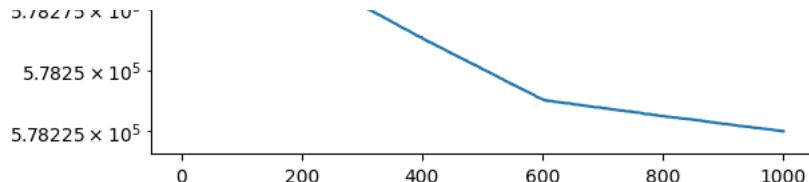
optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-06 , minimum_RMSE: 6070.40 , epoch: 1000 , test_loss: 6077 , train_loss:



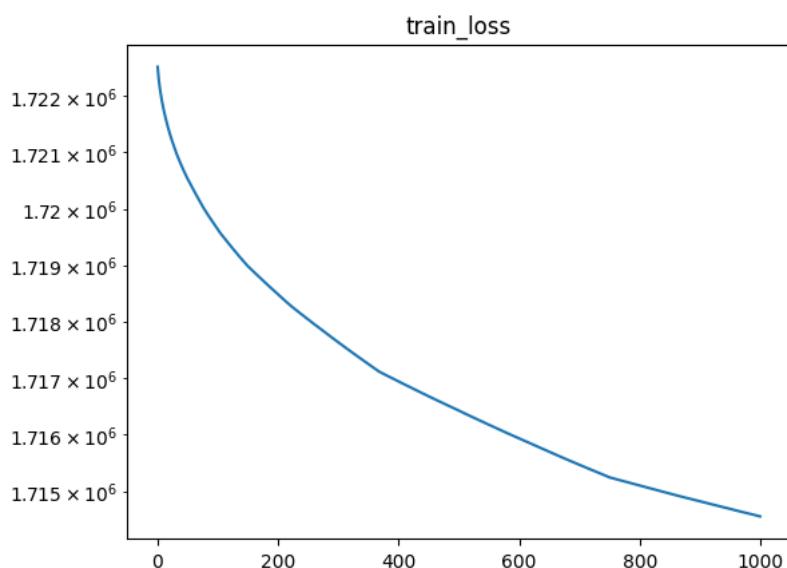
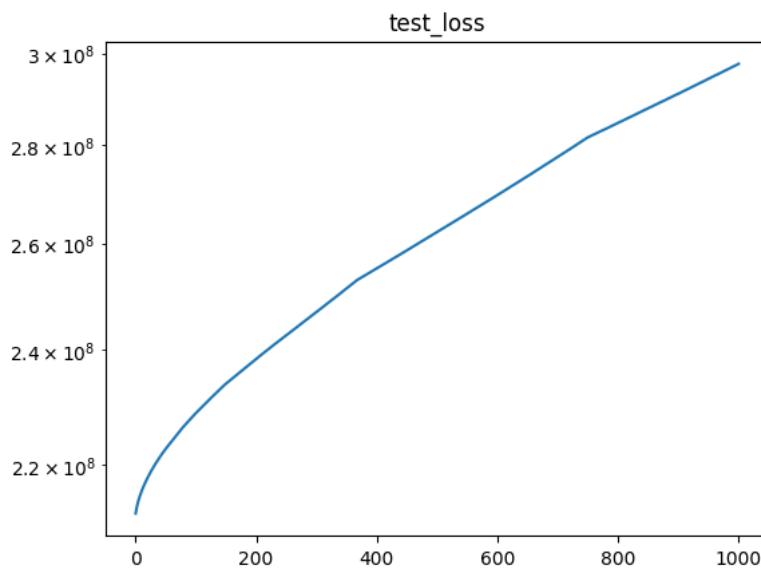


optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-07 , minimum_RMSE: 11813.93 , epoch: 1000 , test_loss: 12188 , train_loss:

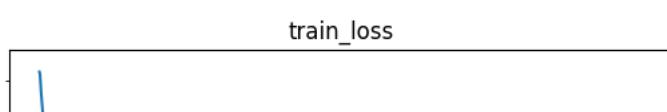
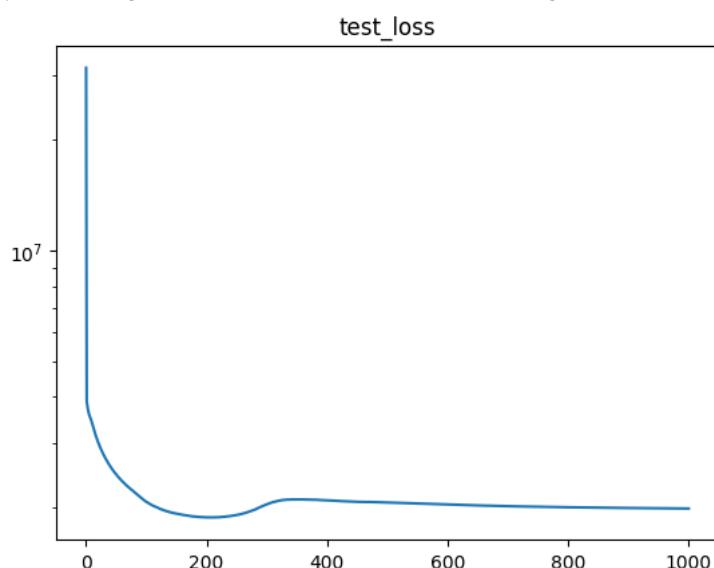


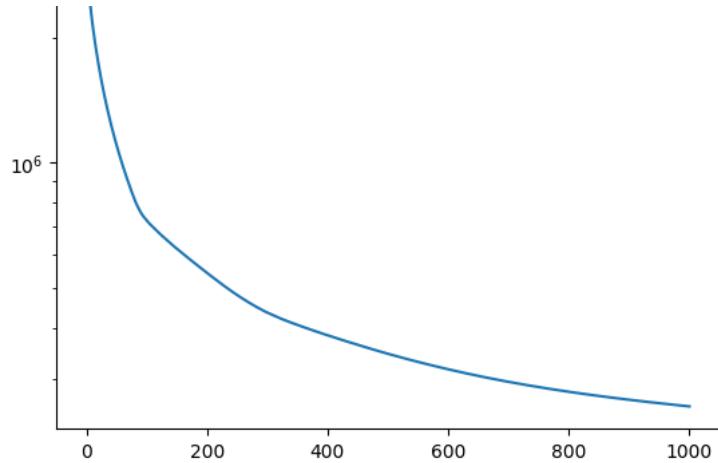


optimizer: Adagrad , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-07 , minimum_RMSE: 14565.80 , epoch: 1000 , test_loss: 17251 , train_loss:



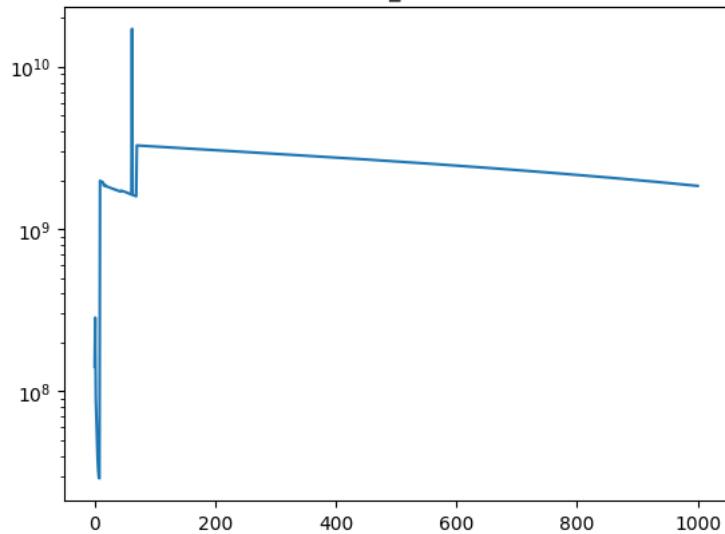
optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 0.1 , minimum_RMSE: 1371.31 , epoch: 1000 , test_loss: 1409 , train_loss:



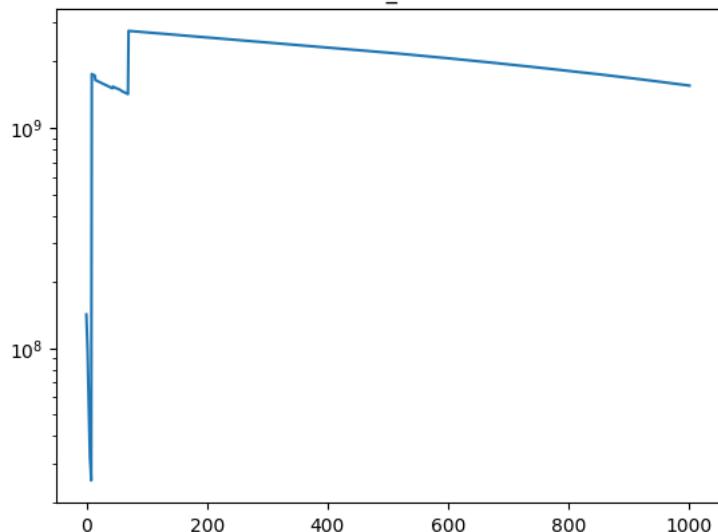


optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 0.5 , minimum_RMSE: 5386.56 , epoch: 1000 , test_loss: 42957 , train_loss:

test_loss

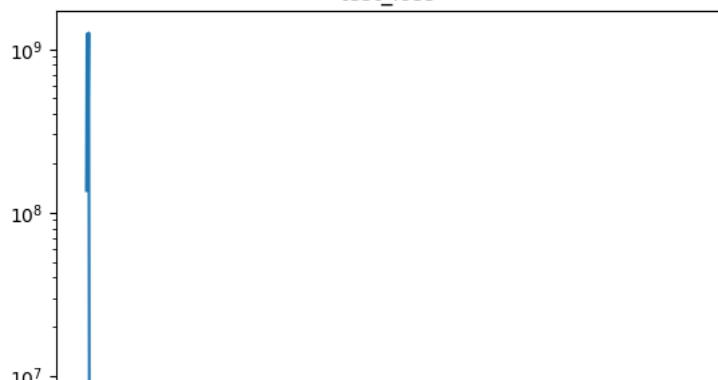


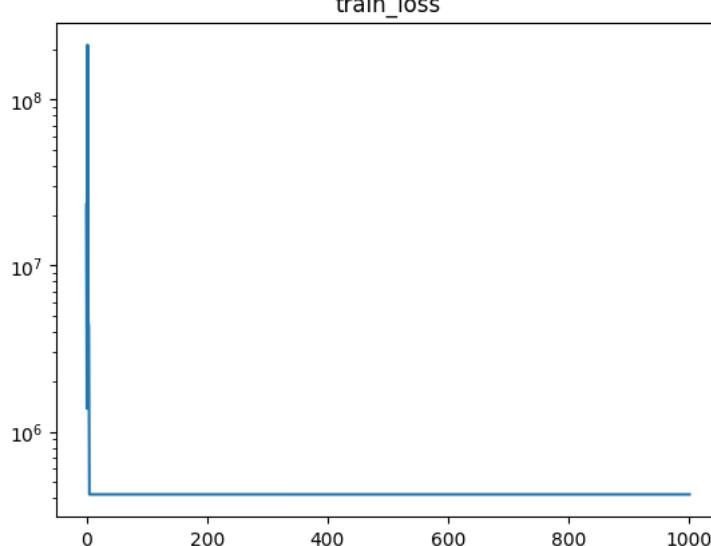
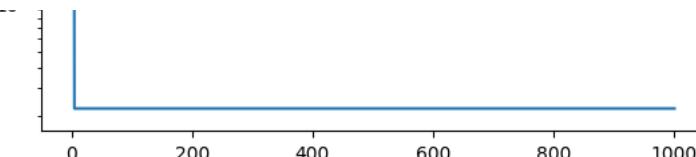
train_loss



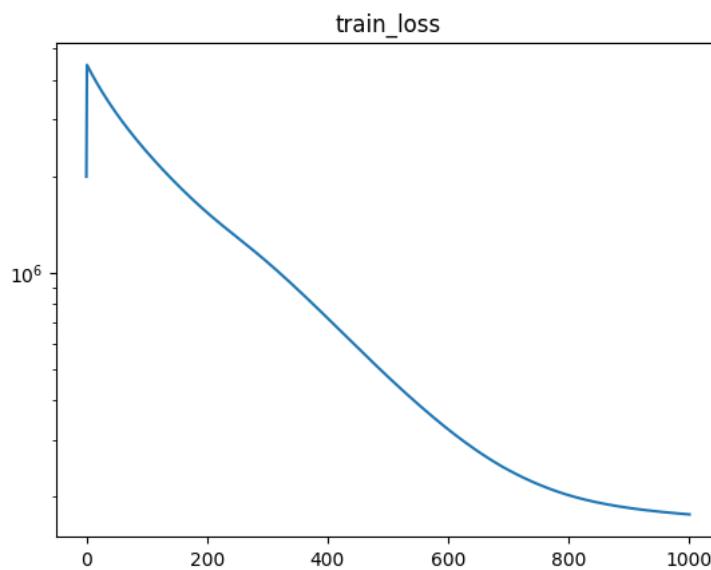
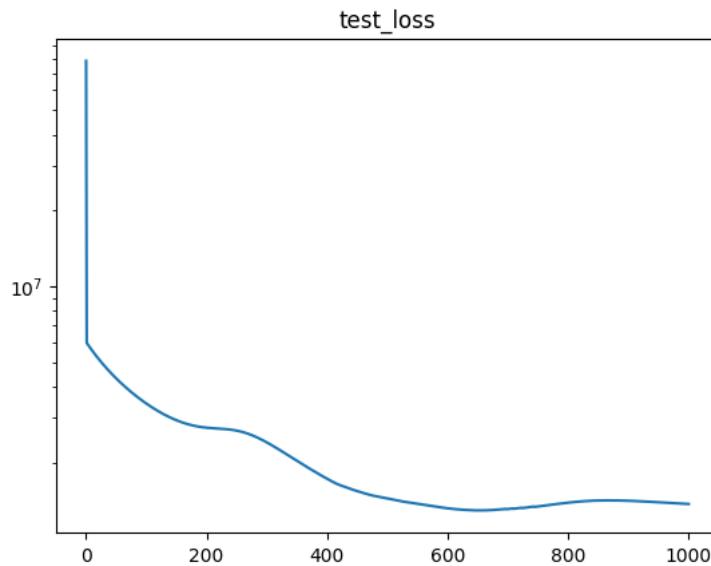
optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 0.01 , minimum_RMSE: 1493.77 , epoch: 1000 , test_loss: 1493 , train_loss:

test_loss

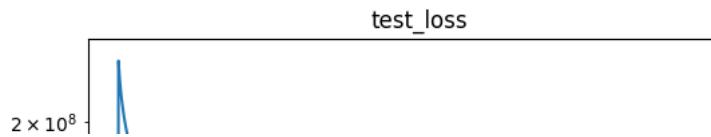


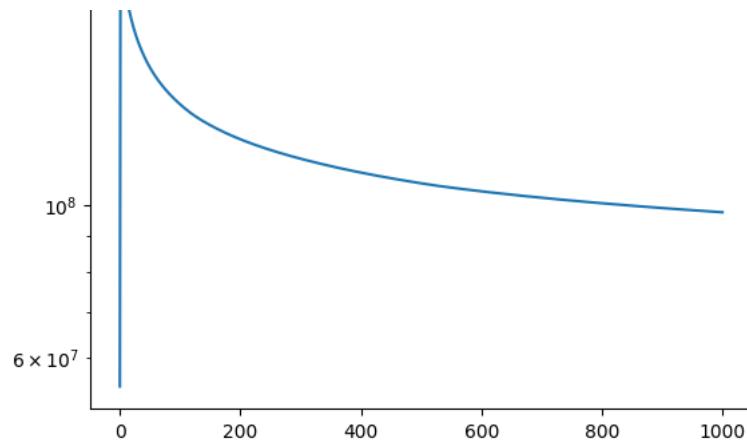


optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 0.05 , minimum_RMSE: 1135.50 , epoch: 1000 , test_loss: 1169 , train_loss:

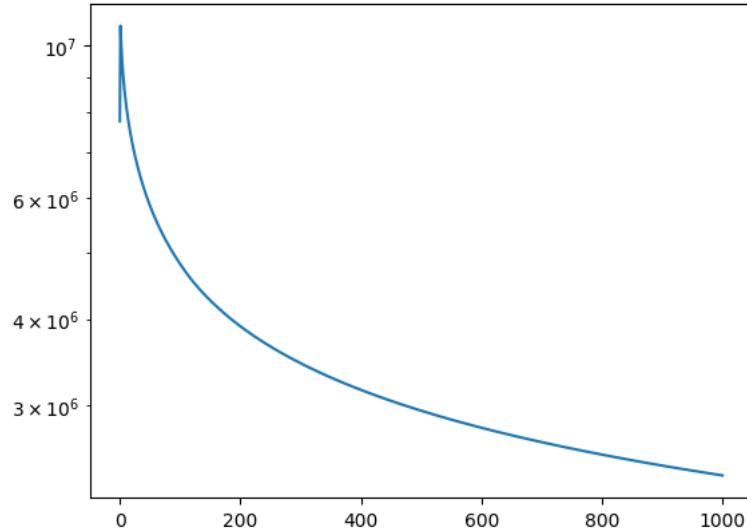


optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 0.001 , minimum_RMSE: 7383.28 , epoch: 1000 , test_loss: 9875 , train_loss:

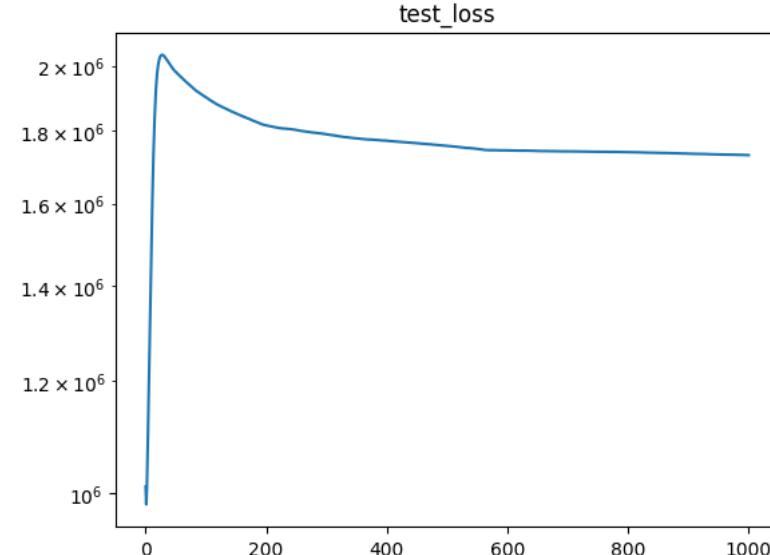




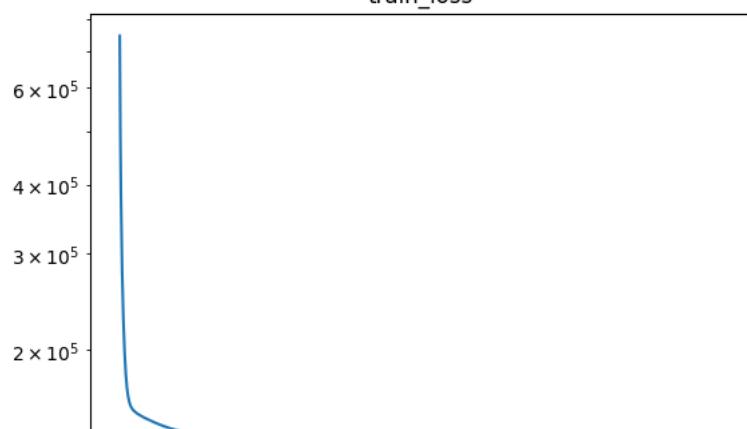
train_loss



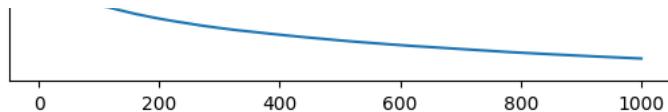
train_loss



test_loss

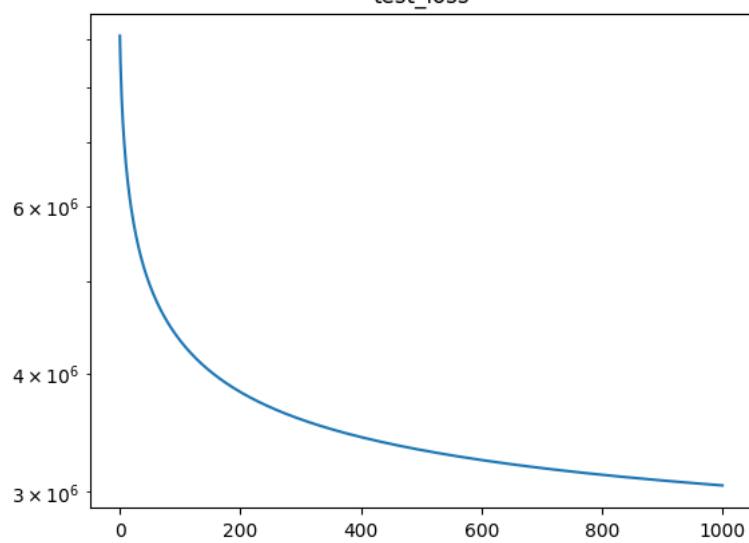


train_loss

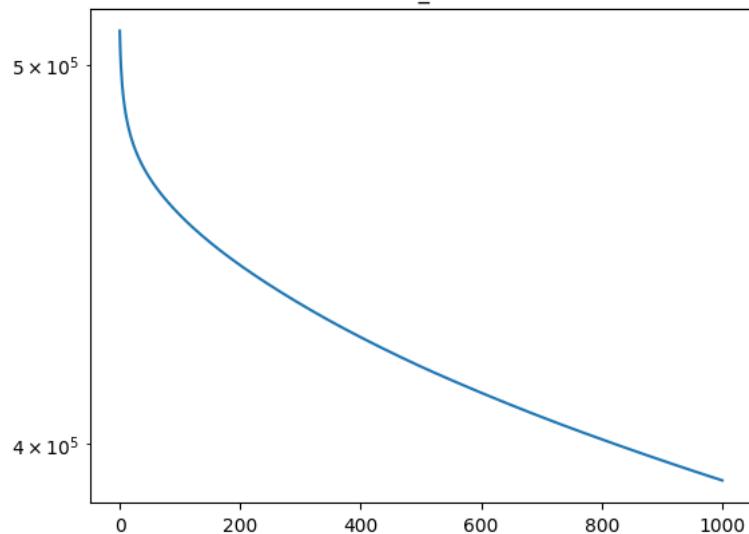


optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 0.0001 , minimum_RMSE: 1745.68 , epoch: 1000 , test_loss: 1745 , train_loss:

test_loss

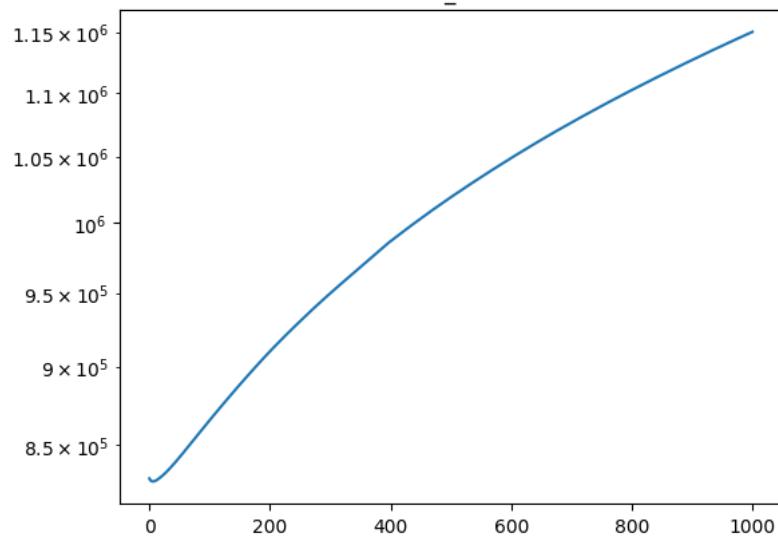


train_loss



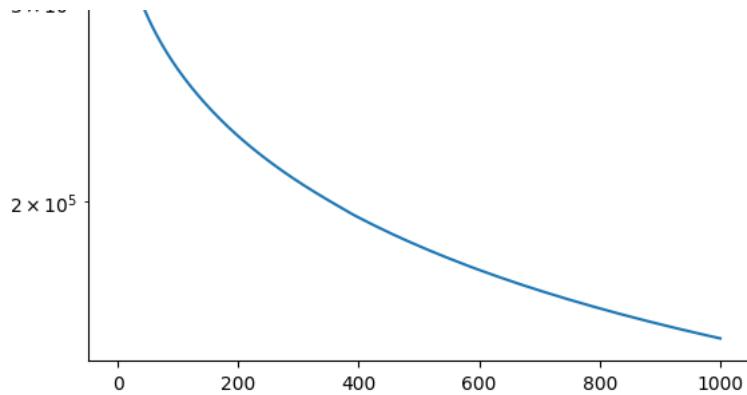
optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 0.0005 , minimum_RMSE: 909.75 , epoch: 1000 , test_loss: 1072 , train_loss:

test_loss

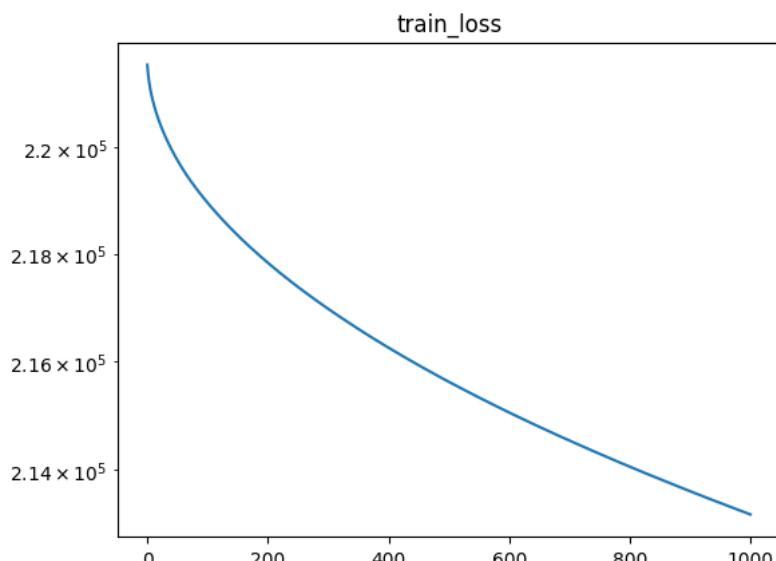
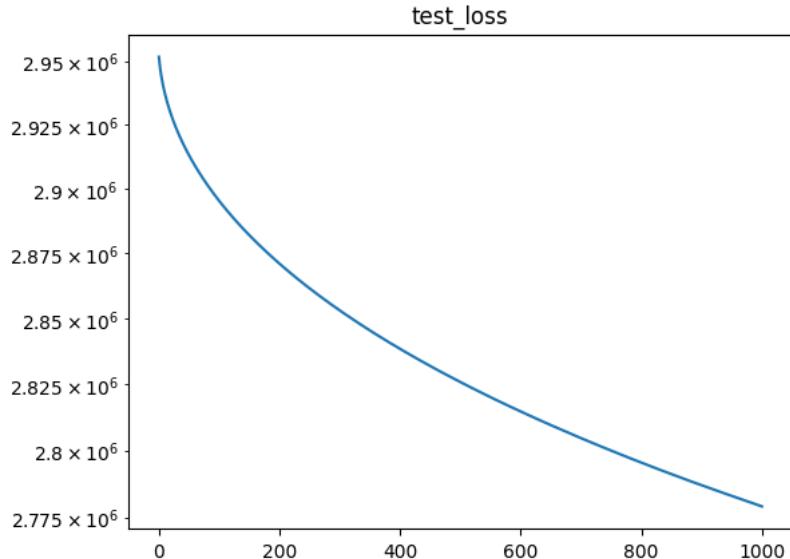


train_loss

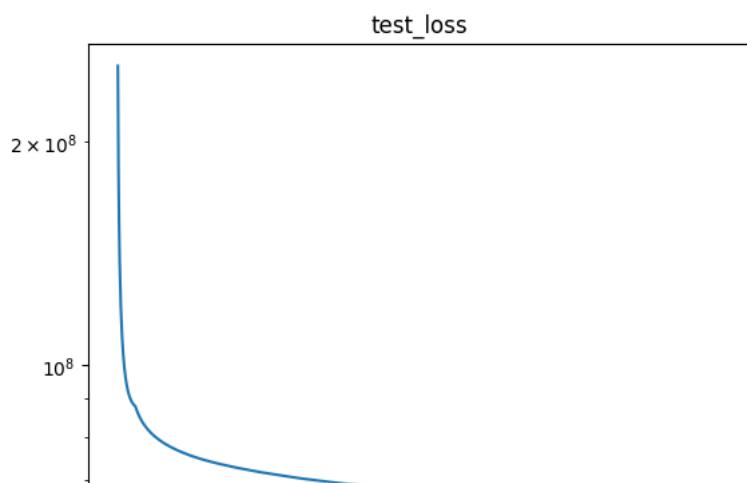


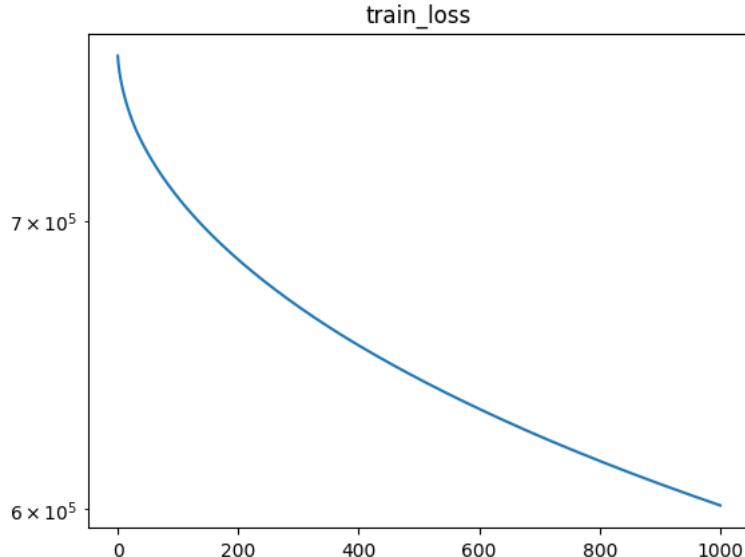
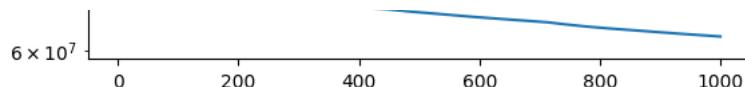


optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-05 , minimum_RMSE: 1667.08 , epoch: 1000 , test_loss: 1667 , train_loss:

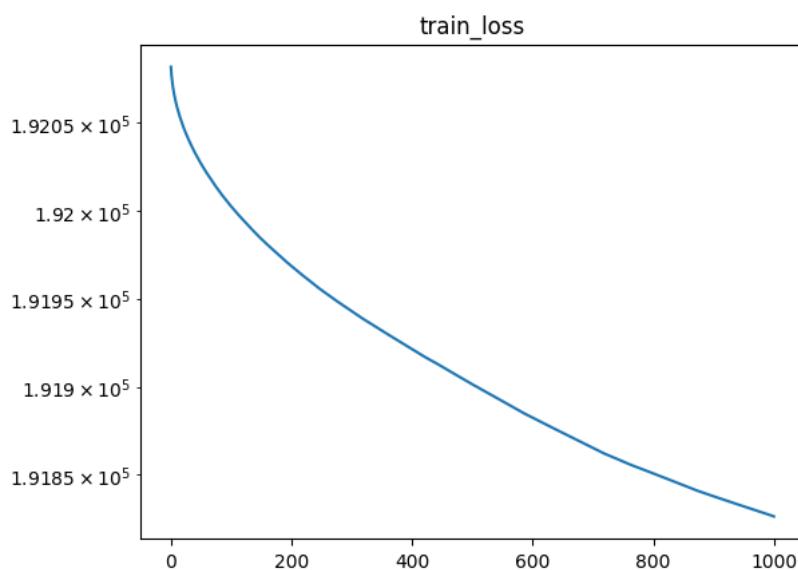
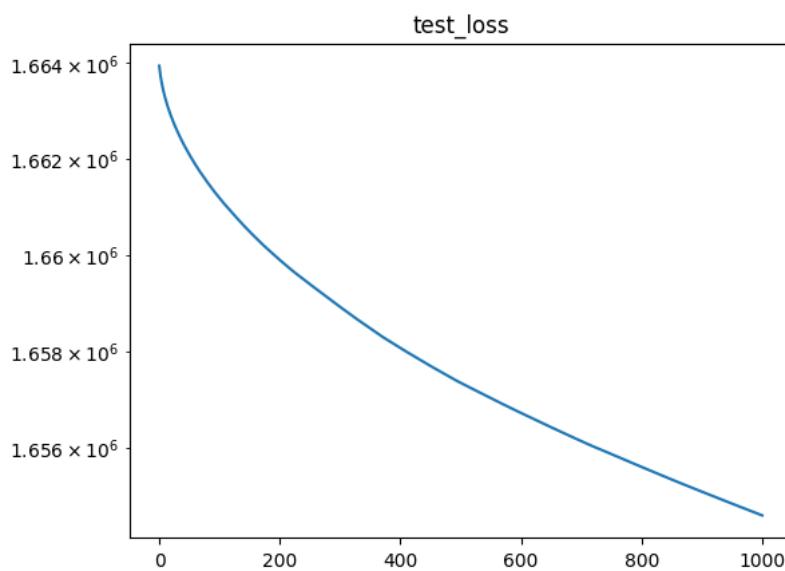


optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 5e-05 , minimum_RMSE: 7916.31 , epoch: 1000 , test_loss: 7916 , train_loss:

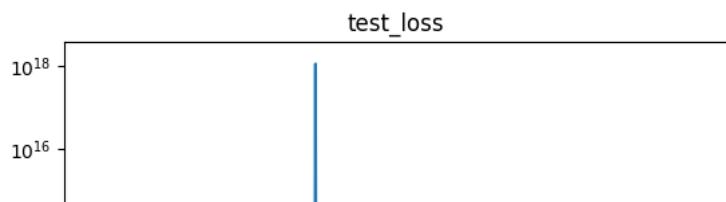


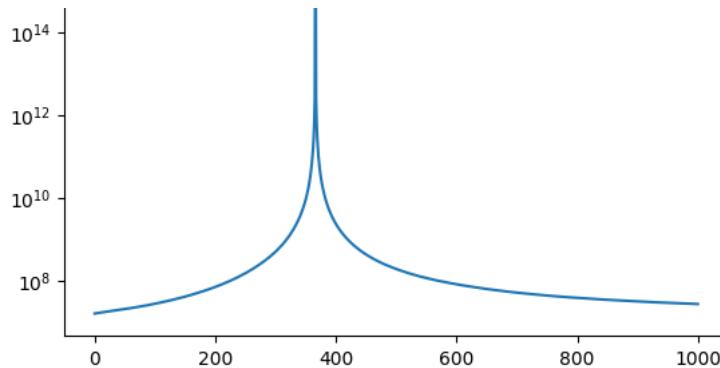


optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-06 , minimum_RMSE: 1286.31 , epoch: 1000 , test_loss: 1286 , train_loss:

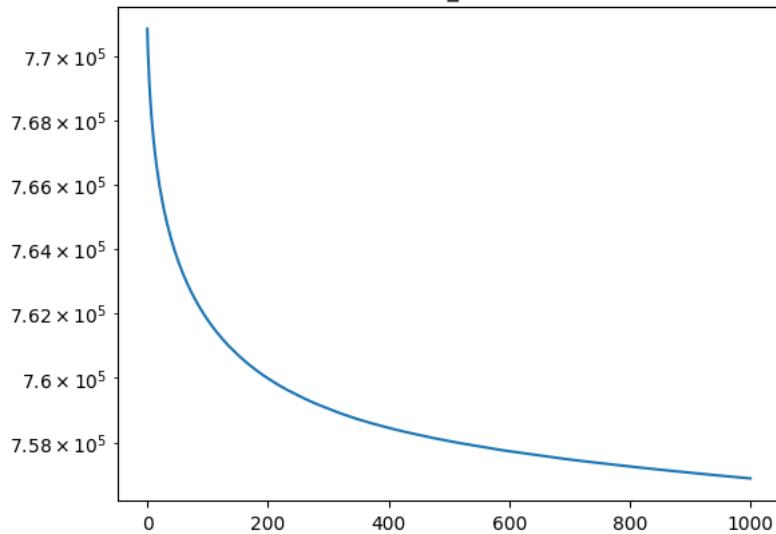


optimizer: Adagrad , n_of_data: 1000 , Bsize: 500 , learningRate: 5e-06 , minimum_RMSE: 4030.78 , epoch: 1000 , test_loss: 5249 , train_loss:

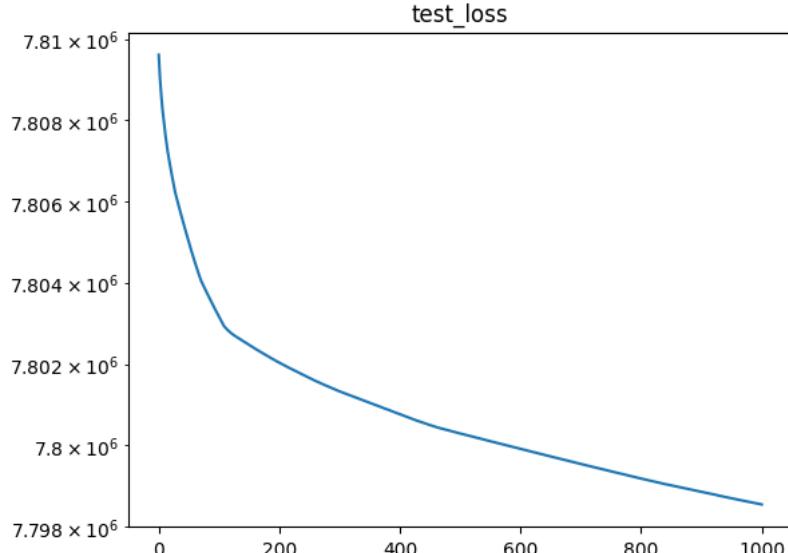




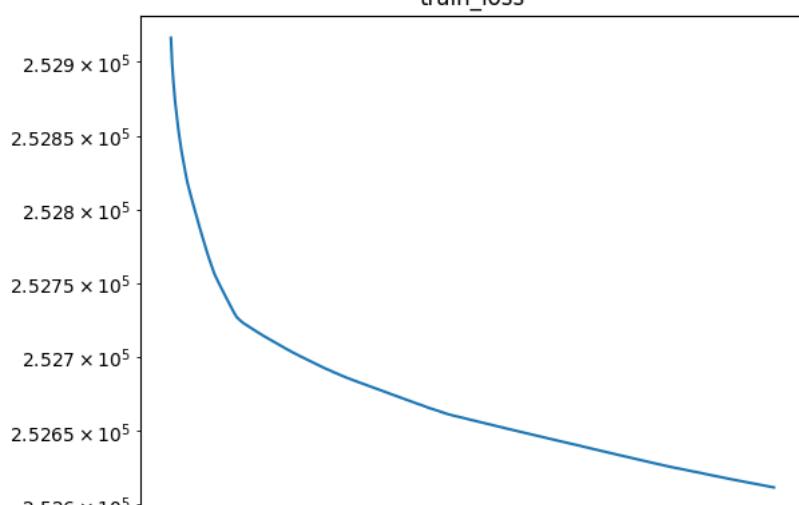
train_loss



train_loss



test_loss

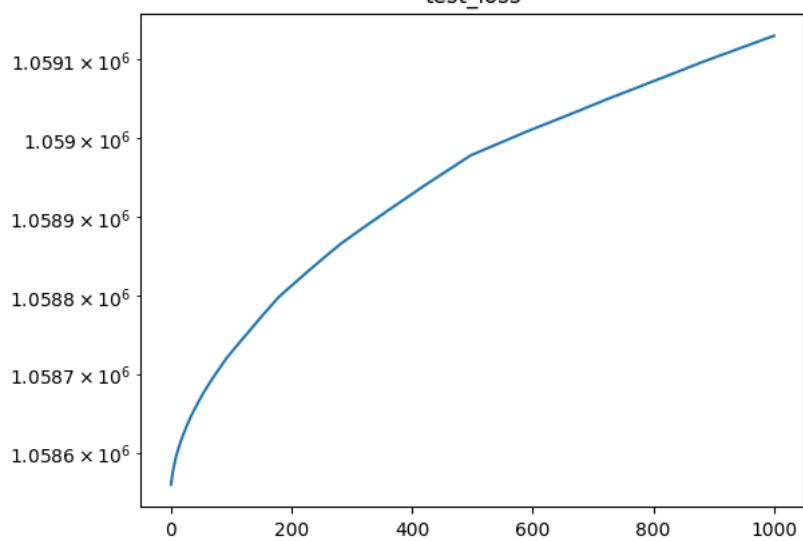


train_loss

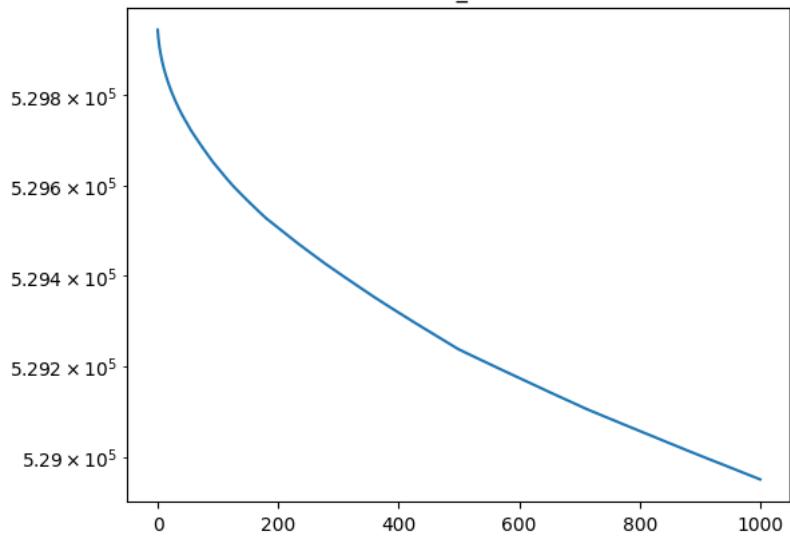
...
...

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

test_loss

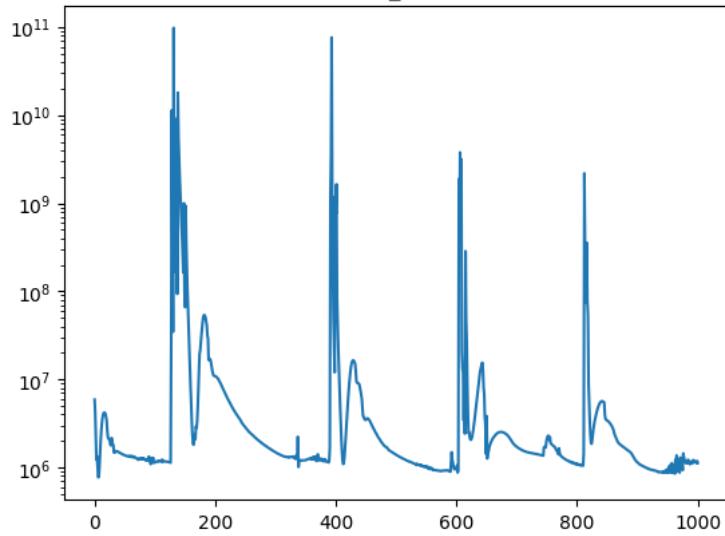


train_loss

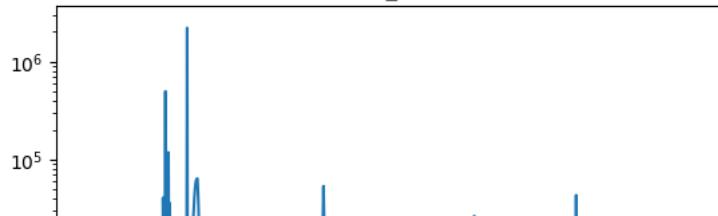


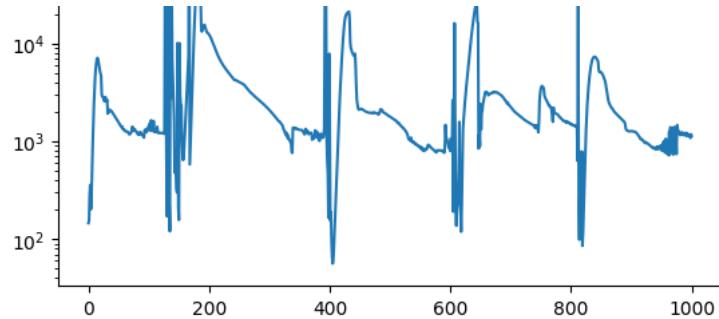
optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 0.1 , minimum_RMSE: 879.14 , epoch: 1000 , test_loss: 1061 , train_loss: 1061

test_loss

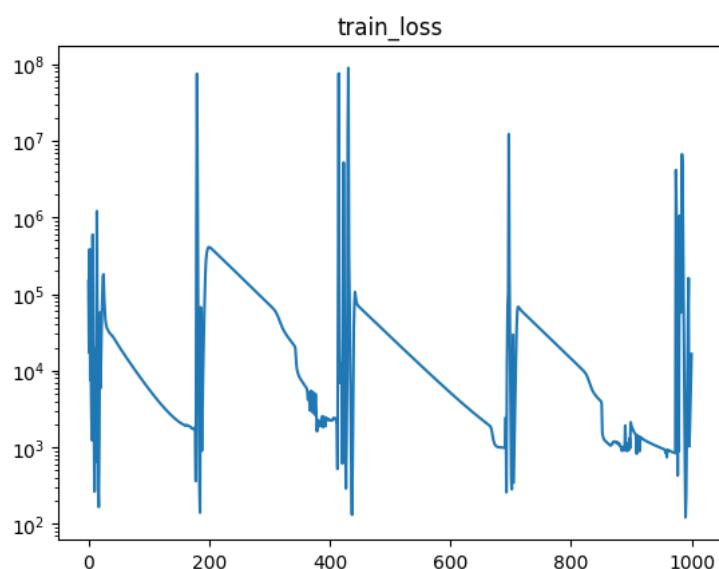
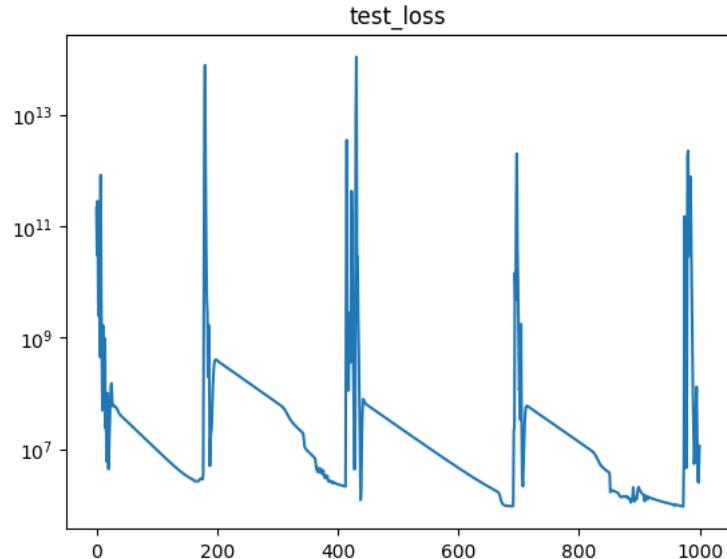


train_loss

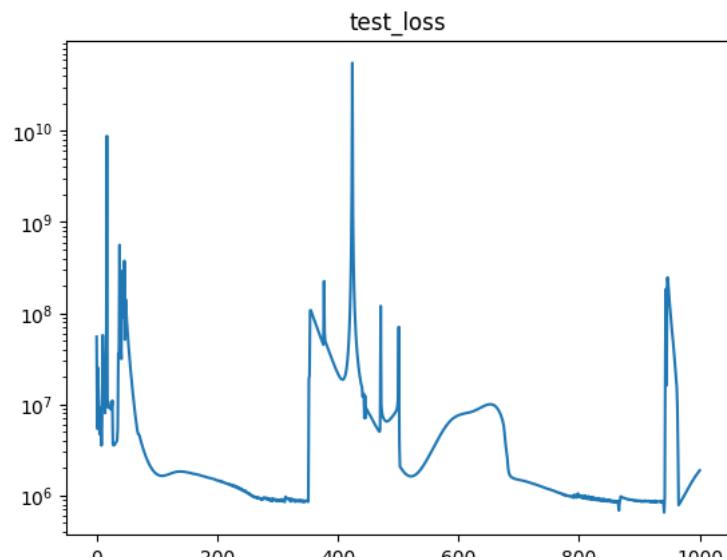


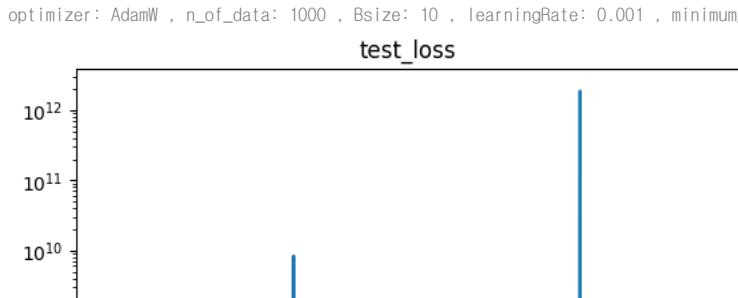
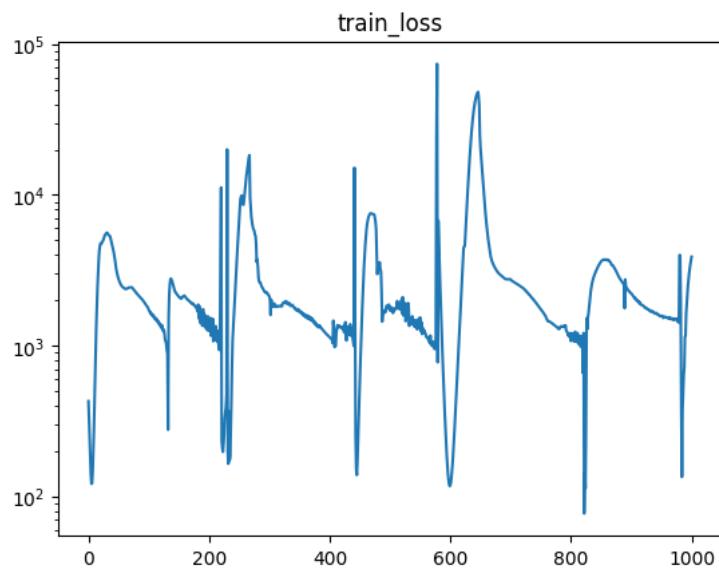
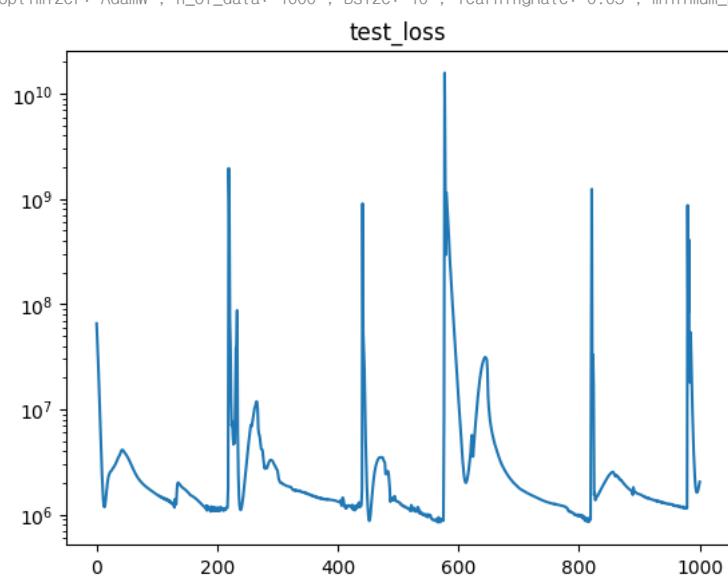
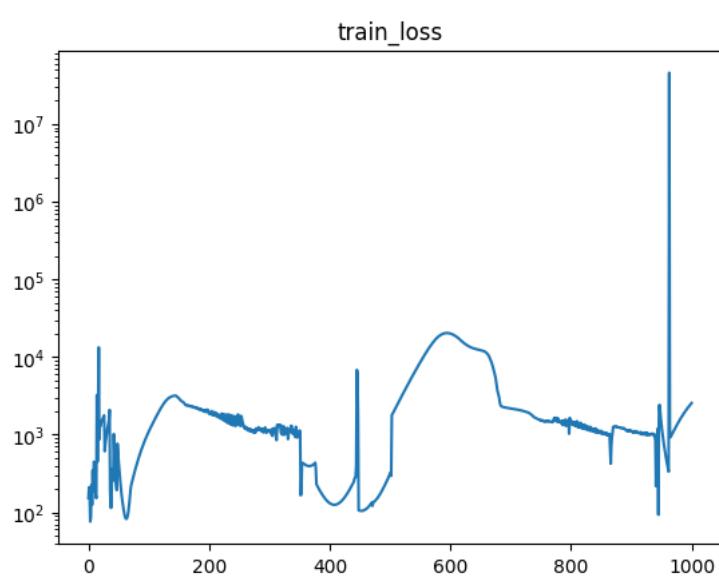


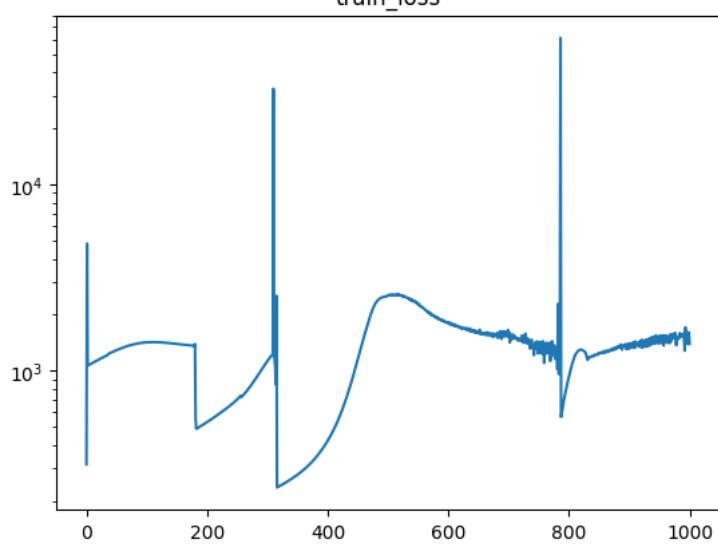
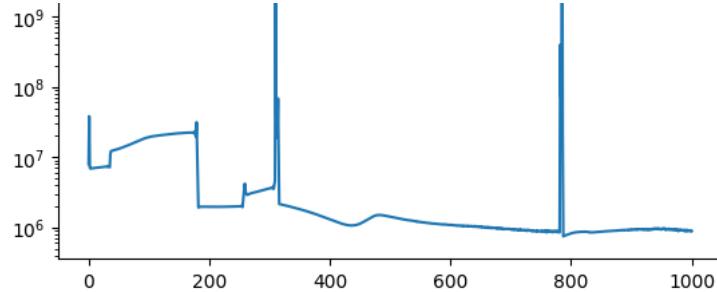
optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 0.5 , minimum_RMSE: 964.70 , epoch: 1000 , test_loss: 3362 , train_loss: 12



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

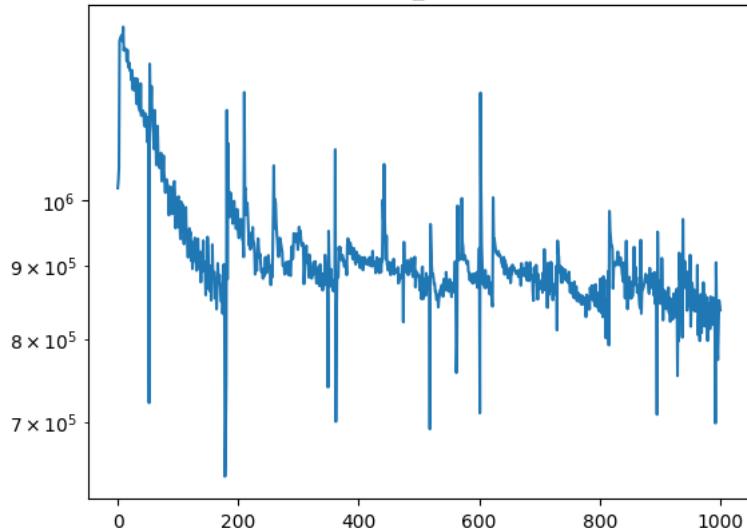




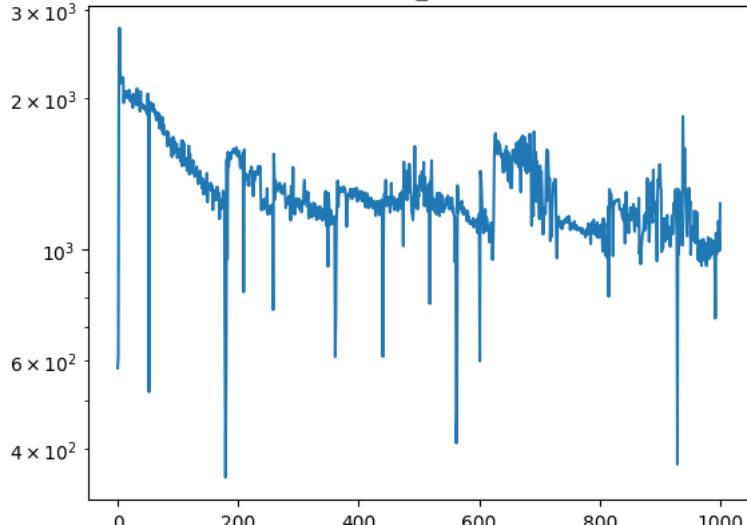


optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 0.005 , minimum_RMSE: 800.60 , epoch: 1000 , test_loss: 915 , train_loss:

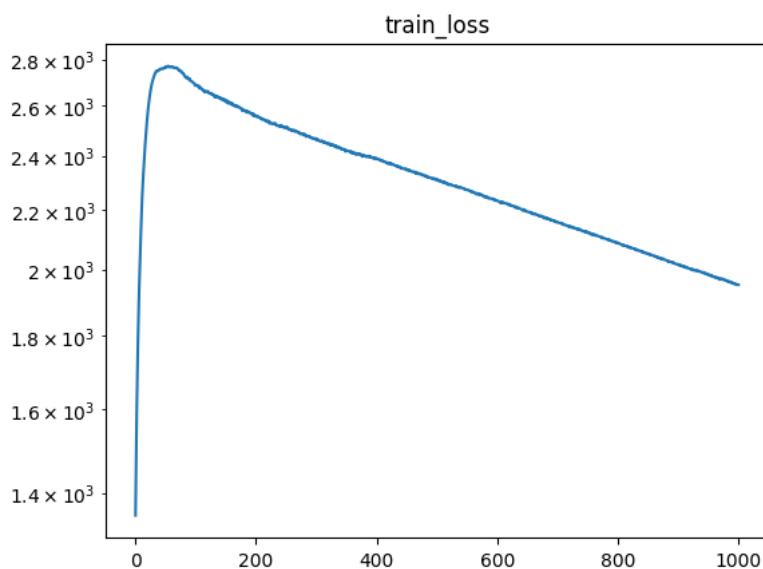
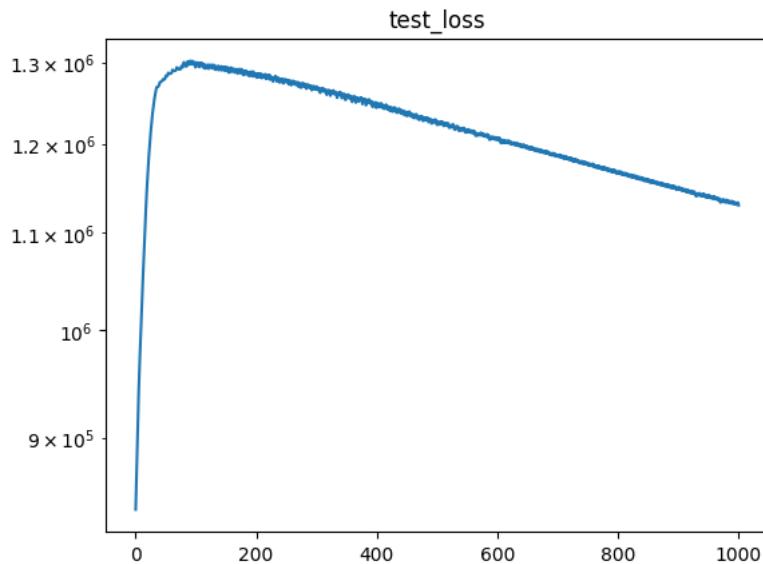
test_loss



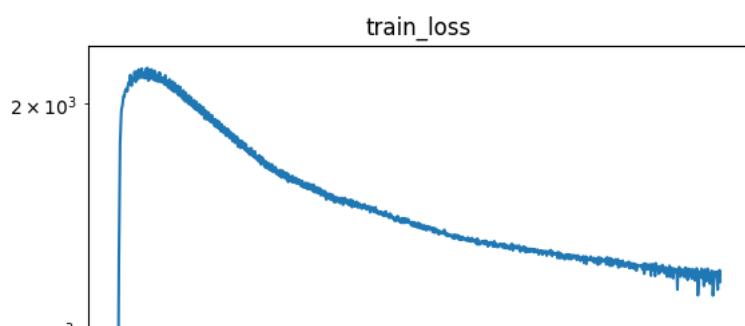
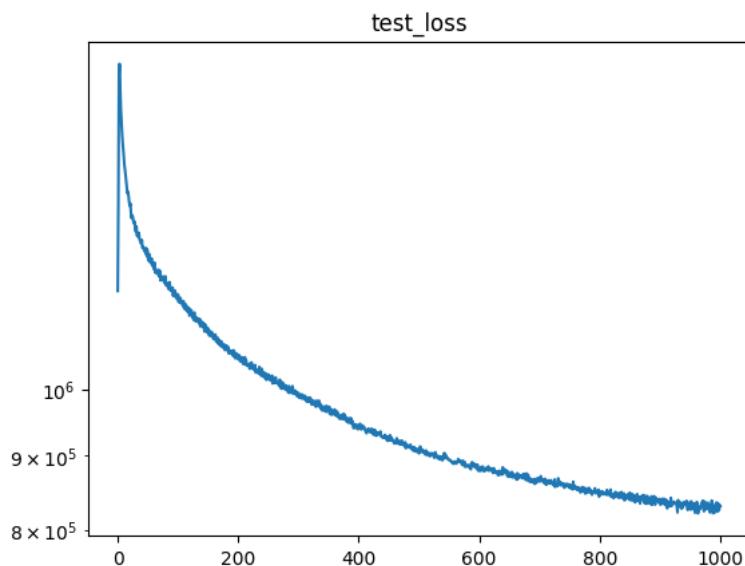
train_loss

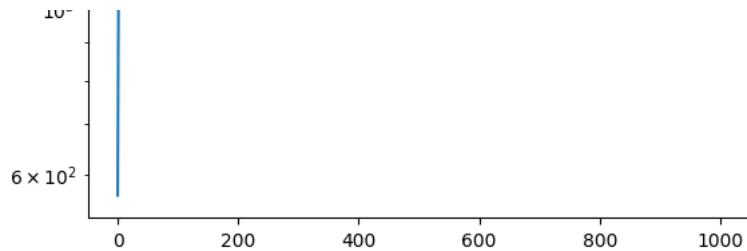


optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 0.0001 , minimum_RMSE: 916.01 , epoch: 1000 , test_loss: 1063 , train_loss:

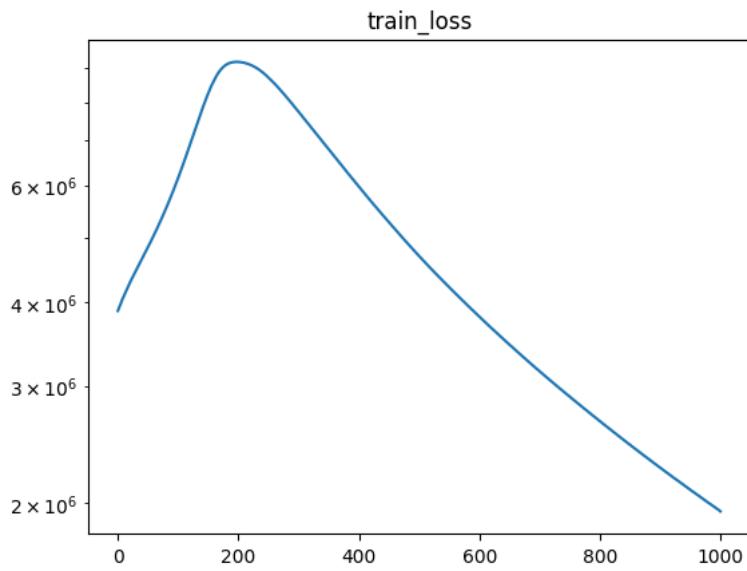
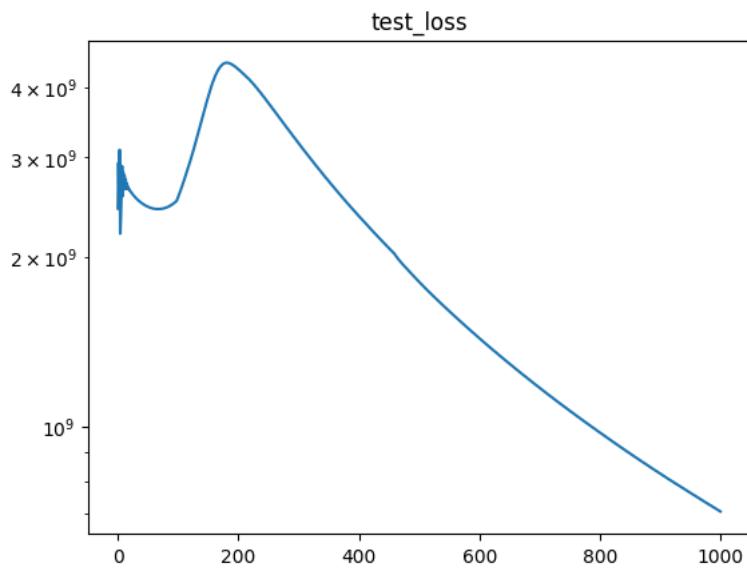


optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 0.0005 , minimum_RMSE: 906.31 , epoch: 1000 , test_loss: 911 , train_loss:

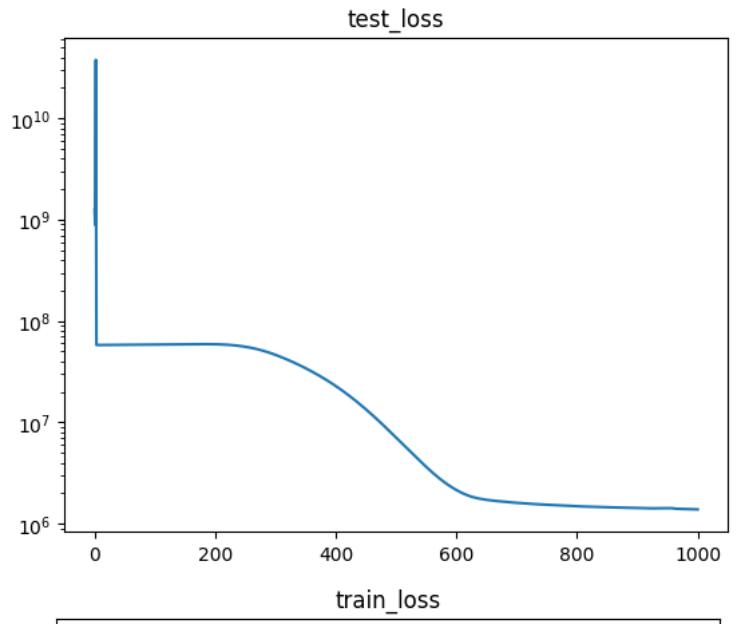


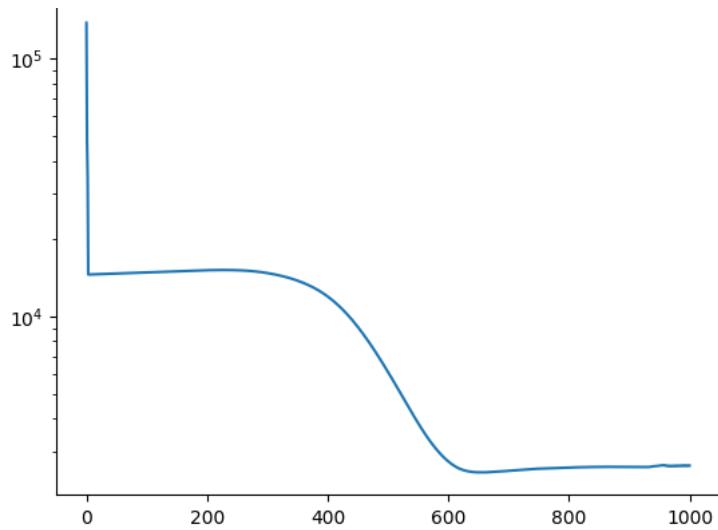


optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-05 , minimum_RMSE: 26597.27 , epoch: 1000 , test_loss: 26597 , train_loss:



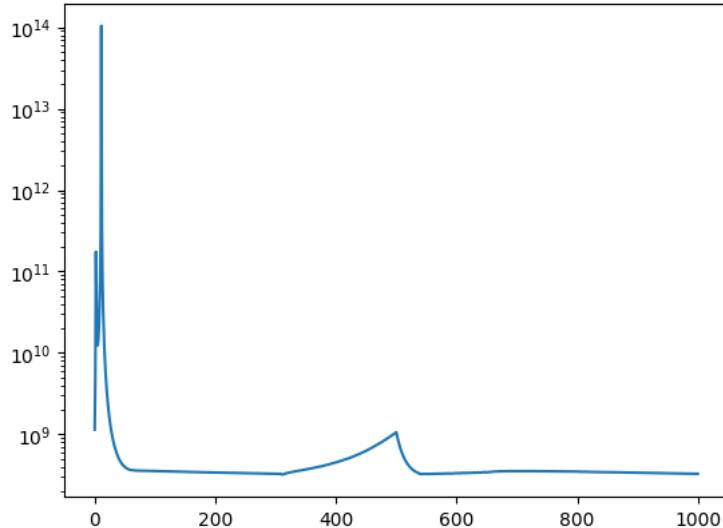
optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 5e-05 , minimum_RMSE: 1175.99 , epoch: 1000 , test_loss: 1176 , train_loss:



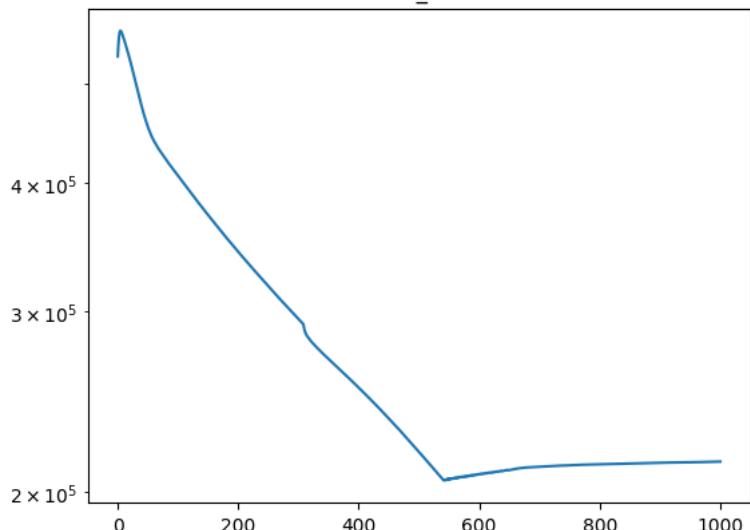


optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-06 , minimum_RMSE: 17902.22 , epoch: 1000 , test_loss: 18081 , train_loss:

test_loss

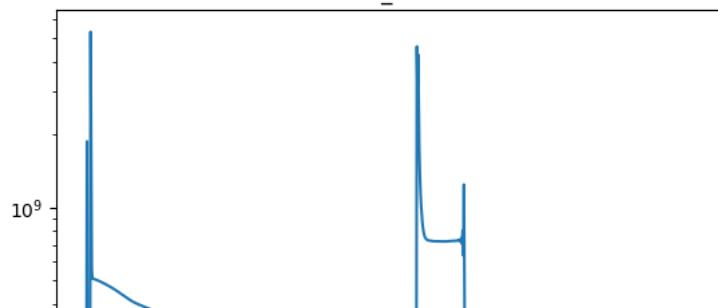


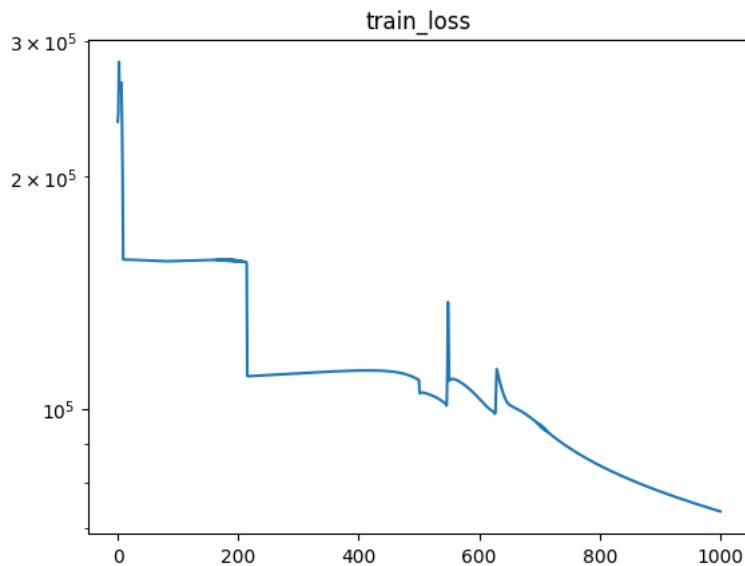
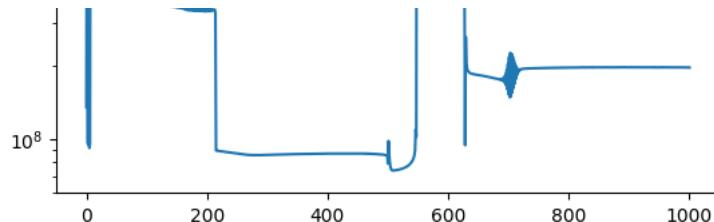
train_loss



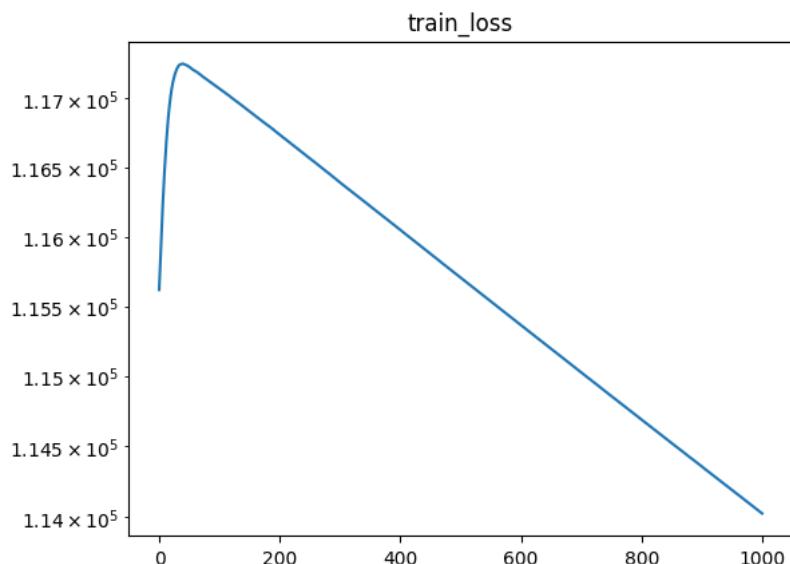
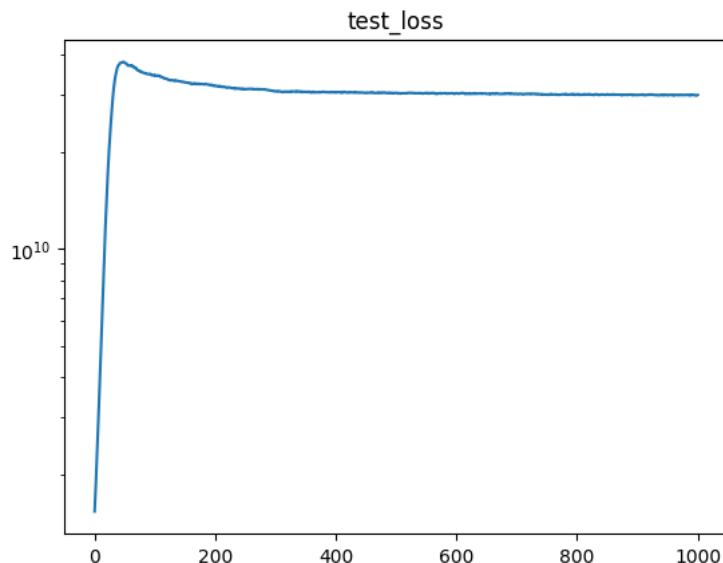
optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 5e-06 , minimum_RMSE: 8599.91 , epoch: 1000 , test_loss: 14039 , train_loss:

test_loss



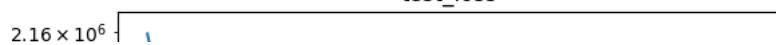


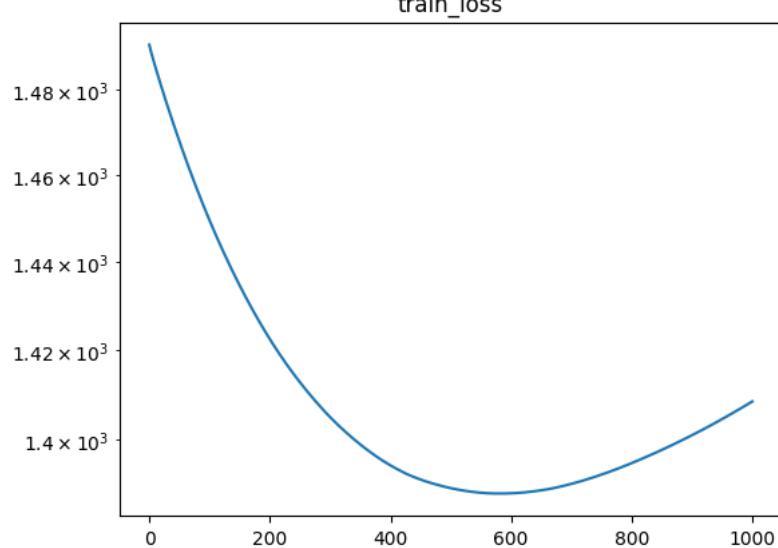
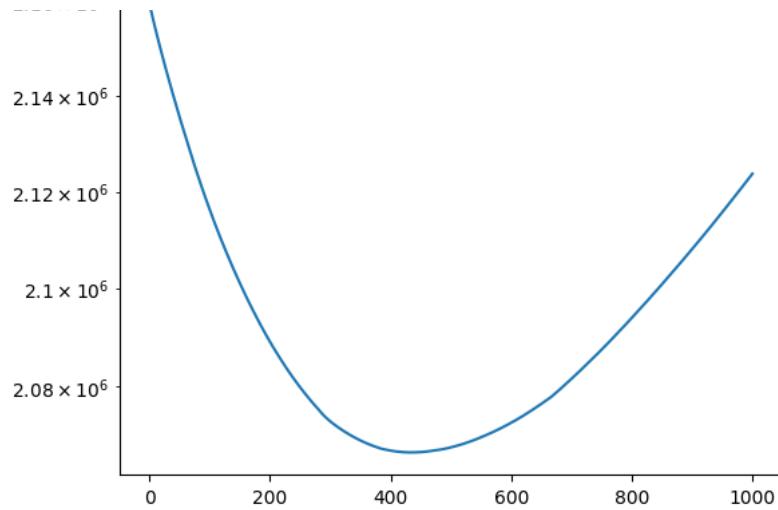
optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-07 , minimum_RMSE: 39123.97 , epoch: 1000 , test_loss: 173274 , train_loss:



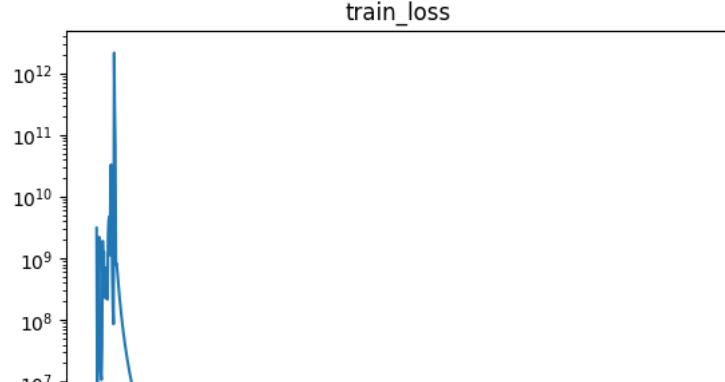
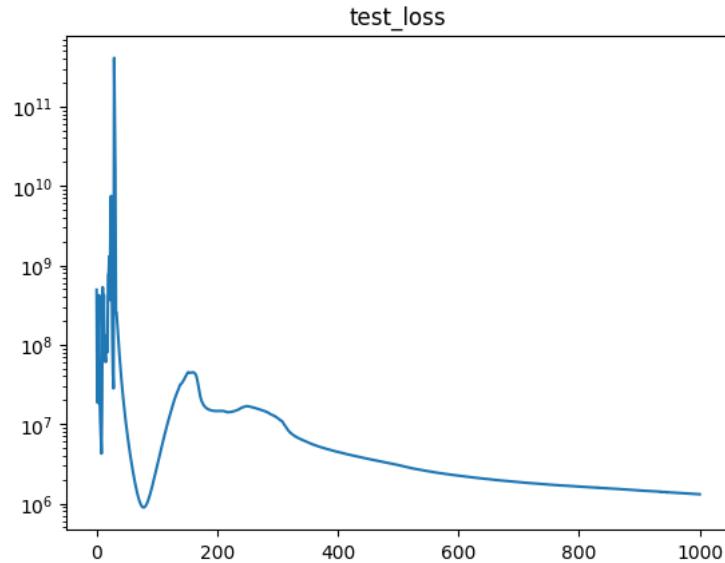
optimizer: AdamW , n_of_data: 1000 , Bsize: 10 , learningRate: 5e-07 , minimum_RMSE: 1437.57 , epoch: 1000 , test_loss: 1457 , train_loss:

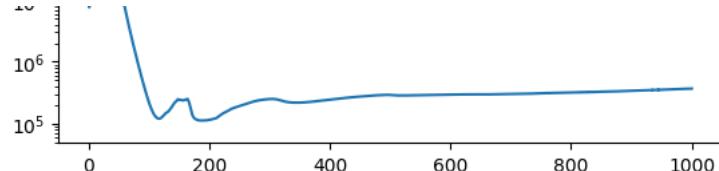
test_loss



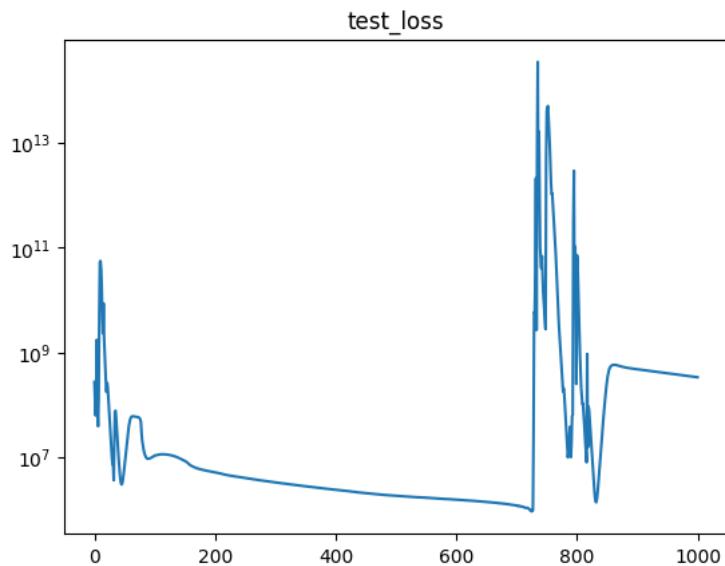


optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 0.1 , minimum_RMSE: 943.94 , epoch: 1000 , test_loss: 1142 , train_loss: 600

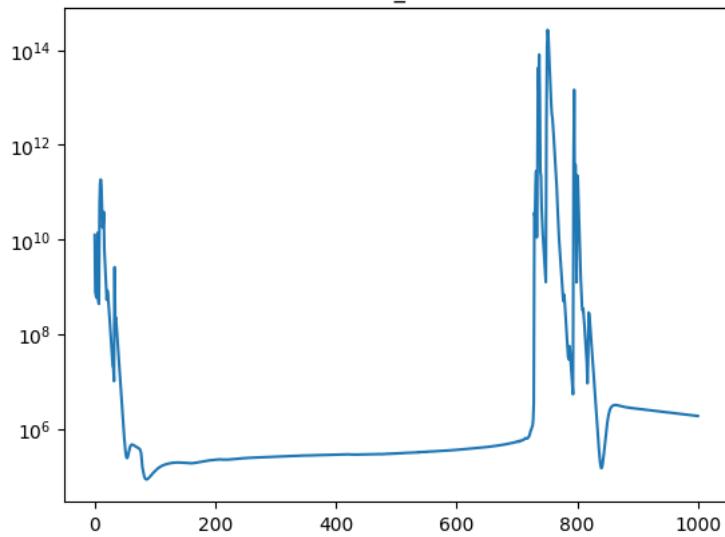




optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 0.5 , minimum_RMSE: 959.74 , epoch: 1000 , test_loss: 18281 , train_loss: 1375

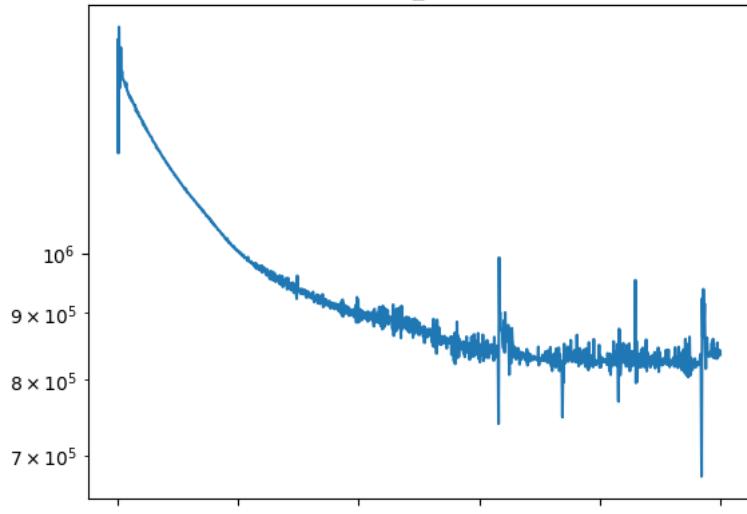


train_loss

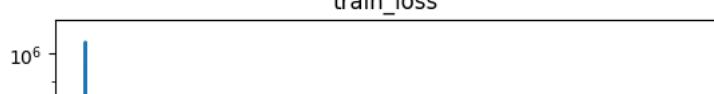


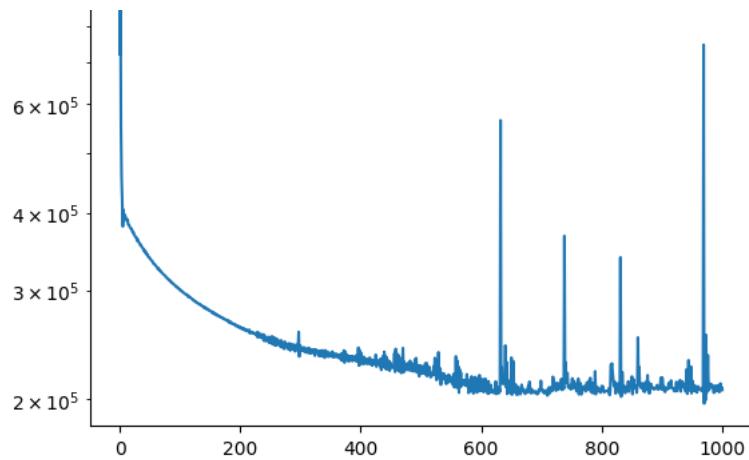
optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 0.01 , minimum_RMSE: 821.53 , epoch: 1000 , test_loss: 914 , train_loss: 4

test_loss



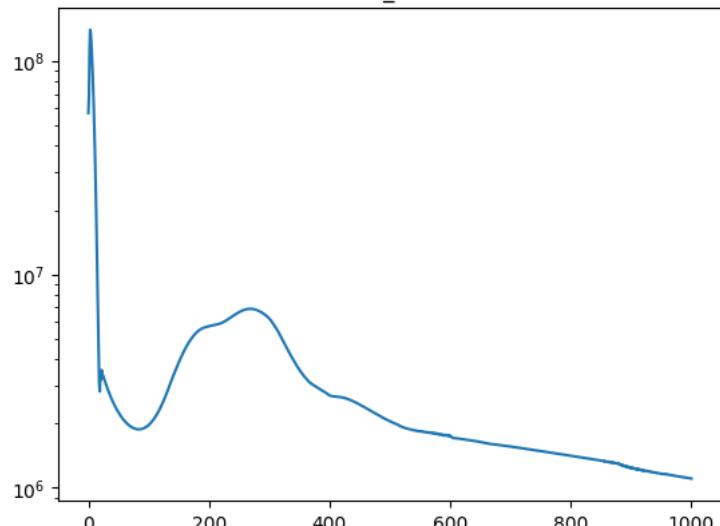
train_loss



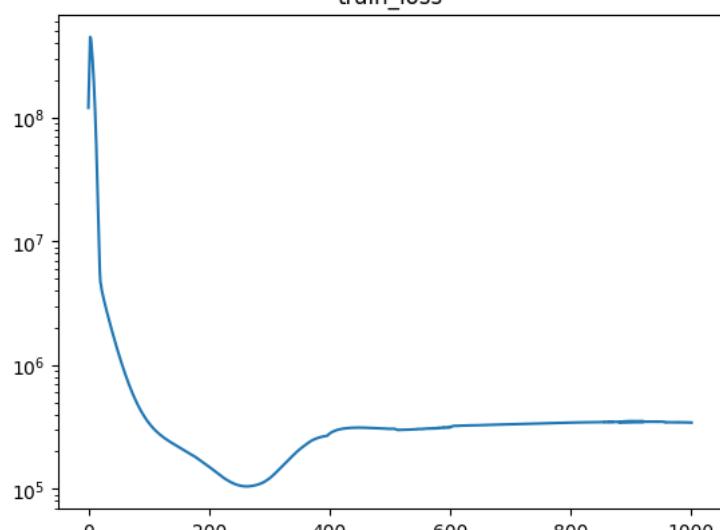


optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 0.05 , minimum_RMSE: 1050.42 , epoch: 1000 , test_loss: 1050 , train_loss:

test_loss

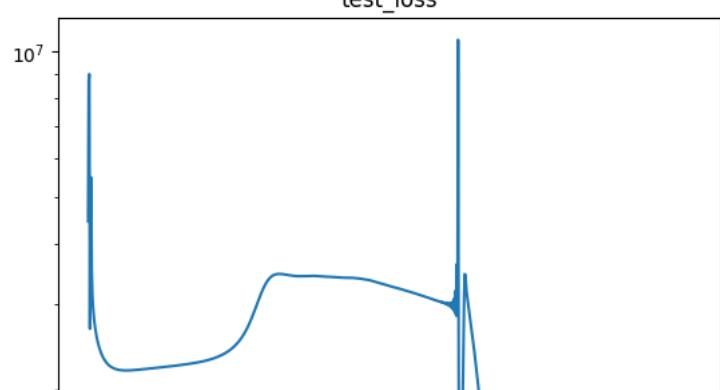


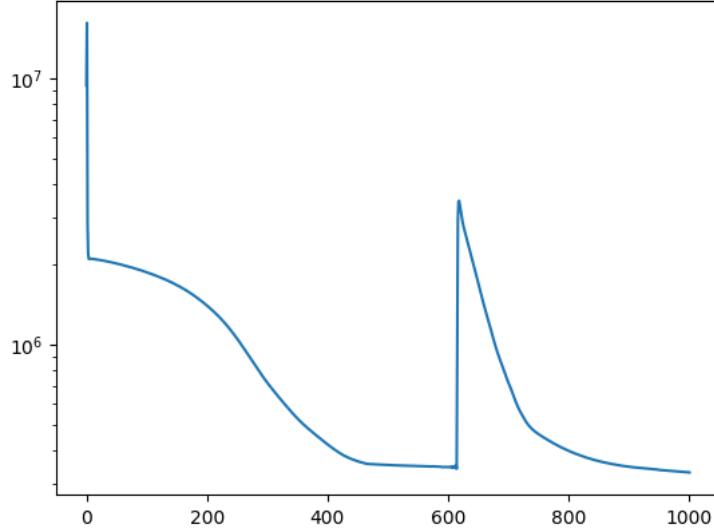
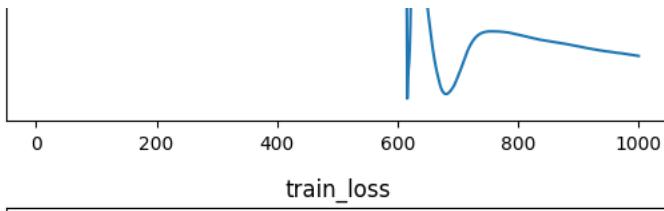
train_loss



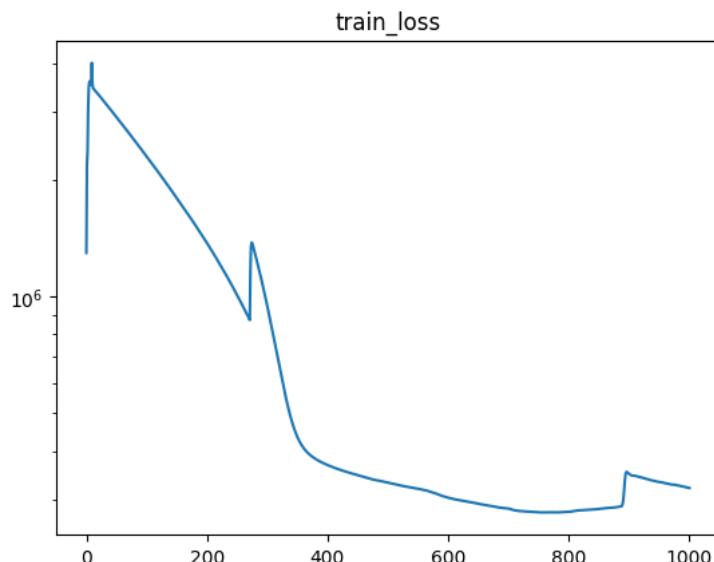
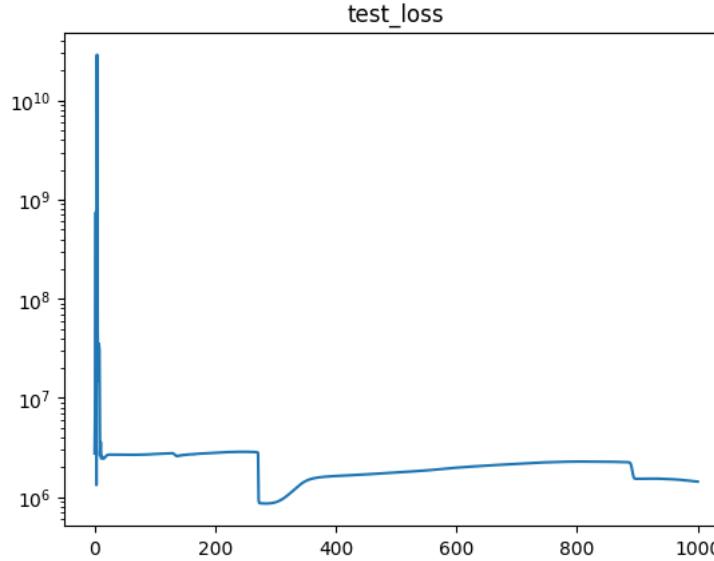
optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 0.001 , minimum_RMSE: 1114.49 , epoch: 1000 , test_loss: 1233 , train_loss:

test_loss

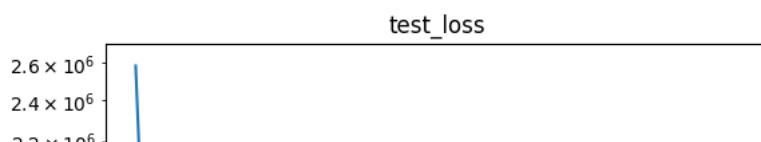


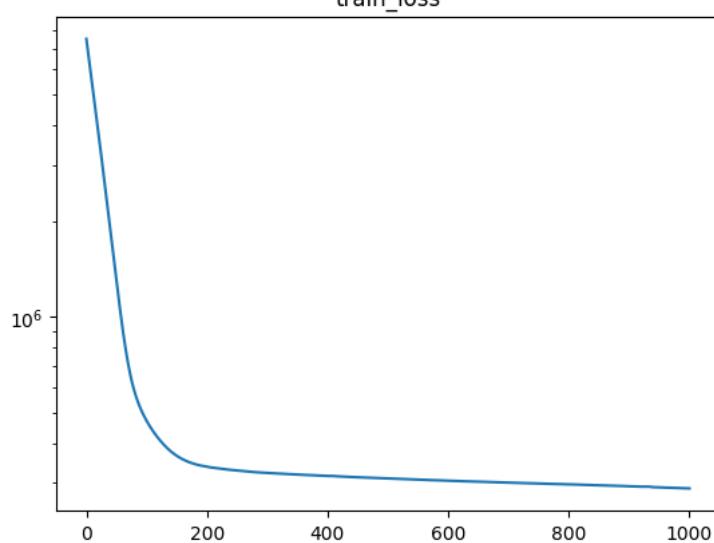
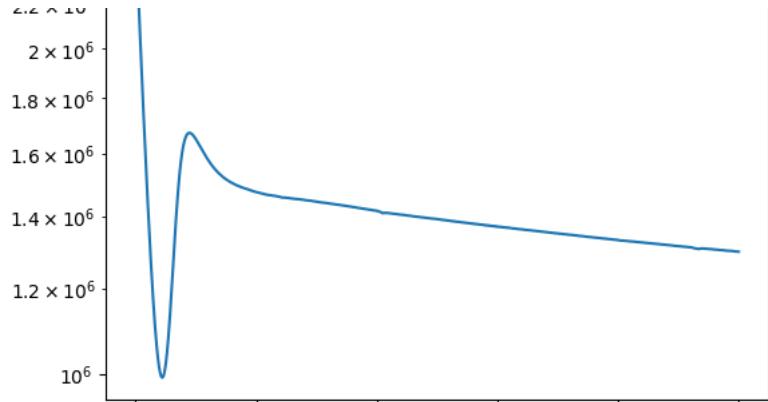


optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 0.005 , minimum_RMSE: 926.32 , epoch: 1000 , test_loss: 1194 , train_loss:

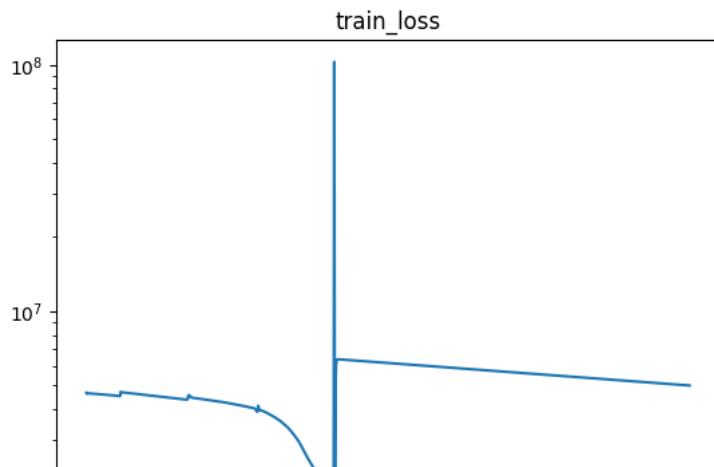
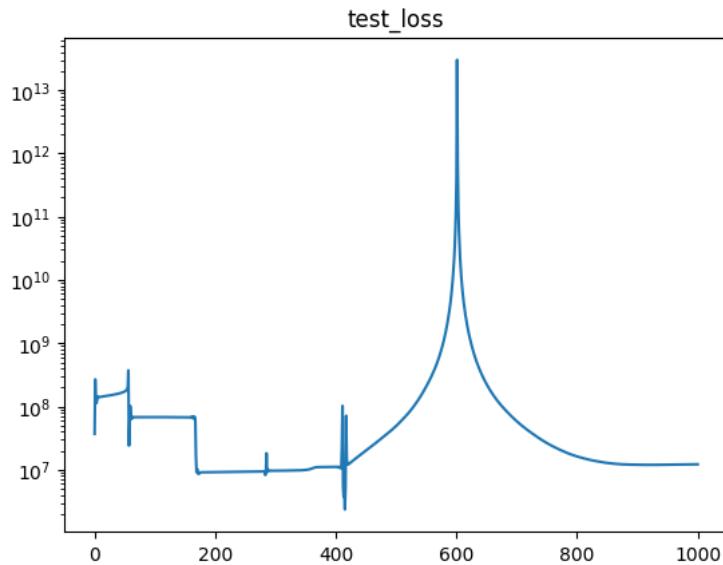


optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 0.0001 , minimum_RMSE: 996.41 , epoch: 1000 , test_loss: 1139 , train_loss:





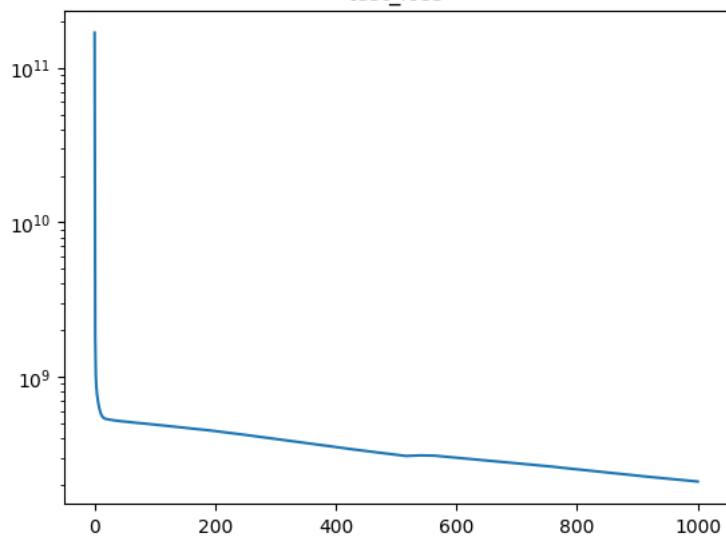
optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 0.0005 , minimum_RMSE: 1538.68 , epoch: 1000 , test_loss: 3495 , train_loss:



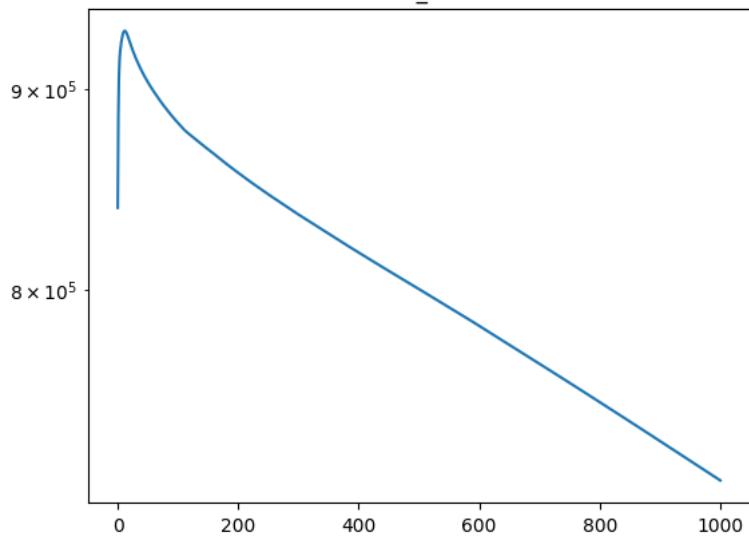


optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-05 , minimum_RMSE: 14503.24 , epoch: 1000 , test_loss: 14503 , train_loss:

test_loss

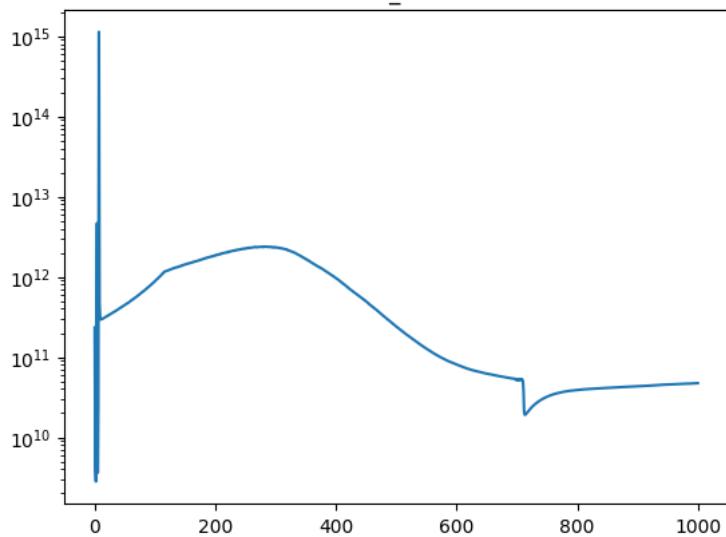


train_loss



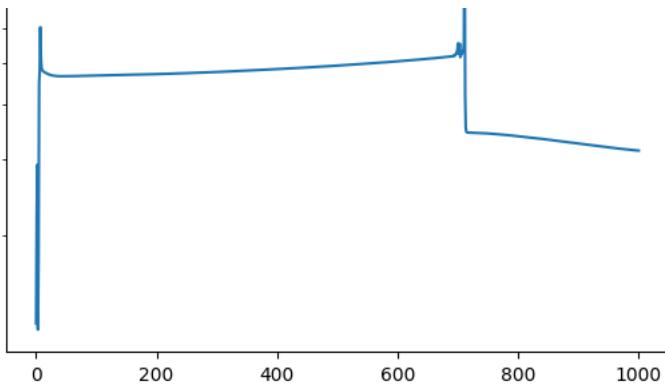
optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 5e-05 , minimum_RMSE: 53109.38 , epoch: 1000 , test_loss: 217581 , train_loss:

test_loss



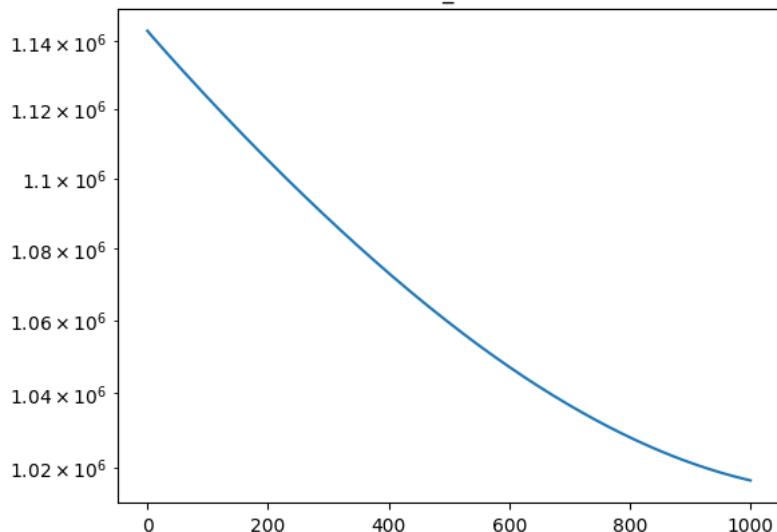
train_loss



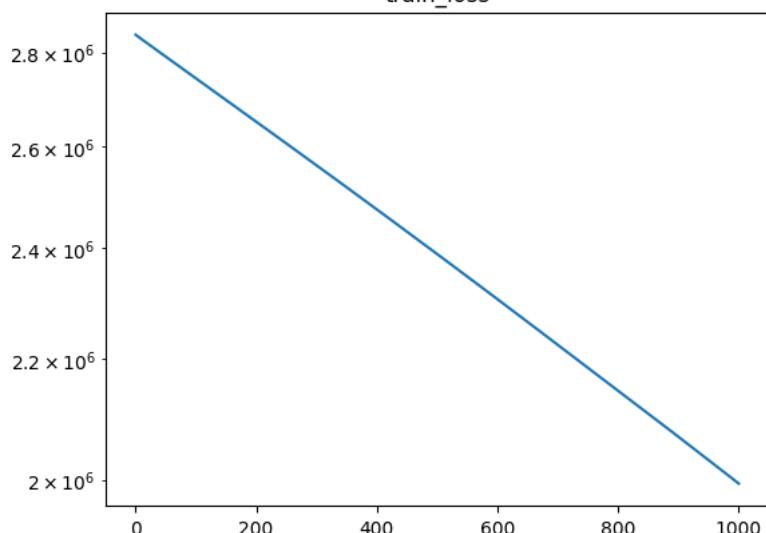


optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-06 , minimum_RMSE: 1008.28 , epoch: 1000 , test_loss: 1008 , train_loss:

test_loss

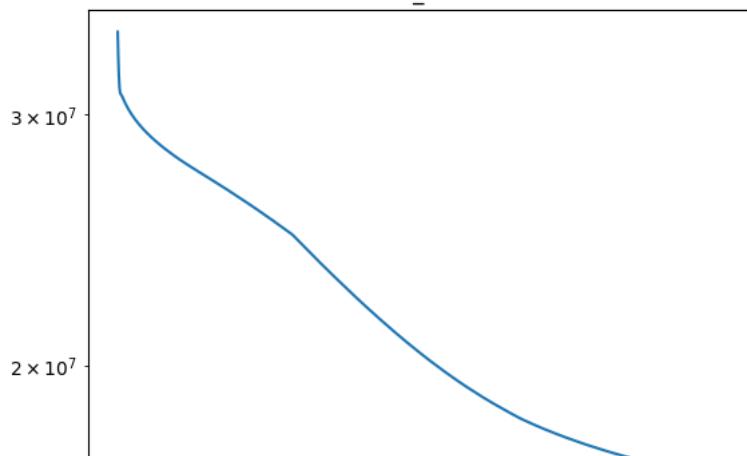


train_loss



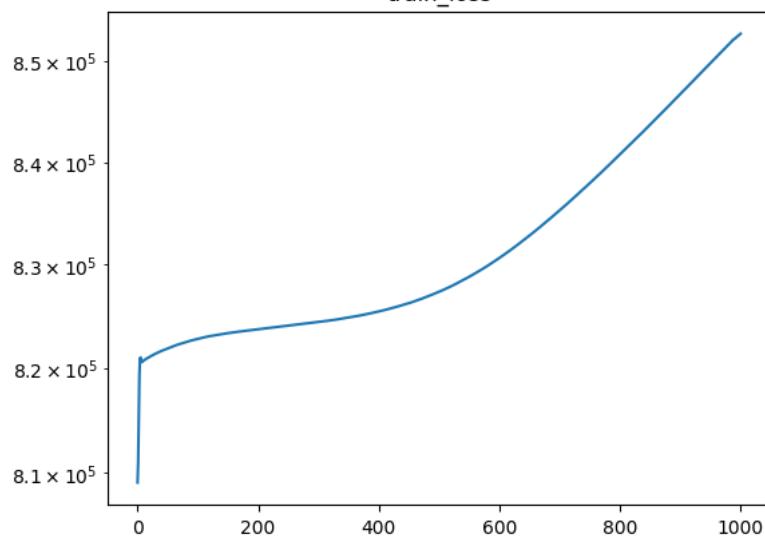
optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 5e-06 , minimum_RMSE: 4071.18 , epoch: 1000 , test_loss: 4071 , train_loss:

test_loss



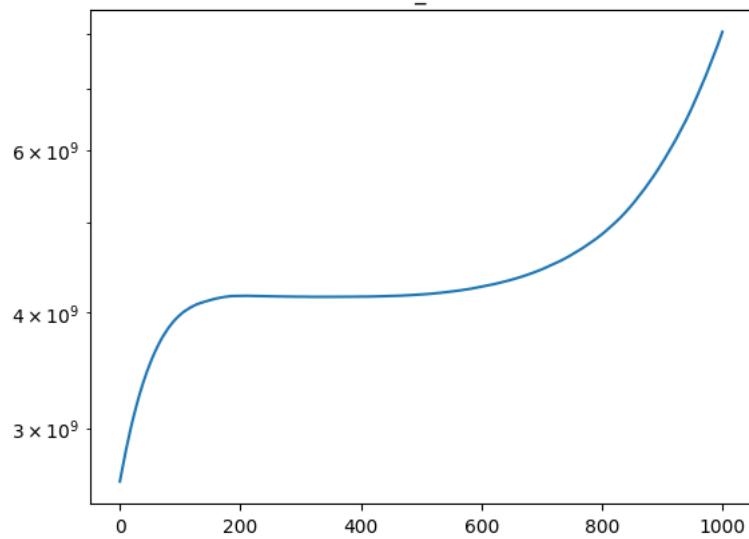


train_loss

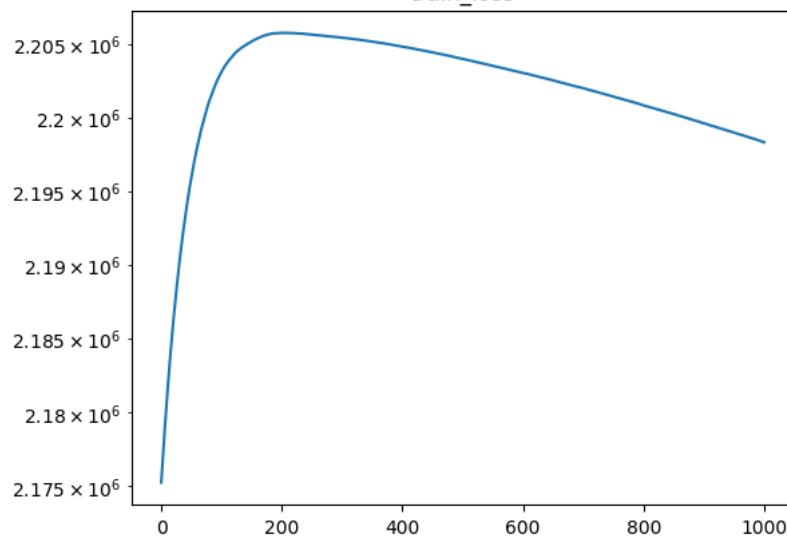


optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-07 , minimum_RMSE: 51265.68 , epoch: 1000 , test_loss: 89735 , train_loss:

test_loss

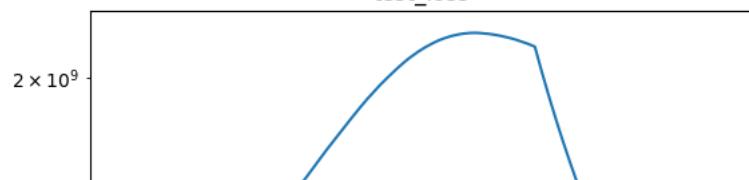


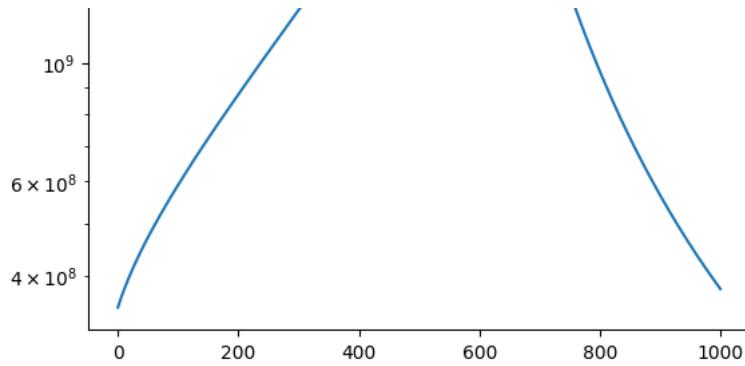
train_loss



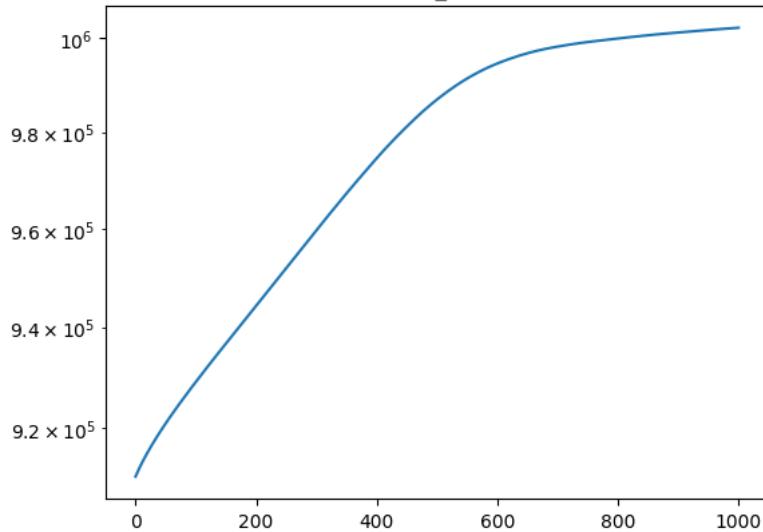
optimizer: AdamW , n_of_data: 1000 , Bsize: 50 , learningRate: 5e-07 , minimum_RMSE: 18669.13 , epoch: 1000 , test_loss: 19426 , train_loss:

test_loss

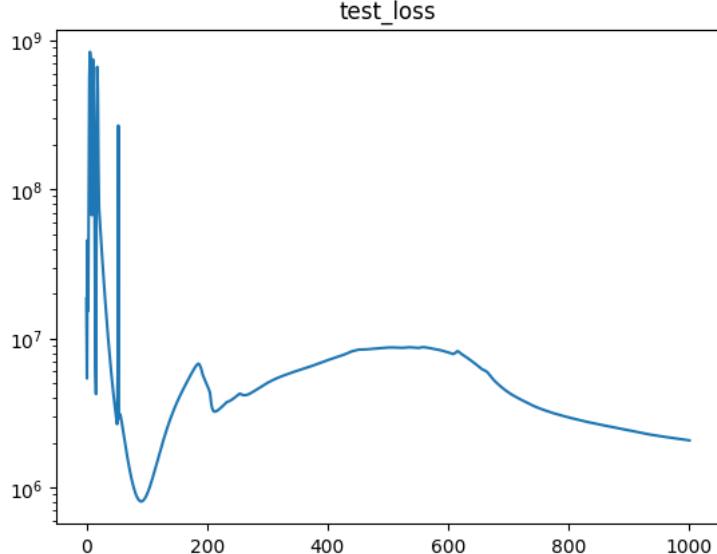




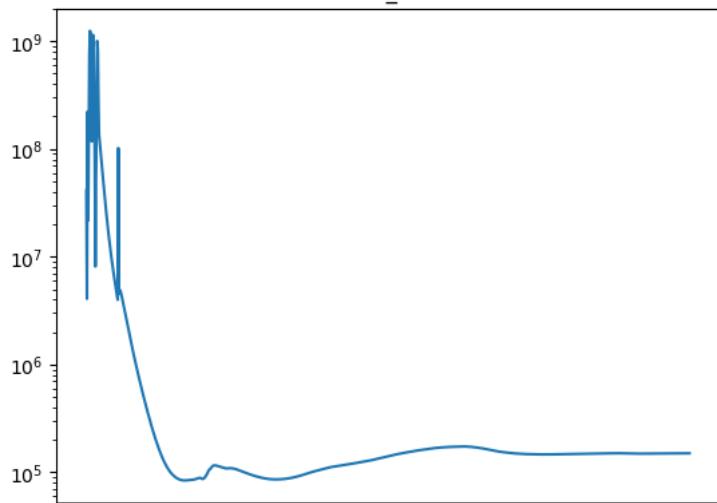
train_loss



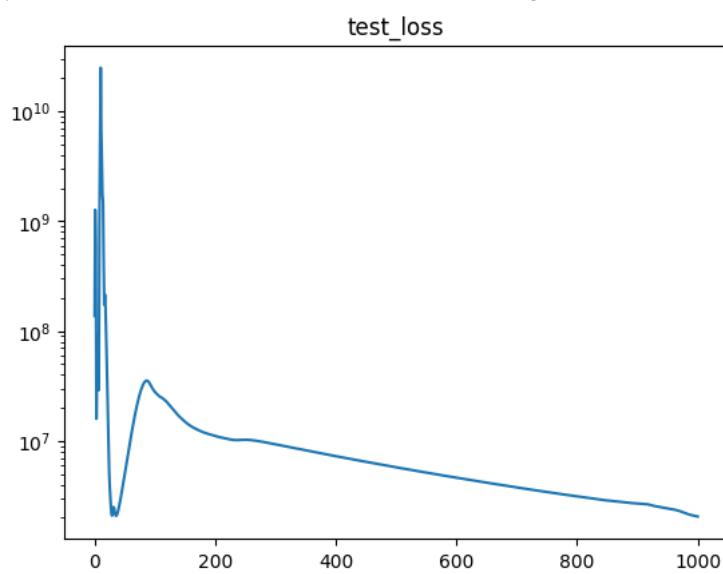
test_loss



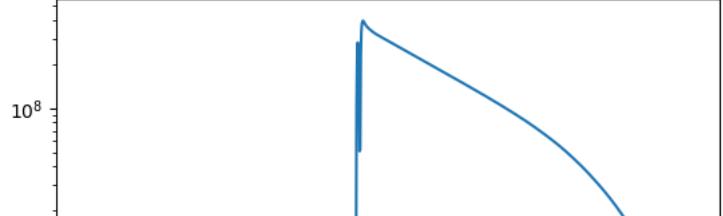
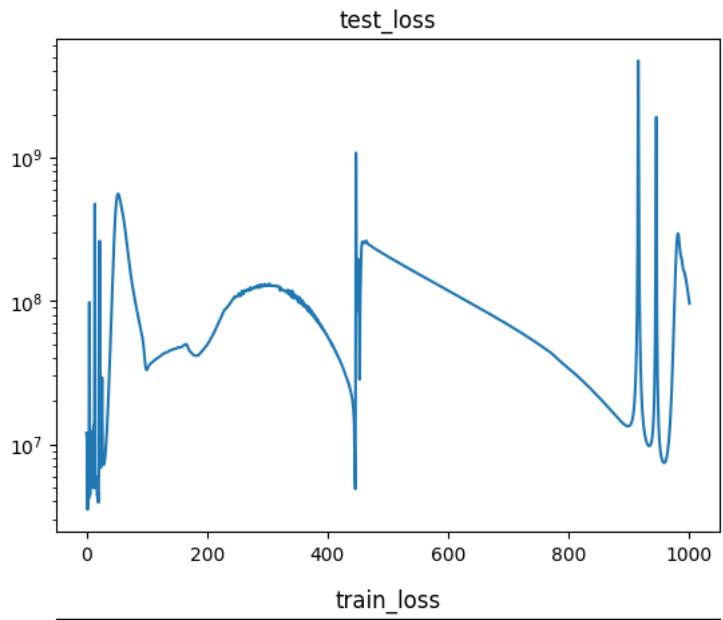
train_loss

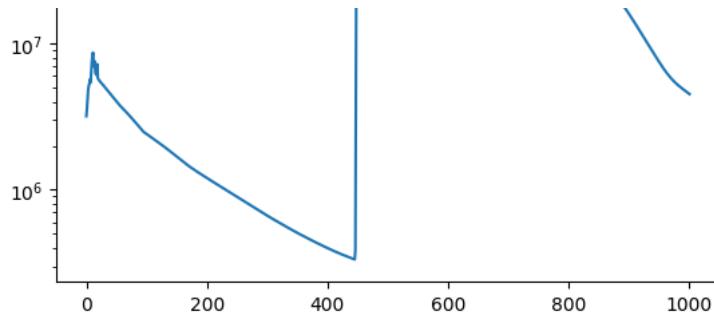


optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 0.5 , minimum_RMSE: 1434.30 , epoch: 1000 , test_loss: 1434 , train_loss:



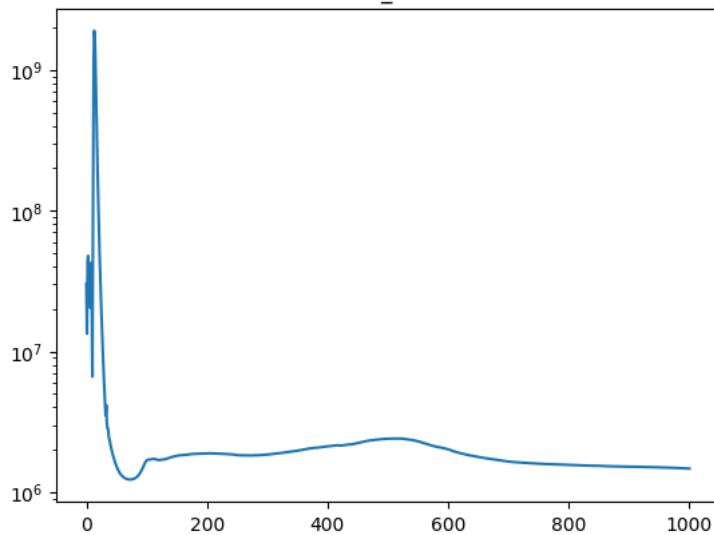
optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 0.01 , minimum_RMSE: 1874.76 , epoch: 1000 , test_loss: 9789 , train_loss:



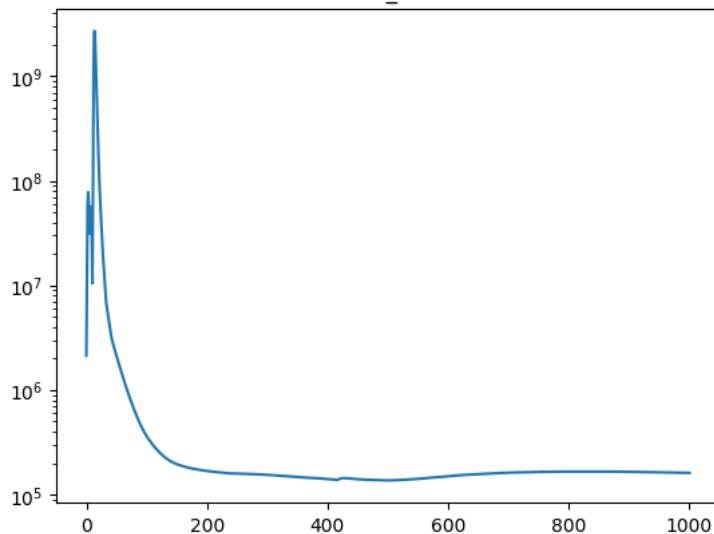


optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 0.05 , minimum_RMSE: 1107.96 , epoch: 1000 , test_loss: 1211 , train_loss:

test_loss

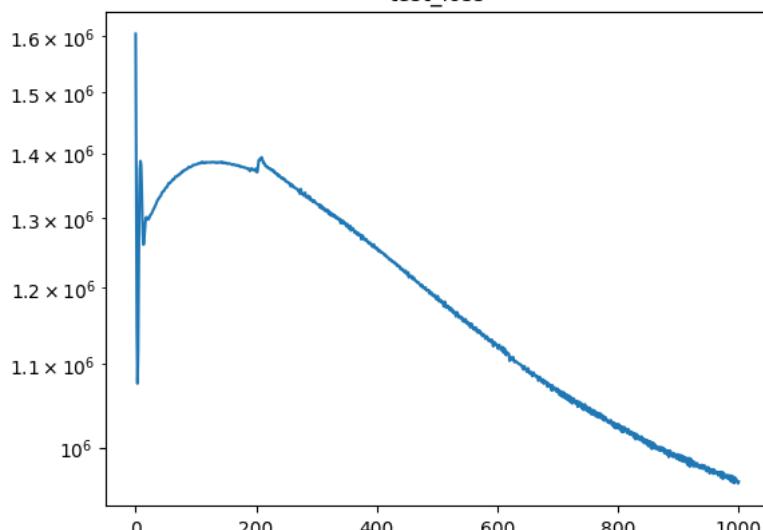


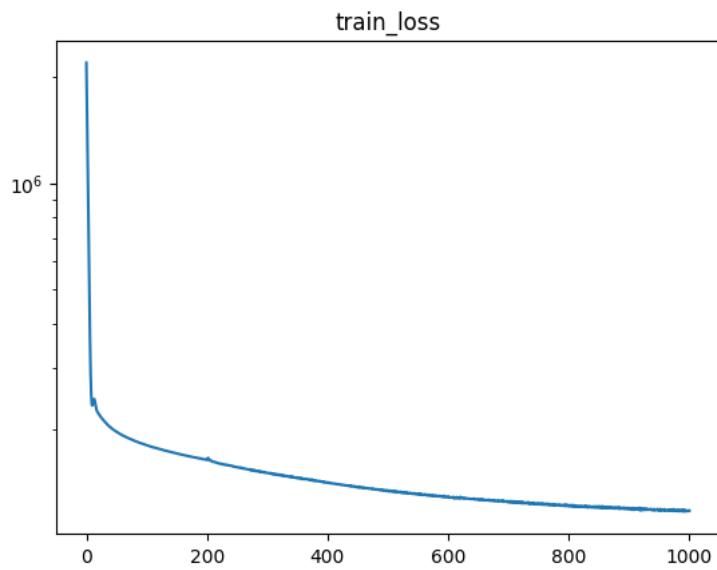
train_loss



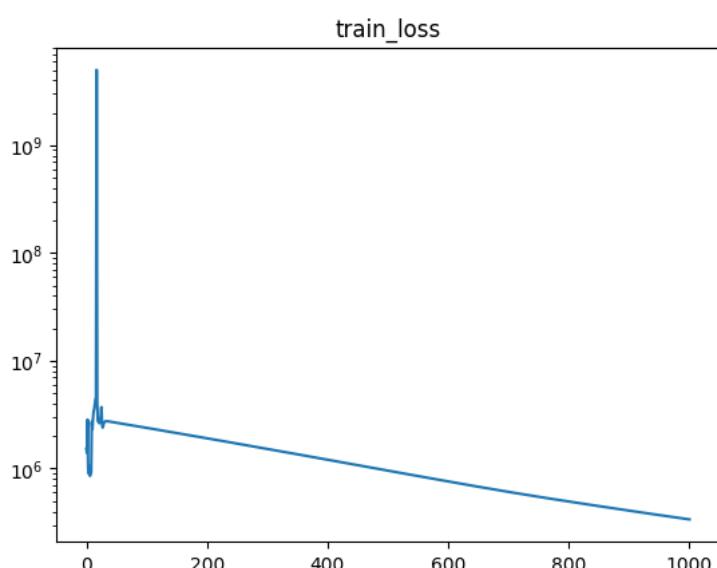
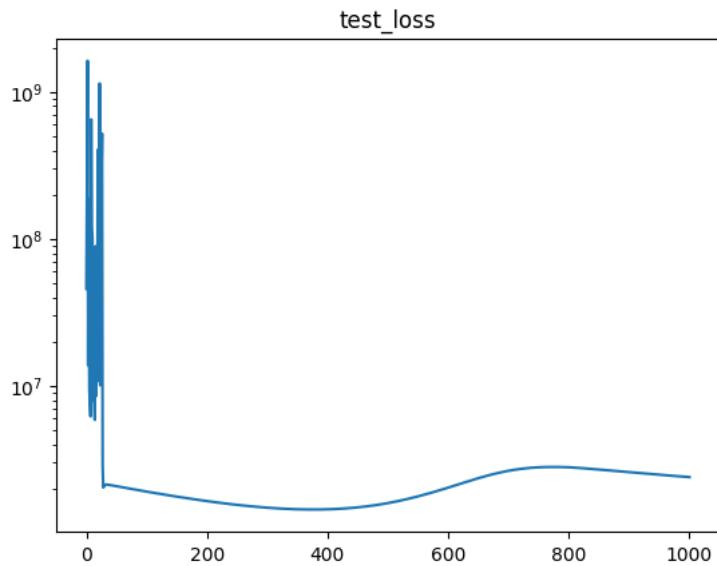
optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 0.001 , minimum_RMSE: 979.86 , epoch: 1000 , test_loss: 980 , train_loss:

test_loss

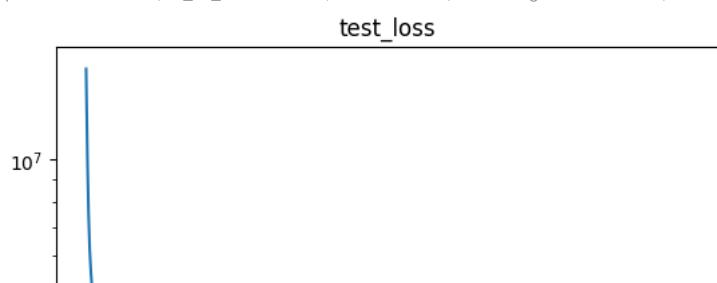


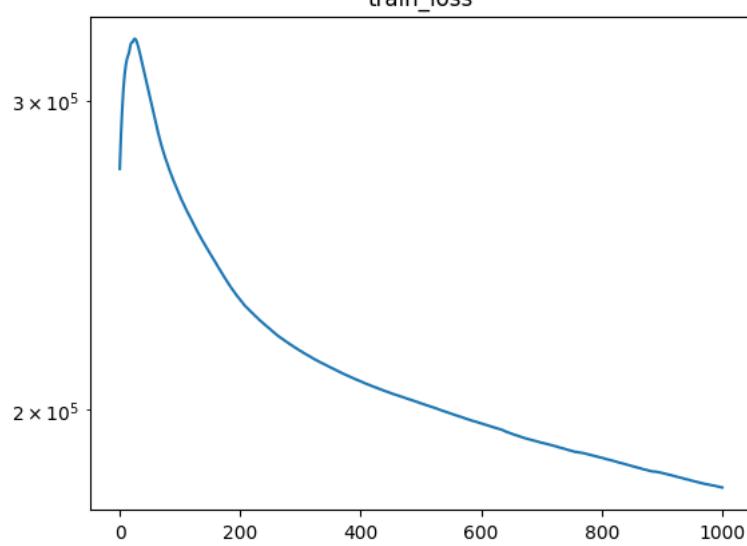
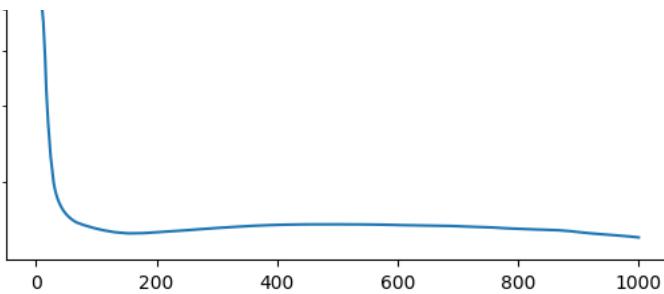


optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 0.005 , minimum_RMSE: 1201.15 , epoch: 1000 , test_loss: 1548 , train_loss:

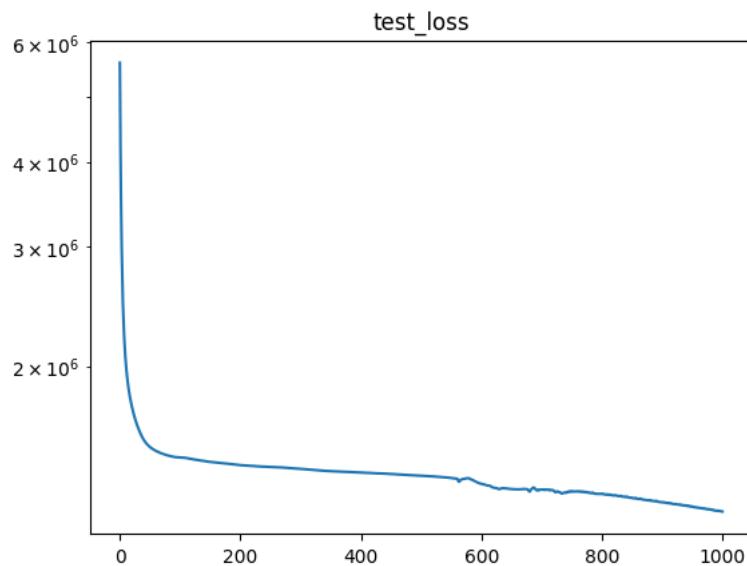


optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 0.0001 , minimum_RMSE: 1221.03 , epoch: 1000 , test_loss: 1221 , train_loss:

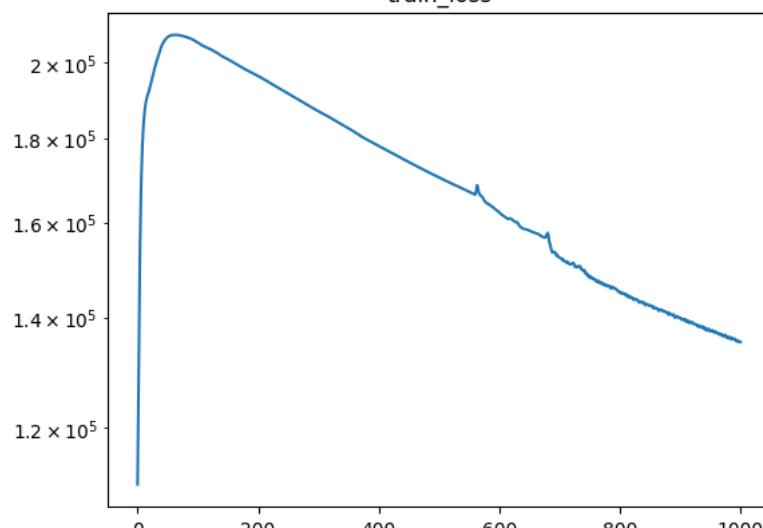




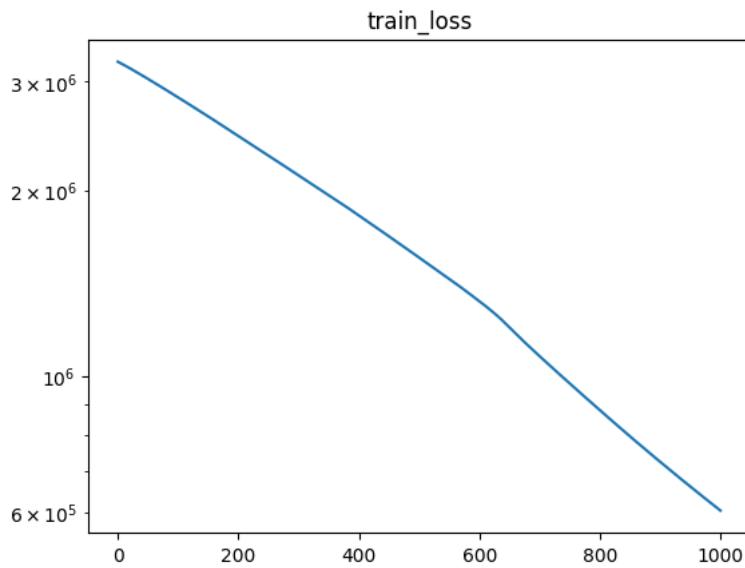
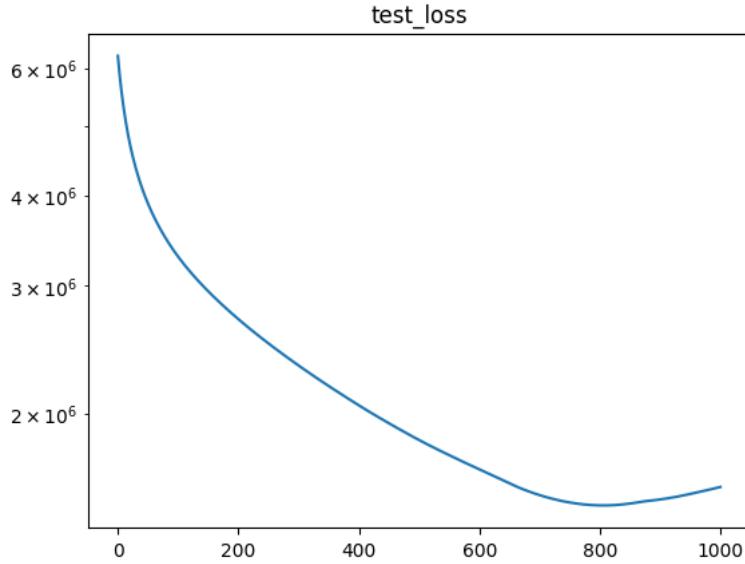
optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 0.0005 , minimum_RMSE: 1104.69 , epoch: 1000 , test_loss: 1104 , train_loss



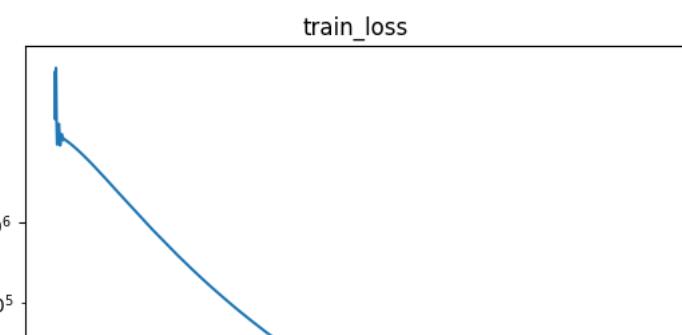
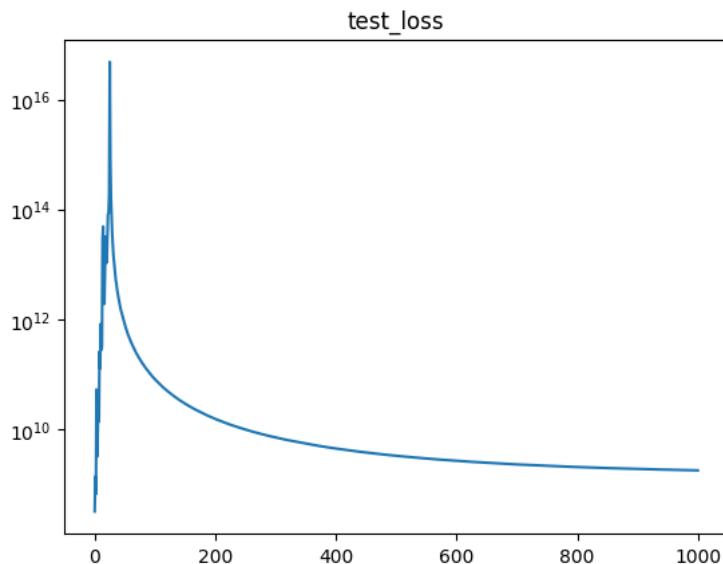
train_loss

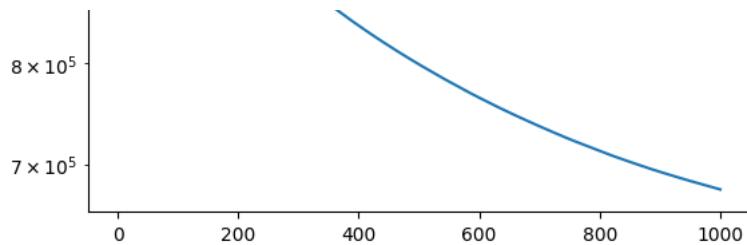


optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-05 , minimum_RMSE: 1221.30 , epoch: 1000 , test_loss: 1257 , train_loss:

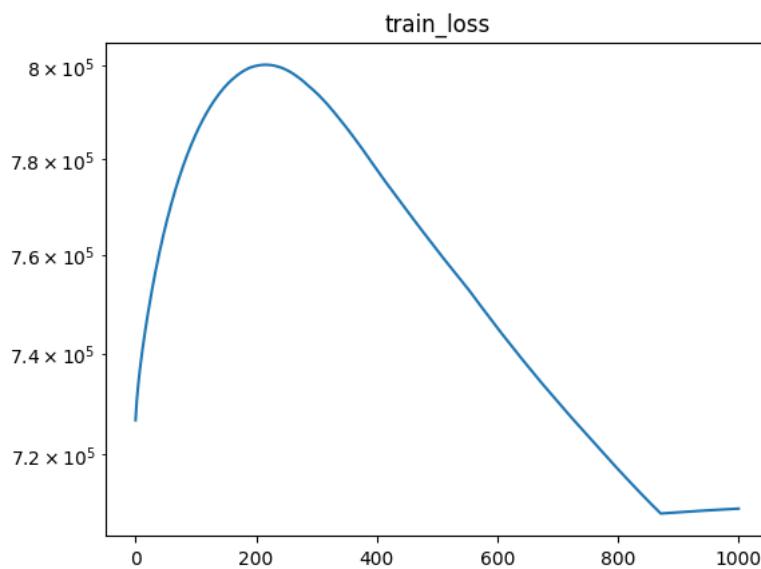
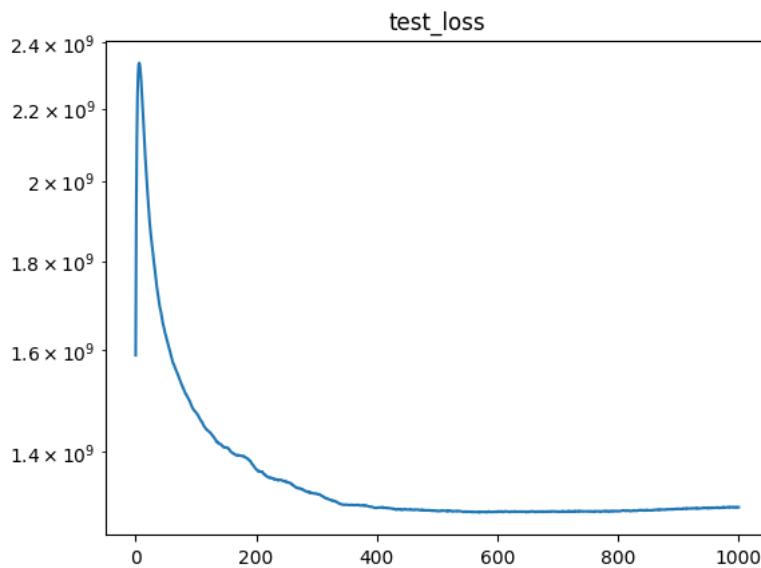


optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-05 , minimum_RMSE: 17972.77 , epoch: 1000 , test_loss: 42466 , train_loss:

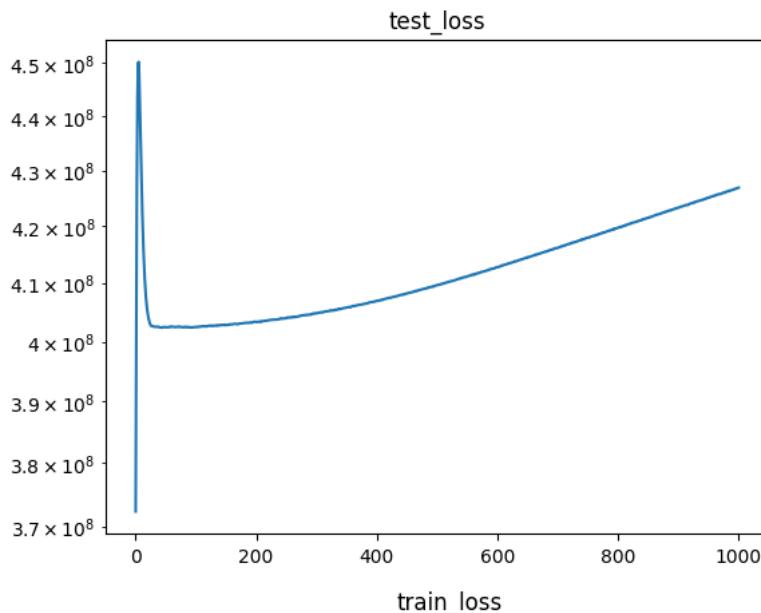


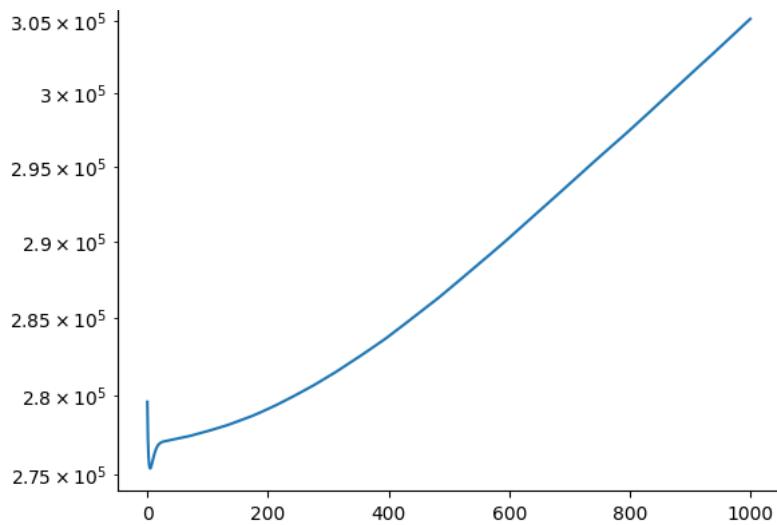


optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-06 , minimum_RMSE: 35950.72 , epoch: 1000 , test_loss: 36074 , train_loss



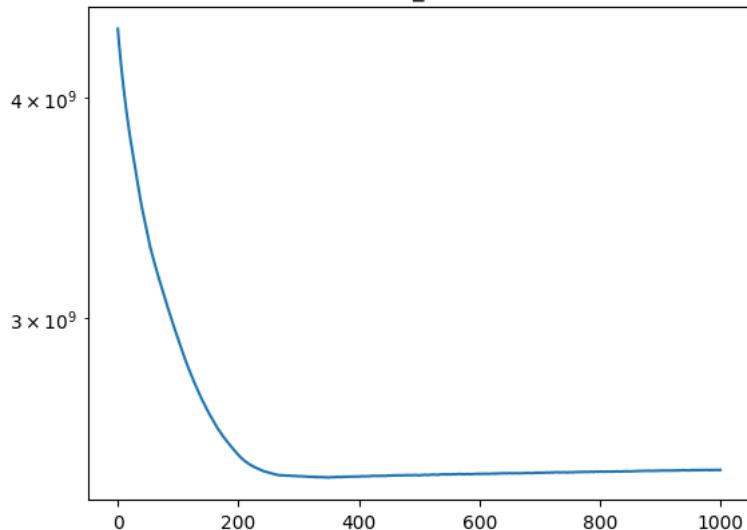
optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-06 , minimum_RMSE: 19295.79 , epoch: 1000 , test_loss: 20662 , train_loss



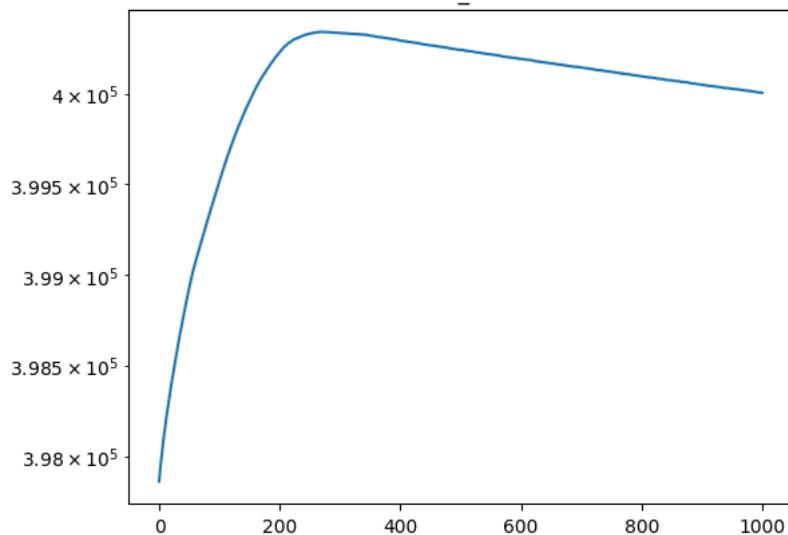


optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-07 , minimum_RMSE: 49366.19 , epoch: 1000 , test_loss: 49612 , train_loss:

test_loss

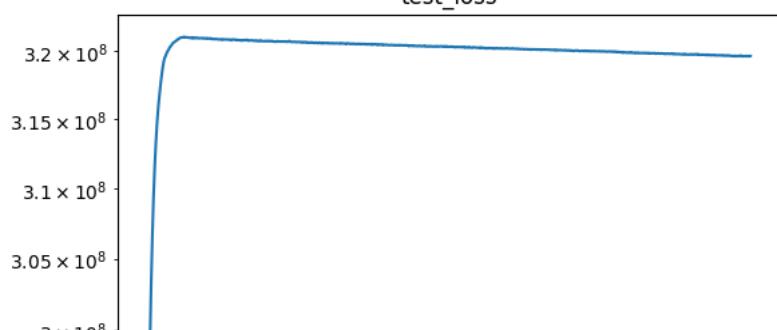


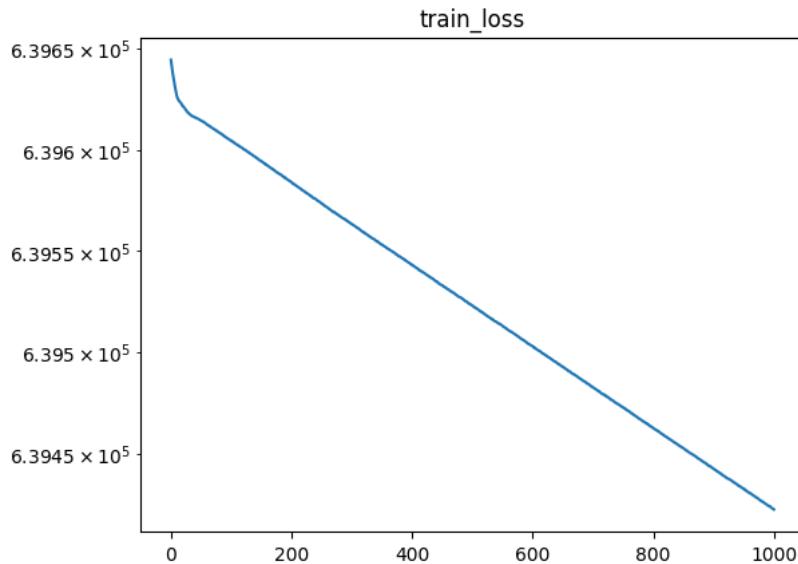
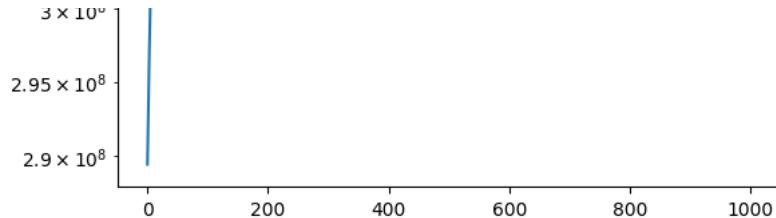
train_loss



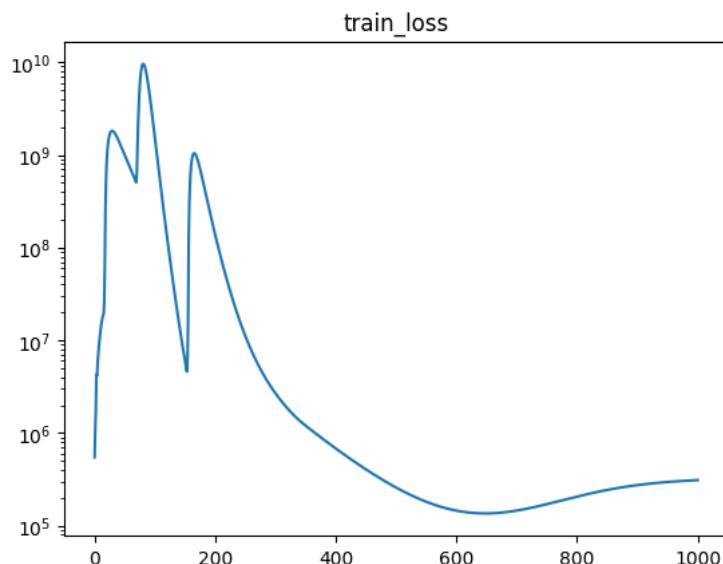
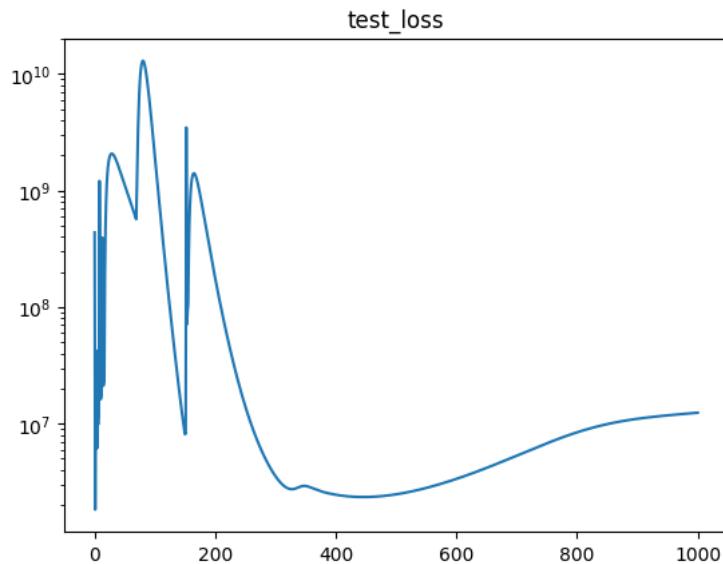
optimizer: AdamW , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-07 , minimum_RMSE: 17013.55 , epoch: 1000 , test_loss: 17878 , train_loss:

test_loss





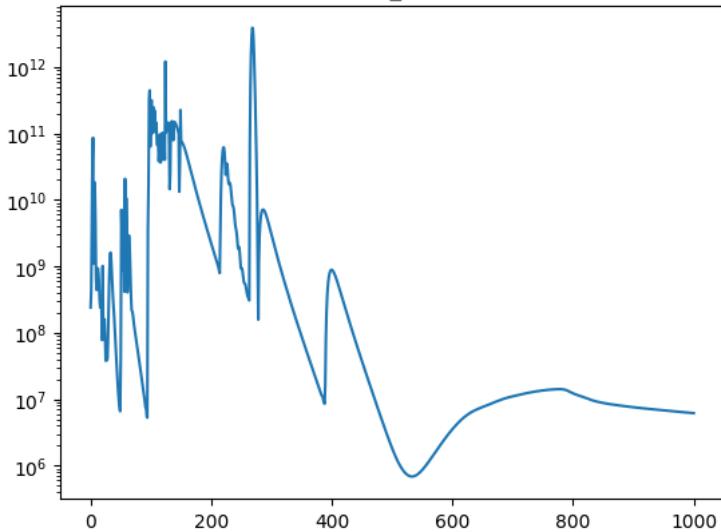
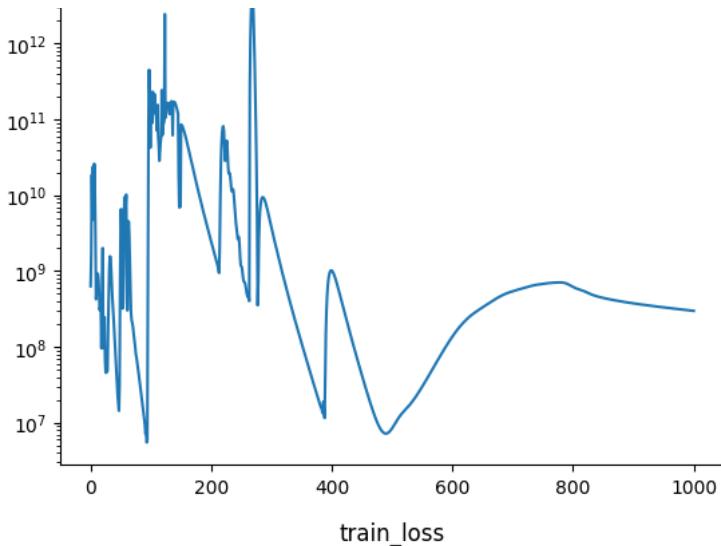
optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 0.1 , minimum_RMSE: 1354.85 , epoch: 1000 , test_loss: 3524 , train_loss:



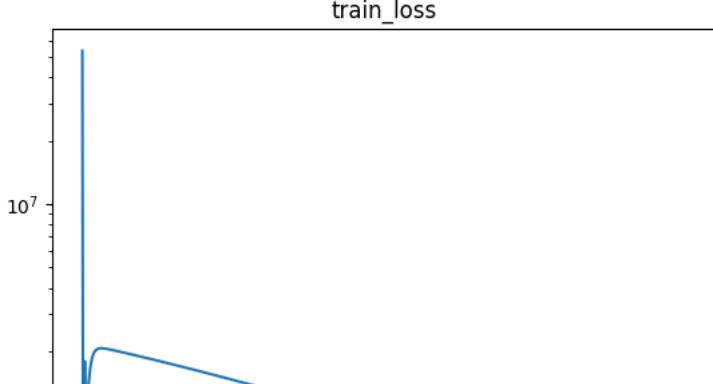
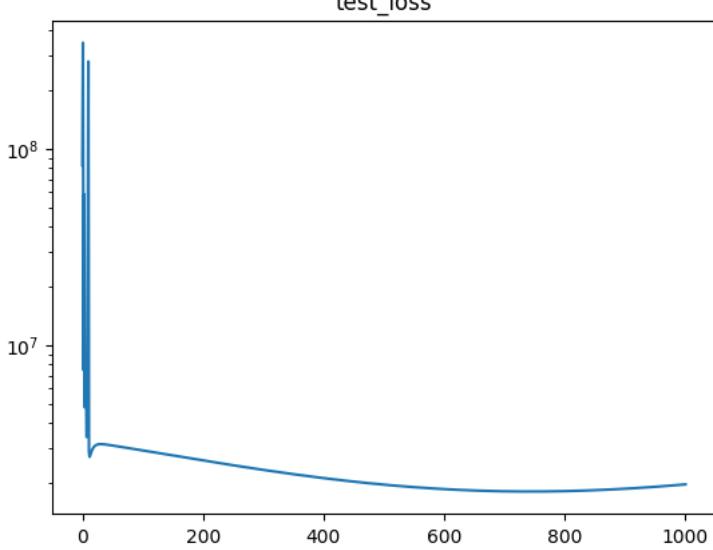
optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 0.5 , minimum_RMSE: 2343.98 , epoch: 1000 , test_loss: 17288 , train_loss: 2

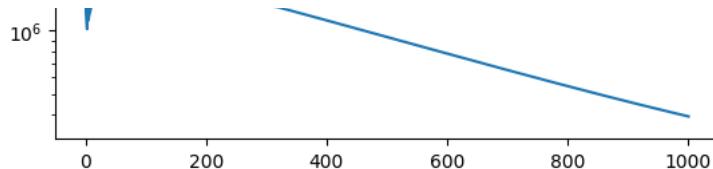
test_loss





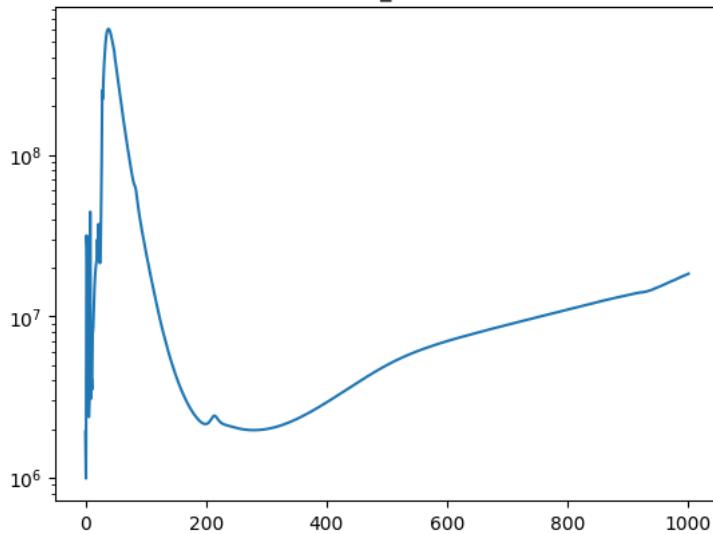
optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 0.01 , minimum_RMSE: 1339.04 , epoch: 1000 , test_loss: 1396 , train_loss:



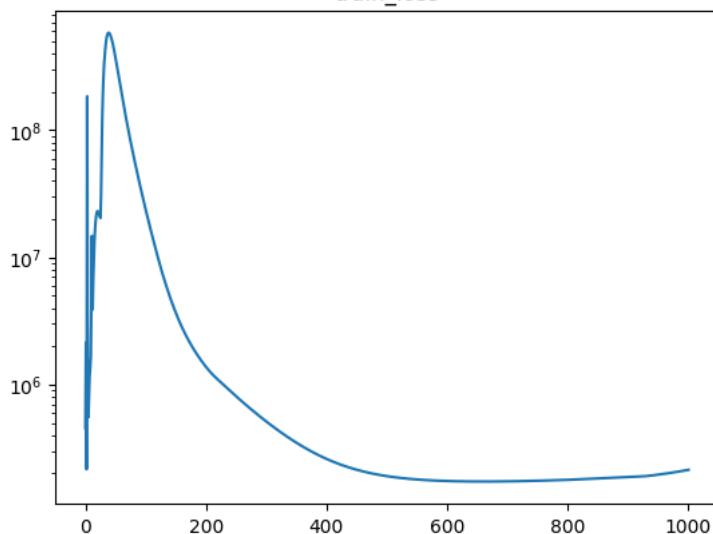


optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 0.05 , minimum_RMSE: 993.79 , epoch: 1000 , test_loss: 4280 , train_loss:

test_loss

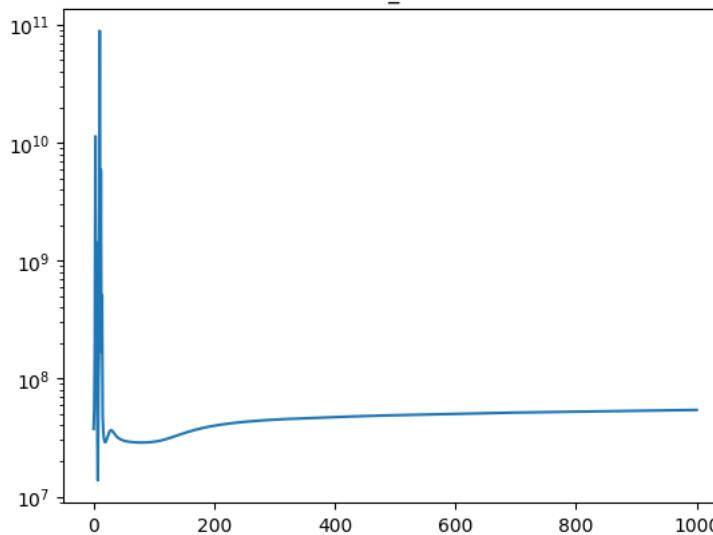


train_loss



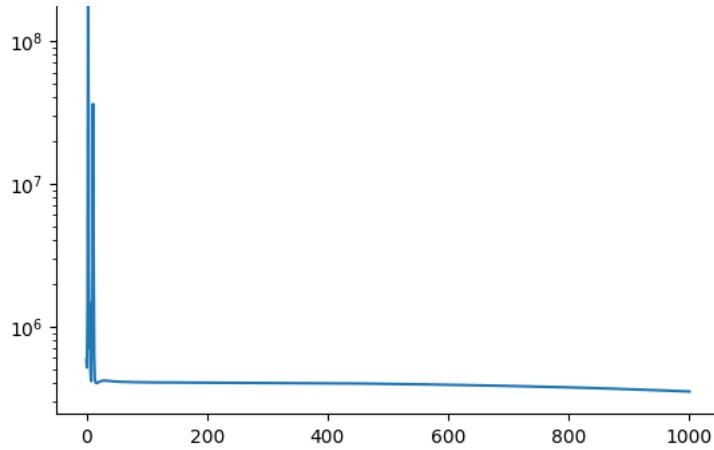
optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 0.001 , minimum_RMSE: 3701.65 , epoch: 1000 , test_loss: 7364 , train_loss:

test_loss

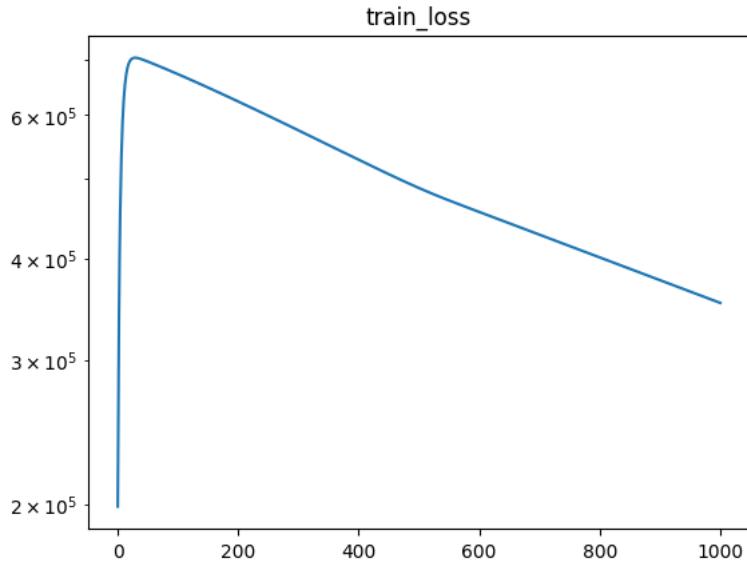
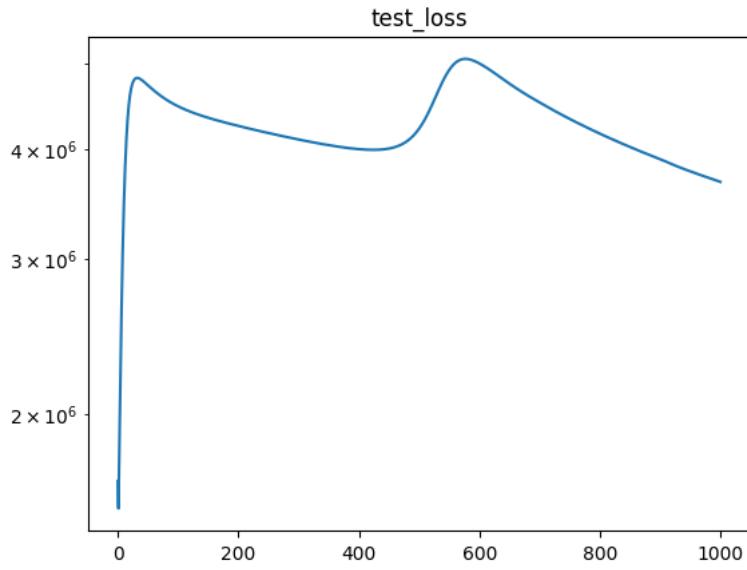


train_loss

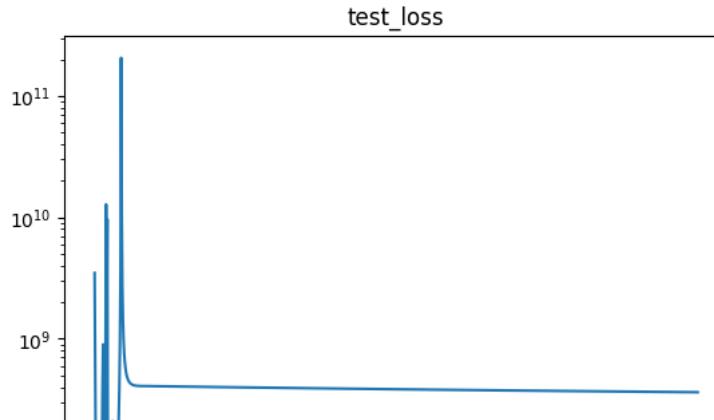


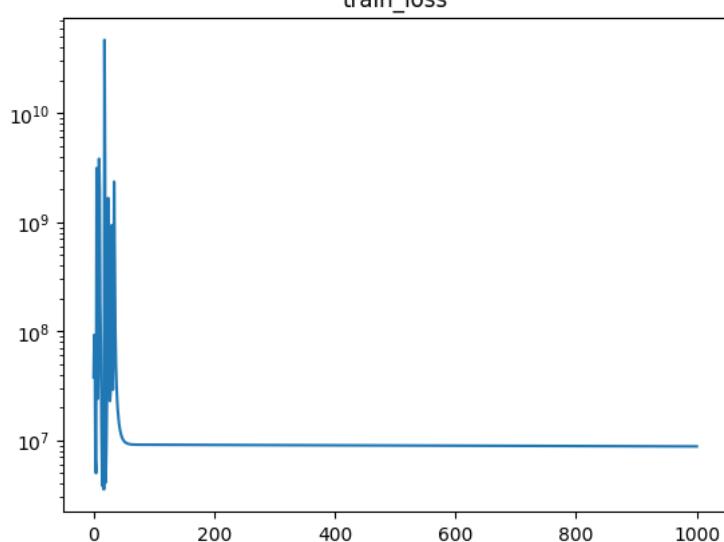
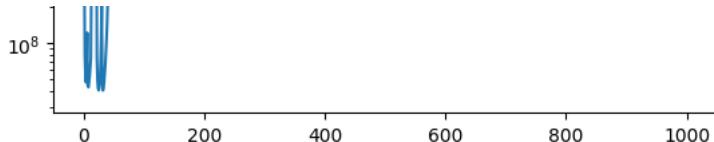


optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 0.005 , minimum_RMSE: 1250.16 , epoch: 1000 , test_loss: 1915 , train_loss:

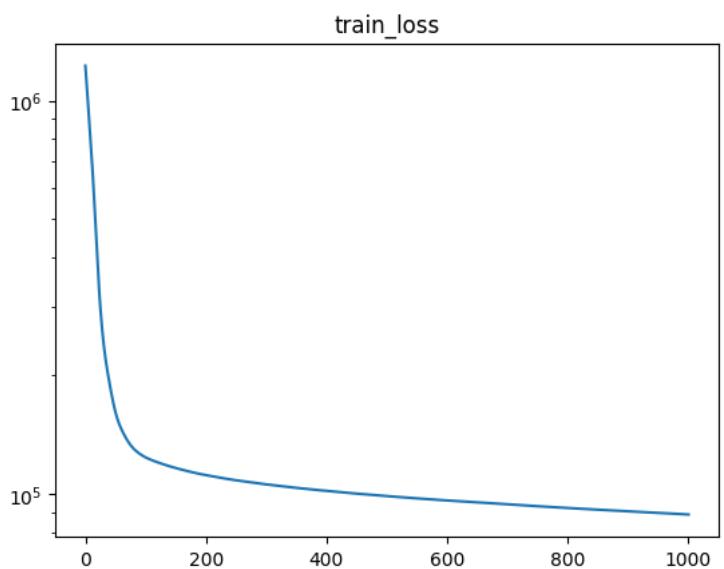
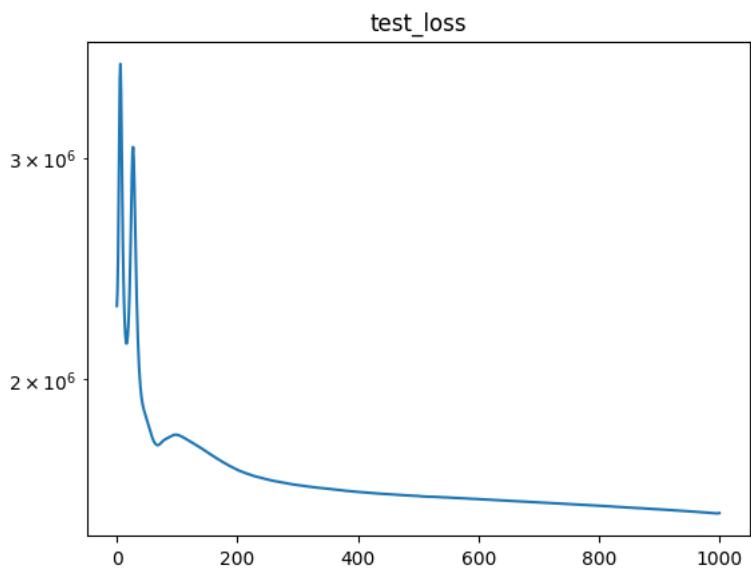


optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 0.0001 , minimum_RMSE: 6387.49 , epoch: 1000 , test_loss: 19039 , train_loss:

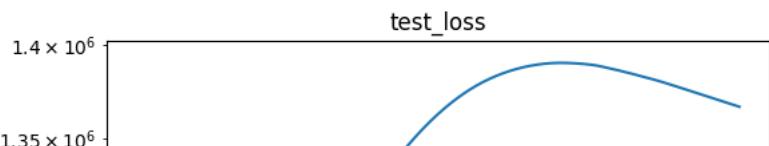


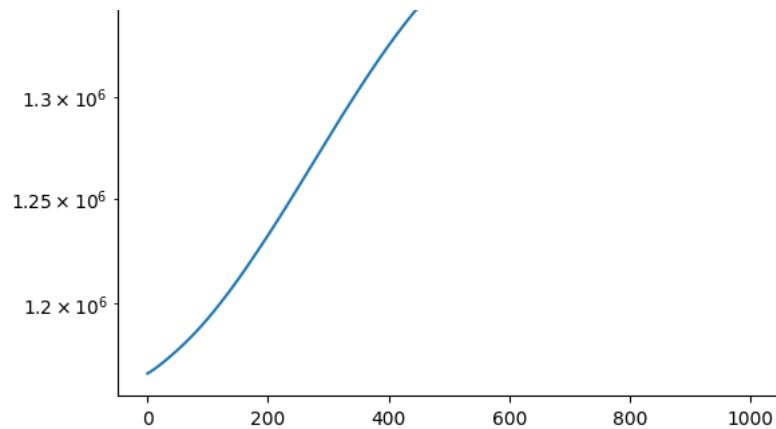


optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 0.0005 , minimum_RMSE: 1250.50 , epoch: 1000 , test_loss: 1250 , train_loss:

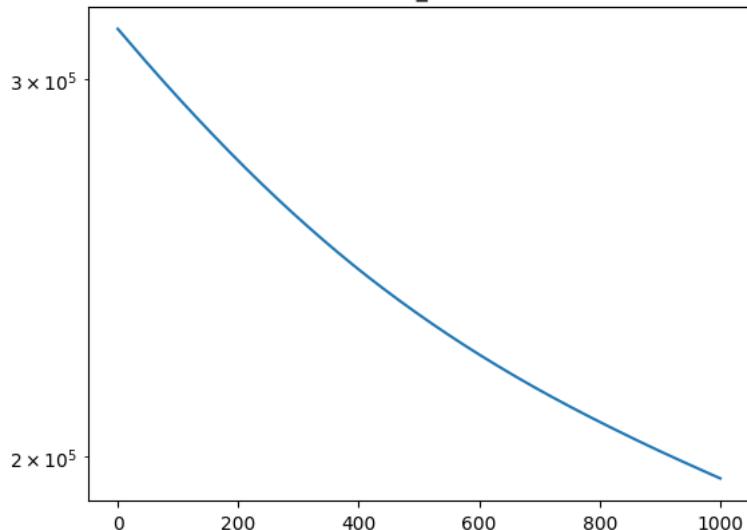


optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-05 , minimum_RMSE: 1080.77 , epoch: 1000 , test_loss: 1169 , train_loss:

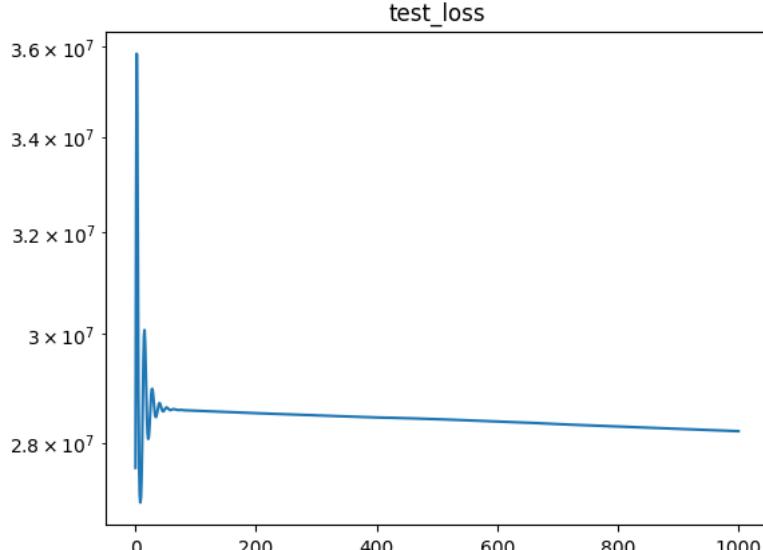




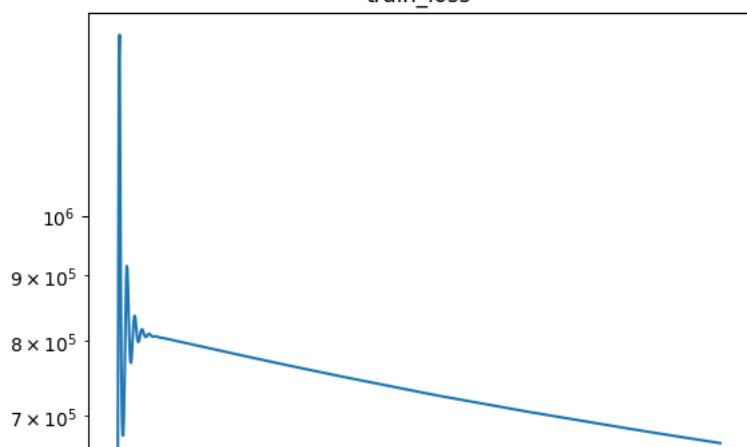
train_loss

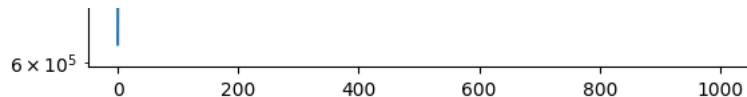


test_loss

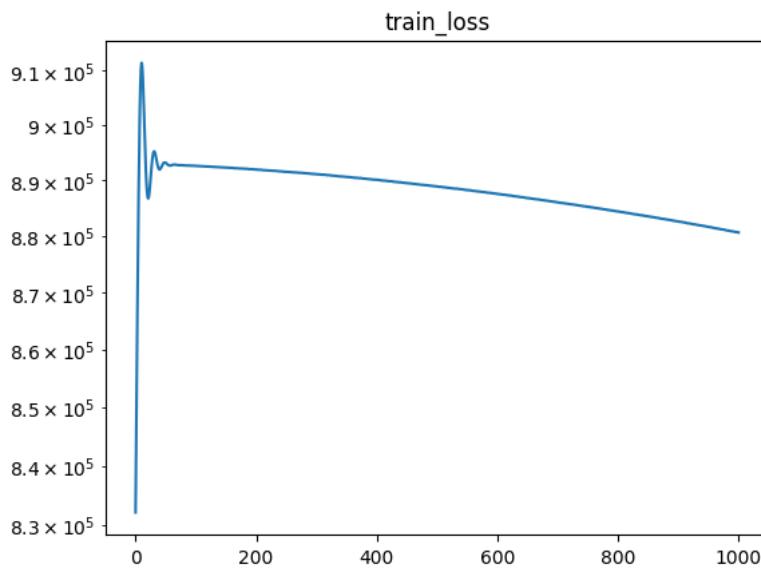
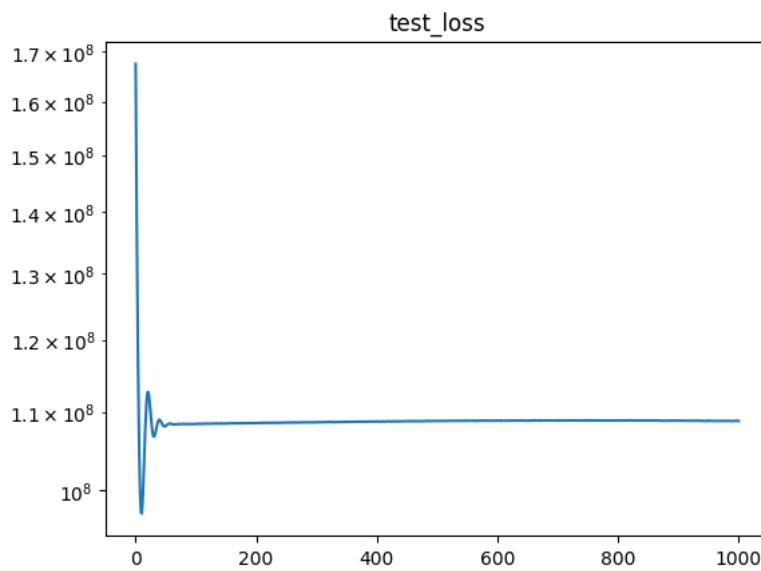


train_loss

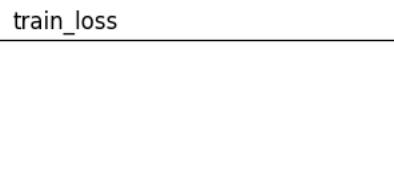
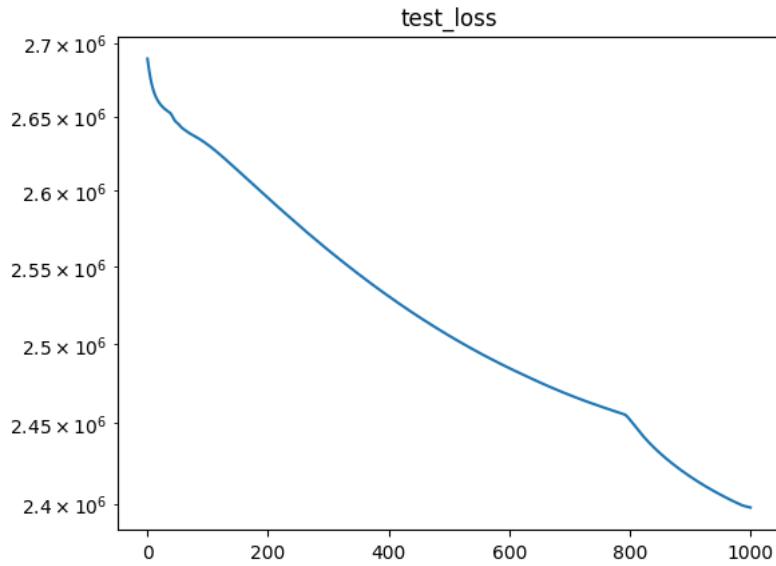


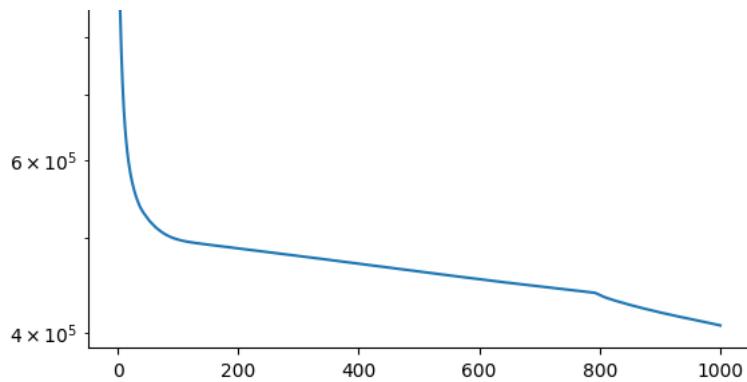


optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-06 , minimum_RMSE: 9862.93 , epoch: 1000 , test_loss: 10429 , train_loss:

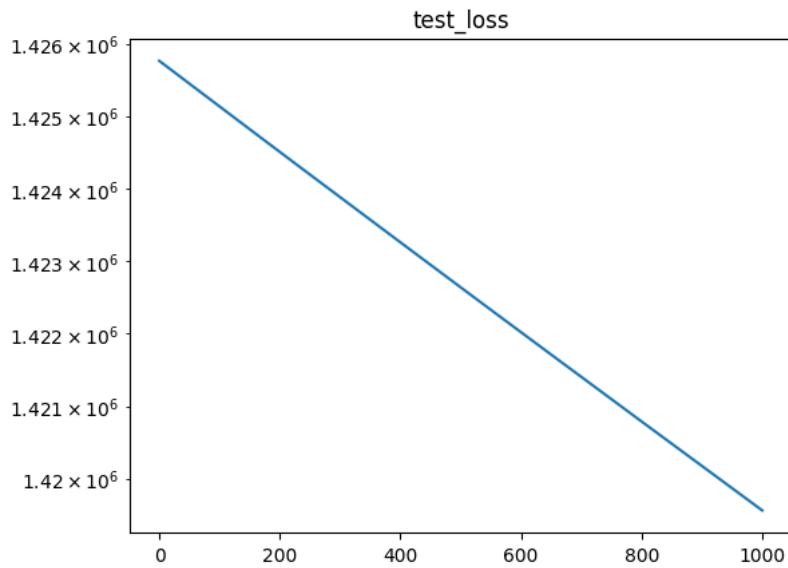


optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 5e-06 , minimum_RMSE: 1548.41 , epoch: 1000 , test_loss: 1548 , train_loss:

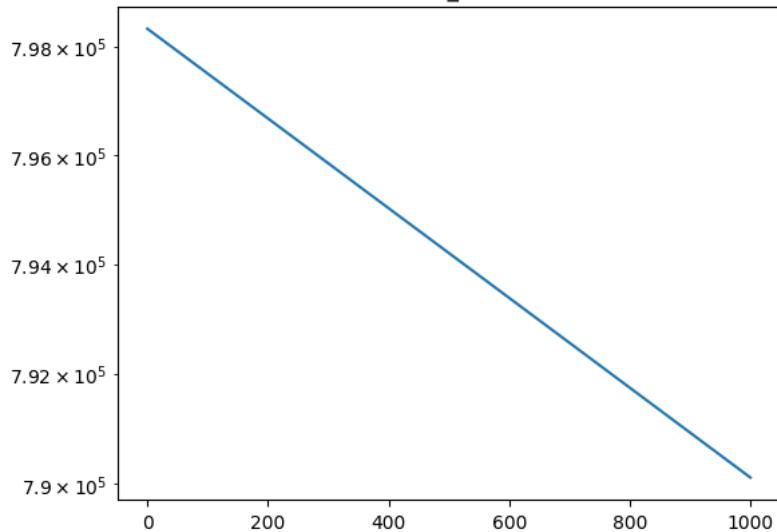




optimizer: AdamW , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-07 , minimum_RMSE: 1191.46 , epoch: 1000 , test_loss: 1191 , train_loss:

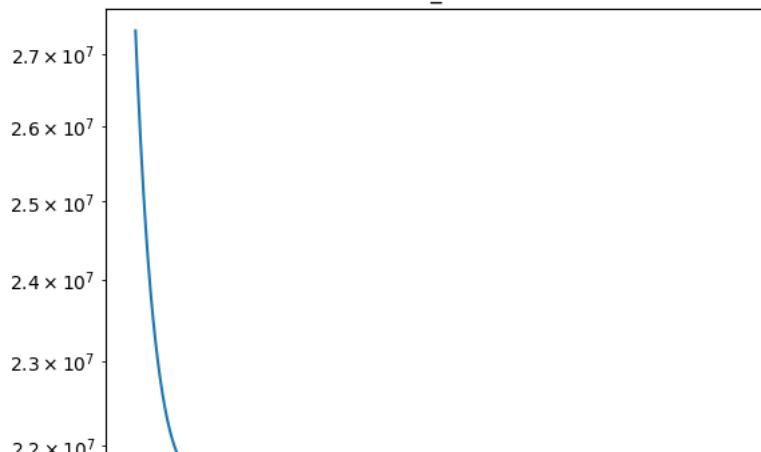


test_loss



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

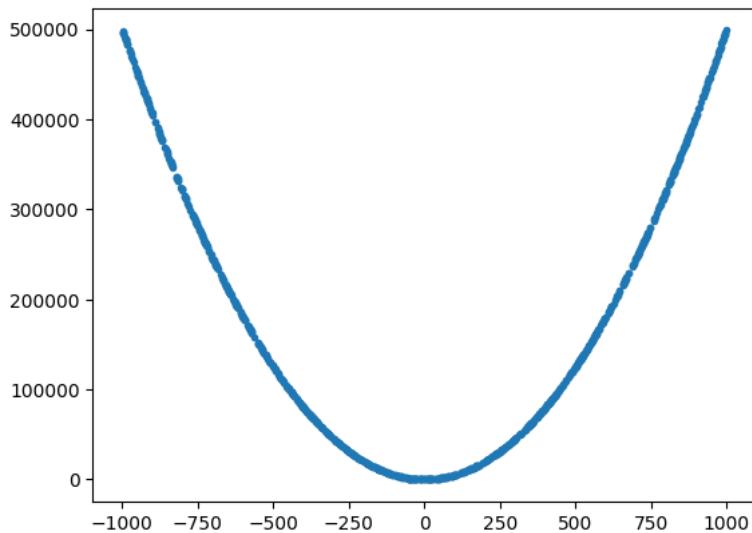
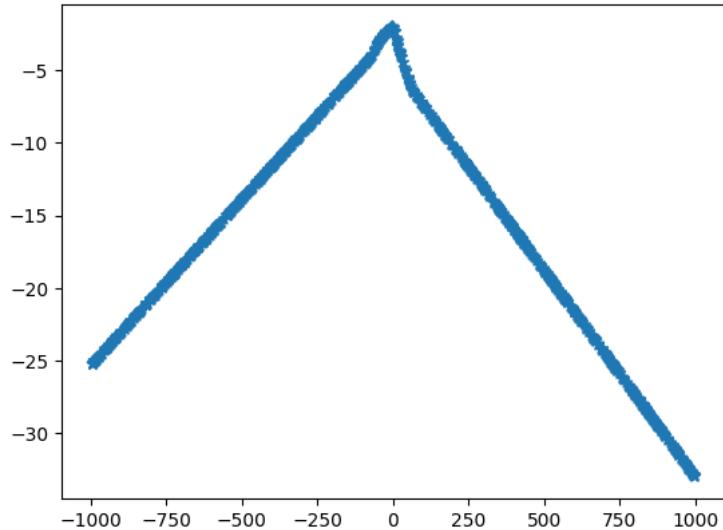
test_loss



1

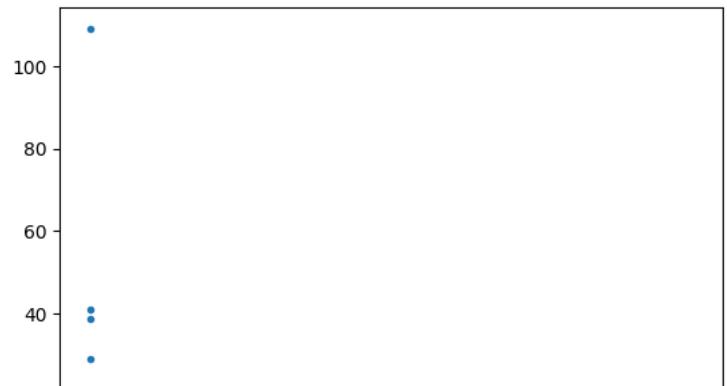
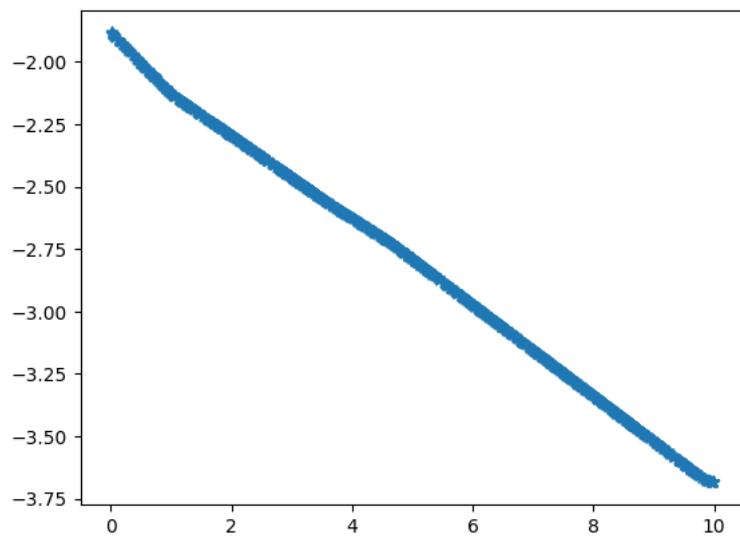


```
1 model.eval()
2 y_pred = model(X2)
3 plt.scatter(X2[:,1].cpu().detach().numpy(),y_pred.cpu().detach().numpy(), marker='*')
4 # plt.scatter(X2[:,1],y_pred, marker='*')
5 plt.show()
6
7
8 plt.scatter(X2[:,1] ,y2, marker='.', )
9 plt.show()
10
11
12 print('all done')
```



all done

```
1 model.eval()
2 y_pred = model(X3)
3 plt.scatter(X3[:,0].cpu().detach().numpy(),y_pred.cpu().detach().numpy(), marker='*')
4 # plt.scatter(X3[:,1],y_pred, marker='*')
5 plt.show()
6
7
8 plt.scatter(X3[:,0] ,y3, marker='.', )
9 plt.show()
10
11
12 print('all done')
```



```
1 plt.plot(history)
2 plt.yscale('log')
3 plt.title('test_loss')
4 plt.show()
5
6 plt.plot(history_train)
7 plt.title('train_loss')
8 plt.yscale('log')
9 plt.show()
```