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

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

```

```

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

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


```

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


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


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


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

```
        y3.append( (X3[-1][-1]**2) / (2*X3[-1][-2])  )


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

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

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

```


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

```
−1000  −750  −500  −250     0     250     500     750     1000
```

```
1 model
```

```
Sequential(
  (0): Linear(in_features=2, out_features=16, bias=True)
  (1): LeakyReLU(negative_slope=0.01)
  (2): Linear(in_features=16, out_features=8, bias=True)
  (3): LeakyReLU(negative_slope=0.01)
  (4): Linear(in_features=8, out_features=1, bias=True)
)
```

```
1 make_model
```

```
'model = nn.Sequential(nn.Linear(dim_list[2], dim_list[16]), nn.LeakyReLU(),nn.Linear(dim_list
```

```
1 make_model
```

```
'model = nn.Sequential(nn.Linear(dim_list[0], dim_list[1]), nn.LeakyReLU(),nn.Linear(dim_list
```

```
1 model
```

```
Sequential(
  (0): Linear(in_features=2, out_features=16, bias=True)
  (1): LeakyReLU(negative_slope=0.01)
  (2): Linear(in_features=16, out_features=8, bias=True)
  (3): LeakyReLU(negative_slope=0.01)
  (4): Linear(in_features=8, out_features=1, bias=True)
)
```

```
1
```

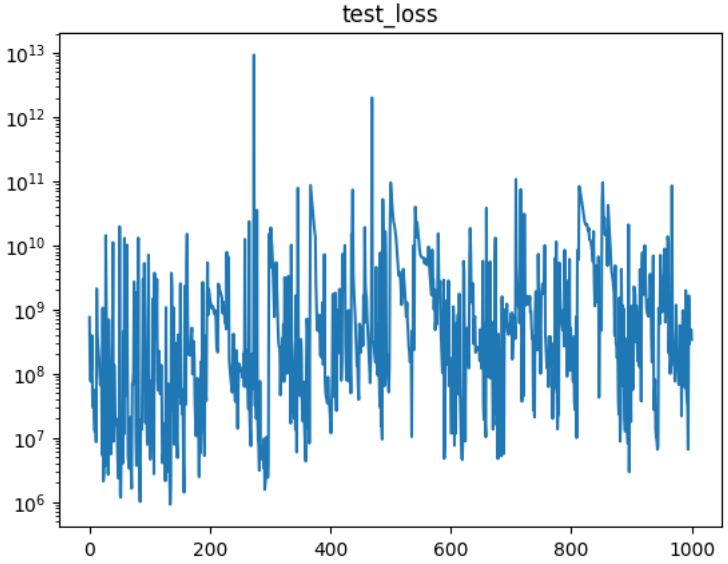
```
1  #default
2
3  """
```

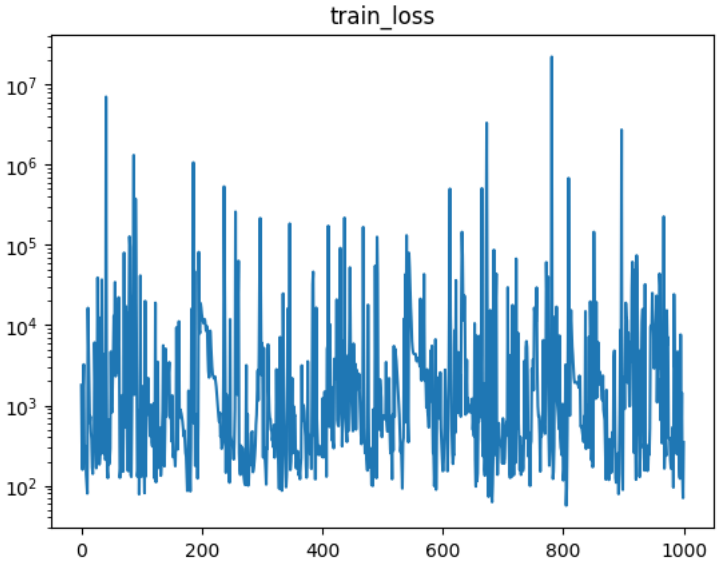
```
Bsize: 500 learningRate: 0.03 RMSE: 111.31 1000 loss: 515.4253401318177

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


dim_list = [2, 16, 8, 1]



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


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

            make_model = 'model = nn.Sequential('
            for layer_num in range( len(dim_list) - 1 ):
```

```
            if layer_num == len(dim_list) - 2:
                make_model = make_model + 'nn.Linear(dim_list[' + str(layer_num) + '], dim_list[' + str(layer_num+1) + '])'
            else:
                make_model = make_model + 'nn.Linear(dim_list[' + str(layer_num) + '], dim_list[' + str(layer_num+1) + ']), nn.LeakyReLU(),'

        exec(make_model)
        #make_model = make_model +

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

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

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

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


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

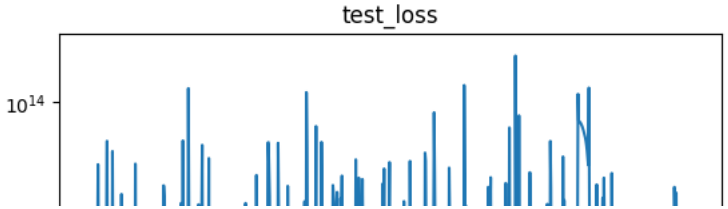
```
                loss = float(loss)

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

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

                mse = float(mse)

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

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

            # restore model and return best accuracy
            model.load_state_dict(best_weights)
            # print("MSE: %.2f" % best_mse)
            # print("RMSE: %.2f" % np.sqrt(best_mse))
            print('optimizer:', opt, ', n_of_data:',N, ', Bsize:', Bsize ,', learningRate:', learningRate , ", minimum_RMSE: %.2f" % np.sqrt(best_mse), ', epoch: %5d' % epoch, ', test_loss: %5d' % np.sqrt(mse), ', tr
            # print(y_pred[:10])
            # print(y_test[:10])
            plt.plot(history)
            plt.yscale('log')
            plt.title('test_loss')
            plt.show()

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

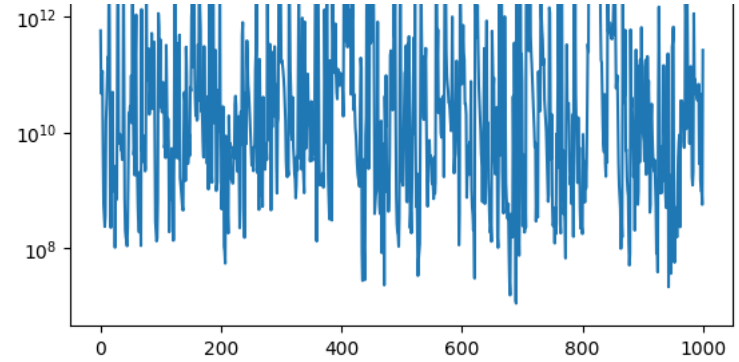
```
optimizer: RMSprop , n_of_data: 1000 , Bsize: 10 , learningRate: 0.1 , minimum_RMSE: 956.44 , epoch:  1000 , test_loss: 18382 , train_loss:
```

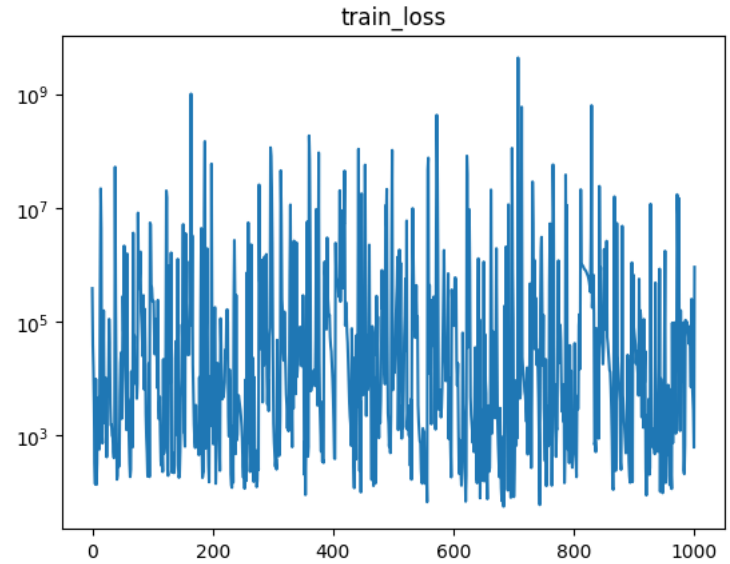




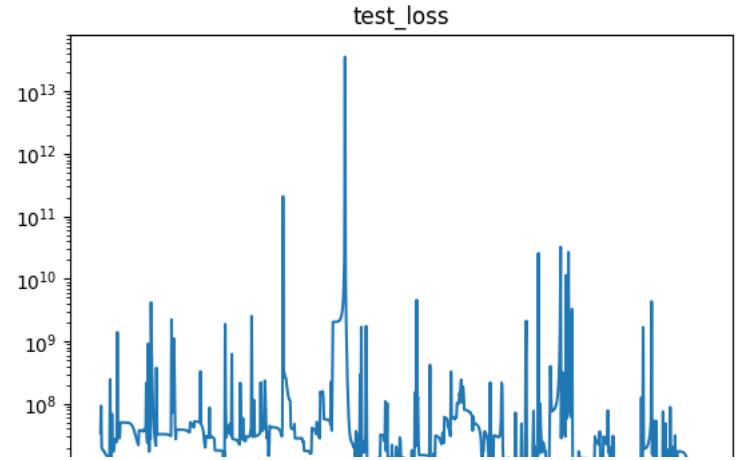

```
optimizer: RMSprop , n_of_data: 1000 , Bsize: 10 , learningRate: 0.5 , minimum_RMSE: 3342.36 , epoch:  1000 , test_loss: 512792 , train_loss
```

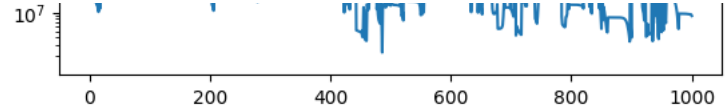


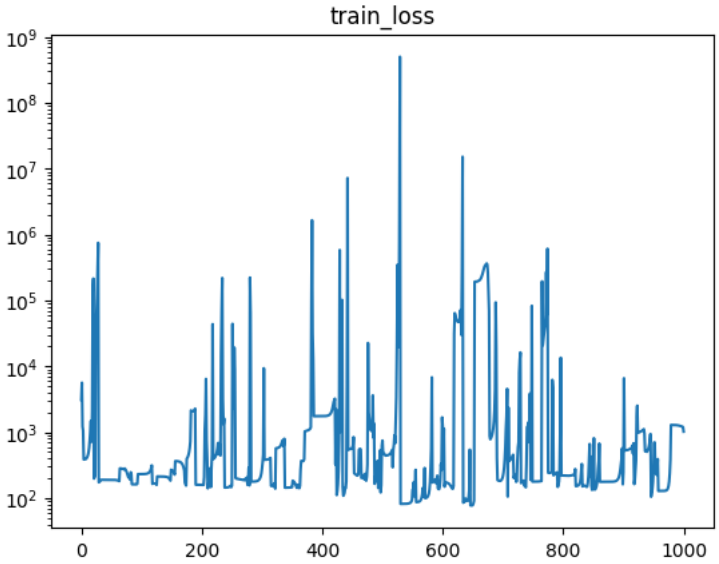




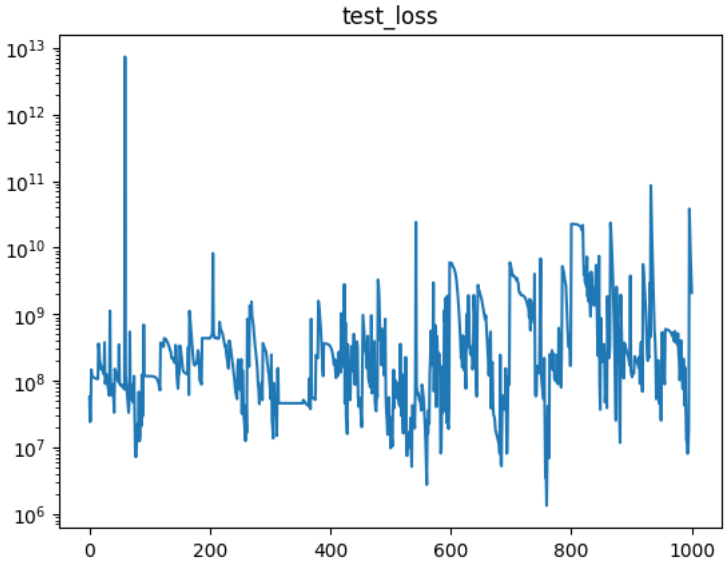
optimizer: RMSprop , n_of_data: 1000 , Bsize: 10 , learningRate: 0.01 , minimum_RMSE: 1513.55 , epoch: 1000 , test_loss: 2967 , train_loss

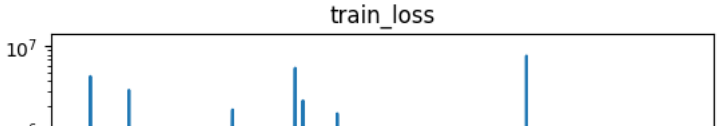


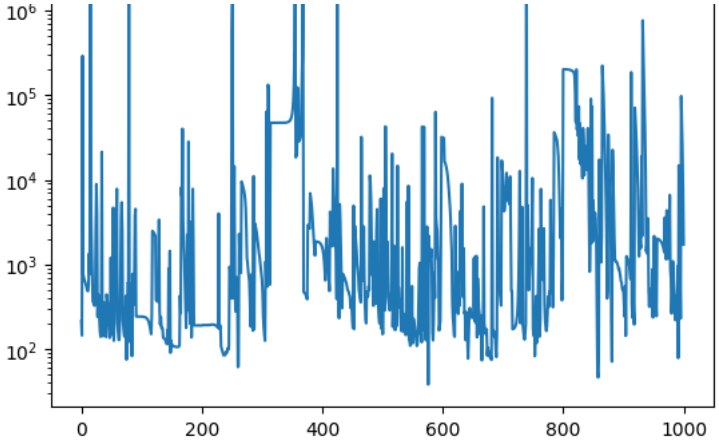




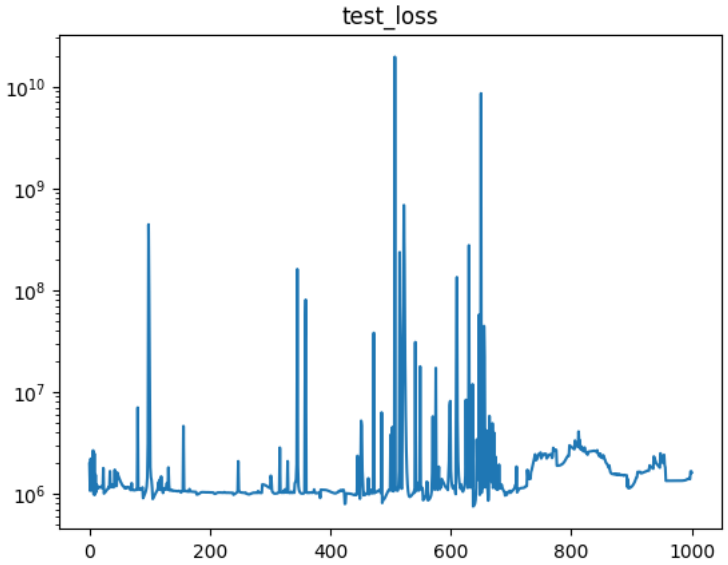
optimizer: RMSprop , n_of_data: 1000 , Bsize: 10 , learningRate: 0.05 , minimum_RMSE: 1154.62 , epoch: 1000 , test_loss: 45808 , train_loss

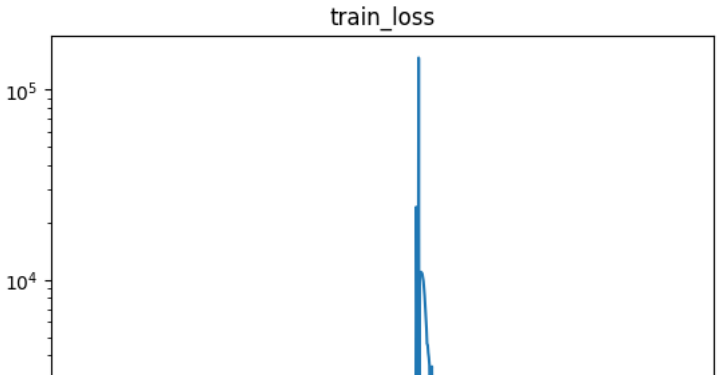


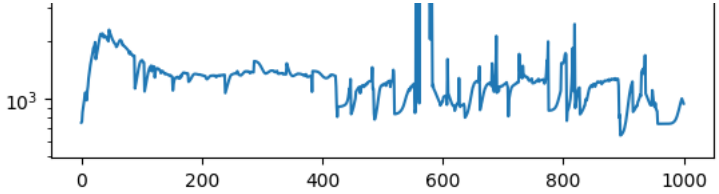




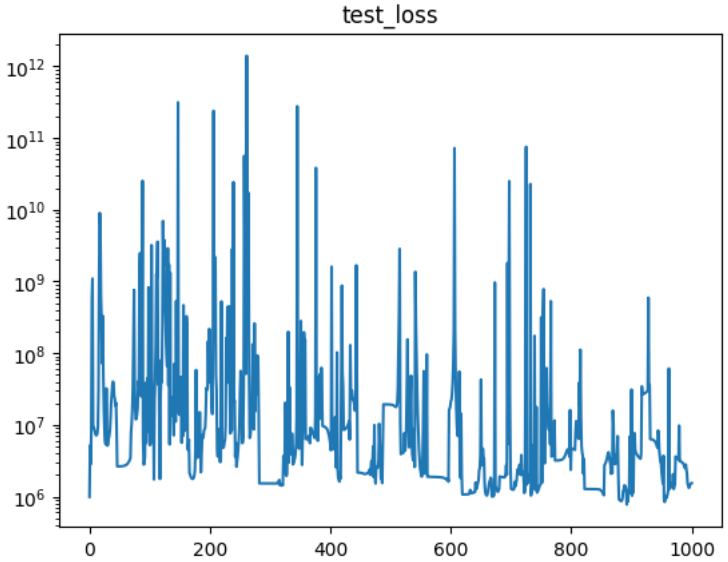
optimizer: RMSprop , n_of_data: 1000 , Bsize: 10 , learningRate: 0.001 , minimum_RMSE: 864.86 , epoch: 1000 , test_loss: 1267 , train_loss

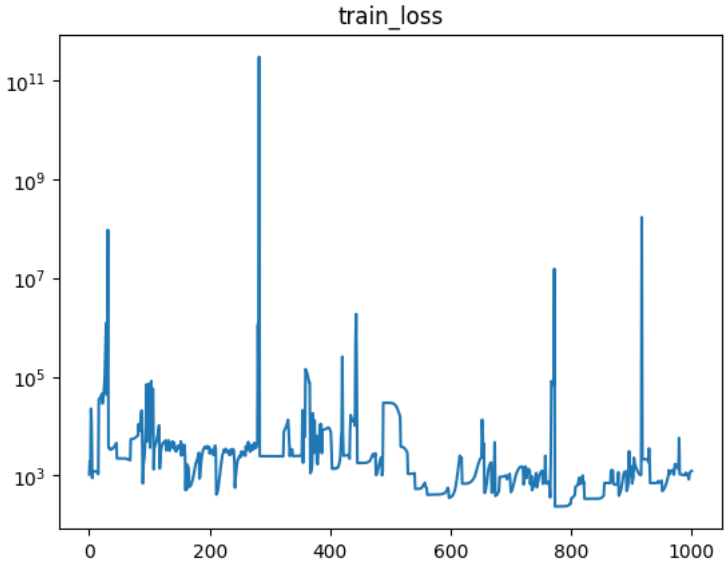






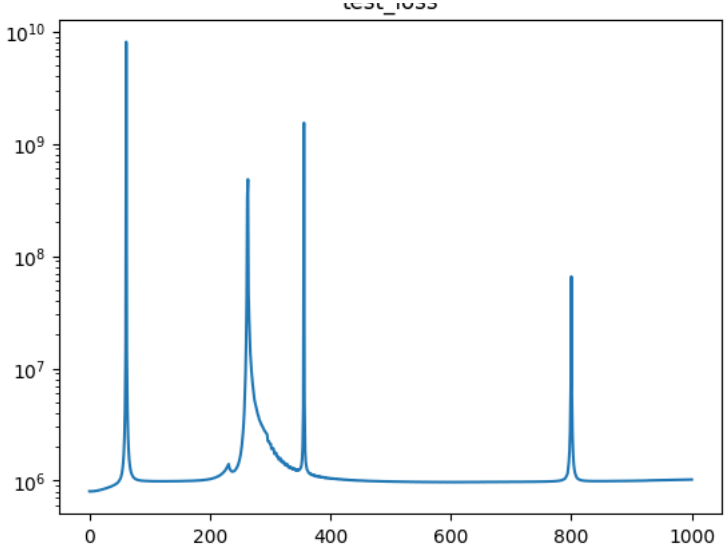
optimizer: RMSprop , n_of_data: 1000 , Bsize: 10 , learningRate: 0.005 , minimum_RMSE: 881.07 , epoch: 1000 , test_loss: 1242 , train_loss

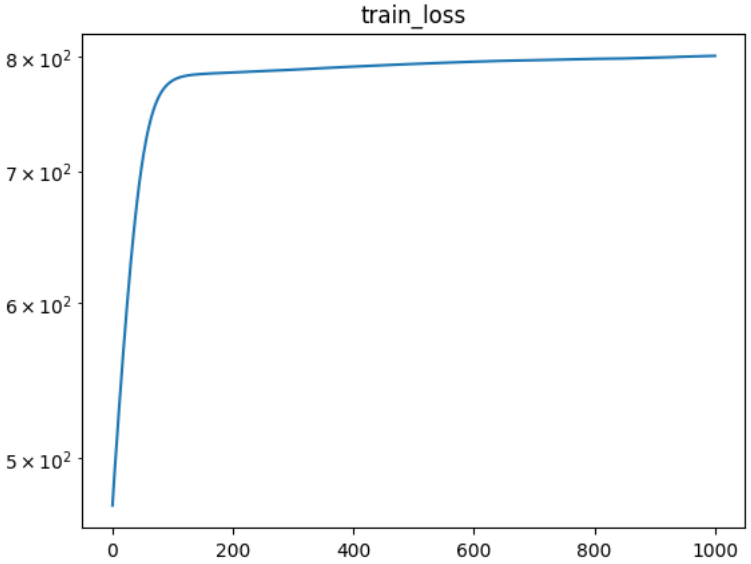




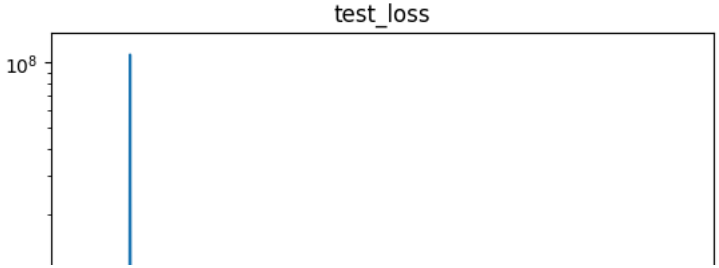
optimizer: RMSprop , n_of_data: 1000 , Bsize: 10 , learningRate: 0.0001 , minimum_RMSE: 896.03 , epoch: 1000 , test_loss: 1011 , train_los

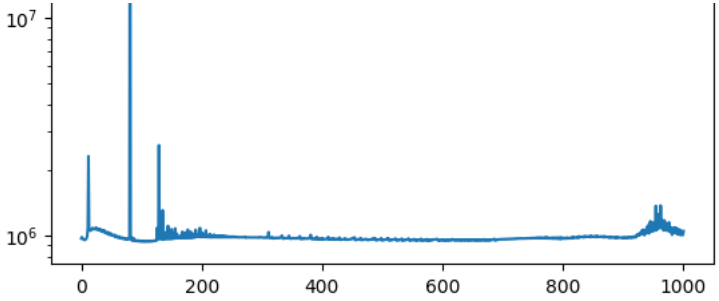
test_loss

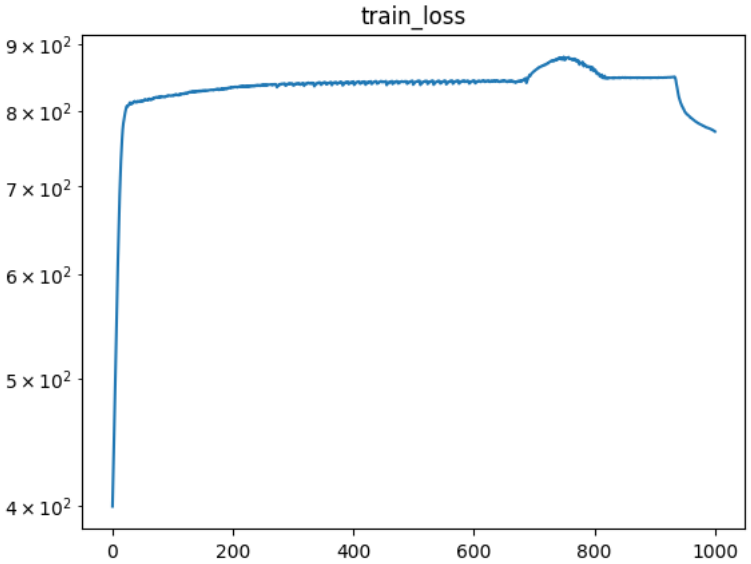




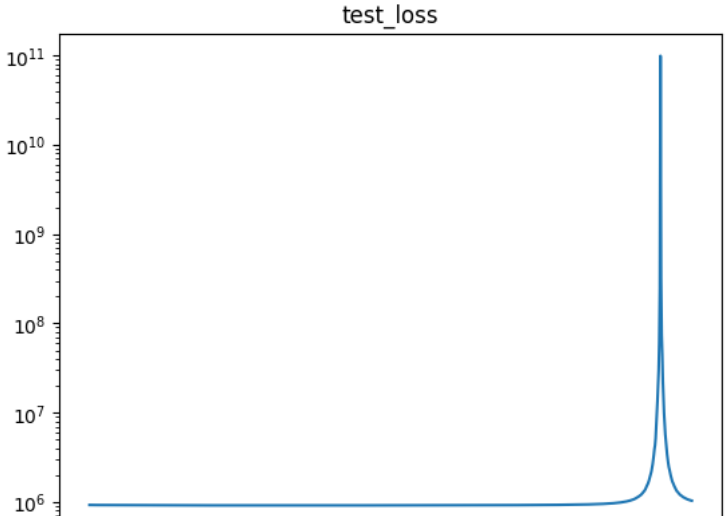
optimizer: RMSprop , n_of_data: 1000 , Bsize: 10 , learningRate: 0.0005 , minimum_RMSE: 967.03 , epoch: 1000 , test_loss: 1022 , train_los

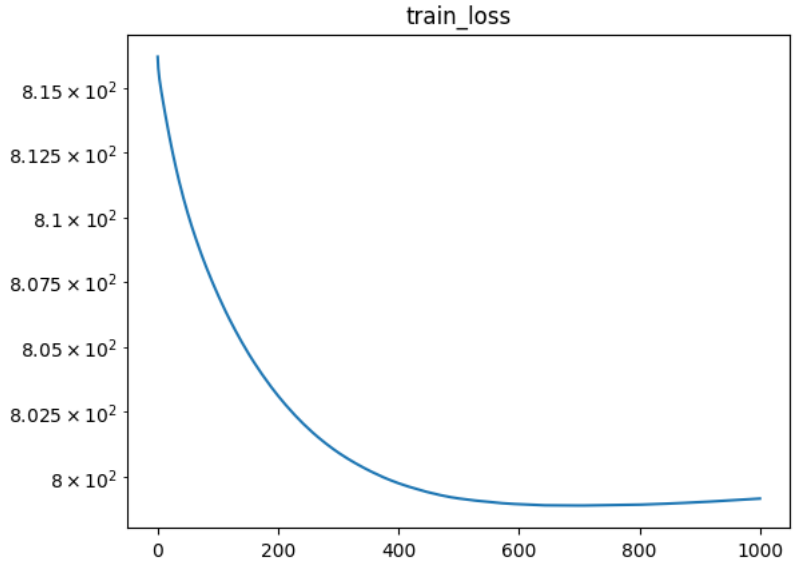




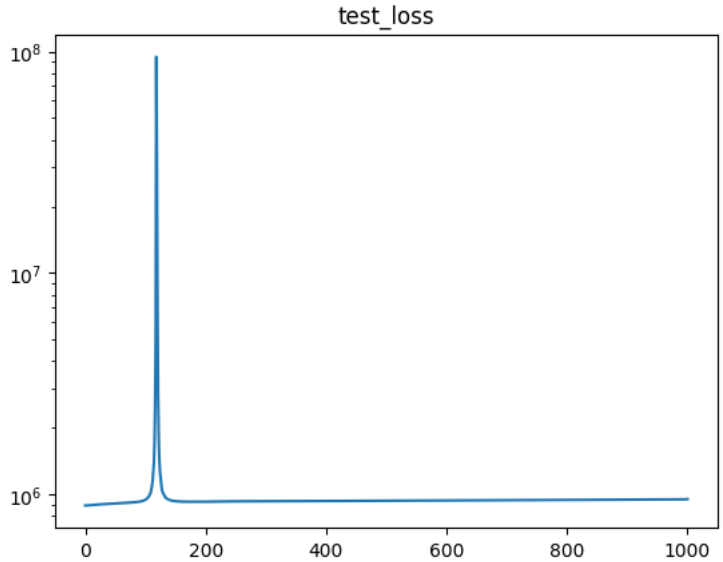


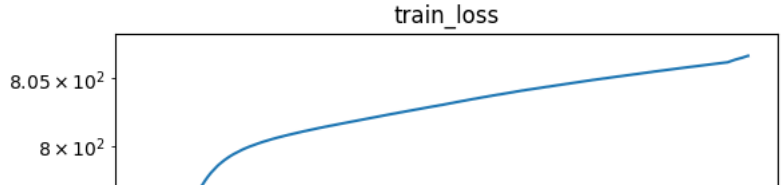
optimizer: RMSprop , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-05 , minimum_RMSE: 958.37 , epoch: 1000 , test_loss: 1018 , train_loss

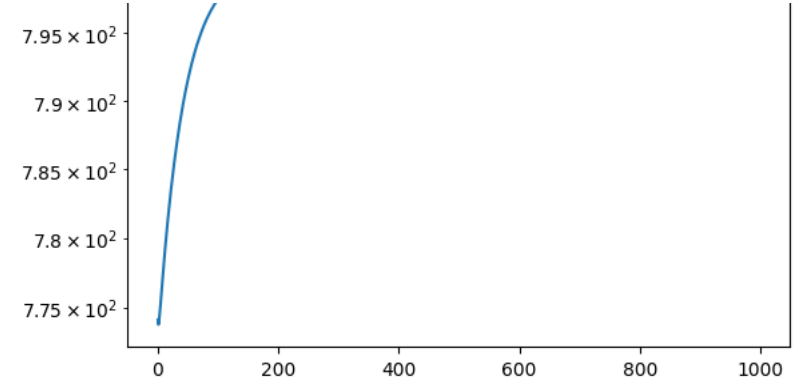




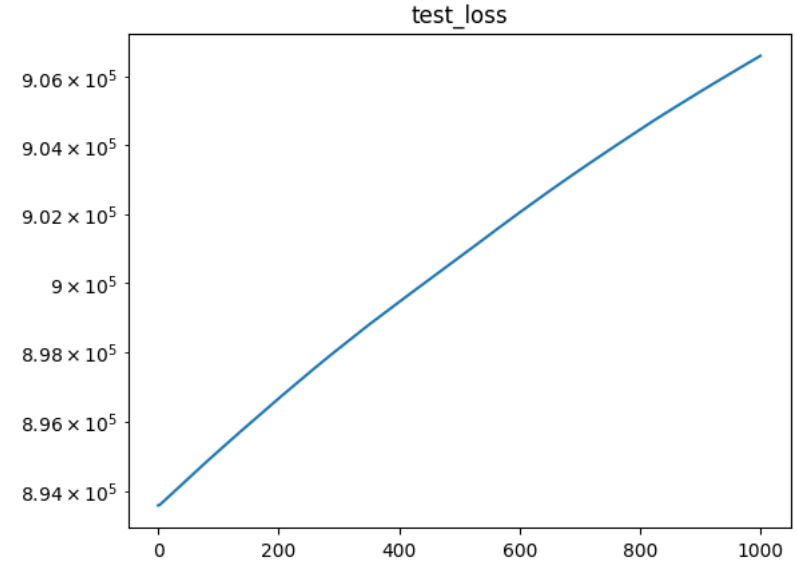
optimizer: RMSprop , n_of_data: 1000 , Bsize: 10 , learningRate: 5e-05 , minimum_RMSE: 945.54 , epoch: 1000 , test_loss: 977 , train_loss

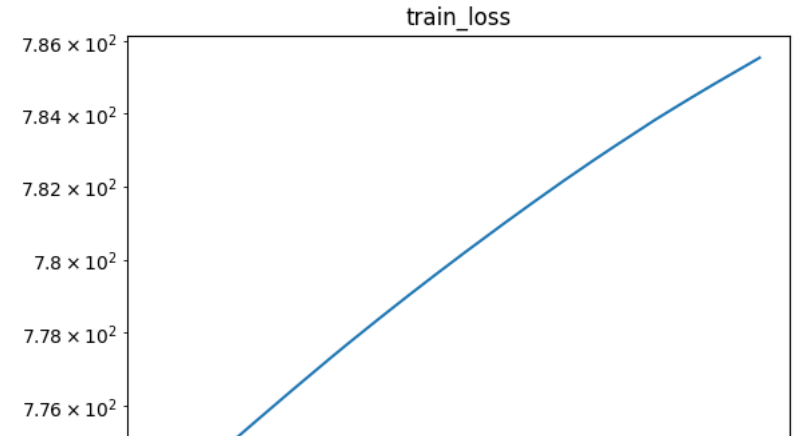


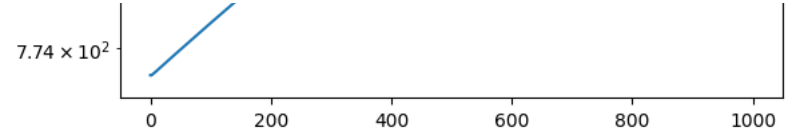




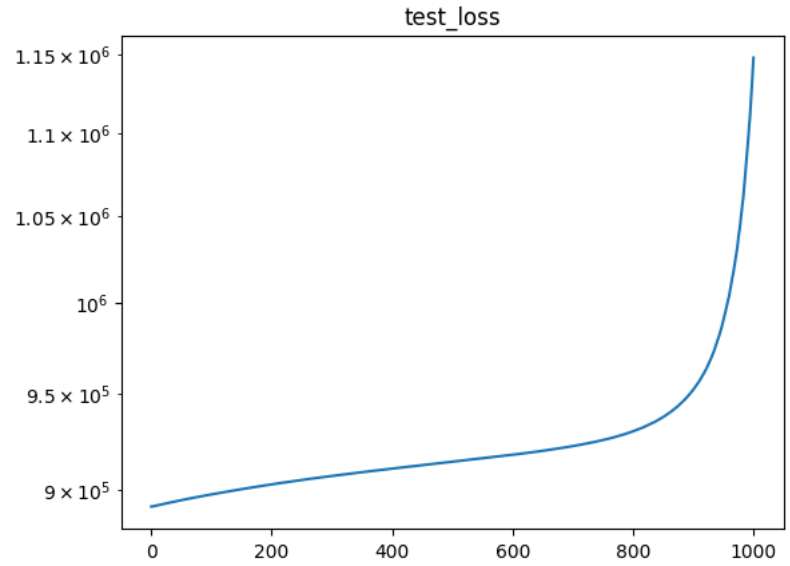
optimizer: RMSprop , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-06 , minimum_RMSE: 945.30 , epoch: 1000 , test_loss: 952 , train_loss

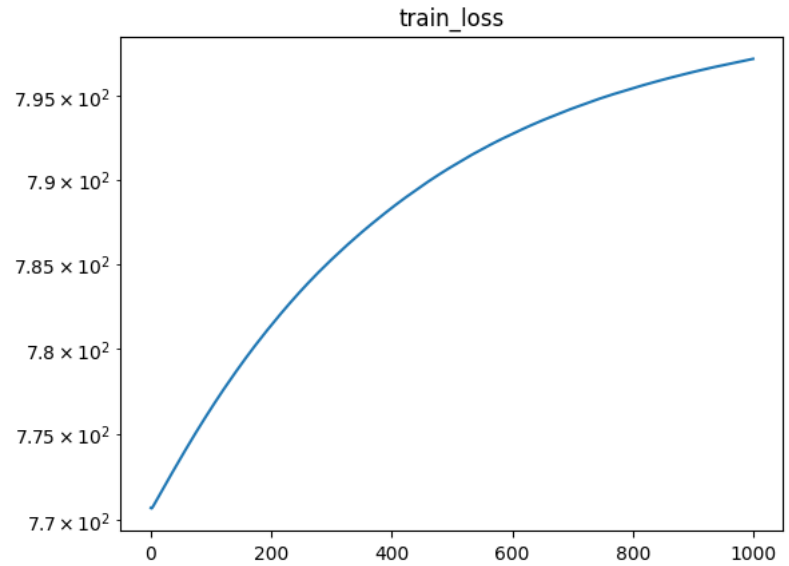




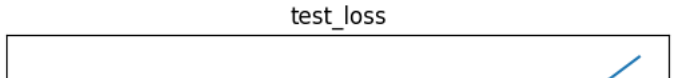


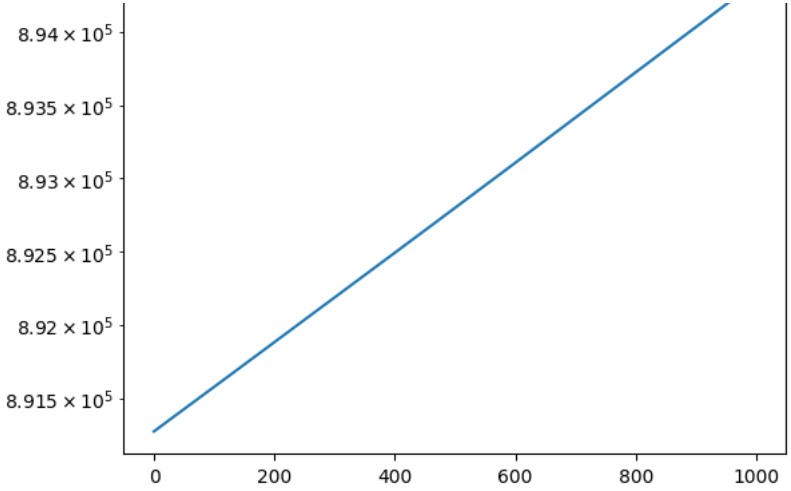
optimizer: RMSprop , n_of_data: 1000 , Bsize: 10 , learningRate: 5e-06 , minimum_RMSE: 944.10 , epoch: 1000 , test_loss: 1071 , train_loss

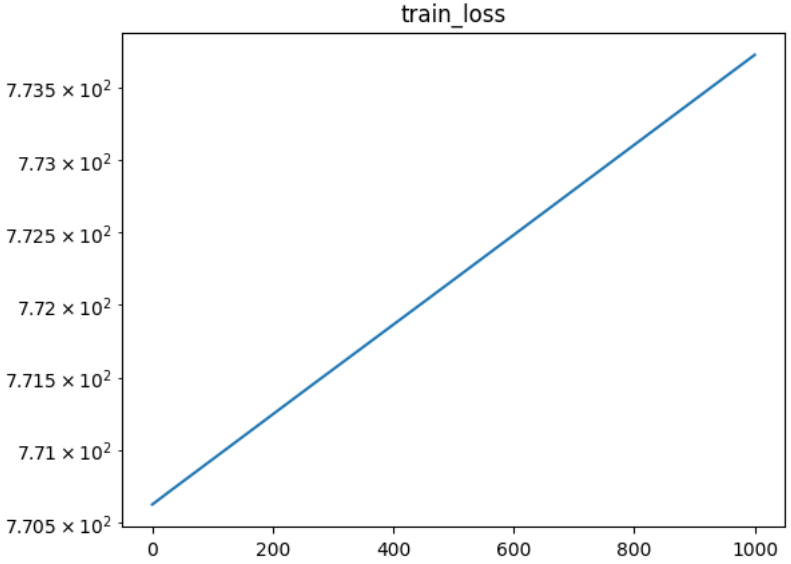




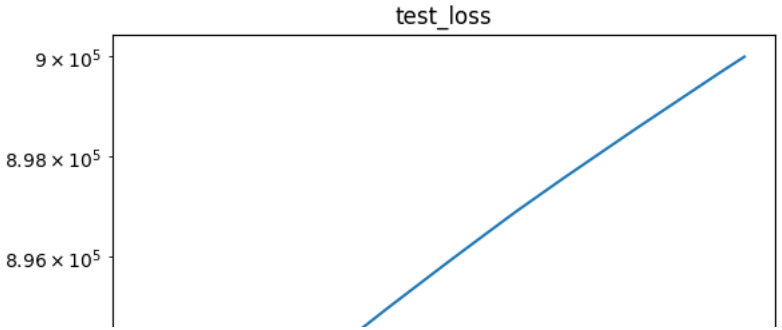
optimizer: RMSprop , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-07 , minimum_RMSE: 944.07 , epoch: 1000 , test_loss: 945 , train_loss

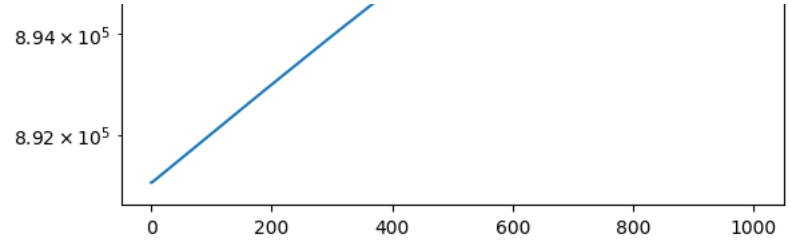


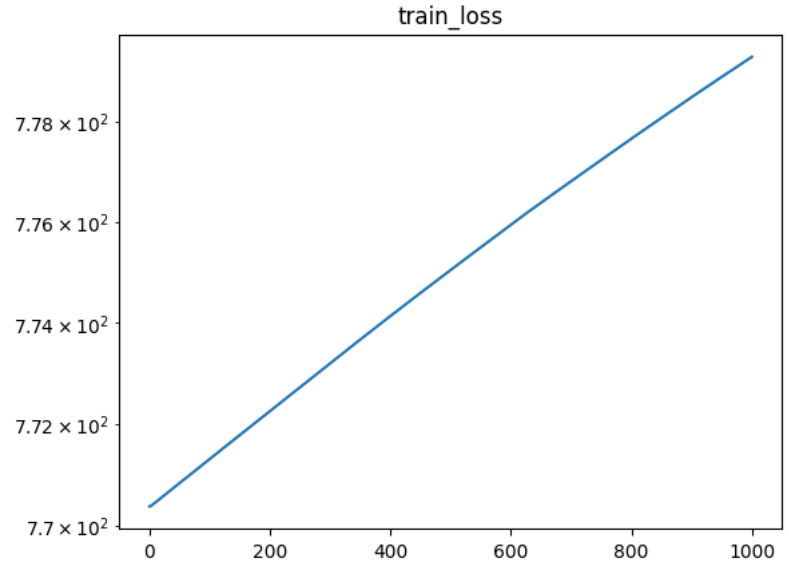




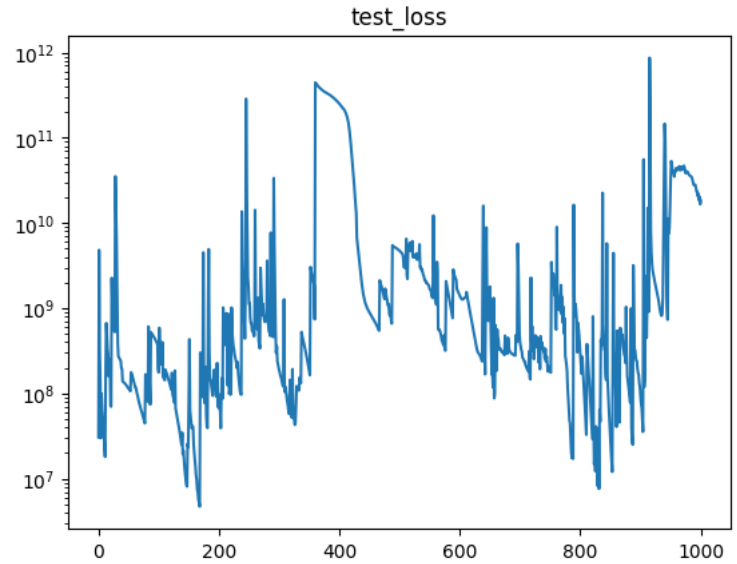
optimizer: RMSprop , n_of_data: 1000 , Bsize: 10 , learningRate: 5e-07 , minimum_RMSE: 943.96 , epoch: 1000 , test_loss: 948 , train_loss

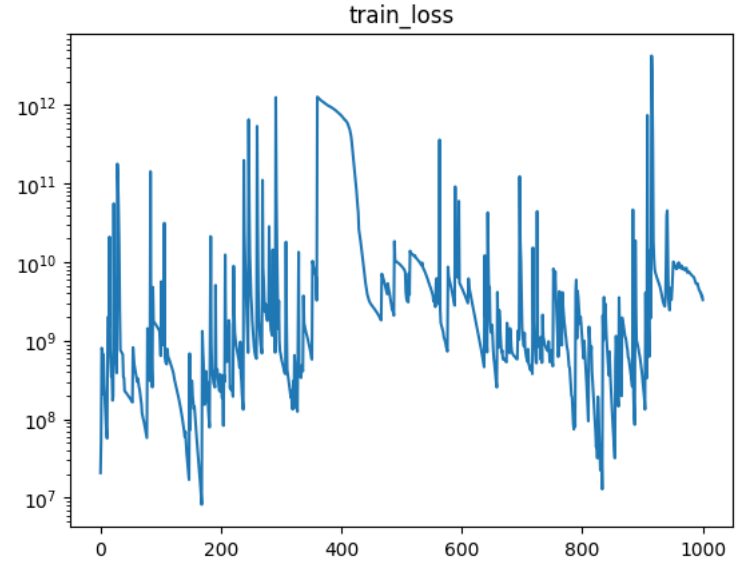




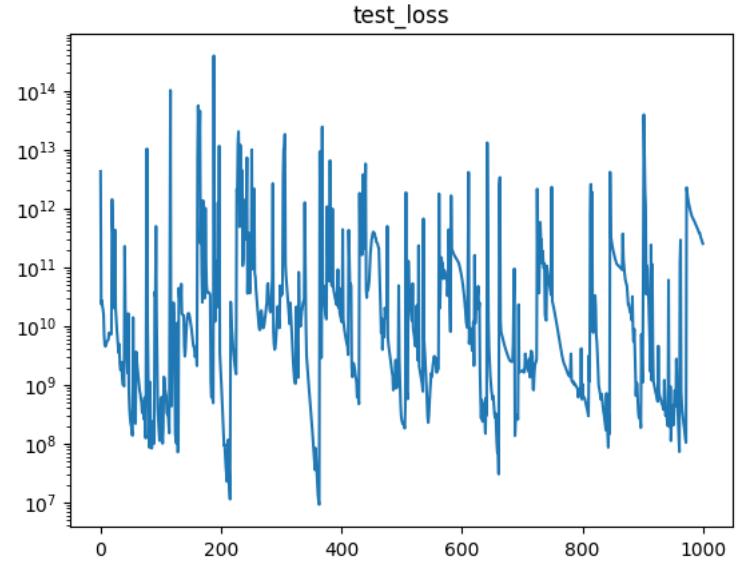


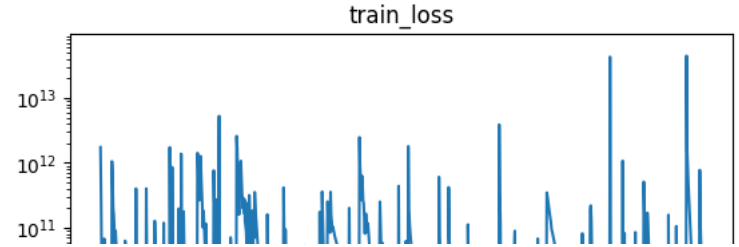
optimizer: RMSprop , n_of_data: 1000 , Bsize: 50 , learningRate: 0.1 , minimum_RMSE: 2170.87 , epoch: 1000 , test_loss: 135551 , train_loss

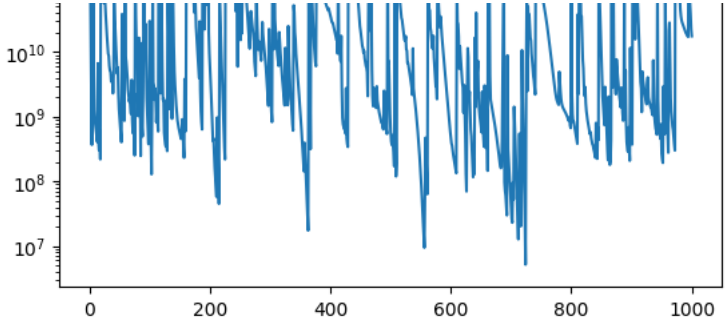




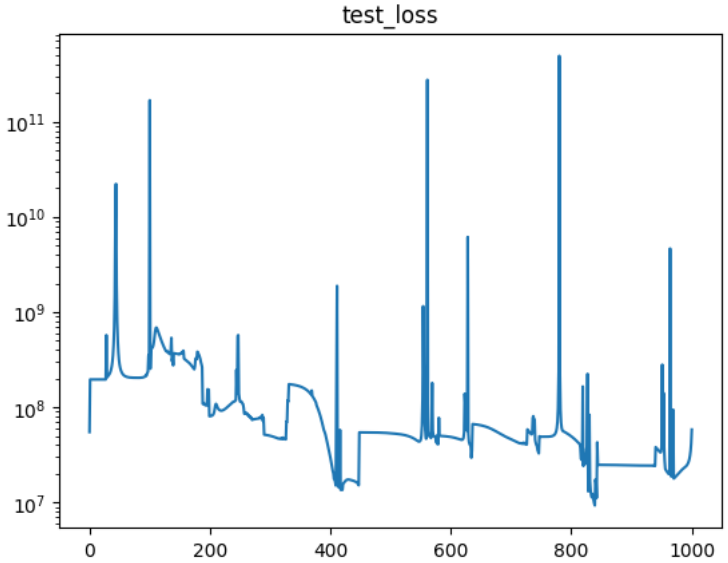
optimizer: RMSprop , n_of_data: 1000 , Bsize: 50 , learningRate: 0.5 , minimum_RMSE: 3031.88 , epoch: 1000 , test_loss: 500819 , train_loss

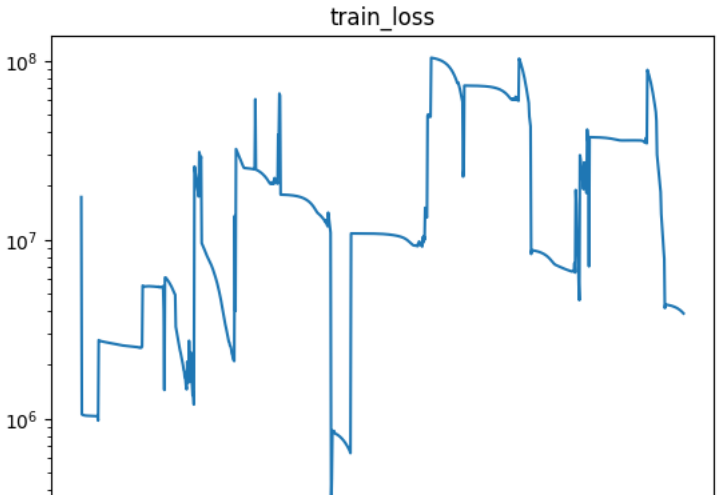




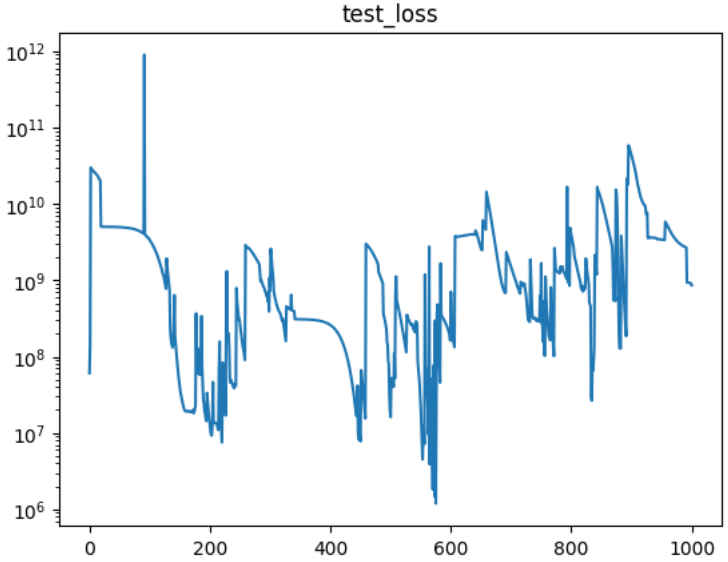


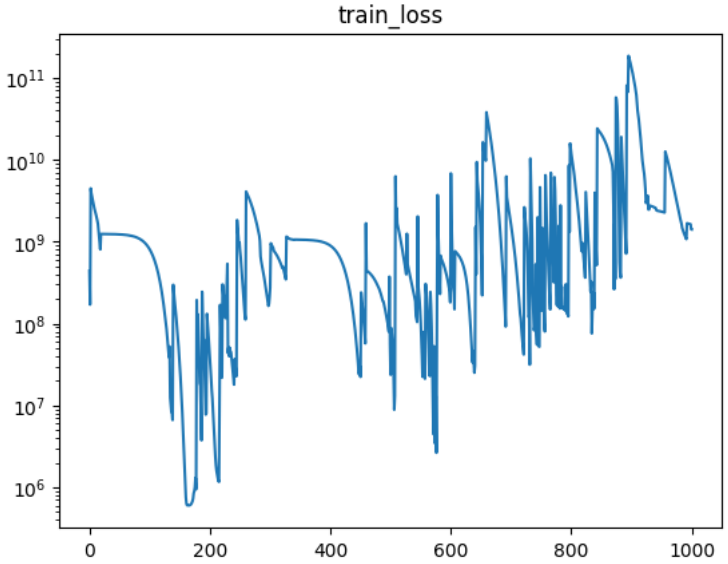
optimizer: RMSprop , n_of_data: 1000 , Bsize: 50 , learningRate: 0.01 , minimum_RMSE: 3051.76 , epoch: 1000 , test_loss: 7634 , train_loss



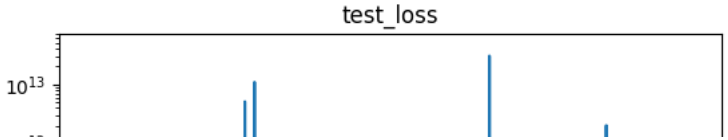


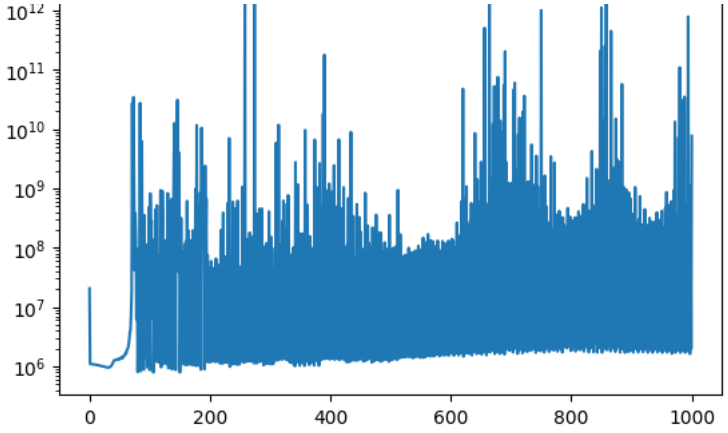
optimizer: RMSprop , n_of_data: 1000 , Bsize: 50 , learningRate: 0.05 , minimum_RMSE: 1092.17 , epoch: 1000 , test_loss: 29389 , train_loss

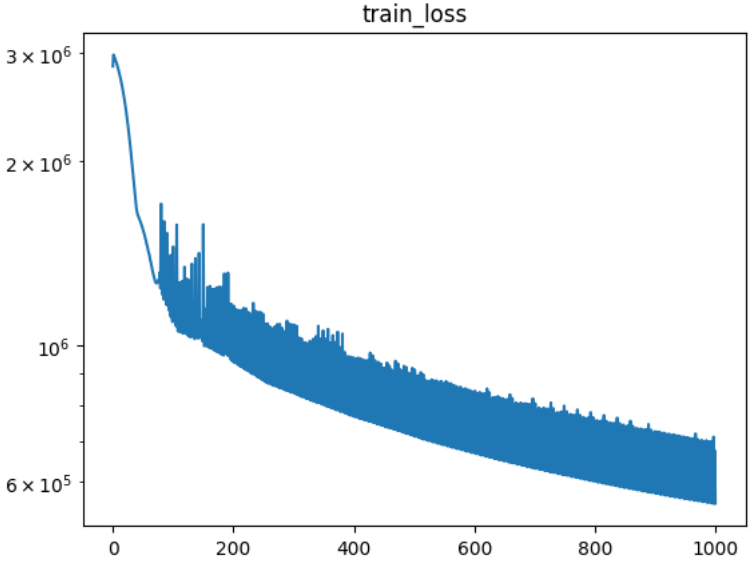




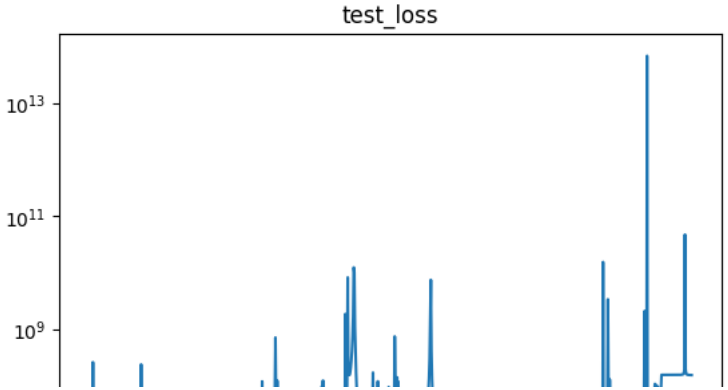
optimizer: RMSprop , n_of_data: 1000 , Bsize: 50 , learningRate: 0.001 , minimum_RMSE: 889.40 , epoch: 1000 , test_loss: 87733 , train_loss

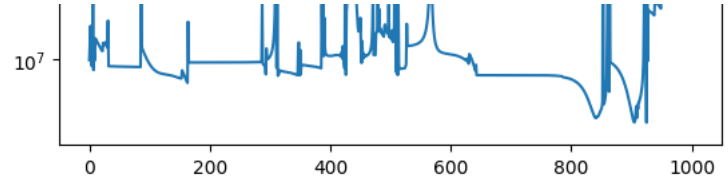


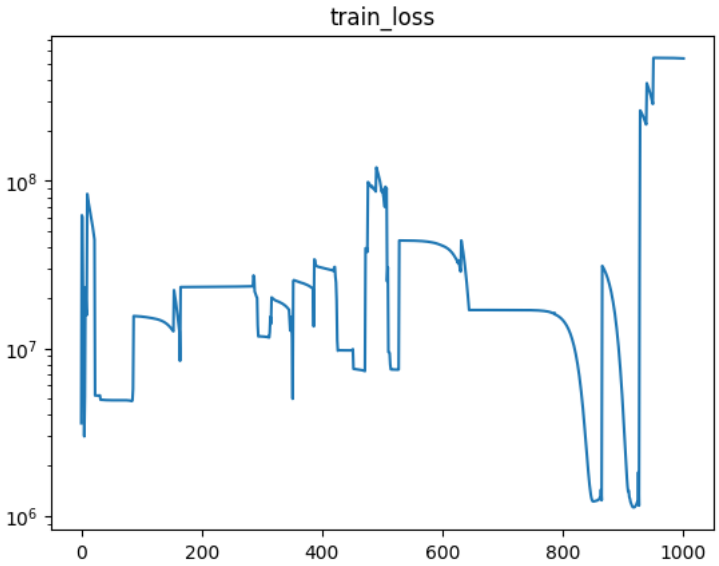




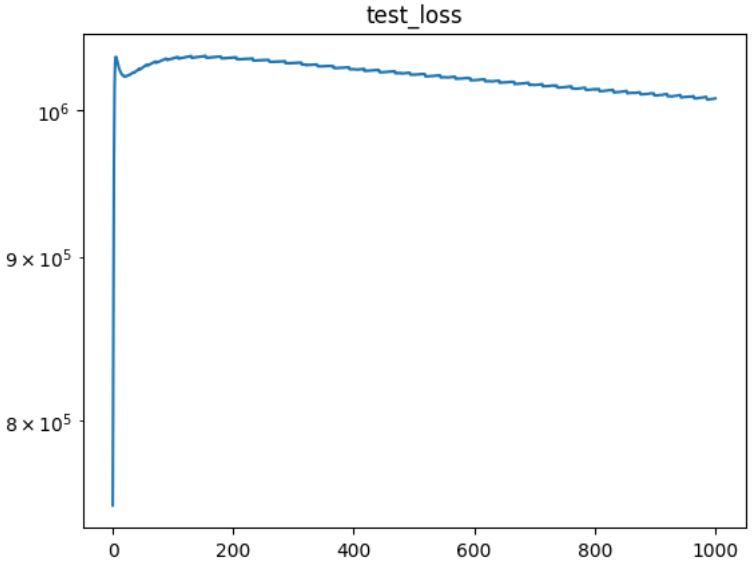
optimizer: RMSprop , n_of_data: 1000 , Bsize: 50 , learningRate: 0.005 , minimum_RMSE: 877.91 , epoch: 1000 , test_loss: 12463 , train_loss

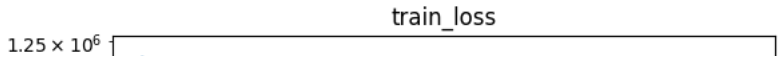


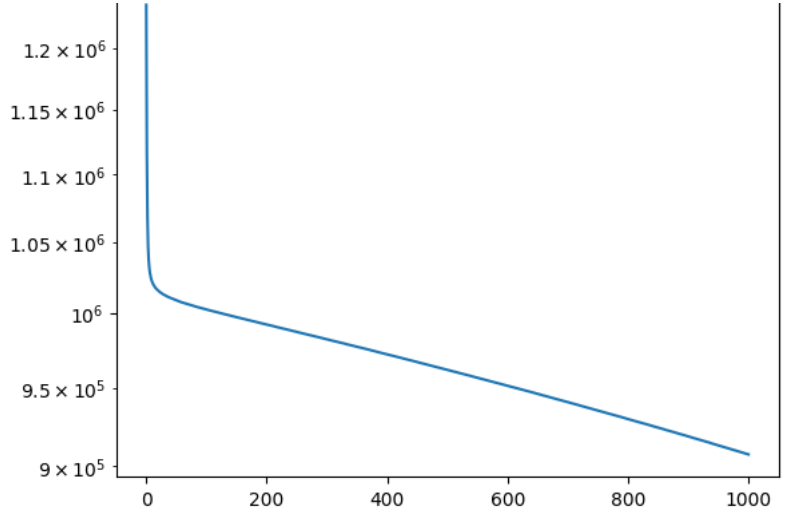




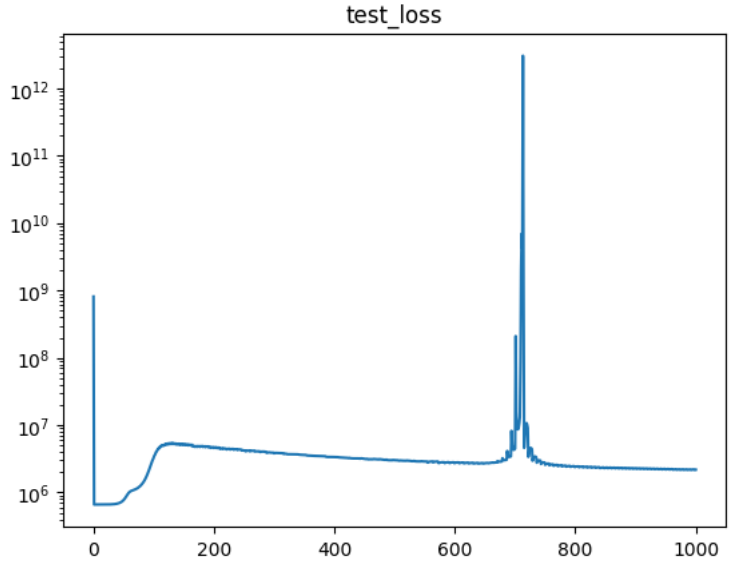
optimizer: RMSprop , n_of_data: 1000 , Bsize: 50 , learningRate: 0.0001 , minimum_RMSE: 867.77 , epoch: 1000 , test_loss: 1004 , train_los

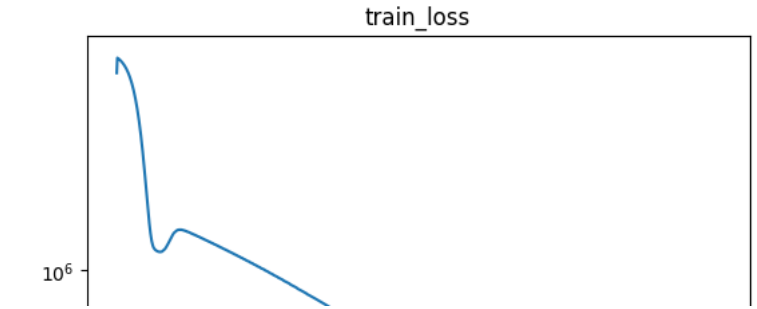


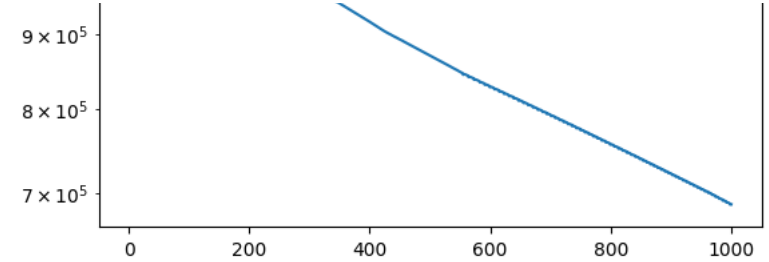




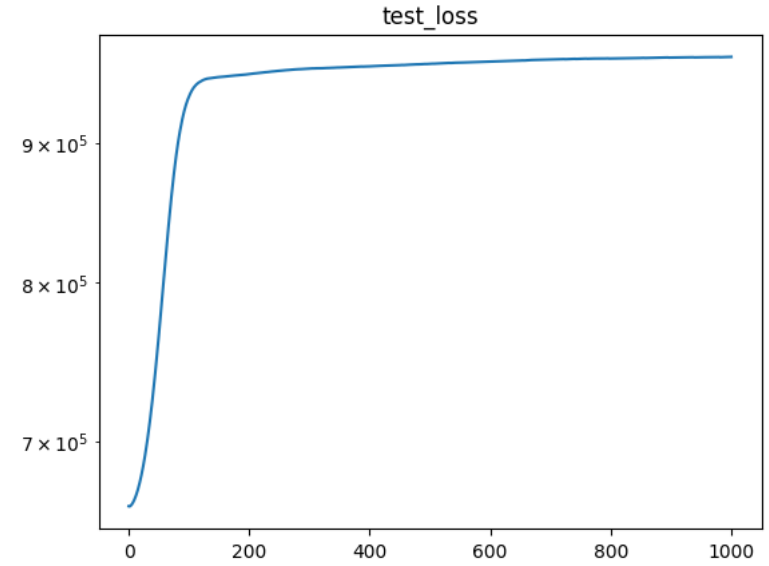
optimizer: RMSprop , n_of_data: 1000 , Bsize: 50 , learningRate: 0.0005 , minimum_RMSE: 814.31 , epoch: 1000 , test_loss: 1477 , train_los

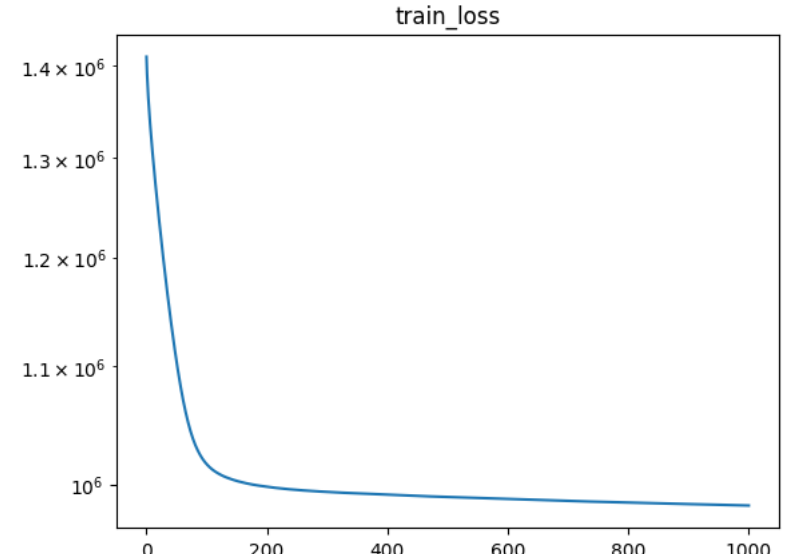




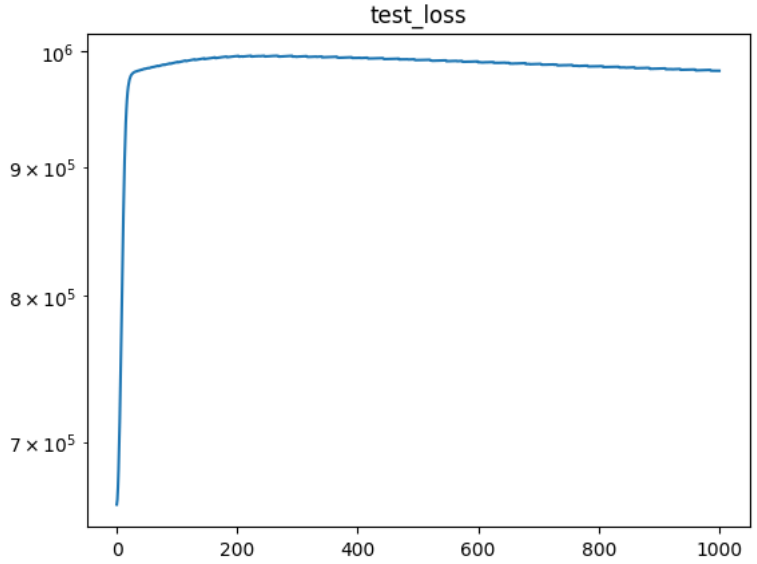


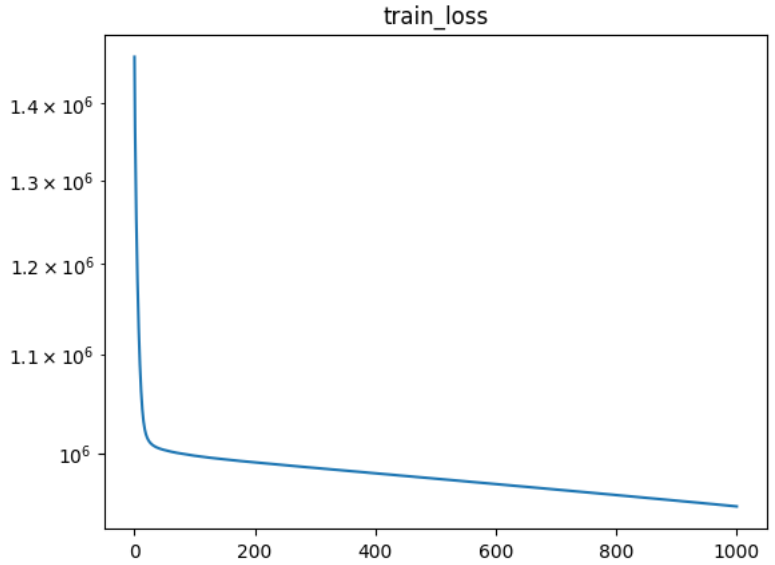
optimizer: RMSprop , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-05 , minimum_RMSE: 814.04 , epoch: 1000 , test_loss: 983 , train_loss



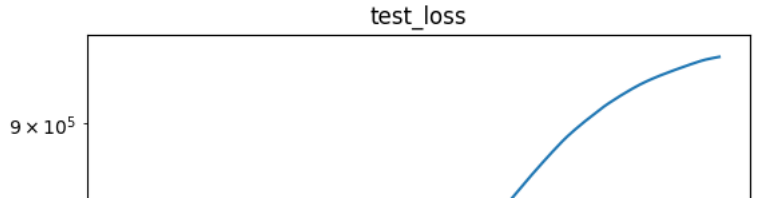


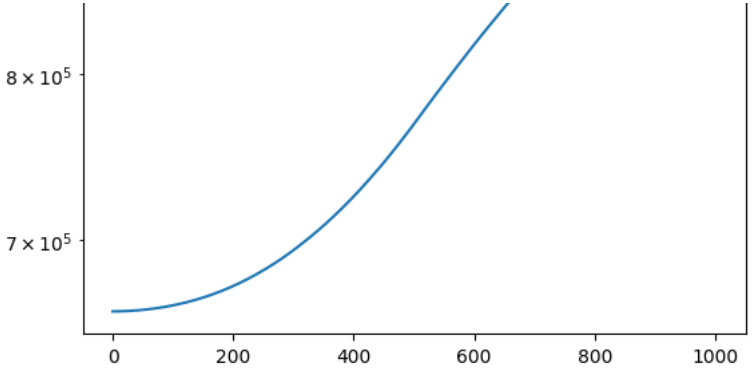
```
optimizer: RMSprop , n_of_data: 1000 , Bsize: 50 , learningRate: 5e-05 , minimum_RMSE: 813.09 , epoch:  1000 , test_loss:   991 , train_loss
```

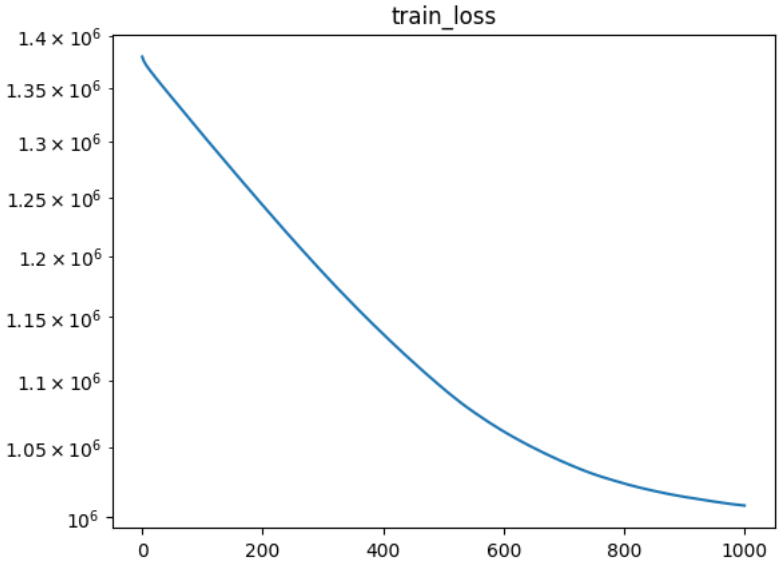




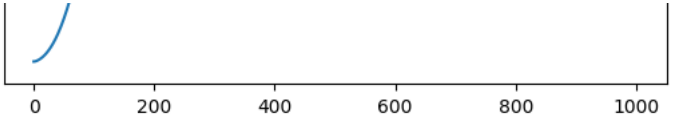

```
optimizer: RMSprop , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-06 , minimum_RMSE: 813.08 , epoch:  1000 , test_loss:   974 , train_loss
```

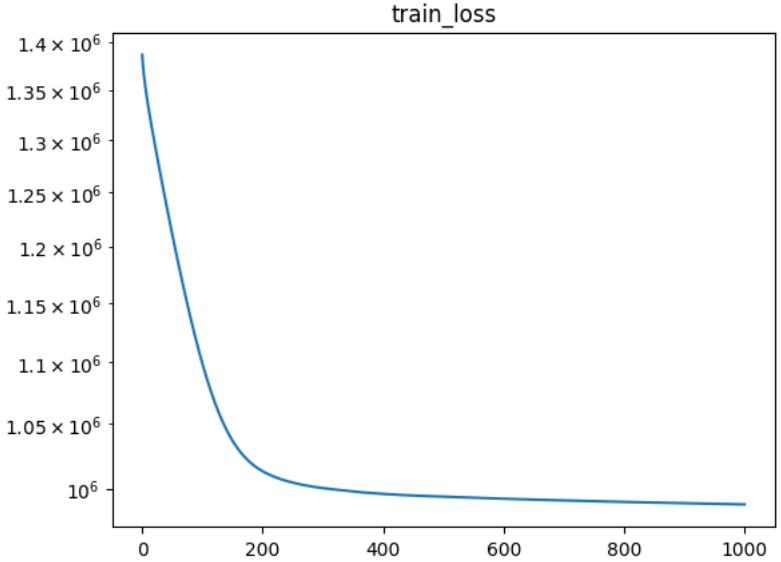




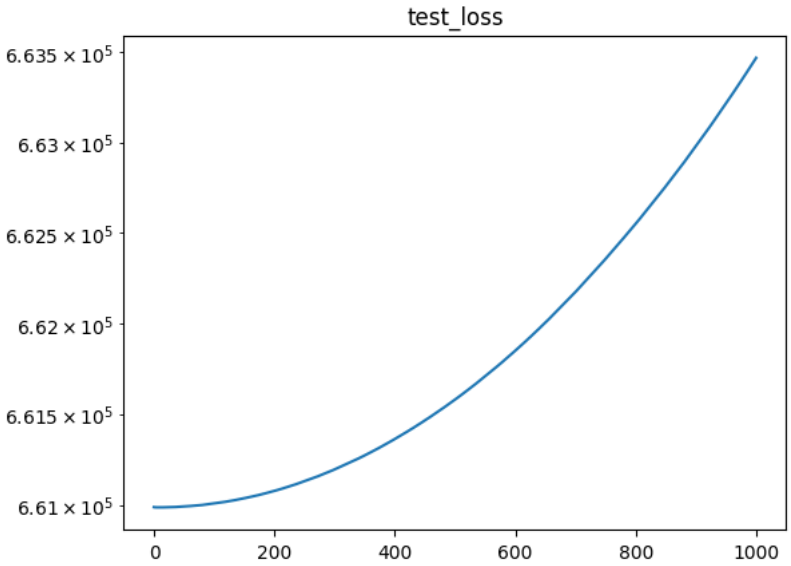


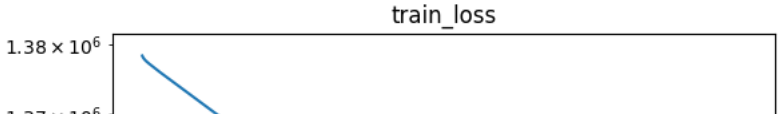
optimizer: RMSprop , n_of_data: 1000 , Bsize: 50 , learningRate: 5e-06 , minimum_RMSE: 813.01 , epoch: 1000 , test_loss: 979 , train_loss

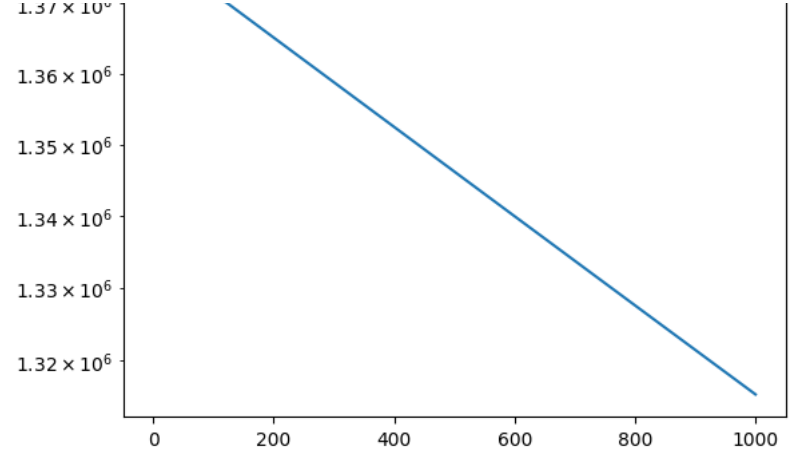




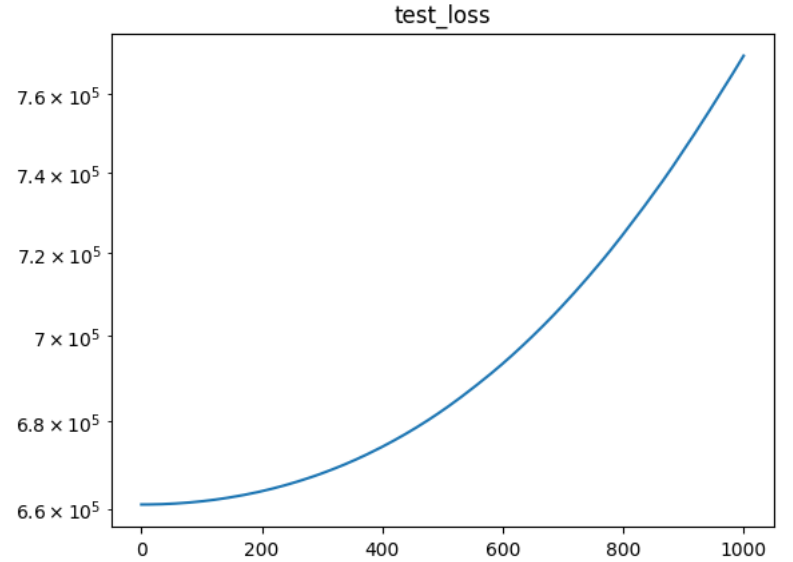
optimizer: RMSprop , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-07 , minimum_RMSE: 813.01 , epoch: 1000 , test_loss: 814 , train_loss

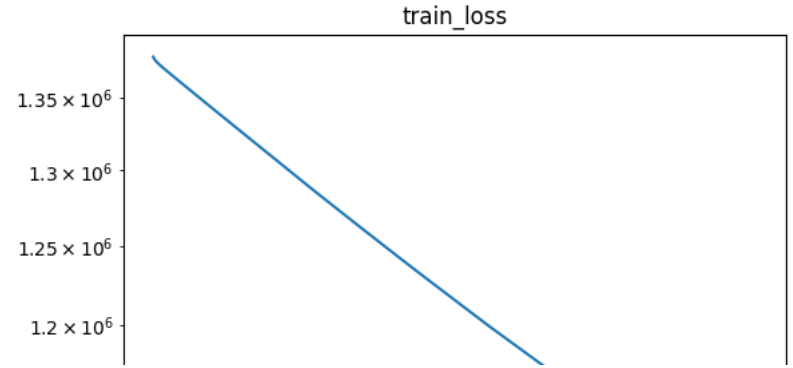


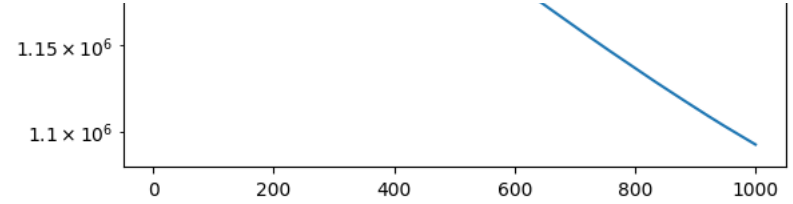




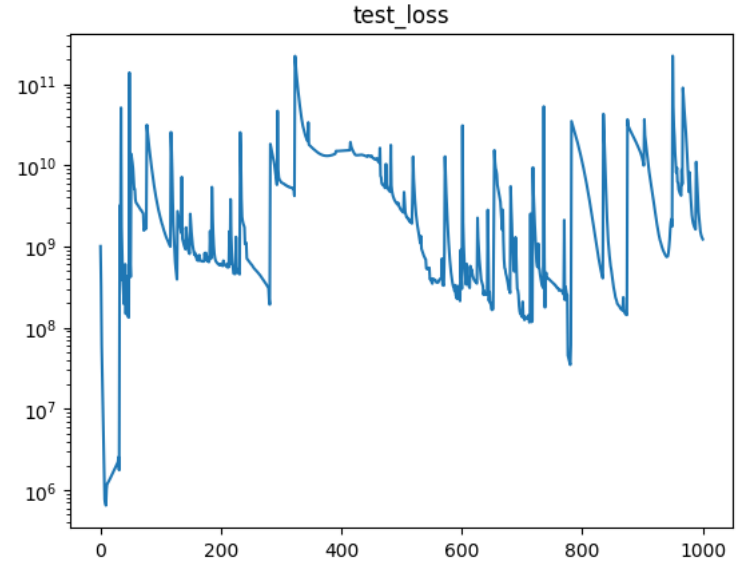
optimizer: RMSprop , n_of_data: 1000 , Bsize: 50 , learningRate: 5e-07 , minimum_RMSE: 813.00 , epoch: 1000 , test_loss: 877 , train_loss

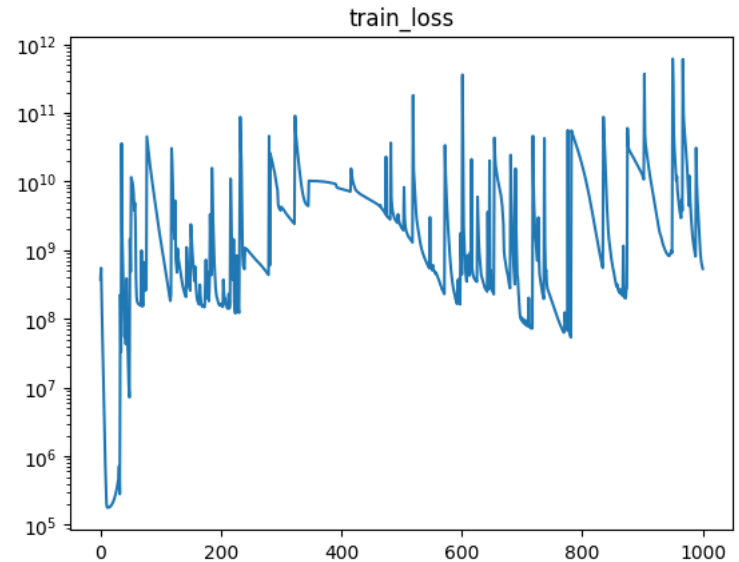




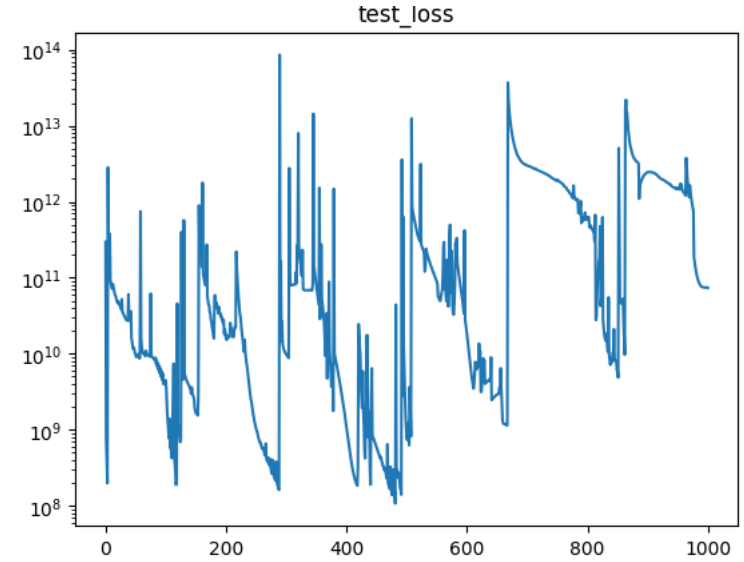


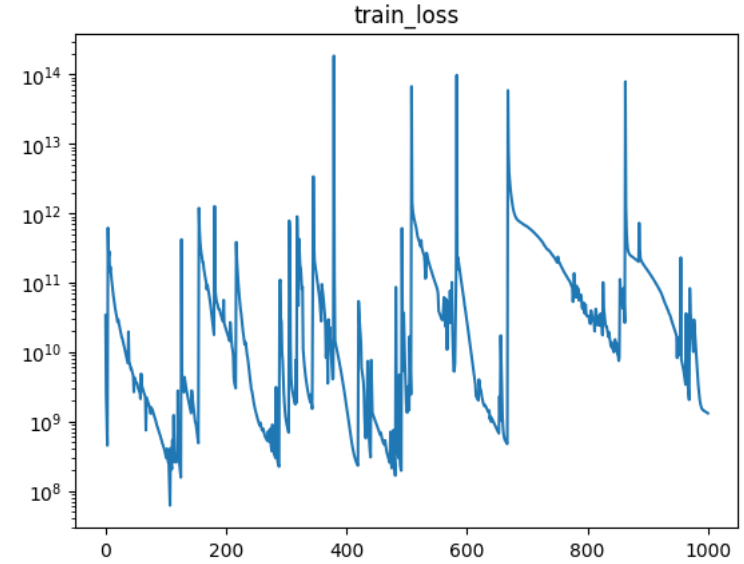
optimizer: RMSprop , n_of_data: 1000 , Bsize: 100 , learningRate: 0.1 , minimum_RMSE: 805.09 , epoch: 1000 , test_loss: 35069 , train_loss:



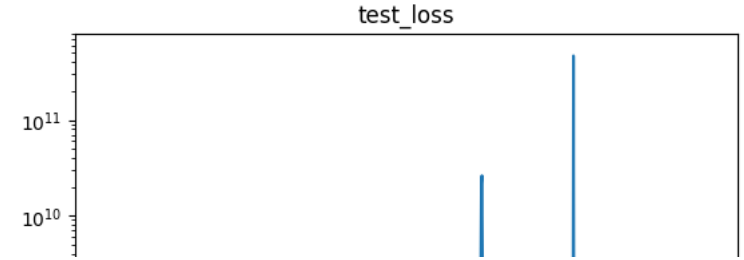


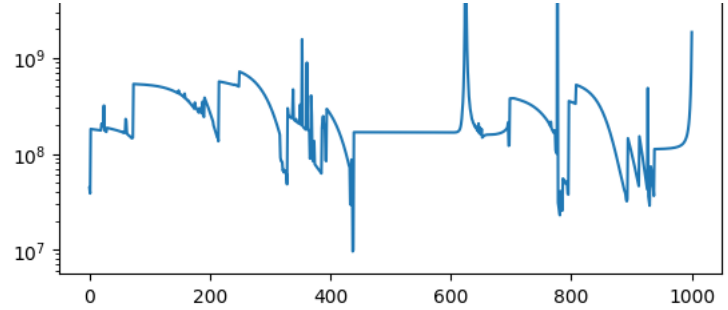
optimizer: RMSprop , n_of_data: 1000 , Bsize: 100 , learningRate: 0.5 , minimum_RMSE: 10329.96 , epoch: 1000 , test_loss: 271299 , train_lo

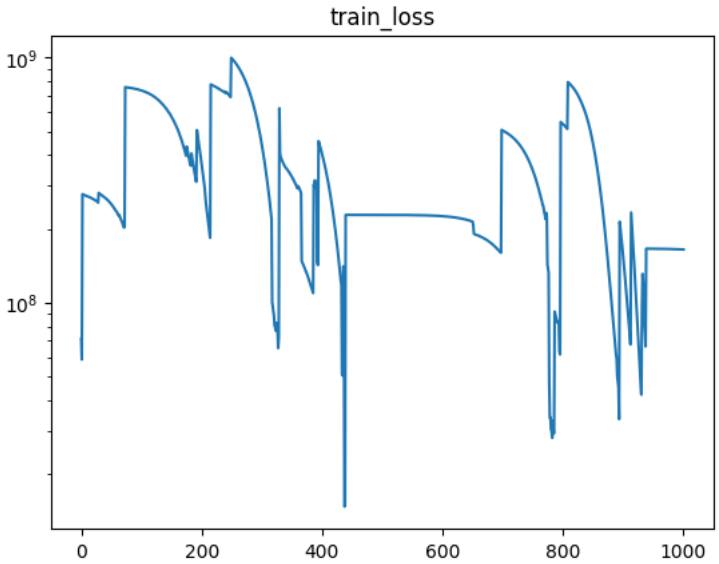




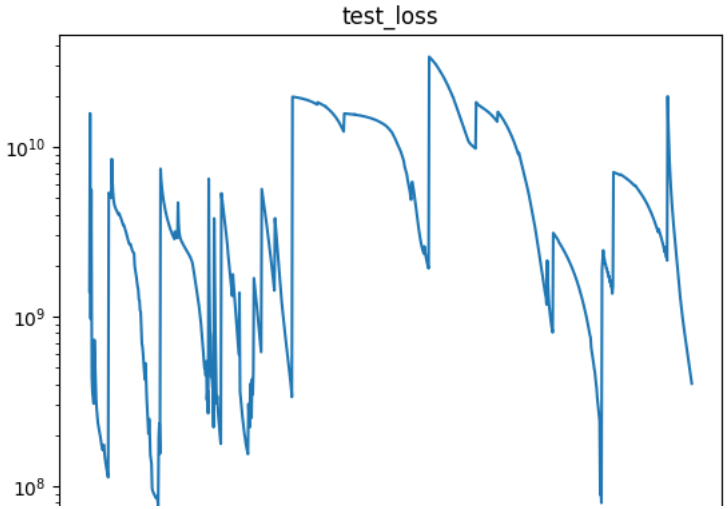
optimizer: RMSprop , n_of_data: 1000 , Bsize: 100 , learningRate: 0.01 , minimum_RMSE: 3098.18 , epoch: 1000 , test_loss: 43078 , train_los

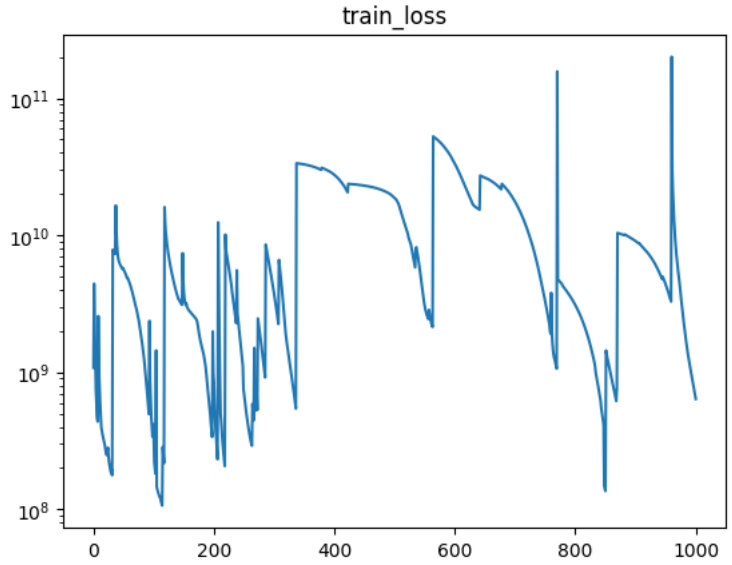




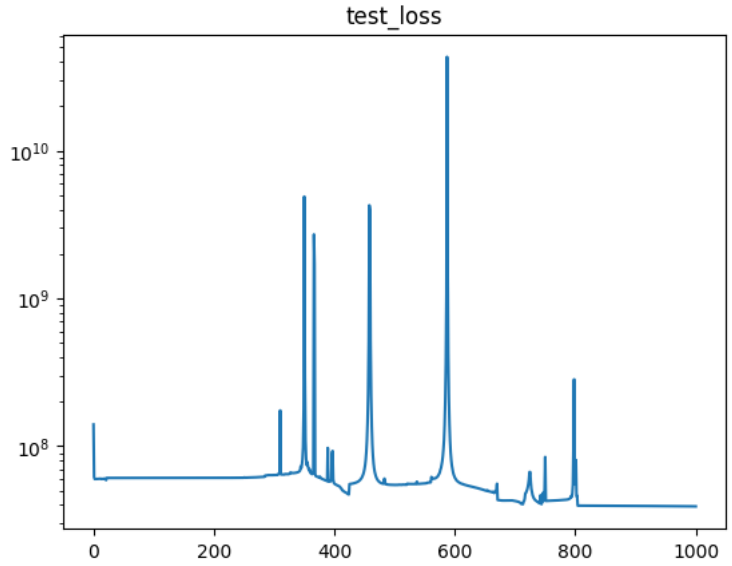


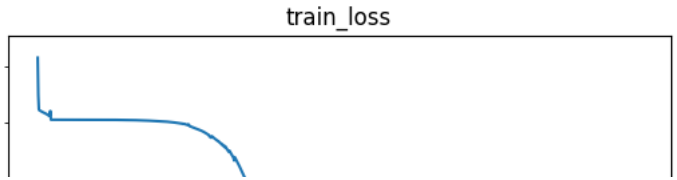
optimizer: RMSprop , n_of_data: 1000 , Bsize: 100 , learningRate: 0.05 , minimum_RMSE: 8764.54 , epoch: 1000 , test_loss: 20052 , train_los

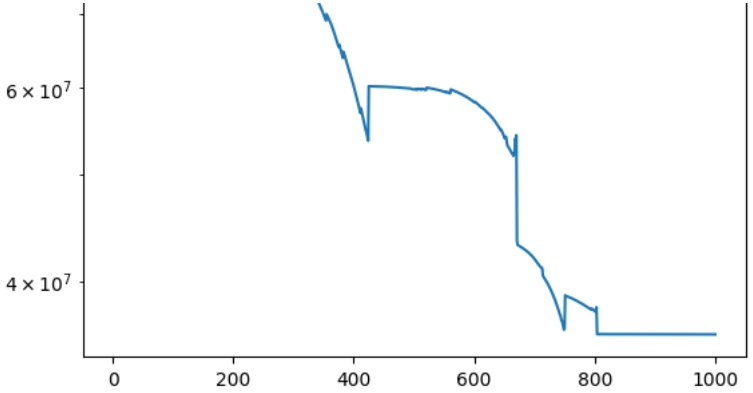




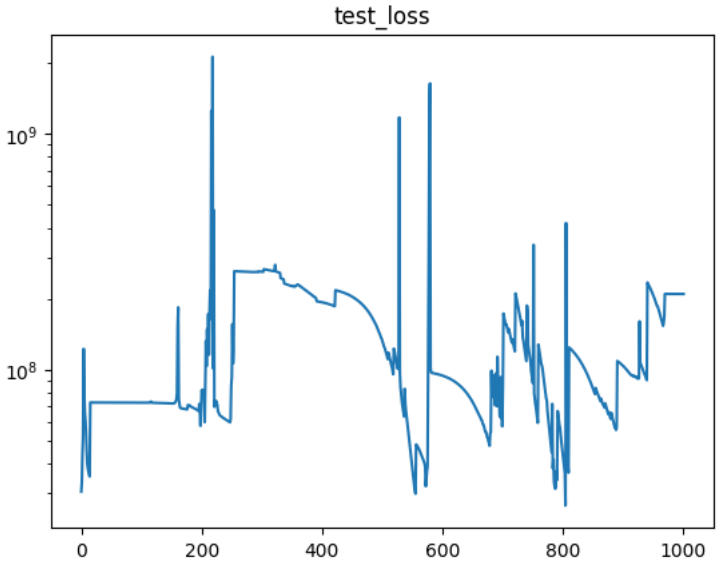
optimizer: RMSprop , n_of_data: 1000 , Bsize: 100 , learningRate: 0.001 , minimum_RMSE: 6255.35 , epoch: 1000 , test_loss: 6255 , train_lc

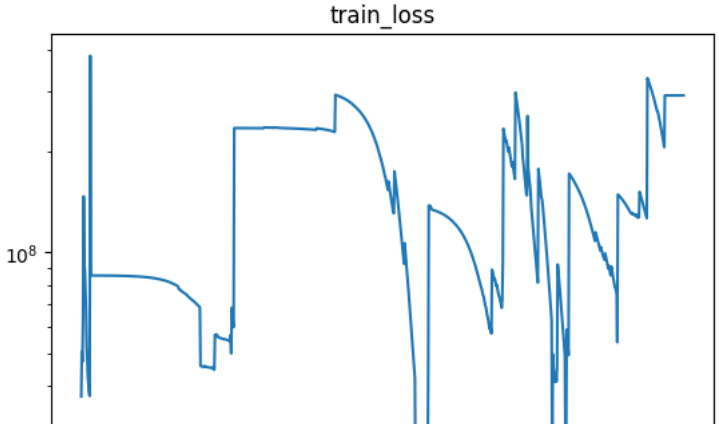


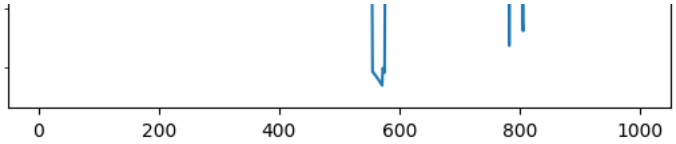




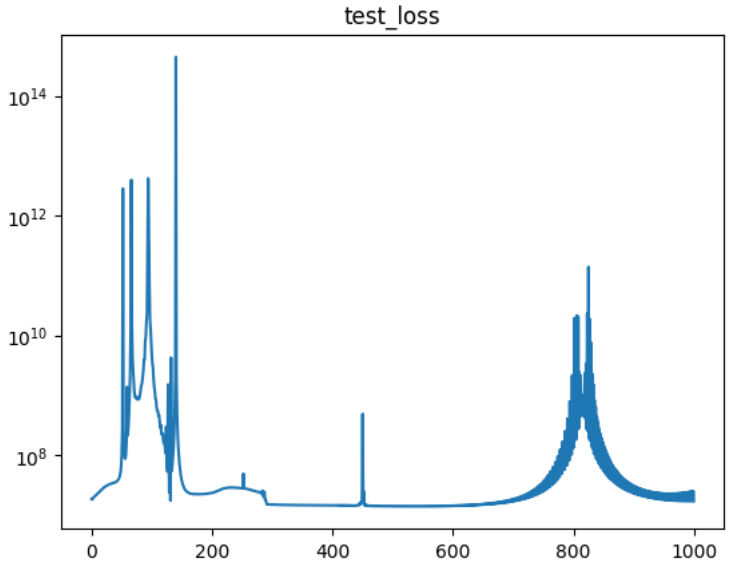
optimizer: RMSprop , n_of_data: 1000 , Bsize: 100 , learningRate: 0.005 , minimum_RMSE: 5162.27 , epoch: 1000 , test_loss: 14484 , train_lc

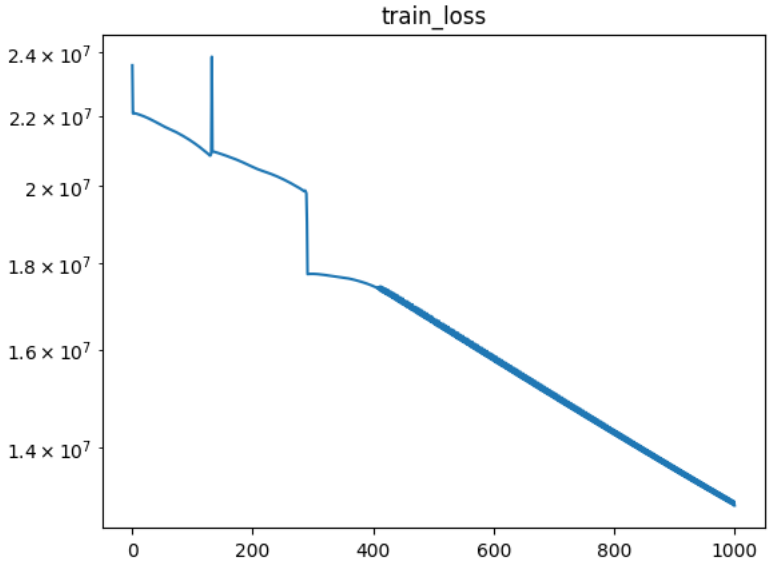




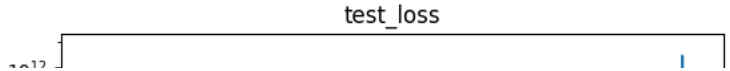


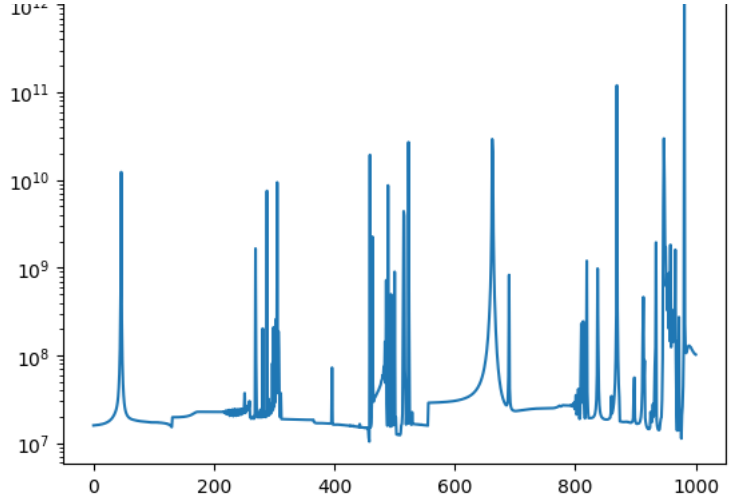
optimizer: RMSprop , n_of_data: 1000 , Bsize: 100 , learningRate: 0.0001 , minimum_RMSE: 3754.79 , epoch: 1000 , test_loss: 4167 , train_l

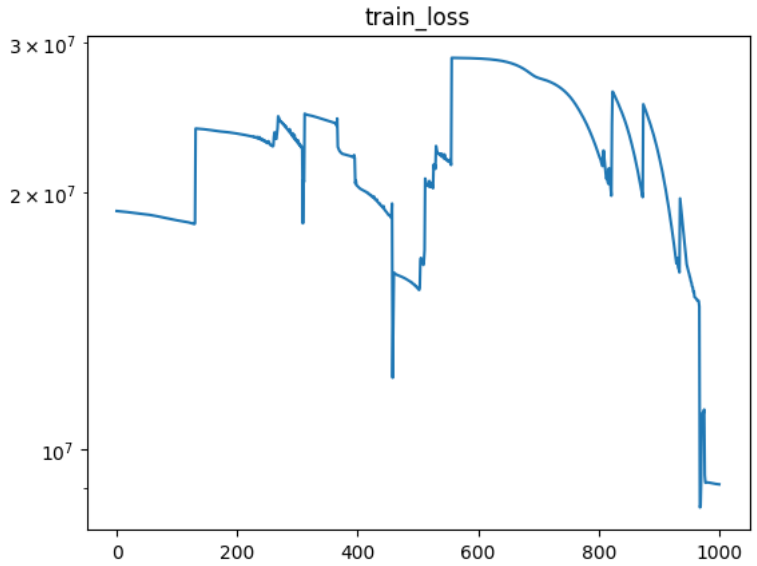




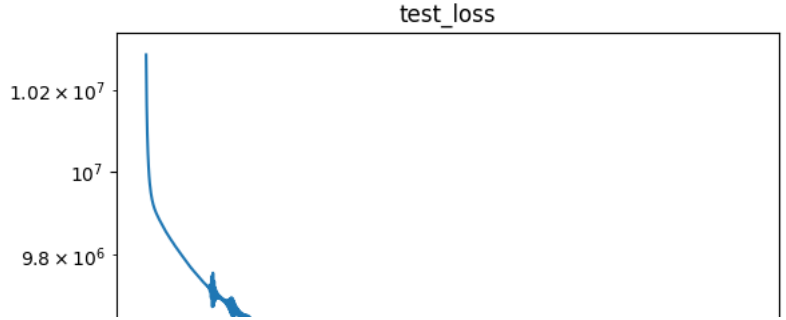
optimizer: RMSprop , n_of_data: 1000 , Bsize: 100 , learningRate: 0.0005 , minimum_RMSE: 3234.86 , epoch: 1000 , test_loss: 10127 , train_l

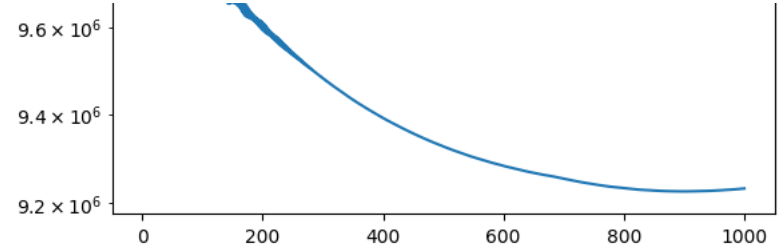


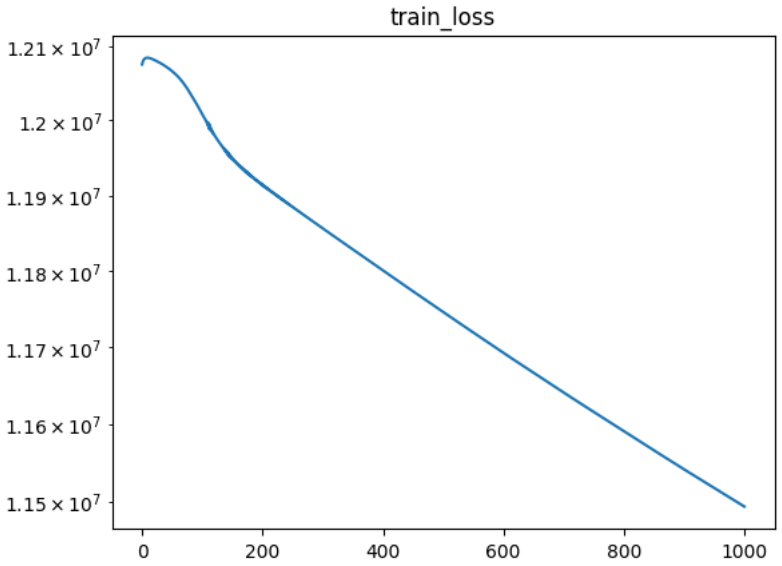




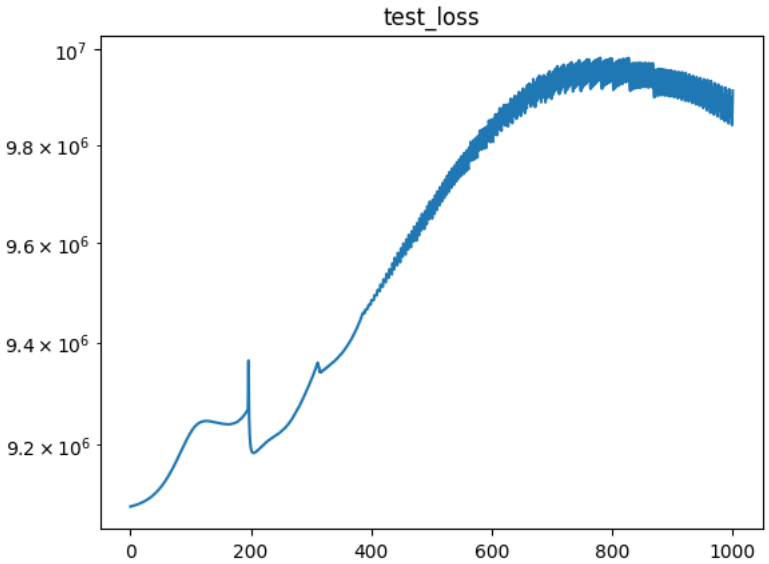
optimizer: RMSprop , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-05 , minimum_RMSE: 3037.40 , epoch: 1000 , test_loss: 3038 , train_lc

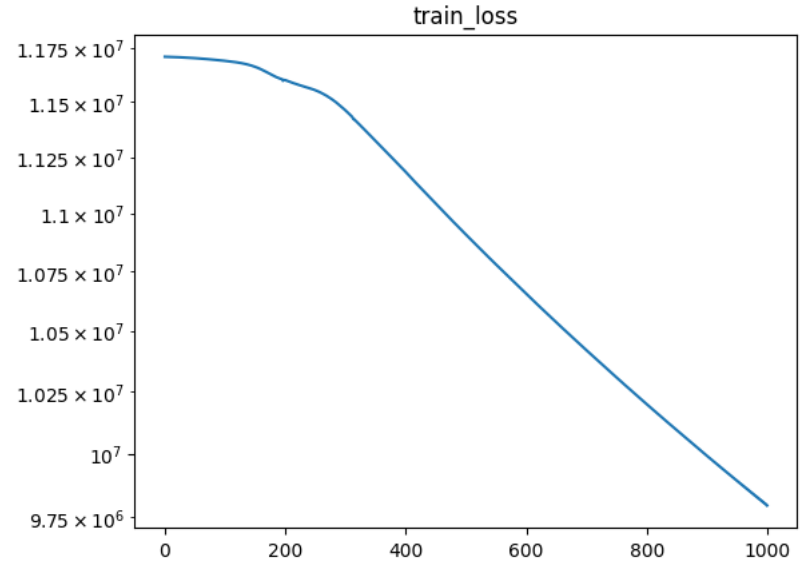




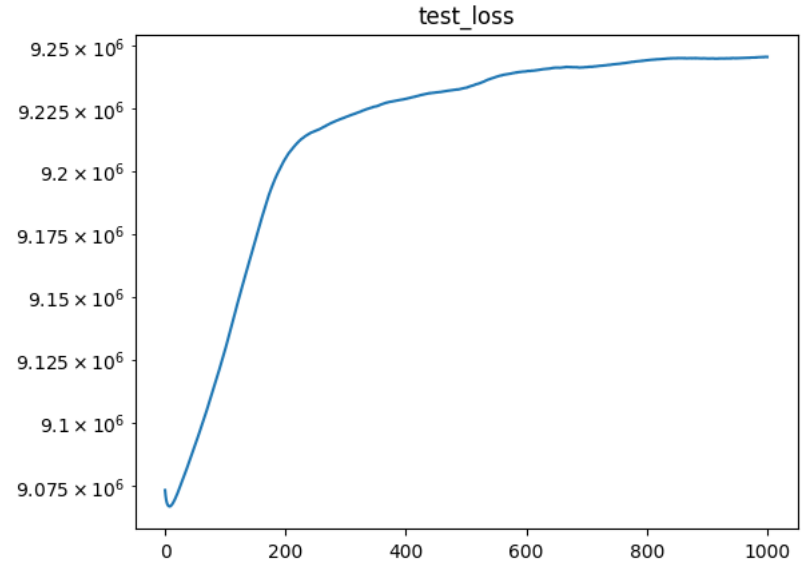


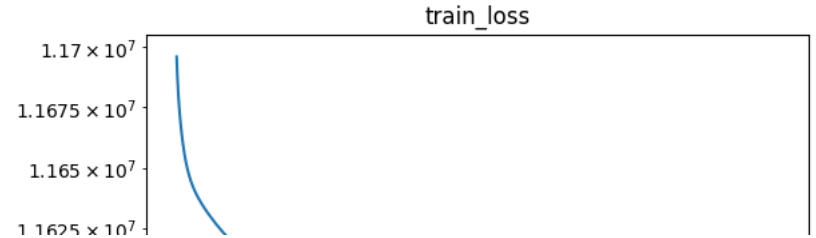
optimizer: RMSprop , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-05 , minimum_RMSE: 3013.59 , epoch: 1000 , test_loss: 3148 , train_lc

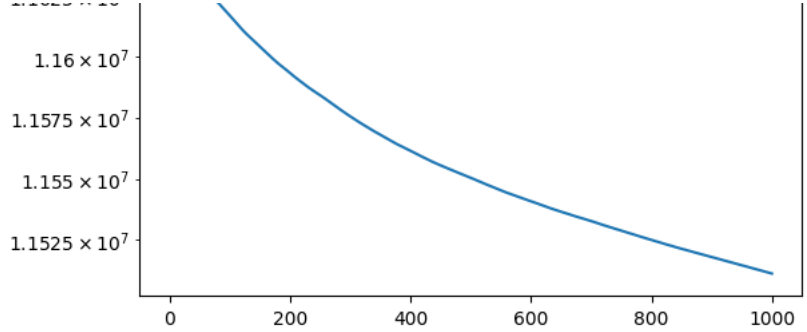




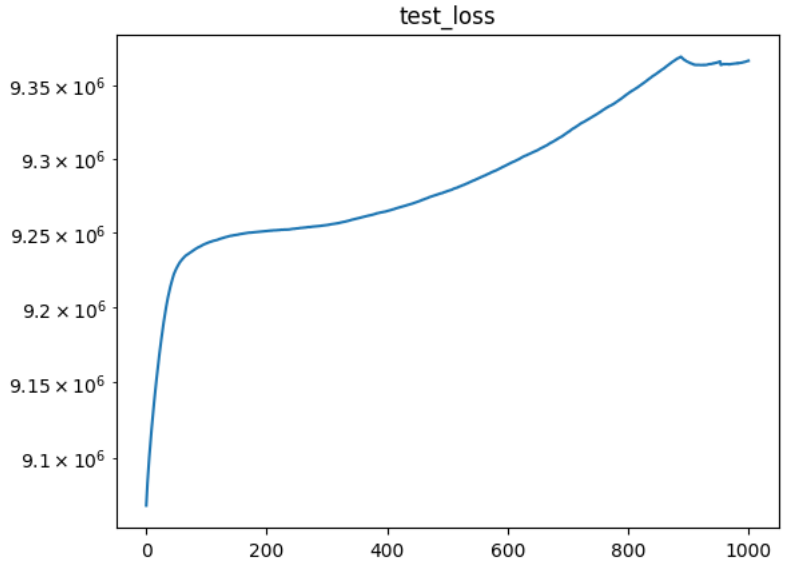
optimizer: RMSprop , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-06 , minimum_RMSE: 3011.14 , epoch: 1000 , test_loss: 3040 , train_lc

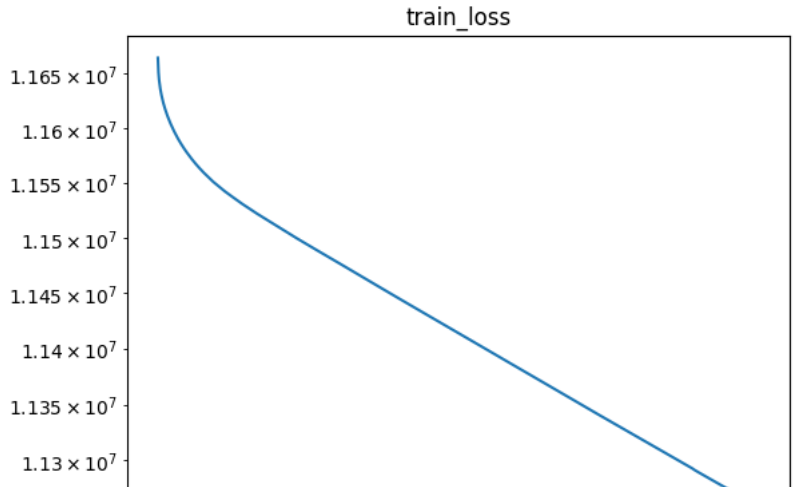


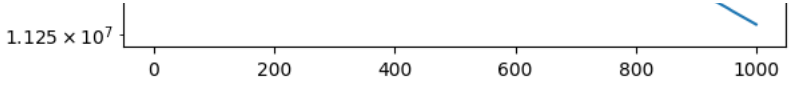




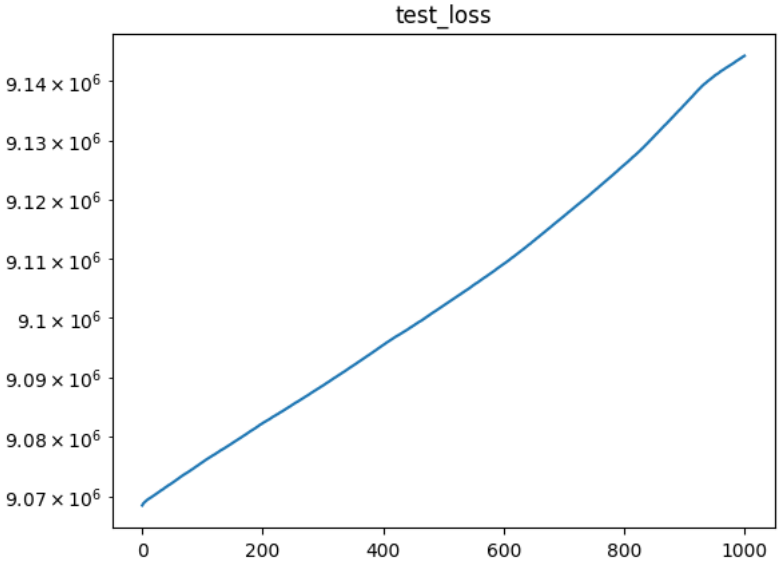
optimizer: RMSprop , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-06 , minimum_RMSE: 3011.36 , epoch: 1000 , test_loss: 3060 , train_lc

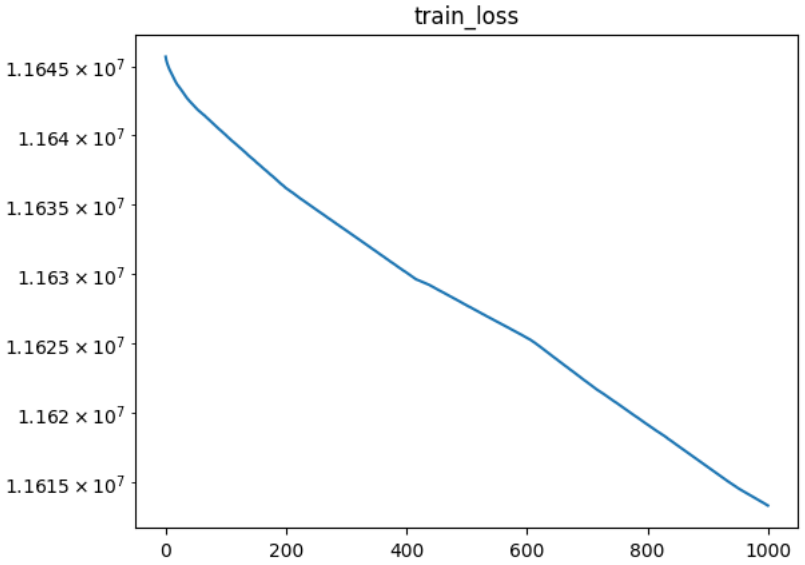




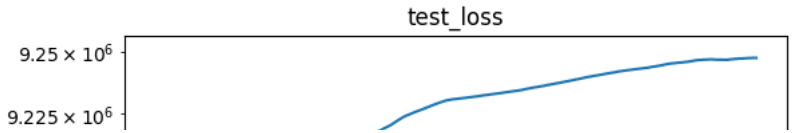


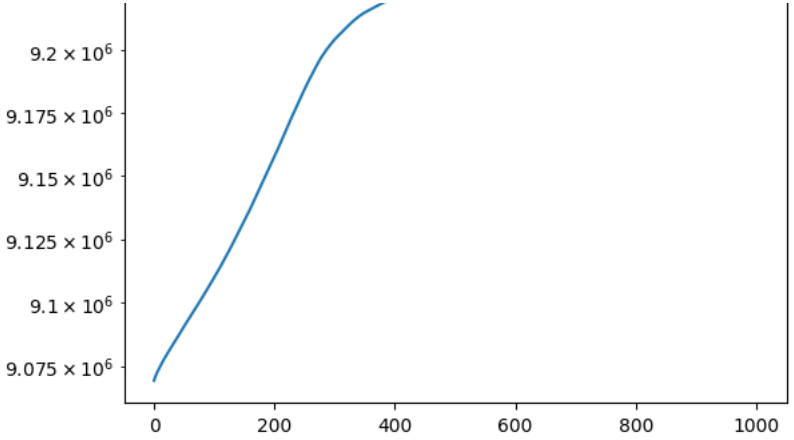
optimizer: RMSprop , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-07 , minimum_RMSE: 3011.39 , epoch: 1000 , test_loss: 3023 , train_lc

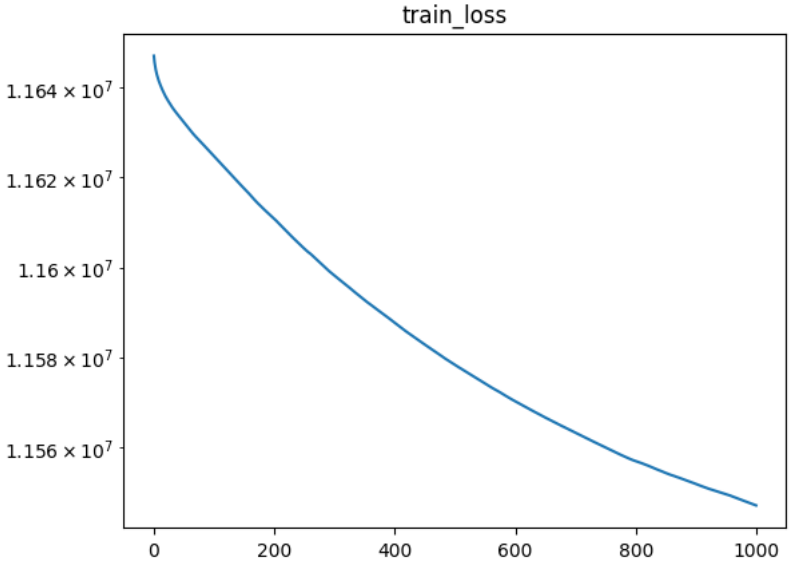




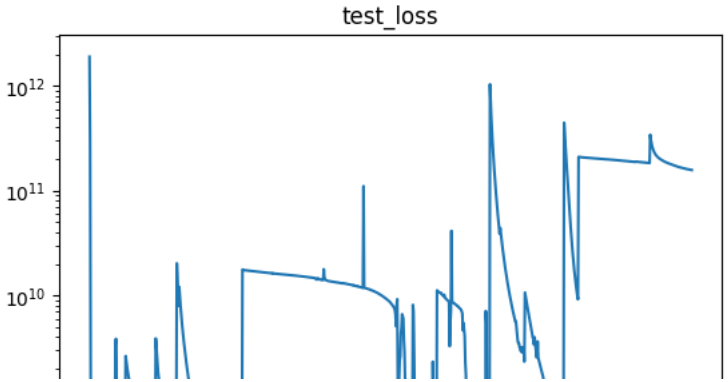
optimizer: RMSprop , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-07 , minimum_RMSE: 3011.54 , epoch: 1000 , test_loss: 3040 , train_lc

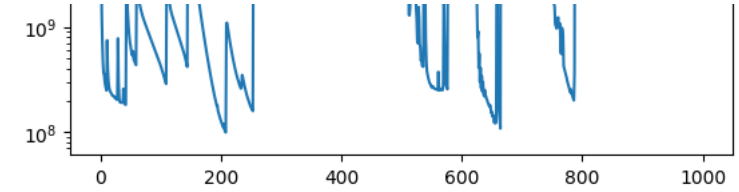


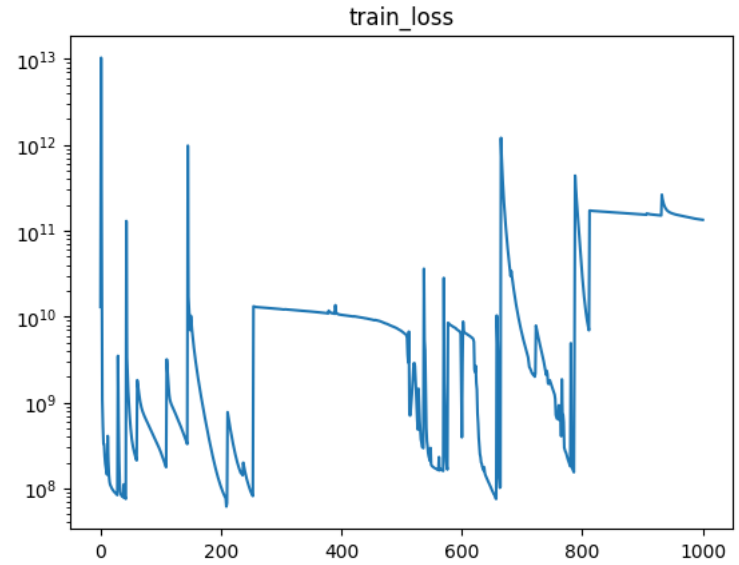




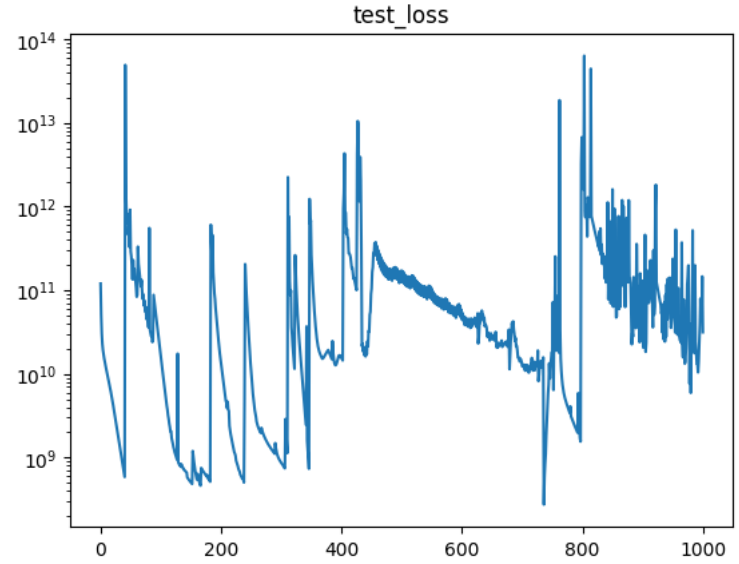
optimizer: RMSprop , n_of_data: 1000 , Bsize: 500 , learningRate: 0.1 , minimum_RMSE: 9953.25 , epoch: 1000 , test_loss: 396494 , train_los

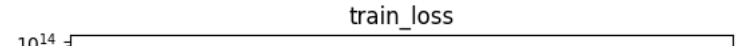


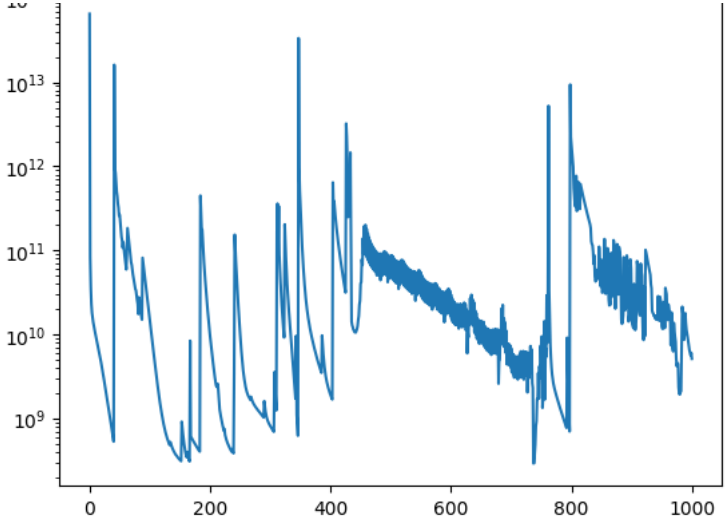




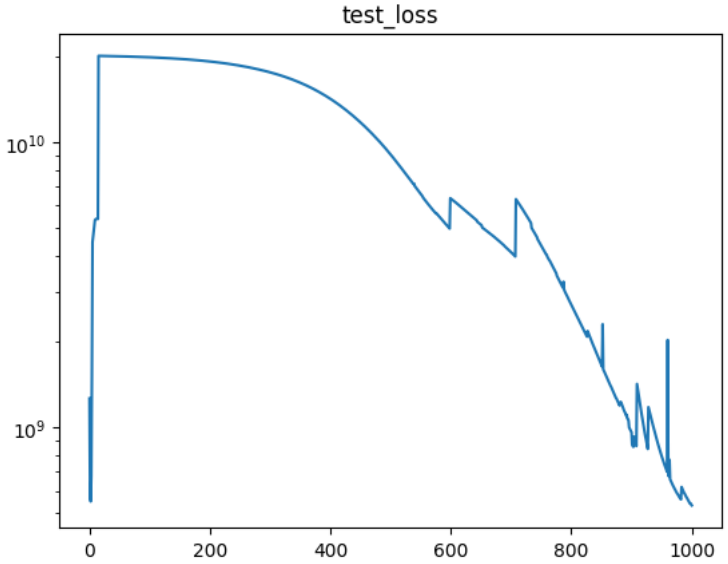
optimizer: RMSprop , n_of_data: 1000 , Bsize: 500 , learningRate: 0.5 , minimum_RMSE: 16560.75 , epoch: 1000 , test_loss: 177420 , train_lc

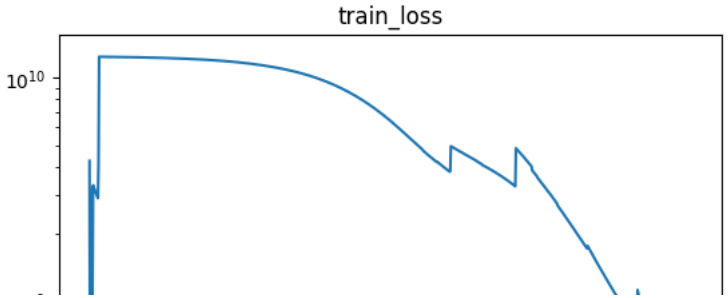


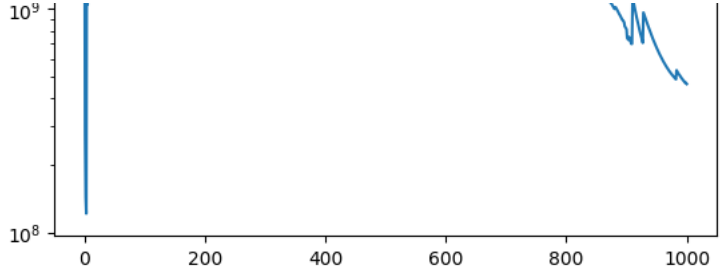




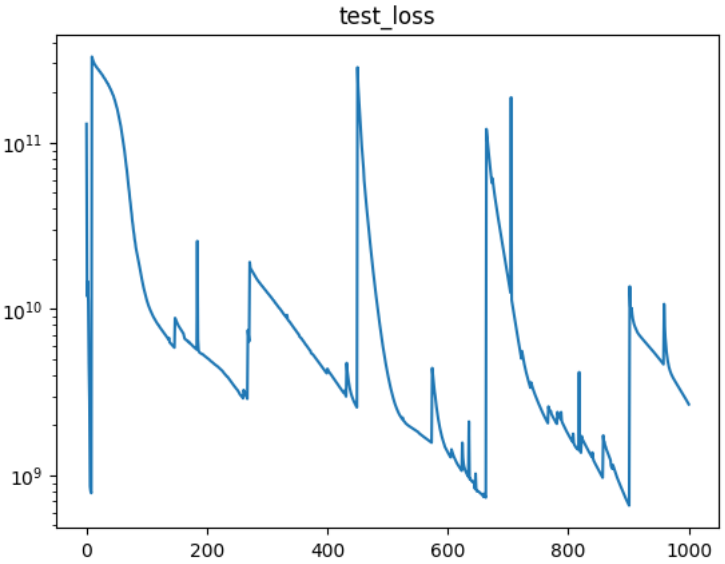
optimizer: RMSprop , n_of_data: 1000 , Bsize: 500 , learningRate: 0.01 , minimum_RMSE: 23086.16 , epoch: 1000 , test_loss: 23086 , train_lc

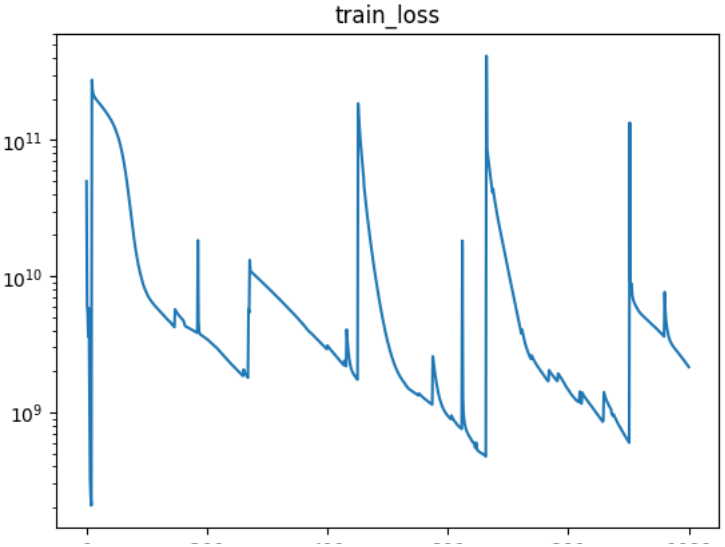


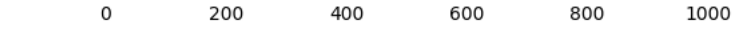




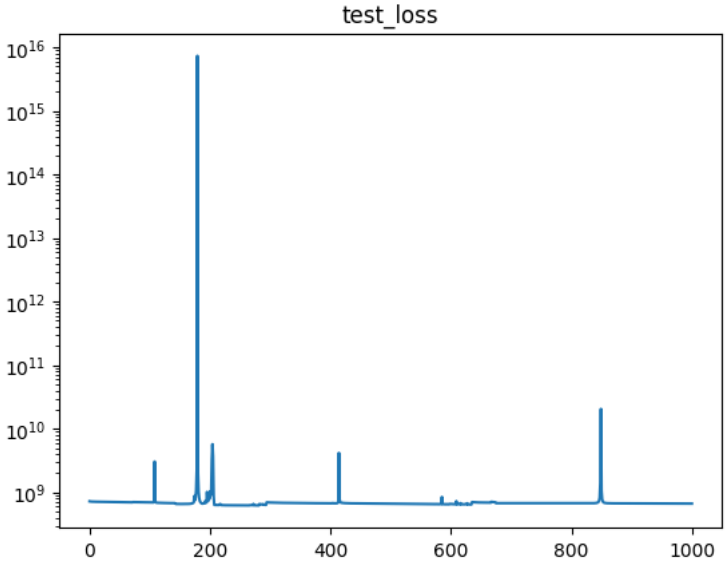
optimizer: RMSprop , n_of_data: 1000 , Bsize: 500 , learningRate: 0.05 , minimum_RMSE: 25670.05 , epoch: 1000 , test_loss: 51713 , train_lo

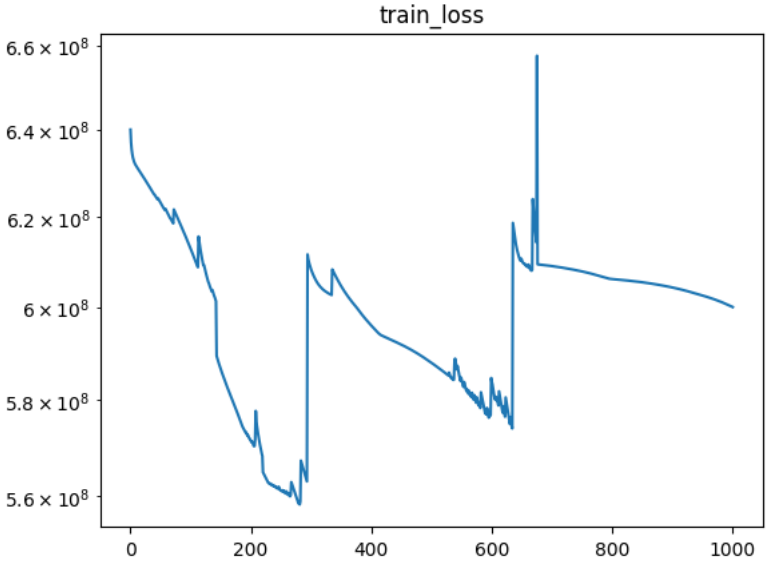




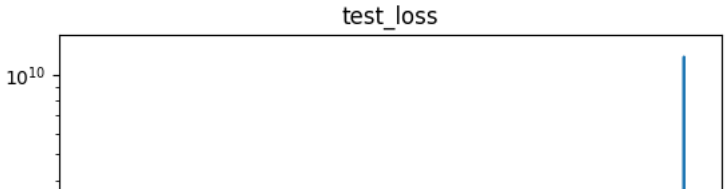


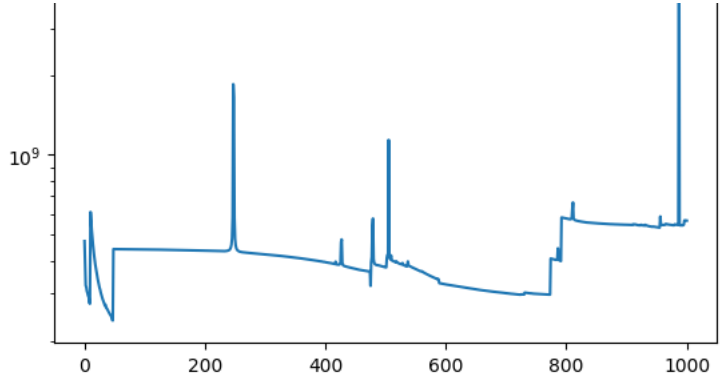
optimizer: RMSprop , n_of_data: 1000 , Bsize: 500 , learningRate: 0.001 , minimum_RMSE: 24955.56 , epoch: 1000 , test_loss: 25778 , train_l

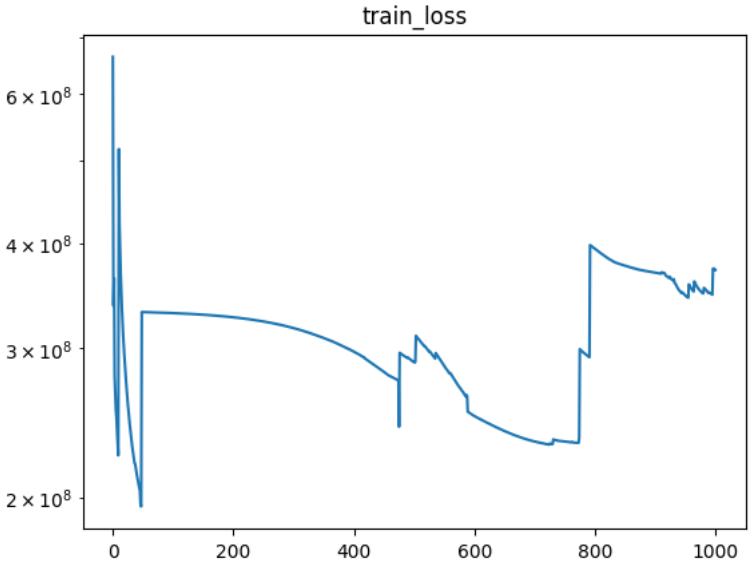




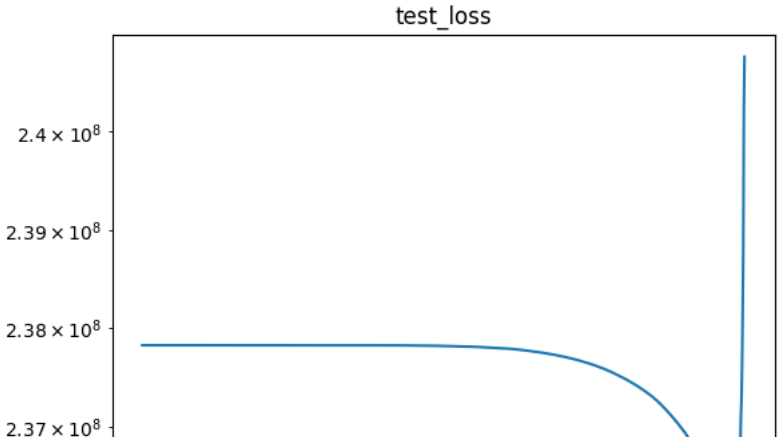
optimizer: RMSprop , n_of_data: 1000 , Bsize: 500 , learningRate: 0.005 , minimum_RMSE: 15427.96 , epoch: 1000 , test_loss: 23780 , train_l

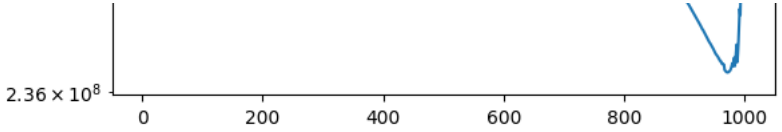


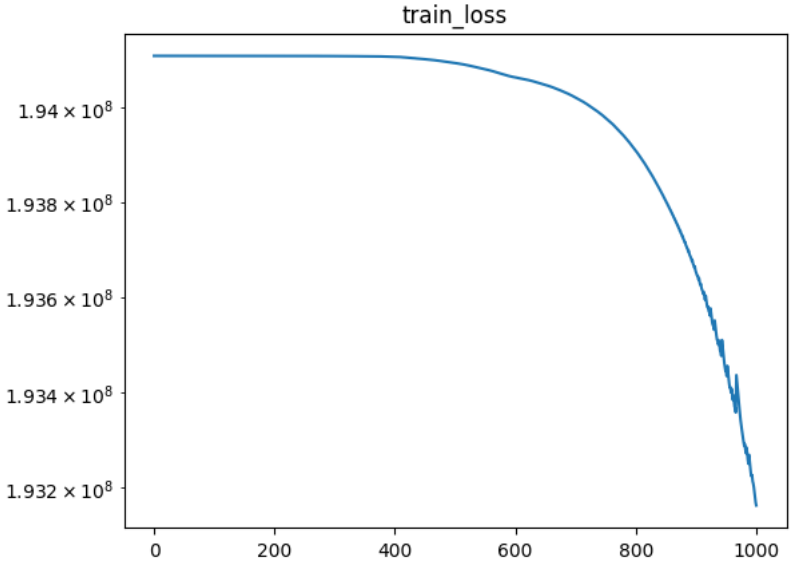




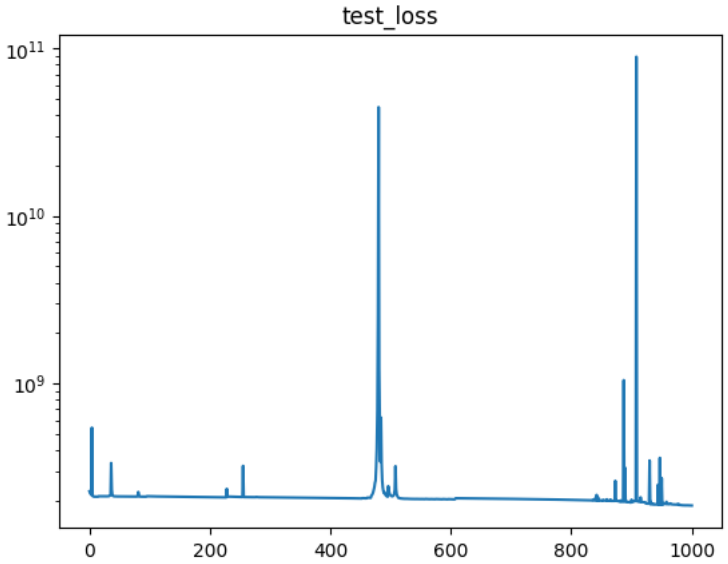
optimizer: RMSprop , n_of_data: 1000 , Bsize: 500 , learningRate: 0.0001 , minimum_RMSE: 15368.60 , epoch: 1000 , test_loss: 15516 , train_

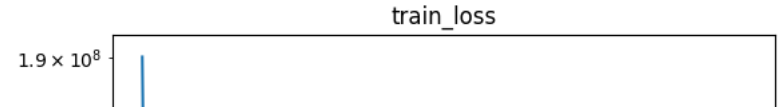


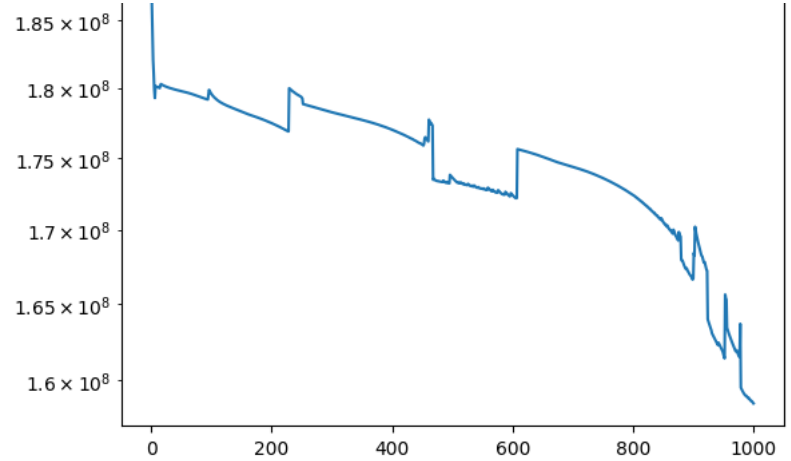




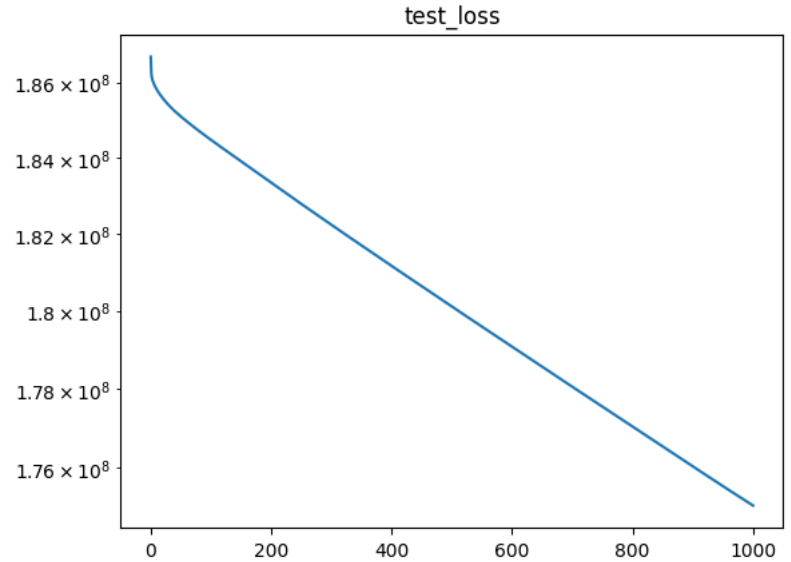
optimizer: RMSprop , n_of_data: 1000 , Bsize: 500 , learningRate: 0.0005 , minimum_RMSE: 13651.30 , epoch: 1000 , test_loss: 13651 , train_

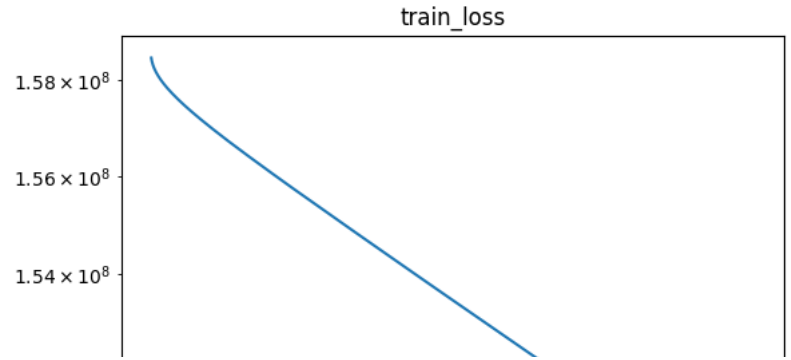


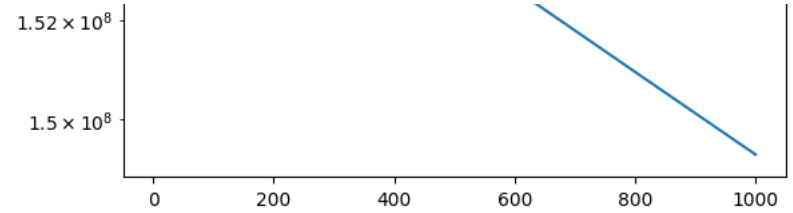




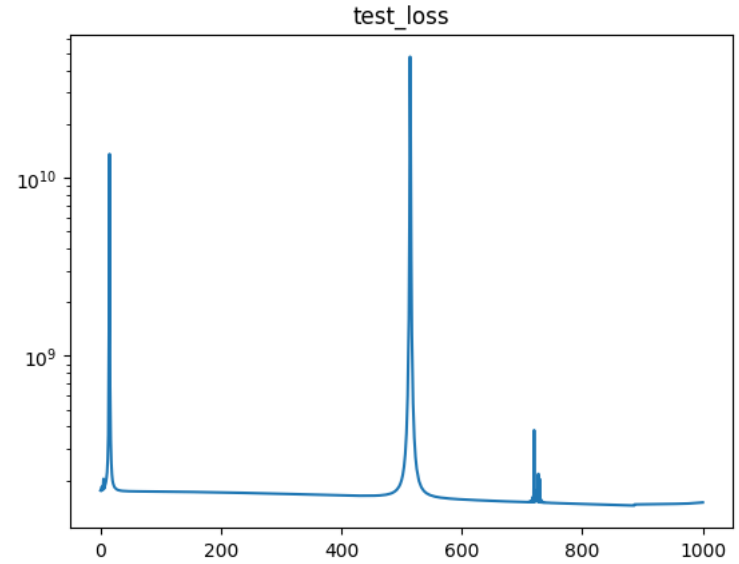
optimizer: RMSprop , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-05 , minimum_RMSE: 13230.98 , epoch: 1000 , test_loss: 13230 , train_l

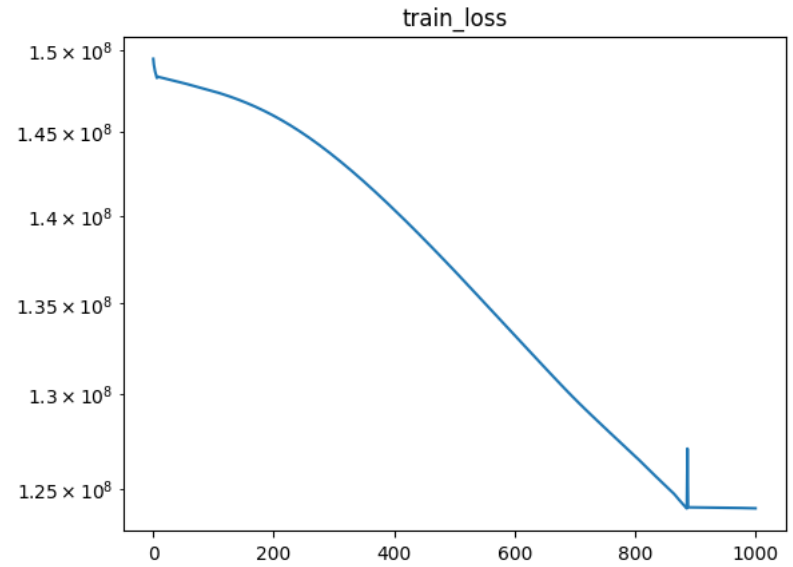




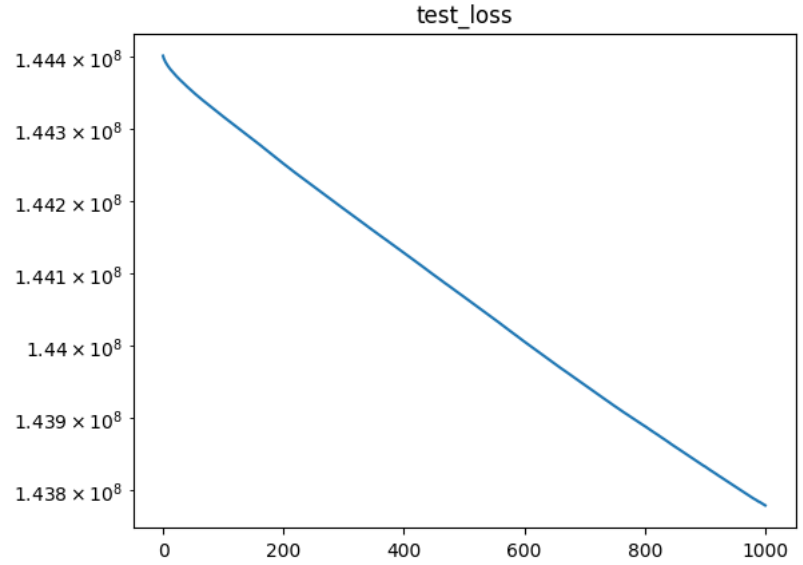


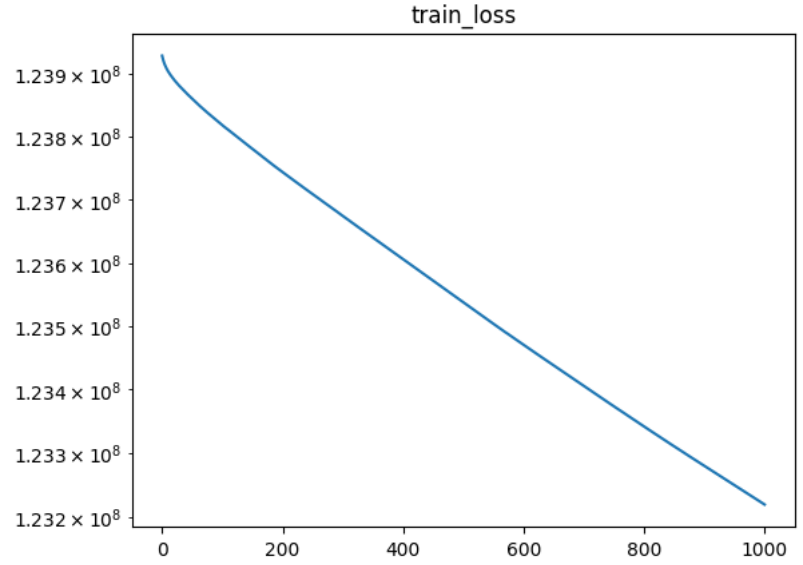
optimizer: RMSprop , n_of_data: 1000 , Bsize: 500 , learningRate: 5e-05 , minimum_RMSE: 12016.86 , epoch: 1000 , test_loss: 12258 , train_l



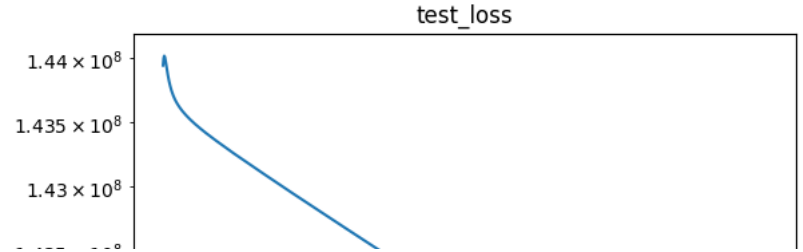


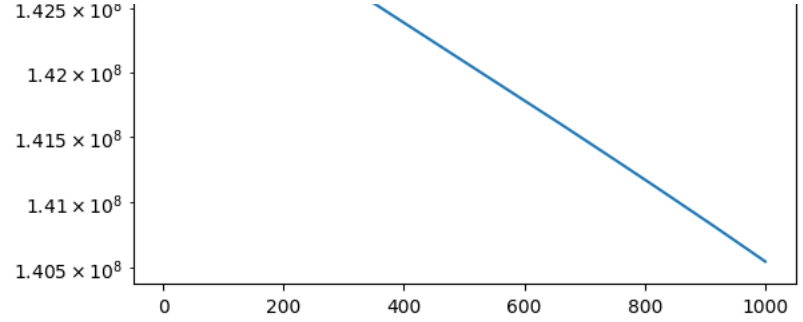
optimizer: RMSprop , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-06 , minimum_RMSE: 11990.81 , epoch: 1000 , test_loss: 11990 , train_l

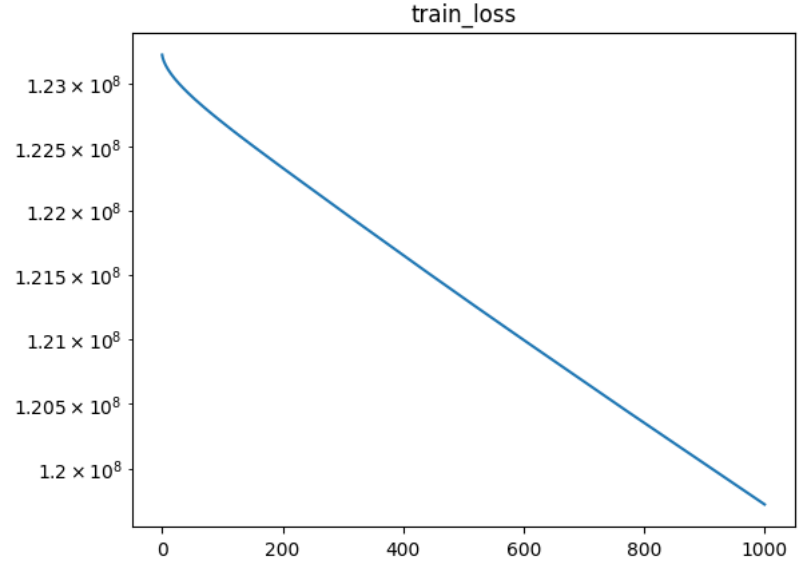




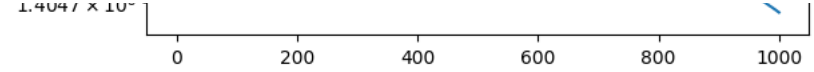
optimizer: RMSprop , n_of_data: 1000 , Bsize: 500 , learningRate: 5e-06 , minimum_RMSE: 11855.24 , epoch: 1000 , test_loss: 11855 , train_l

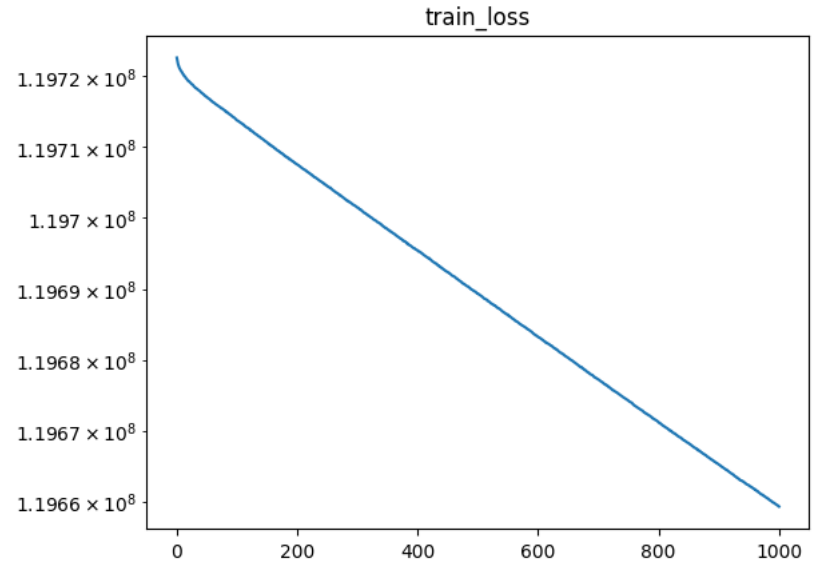




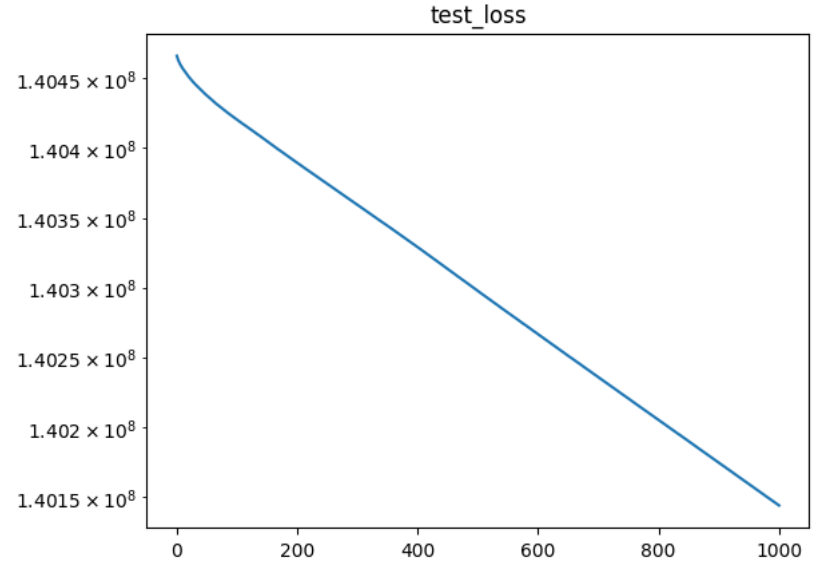


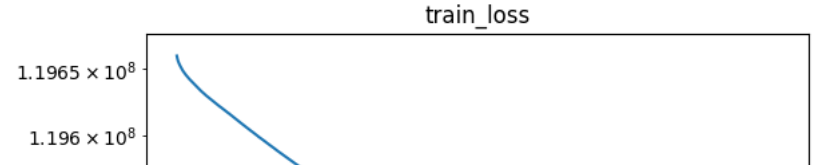
optimizer: RMSprop , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-07 , minimum_RMSE: 11851.93 , epoch: 1000 , test_loss: 11851 , train_l

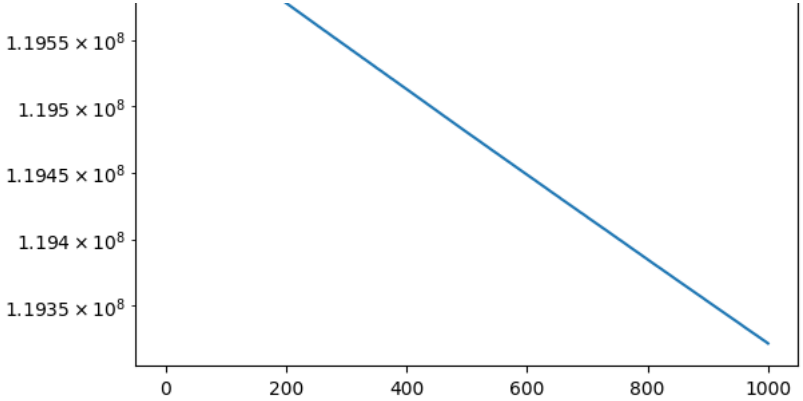




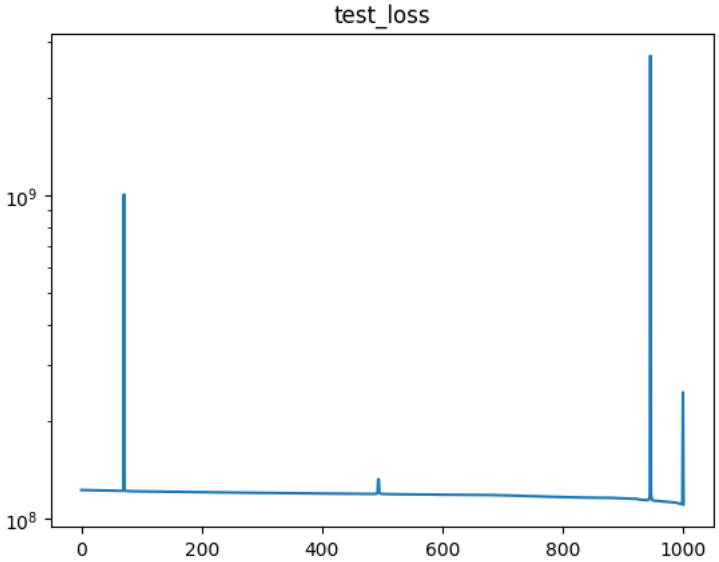
optimizer: RMSprop , n_of_data: 1000 , Bsize: 500 , learningRate: 5e-07 , minimum_RMSE: 11838.25 , epoch: 1000 , test_loss: 11838 , train_l

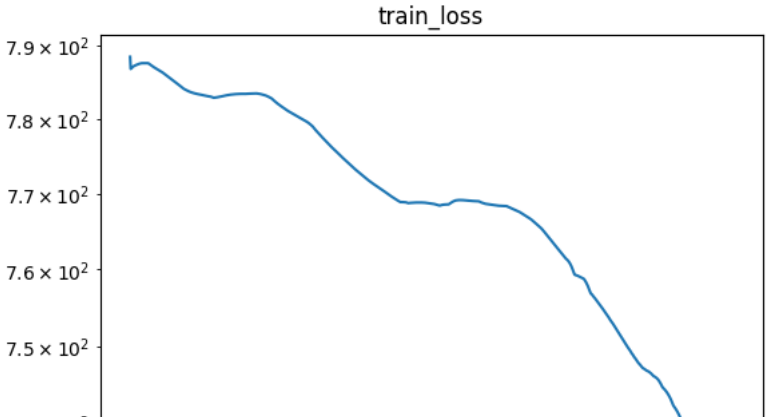


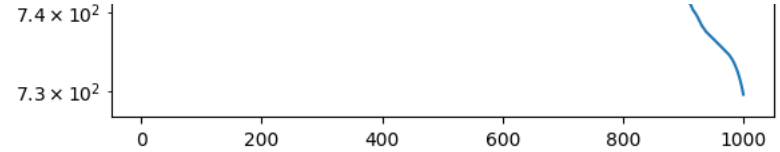




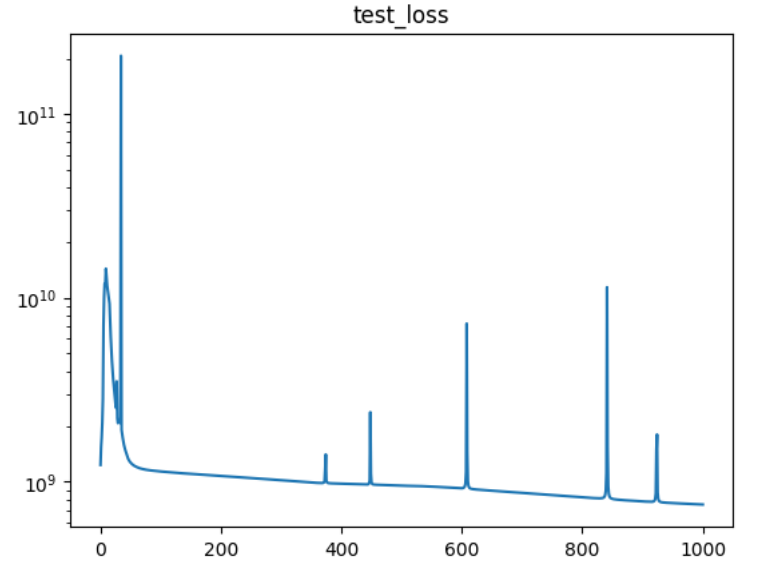
optimizer: Rprop , n_of_data: 1000 , Bsize: 10 , learningRate: 0.1 , minimum_RMSE: 10526.33 , epoch: 1000 , test_loss: 10526 , train_loss:

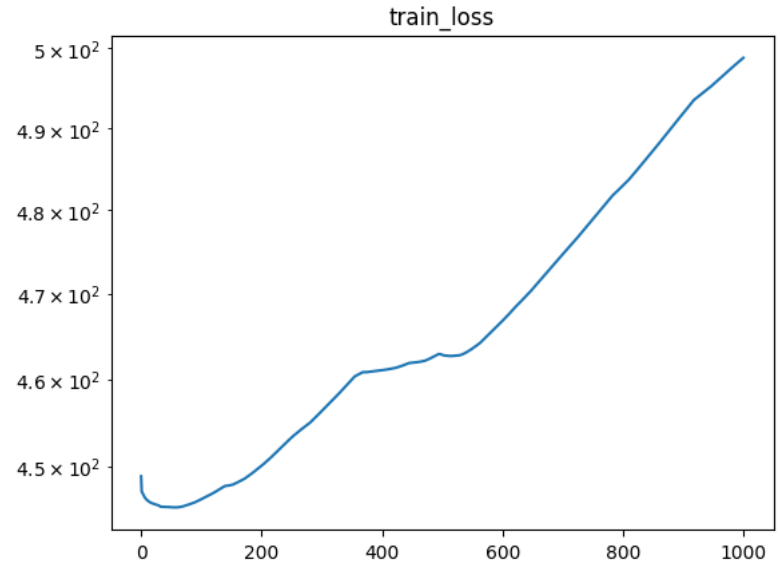




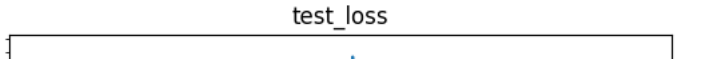


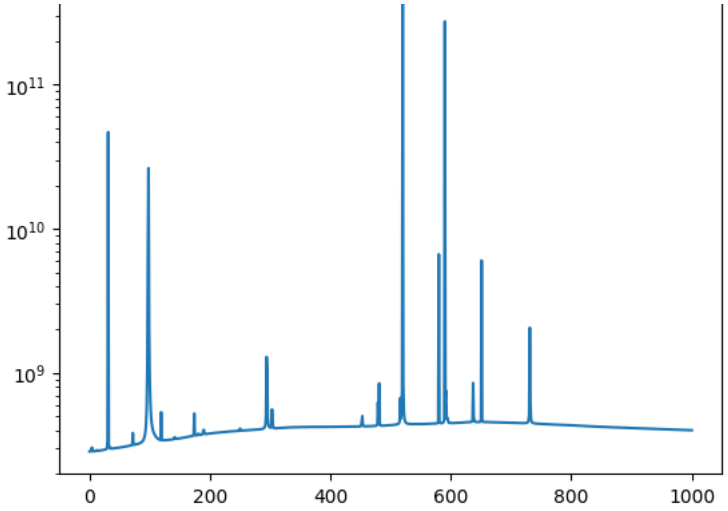
optimizer: Rprop , n_of_data: 1000 , Bsize: 10 , learningRate: 0.5 , minimum_RMSE: 27431.49 , epoch: 1000 , test_loss: 27431 , train_loss:

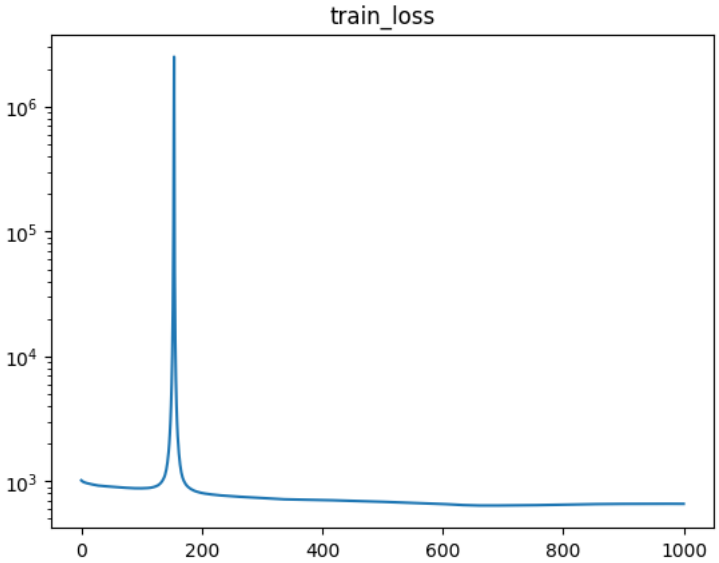




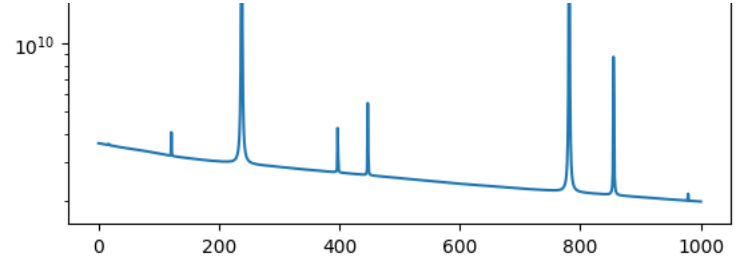
optimizer: Rprop , n_of_data: 1000 , Bsize: 10 , learningRate: 0.01 , minimum_RMSE: 16907.41 , epoch: 1000 , test_loss: 20023 , train_loss:

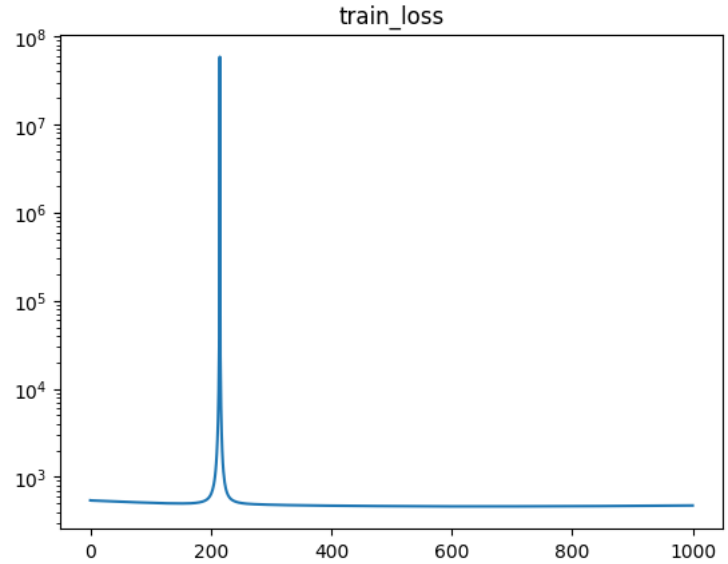




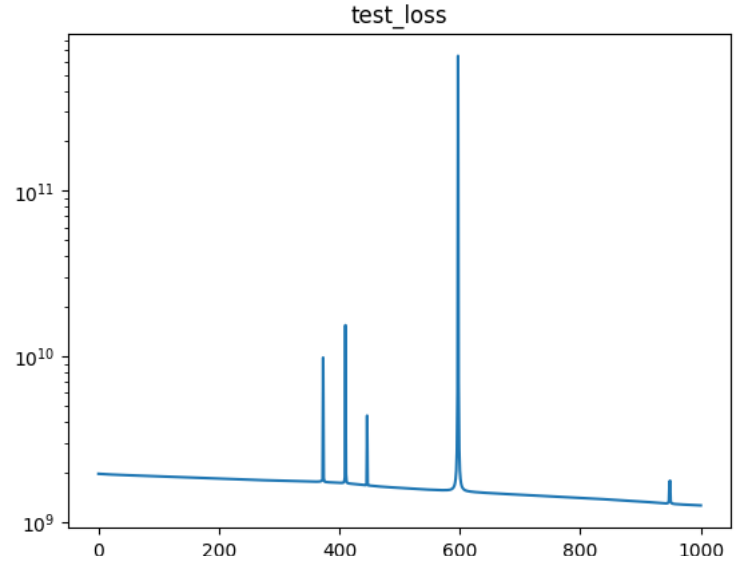


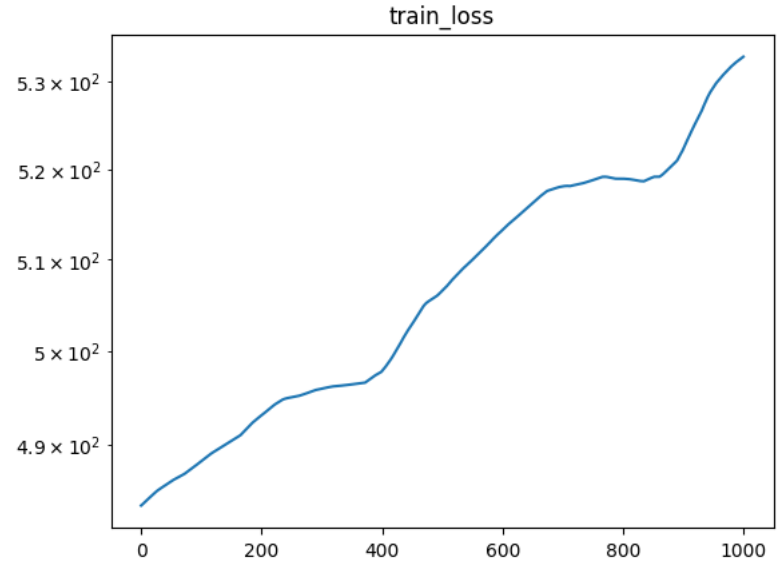
optimizer: Rprop , n_of_data: 1000 , Bsize: 10 , learningRate: 0.05 , minimum_RMSE: 44661.77 , epoch: 1000 , test_loss: 44661 , train_loss:



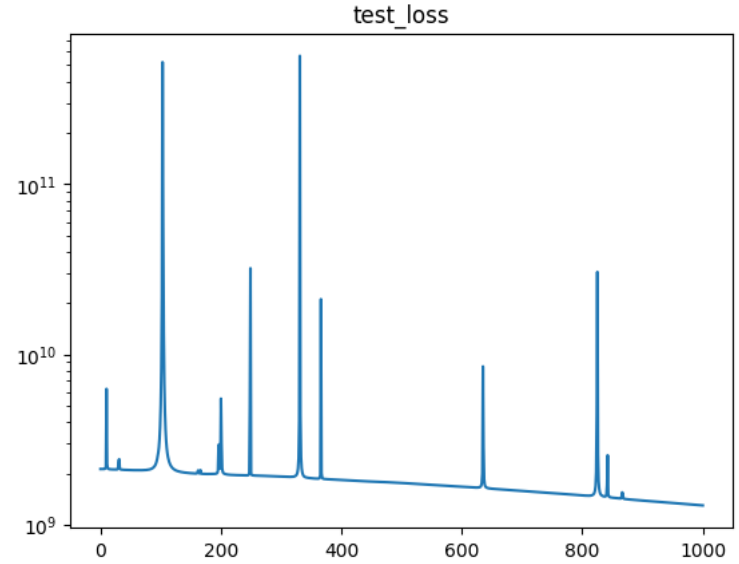


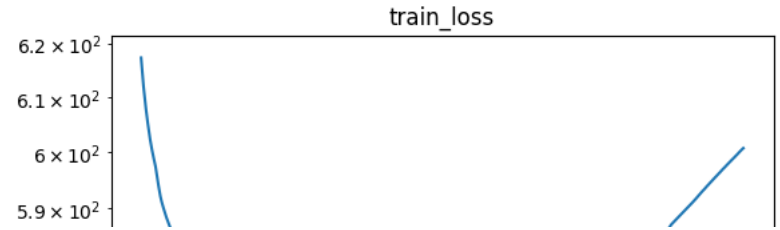
optimizer: Rprop , n_of_data: 1000 , Bsize: 10 , learningRate: 0.001 , minimum_RMSE: 35466.54 , epoch: 1000 , test_loss: 35466 , train_loss

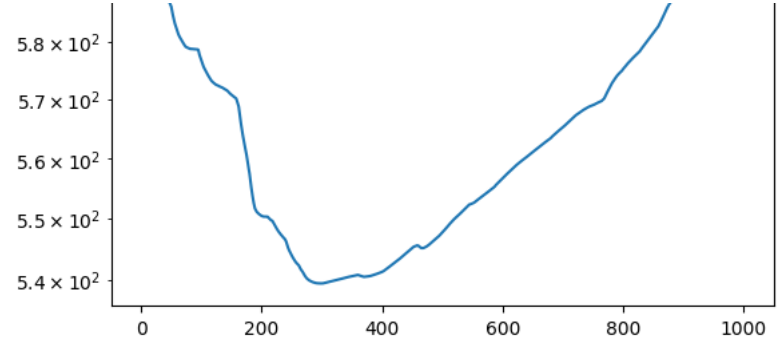




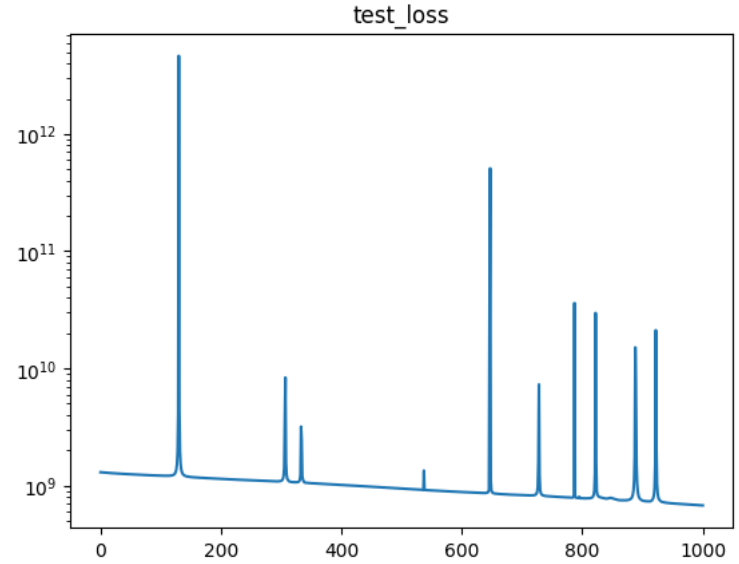
optimizer: Rprop , n_of_data: 1000 , Bsize: 10 , learningRate: 0.005 , minimum_RMSE: 36031.23 , epoch: 1000 , test_loss: 36031 , train_loss

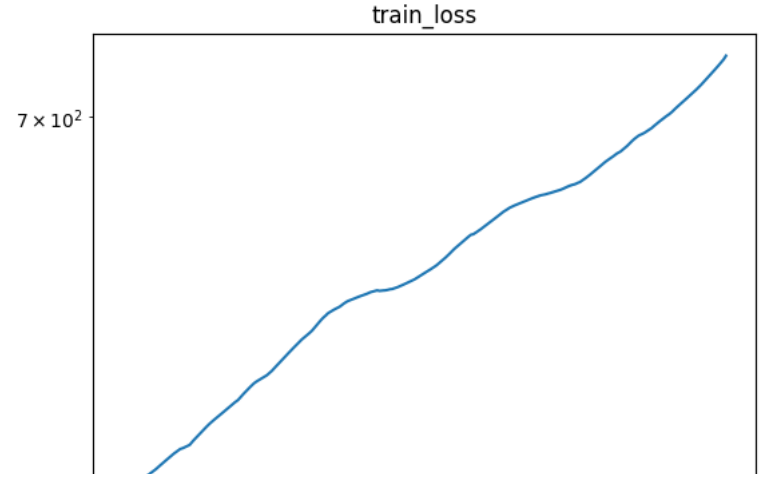


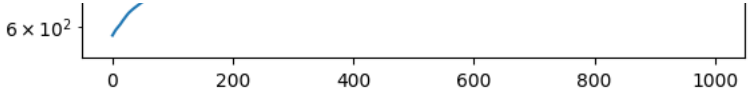




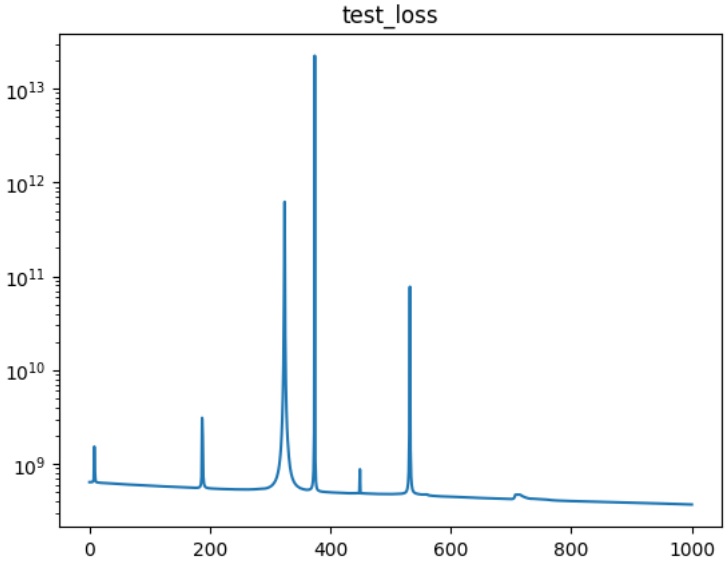
optimizer: Rprop , n_of_data: 1000 , Bsize: 10 , learningRate: 0.0001 , minimum_RMSE: 26026.60 , epoch: 1000 , test_loss: 26026 , train_los

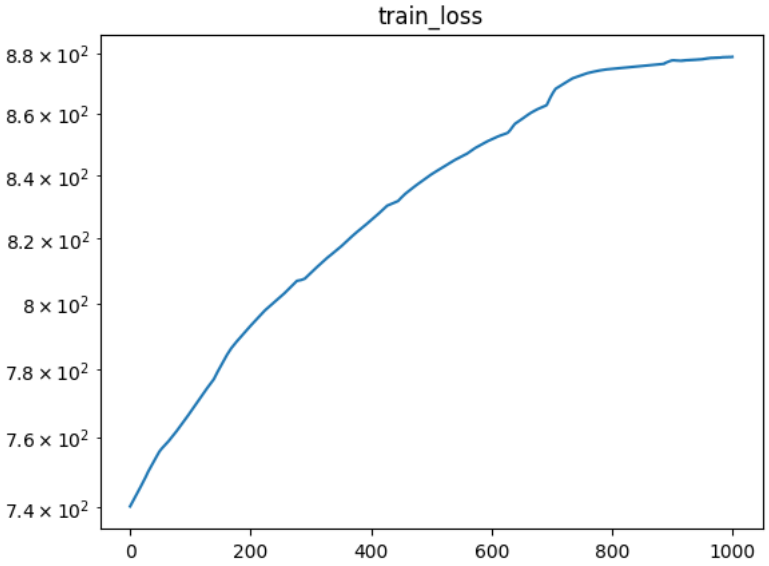




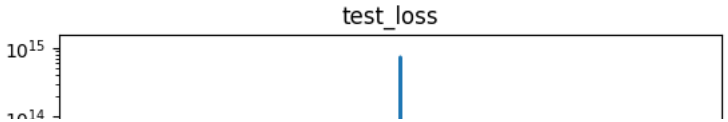


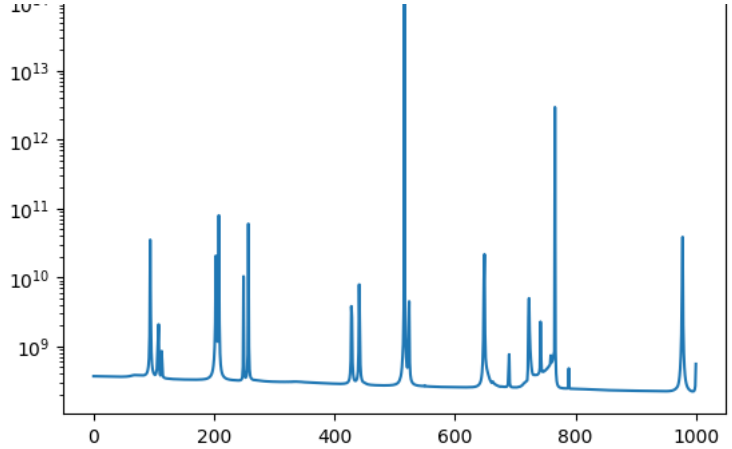
optimizer: Rprop , n_of_data: 1000 , Bsize: 10 , learningRate: 0.0005 , minimum_RMSE: 19201.55 , epoch: 1000 , test_loss: 19201 , train_los

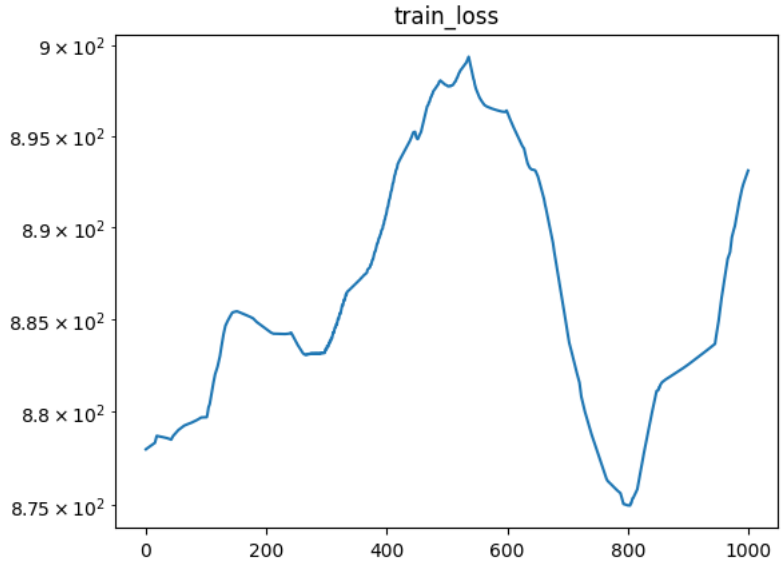




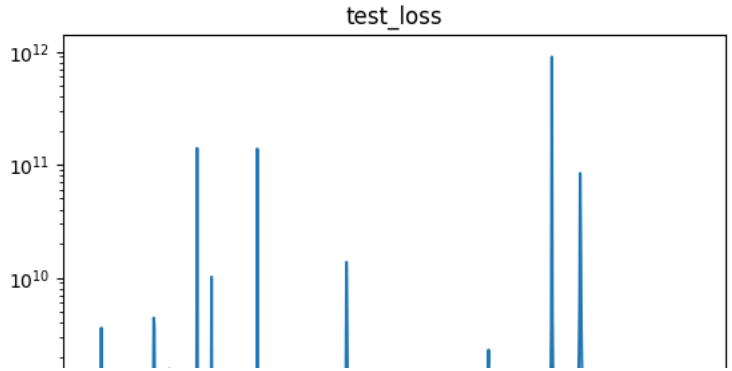
optimizer: Rprop , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-05 , minimum_RMSE: 14897.35 , epoch: 1000 , test_loss: 23565 , train_loss

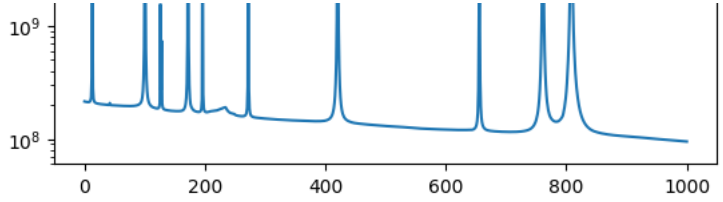


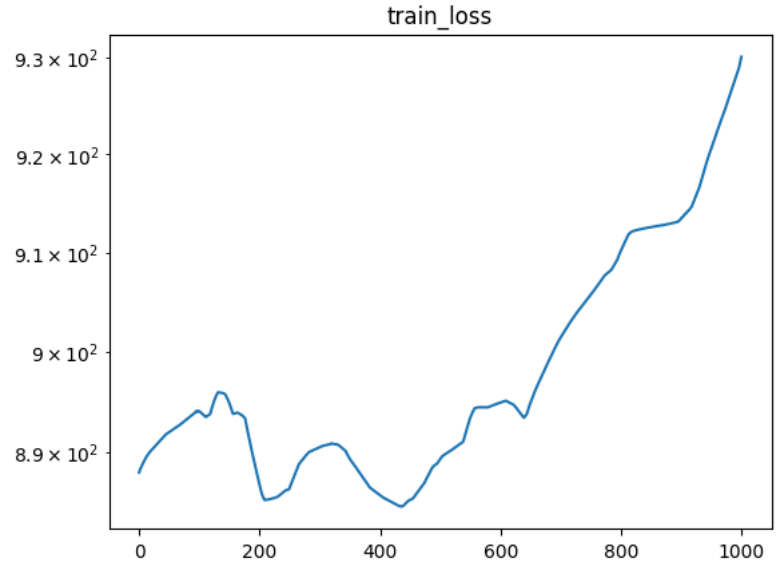




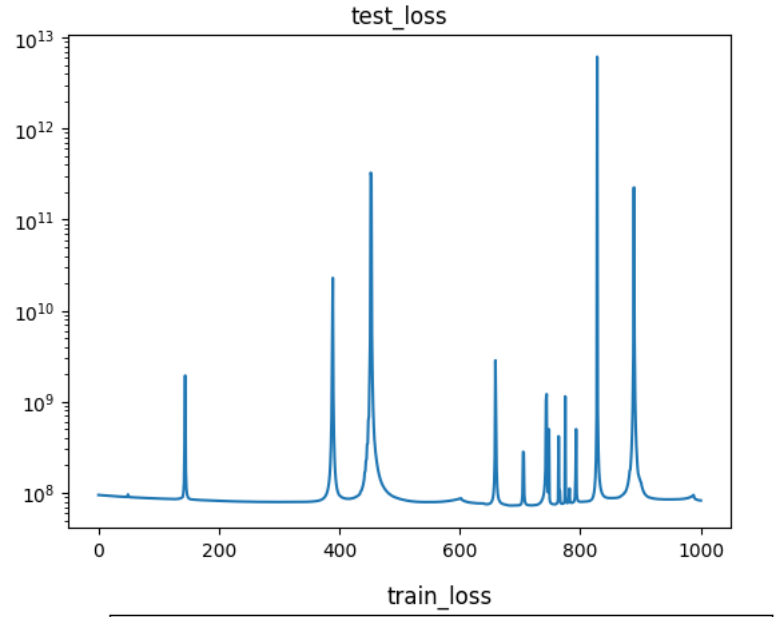
optimizer: Rprop , n_of_data: 1000 , Bsize: 10 , learningRate: 5e-05 , minimum_RMSE: 9736.63 , epoch: 1000 , test_loss: 9736 , train_loss:

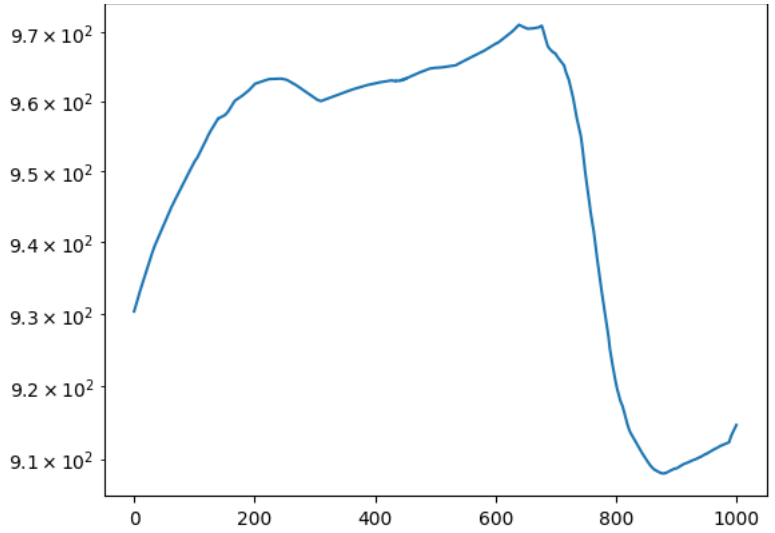




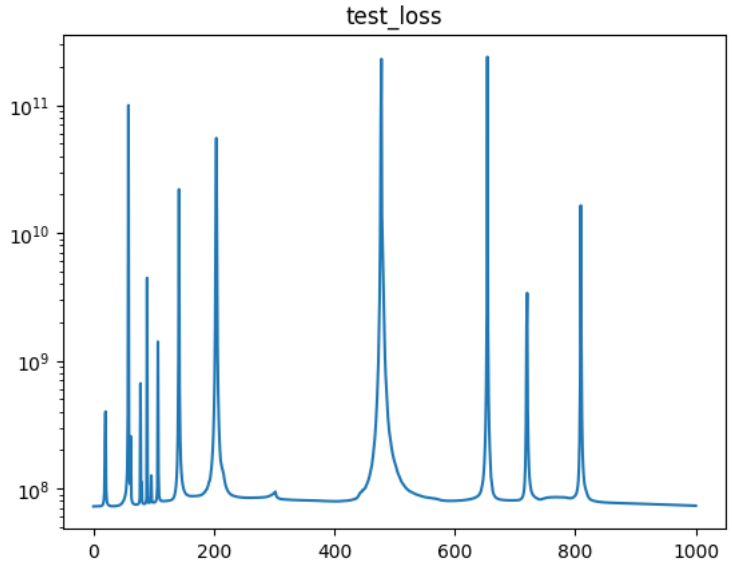


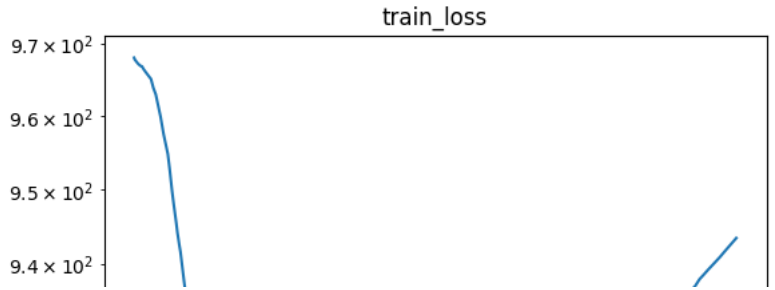
optimizer: Rprop , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-06 , minimum_RMSE: 8517.88 , epoch: 1000 , test_loss: 9079 , train_loss:

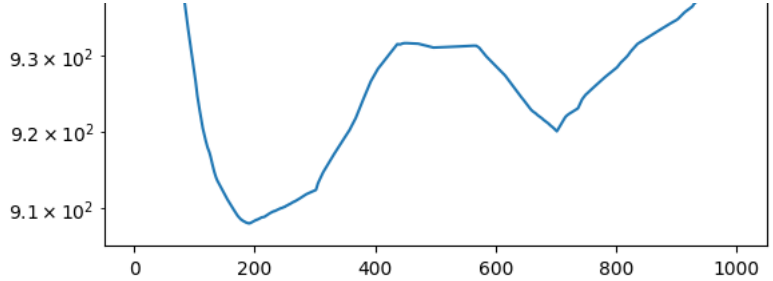




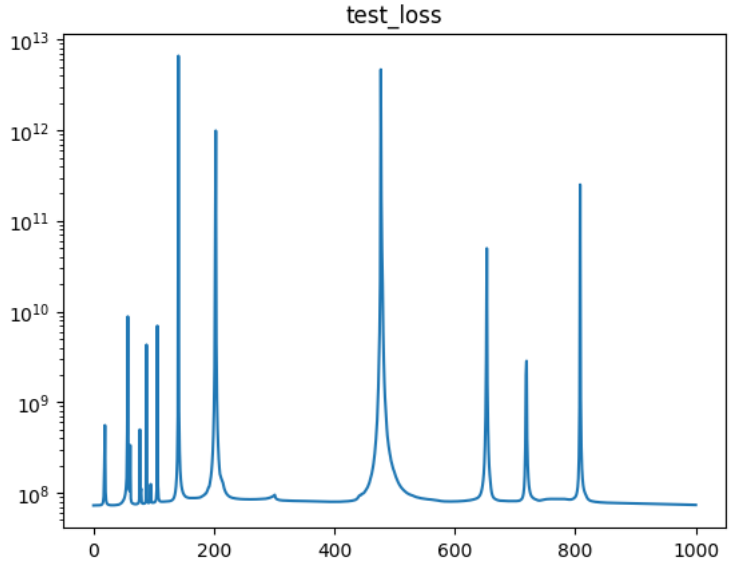
optimizer: Rprop , n_of_data: 1000 , Bsize: 10 , learningRate: 5e-06 , minimum_RMSE: 8519.53 , epoch: 1000 , test_loss: 8572 , train_loss:

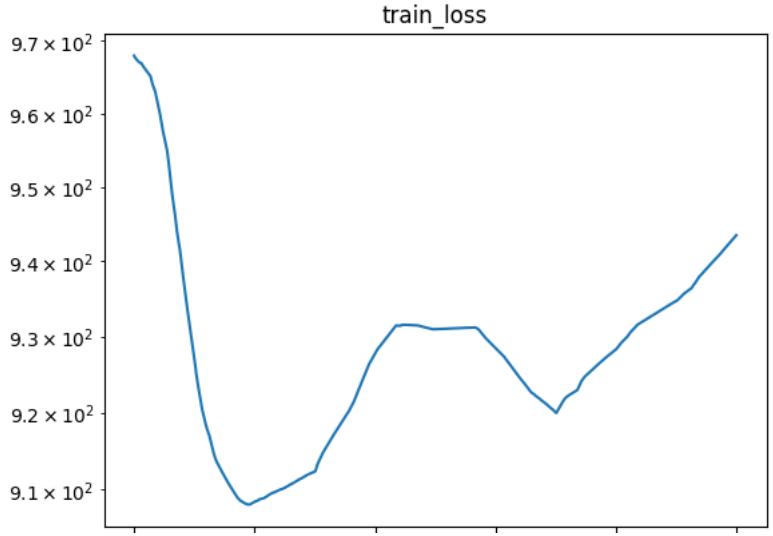




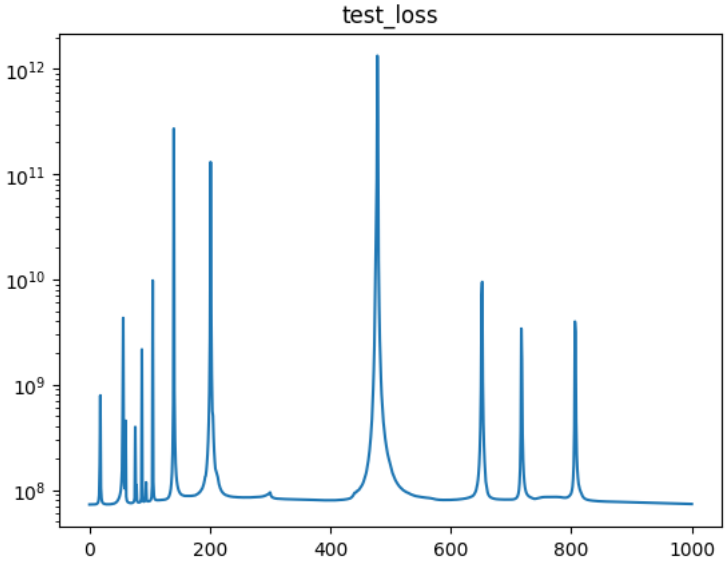


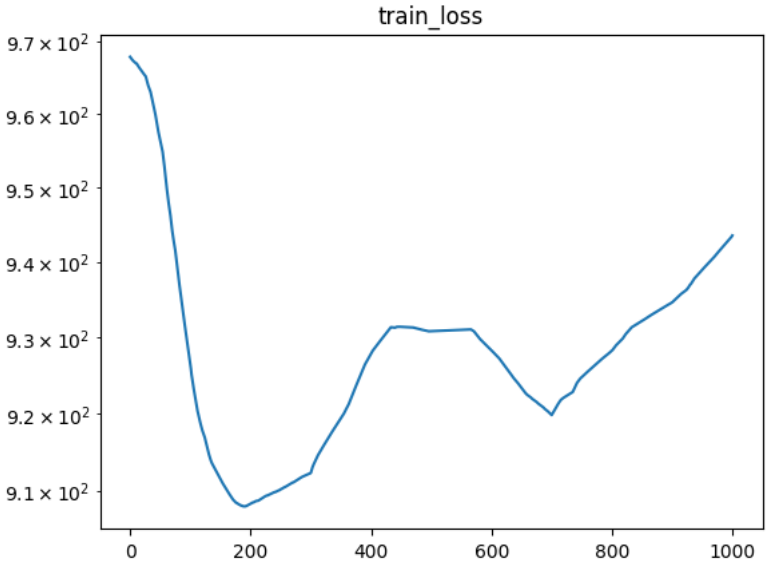
optimizer: Rprop , n_of_data: 1000 , Bsize: 10 , learningRate: 1e-07 , minimum_RMSE: 8519.86 , epoch: 1000 , test_loss: 8571 , train_loss:



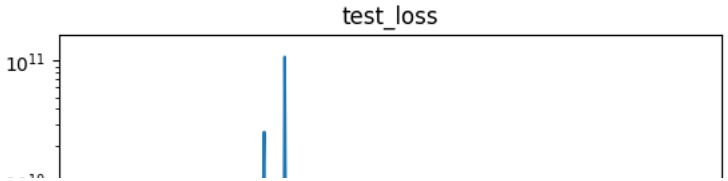


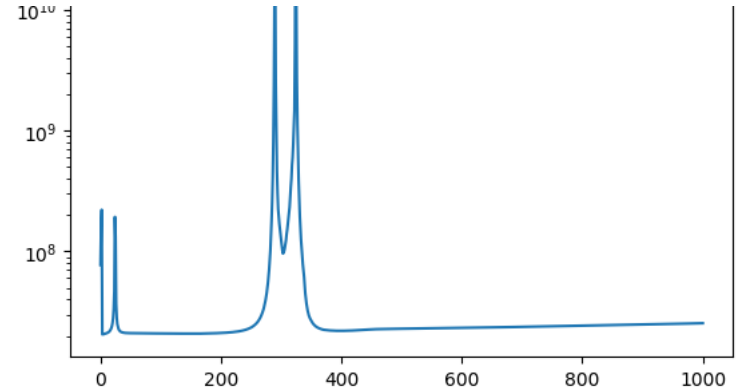
optimizer: Rprop , n_of_data: 1000 , Bsize: 10 , learningRate: 5e-07 , minimum_RMSE: 8519.97 , epoch: 1000 , test_loss: 8568 , train_loss:

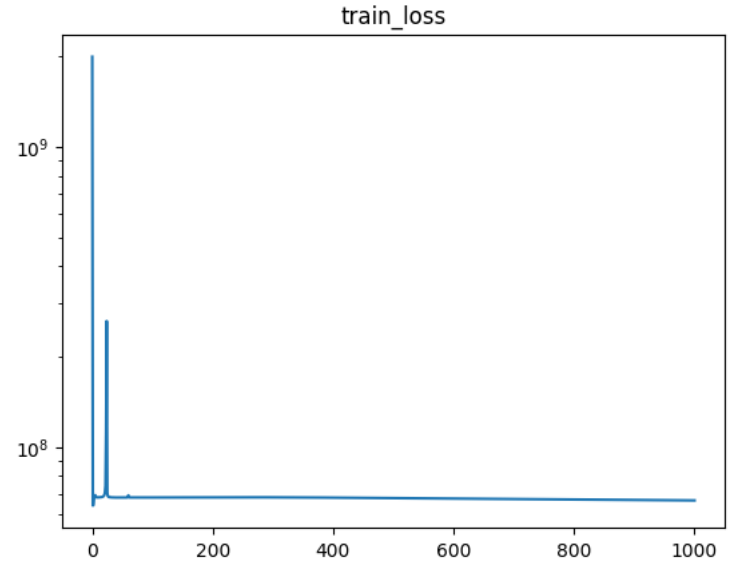




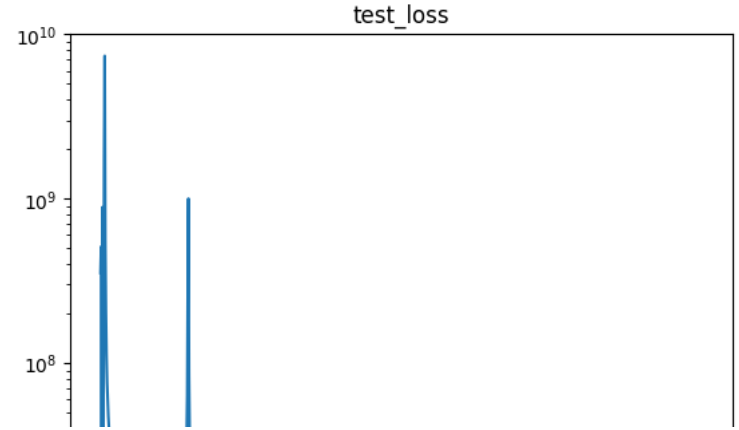
optimizer: Rprop , n_of_data: 1000 , Bsize: 50 , learningRate: 0.1 , minimum_RMSE: 4533.11 , epoch: 1000 , test_loss: 5043 , train_loss:

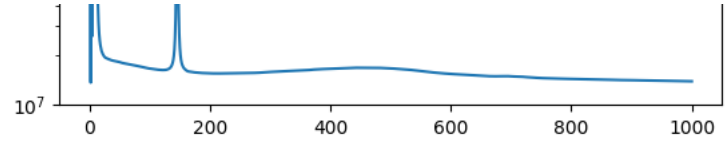


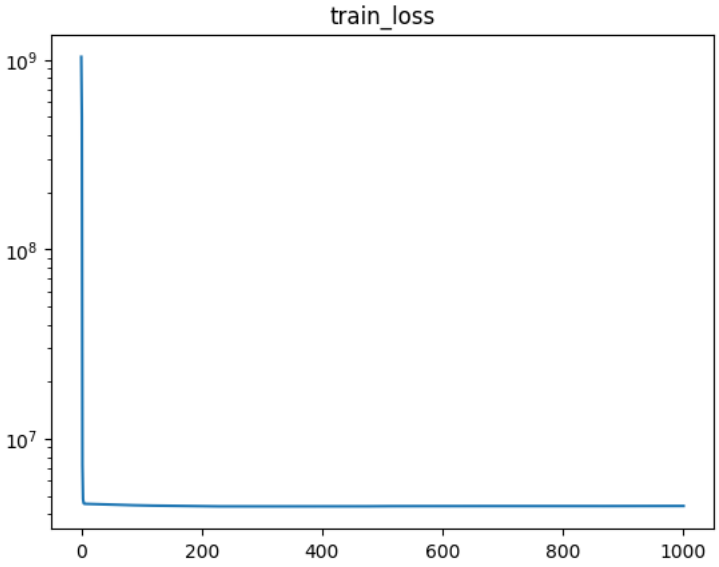




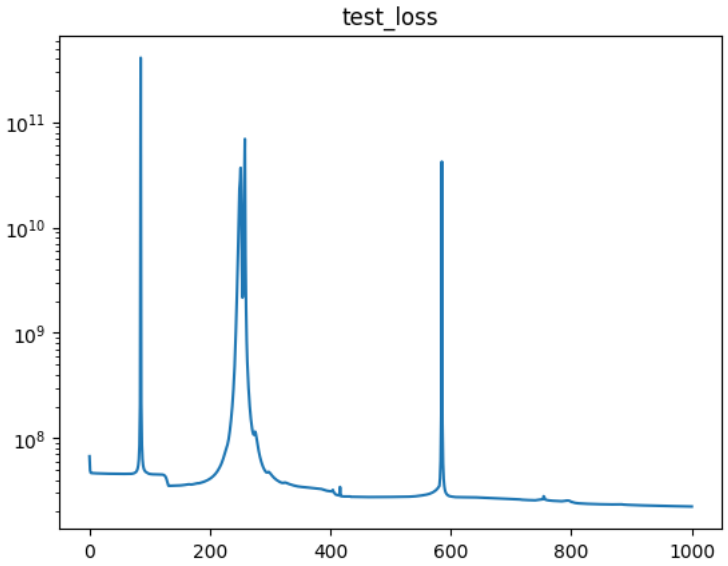
optimizer: Rprop , n_of_data: 1000 , Bsize: 50 , learningRate: 0.5 , minimum_RMSE: 3684.49 , epoch: 1000 , test_loss: 3716 , train_loss:

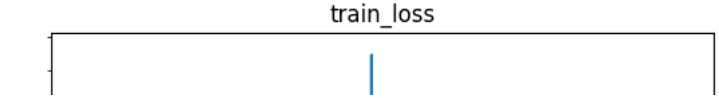


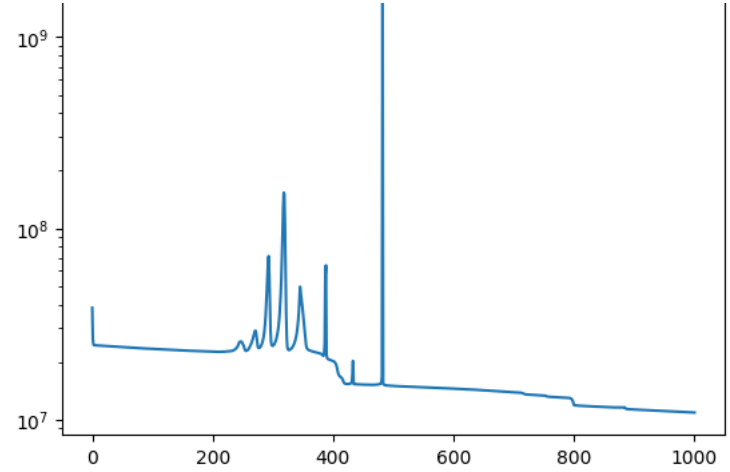




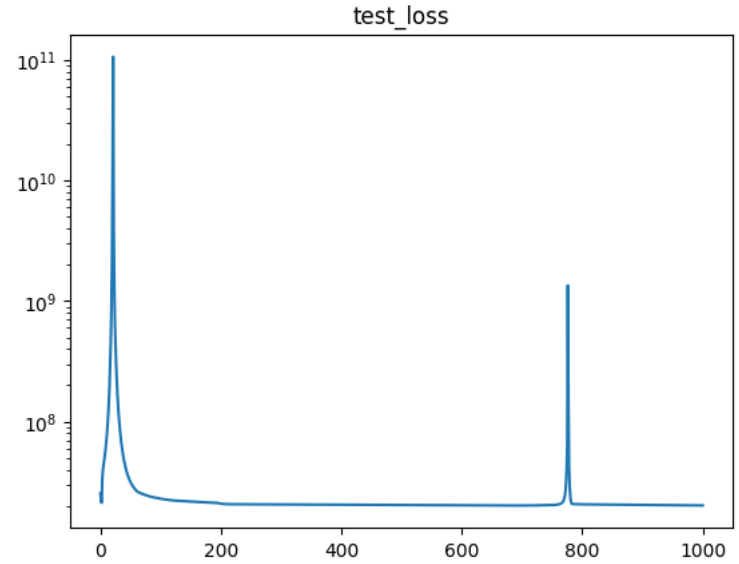
optimizer: Rprop , n_of_data: 1000 , Bsize: 50 , learningRate: 0.01 , minimum_RMSE: 4730.37 , epoch: 1000 , test_loss: 4730 , train_loss:

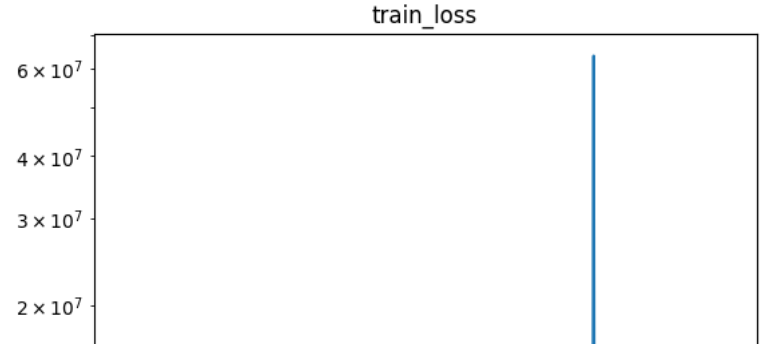


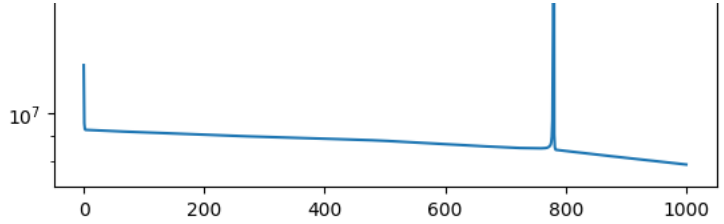




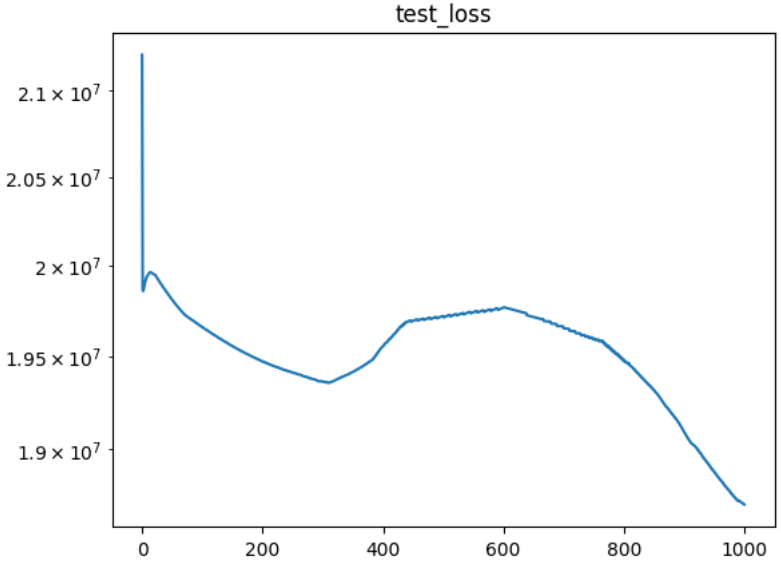
optimizer: Rprop , n_of_data: 1000 , Bsize: 50 , learningRate: 0.05 , minimum_RMSE: 4486.07 , epoch: 1000 , test_loss: 4489 , train_loss:

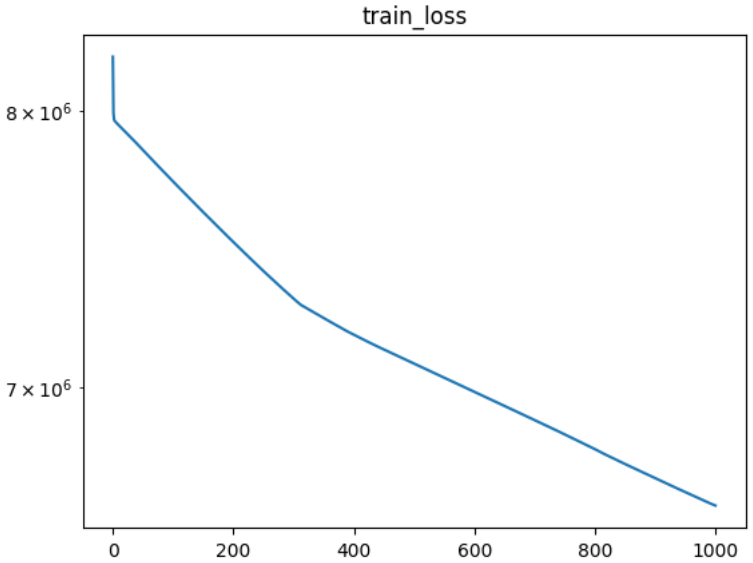




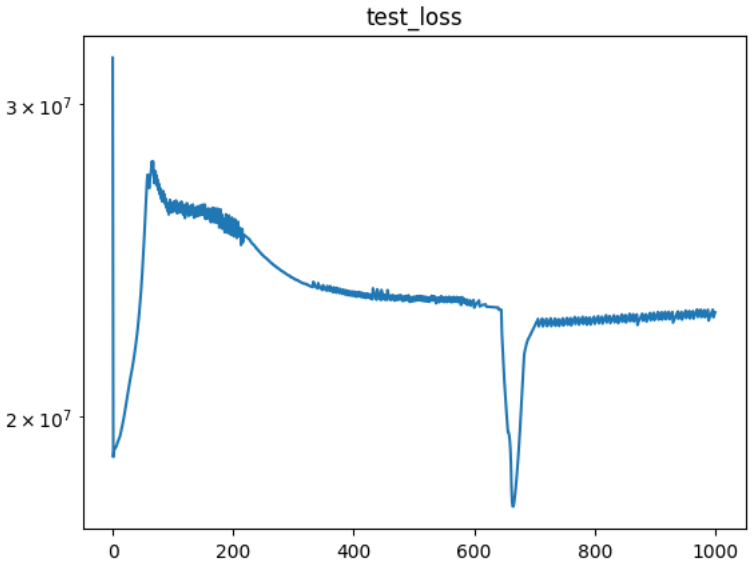


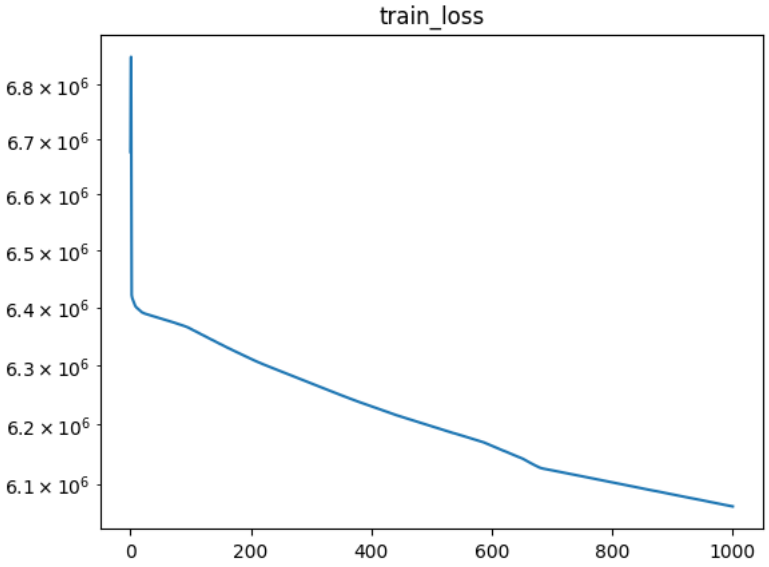
optimizer: Rprop , n_of_data: 1000 , Bsize: 50 , learningRate: 0.001 , minimum_RMSE: 4325.70 , epoch: 1000 , test_loss: 4325 , train_loss:



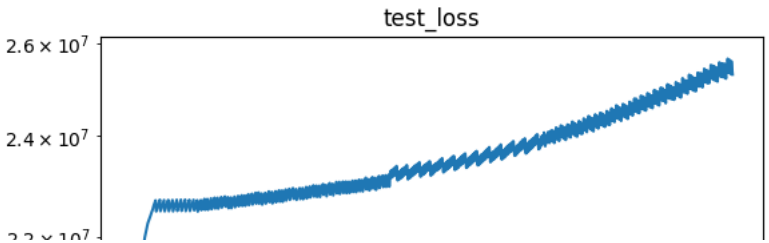


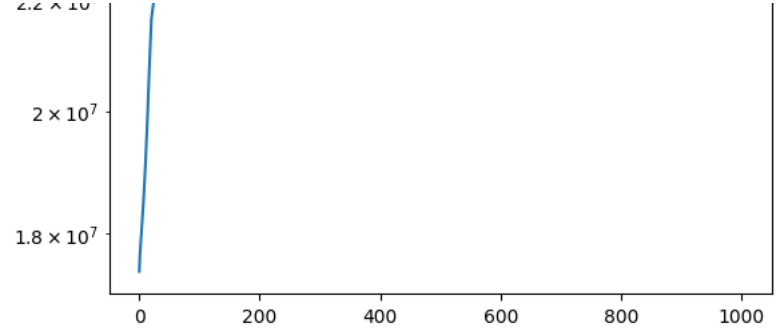
optimizer: Rprop , n_of_data: 1000 , Bsize: 50 , learningRate: 0.005 , minimum_RMSE: 4219.13 , epoch: 1000 , test_loss: 4784 , train_loss:

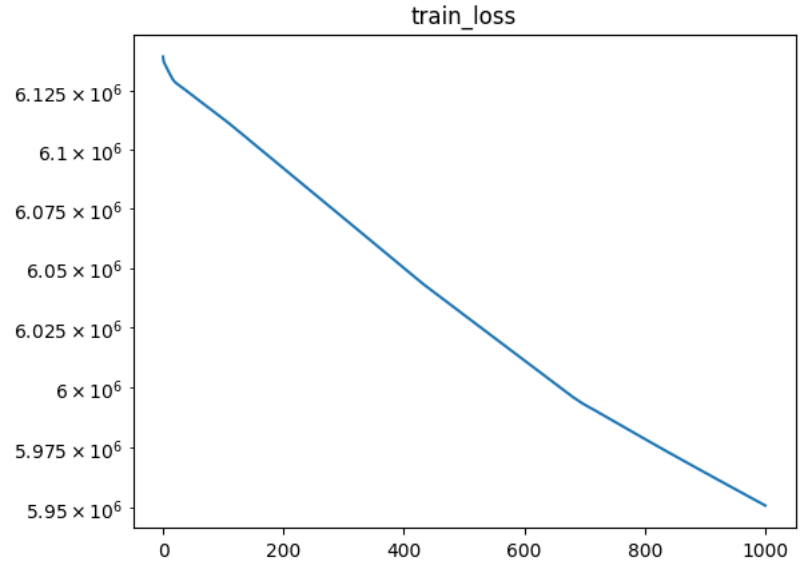




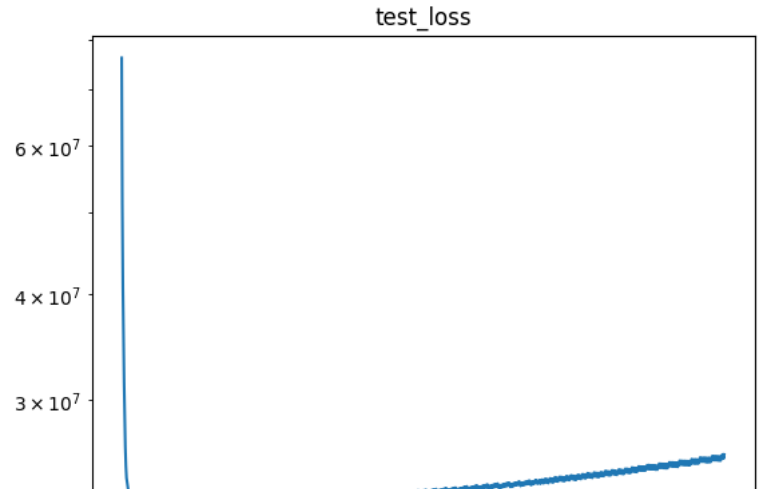
optimizer: Rprop , n_of_data: 1000 , Bsize: 50 , learningRate: 0.0001 , minimum_RMSE: 4174.00 , epoch: 1000 , test_loss: 5030 , train_loss

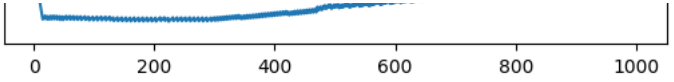


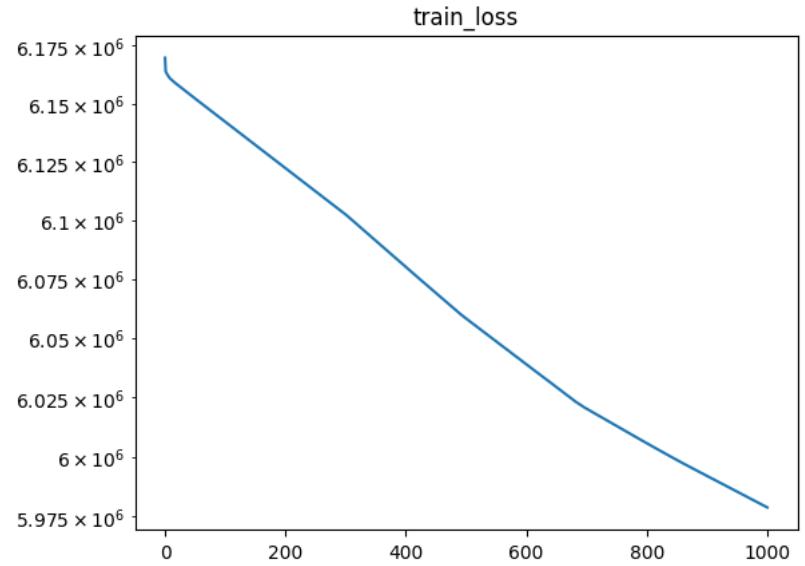




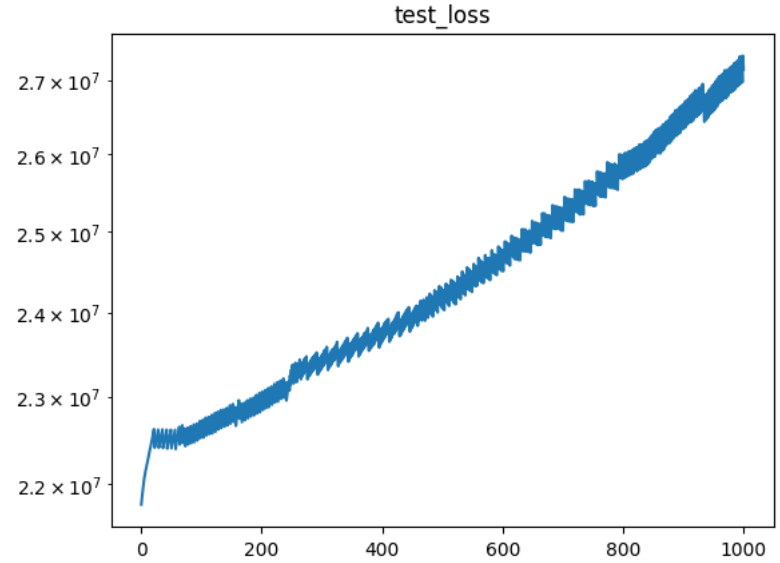
optimizer: Rprop , n_of_data: 1000 , Bsize: 50 , learningRate: 0.0005 , minimum_RMSE: 4733.87 , epoch: 1000 , test_loss: 5087 , train_loss

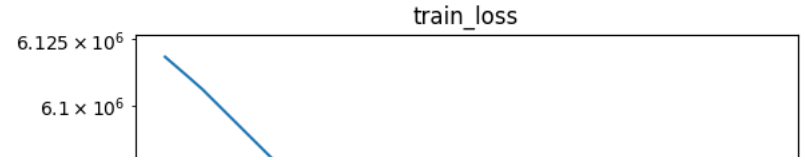


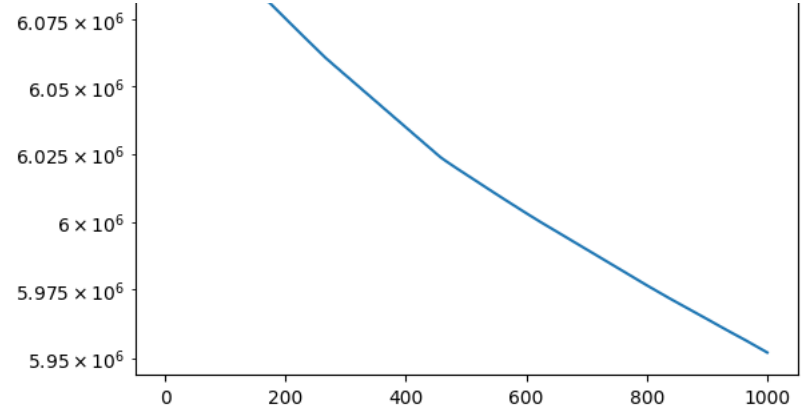




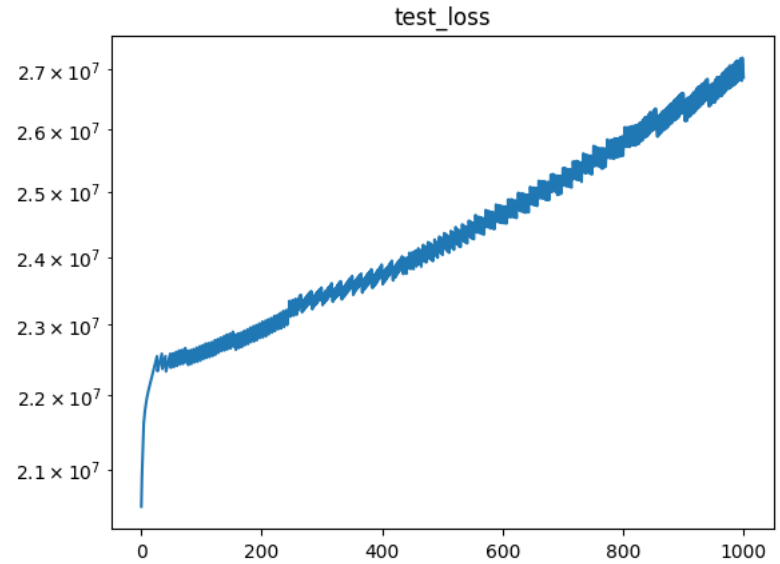
optimizer: Rprop , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-05 , minimum_RMSE: 4666.18 , epoch: 1000 , test_loss: 5210 , train_loss:

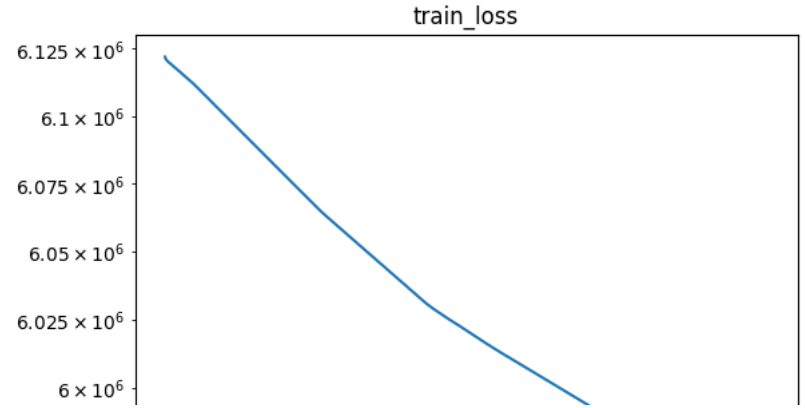


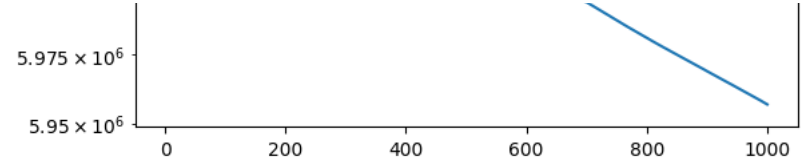




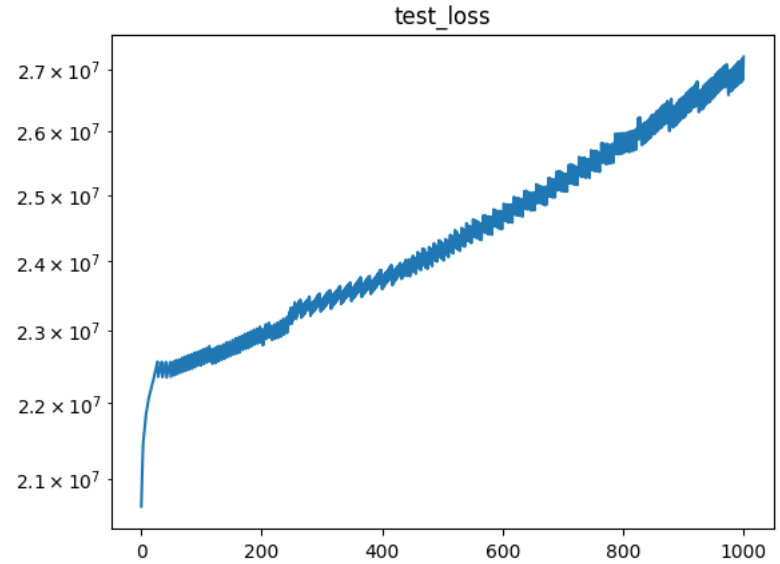
optimizer: Rprop , n_of_data: 1000 , Bsize: 50 , learningRate: 5e-05 , minimum_RMSE: 4529.03 , epoch: 1000 , test_loss: 5183 , train_loss:

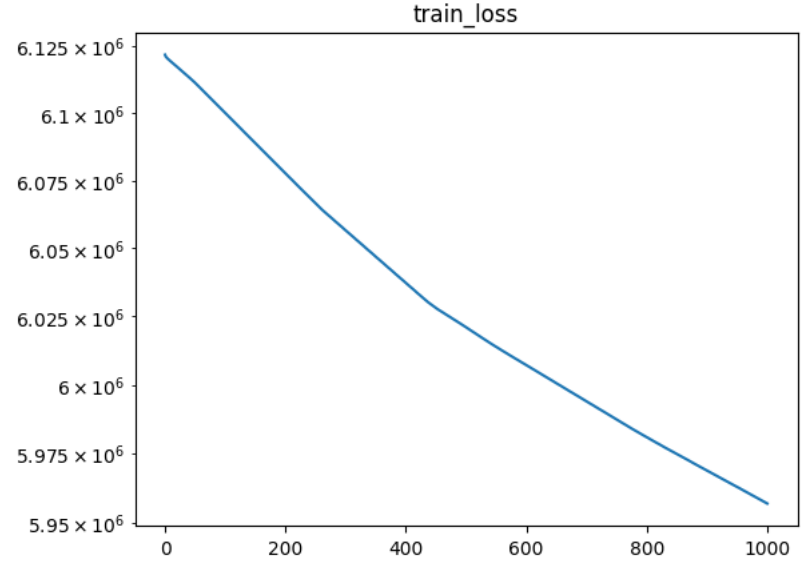




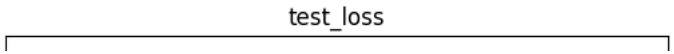


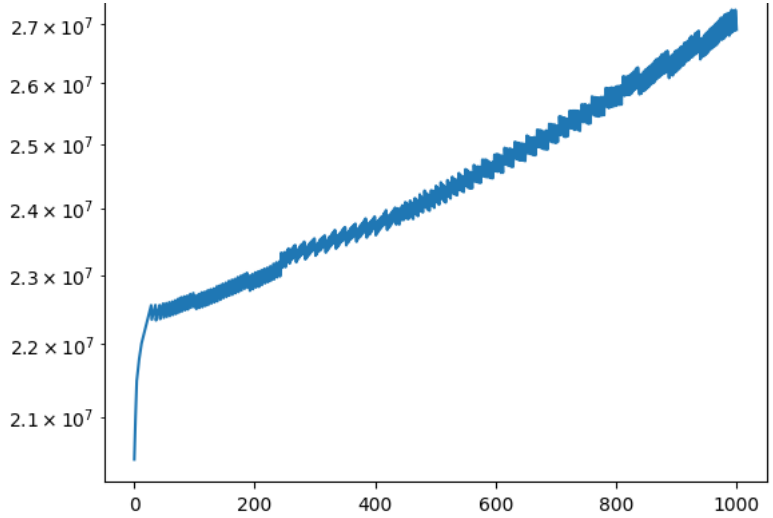
optimizer: Rprop , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-06 , minimum_RMSE: 4543.62 , epoch: 1000 , test_loss: 5216 , train_loss:

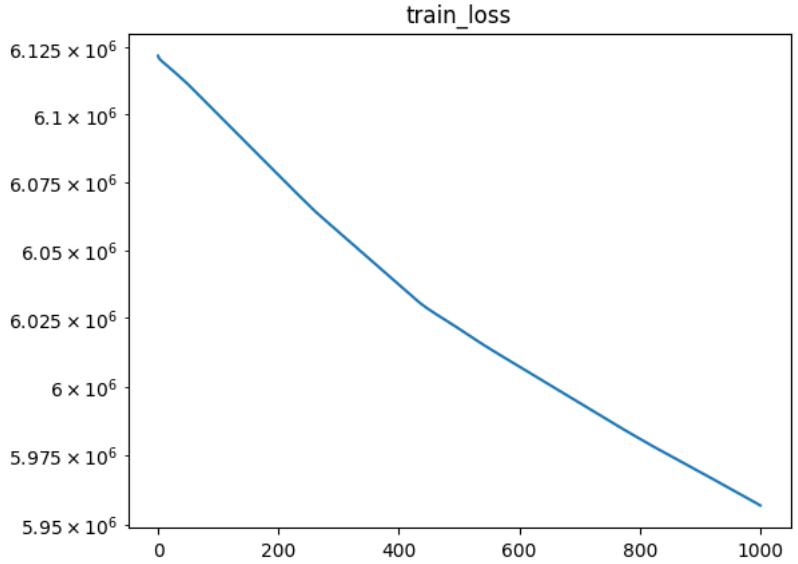




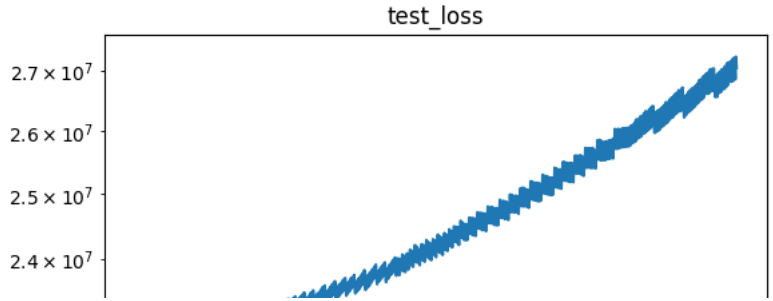
optimizer: Rprop , n_of_data: 1000 , Bsize: 50 , learningRate: 5e-06 , minimum_RMSE: 4520.99 , epoch: 1000 , test_loss: 5186 , train_loss:

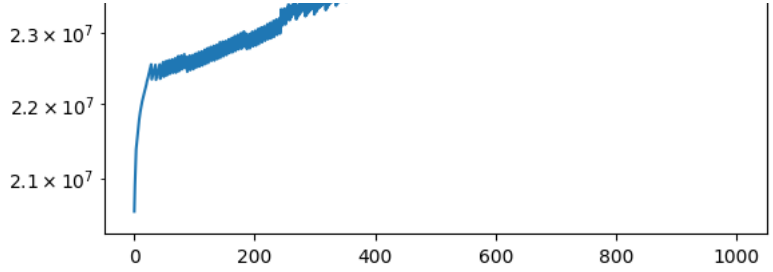


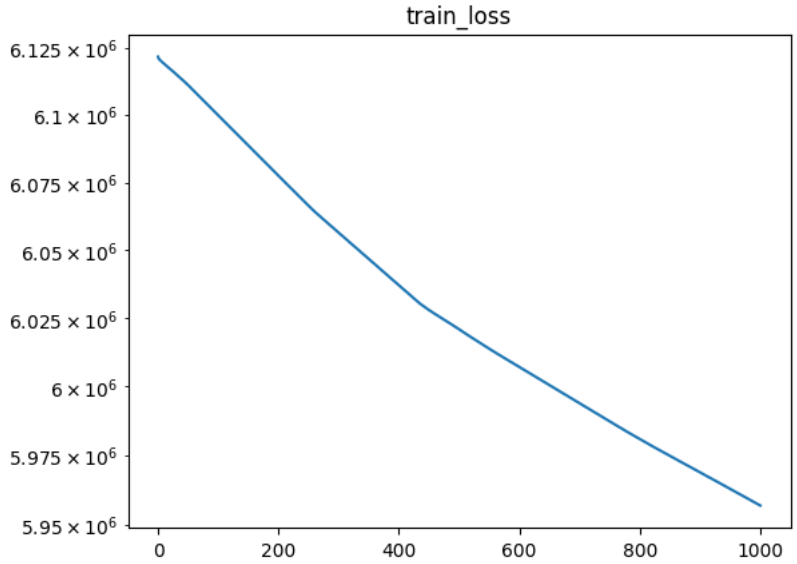




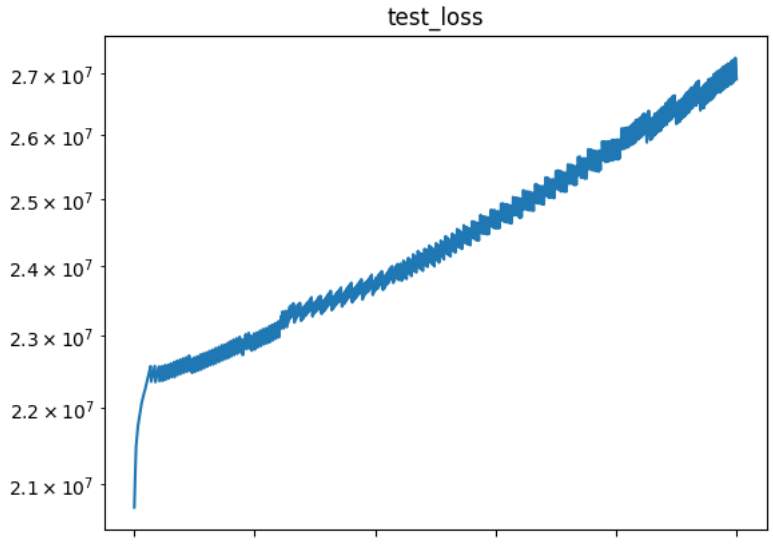
optimizer: Rprop , n_of_data: 1000 , Bsize: 50 , learningRate: 1e-07 , minimum_RMSE: 4535.82 , epoch: 1000 , test_loss: 5200 , train_loss:

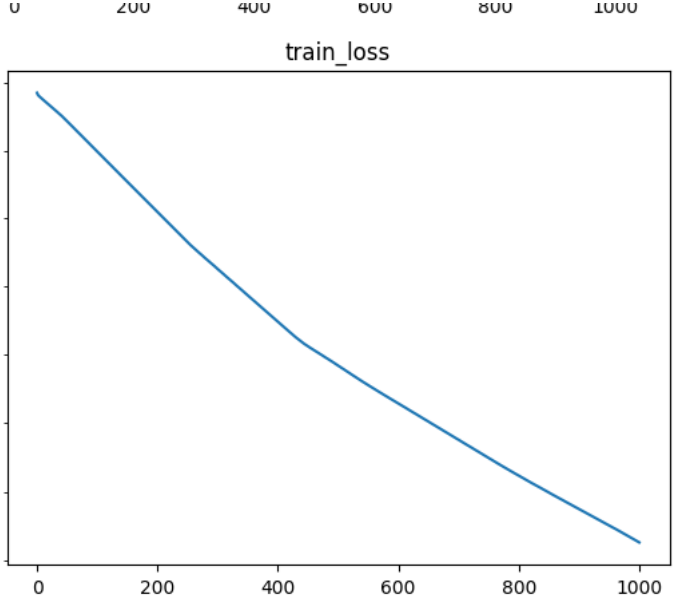




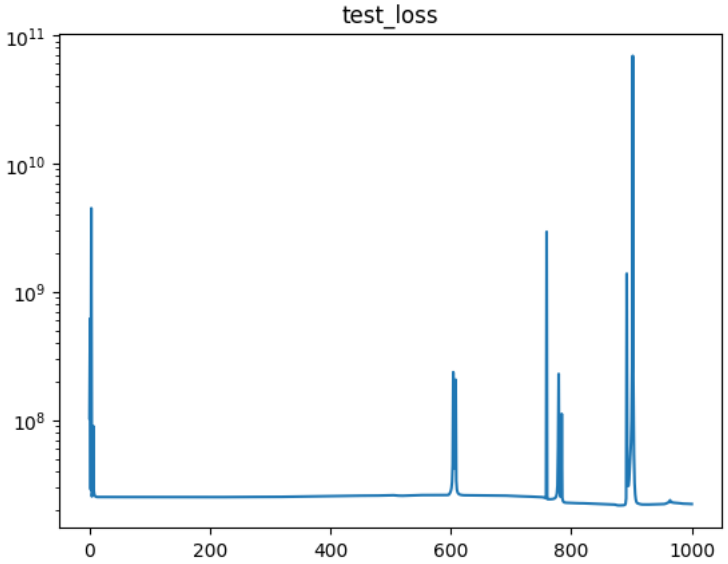


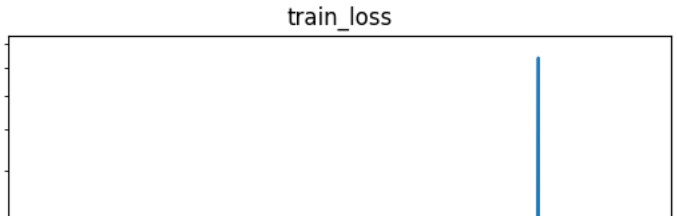
optimizer: Rprop , n_of_data: 1000 , Bsize: 50 , learningRate: 5e-07 , minimum_RMSE: 4550.40 , epoch: 1000 , test_loss: 5187 , train_loss:

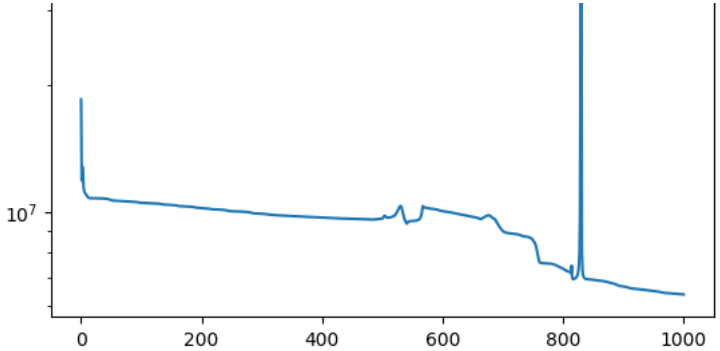




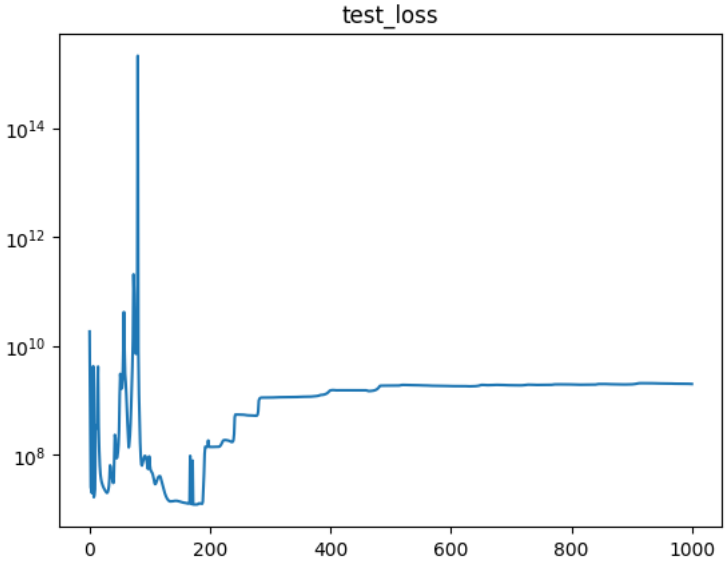
optimizer: Rprop , n_of_data: 1000 , Bsize: 100 , learningRate: 0.1 , minimum_RMSE: 4657.25 , epoch: 1000 , test_loss: 4715 , train_loss:

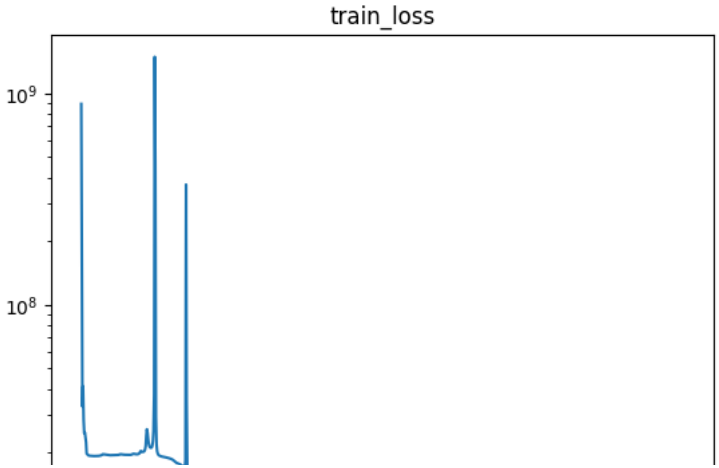


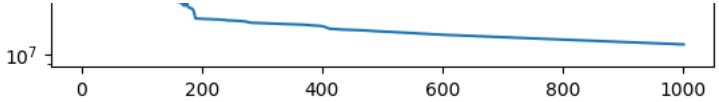




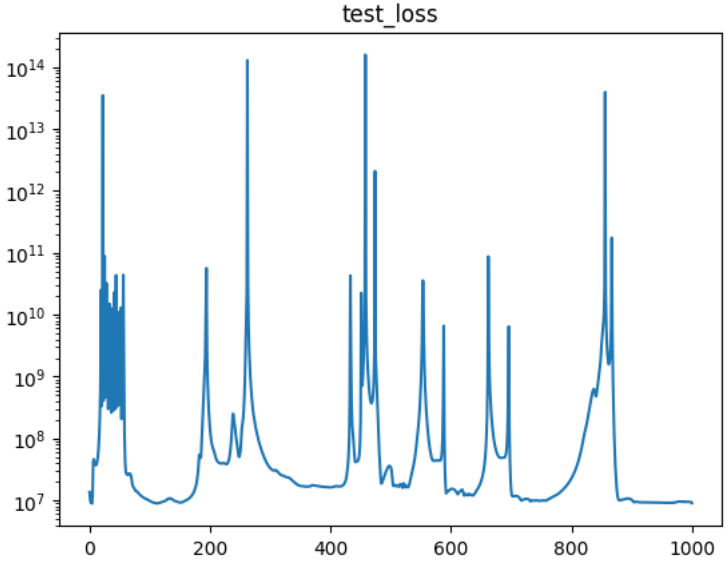
optimizer: Rprop , n_of_data: 1000 , Bsize: 100 , learningRate: 0.5 , minimum_RMSE: 3492.68 , epoch: 1000 , test_loss: 44639 , train_loss:

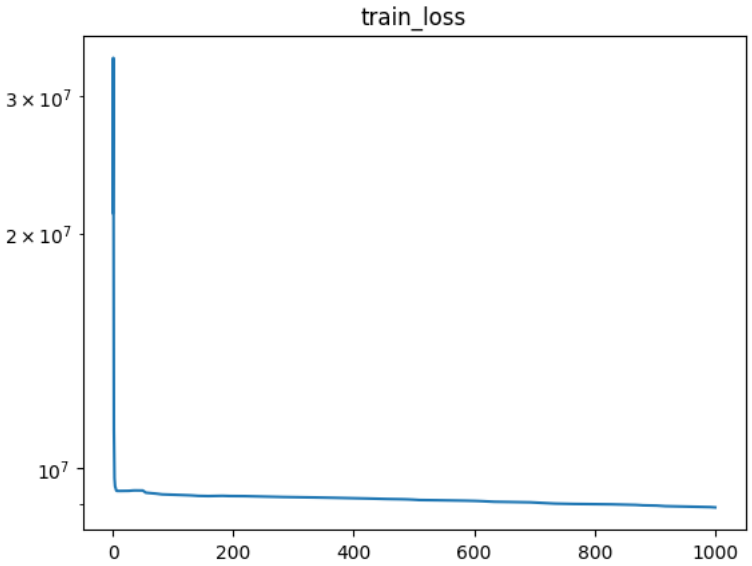




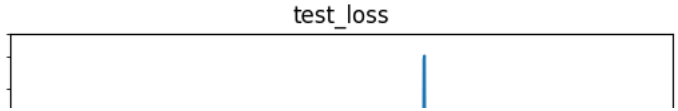


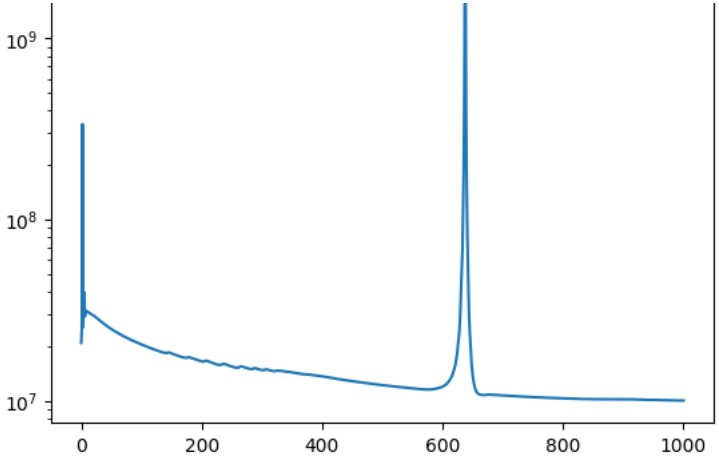
optimizer: Rprop , n_of_data: 1000 , Bsize: 100 , learningRate: 0.01 , minimum_RMSE: 2984.33 , epoch: 1000 , test_loss: 3012 , train_loss:

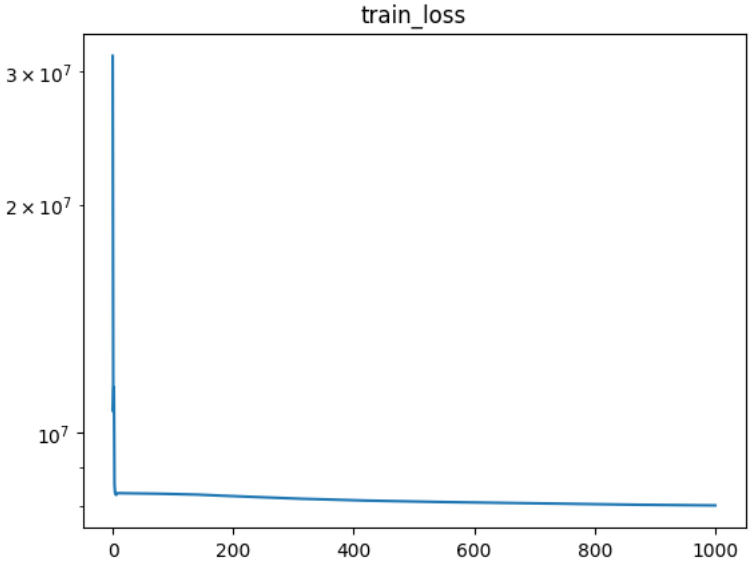




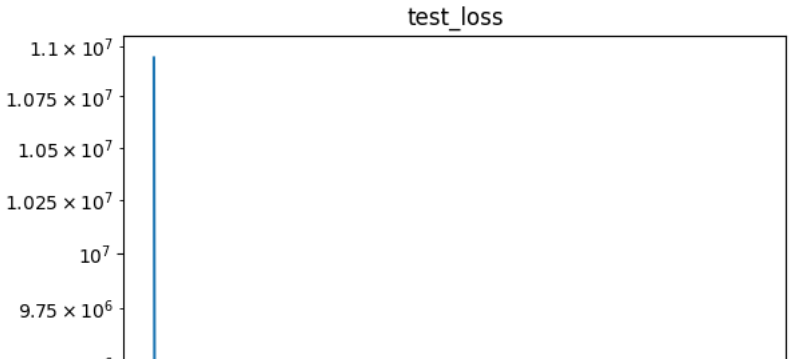
optimizer: Rprop , n_of_data: 1000 , Bsize: 100 , learningRate: 0.05 , minimum_RMSE: 3163.58 , epoch: 1000 , test_loss: 3163 , train_loss:

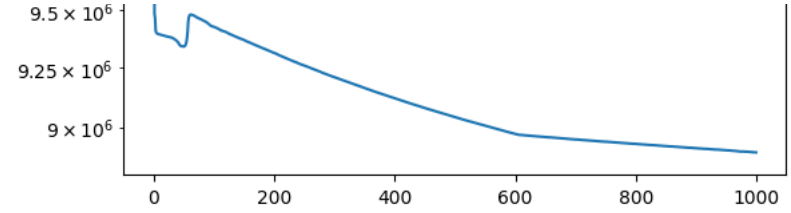


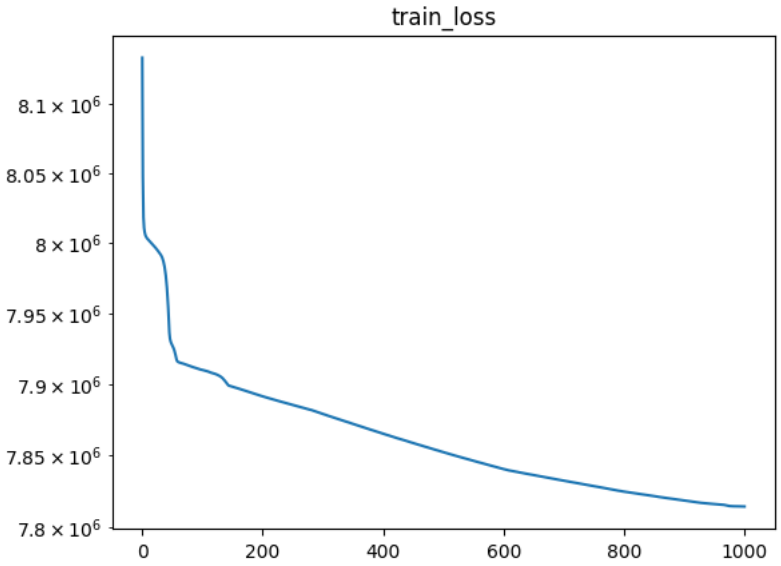




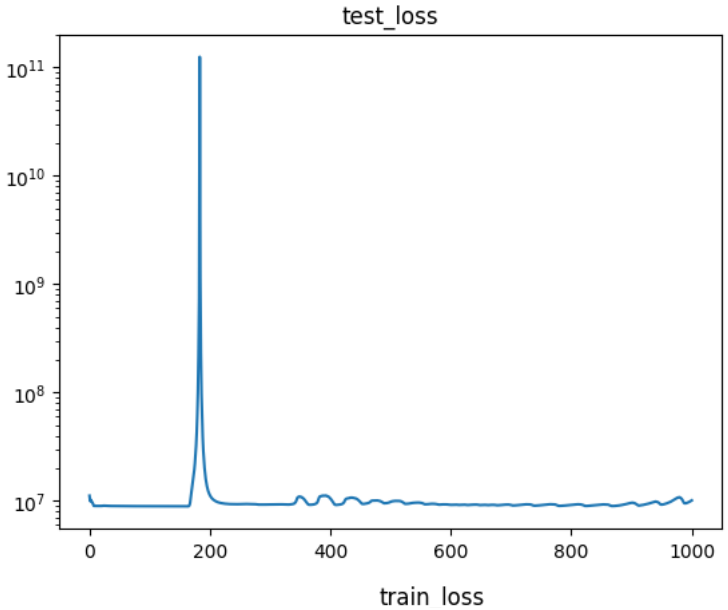
optimizer: Rprop , n_of_data: 1000 , Bsize: 100 , learningRate: 0.001 , minimum_RMSE: 2982.90 , epoch: 1000 , test_loss: 2982 , train_loss

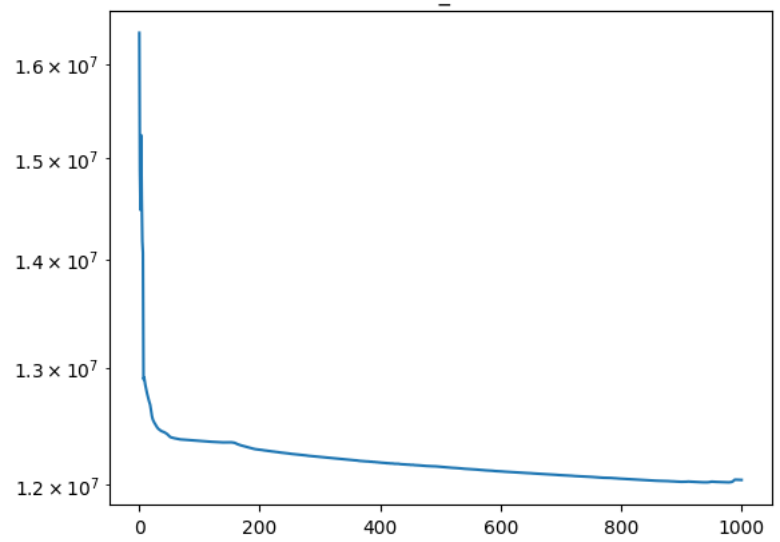




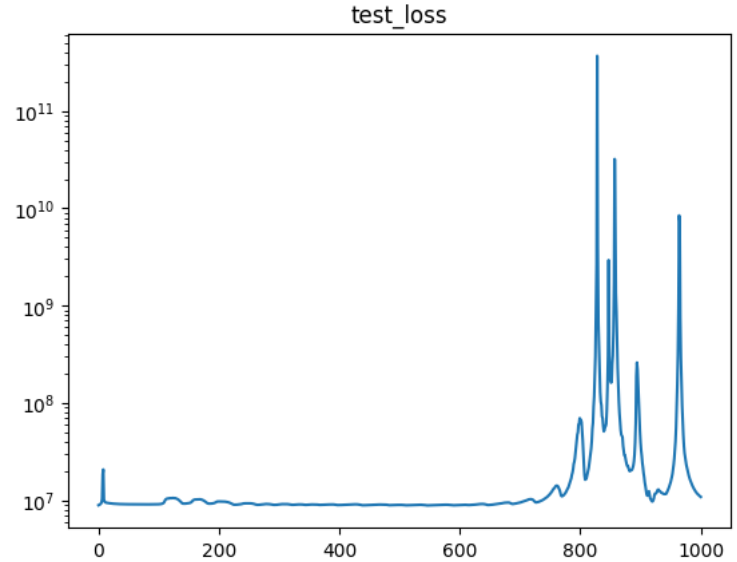


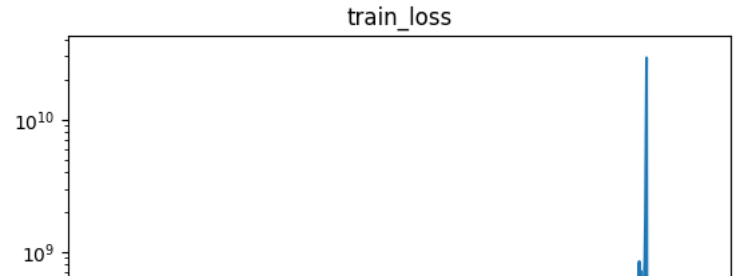
optimizer: Rprop , n_of_data: 1000 , Bsize: 100 , learningRate: 0.005 , minimum_RMSE: 2982.69 , epoch: 1000 , test_loss: 3180 , train_loss

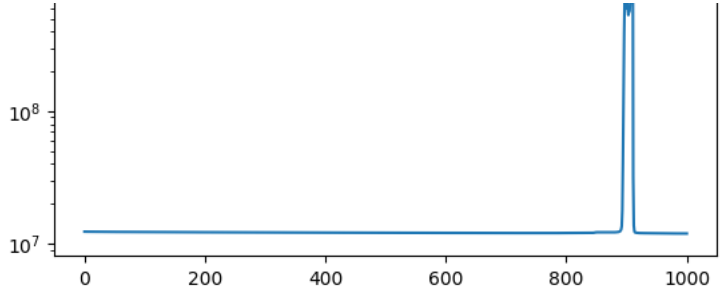




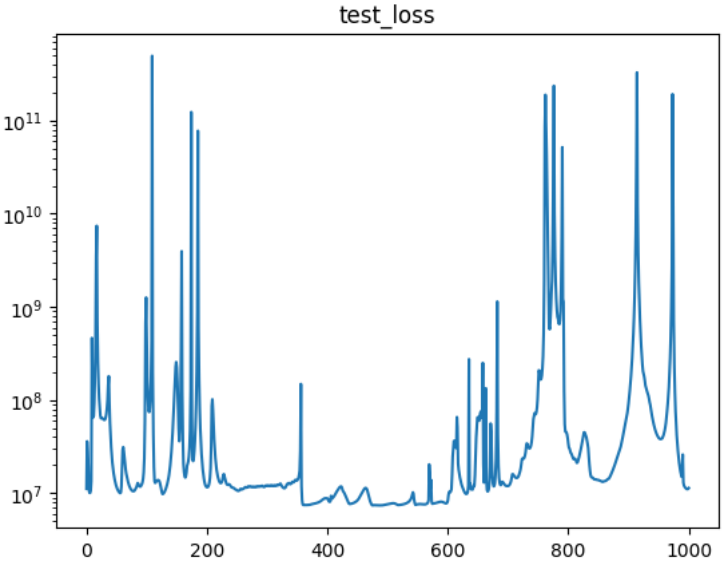
optimizer: Rprop , n_of_data: 1000 , Bsize: 100 , learningRate: 0.0001 , minimum_RMSE: 2976.20 , epoch: 1000 , test_loss: 3291 , train_los

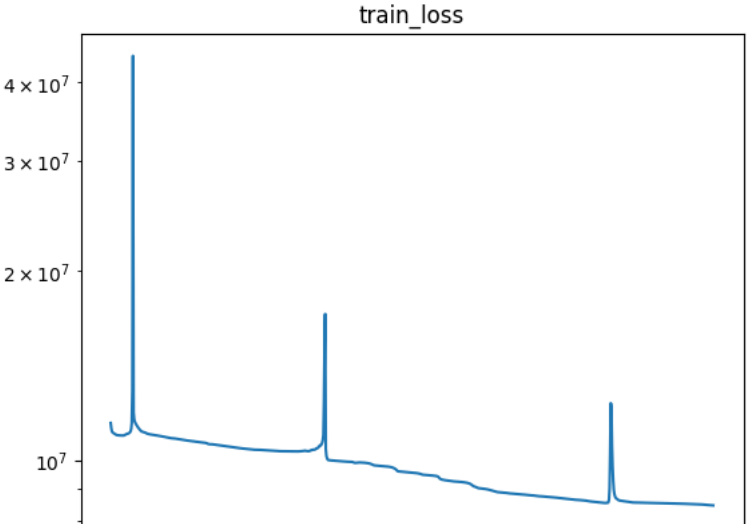




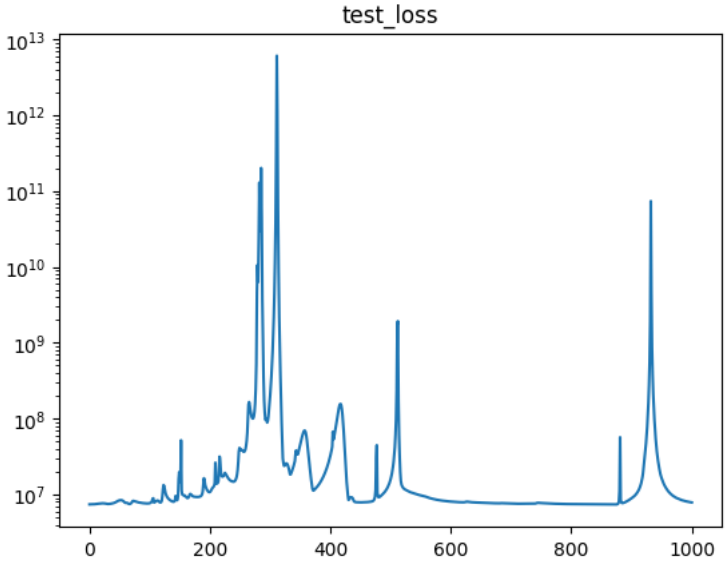


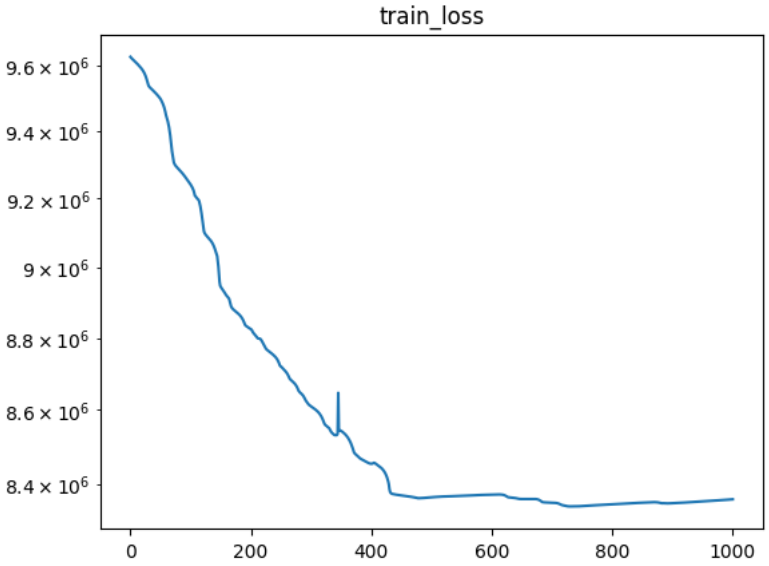
optimizer: Rprop , n_of_data: 1000 , Bsize: 100 , learningRate: 0.0005 , minimum_RMSE: 2721.14 , epoch: 1000 , test_loss: 3372 , train_los



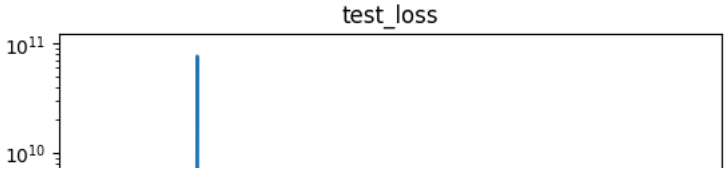


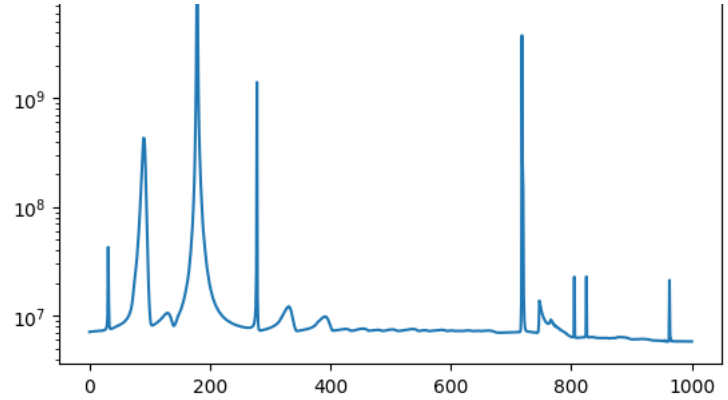
optimizer: Rprop , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-05 , minimum_RMSE: 2714.02 , epoch: 1000 , test_loss: 2801 , train_loss

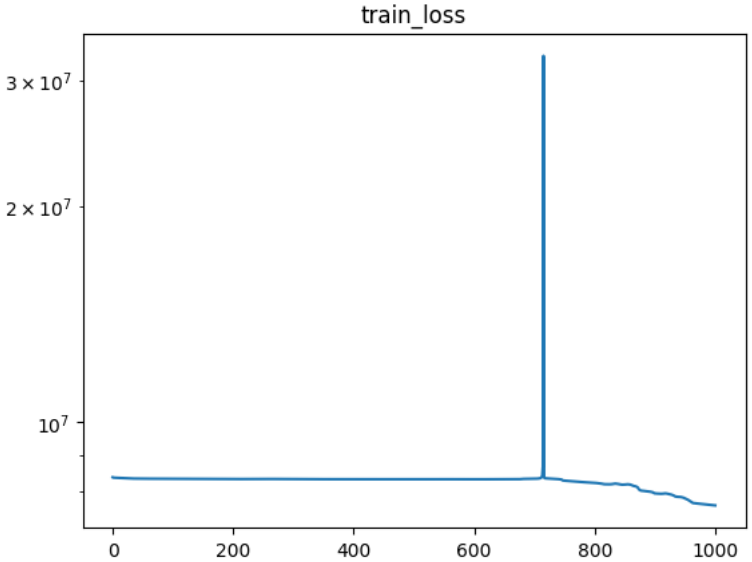




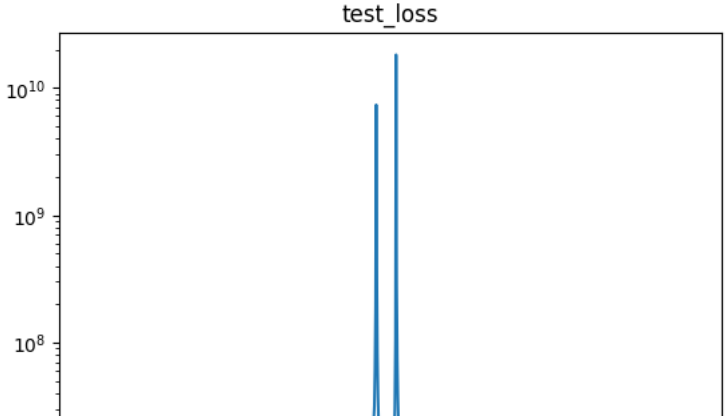
optimizer: Rprop , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-05 , minimum_RMSE: 2412.80 , epoch: 1000 , test_loss: 2414 , train_loss

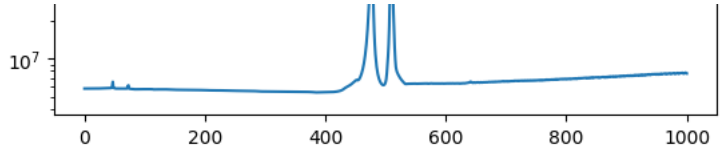


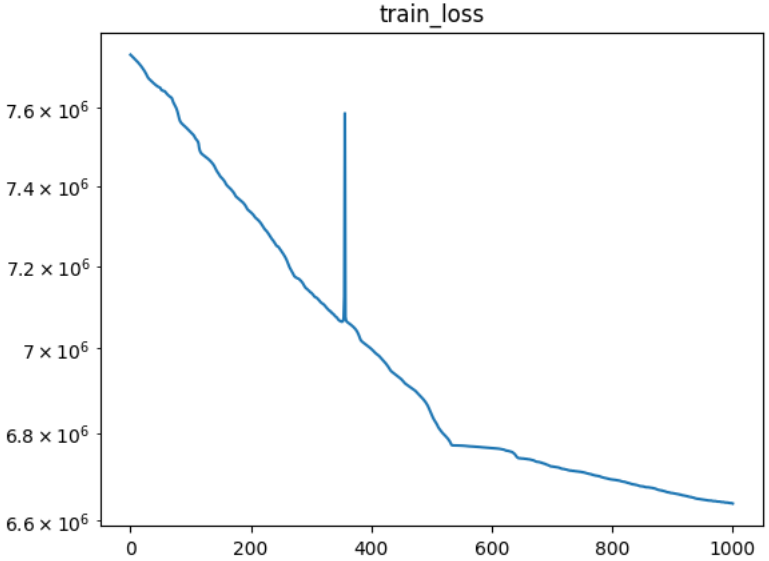




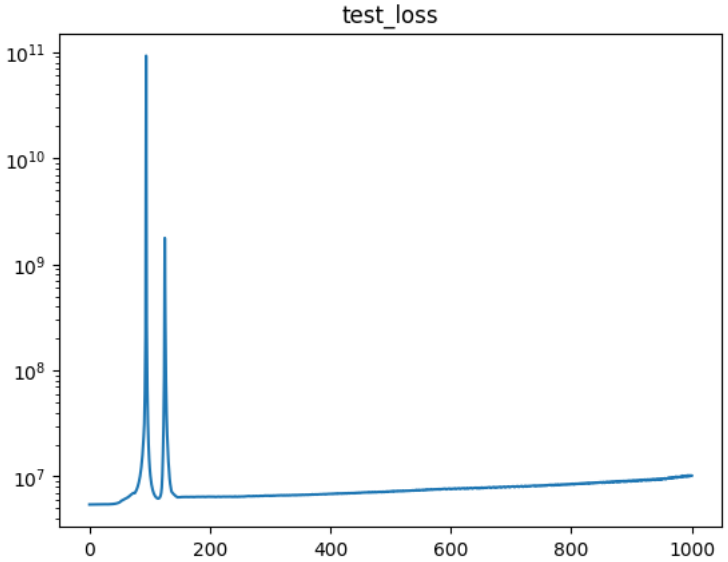
optimizer: Rprop , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-06 , minimum_RMSE: 2330.42 , epoch: 1000 , test_loss: 2764 , train_loss

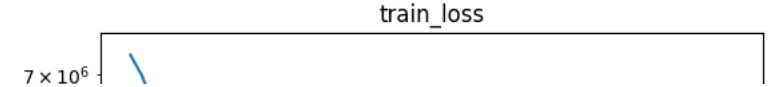


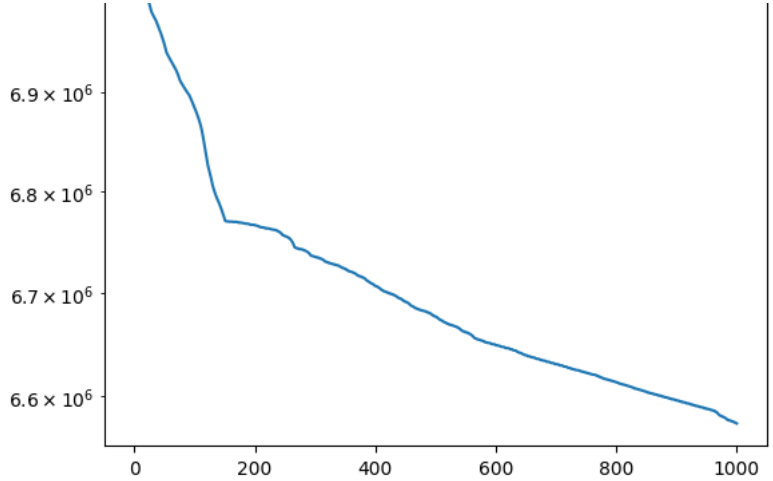




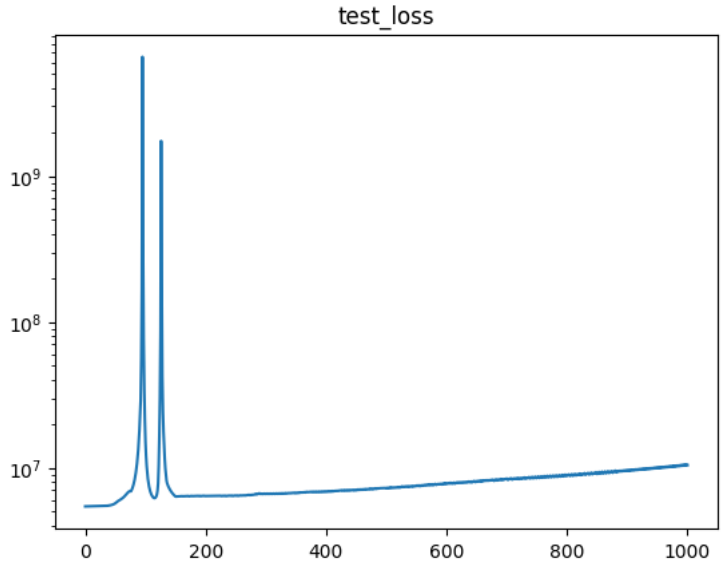
optimizer: Rprop , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-06 , minimum_RMSE: 2330.24 , epoch: 1000 , test_loss: 3187 , train_loss

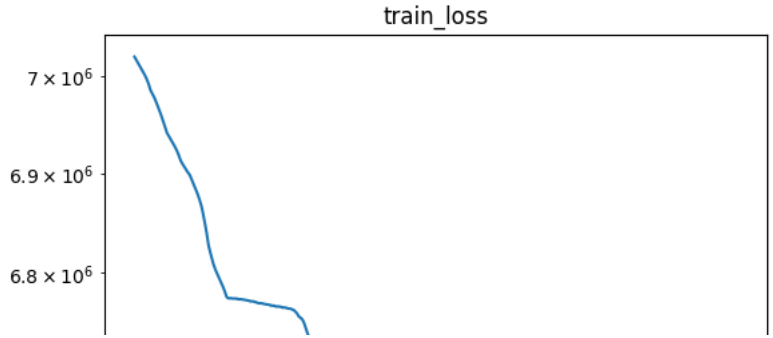


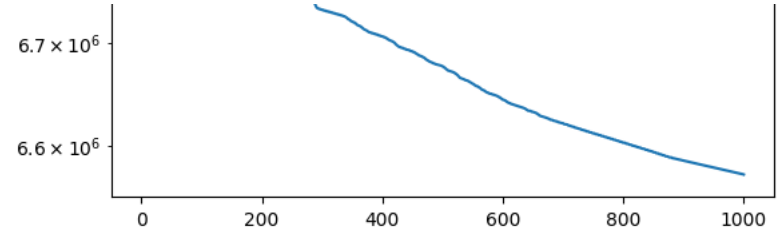




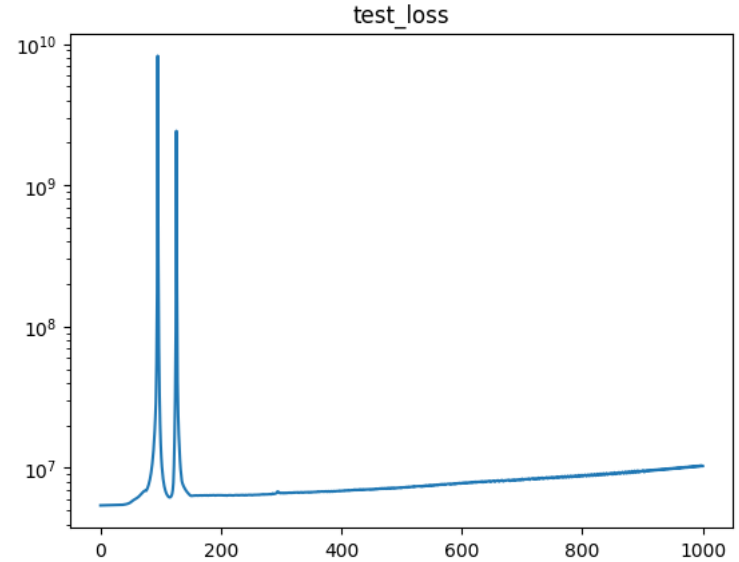
optimizer: Rprop , n_of_data: 1000 , Bsize: 100 , learningRate: 1e-07 , minimum_RMSE: 2330.26 , epoch: 1000 , test_loss: 3226 , train_loss

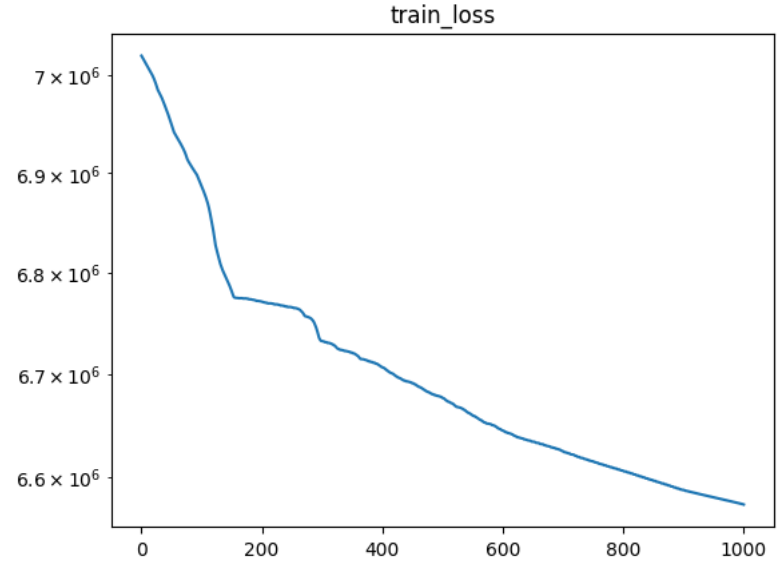






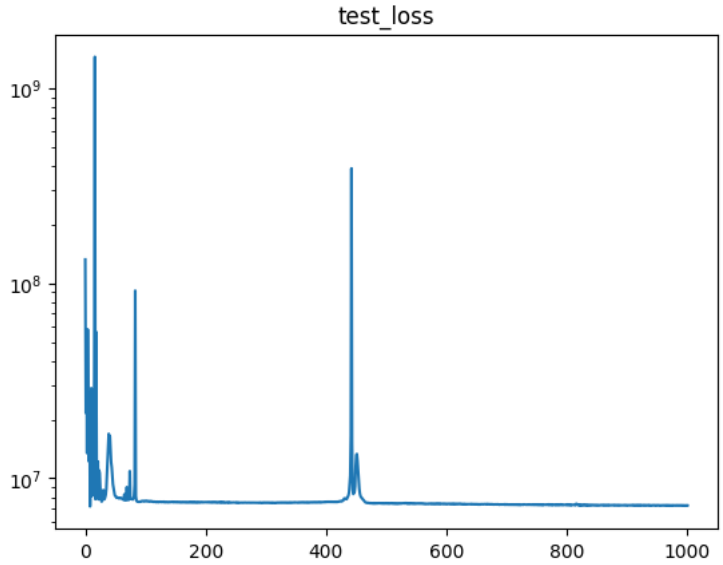
optimizer: Rprop , n_of_data: 1000 , Bsize: 100 , learningRate: 5e-07 , minimum_RMSE: 2330.28 , epoch: 1000 , test_loss: 3210 , train_loss

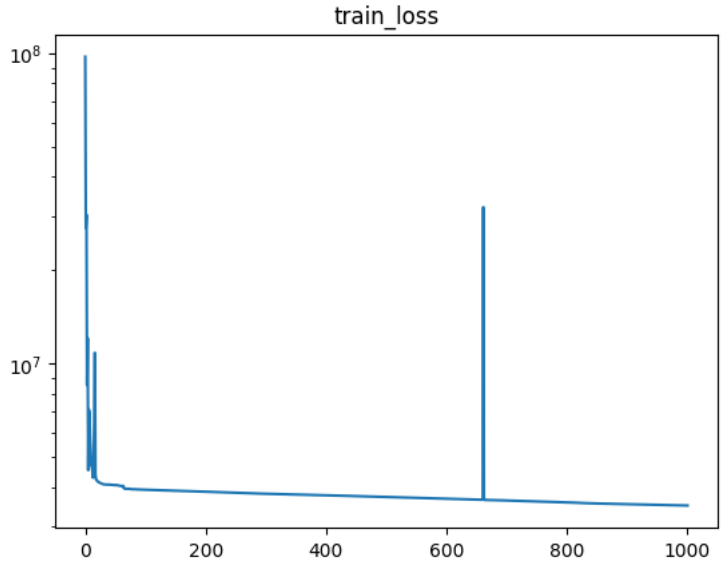




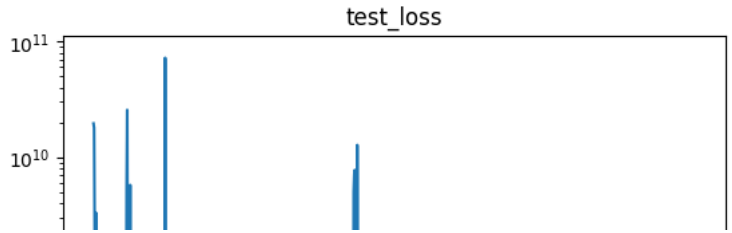
optimizer: Rprop , n_of_data: 1000 , Bsize: 500 , learningRate: 0.1 , minimum_RMSE: 2677.16 , epoch: 1000 , test_loss: 2692 , train_loss:

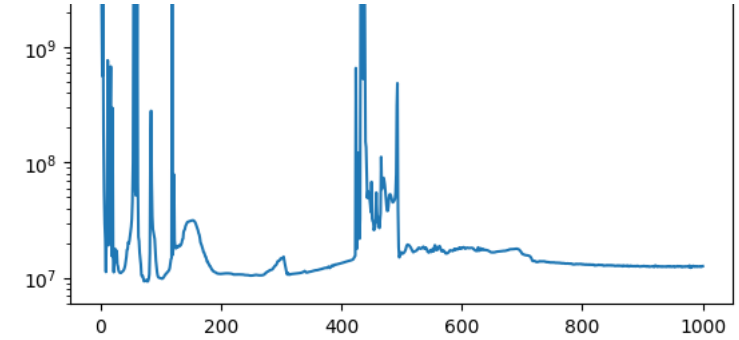
optimizer: Rprop , n_of_data: 1000 , Bsize: 500 , learningRate: 0.1 , minimum_RMSE: 2077.16 , epoch: 1000 , test_loss: 2092 , train_loss:

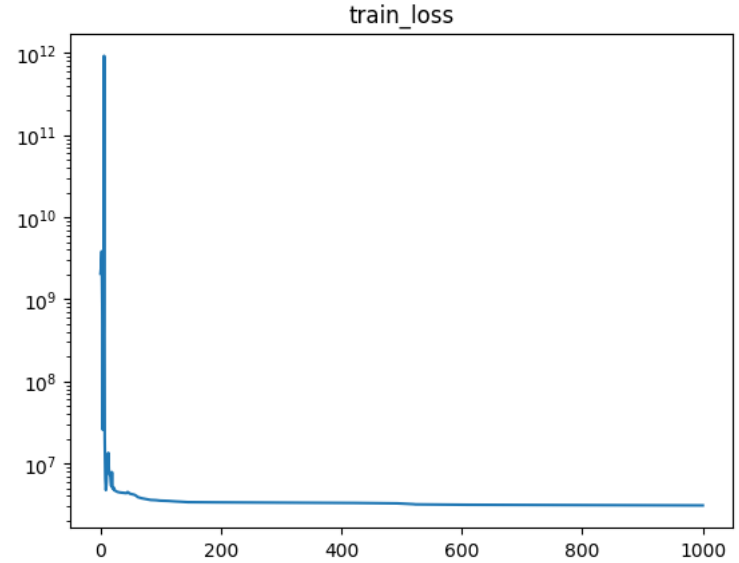




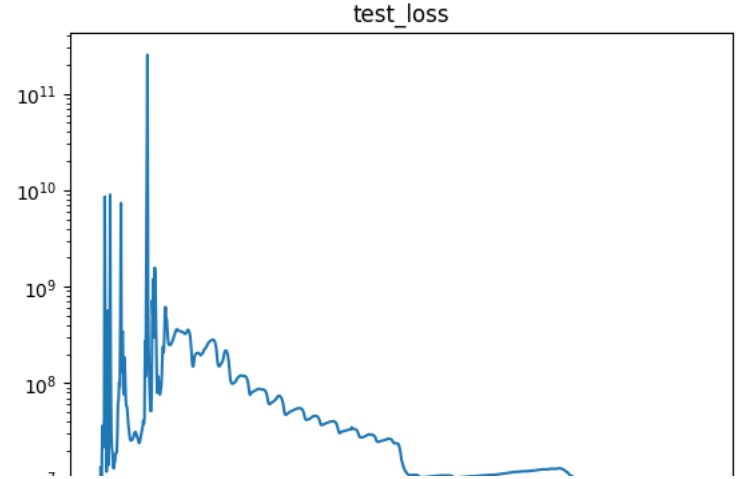
optimizer: Rprop , n_of_data: 1000 , Bsize: 500 , learningRate: 0.5 , minimum_RMSE: 3053.62 , epoch: 1000 , test_loss: 3557 , train_loss:

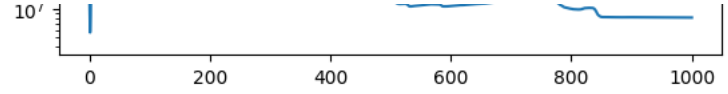


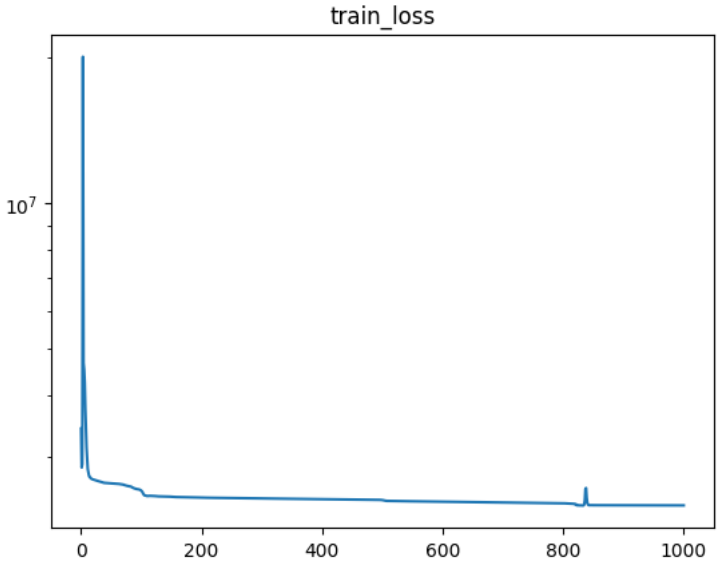




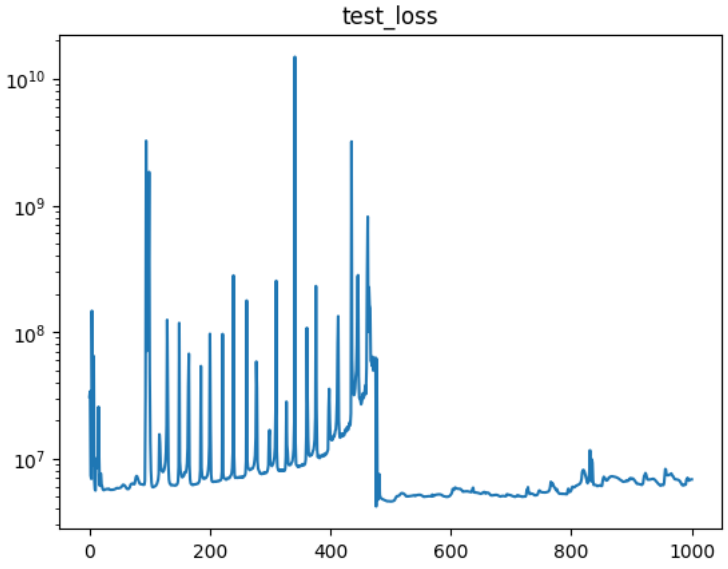
optimizer: Rprop , n_of_data: 1000 , Bsize: 500 , learningRate: 0.01 , minimum_RMSE: 2374.63 , epoch: 1000 , test_loss: 2833 , train_loss:

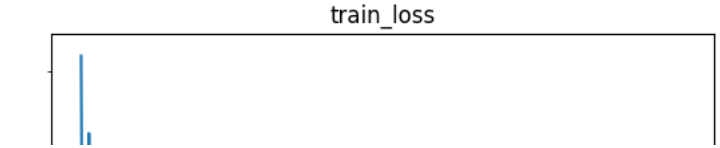


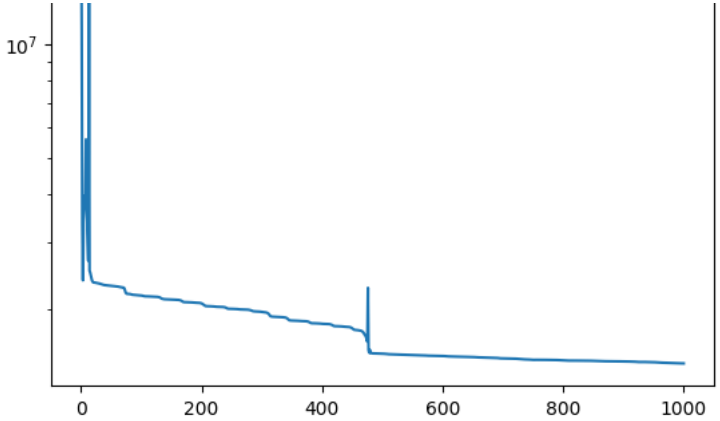




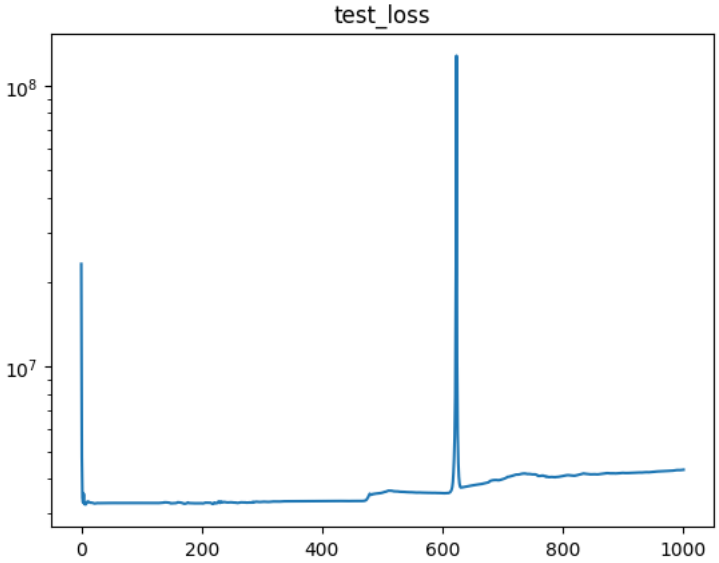
optimizer: Rprop , n_of_data: 1000 , Bsize: 500 , learningRate: 0.05 , minimum_RMSE: 2046.32 , epoch: 1000 , test_loss: 2615 , train_loss:

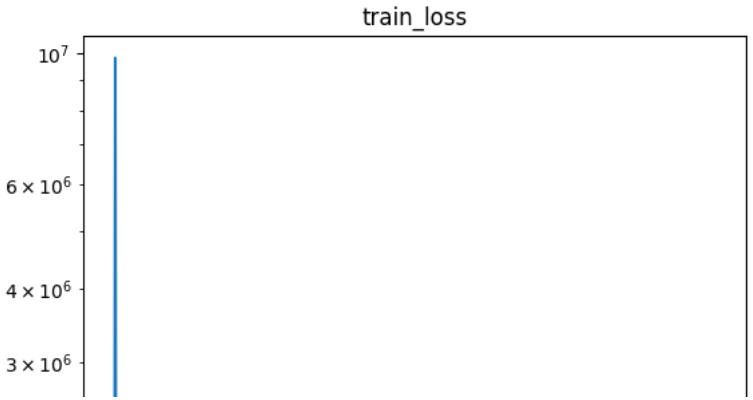


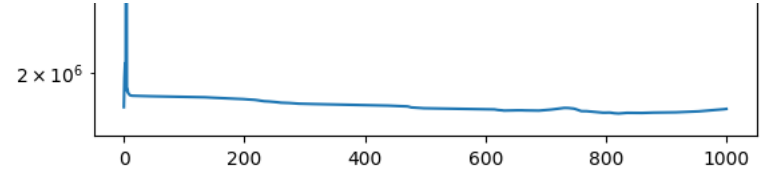




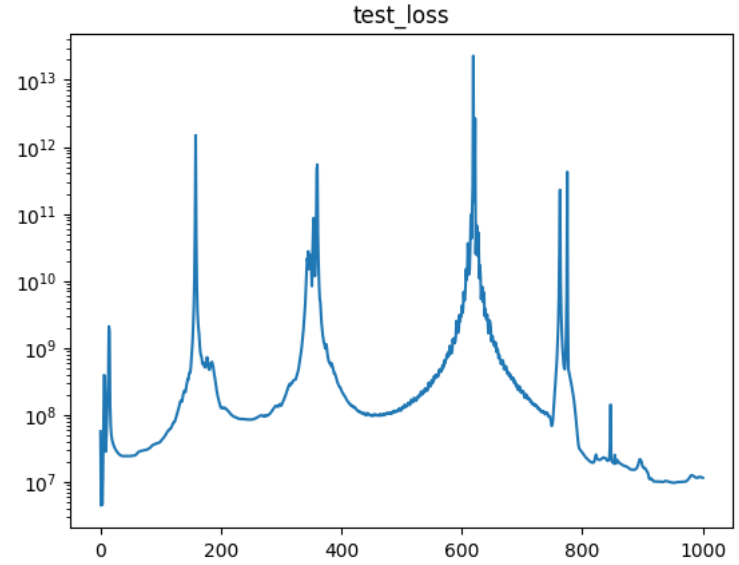
optimizer: Rprop , n_of_data: 1000 , Bsize: 500 , learningRate: 0.001 , minimum_RMSE: 1798.26 , epoch: 1000 , test_loss: 2073 , train_loss

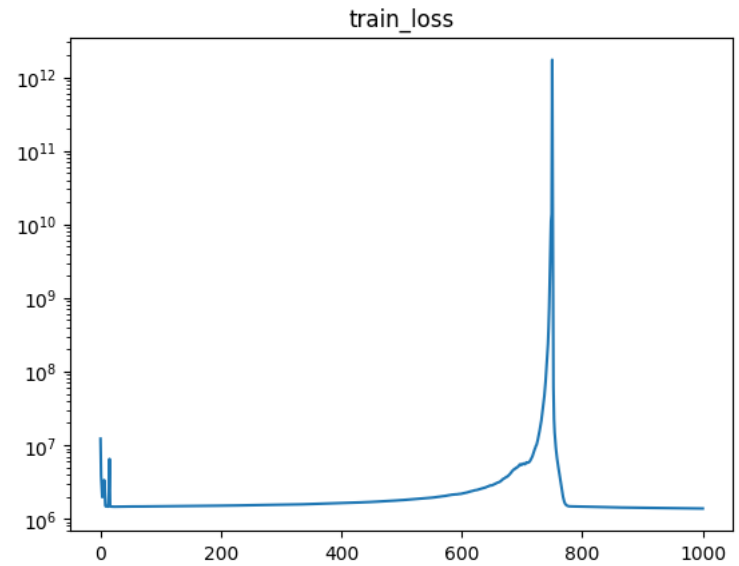




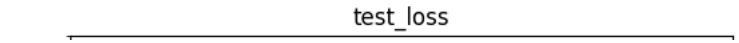


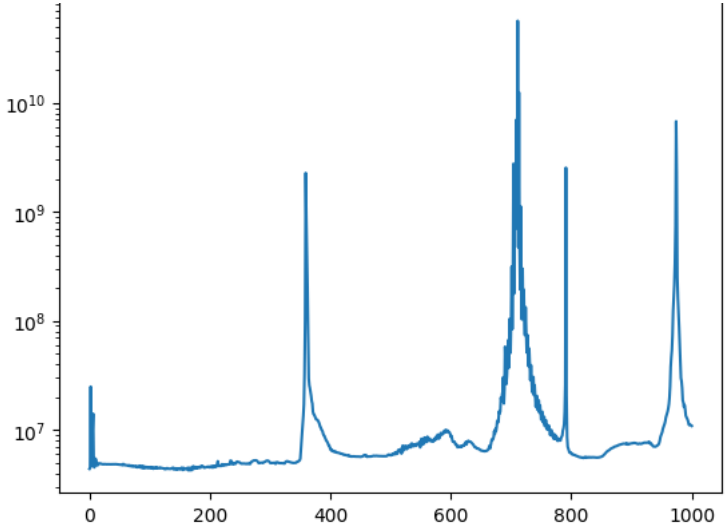
optimizer: Rprop , n_of_data: 1000 , Bsize: 500 , learningRate: 0.005 , minimum_RMSE: 2116.92 , epoch: 1000 , test_loss: 3400 , train_loss

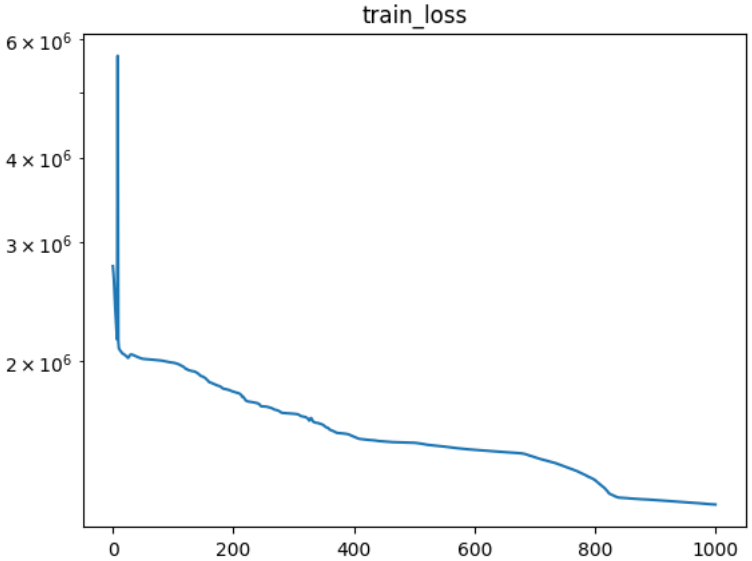




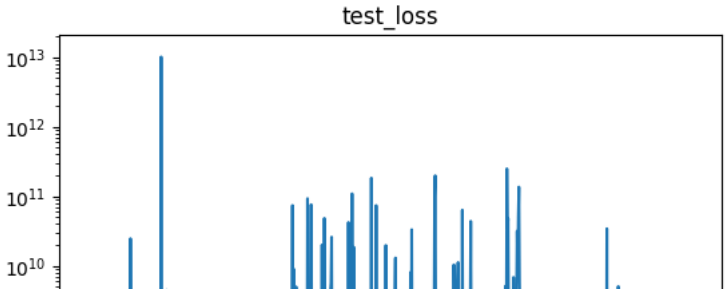
optimizer: Rprop , n_of_data: 1000 , Bsize: 500 , learningRate: 0.0001 , minimum_RMSE: 2062.76 , epoch: 1000 , test_loss: 3300 , train_los

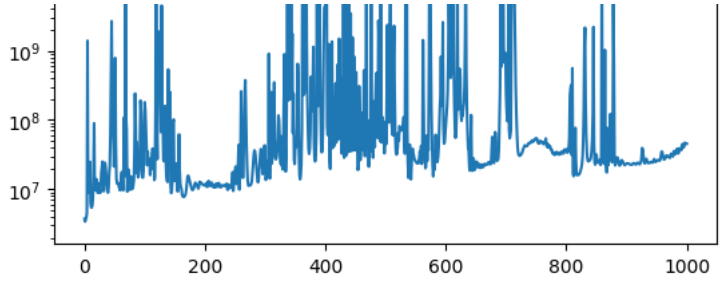


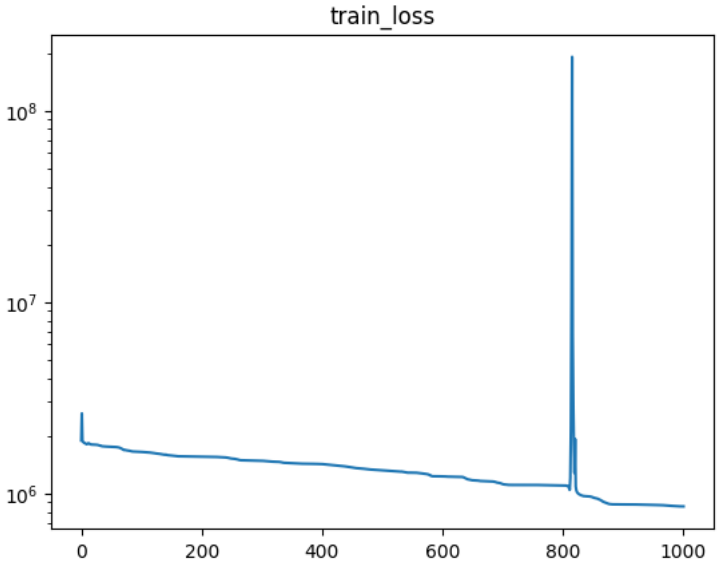




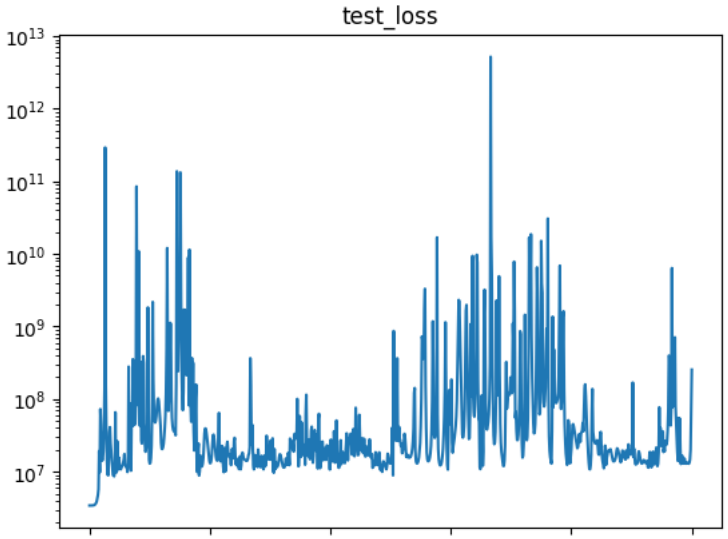
optimizer: Rprop , n_of_data: 1000 , Bsize: 500 , learningRate: 0.0005 , minimum_RMSE: 1851.63 , epoch: 1000 , test_loss: 6729 , train_los

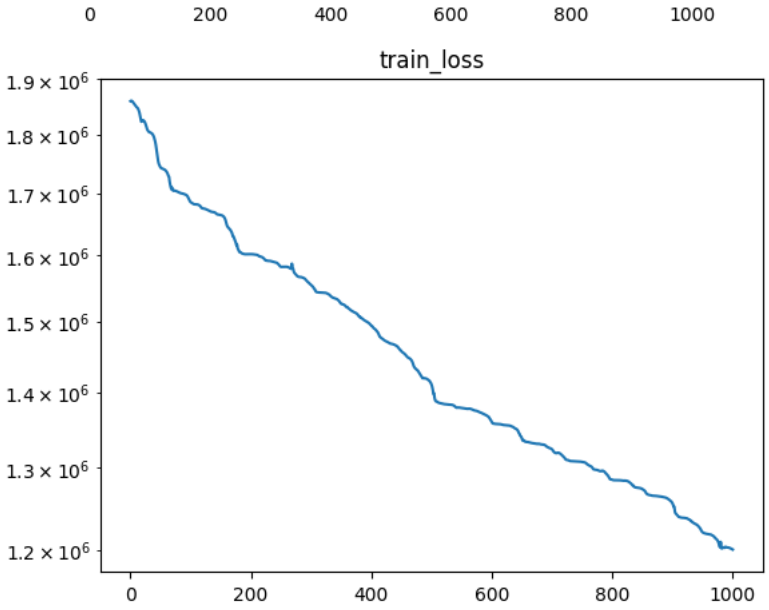




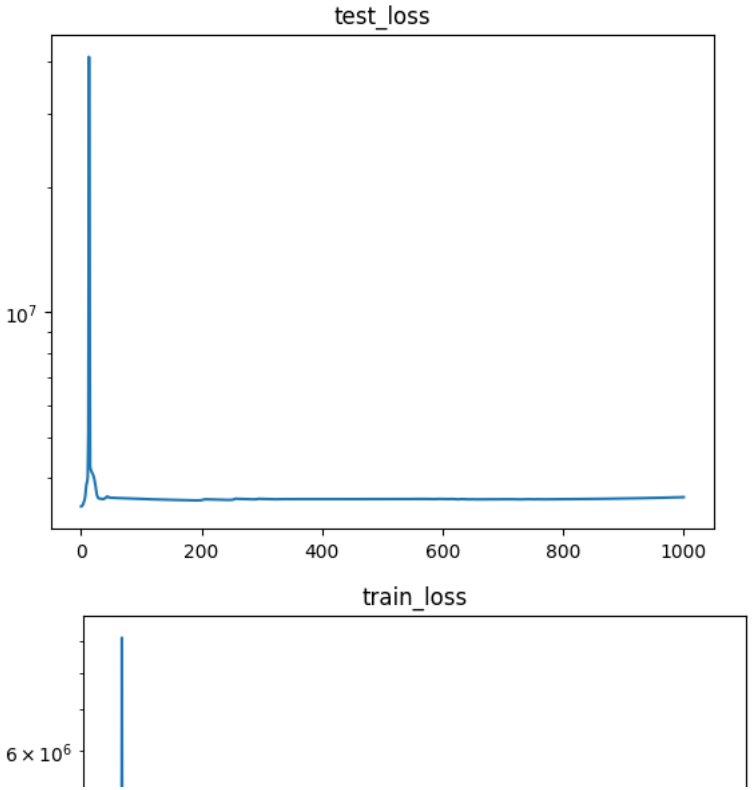


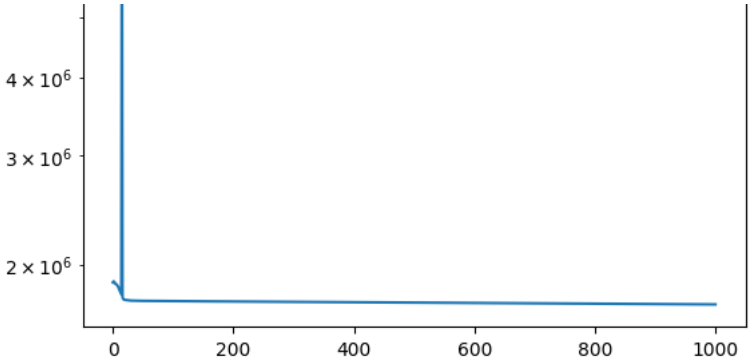
optimizer: Rprop , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-05 , minimum_RMSE: 1850.32 , epoch: 1000 , test_loss: 15961 , train_loss



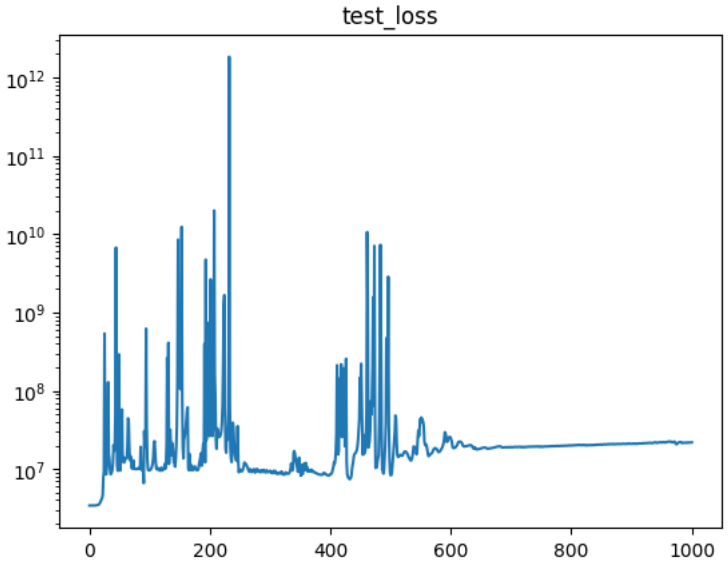


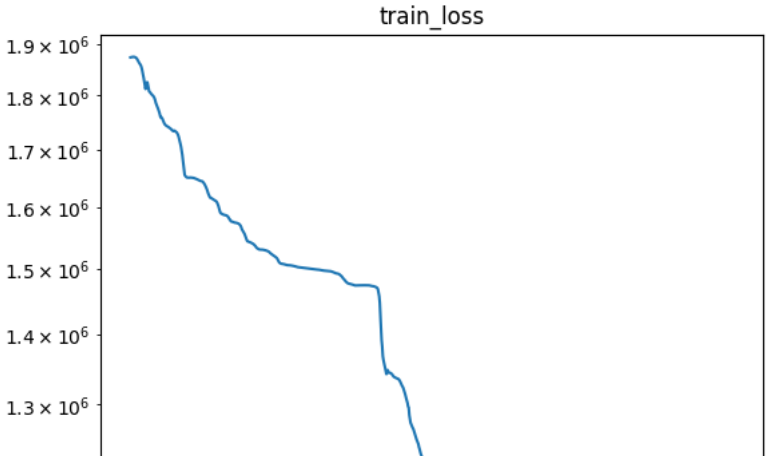
optimizer: Rprop , n_of_data: 1000 , Bsize: 500 , learningRate: 5e-05 , minimum_RMSE: 1849.33 , epoch: 1000 , test_loss: 1897 , train_loss

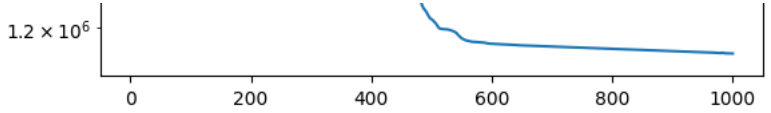




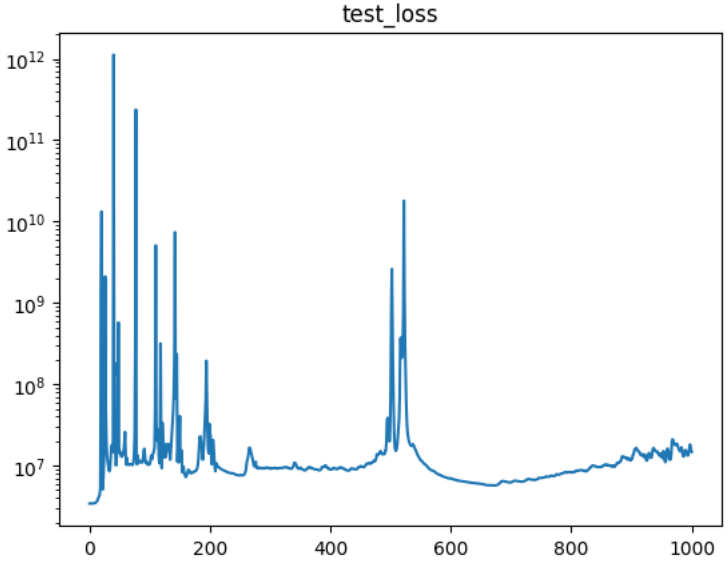
optimizer: Rprop , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-06 , minimum_RMSE: 1849.37 , epoch: 1000 , test_loss: 4682 , train_loss

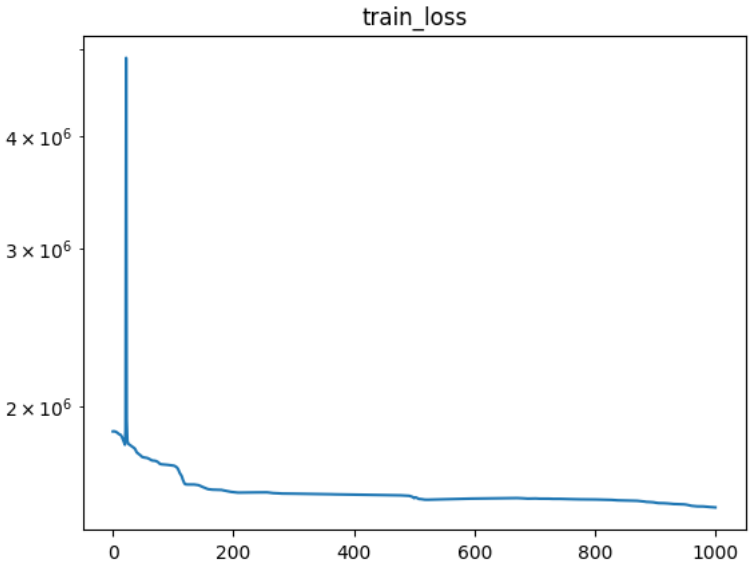




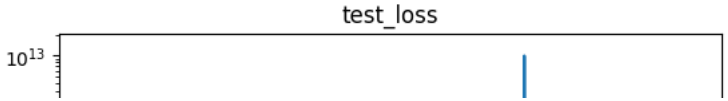


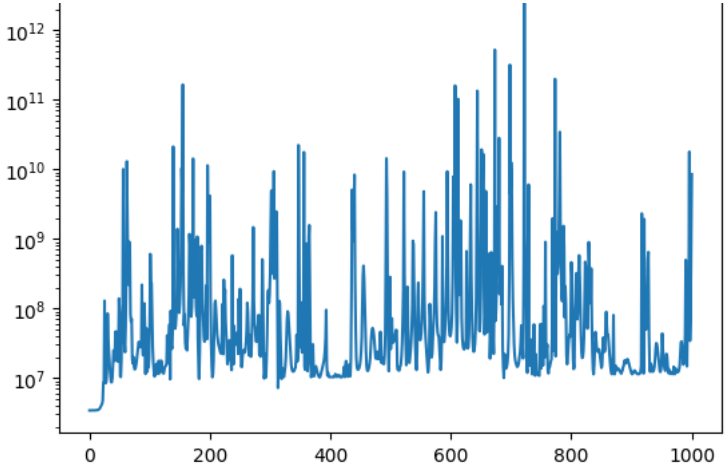
optimizer: Rprop , n_of_data: 1000 , Bsize: 500 , learningRate: 5e-06 , minimum_RMSE: 1849.55 , epoch: 1000 , test_loss: 3844 , train_loss

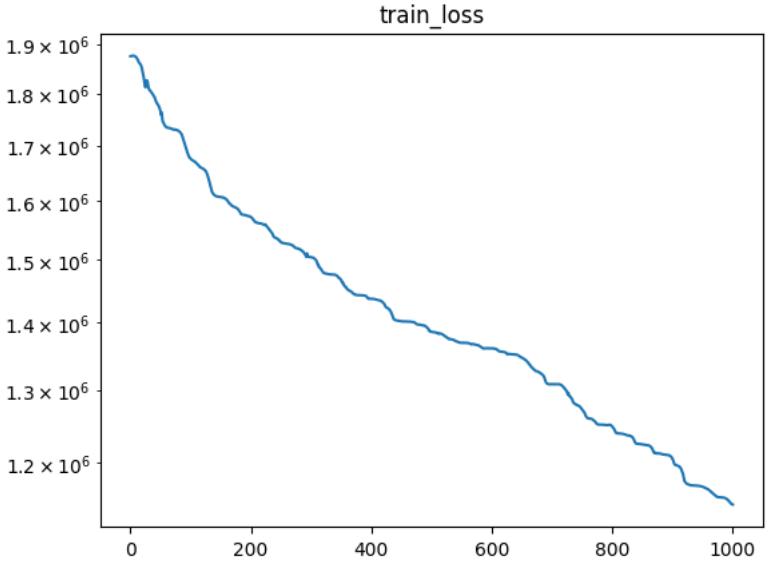




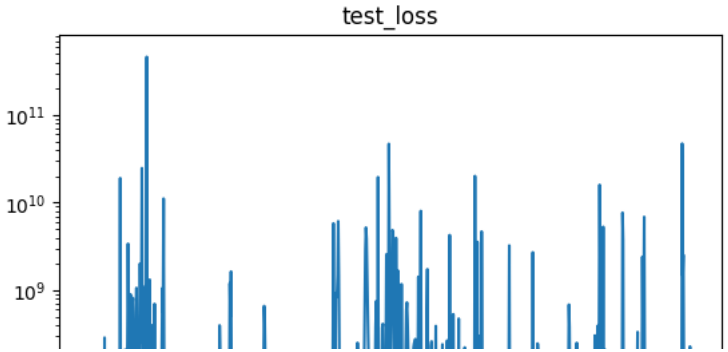
optimizer: Rprop , n_of_data: 1000 , Bsize: 500 , learningRate: 1e-07 , minimum_RMSE: 1849.60 , epoch: 1000 , test_loss: 92094 , train_loss

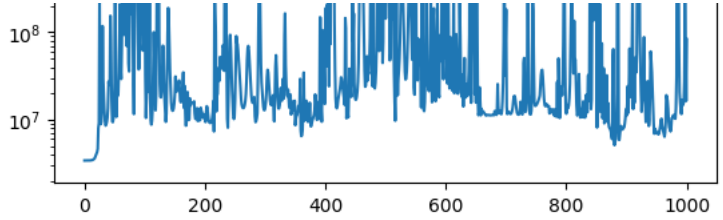


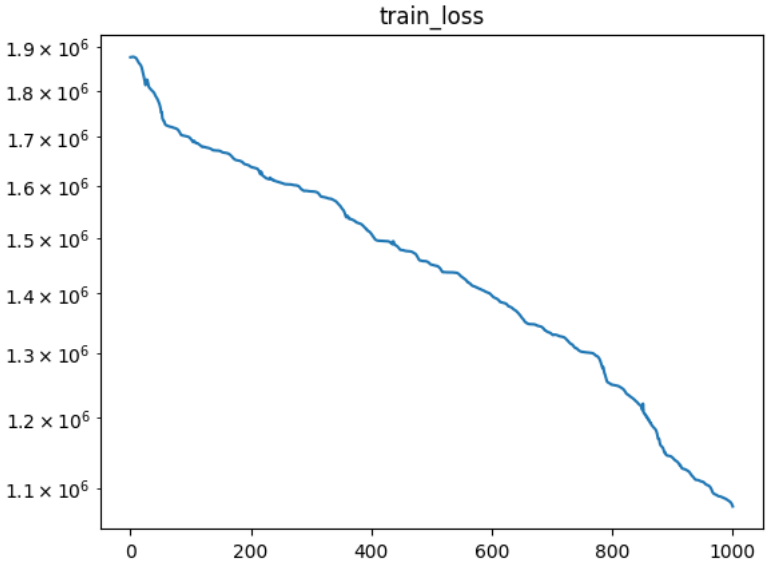




optimizer: Rprop , n_of_data: 1000 , Bsize: 500 , learningRate: 5e-07 , minimum_RMSE: 1849.64 , epoch: 1000 , test_loss: 9122 , train_loss



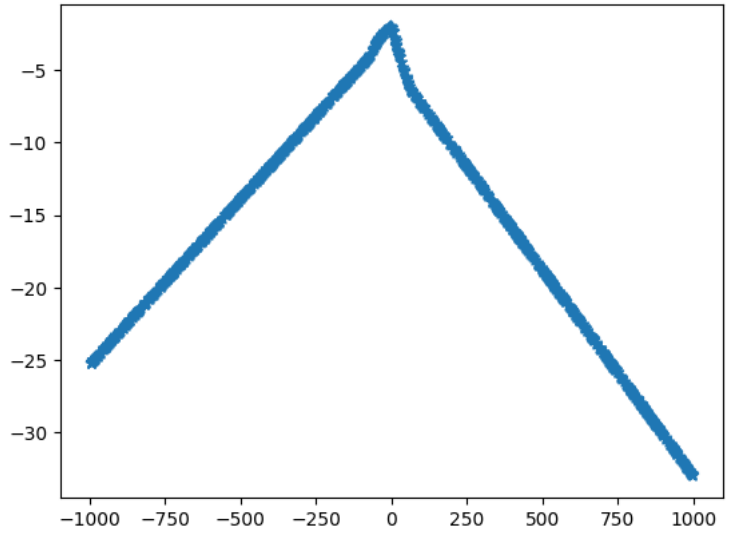


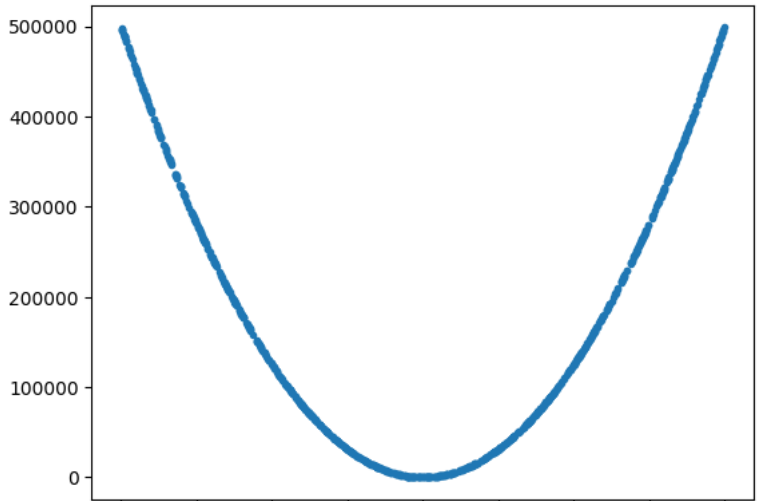



```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-54-7024aaa558bb> in <cell line: 41>()
    138
    139             # restore model and return best accuracy
```

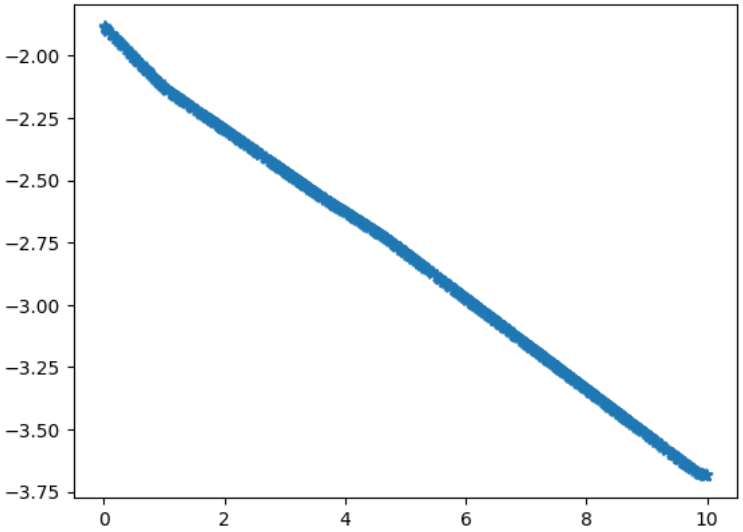
```
1
```

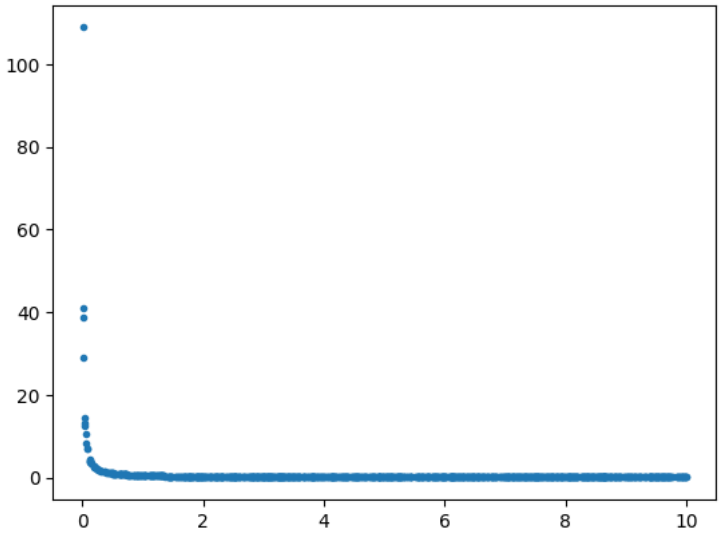
```
1 model.eval()
2 y_pred = model(X2)
3 plt.scatter(X2[:,1].cpu().detach().numpy(),y_pred.cpu().detach().numpy(), marker='*')
4 # plt.scatter(X2[:,1],y_pred, marker='*')
5 plt.show()
6
7
8 plt.scatter(X2[:,1] ,y2, marker='.', )
9 plt.show()
10
11
12 print('all done')
```





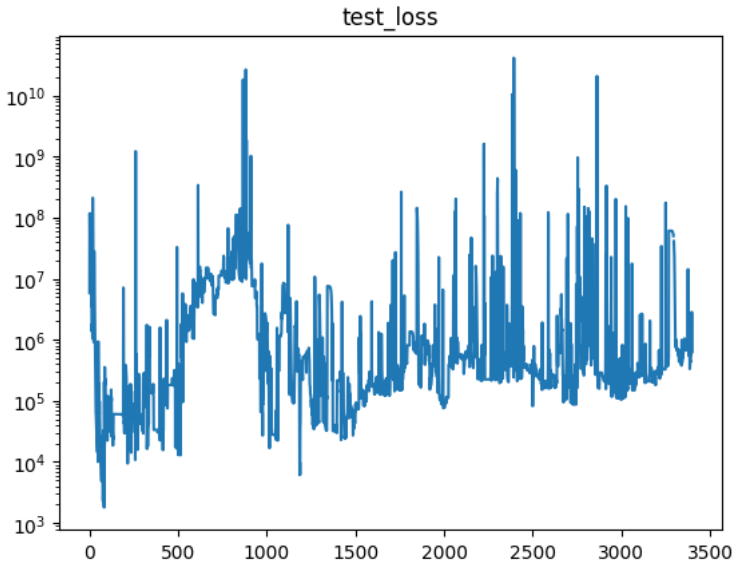

```
model.eval()
y_pred = model(X3)
plt.scatter(X3[:,0].cpu().detach().numpy(),y_pred.cpu().detach().numpy(), marker='*')
# plt.scatter(X3[:,1],y_pred, marker='*')
plt.show()


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


print('all done')
```

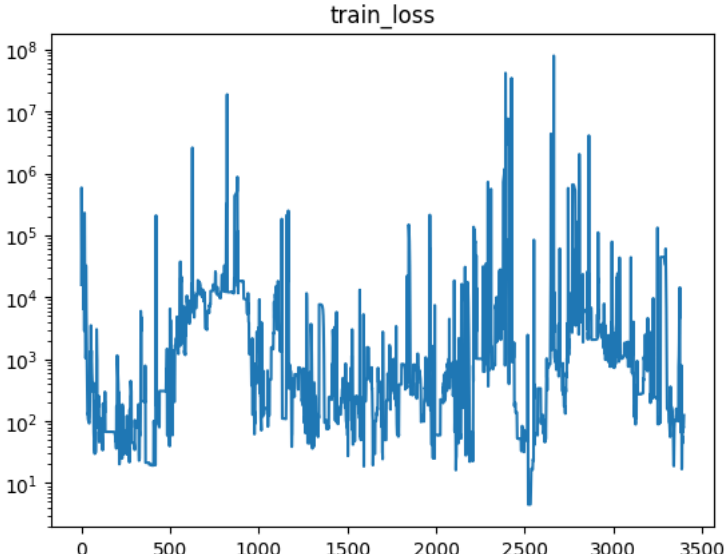





```
all done
```

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

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






```
1
```

```
1 # when create new data
2
3 # N, D_in, H, D_out = 64, 1000, 100, 10
4 N, D_in, D_out = 100000, 2, 1
5
6 # 입력과 출력 위한 랜덤 텐서
7 X = []
```

```
y = []
for j in range(N):
    X.append([])
    #y.append([])
    for i in range(D_out):
        X[-1].append( np.random.uniform(low=0.0, high=1000.0, size=None) )
        X[-1].append( np.random.uniform(low=-1000.0, high=1000.0, size=None) )
        y.append( (X[-1][-1]**2) / (2*X[-1][-2])  )


X = torch.Tensor(X)
y = torch.Tensor(y)

# nn package를 이용하여 여러 층으로 정의된 모델 생성
# nn.Sequential은 다른 모듈을 담을 수 있는 모듈이며 담겨진 모듈은 순서대로 연결
# Linear 모듈은 곧 Affine 모듈

# Read data
# data = fetch_california_housing()
# X, y = data.data, data.target

# train-test split for model evaluation

import time
# >>> # Save to file
# >>> x = torch.tensor([0, 1, 2, 3, 4])
# >>> torch.save(x, 'tensor.pt')

torch.save(X, 'KEdataX_'+str(time.time())+'.pt' )
torch.save(y, 'KEdataY_'+str(time.time())+'.pt' )


```

✓ 0초 오후 11:29에 완료됨