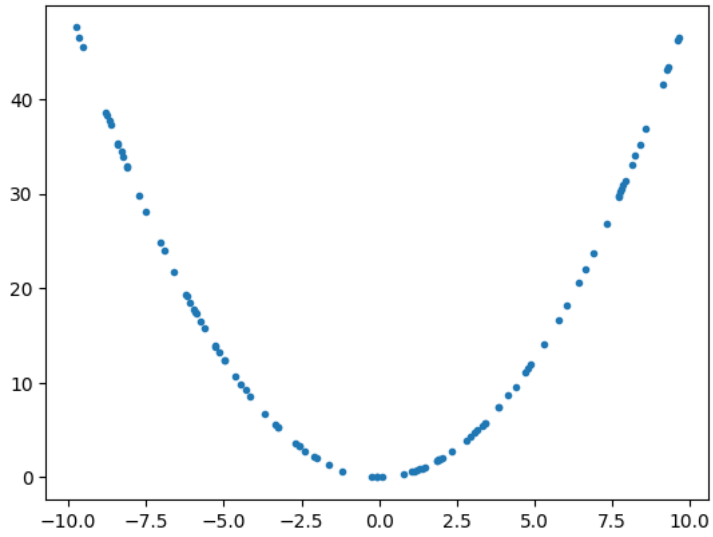
```
from google.colab import drive
drive.mount('/content/drive')
```

```
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 time
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 = 100, 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=-10.0, high=10.0, 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=0.1, 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)
```

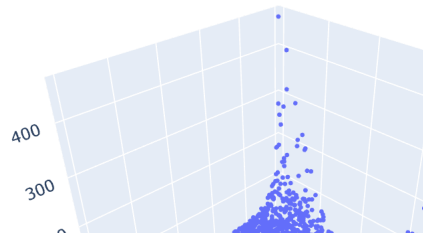
```
plt.scatter(X2[:,1] ,y2, marker='.', )
plt.show()
plt.scatter(X3[:,0] ,y3, marker='.', )
plt.show()

```




```
1
2 # X = torch.load('/content/drive/MyDrive/0_318lab/SCMP_ML/KEdataX_1000_1691036428.pt')
3 # y = torch.load('/content/drive/MyDrive/0_318lab/SCMP_ML/KEdataY_1000_1691036428.pt')
4
5 X = torch.load('/content/drive/MyDrive/0_318lab/SCMP_ML/KEdataX_N_10000_Interval_10_1691050560.pt')
6 y = torch.load('/content/drive/MyDrive/0_318lab/SCMP_ML/KEdataY_N_10000_Interval_10_1691050560.pt')
7
8 X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, shuffle=False)
9 # X2_train, X2_test, y2_train, y2_test = train_test_split(X, y, train_size=0.7, shuffle=True)
10 # X3_train, X3_test, y3_train, y3_test = train_test_split(X, y, train_size=0.7, shuffle=True)
11
12 # Convert to 2D PyTorch tensors
13 #X_train = torch.tensor(X_train, dtype=torch.float32)
14 #y_train = torch.tensor(y_train, dtype=torch.float32).reshape(-1, 1)
15 #X_test = torch.tensor(X_test, dtype=torch.float32)
16 #y_test = torch.tensor(y_test, dtype=torch.float32).reshape(-1, 1)
17
18
19
```

```
1 # X_train, X_test, y_train, y_test = train_test_split(X2, y2, train_size=0.7, shuffle=False)
2
```

```
1 import plotly.express as px
2 # df = px.data.iris()
3 # fig = px.scatter_3d(x=X[:,0], y=X[:,1], z=y, size=torch.ones(len(y_pred))*0.001)
4 fig = px.scatter_3d(x=X_train[:,0], y=X_train[:,1], z=y_train) #size=torch.ones(len(y_pred))*0.001)
5 fig.update_traces(marker_size=1)
6 fig.show()
```

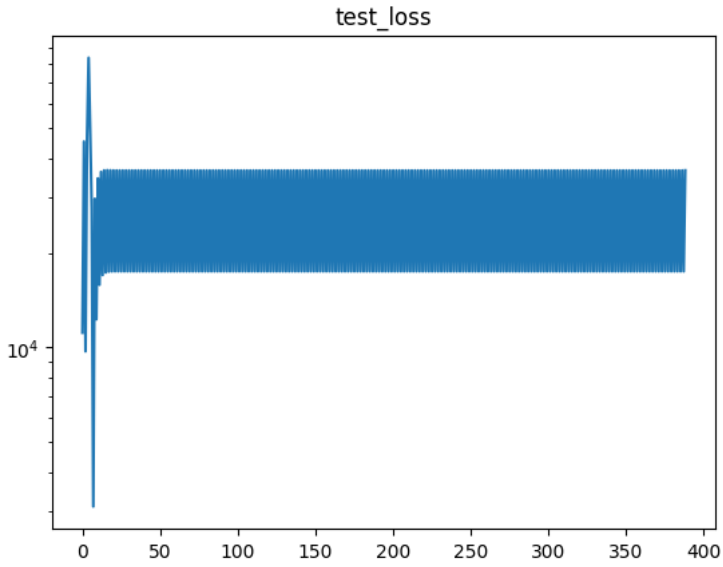


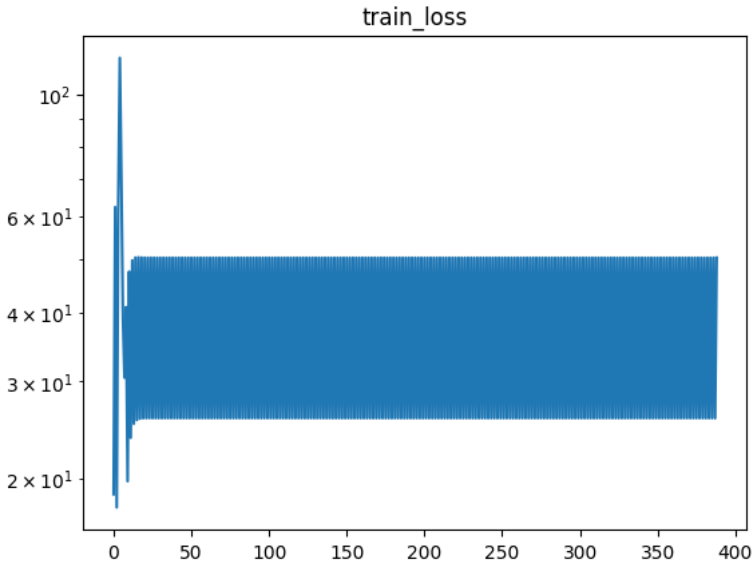

```
#default

"""
try
torch.nn.LogSigmoid(*args, **kwargs)
torch.nn.Softplus(beta=1, threshold=20)
"""
# for learningRate in range(10): #


dim_list = [2, 20, 20, 20, 20, 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.

init_time = time.time()
print('start')
for act in [
            # 'LeakyReLU',
            # 'LogSigmoid',
            'Softplus']:

    for opt in [
                # 'Adadelta',
                # 'Adagrad',
                'Adam',
                # 'AdamW',
                # 'SparseAdam',
                # 'Adamax',
                # 'ASGD',
                # 'LBFGS',
                # 'NAdam',
                # 'RAdam',
                # 'RMSprop',
                # 'Rprop',
                # 'SGD'
                ]:
    # try:
        # if opt == 'Adam':# --------------delete later----------------------------------------------------------------------------------------------------------------
        #     continue# --------------delete later----------------------------------------------------------------------------------------------------------------
        for Bsize in [ 10, 50, 100, 500]:
        # for Bsize in [ 10, 100, 1000]:
        # for Bsize in [ 1000]:
            # print('Bsize:', Bsize, end =' ')
            #before_loss_twos =  torch.ones(Bsize, dtype=torch.float32)*2
            # for learningRate in [ 1e-1, 1e-2, 1e-3, 1e-4,1e-5, 1e-6, ]:
            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, ]:
            # for learningRate in [   1e-4, ]:
                # 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.LeakyReL
                        make_model = make_model + 'nn.Linear(dim_list[' + str(layer_num) + '], dim_list[' + str(layer_num+1) + ']), nn.' + act +


                exec(make_model)
                #make_model = make_model +

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

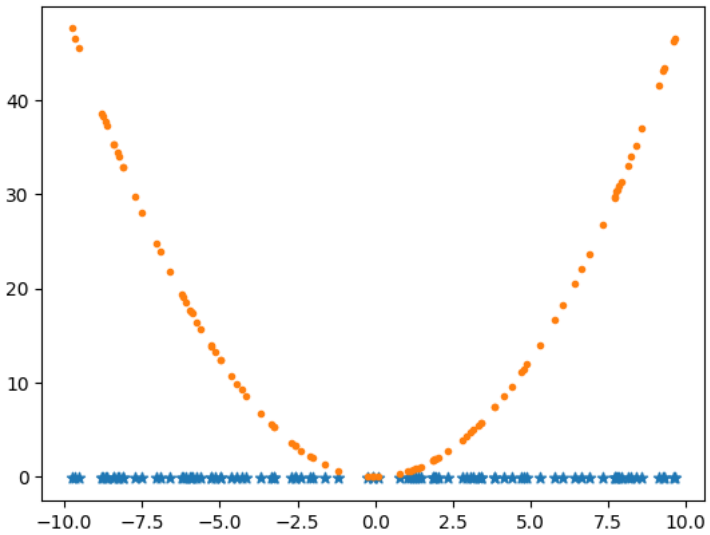
```
#     nn.LeakyReLU(),
#     nn.Linear(dim_list[1], dim_list[2]),
#     nn.LeakyReLU(),
#     nn.Linear(dim_list[2], dim_list[3])
#     # nn.Linear(2, 3),
#     # nn.LeakyReLU(),
#     # nn.Linear(3, 2),
#     # nn.LeakyReLU(),
#     # nn.Linear(2, 1)
# )

# 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 = 20000+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):
second_time = time.time()

epoch = 0
while(1):

    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).squeeze()

            before_loss = torch.mean( (y_pred/y_batch-1)**2 + (y_batch/y_pred-1)**2 + (y_pred-y_batch)**2 )
            # before_loss_twos =  torch.ones(len(y_pred), dtype=torch.float32)*2


            loss = before_loss**(1/2)
            # loss = loss_fn(y_pred, y_batch)
            # loss = loss_fn(before_loss, before_loss_twos + y_batch)

            # backward pass
            optimizer.zero_grad()
            loss.backward()

            # update weights
            optimizer.step()
            # print progress
            bar.set_postfix(mse=float(loss))
    #print(y_pred)
    loss = float(loss)

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

    before_loss = torch.mean(  (y_pred/y_test-1)**2 + (y_test/y_pred-1)**2 + (y_pred-y_test)**2 )
    # before_loss = y_pred/y_test + y_test/y_pred + y_pred
    # before_loss_twos =  torch.ones(len(y_pred), dtype=torch.float32)*2
    # mse = loss_fn(y_pred, y_test)
    # mse = loss_fn(before_loss, before_loss_twos + y_test)
    mse = before_loss**(1/2)

    mse = float(mse)

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

    epoch_time = ( time.time()-second_time )

```

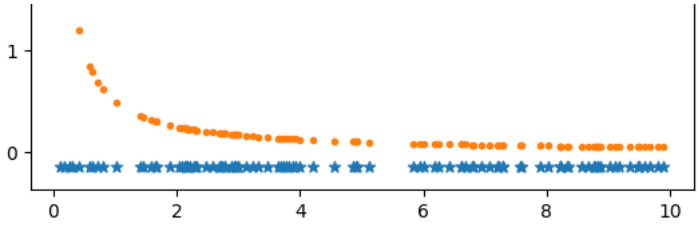
```
                    #init_time = time.time()
                    # if epoch % 100 == 0:
                    #     # print('epoch: %5d' % epoch, 'test_loss: %.2f' % (mse), 'train_loss: %.2f' % (loss), 'est_time: %.2f' % (( ( epoch_tim
                    #     print('epoch: %5d' % epoch, ', test_loss: %.2f' % (mse), ', train_loss: %.2f' % (loss), ', est_time: %.3f' % ( ( epoch_
                    if (mse<20) or (str(mse)=='nan') or ( ( epoch_time )  / 60 > 5  ):
                        break
                    epoch = epoch + 1

                # 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(
                      'activateion:', act,
                      ', optimizer:', opt,
                      ', n_of_data:',N,
                      ', Bsize:', Bsize ,
                      ', learningRate:', learningRate ,
                      ", minimum_RMSE: %.2f" % (best_mse),
                      ', epoch: %5d' % epoch,
                      ', test_loss: %2f' % (mse),
                      ', train_loss: %2f' % (loss),
                      ', layer_dim_list:', dim_list,
                      ', passed_time: %.3f' % ( ( epoch_time )  / 60  ),  'm',
                      ', passed_time_accum: %.3f' % ( ( init_time - time.time() )  / 60  ),  'm'
                      )
                # 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()

                model.eval()
                y_pred = model(X2)
                plt.scatter(X2[:,1].cpu().detach().numpy(),y_pred.cpu().detach().numpy(), marker='*')
                plt.scatter(X2[:,1] ,y2, marker='.', )
                plt.show()

                model.eval()
                y_pred = model(X3)
                plt.scatter(X3[:,0].cpu().detach().numpy(),y_pred.cpu().detach().numpy(), marker='*')
                plt.scatter(X3[:,0] ,y3, marker='.', )
                plt.show()

                print('finish----------------------------------------------------------------------------------------------------------------------------------------------------------')
                print('')

    # except:
    #     print('error when:', opt, 'error when:', opt, 'error when:', opt, 'error when:', opt, 'error when:', opt, 'error when:', opt, 'error wh
print('all done')
# ----------------------------------------------------------------------------------------------------
# ----------------------------------------------------------------------------------------------------
```

```
start
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 10 , learningRate: 0.1 , minimum_RMSE: 3115.68 , epoch:  388 , test_loss: 367
```



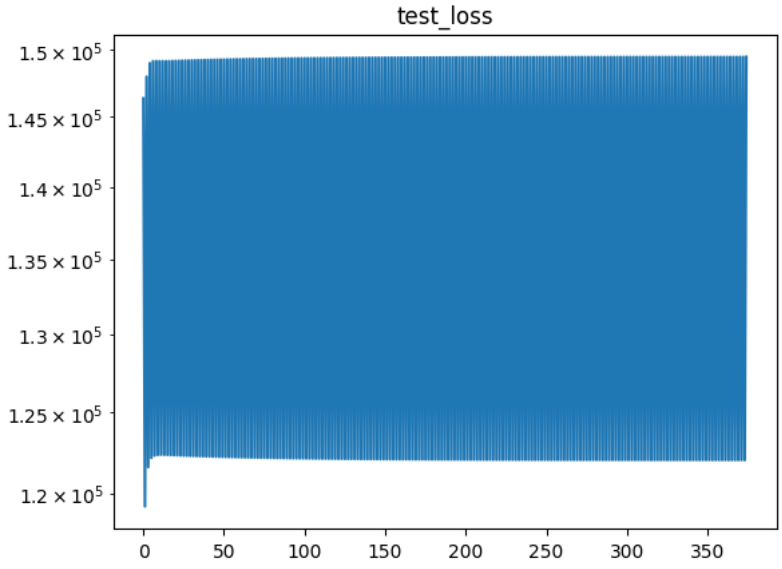


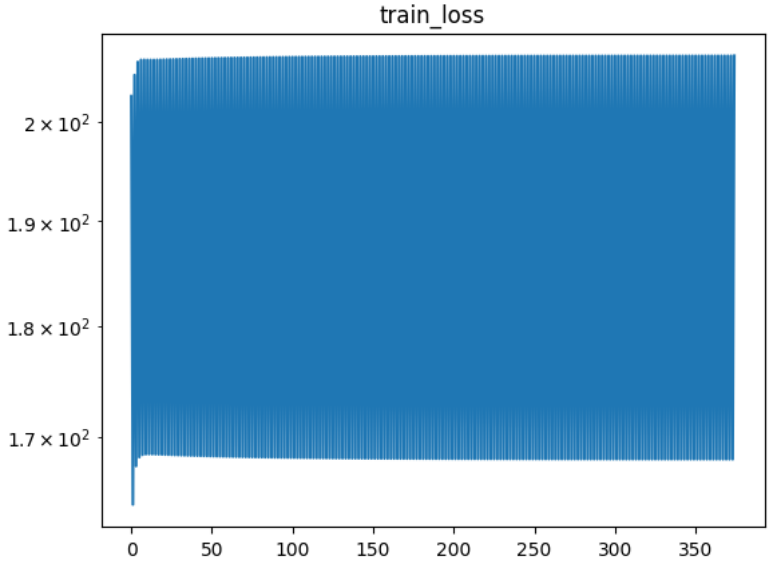


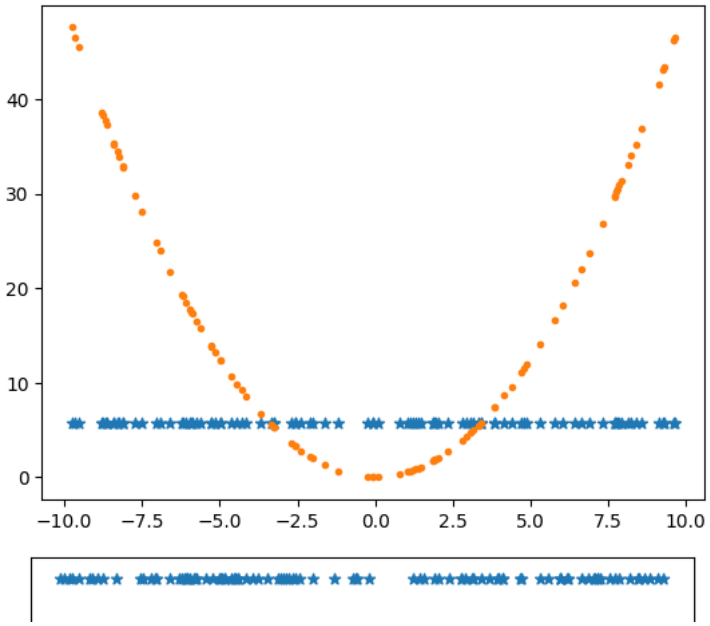


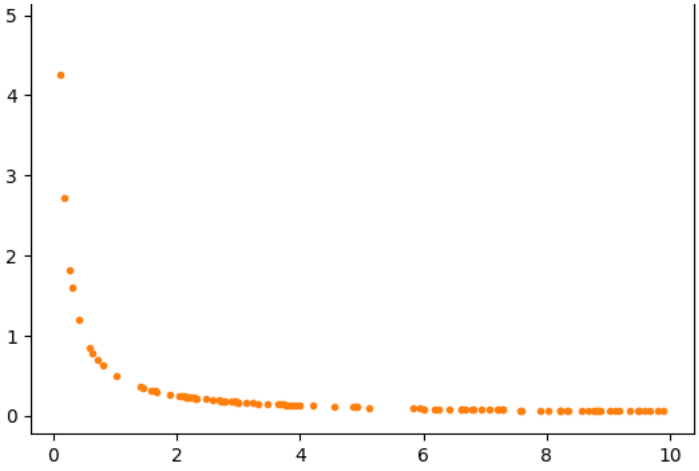
finish----------

activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 10 , learningRate: 0.5 , minimum_RMSE: 119228.55 , epoch: 374 , test_loss: 1



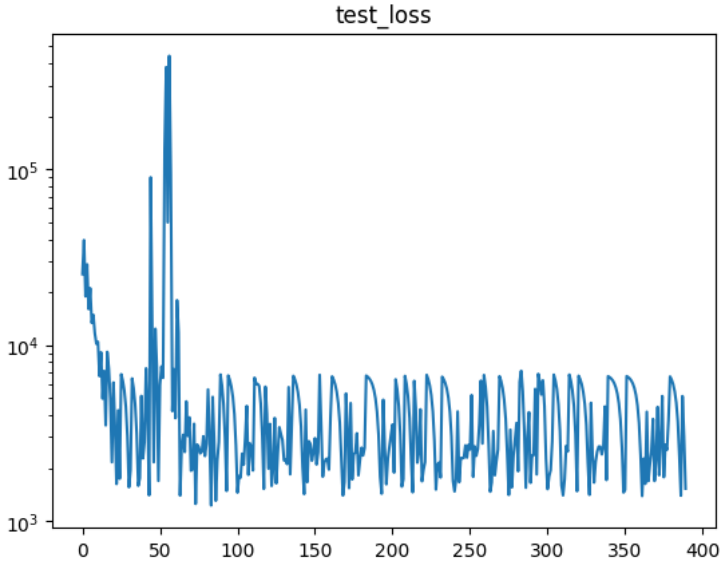


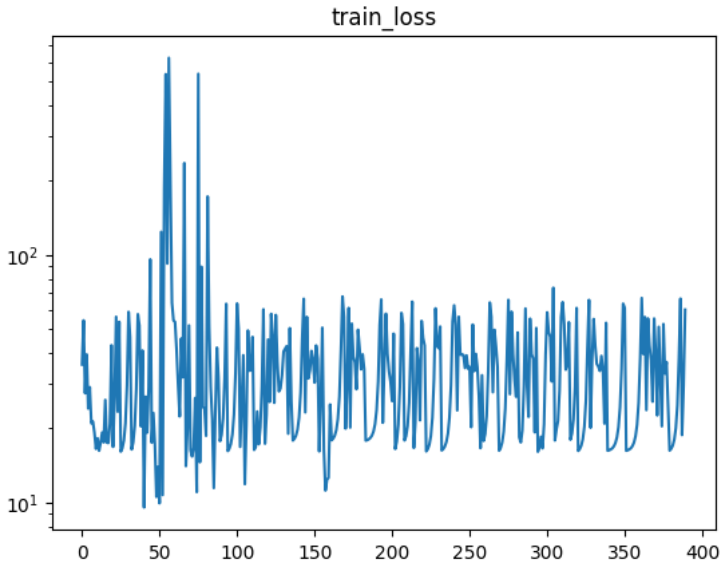


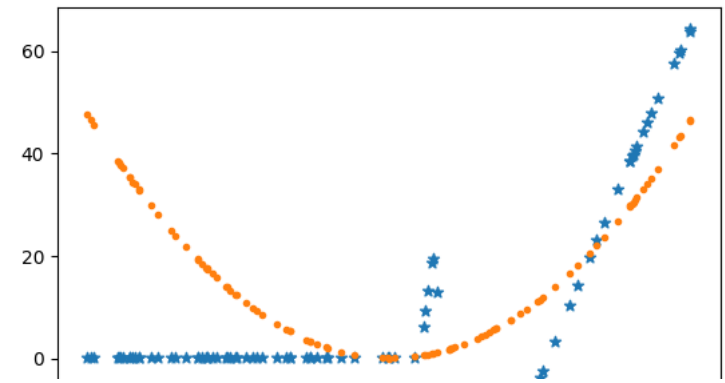


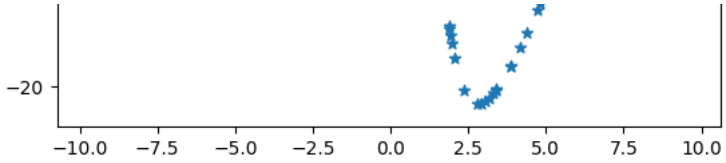
```
finish----------------------------------------------------------------------------------------------------------------------------------------------------------------
```

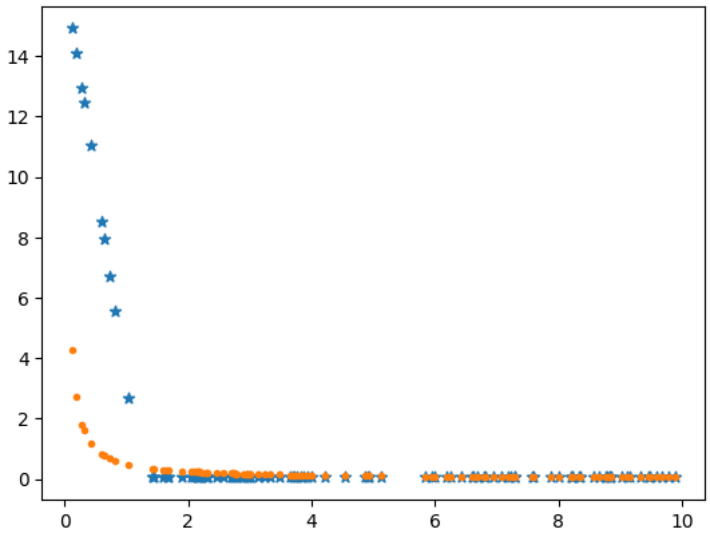
```
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 10 , learningRate: 0.01 , minimum_RMSE: 1227.02 , epoch:  389 , test_loss: 15
```





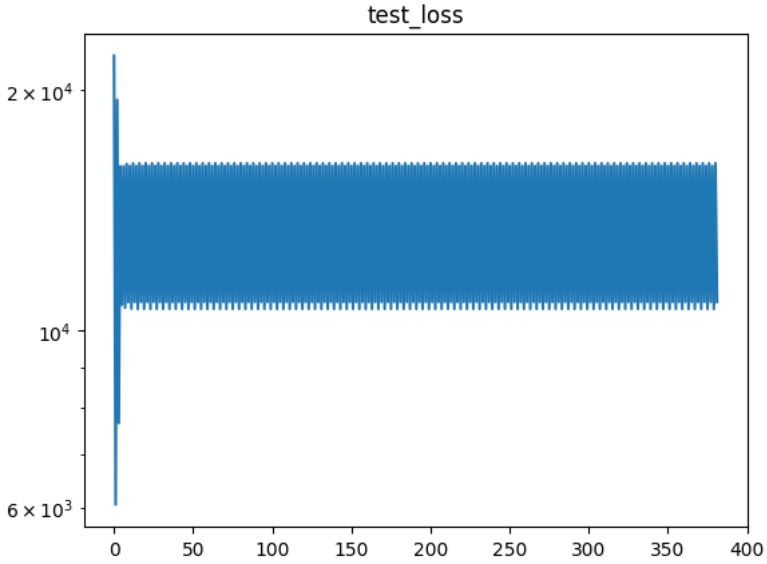


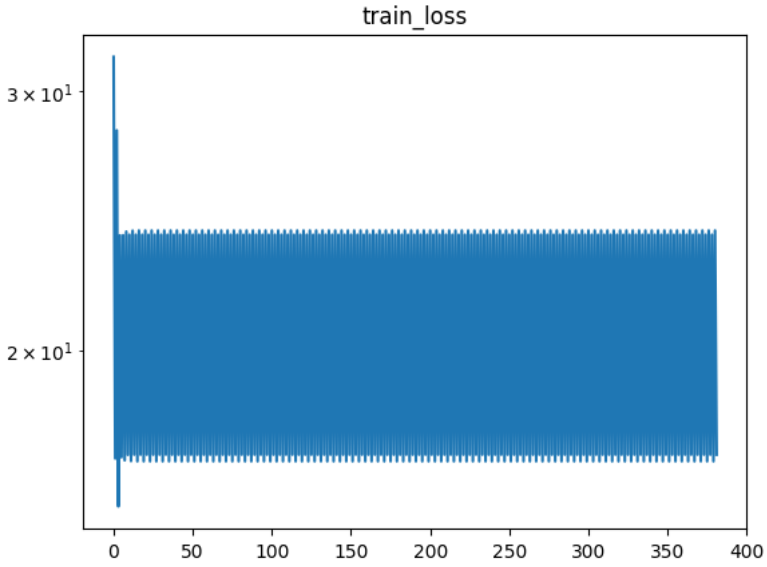


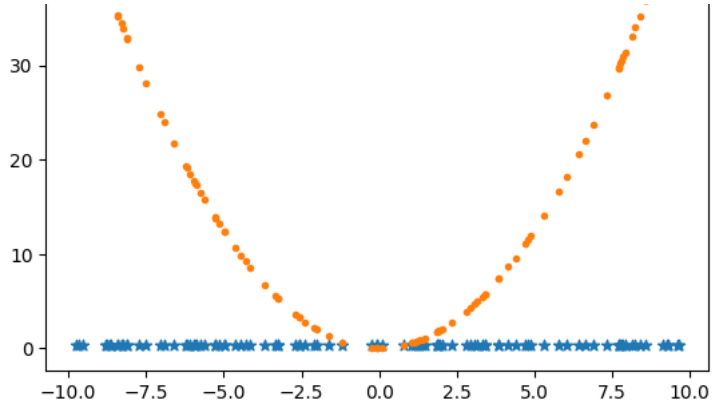


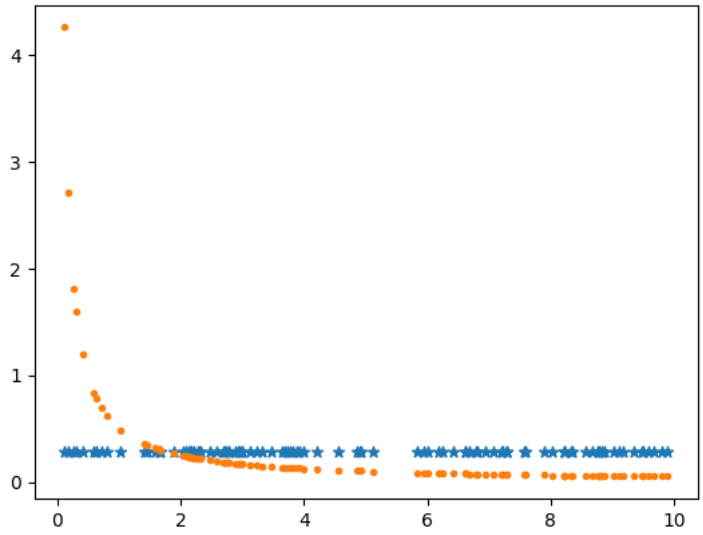

```
finish----------------------------------------------------------------------------------------------------------------------------------------------------------------
```

```
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 10 , learningRate: 0.05 , minimum_RMSE: 6055.33 , epoch:  381 , test_loss: 10
```



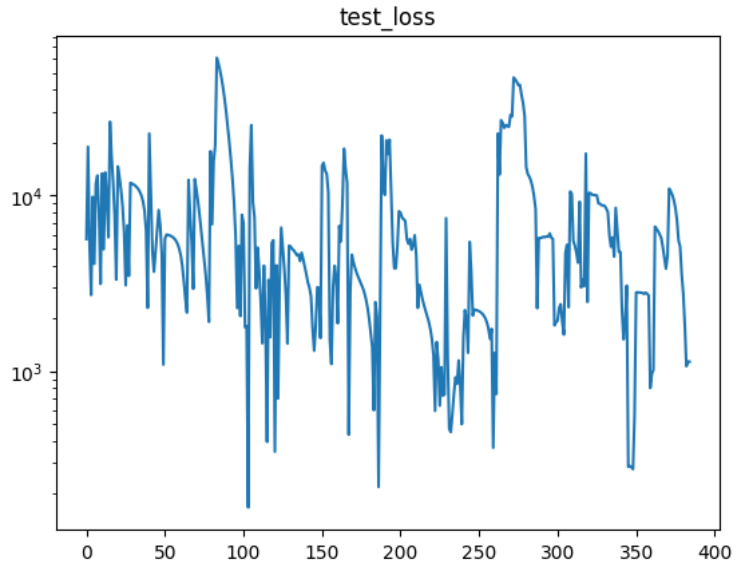


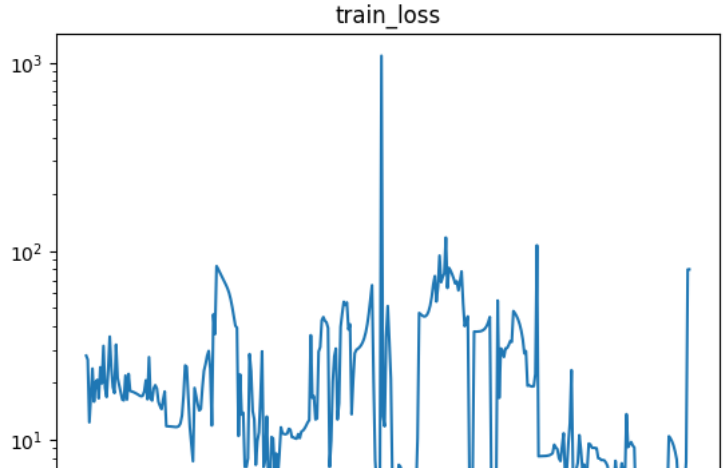


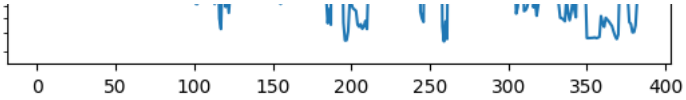


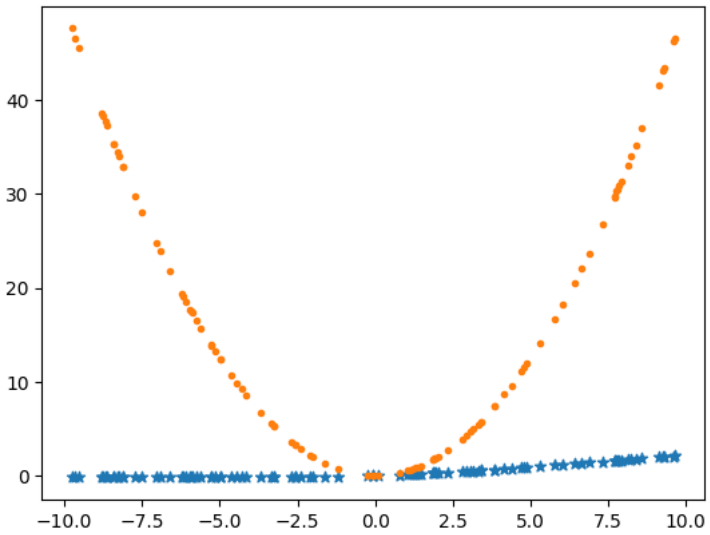

```
finish----------------------------------------------------------------------------------------------------------------------------------------------------------------
```

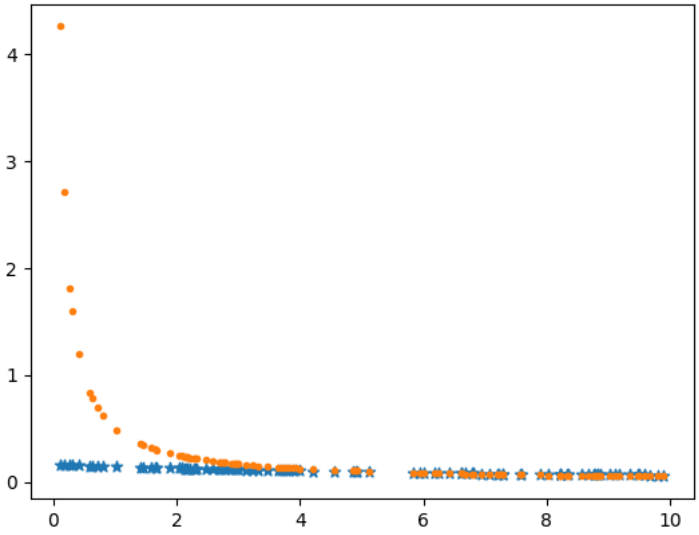
```
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 10 , learningRate: 0.001 , minimum_RMSE: 167.88 , epoch:   384 , test_loss: 11
```





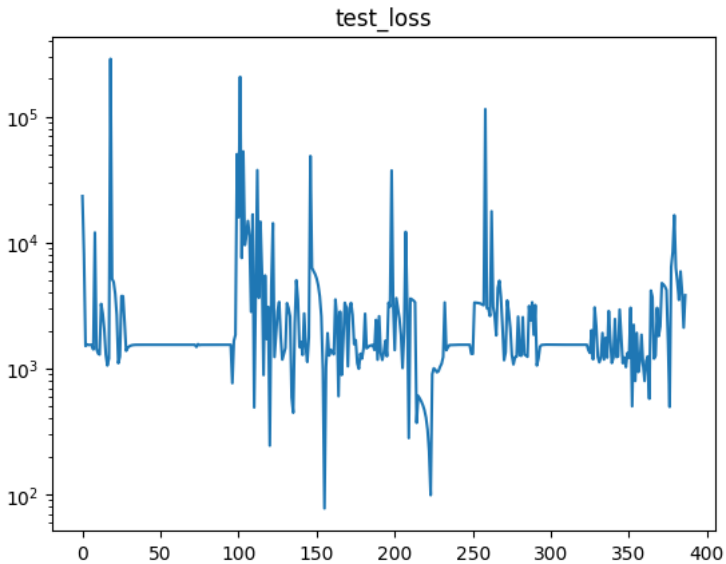


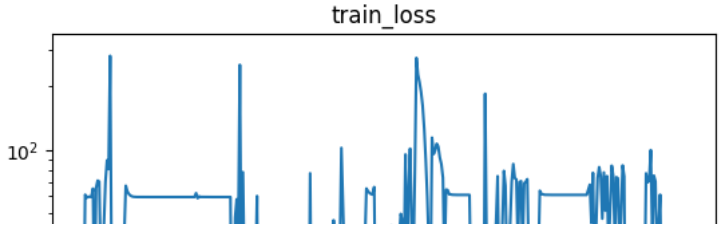


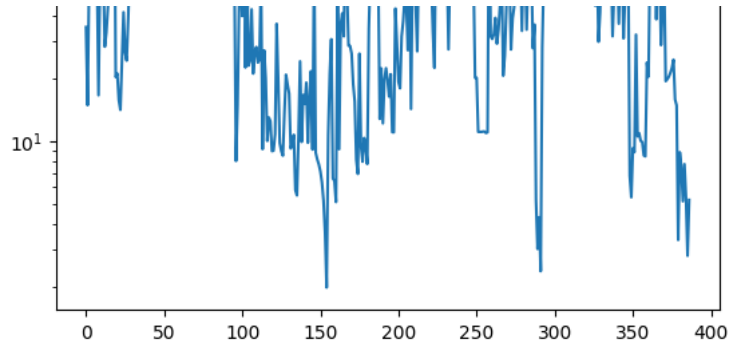


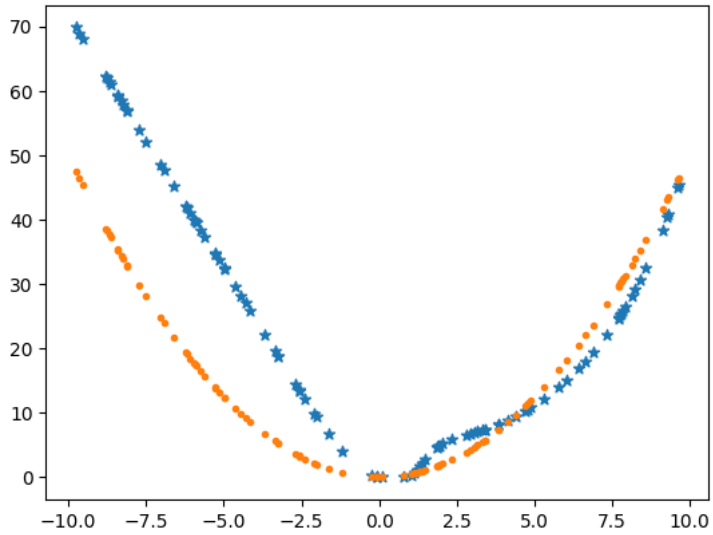
finish----------

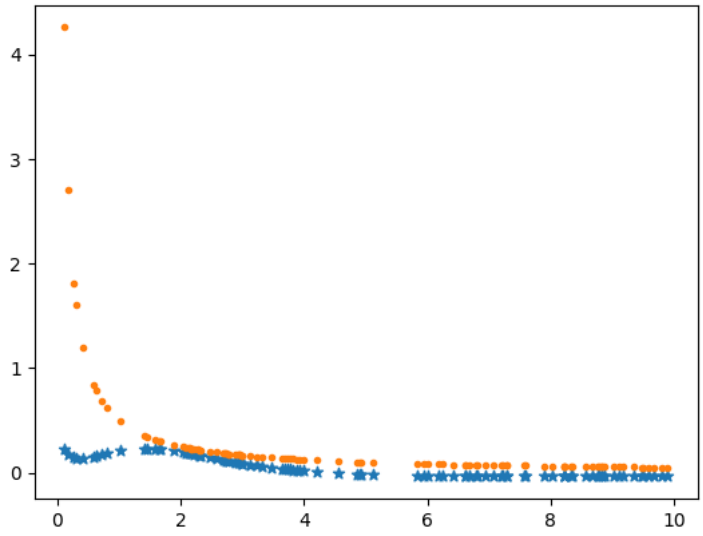
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 10 , learningRate: 0.005 , minimum_RMSE: 77.75 , epoch: 386 , test_loss: 382





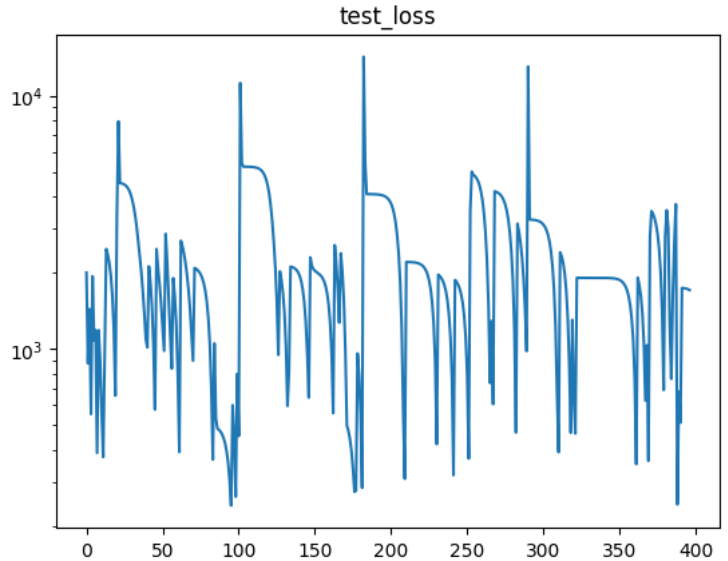


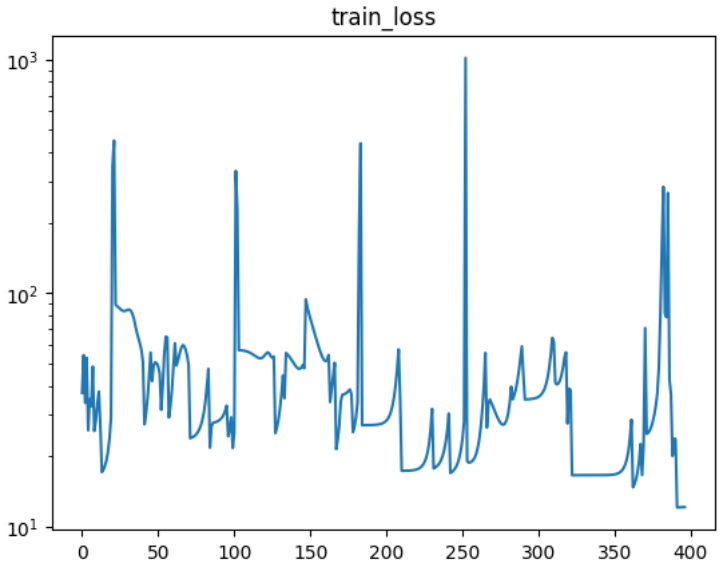


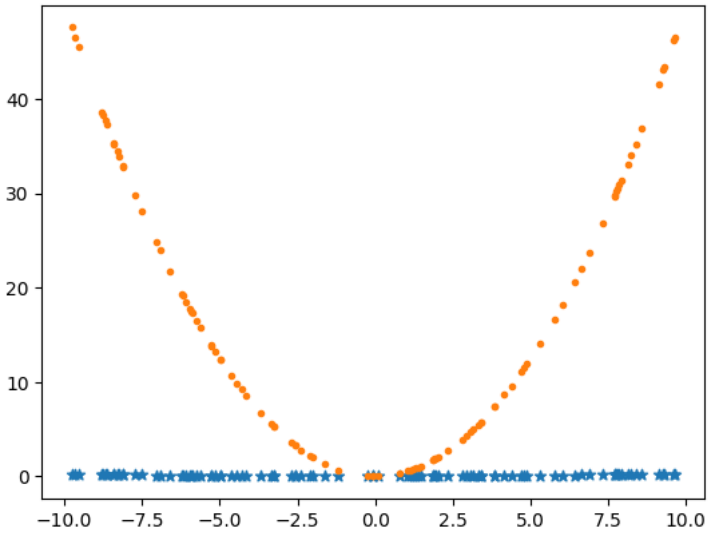


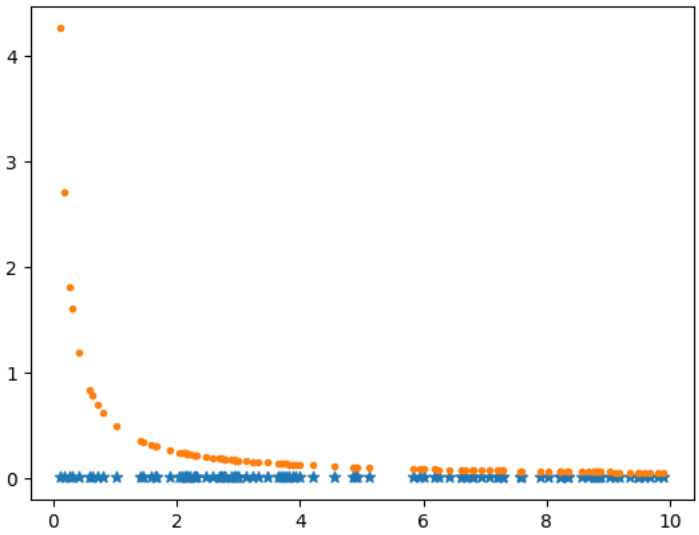

```
finish----------------------------------------------------------------------------------------------------
```

```
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 10 , learningRate: 0.0001 , minimum_RMSE: 241.47 , epoch:  396 , test_loss: 1
```



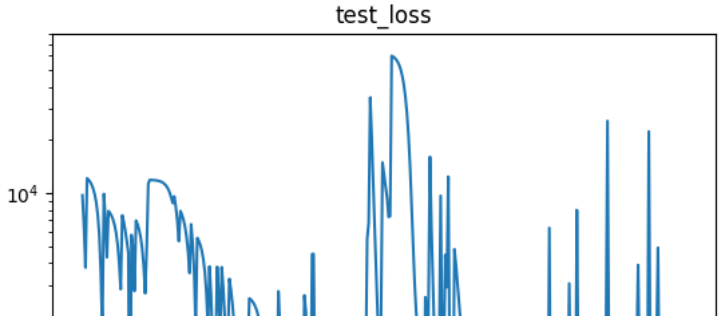


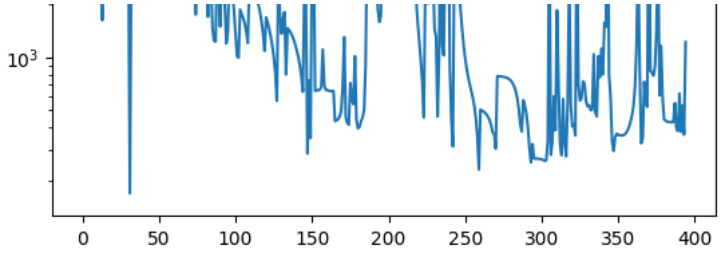


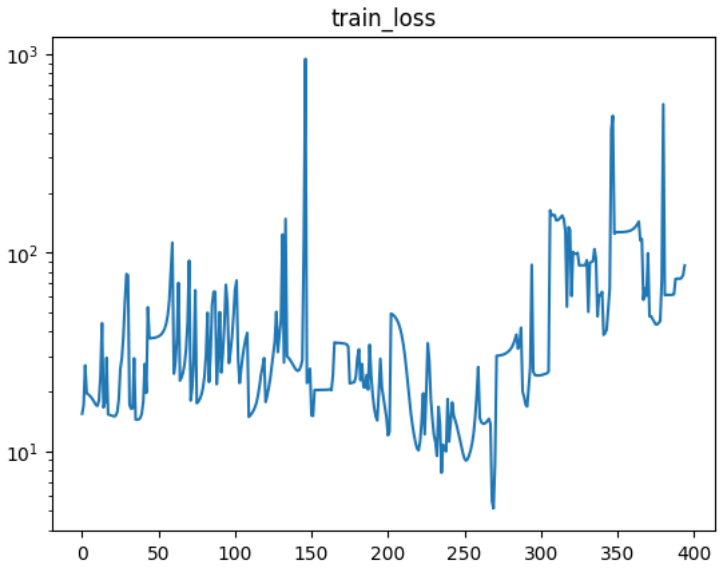


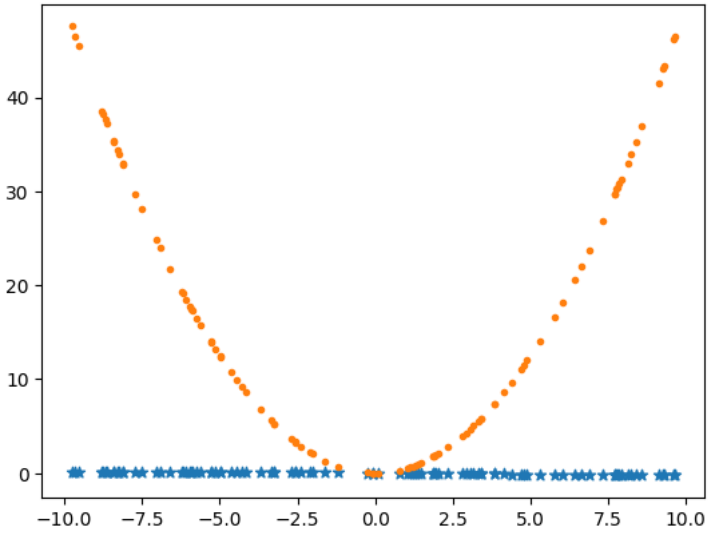
finish----------

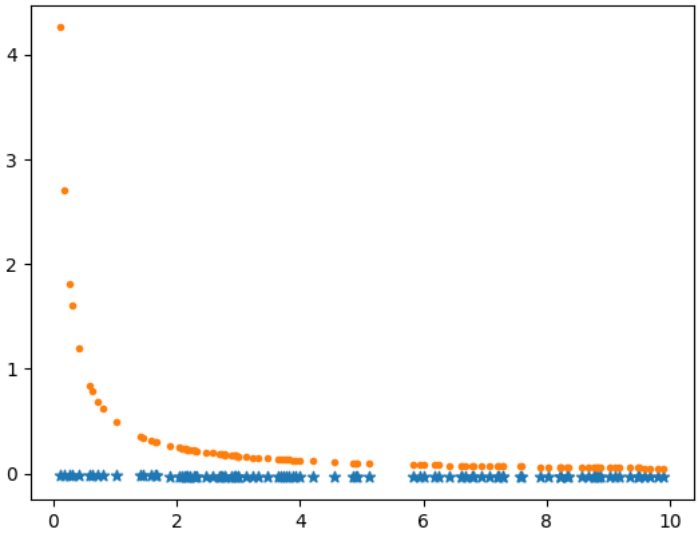
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 10 , learningRate: 0.0005 , minimum_RMSE: 168.95 , epoch: 394 , test_loss: 1





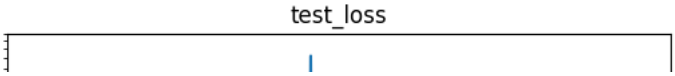


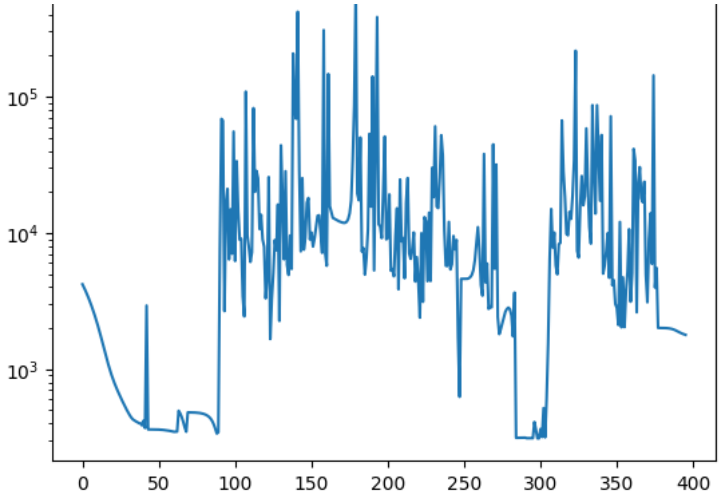


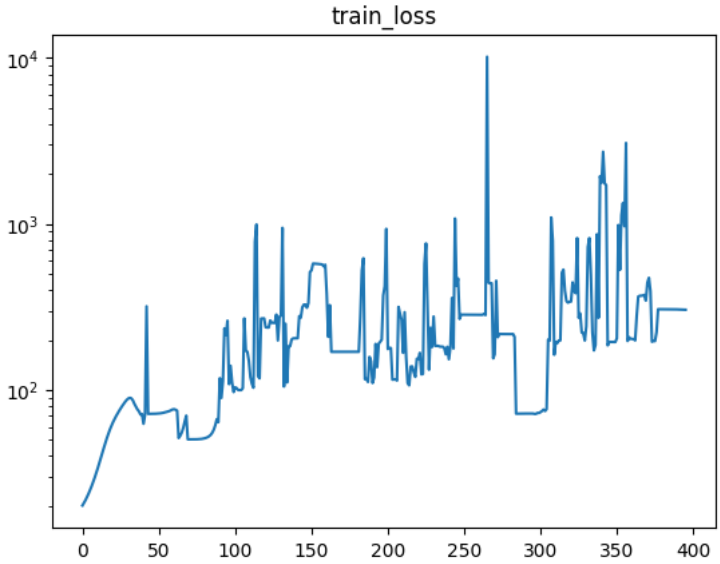


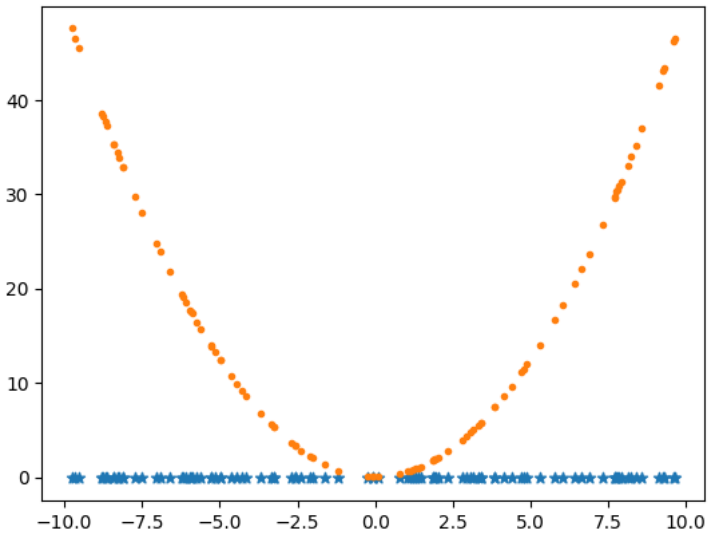
finish----------

activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 10 , learningRate: 1e-05 , minimum_RMSE: 310.15 , epoch: 395 , test_loss: 17




10^5
10^4
10^3
0 50 100 150 200 250 300 350 400

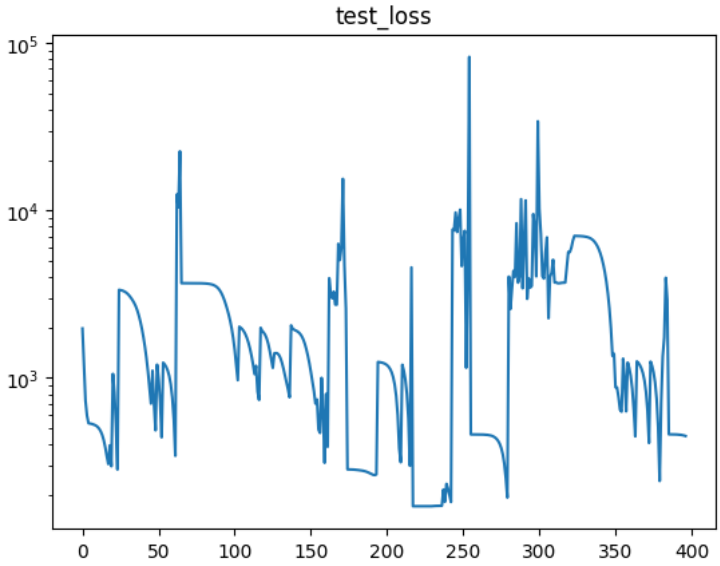

train_loss
10^4
10^3
10^2
0 50 100 150 200 250 300 350 400

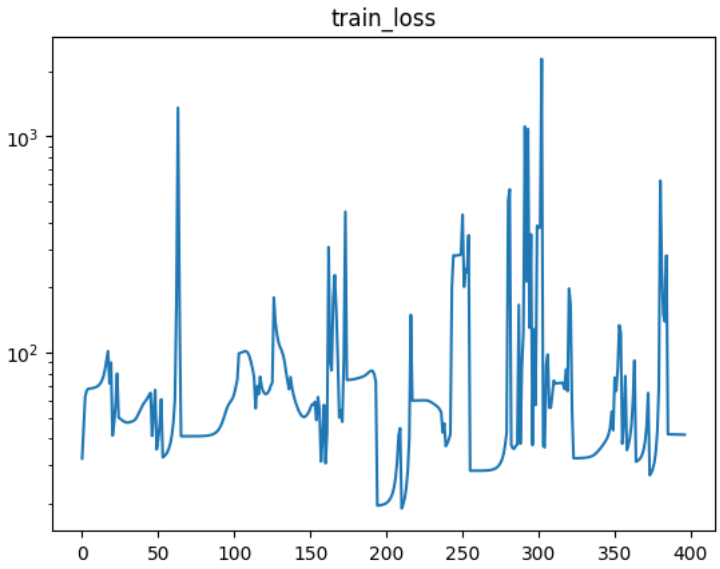

40
30
20
10
0
−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0 7.5 10.0

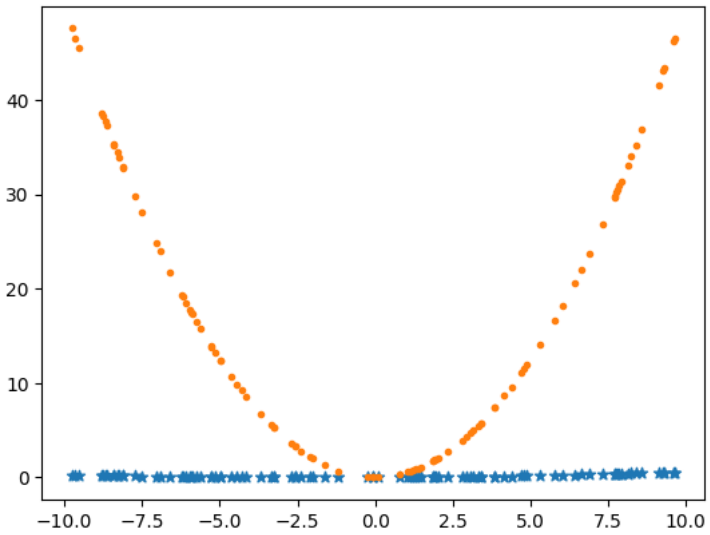

4
3
2
1

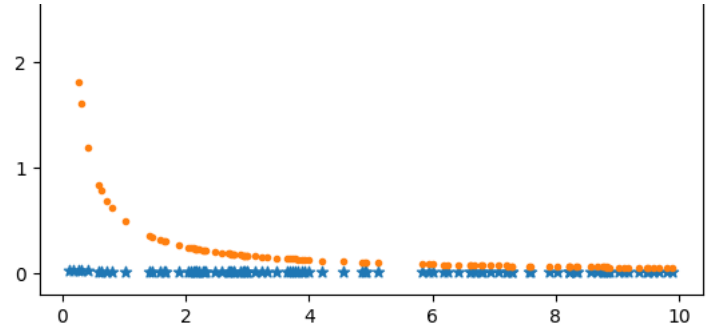
finish----------

activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 10 , learningRate: 5e-05 , minimum_RMSE: 170.68 , epoch: 396 , test_loss: 44



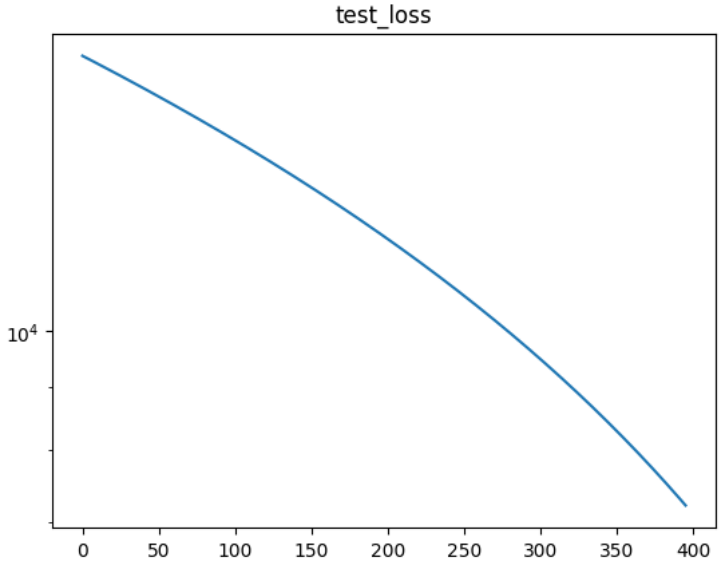


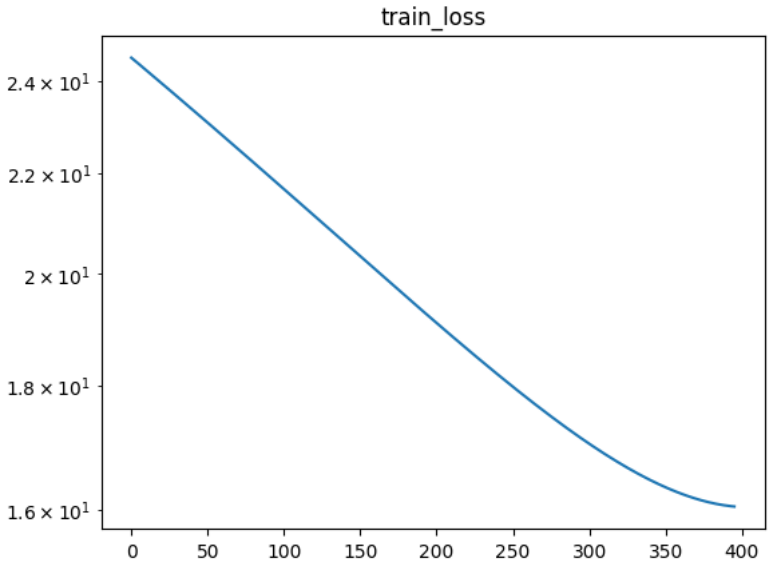


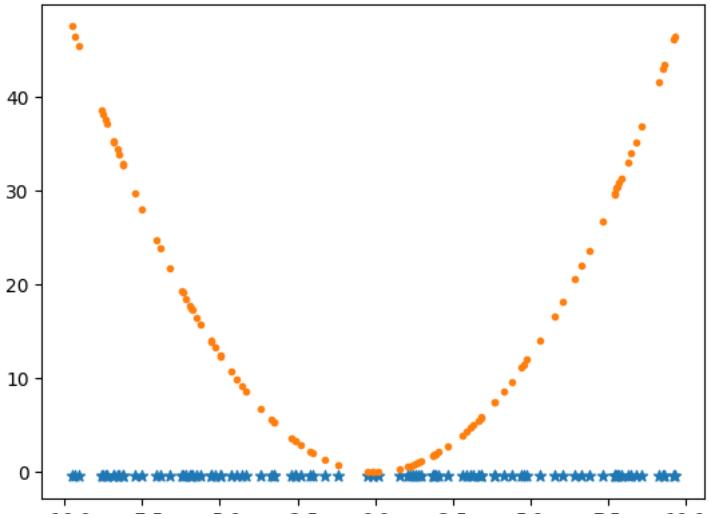


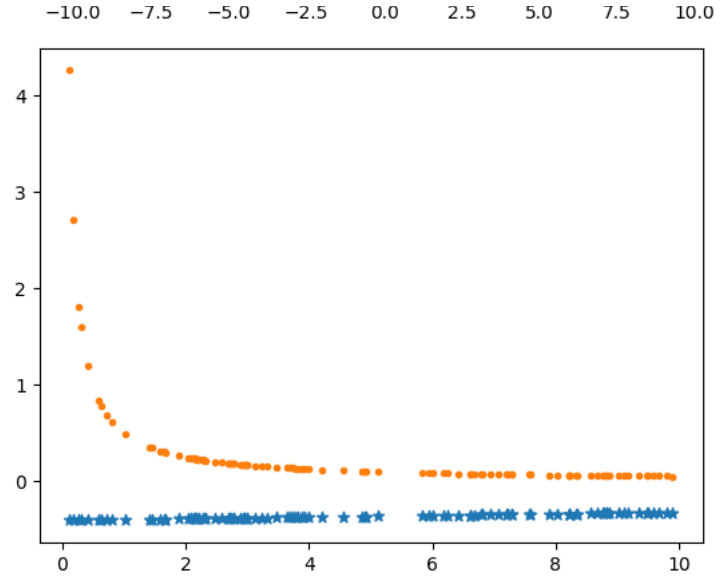
finish----------

activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 10 , learningRate: 1e-06 , minimum_RMSE: 7217.88 , epoch: 395 , test_loss: 7



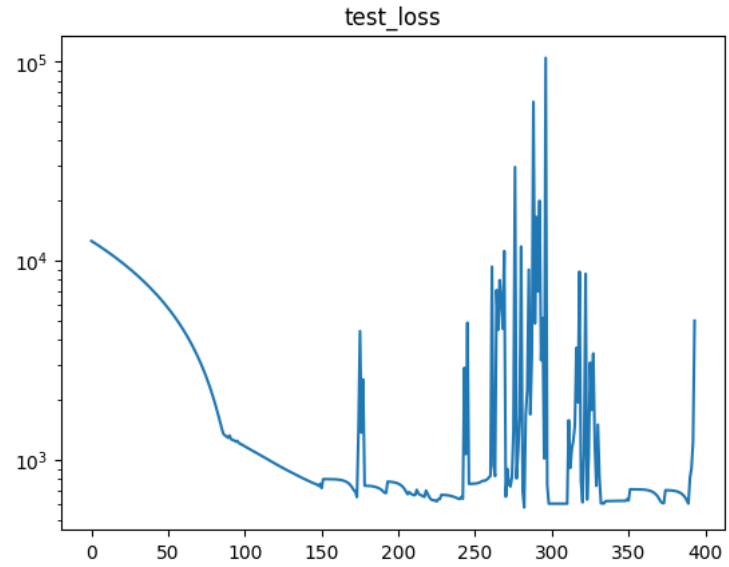


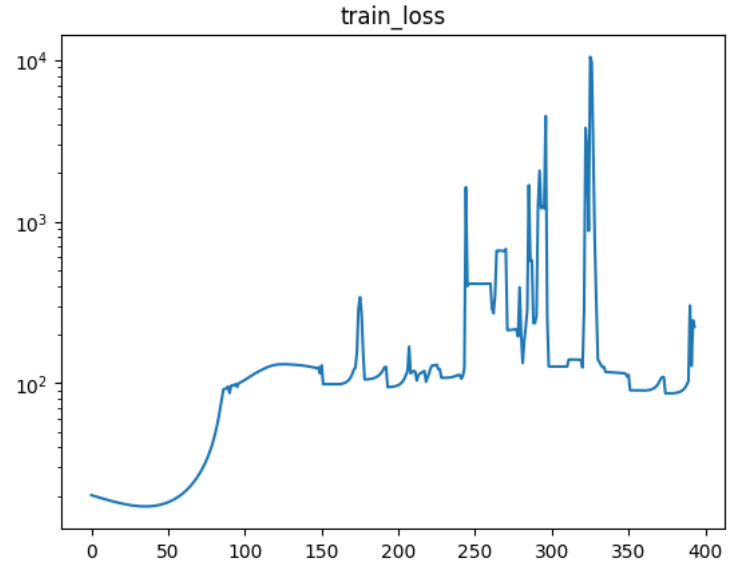


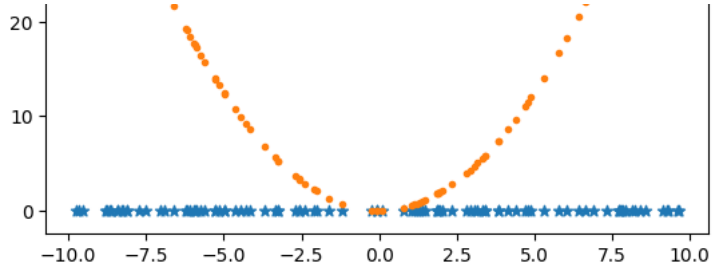


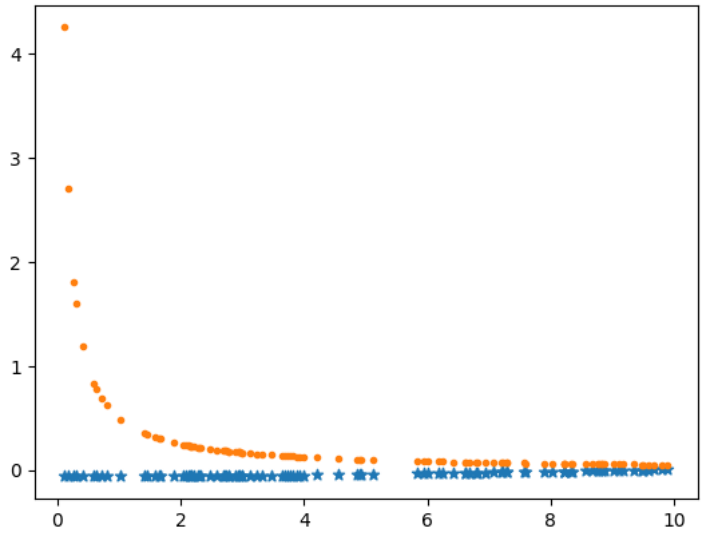
finish----------

activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 10 , learningRate: 5e-06 , minimum_RMSE: 575.59 , epoch: 393 , test_loss: 49



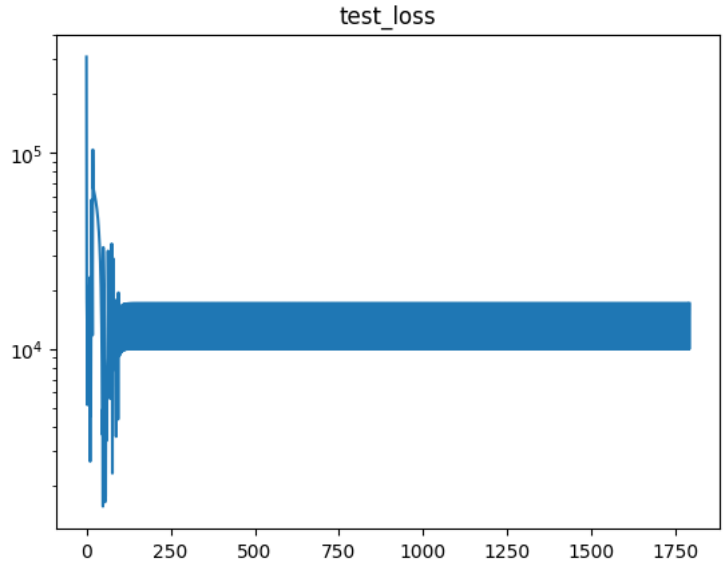


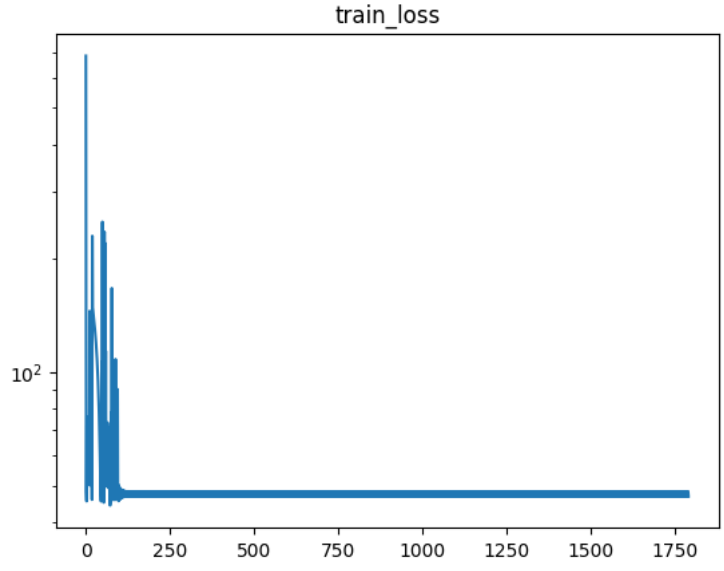


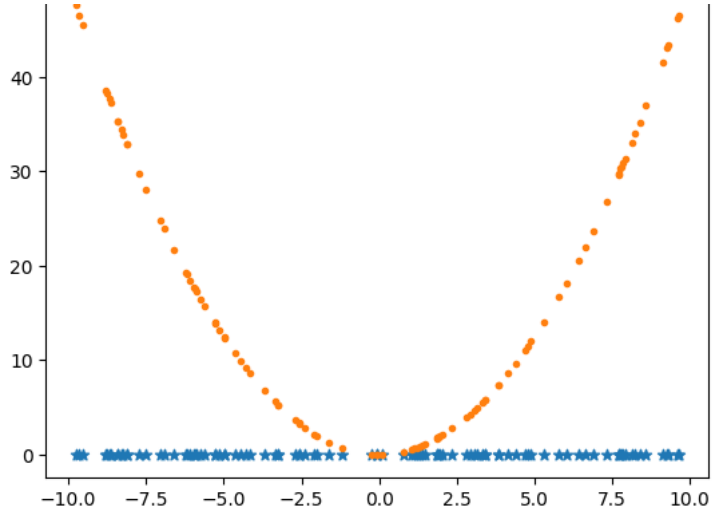


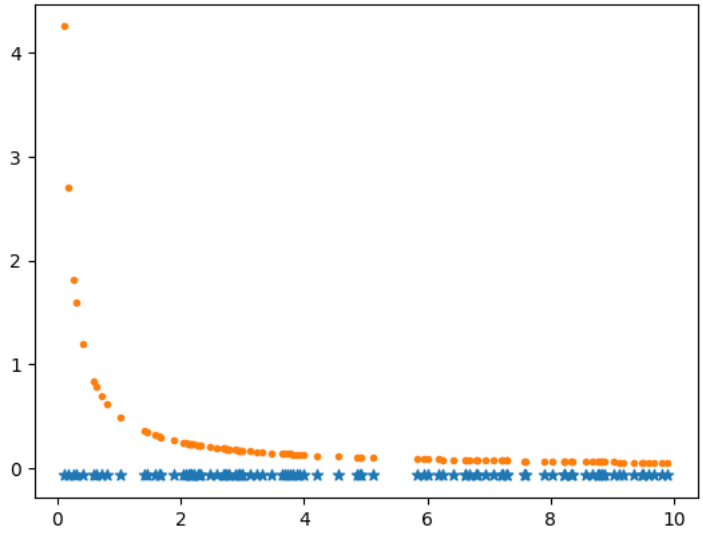
finish----------

activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 50 , learningRate: 0.1 , minimum_RMSE: 1577.01 , epoch: 1791 , test_loss: 170



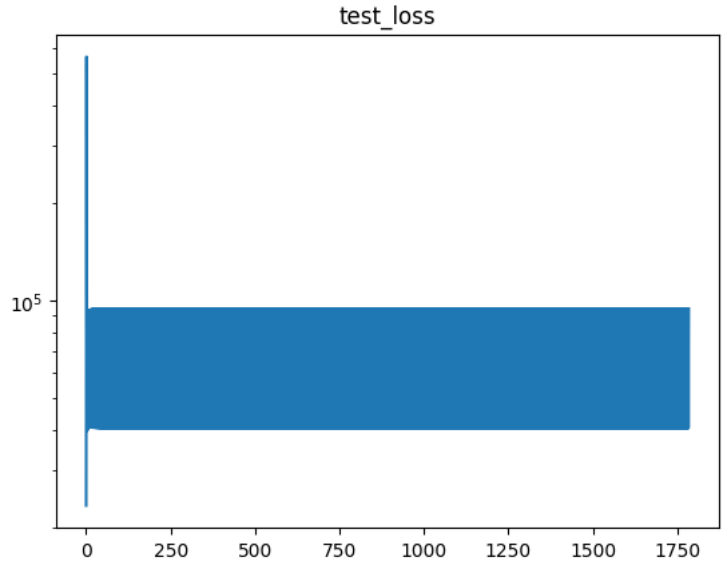


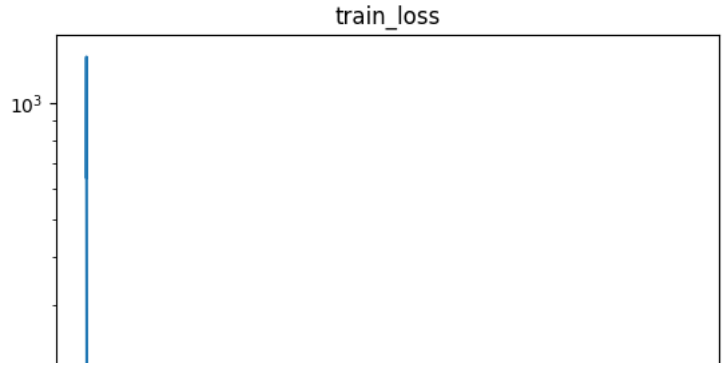


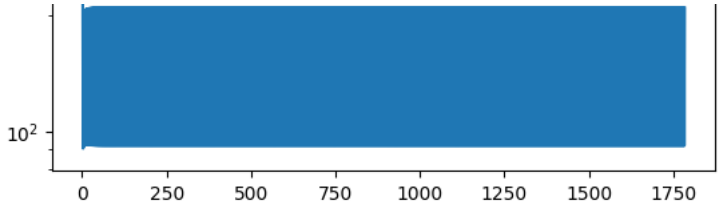


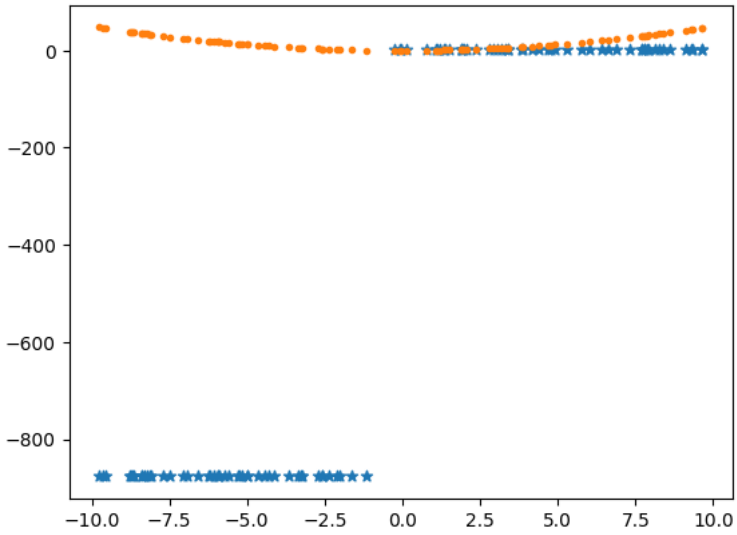
finish----------

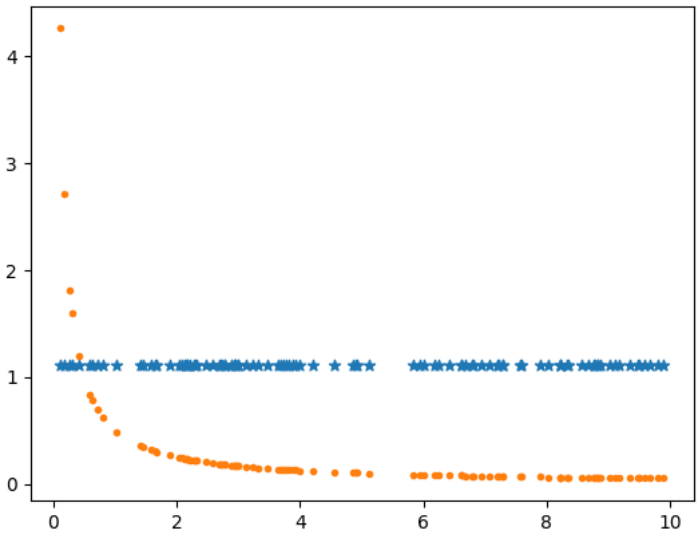
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 50 , learningRate: 0.5 , minimum_RMSE: 23365.86 , epoch: 1784 , test_loss: 94





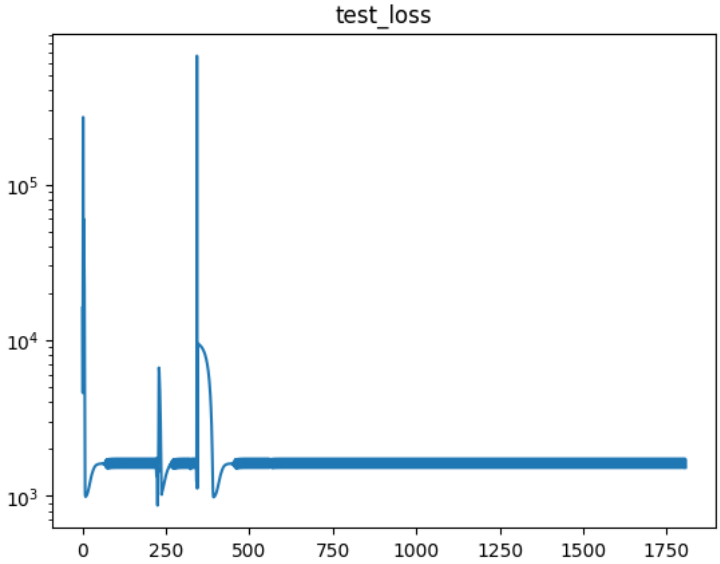


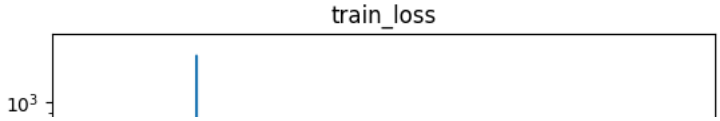


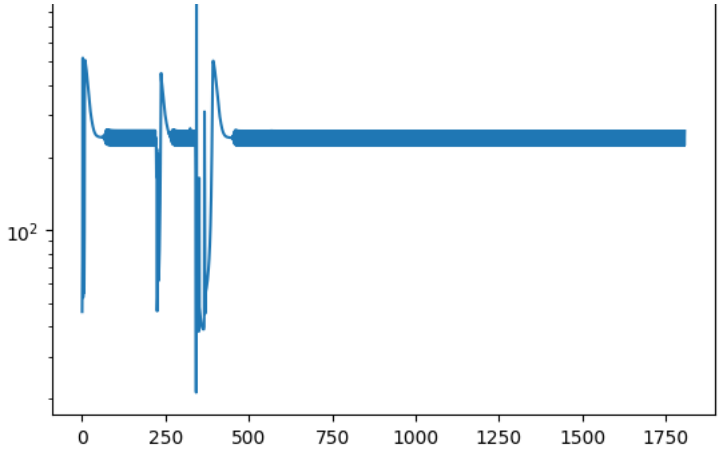


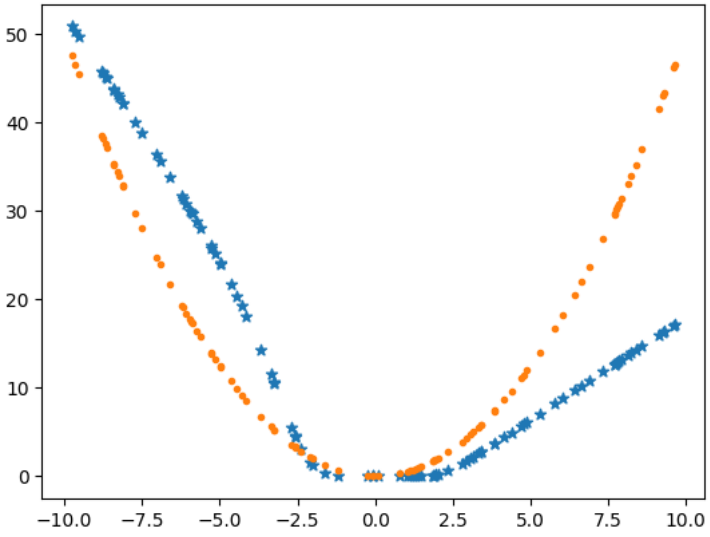
finish----------

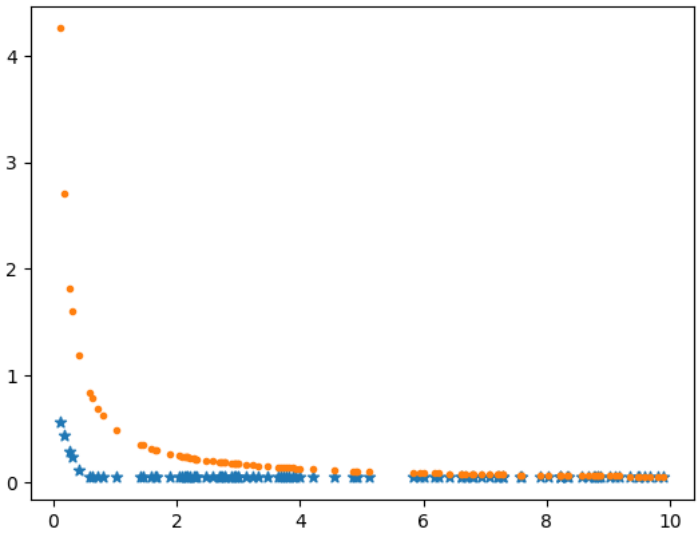
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 50 , learningRate: 0.01 , minimum_RMSE: 870.28 , epoch: 1807 , test_loss: 152





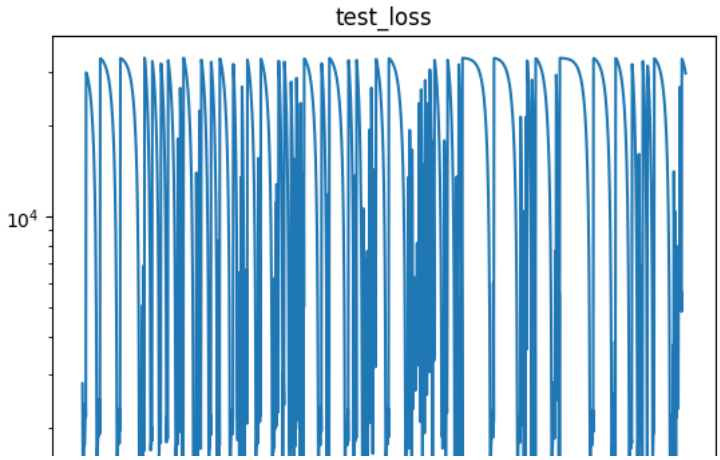


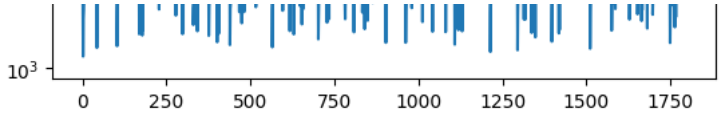


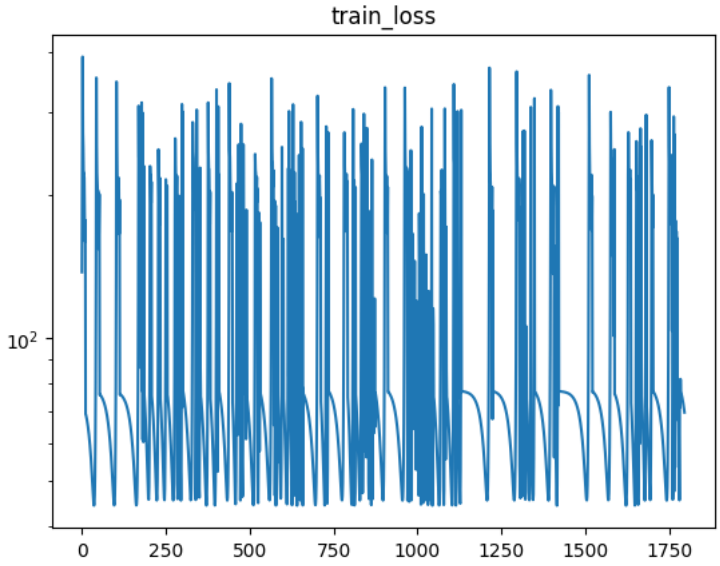


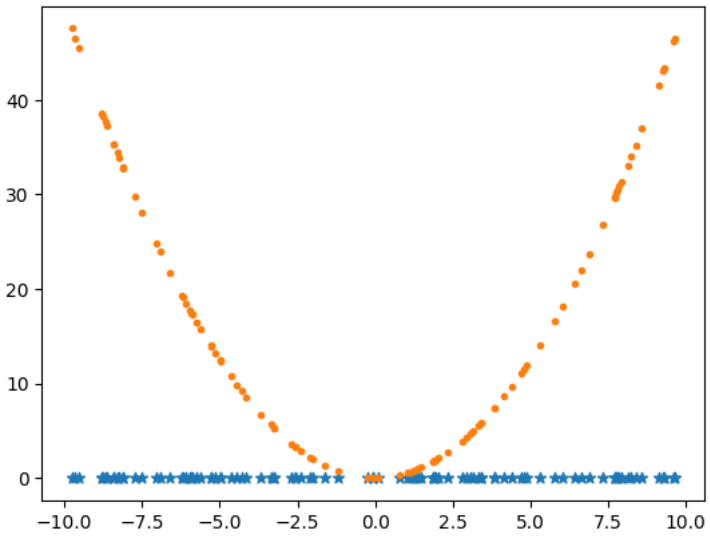
finish----------

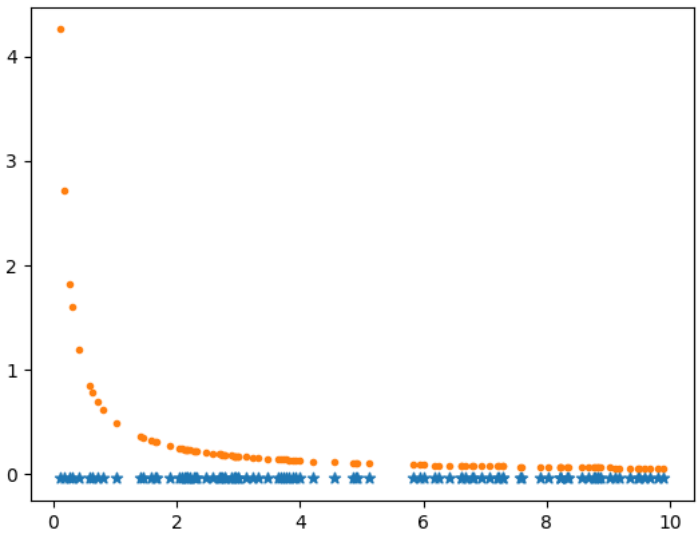
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 50 , learningRate: 0.05 , minimum_RMSE: 1094.93 , epoch: 1795 , test_loss: 29





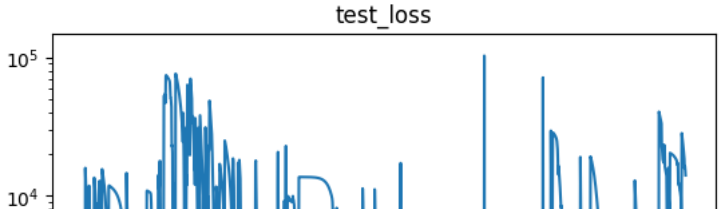


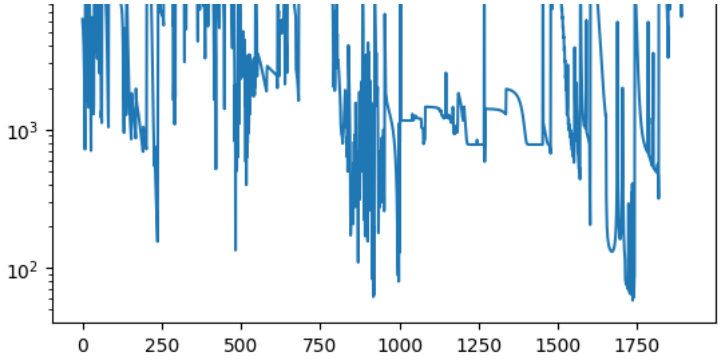


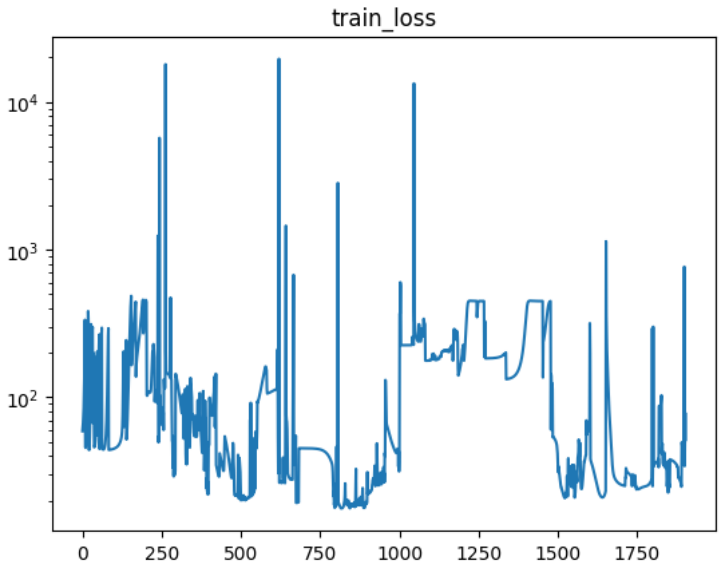


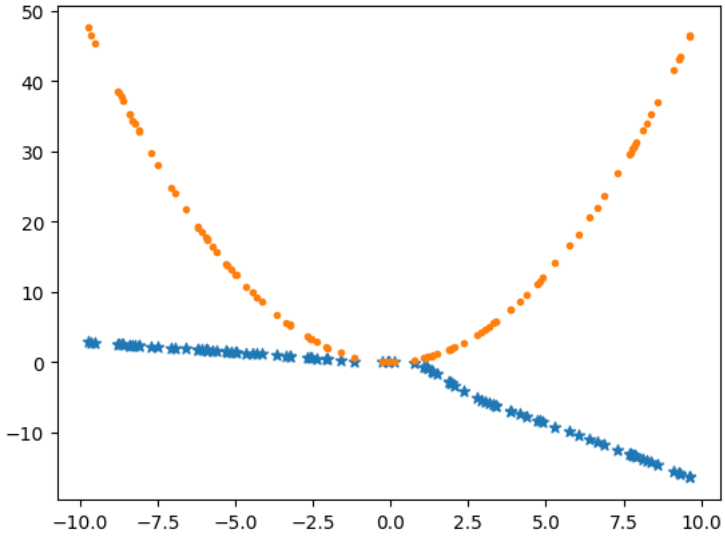

```
finish----------------------------------------------------------------------------------------------------------------------------------------------------------------
```

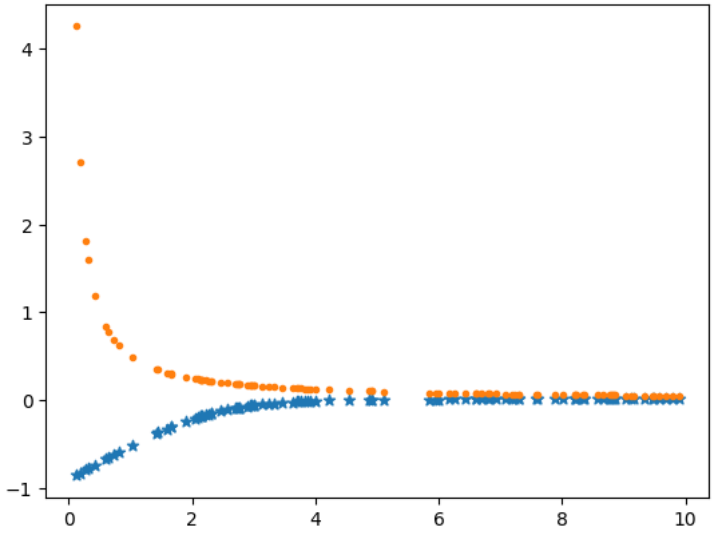
```
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 50 , learningRate: 0.001 , minimum_RMSE: 58.14 , epoch:  1903 , test_loss: 140
```




10^3
10^2
0 250 500 750 1000 1250 1500 1750

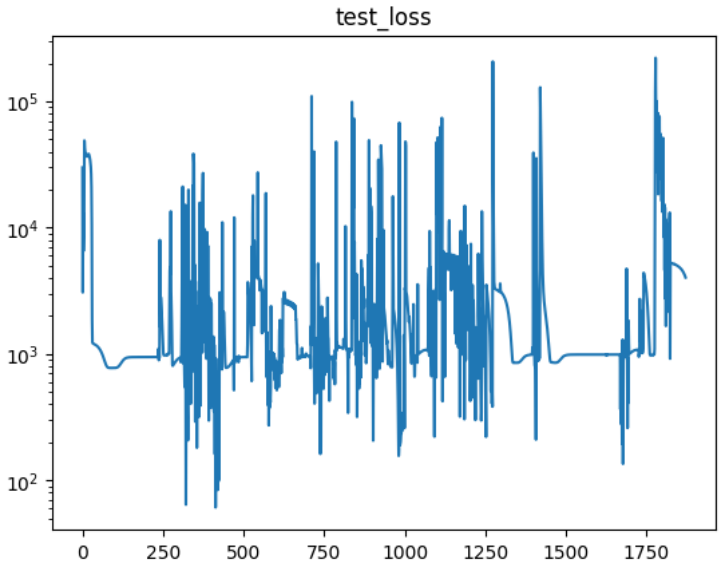

train_loss
10^4
10^3
10^2
0 250 500 750 1000 1250 1500 1750

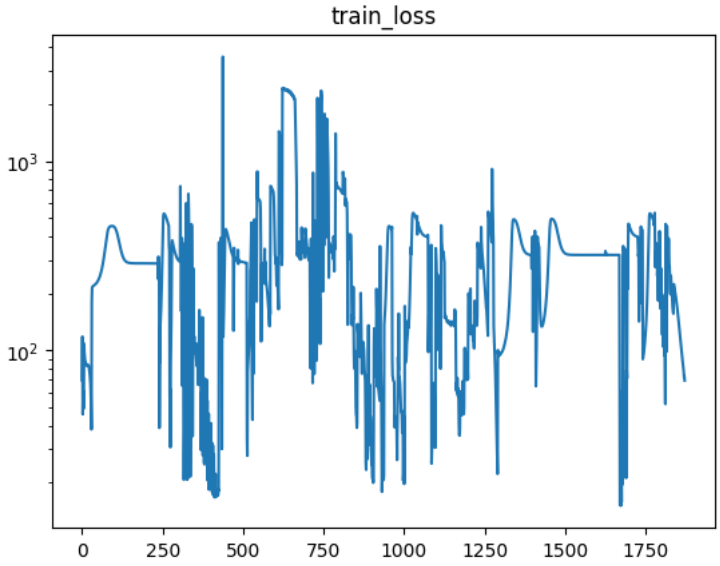

50 40 30 20 10 0 −10
−10.0 −7.5 −5.0 −2.5 0.0 2.5 5.0 7.5 10.0

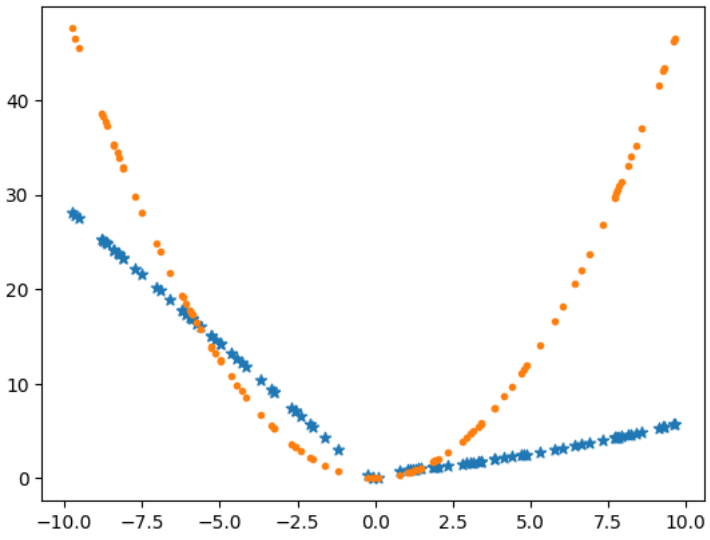

4 3 2 1 0 −1
0 2 4 6 8 10

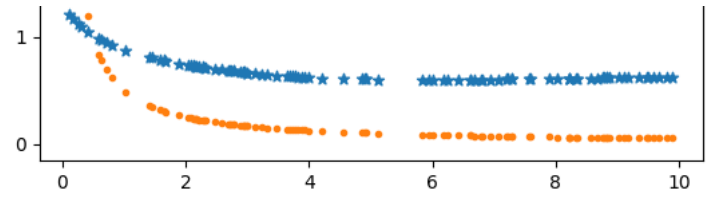
finish

```
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 50 , learningRate: 0.005 , minimum_RMSE: 61.11 , epoch:  1871 , test_loss: 403
```

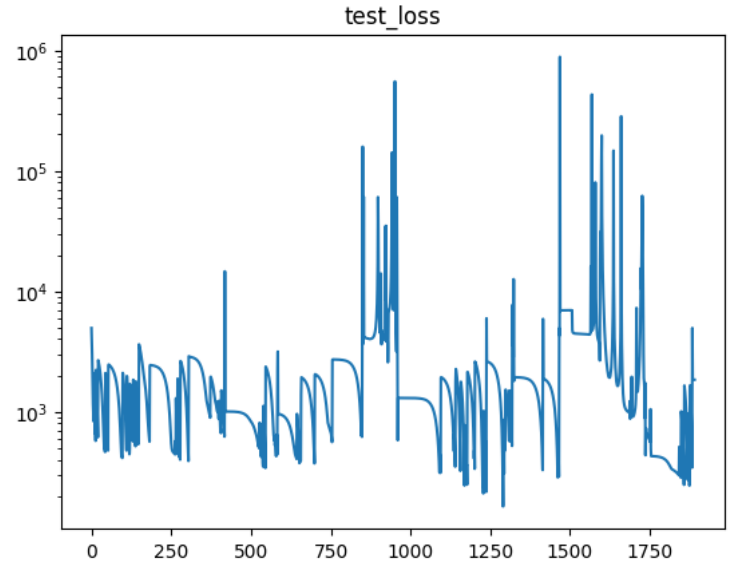


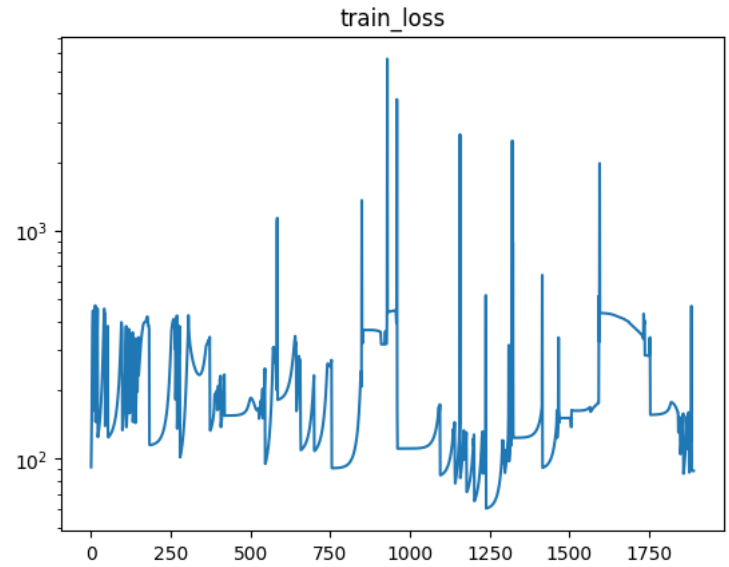


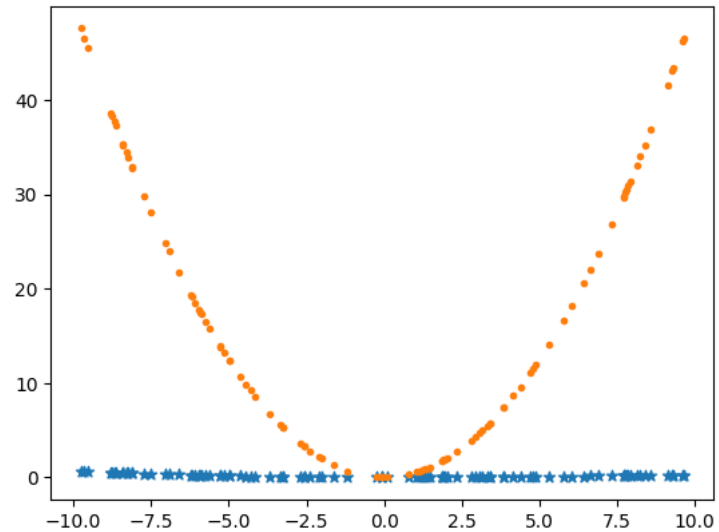


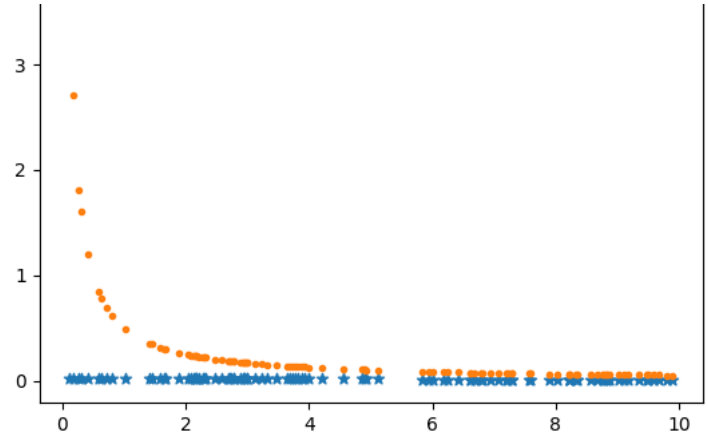


```
finish----------------------------------------------------------------------------------------------------------------------------------------------------------------

activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 50 , learningRate: 0.0001 , minimum_RMSE: 163.81 , epoch:  1890 , test_loss: 1
```



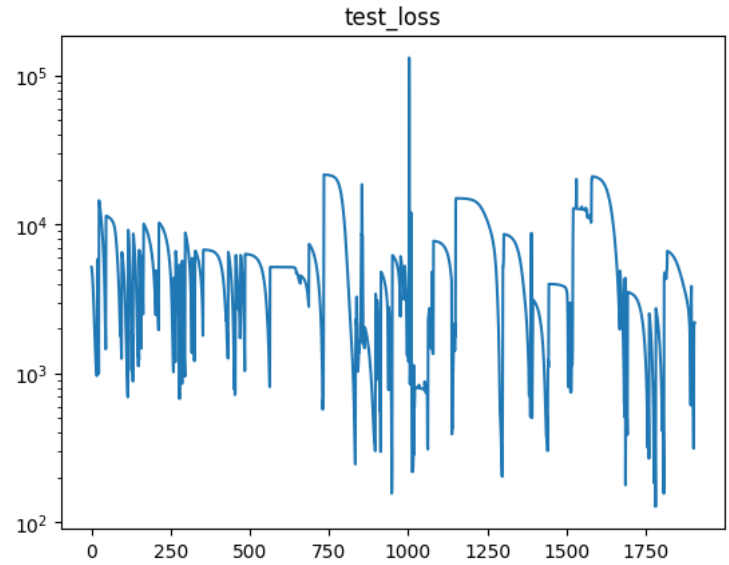


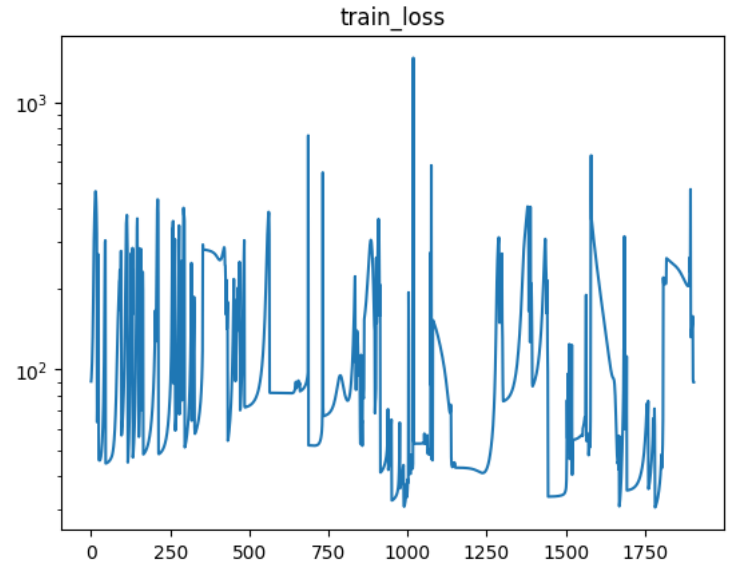


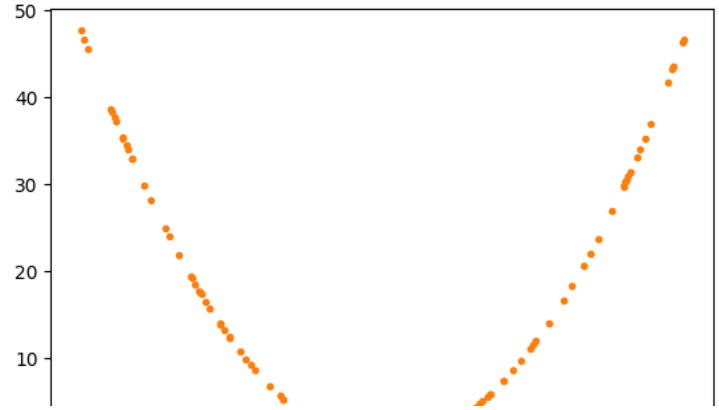


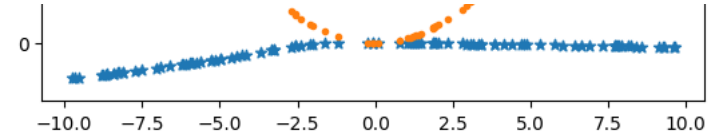
finish----------

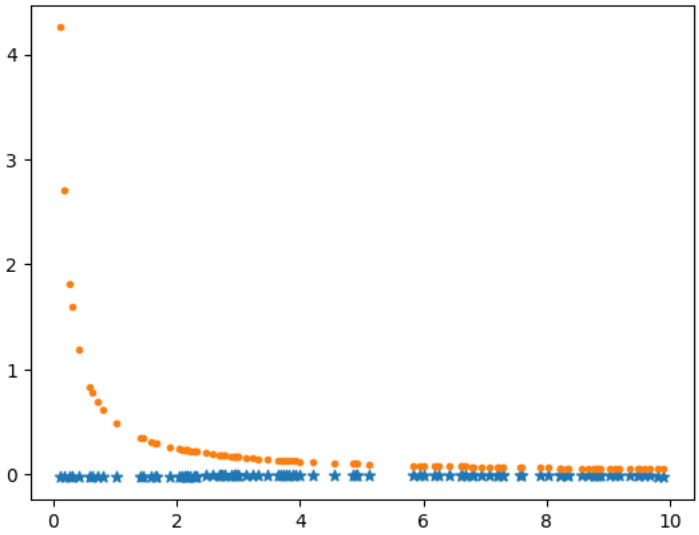
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 50 , learningRate: 0.0005 , minimum_RMSE: 127.92 , epoch: 1904 , test_loss: 2





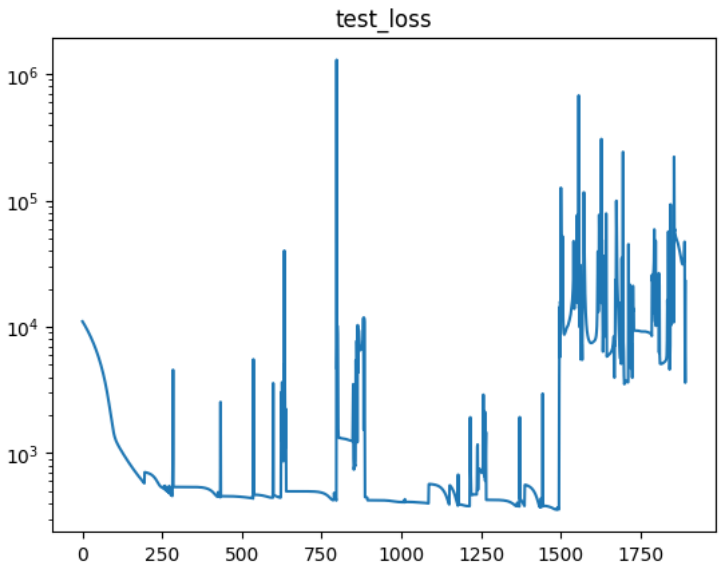


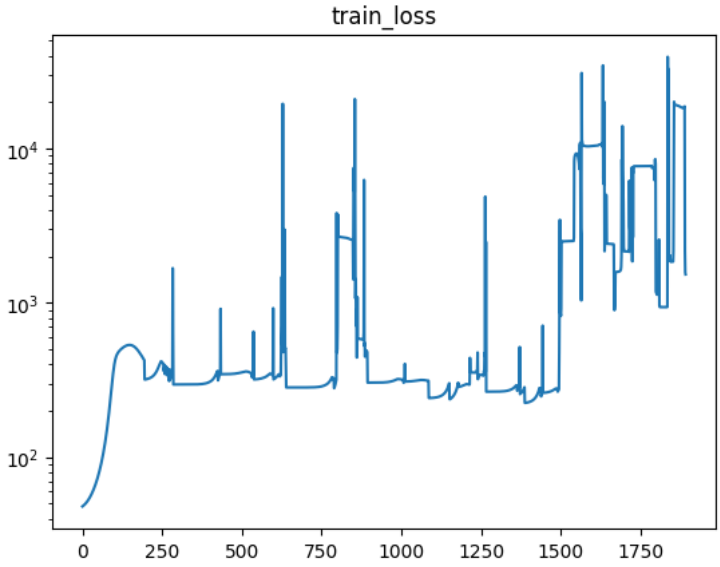


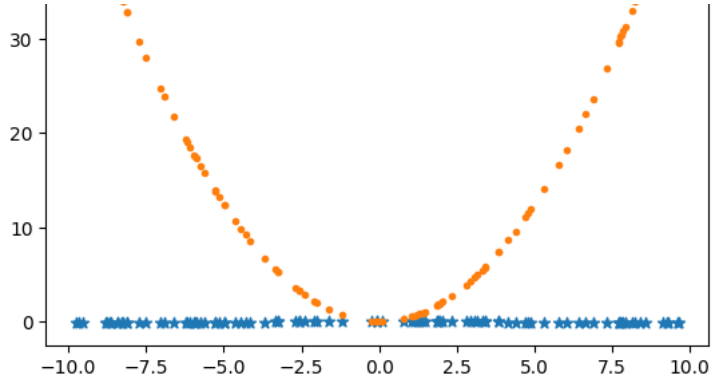


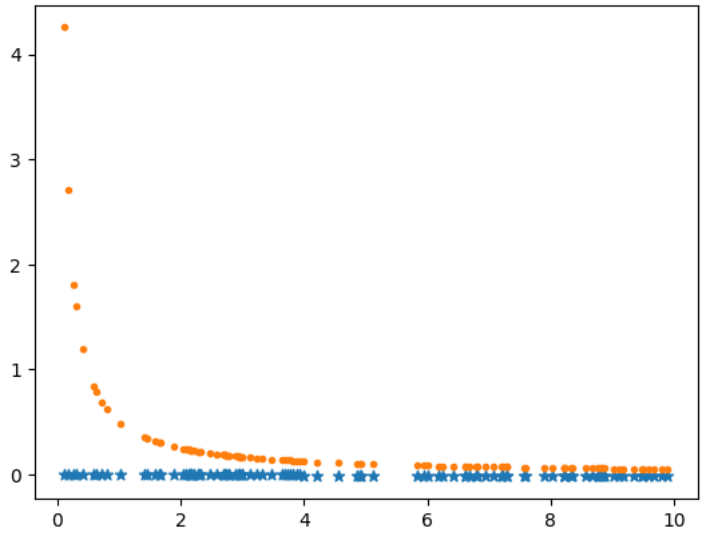
finish----------

activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 50 , learningRate: 1e-05 , minimum_RMSE: 359.21 , epoch: 1891 , test_loss: 22



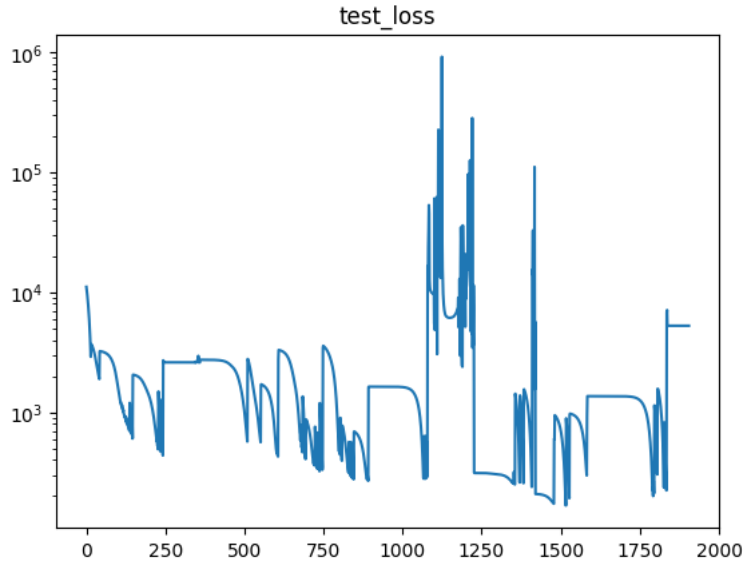


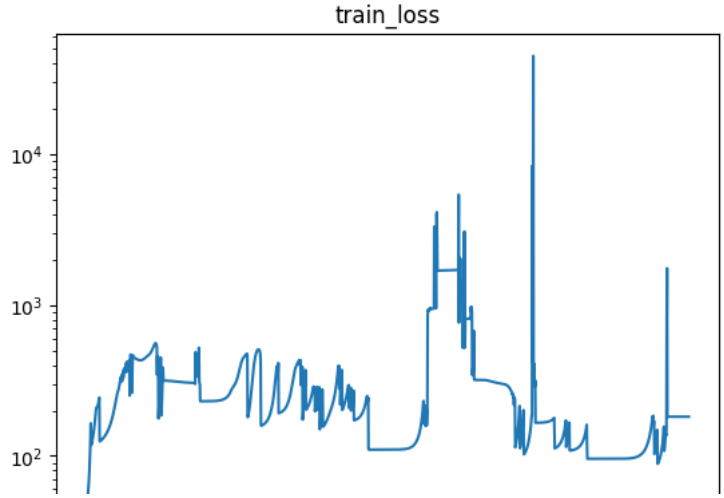


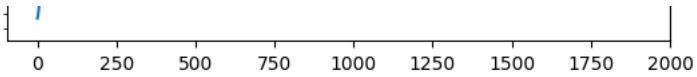


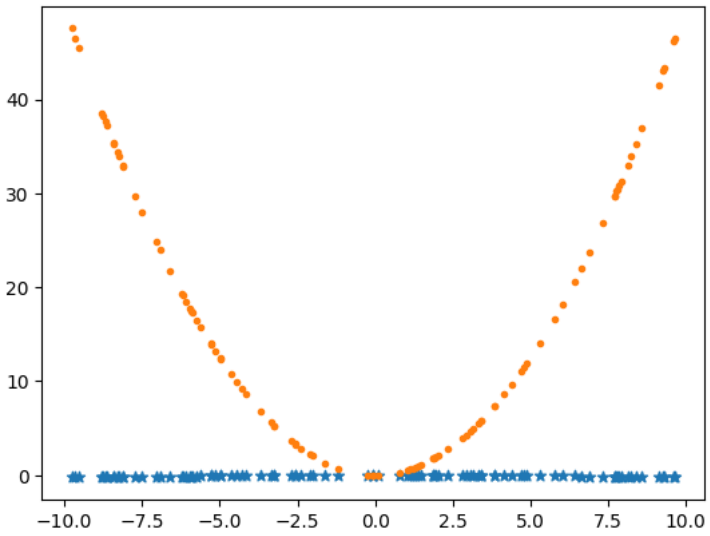
finish----------

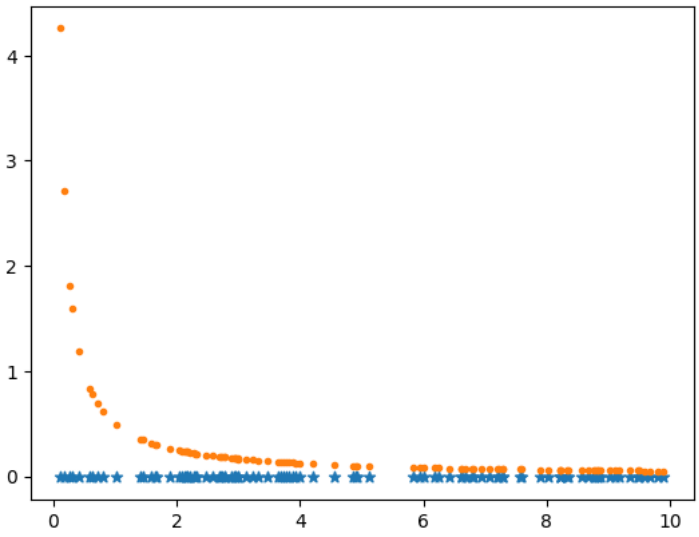
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 50 , learningRate: 5e-05 , minimum_RMSE: 166.91 , epoch: 1905 , test_loss: 52





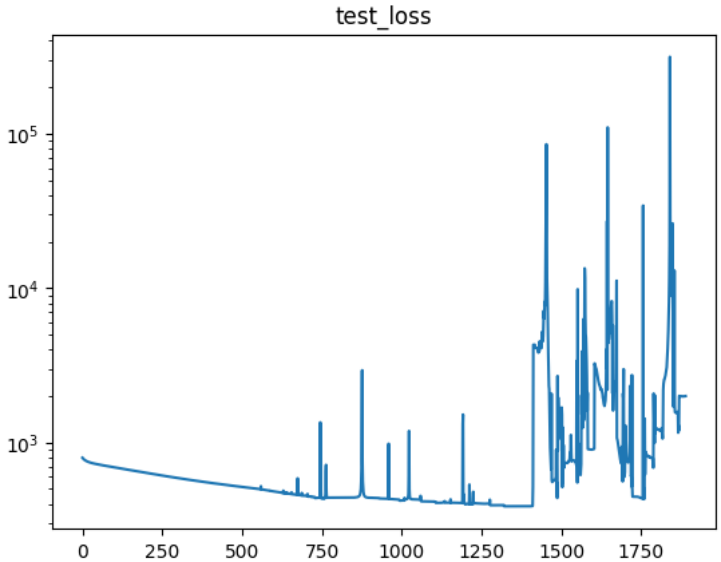


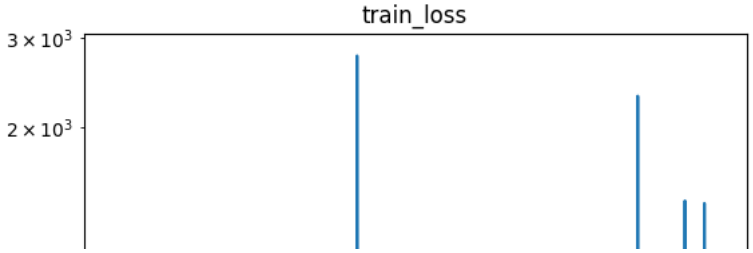


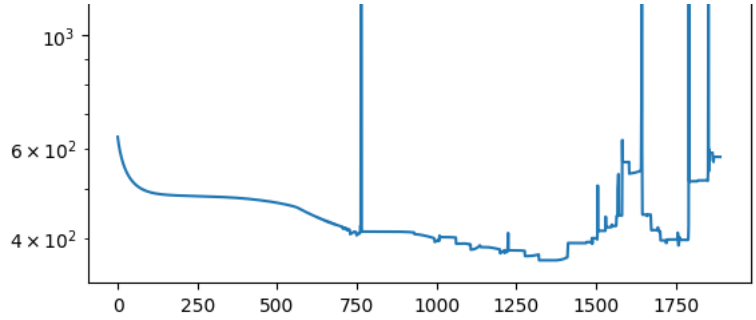


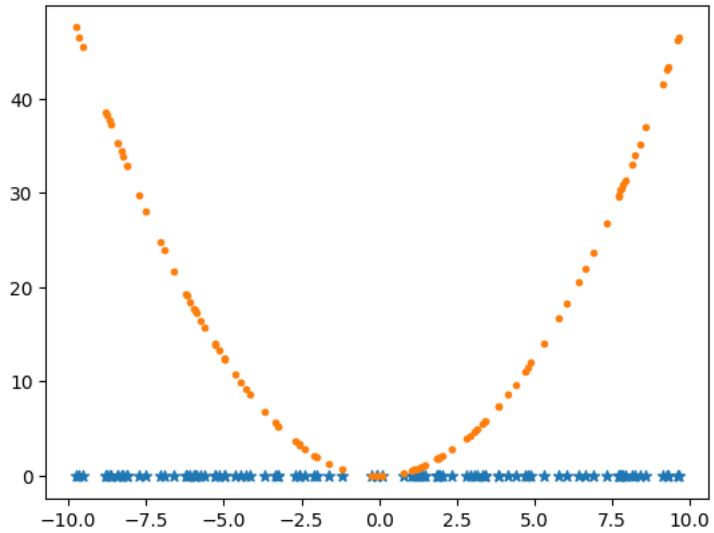
finish----------

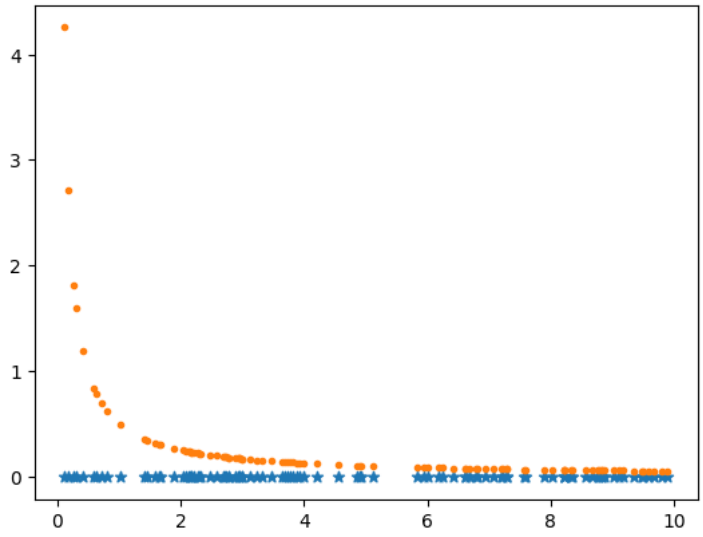
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 50 , learningRate: 1e-06 , minimum_RMSE: 385.74 , epoch: 1889 , test_loss: 19





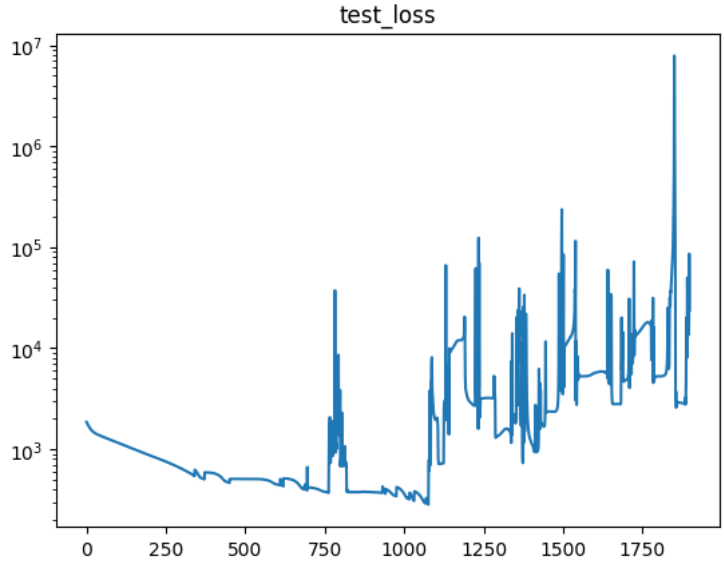


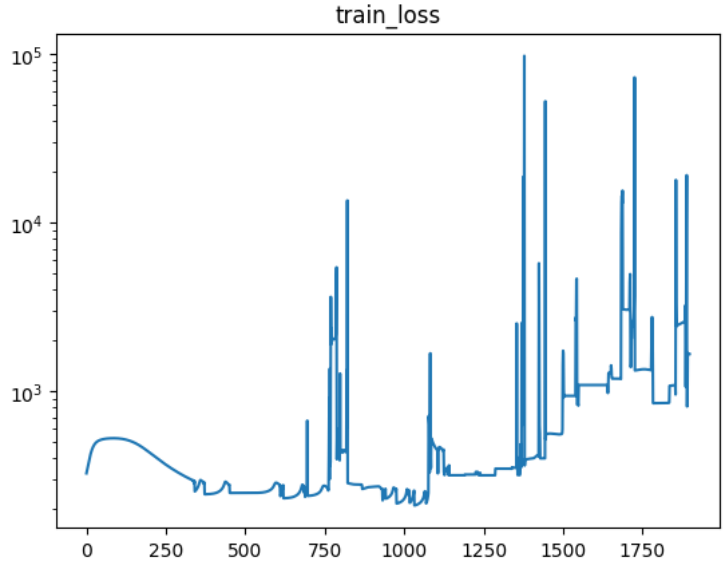


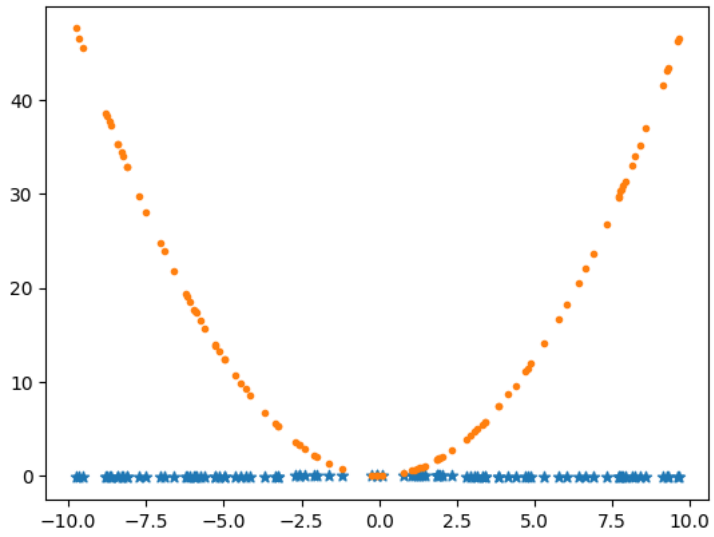


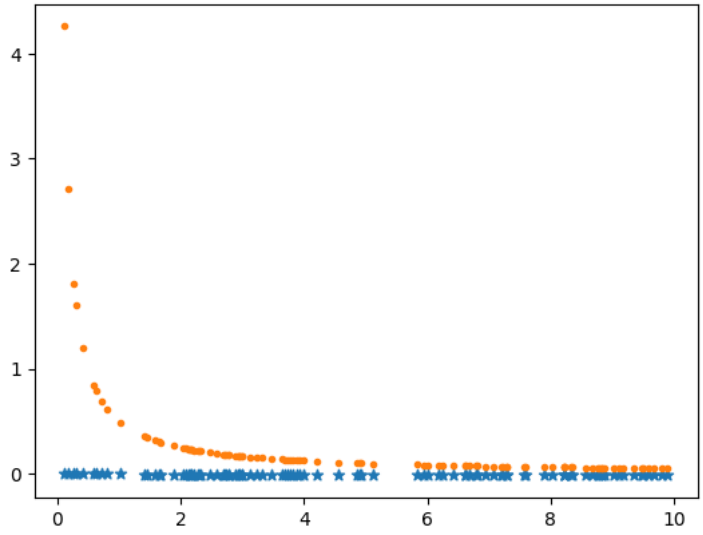
finish----------

activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 50 , learningRate: 5e-06 , minimum_RMSE: 283.59 , epoch: 1898 , test_loss: 23



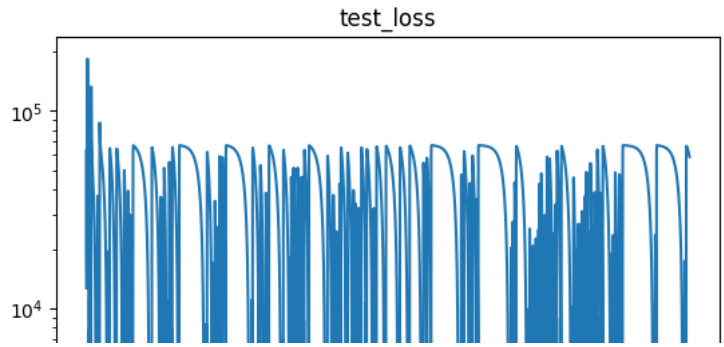


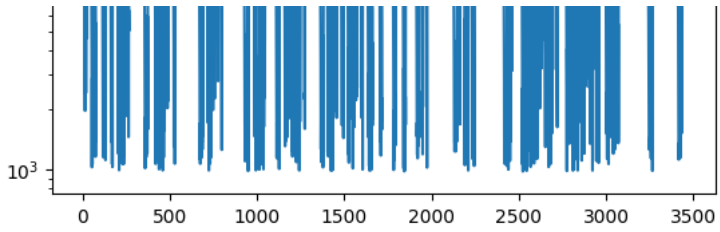


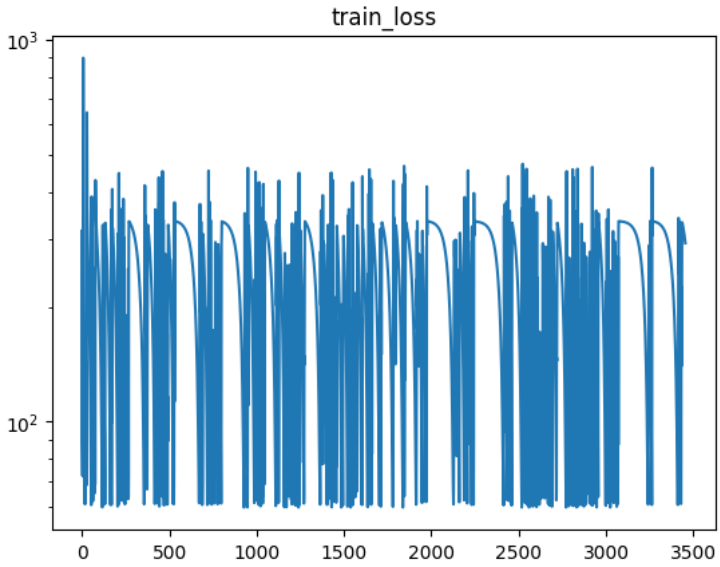


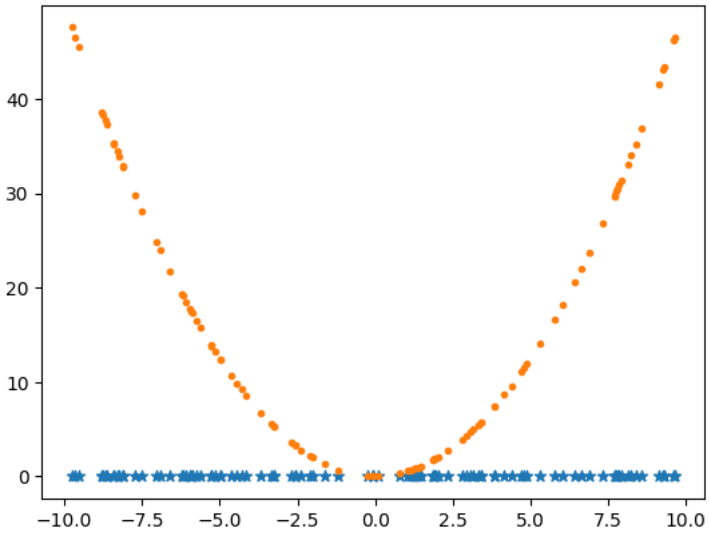
finish----------

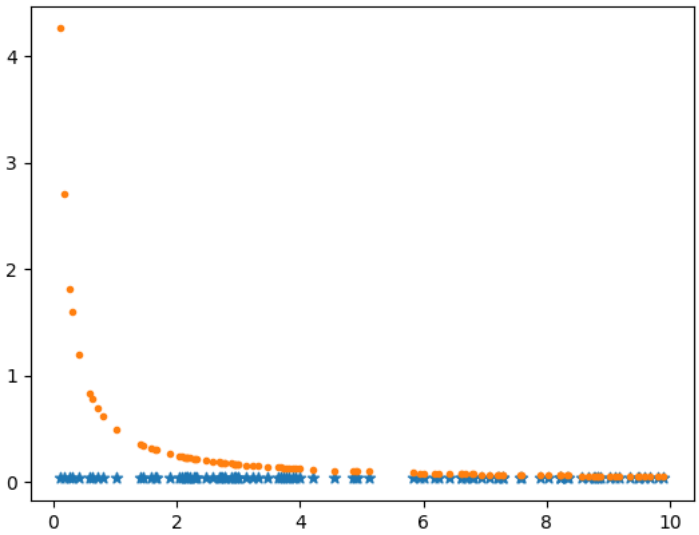
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 100 , learningRate: 0.1 , minimum_RMSE: 975.36 , epoch: 3455 , test_loss: 584





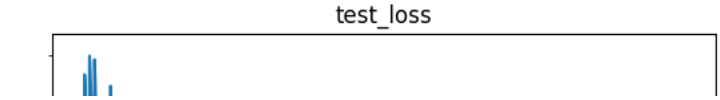


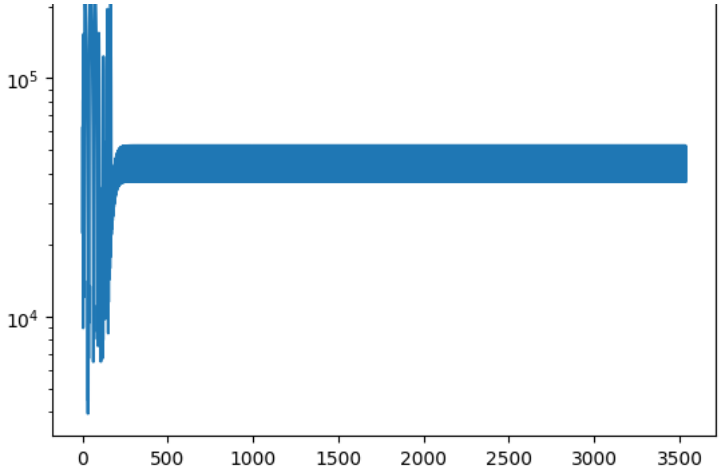


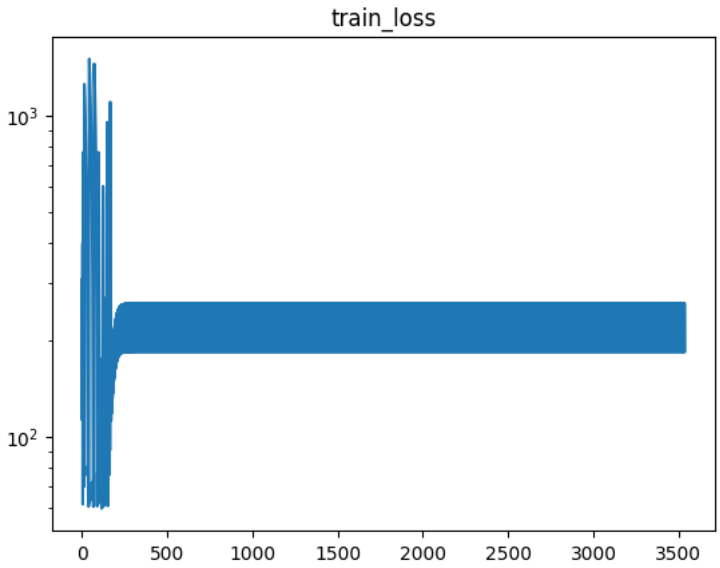


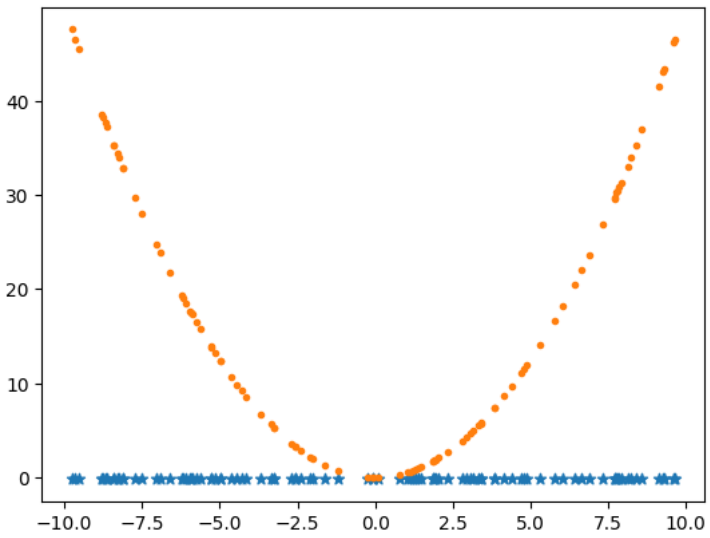
finish----------

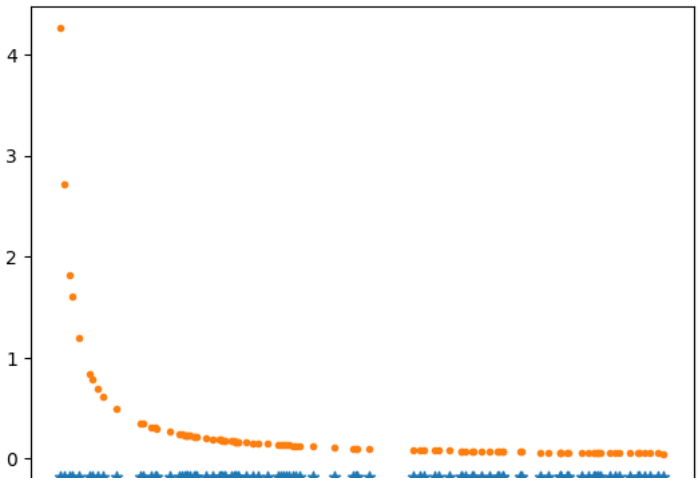
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 100 , learningRate: 0.5 , minimum_RMSE: 3937.06 , epoch: 3536 , test_loss: 36

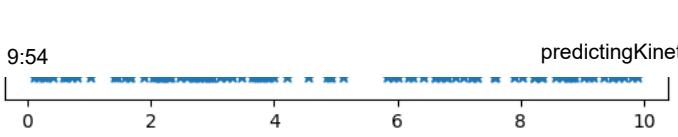



10^5
10^4
0
500
1000
1500
2000
2500
3000
3500


train_loss
10^3
10^2
0
500
1000
1500
2000
2500
3000
3500

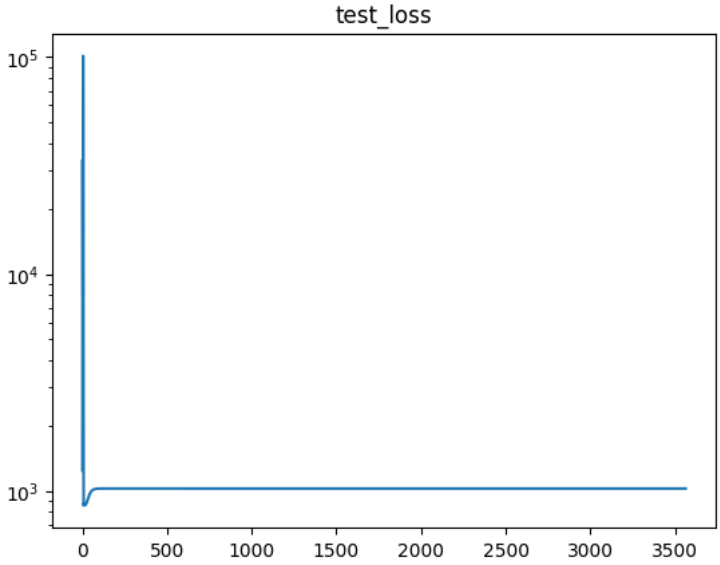

40
30
20
10
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

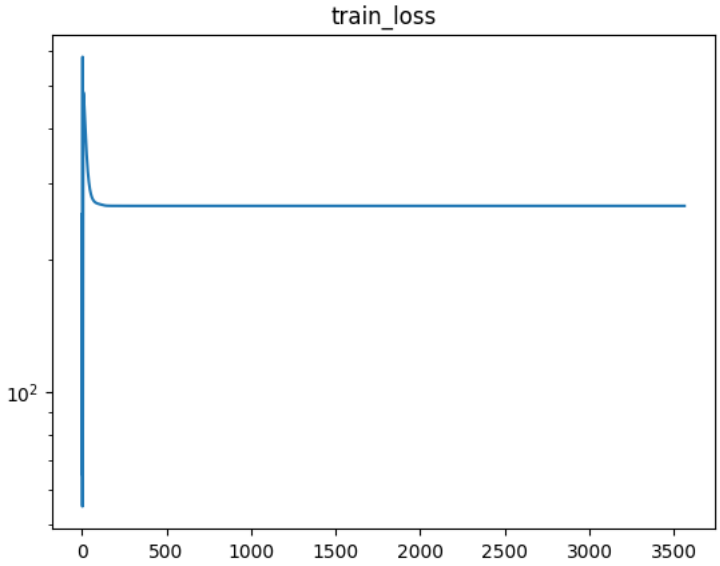

4
3
2
1
0

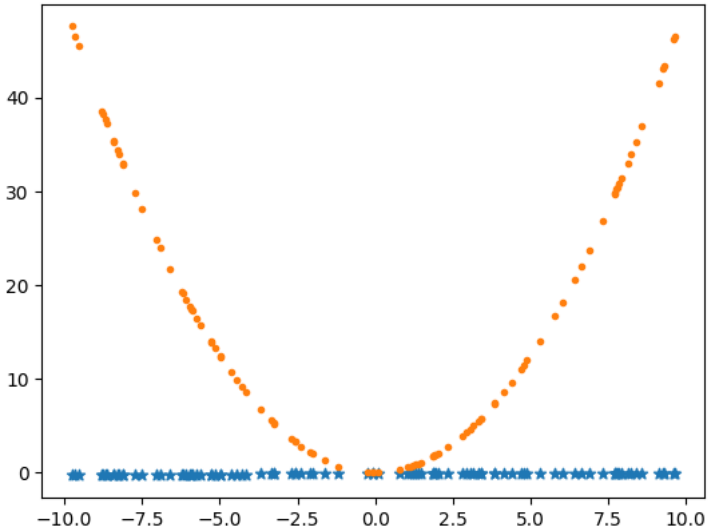


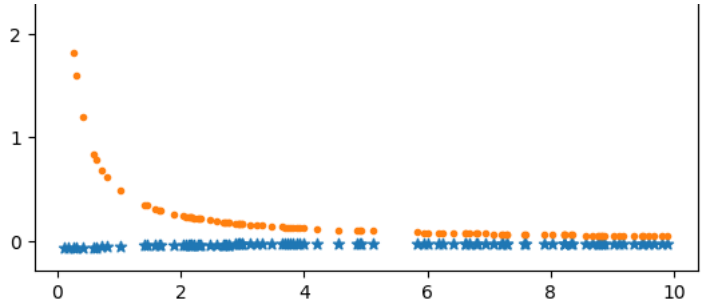
finish----------

activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 100 , learningRate: 0.01 , minimum_RMSE: 858.36 , epoch: 3563 , test_loss: 10



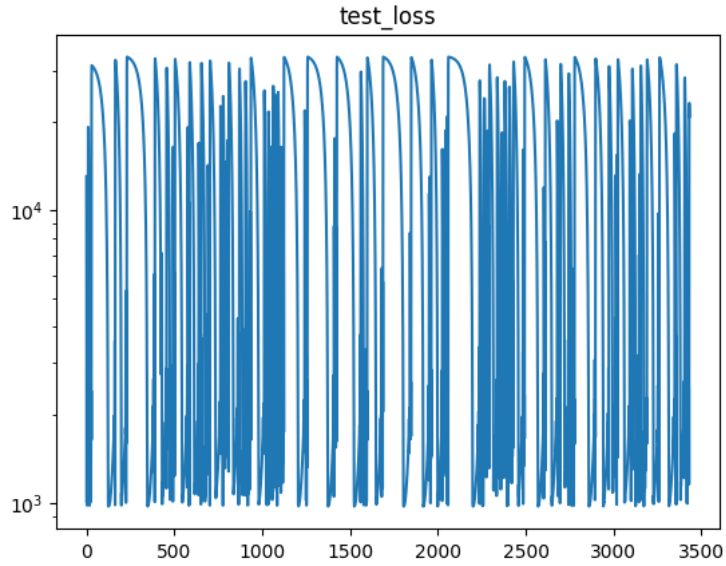


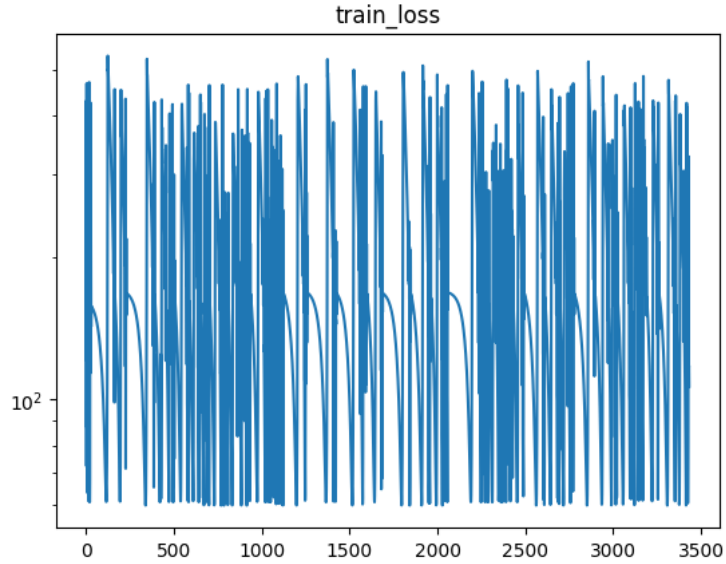


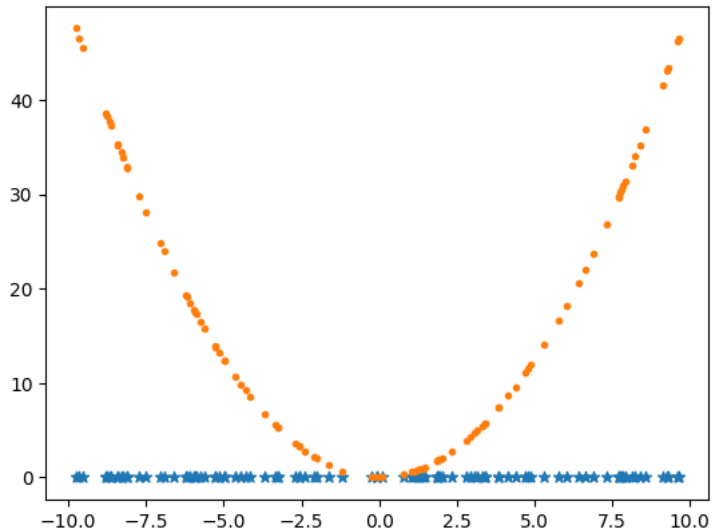


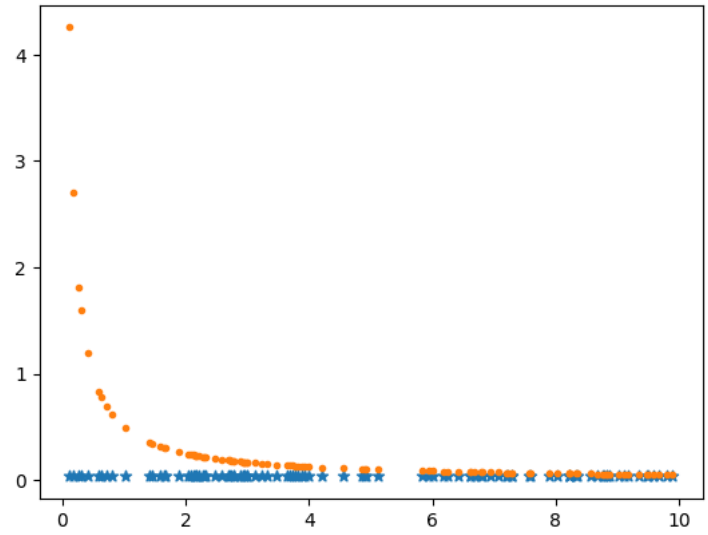
finish----------

activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 100 , learningRate: 0.05 , minimum_RMSE: 971.75 , epoch: 3432 , test_loss: 20



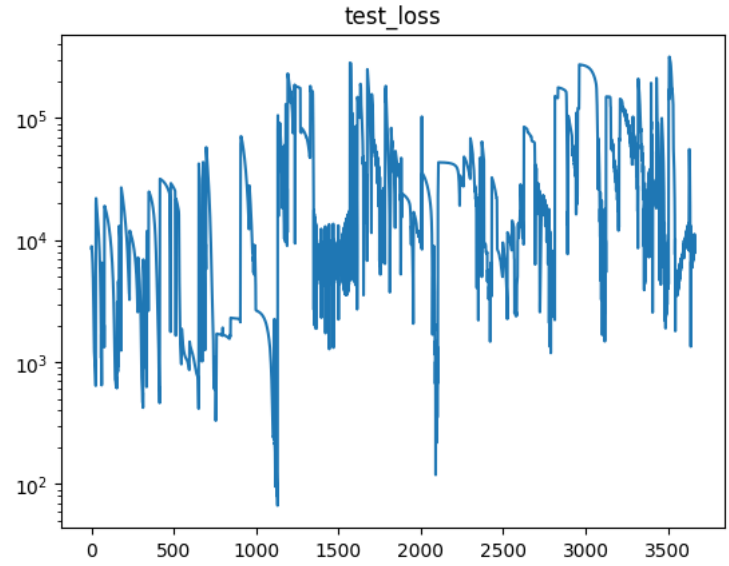


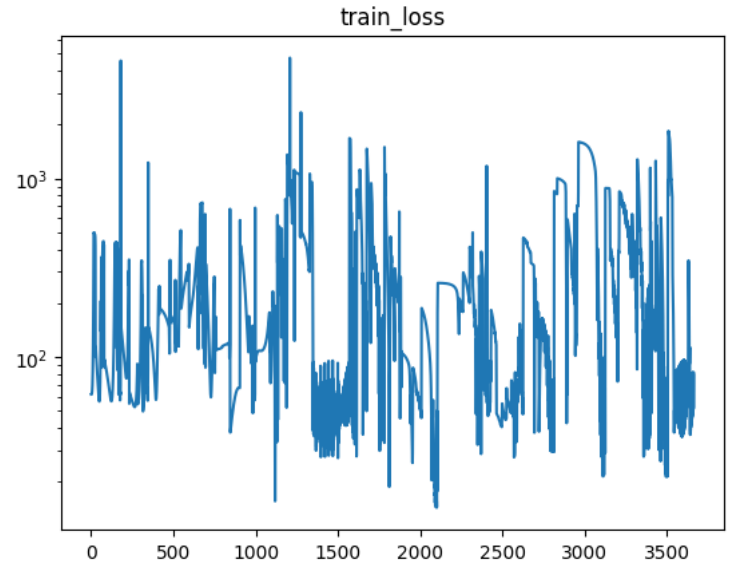


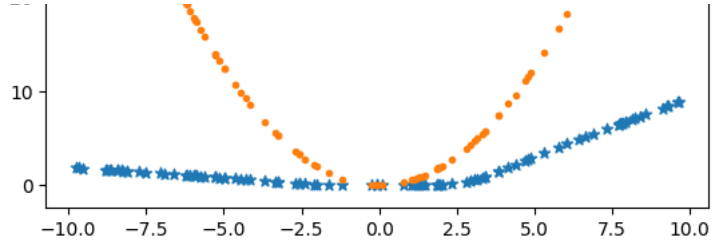


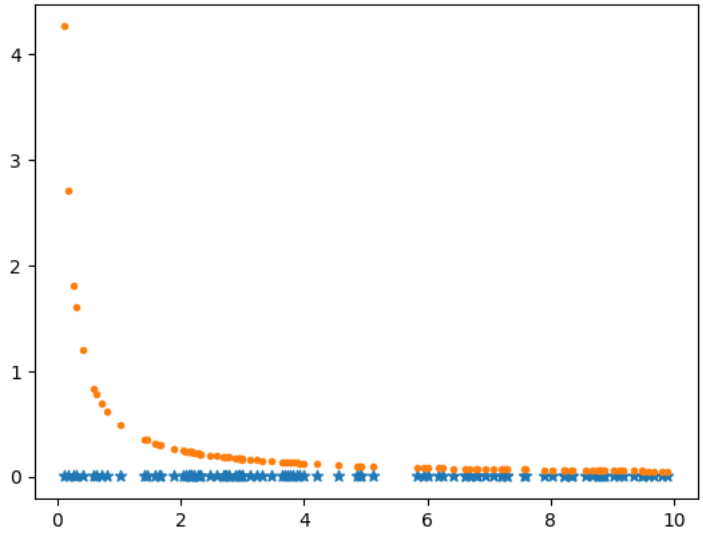
finish----------

activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 100 , learningRate: 0.001 , minimum_RMSE: 66.68 , epoch: 3662 , test_loss: 79



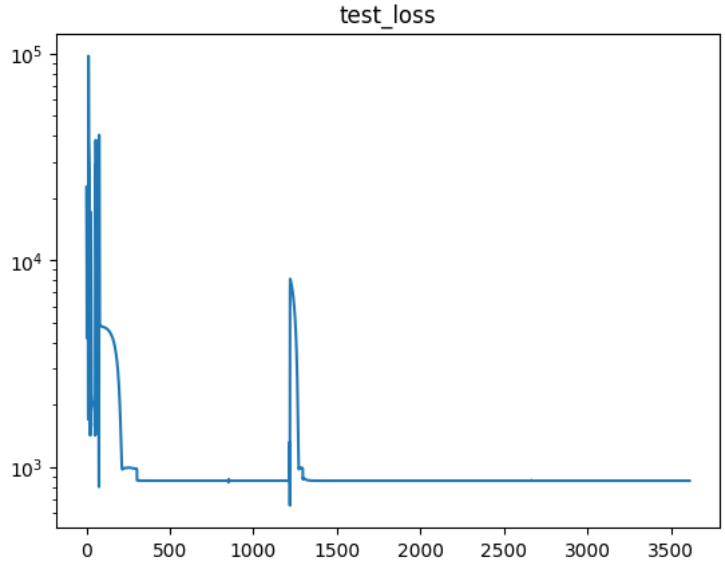


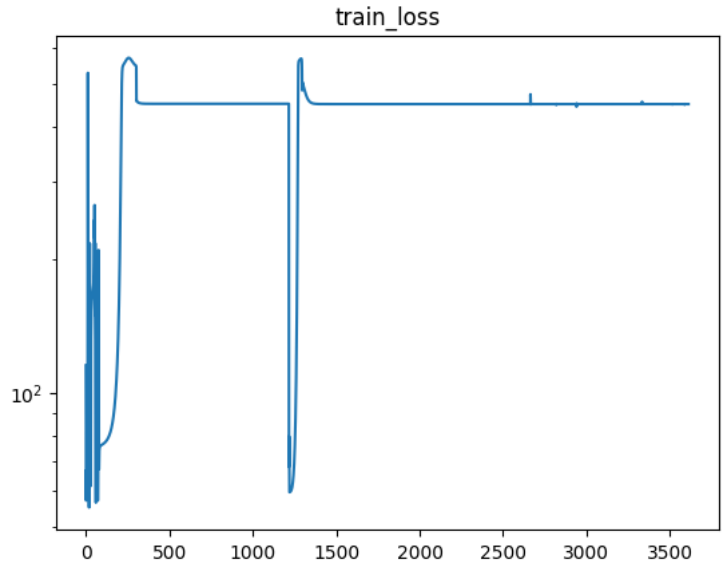


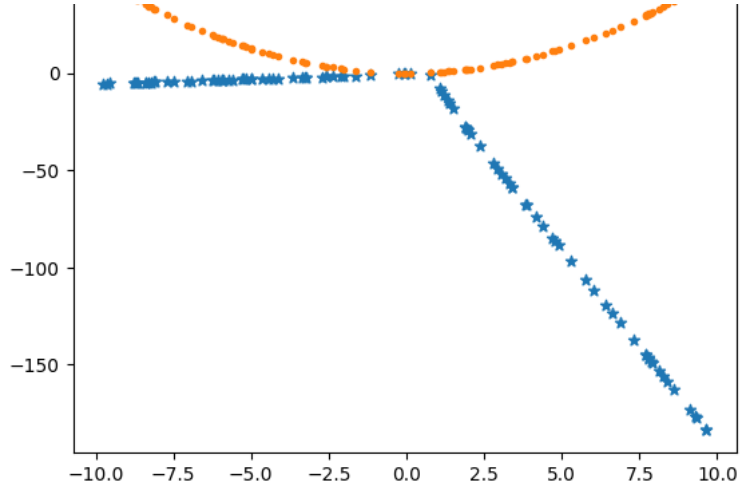


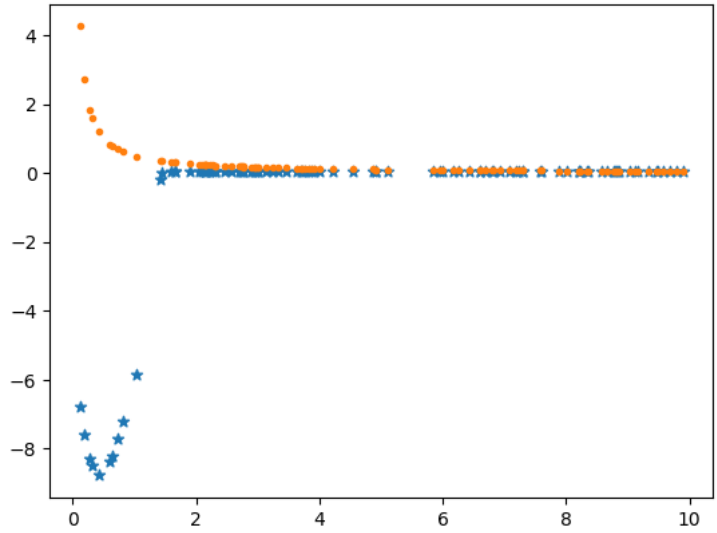
finish----------

activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 100 , learningRate: 0.005 , minimum_RMSE: 650.98 , epoch: 3611 , test_loss: 8



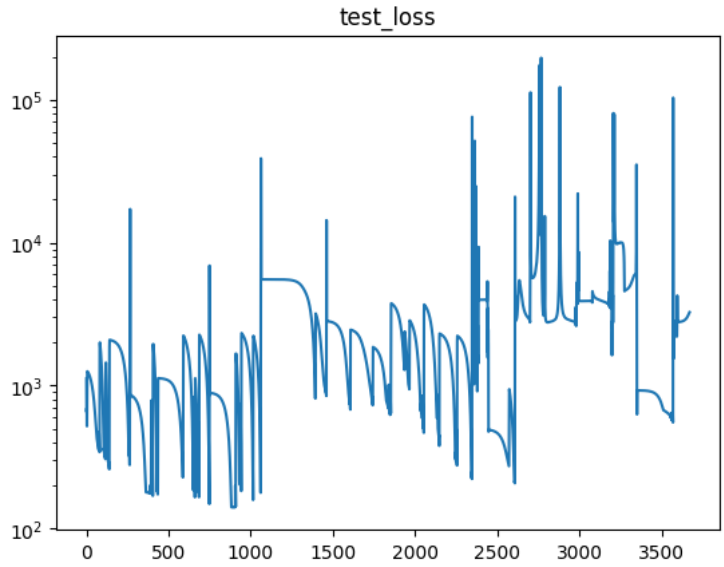


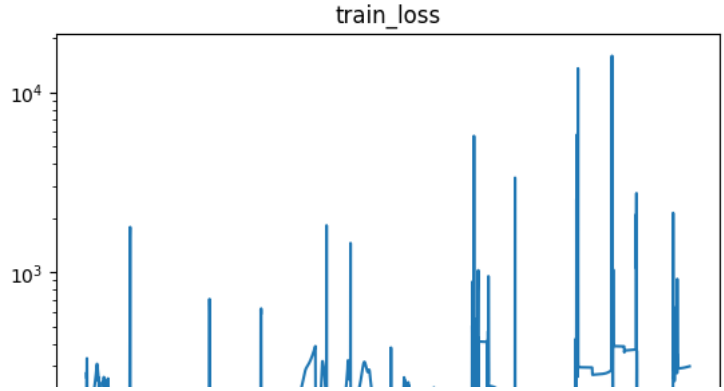


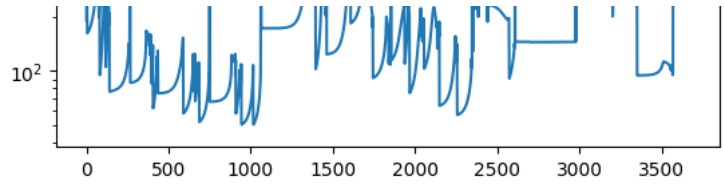


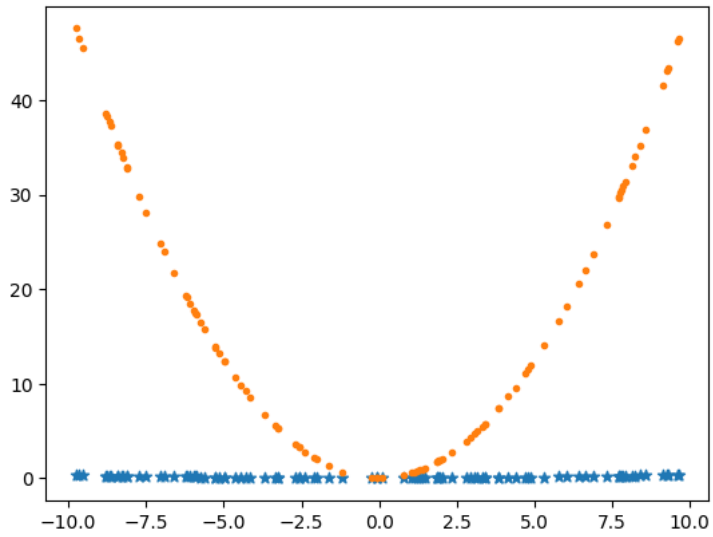
finish----------

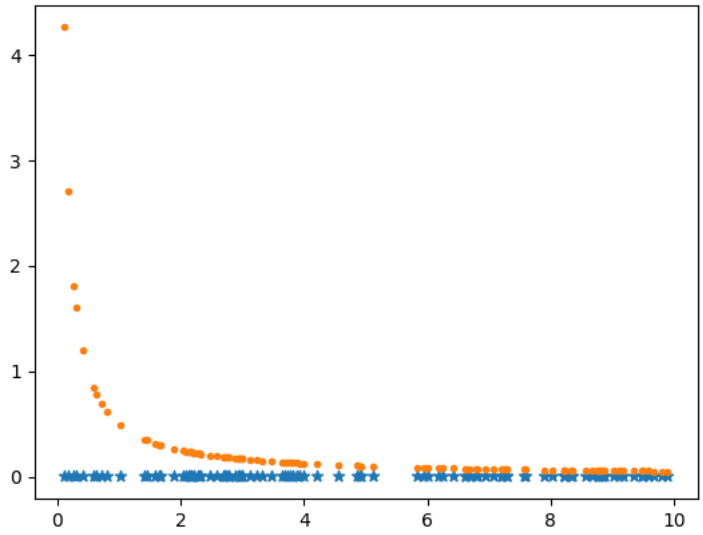
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 100 , learningRate: 0.0001 , minimum_RMSE: 140.26 , epoch: 3670 , test_loss:





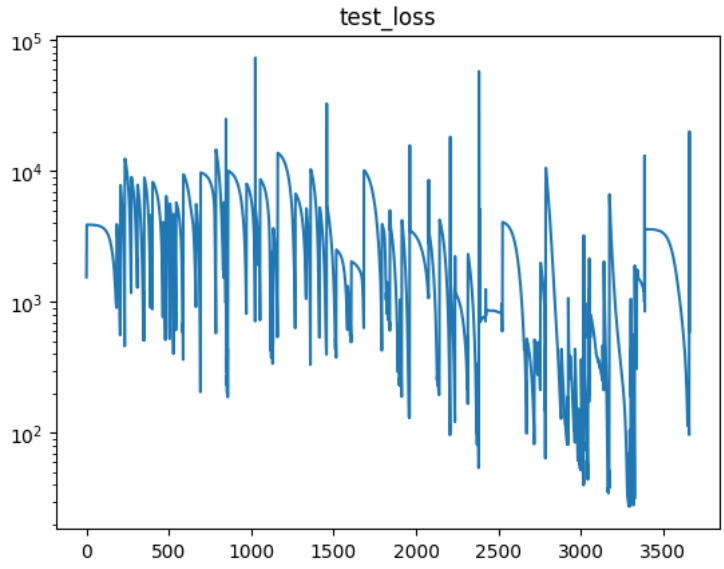


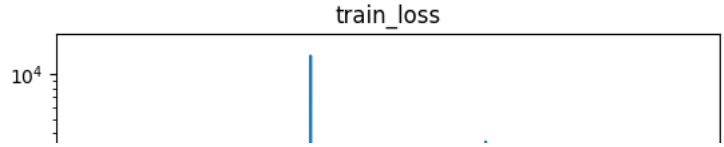


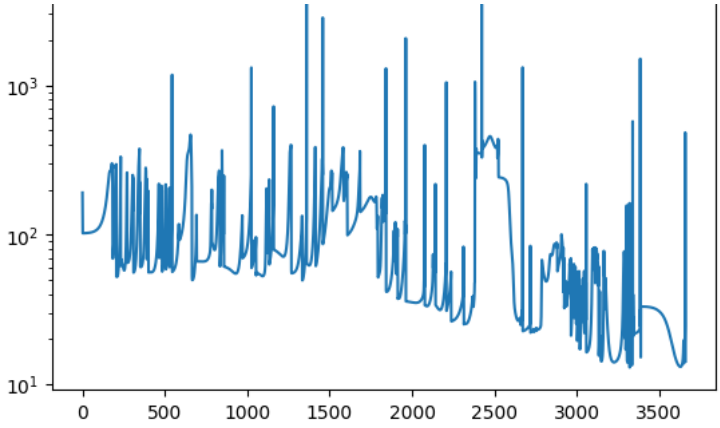


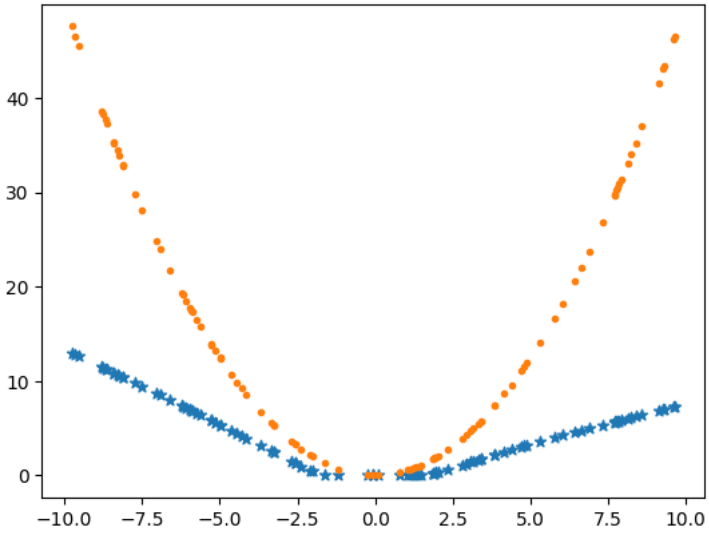
finish----------

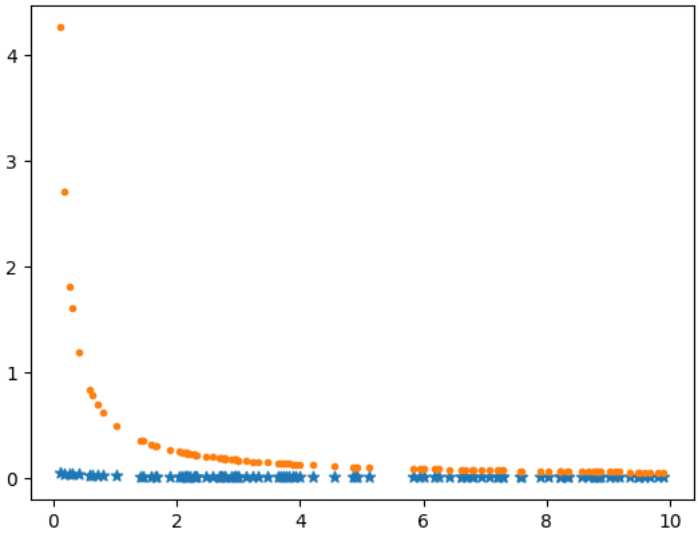
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 100 , learningRate: 0.0005 , minimum_RMSE: 27.47 , epoch: 3660 , test_loss: 5





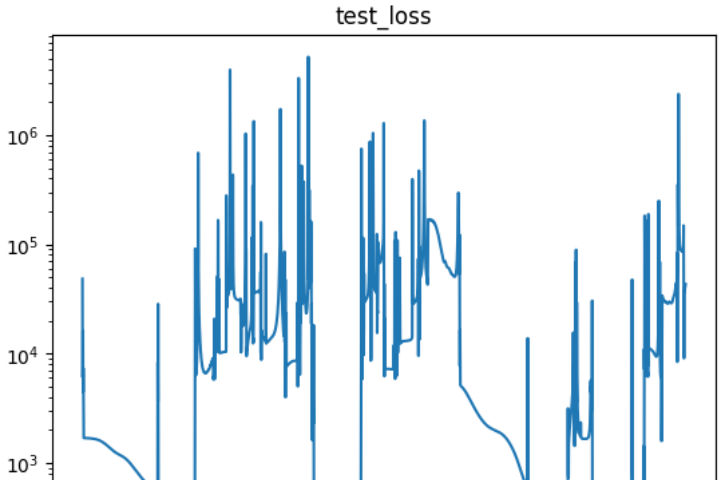


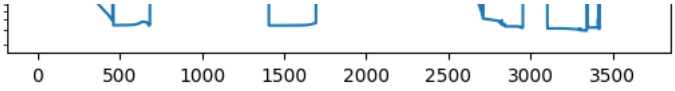


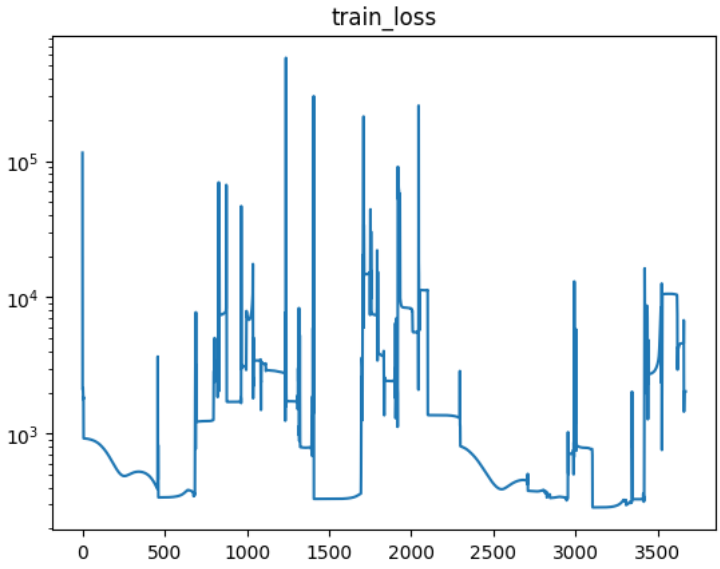


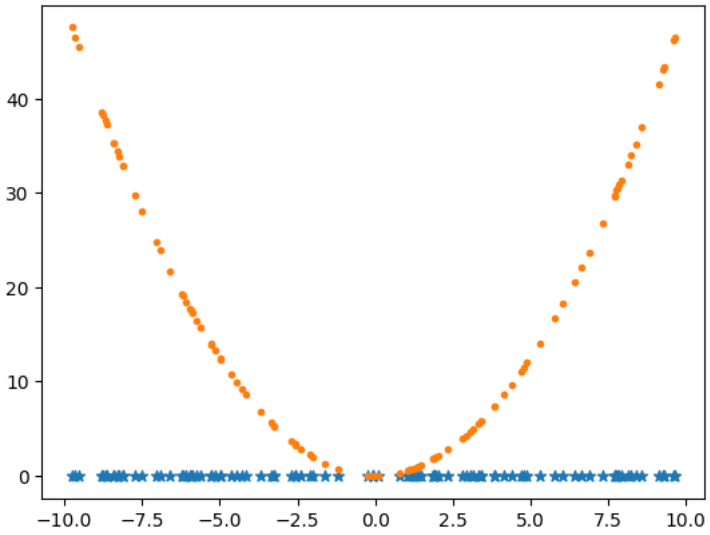
finish----------

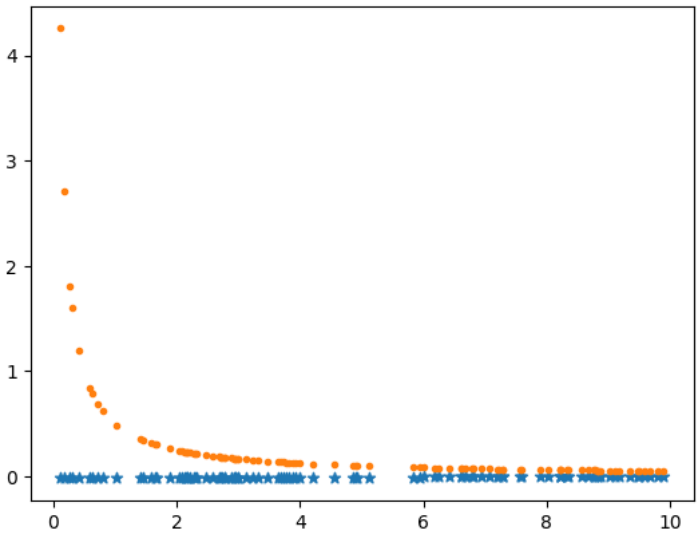
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 100 , learningRate: 1e-05 , minimum_RMSE: 398.66 , epoch: 3669 , test_loss: 4





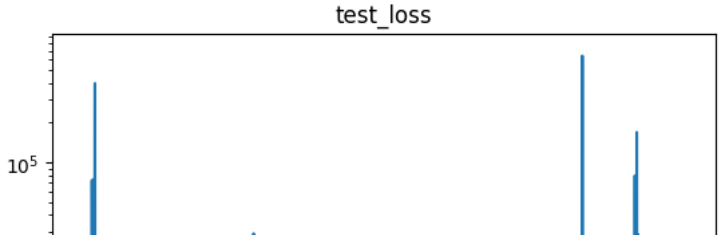


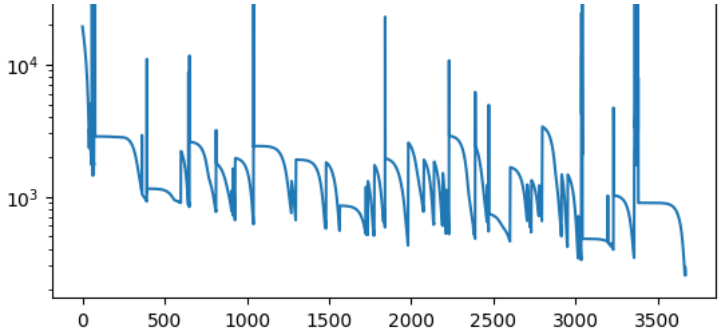


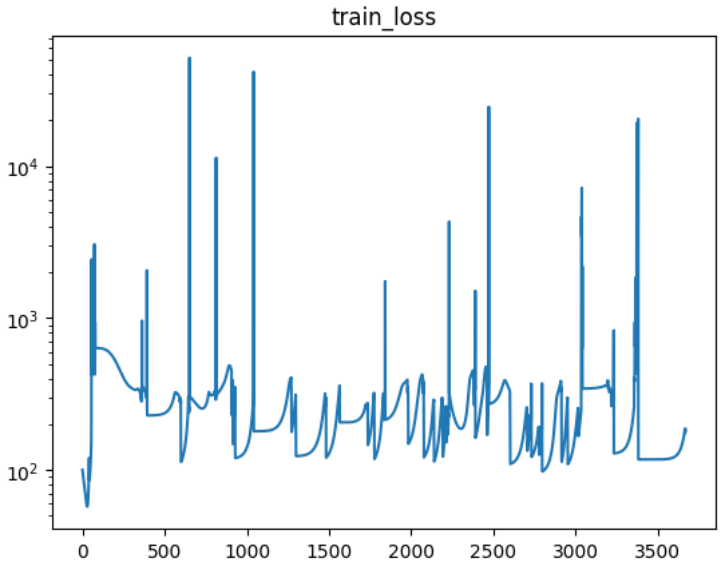


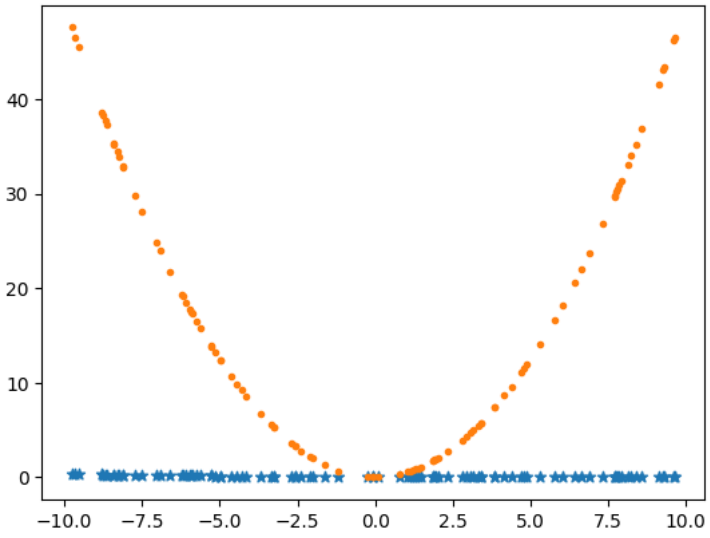
finish----------

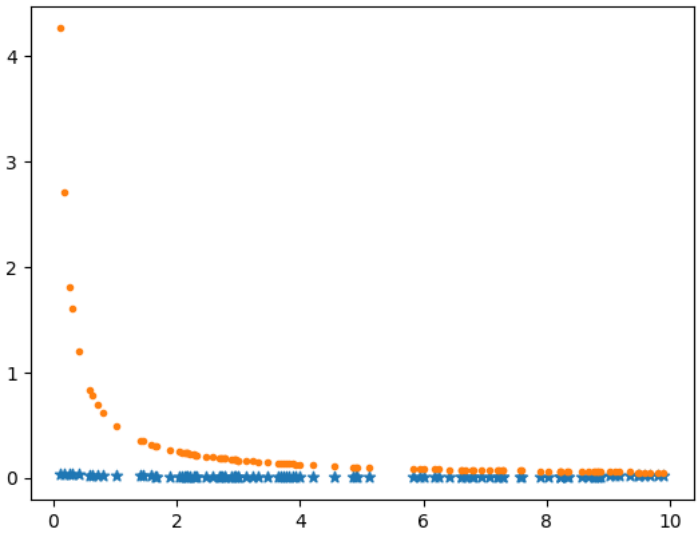
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 100 , learningRate: 5e-05 , minimum_RMSE: 252.92 , epoch: 3668 , test_loss: 2





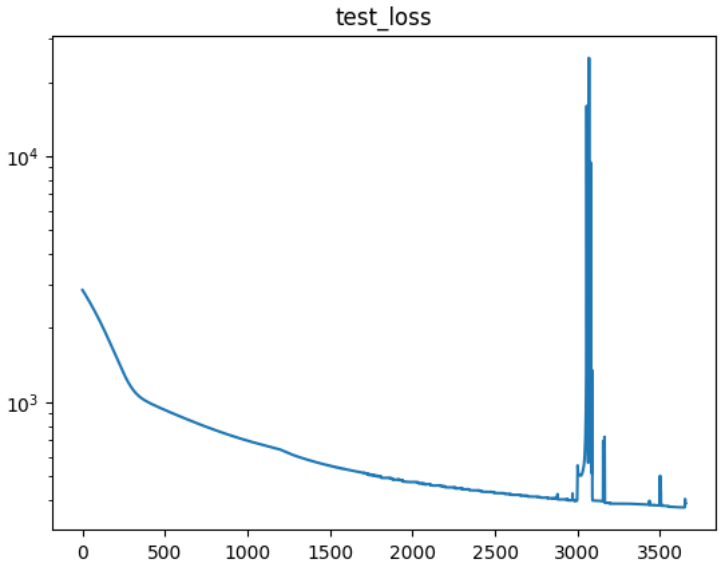


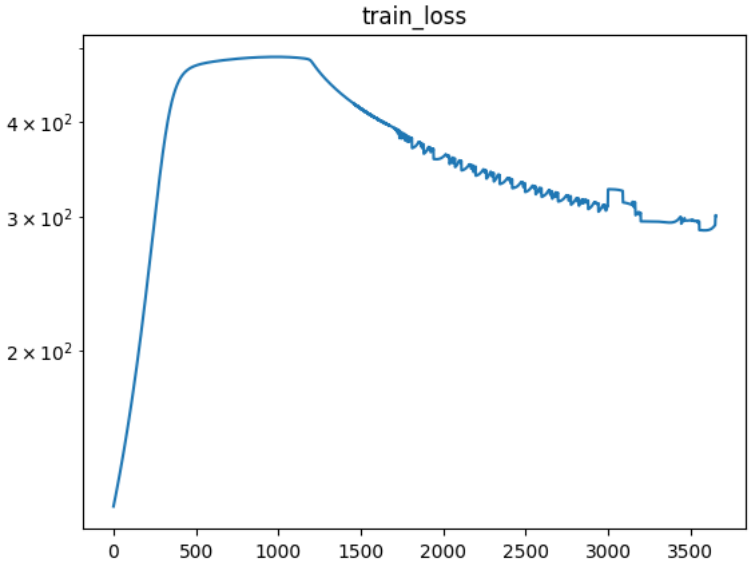


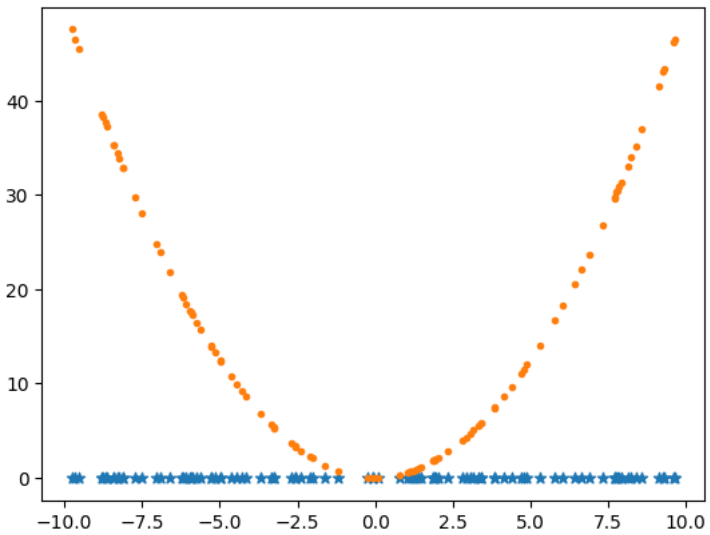


finish----------

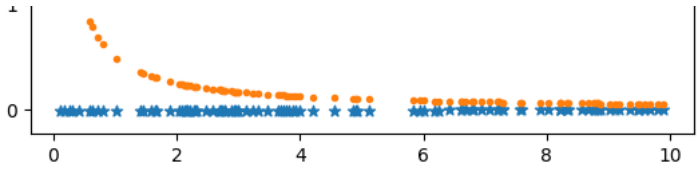
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 100 , learningRate: 1e-06 , minimum_RMSE: 373.60 , epoch: 3655 , test_loss: 3


test_loss
10^4
10^3
0
500
1000
1500
2000
2500
3000
3500


train_loss
4×10^2
3×10^2
2×10^2
0
500
1000
1500
2000
2500
3000
3500

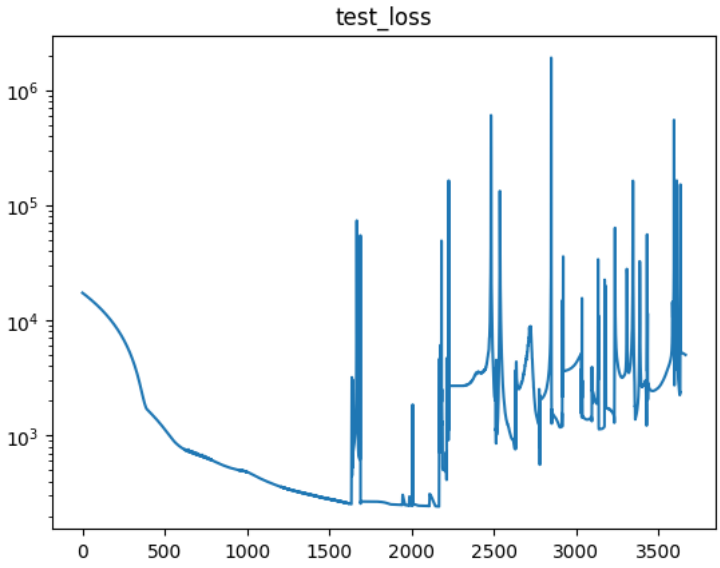

40
30
20
10
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

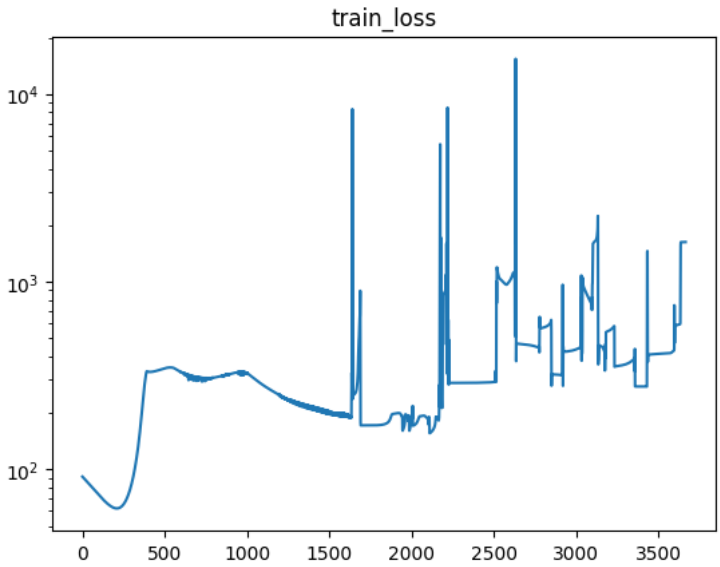

4
3
2

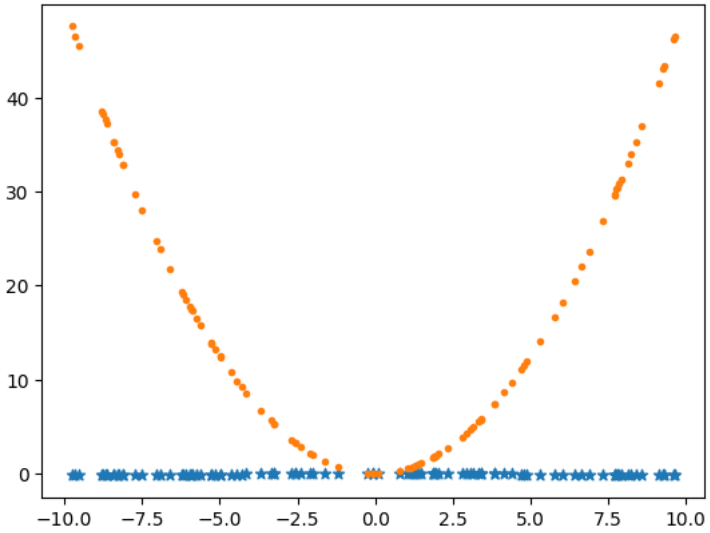


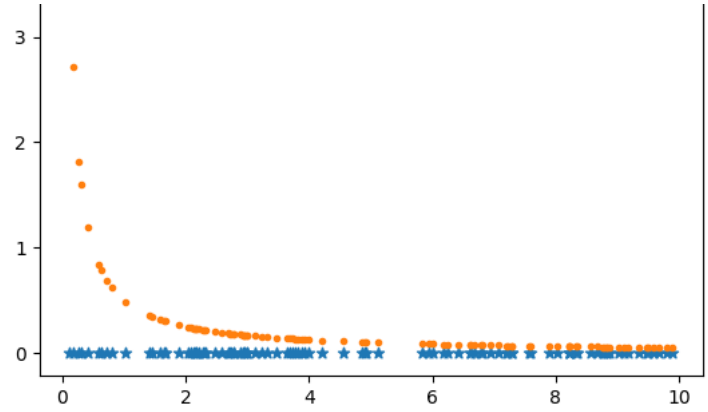
finish----------

activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 100 , learningRate: 5e-06 , minimum_RMSE: 241.33 , epoch: 3667 , test_loss: 5



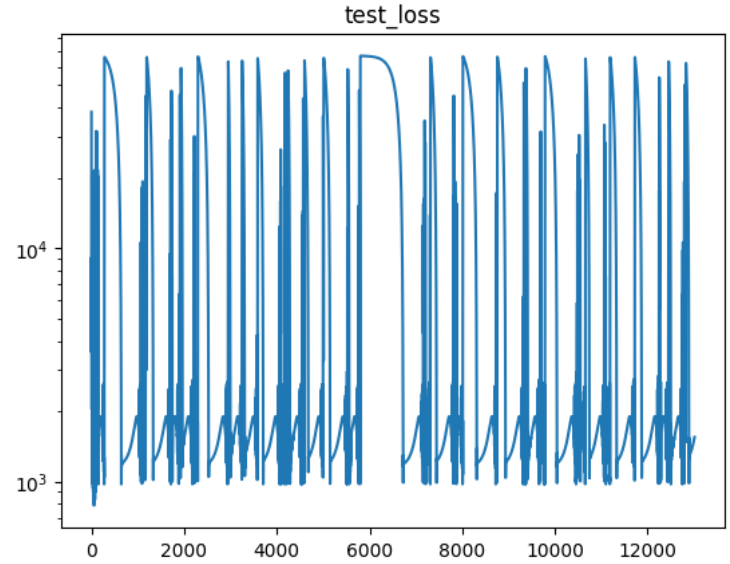


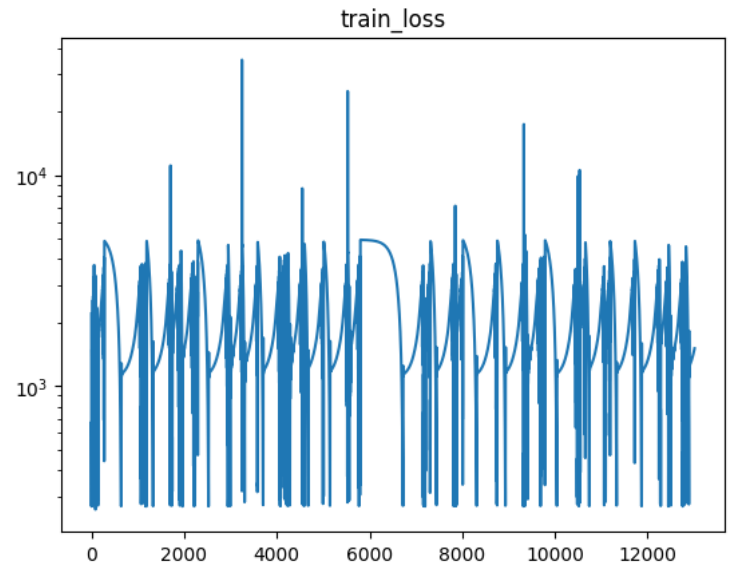


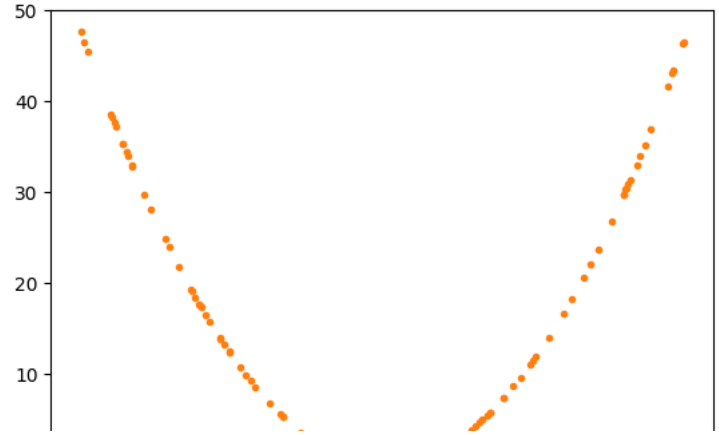


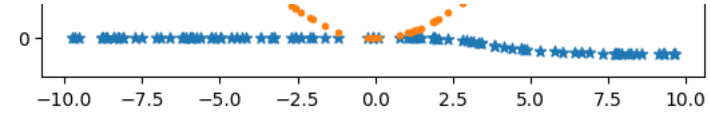
finish----------

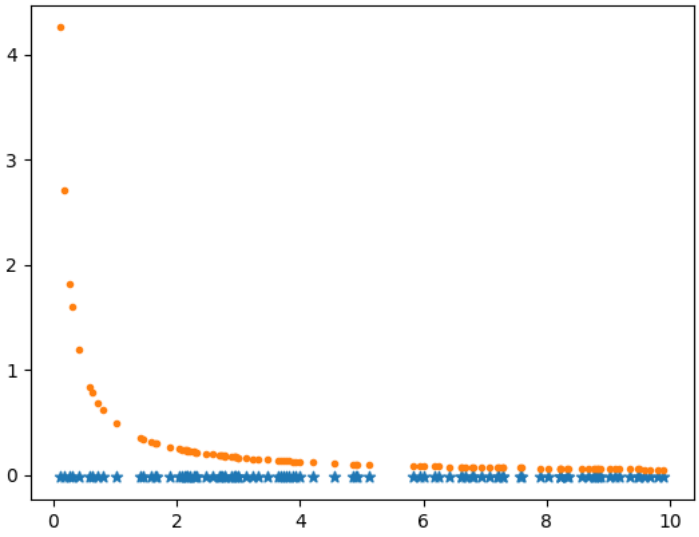
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 500 , learningRate: 0.1 , minimum_RMSE: 788.63 , epoch: 13012 , test_loss: 154





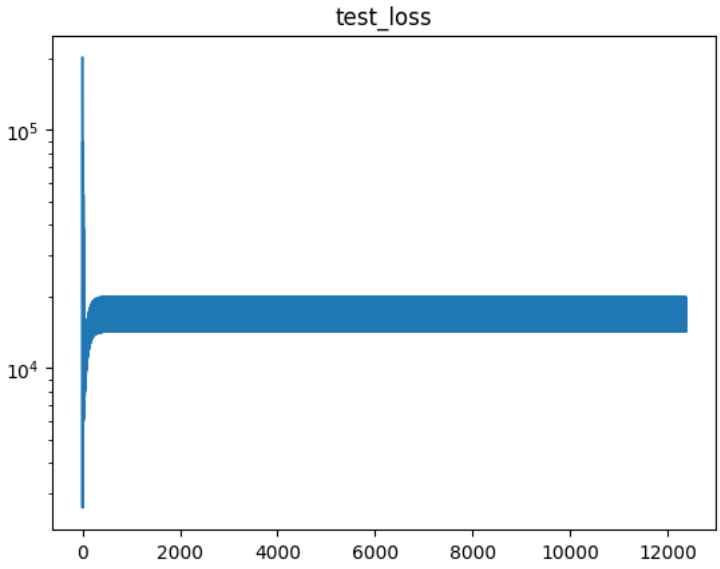


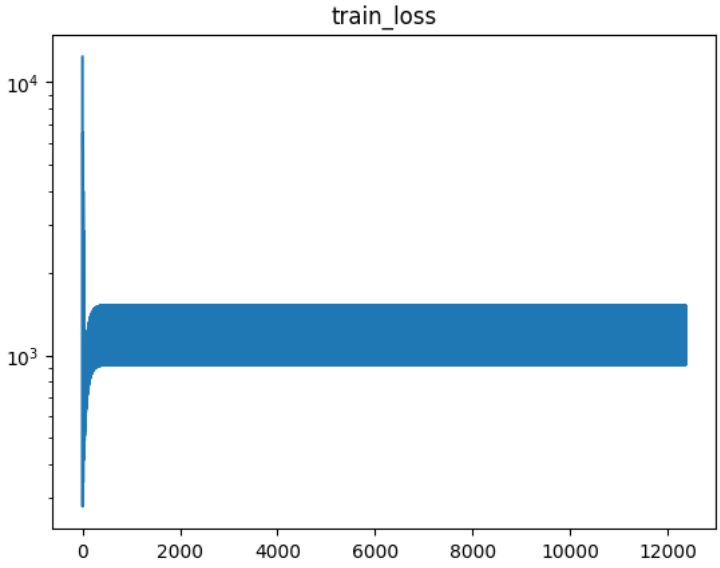


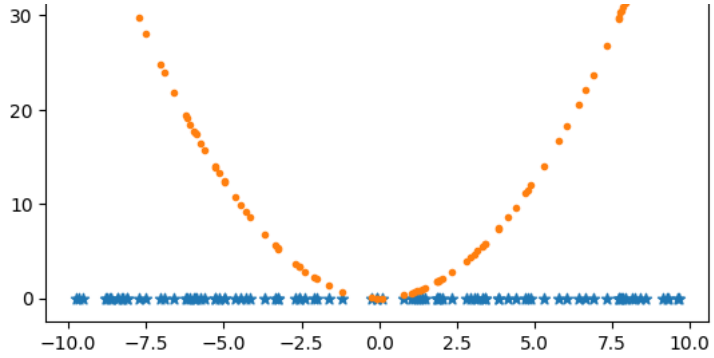


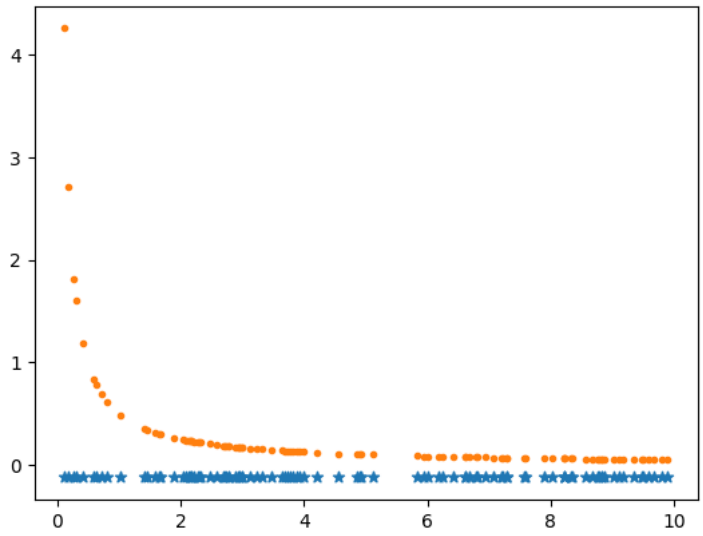
finish----------

activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 500 , learningRate: 0.5 , minimum_RMSE: 2607.01 , epoch: 12365 , test_loss: 14



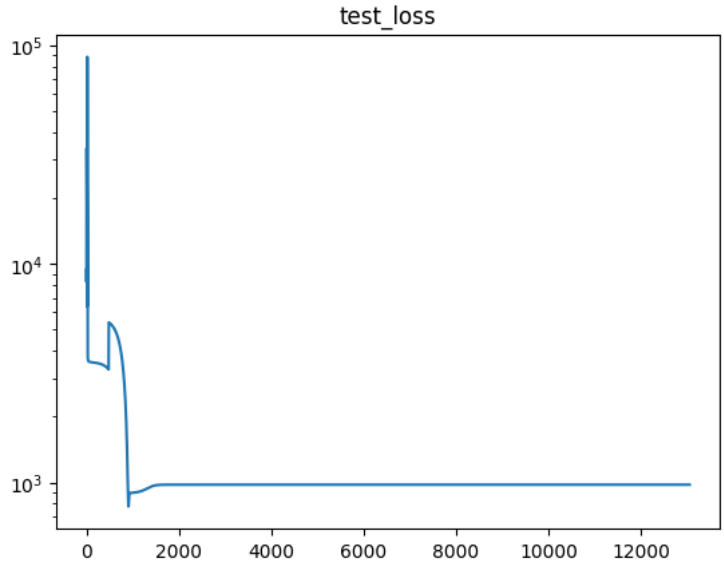


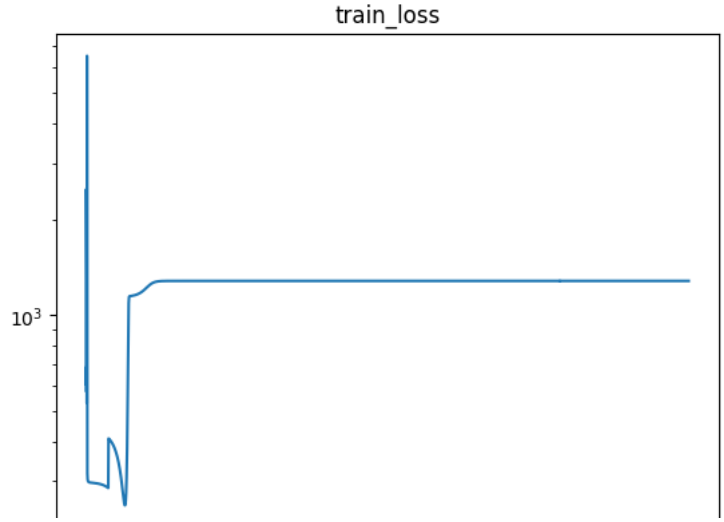


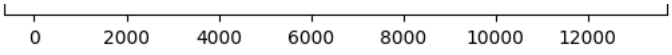


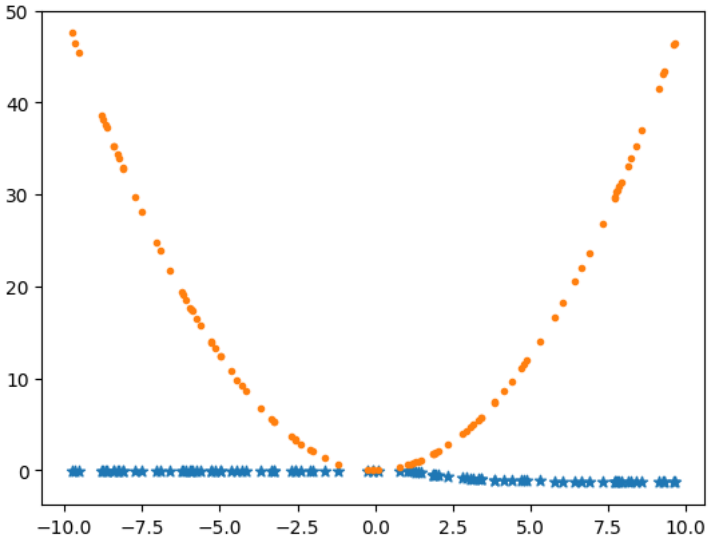
finish----------

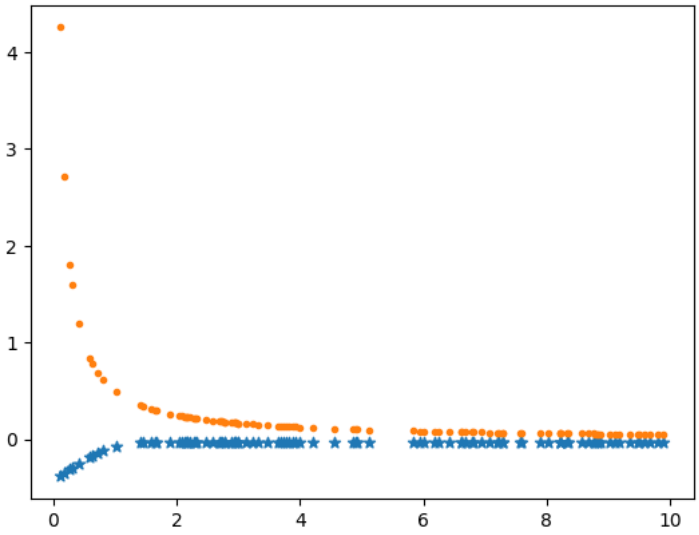
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 500 , learningRate: 0.01 , minimum_RMSE: 776.44 , epoch: 13057 , test_loss: 97





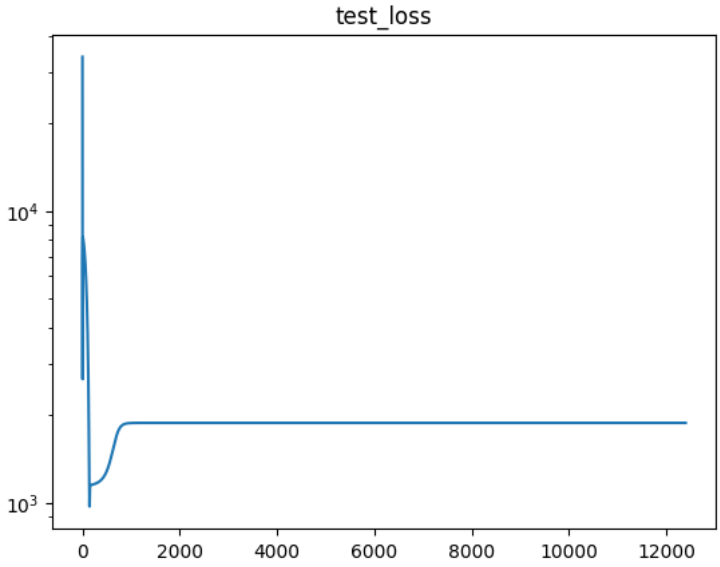


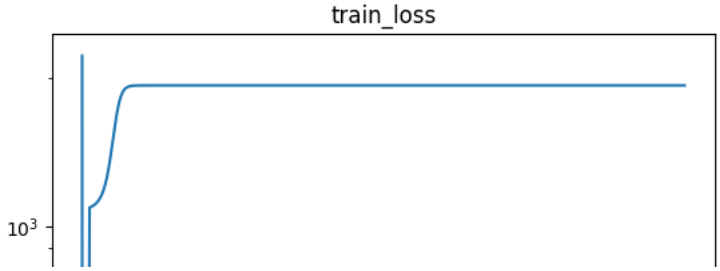


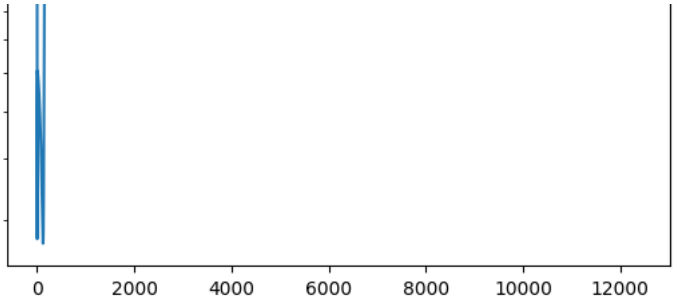


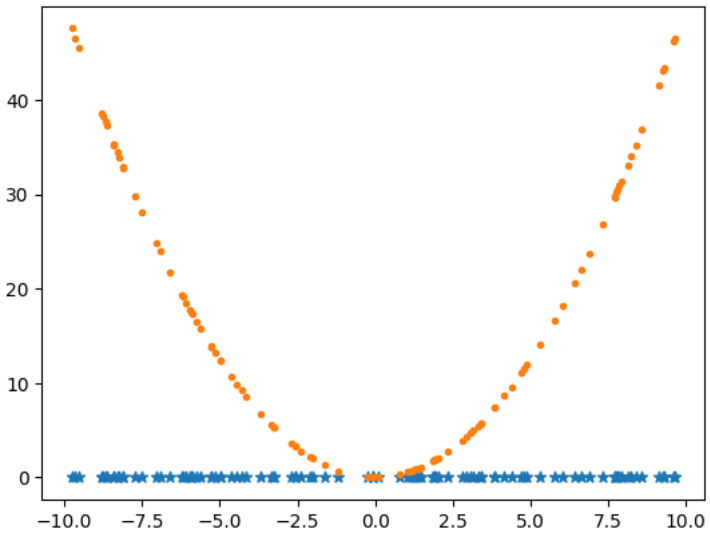

```
finish----------------------------------------------------------------------------------------------------------------------------------------------------------------
```

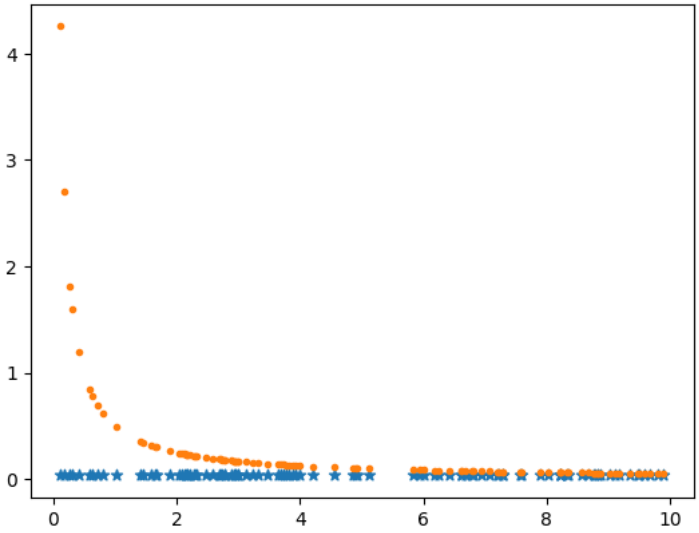
```
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 500 , learningRate: 0.05 , minimum_RMSE: 971.68 , epoch: 12401 , test_loss: 18
```





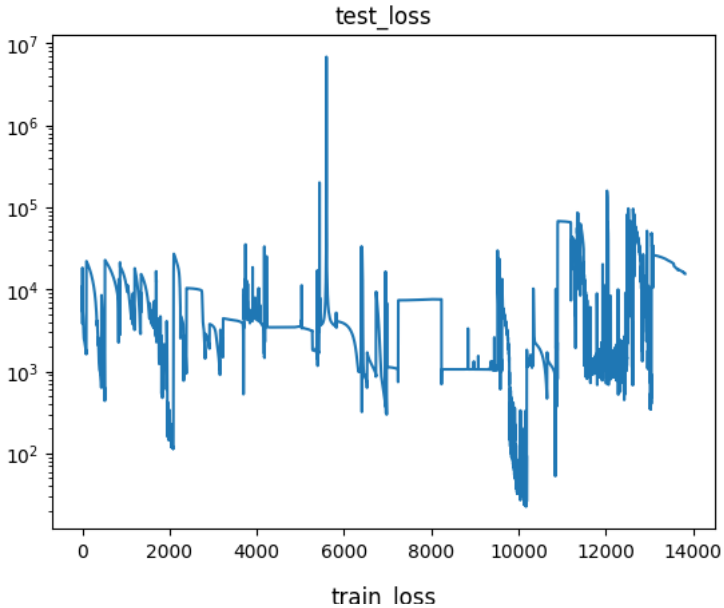


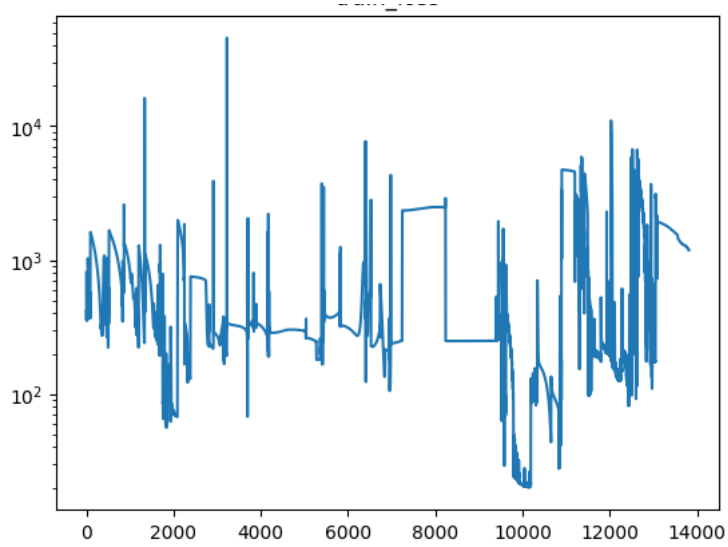


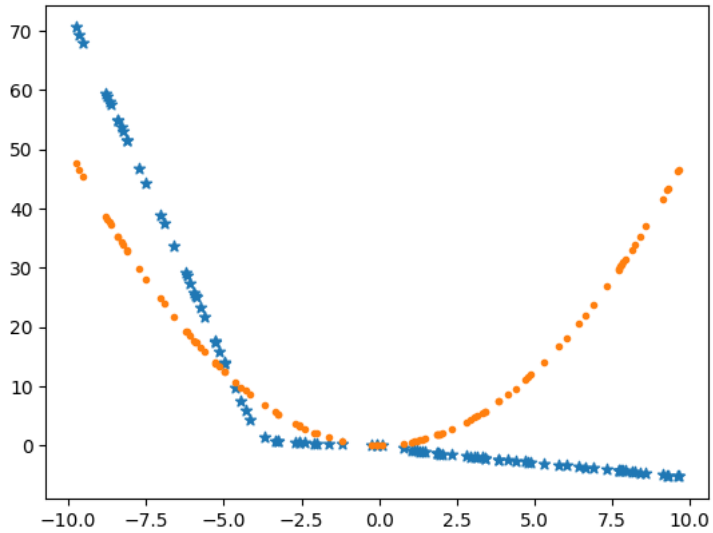


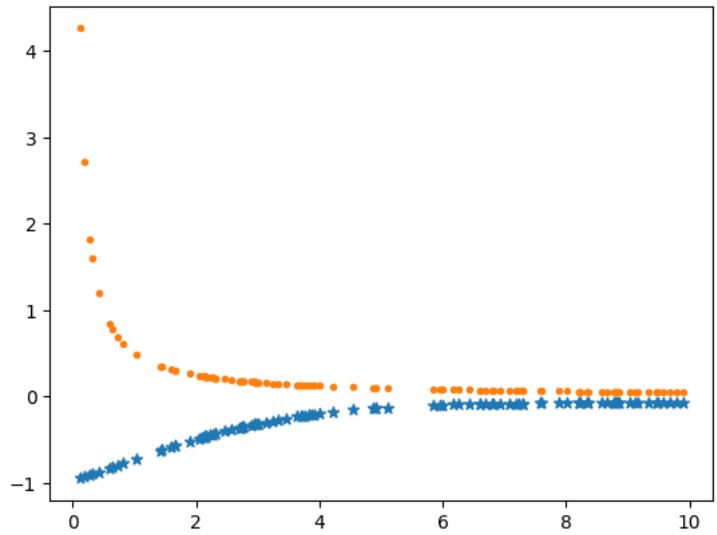
finish----------

activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 500 , learningRate: 0.001 , minimum_RMSE: 22.67 , epoch: 13813 , test_loss: 15



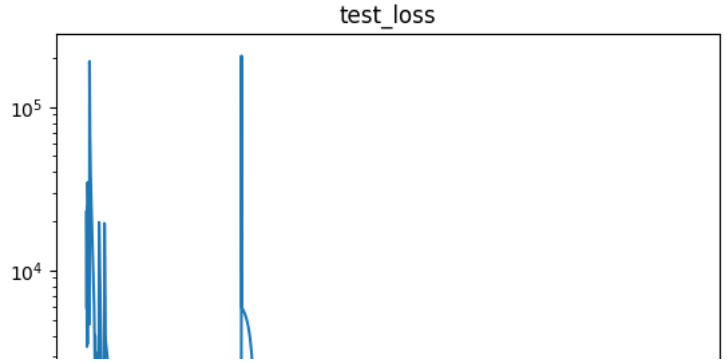


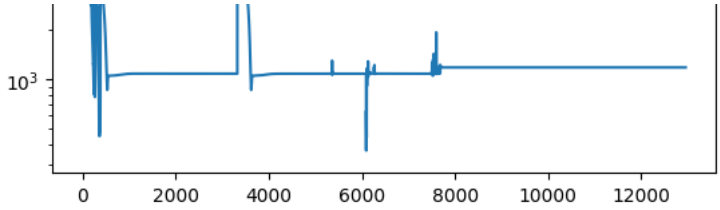


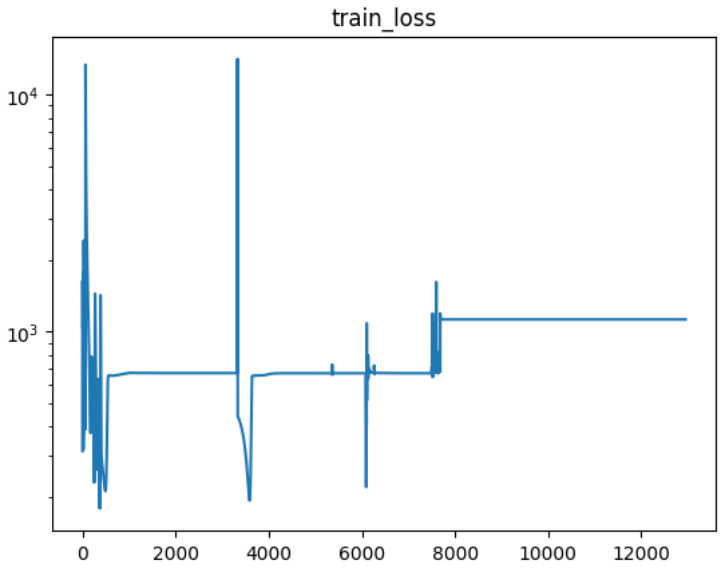


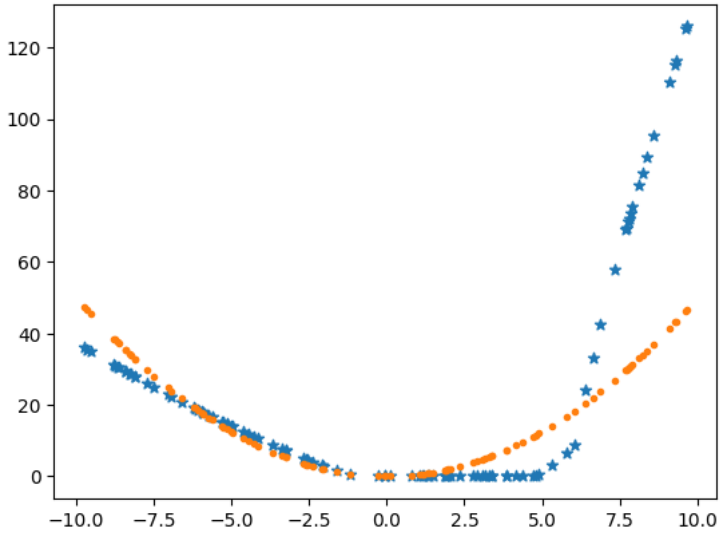

```
finish----------------------------------------------------------------------------------------------------------------------------------------------------------------
```

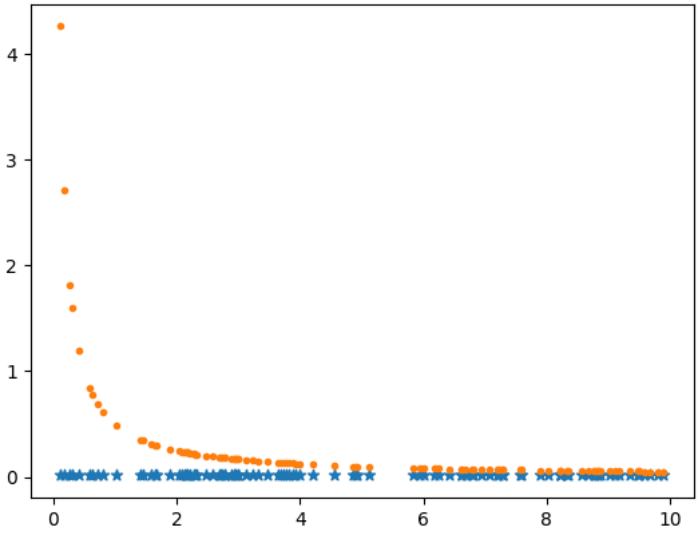
```
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 500 , learningRate: 0.005 , minimum_RMSE: 365.54 , epoch: 12951 , test_loss: 1
```





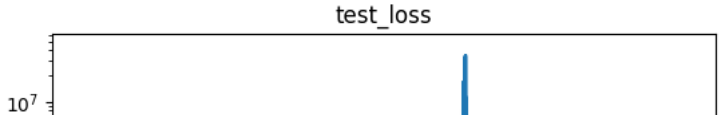


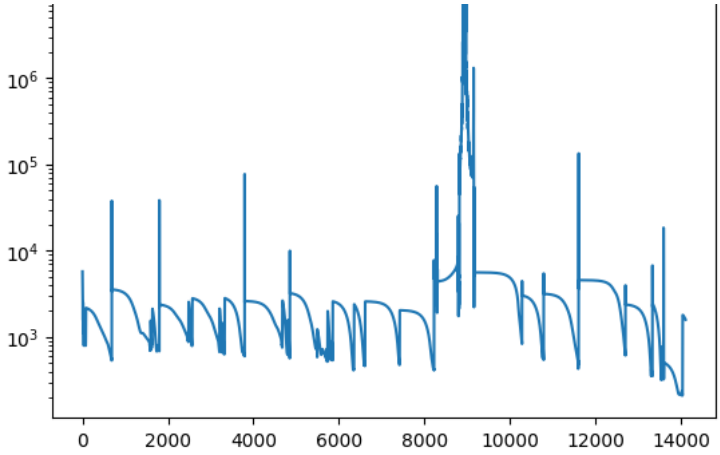


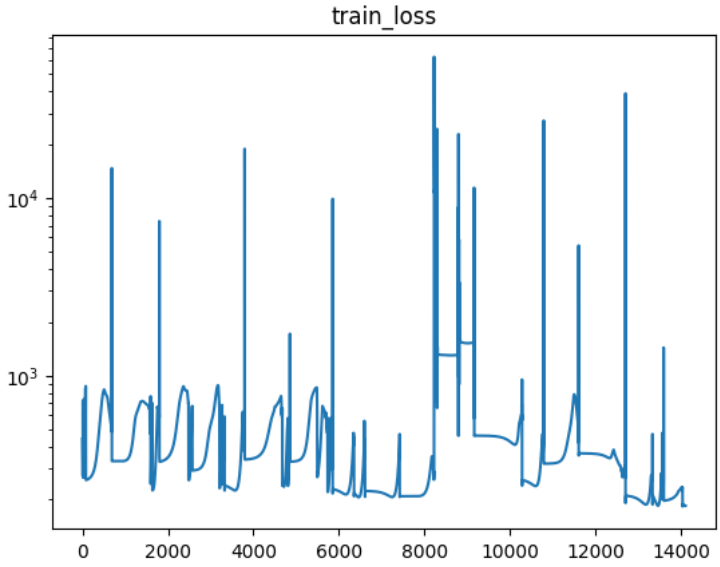


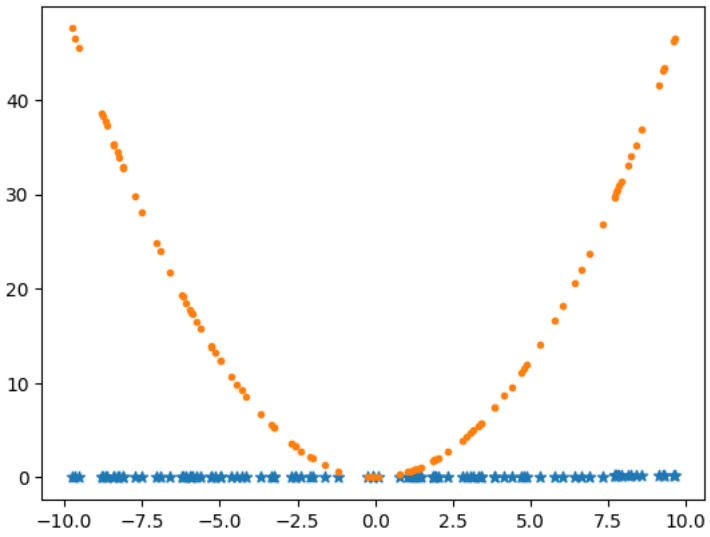
finish----------

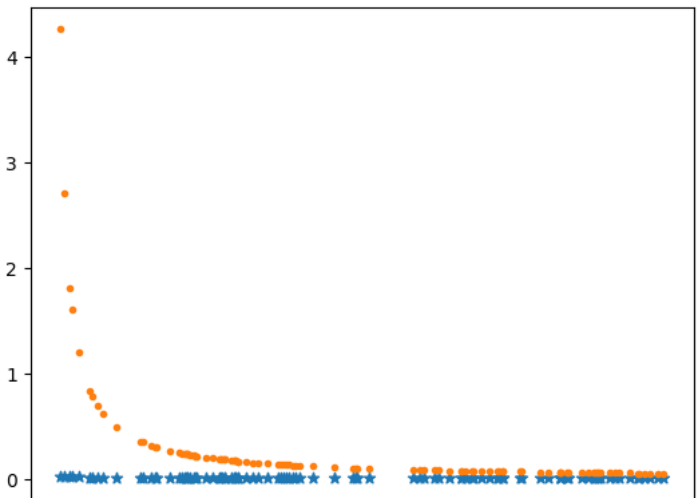
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 500 , learningRate: 0.0001 , minimum_RMSE: 210.54 , epoch: 14111 , test_loss:




10^6
10^5
10^4
10^3
0
2000
4000
6000
8000
10000
12000
14000

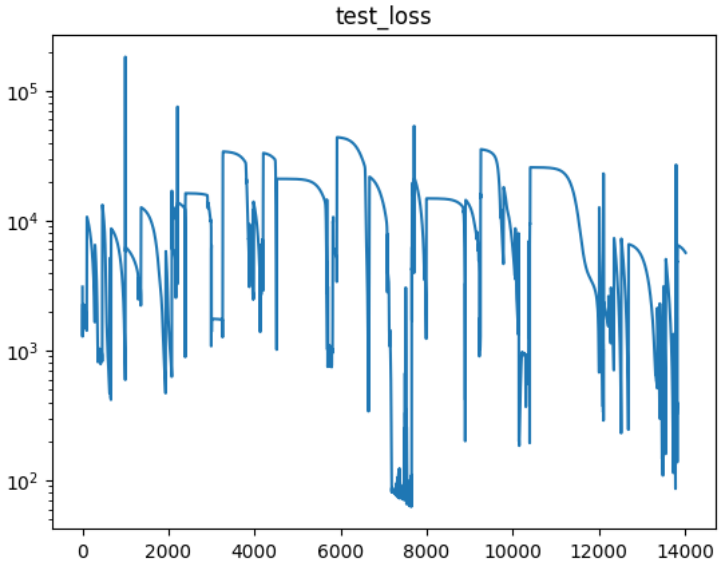

train_loss
10^4
10^3
0
2000
4000
6000
8000
10000
12000
14000

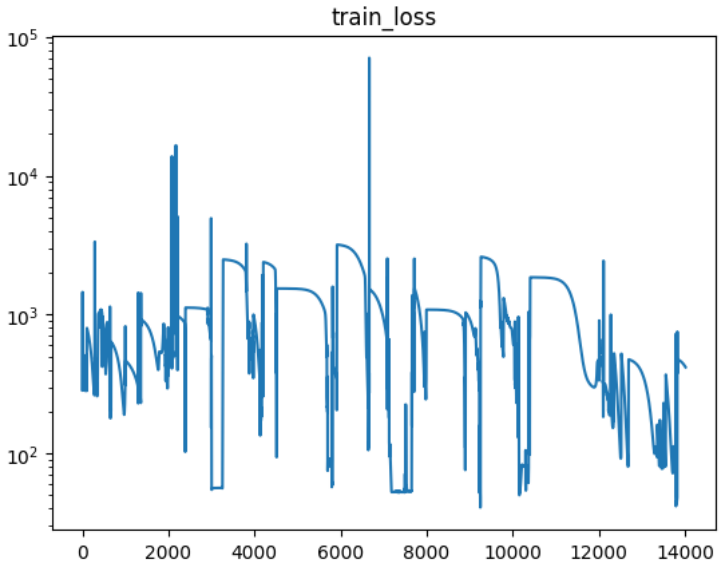

40
30
20
10
0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0

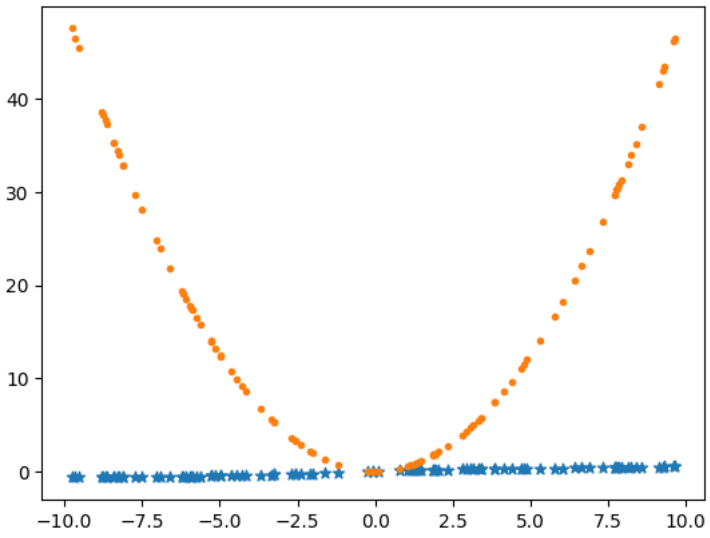

4
3
2
1
0

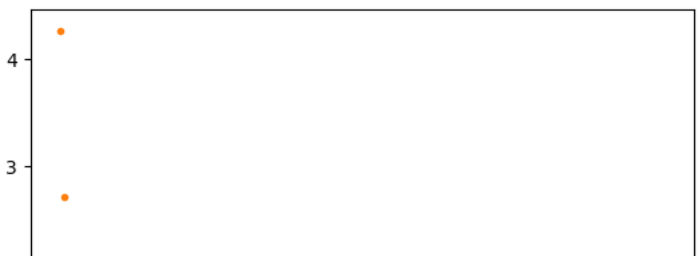
finish----------

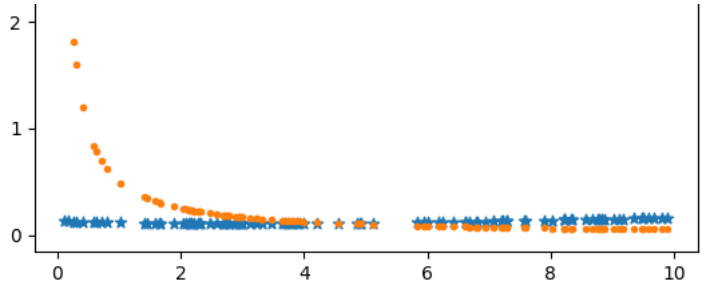
activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 500 , learningRate: 0.0005 , minimum_RMSE: 63.03 , epoch: 14017 , test_loss: 5





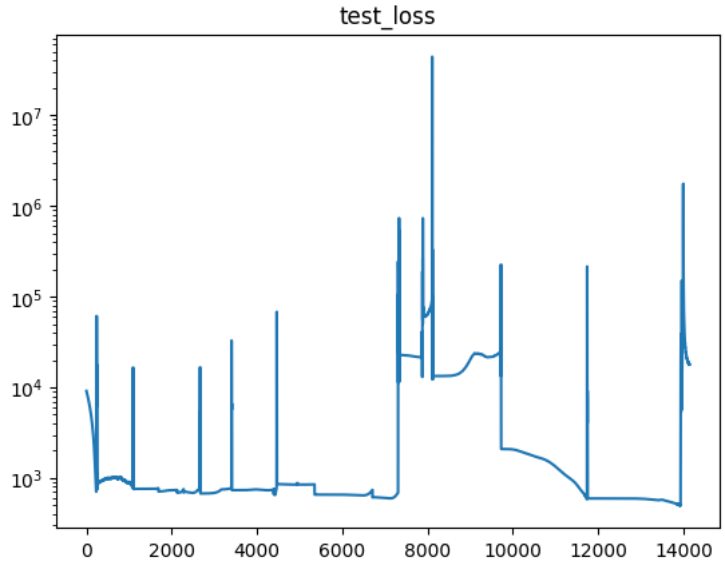


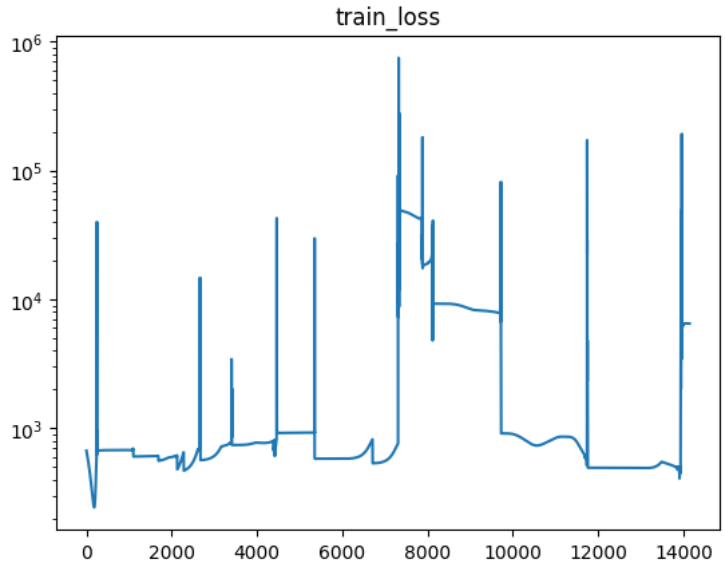


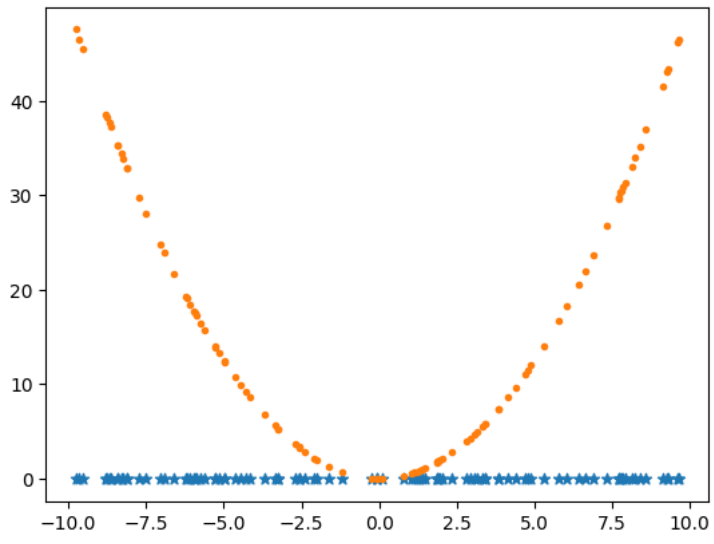


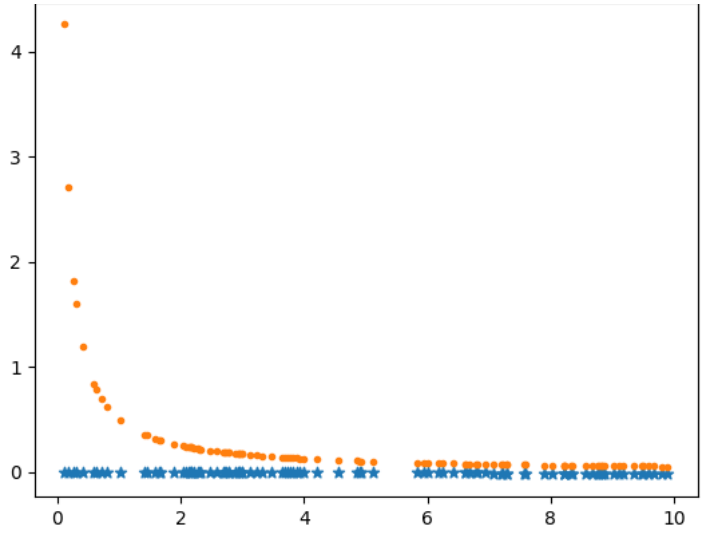
finish----------

activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 500 , learningRate: 1e-05 , minimum_RMSE: 497.84 , epoch: 14141 , test_loss: 1



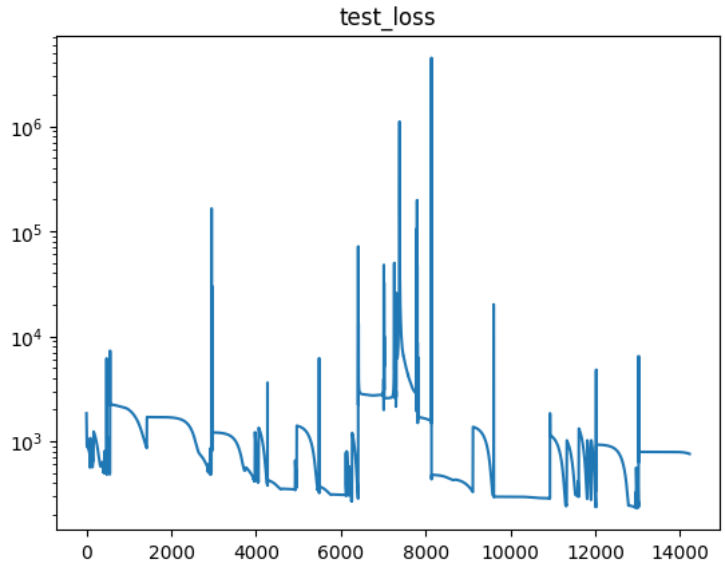


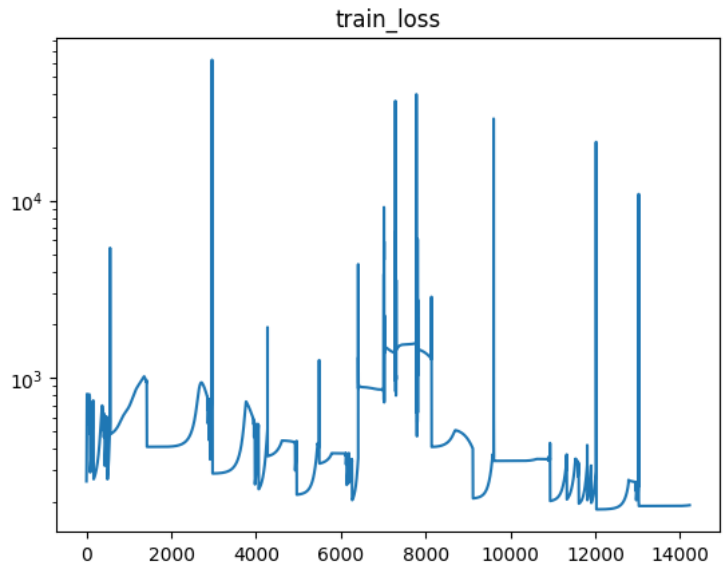


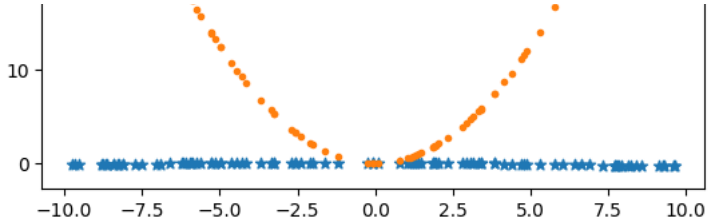


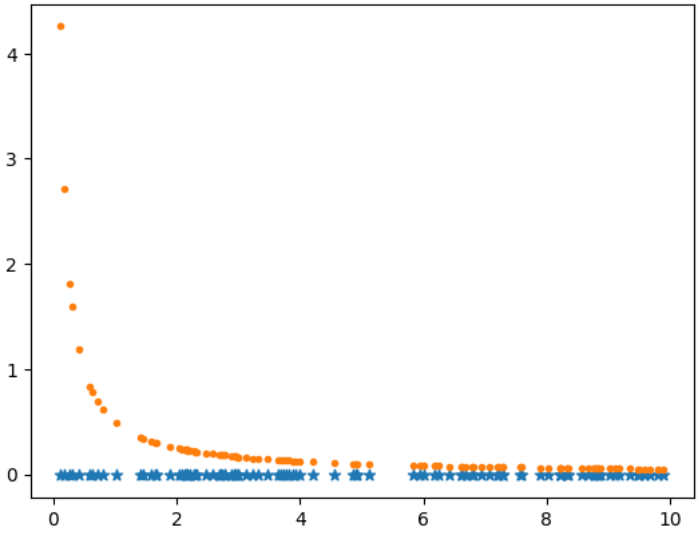
finish----------

activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 500 , learningRate: 5e-05 , minimum_RMSE: 233.48 , epoch: 14226 , test_loss: 7



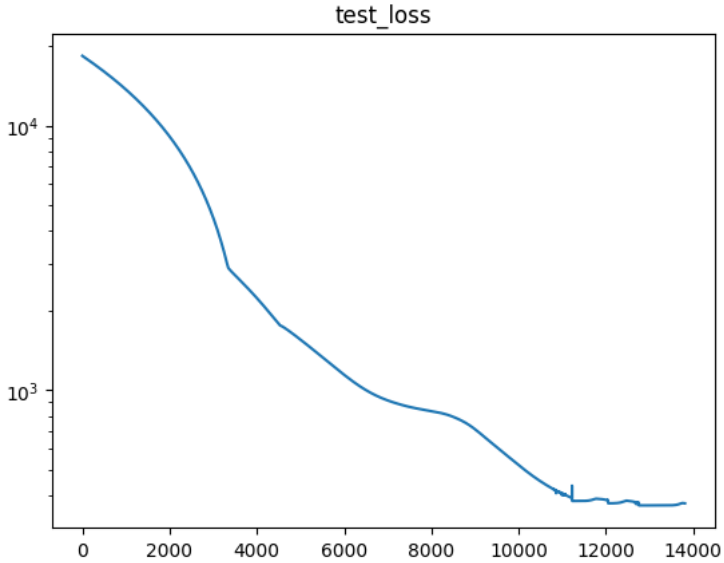


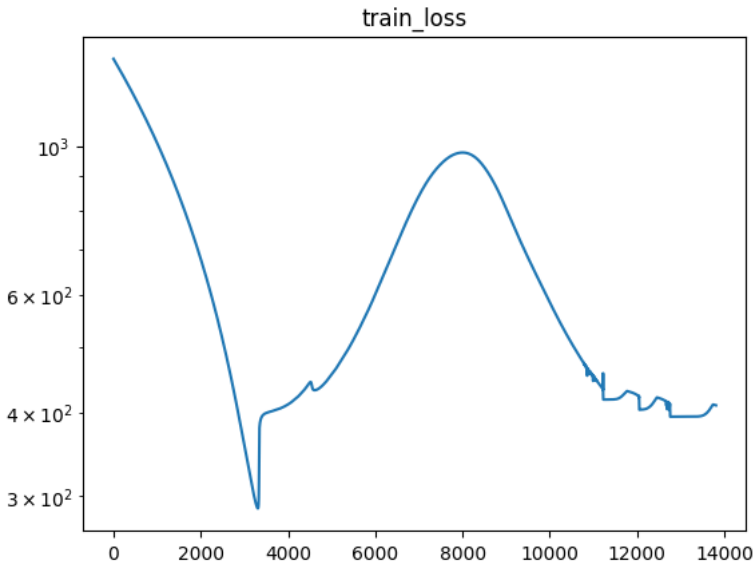


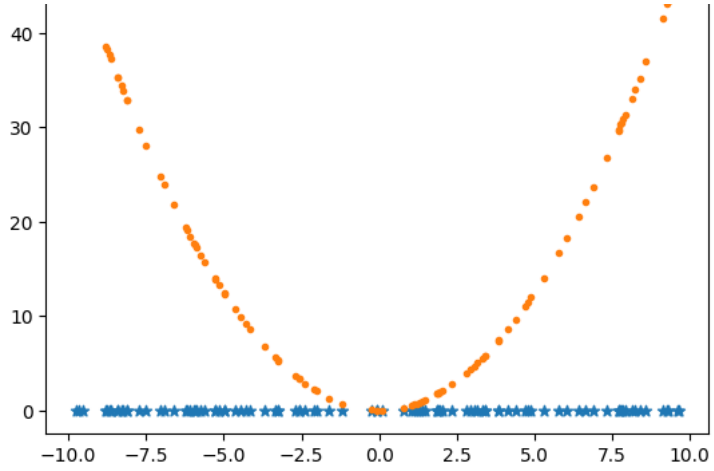


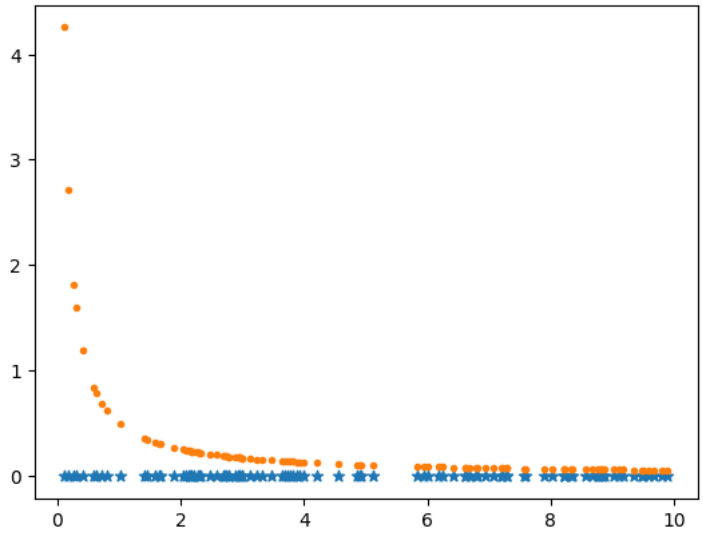
finish----------

activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 500 , learningRate: 1e-06 , minimum_RMSE: 366.90 , epoch: 13808 , test_loss: 3



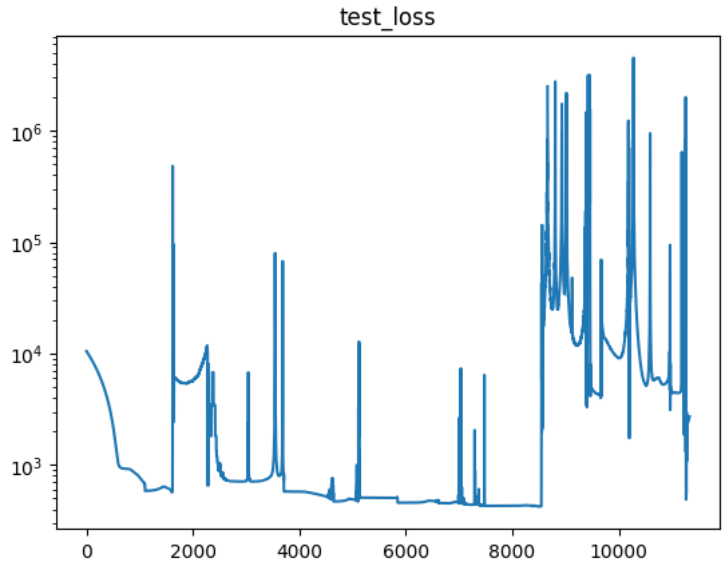


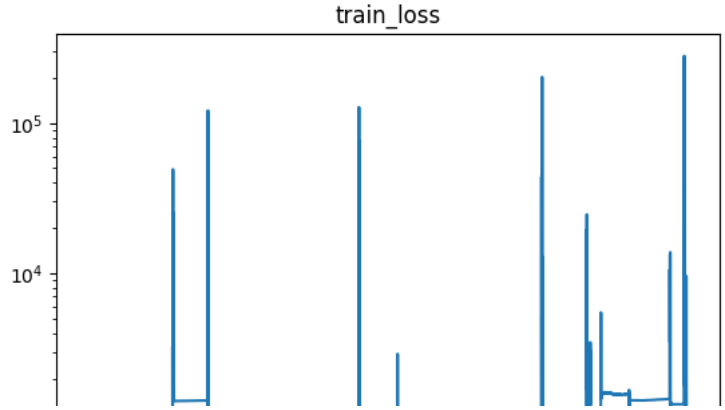


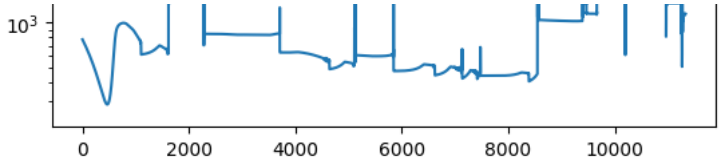


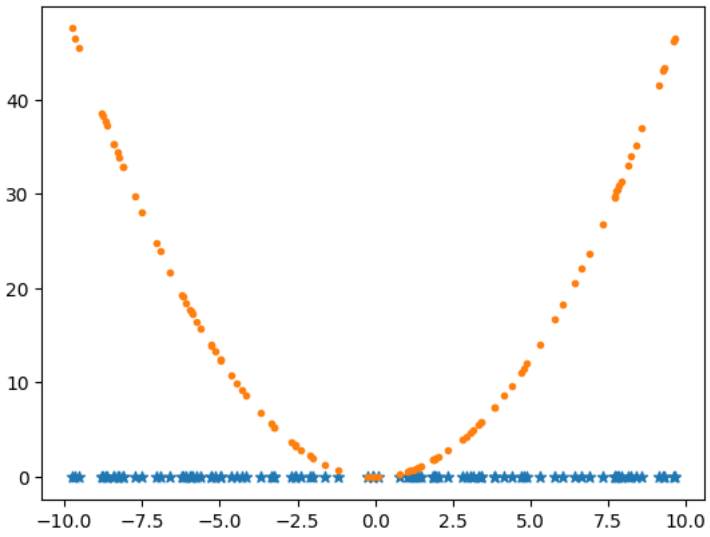
finish----------

activateion: Softplus , optimizer: Adam , n_of_data: 100 , Bsize: 500 , learningRate: 5e-06 , minimum_RMSE: 423.42 , epoch: 11332 , test_loss: n










```
1 y_pred[:10]
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-2-f2f4ca26448f> in <cell line: 1>()
----> 1 y_pred[:10]

NameError: name 'y_pred' is not defined
```

STACK OVERFLOW 검색

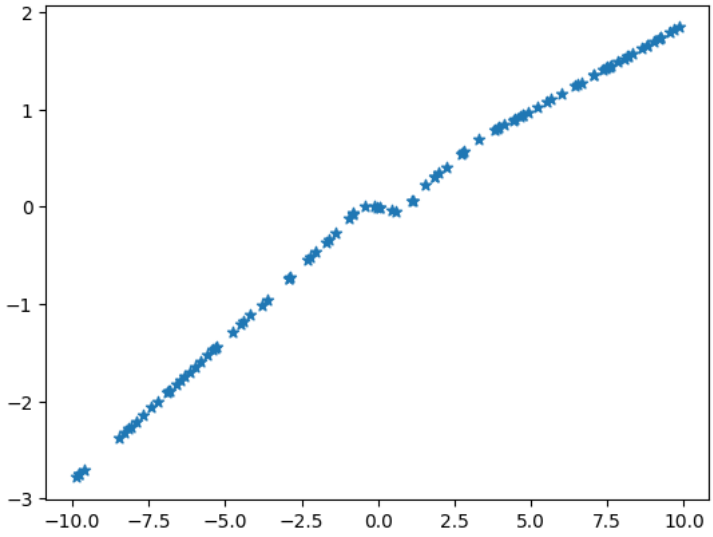
```
1 y_test[:10]
```

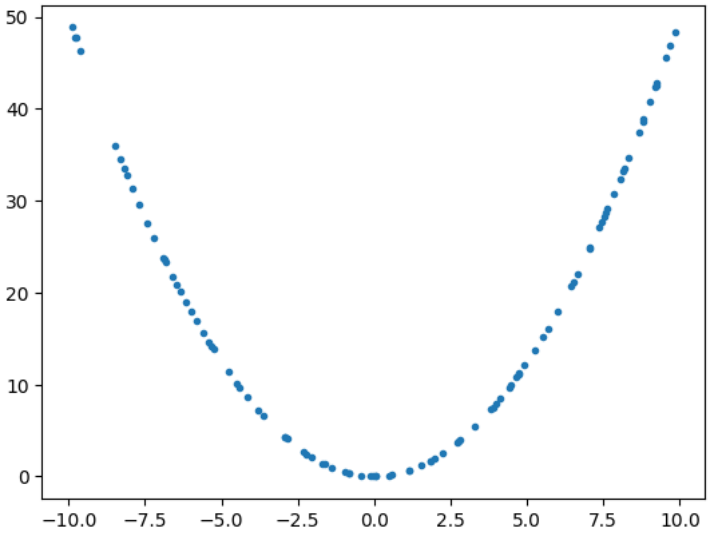
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-3-afb1297a6a30> in <cell line: 1>()
----> 1 y_test[:10]

NameError: name 'y_test' is not defined
```

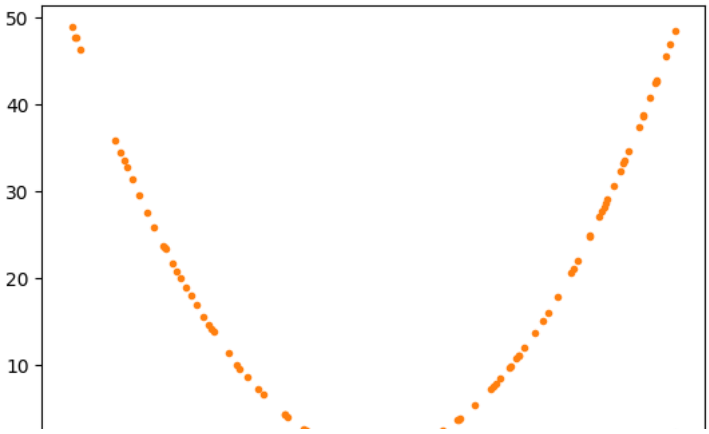
STACK OVERFLOW 검색

```
1
```

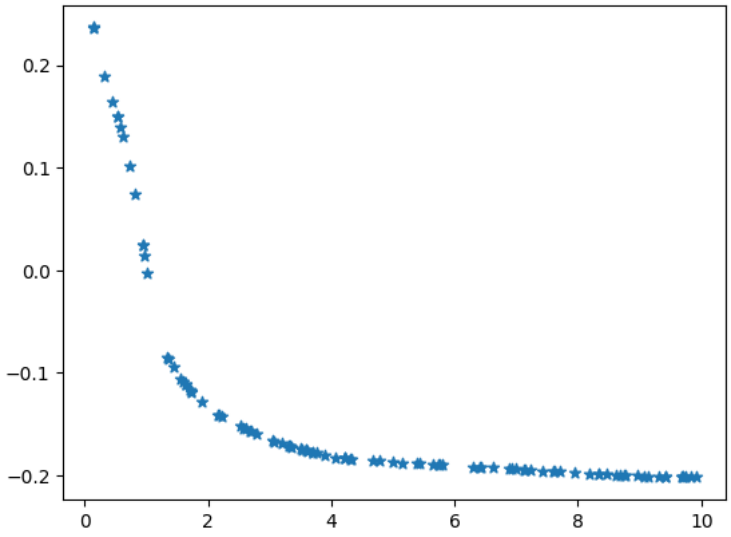
```
1 model.eval()
2 y_pred = model(X2)
3 plt.scatter(X2[:,1].cpu().detach().numpy(),y_pred.cpu().detach().numpy(), marker='*')
4 # plt.scatter(X2[:,1],y_pred, marker='*')
5 plt.show()
6
7
8 plt.scatter(X2[:,1] ,y2, marker='.', )
9 plt.show()
10
11 plt.scatter(X2[:,1].cpu().detach().numpy(),y_pred.cpu().detach().numpy(), marker='*')
12 plt.scatter(X2[:,1] ,y2, marker='.', )
13 plt.show()
14
15
16 print('all done')
```

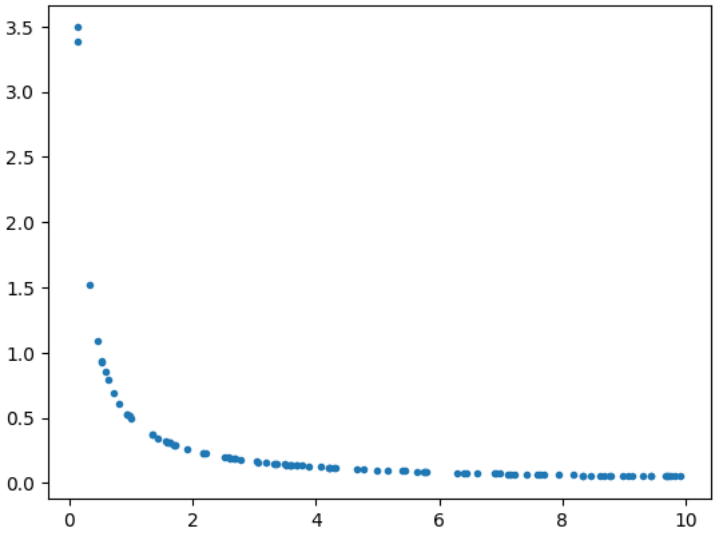


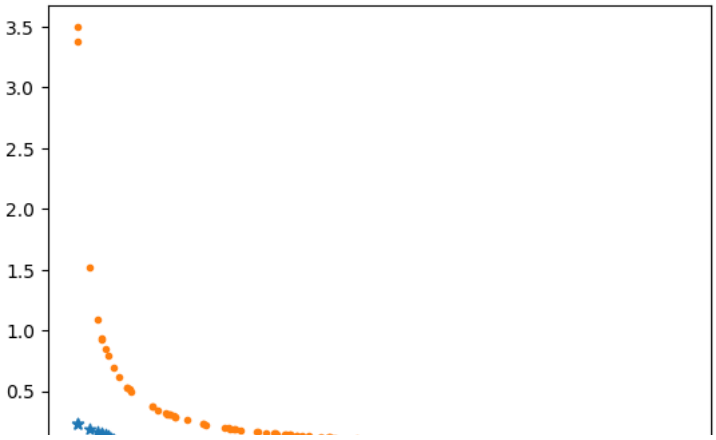




```
model.eval()
y_pred = model(X3)
plt.scatter(X3[:,0].cpu().detach().numpy(),y_pred.cpu().detach().numpy(), marker='*')
# plt.scatter(X3[:,1],y_pred, marker='*')
plt.show()


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

plt.scatter(X3[:,0].cpu().detach().numpy(),y_pred.cpu().detach().numpy(), marker='*')
plt.scatter(X3[:,0] ,y3, marker='.', )
plt.show()
print('all done')
```








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

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

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-68689b04b4ff> in <cell line: 1>()
----> 1 
      2 
      3 plt.title('test loss')
```

```
# when create new data

# N, D_in, H, D_out = 64, 1000, 100, 10
for m in [10, 100, 1000]:
    for k in [1000, 10000, 100000]:
        N, D_in, D_out = k, 2, 1

        # 입력과 출력 위한 랜덤 텐서
        X = []
        y = []
        for j in range(N):
            X.append([])
            #y.append([])
            for i in range(D_out):
                X[-1].append( np.random.uniform(low=0.1, high=10.0, size=None) )
                X[-1].append( np.random.uniform(low=-m, high=m, 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, '/content/drive/MyDrive/0_318lab/SCMP_ML/'+'KEdataX_N_'+ str(N) +'_Interval_'+str(m)+'_'+str(int(time.time()))+'.pt' )
        torch.save(y, '/content/drive/MyDrive/0_318lab/SCMP_ML/'+'KEdataY_N_'+ str(N) +'_Interval_'+str(m)+'_'+str(int(time.time()))+'.pt' )


```

```
1
```