

Dataset3-Boston__output__1

November 3, 2021

1 Dataset 1 - Regression

1.1 Experiment Details

The aim of the experiment is to verify if the: 1. ABC_GAN model corrects model misspecification
2. ABC_GAN model performs better and converges faster than a simple C-GAN model

In the experiment we predict the distribution that represents the real data and simulate realistic fake data points using statistical model, C-GAN and ABC-GAN model with 3 priors. We analyze and compare their performance using metrics like mean squared error, mean absolute error, manhattan distance and euclidean distance between y_{real} and y_{pred}

The models are as follows:

1. The statistical model assumes the distribution $Y = \beta X + \mu$ where $\mu \sim N(0, 1)$
2. The Conditional GAN consists of
 1. Generator with 2 hidden layers with 100 nodes each and ReLu activation.
 2. Discriminator with 2 hidden layers with 25 and 50 nodes and ReLu activation. We use Adam's optimiser and BCE Logit Loss to train the model. The input to the Generator of the GAN is (x,e) where x are the features and $e \sim N(0, 1)$. The discriminator output is linear.
3. The ABC GAN Model consists of
 1. ABC generator is defined as follows:
 1. $Y = 1 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \dots + \beta_n x_n + N(0, \sigma)$ where $\sigma = 0.1$
 2. $\beta_i \sim N(0, \sigma^*)$ when $\mu = 0$ else $\beta_i \sim N(\beta_i^*, \sigma^*)$ where β_i^* s are coefficients obtained from statistical model
 3. σ^* takes the values 0.01, 0.1 and 1
 2. C-GAN network is as defined above. However the input to the Generator of the GAN is (x, y_{abc}) where y_{abc} is the output of the ABC Generator.

1.2 Import Libraries

```
[1]: import warnings
import sys
sys.path.insert(0, '../src')
warnings.filterwarnings('ignore')
```

```
[2]: import train_test
import ABC_train_test
import bostonDataset
import network
import statsModel
import performanceMetrics
import dataset
import sanityChecks
import torch
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from torch.utils.data import Dataset, DataLoader
from torch import nn
```

1.3 Parameters

General Parameters

1. Number of Samples
2. Number of features

ABC-Generator parameters are as mentioned below: 1. mean : 1 ($\beta \sim N(\beta^*, \sigma)$ where β^* are coefficients of statistical model) or 1 ($\beta \sim N(0, \sigma)$) 2. std : $\sigma = 1, 0.1, 0.01$ (standard deviation)

```
[3]: n_features = 13
n_samples= 506

#ABC Generator Parameters
mean = 1
variance = 0.001
```

```
[4]: # Parameters
mean = 1
variance = 0.1
```

1.4 Dataset

Generate a random regression problem

$$Y = 1 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \dots + \beta_n x_n + N(0, \sigma) \text{ where } \sigma = 0.1$$

```
[5]: X,Y = bostonDataset.boston_data()
```

	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	\
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	

	X12	X13	Y
0	396.90	4.98	24.0
1	396.90	9.14	21.6
2	392.83	4.03	34.7
3	394.63	2.94	33.4
4	396.90	5.33	36.2

1.5 Stats Model

```
[6]: [coeff,y_pred] = statsModel.statsModel(X,Y)
```

No handles with labels found to put in legend.

```

=====
                        OLS Regression Results
=====
Dep. Variable:                Y      R-squared:                0.741
Model:                        OLS    Adj. R-squared:            0.734
Method:                        Least Squares    F-statistic:            108.1
Date:                          Wed, 03 Nov 2021    Prob (F-statistic):      6.72e-135
Time:                          20:36:02    Log-Likelihood:          -376.55
No. Observations:              506    AIC:                    781.1
Df Residuals:                  492    BIC:                    840.3
Df Model:                      13
Covariance Type:               nonrobust
=====

```

	coef	std err	t	P> t	[0.025	0.975]
const	-1.635e-15	0.023	-7.12e-14	1.000	-0.045	0.045
x1	-0.1010	0.031	-3.287	0.001	-0.161	-0.041
x2	0.1177	0.035	3.382	0.001	0.049	0.186
x3	0.0153	0.046	0.334	0.738	-0.075	0.105
x4	0.0742	0.024	3.118	0.002	0.027	0.121
x5	-0.2238	0.048	-4.651	0.000	-0.318	-0.129
x6	0.2911	0.032	9.116	0.000	0.228	0.354
x7	0.0021	0.040	0.052	0.958	-0.077	0.082
x8	-0.3378	0.046	-7.398	0.000	-0.428	-0.248
x9	0.2897	0.063	4.613	0.000	0.166	0.413
x10	-0.2260	0.069	-3.280	0.001	-0.361	-0.091
x11	-0.2243	0.031	-7.283	0.000	-0.285	-0.164
x12	0.0924	0.027	3.467	0.001	0.040	0.145
x13	-0.4074	0.039	-10.347	0.000	-0.485	-0.330

```

=====
Omnibus:                    178.041    Durbin-Watson:                1.078
Prob(Omnibus):              0.000    Jarque-Bera (JB):              783.126
Skew:                      1.521    Prob(JB):                      8.84e-171
Kurtosis:                   8.281    Cond. No.                      9.82
=====

```

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Parameters: const -1.634977e-15

x1 -1.010171e-01

x2 1.177152e-01

x3 1.533520e-02

x4 7.419883e-02

x5 -2.238480e-01

x6 2.910565e-01

x7 2.118638e-03

x8 -3.378363e-01

x9 2.897491e-01

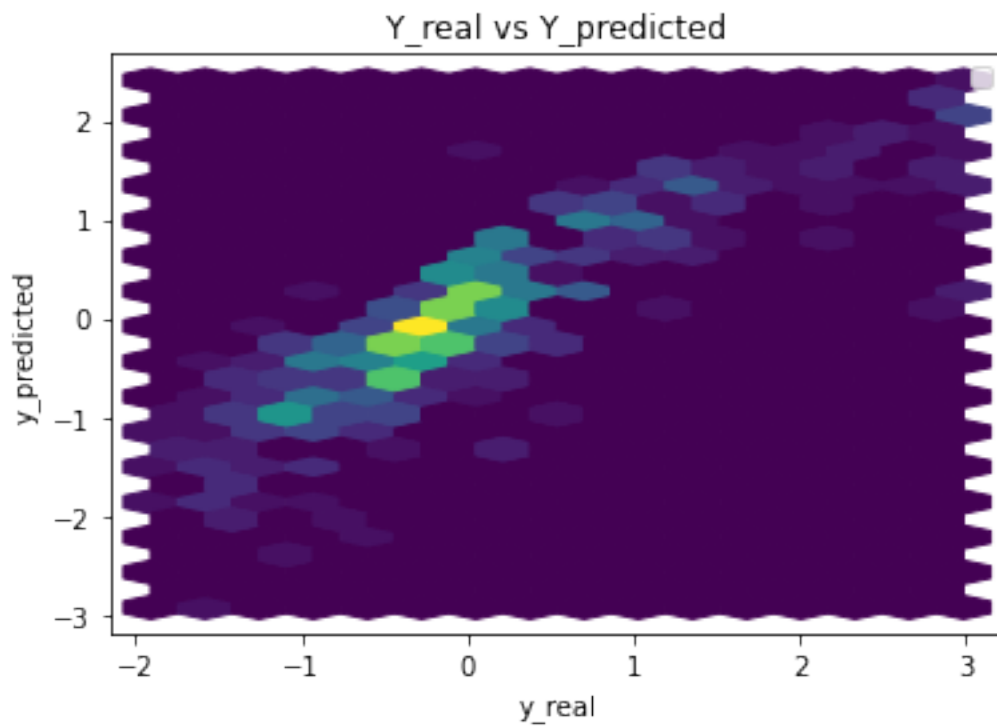
x10 -2.260317e-01

x11 -2.242712e-01

x12 9.243223e-02

x13 -4.074469e-01

dtype: float64



Performance Metrics

Mean Squared Error: 0.2593573358905904

Mean Absolute Error: 0.3559924576478399

Manhattan distance: 180.1321835698068

Euclidean distance: 11.45577635783096

1.6 Common Training Parameters (GAN & ABC_GAN)

```
[7]: n_epochs = 1000
     error = 0.1
     batch_size = n_samples
```

1.7 GAN Model

```
[8]: real_dataset = dataset.CustomDataset(X,Y)
     device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

Training GAN for n_epochs number of epochs

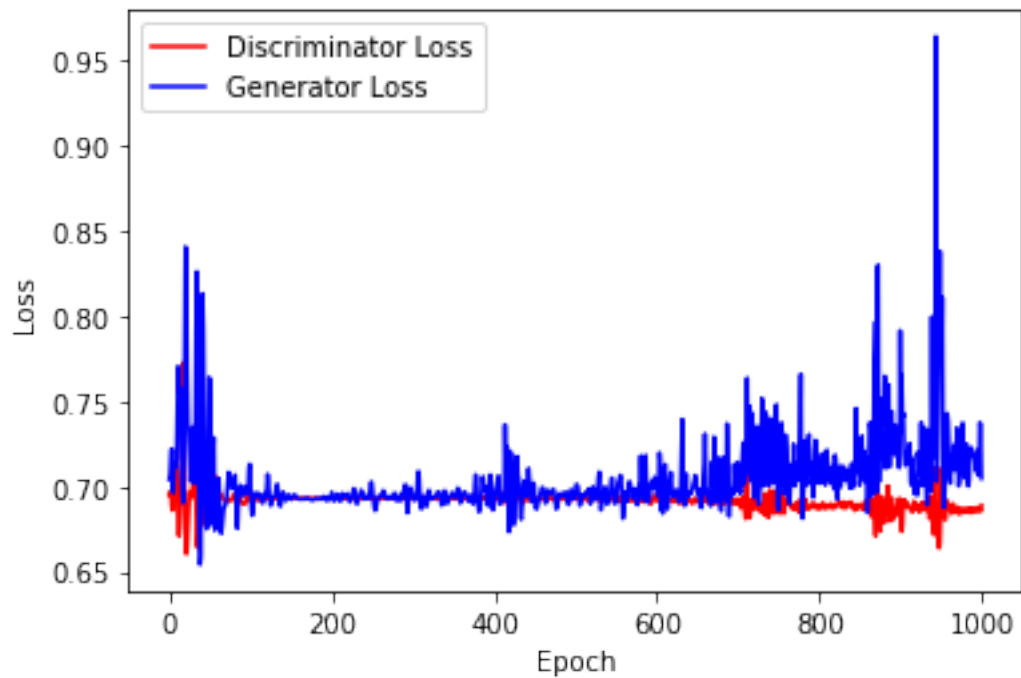
```
[9]: generator = network.Generator(n_features+2)
     discriminator = network.Discriminator(n_features+2)

     criterion = torch.nn.BCEWithLogitsLoss()
     gen_opt = torch.optim.Adam(generator.parameters(), lr=0.01, betas=(0.5, 0.999))
     disc_opt = torch.optim.Adam(discriminator.parameters(), lr=0.01, betas=(0.5, 0.
     ↪999))
```

```
[10]: print(generator)
      print(discriminator)
```

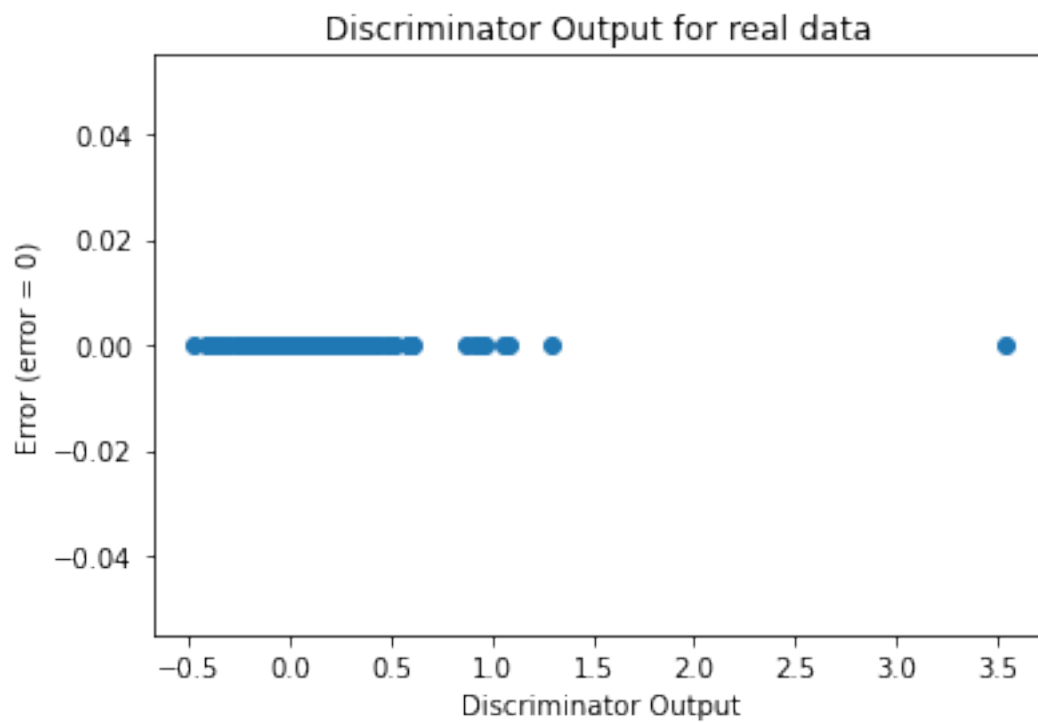
```
Generator(
  (hidden1): Linear(in_features=15, out_features=100, bias=True)
  (hidden2): Linear(in_features=100, out_features=100, bias=True)
  (output): Linear(in_features=100, out_features=1, bias=True)
  (relu): ReLU()
)
Discriminator(
  (hidden1): Linear(in_features=15, out_features=25, bias=True)
  (hidden2): Linear(in_features=25, out_features=50, bias=True)
  (output): Linear(in_features=50, out_features=1, bias=True)
  (relu): ReLU()
)
```

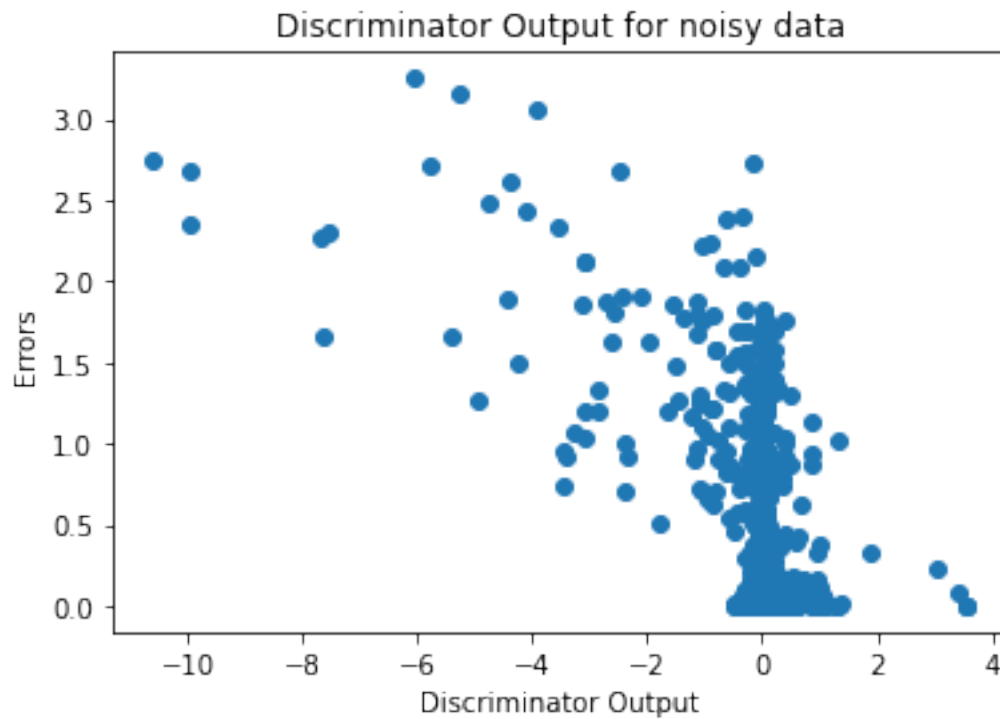
```
[11]: train_test.
     ↪training_GAN(discriminator,generator,disc_opt,gen_opt,real_dataset,batch_size,
     ↪n_epochs,criterion,device)
```



```
[12]: GAN1_metrics = train_test.test_generator(generator,real_dataset,device)
```

```
[13]: sanityChecks.discProbVsError(real_dataset,discriminator,device)
```



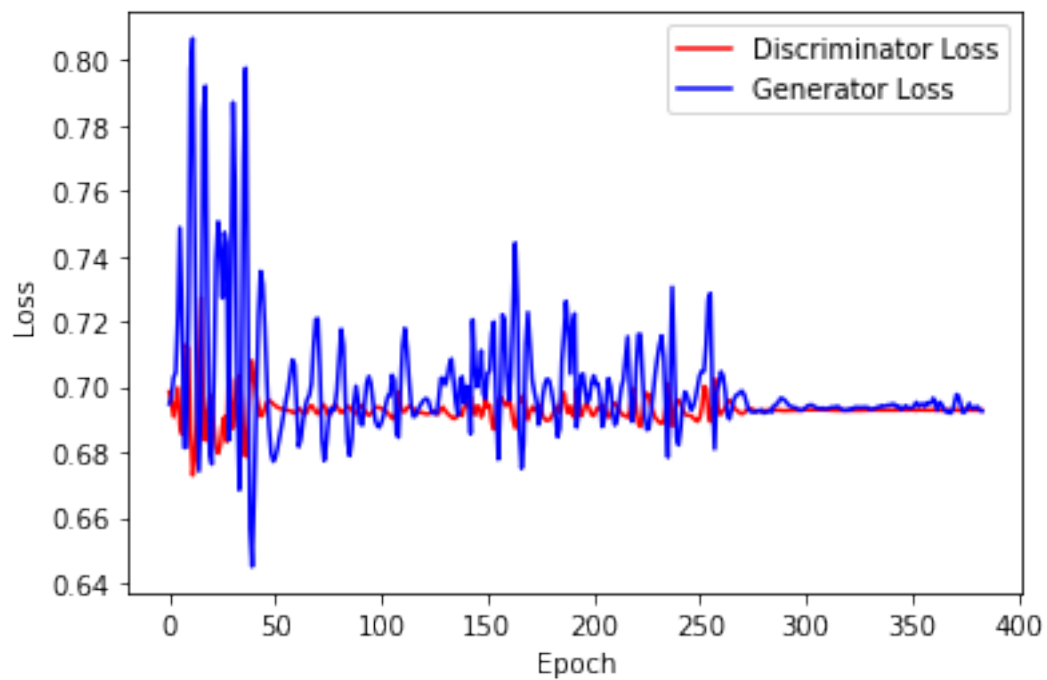


Training GAN until mse of y_pred is > 0.1 or n_epochs < 30000

```
[14]: generator2 = network.Generator(n_features+2)
discriminator2 = network.Discriminator(n_features+2)
criterion = torch.nn.BCEWithLogitsLoss()
gen_opt = torch.optim.Adam(generator2.parameters(), lr=0.01, betas=(0.5, 0.999))
disc_opt = torch.optim.Adam(discriminator2.parameters(), lr=0.01, betas=(0.5, 0.
↪ 999))
```

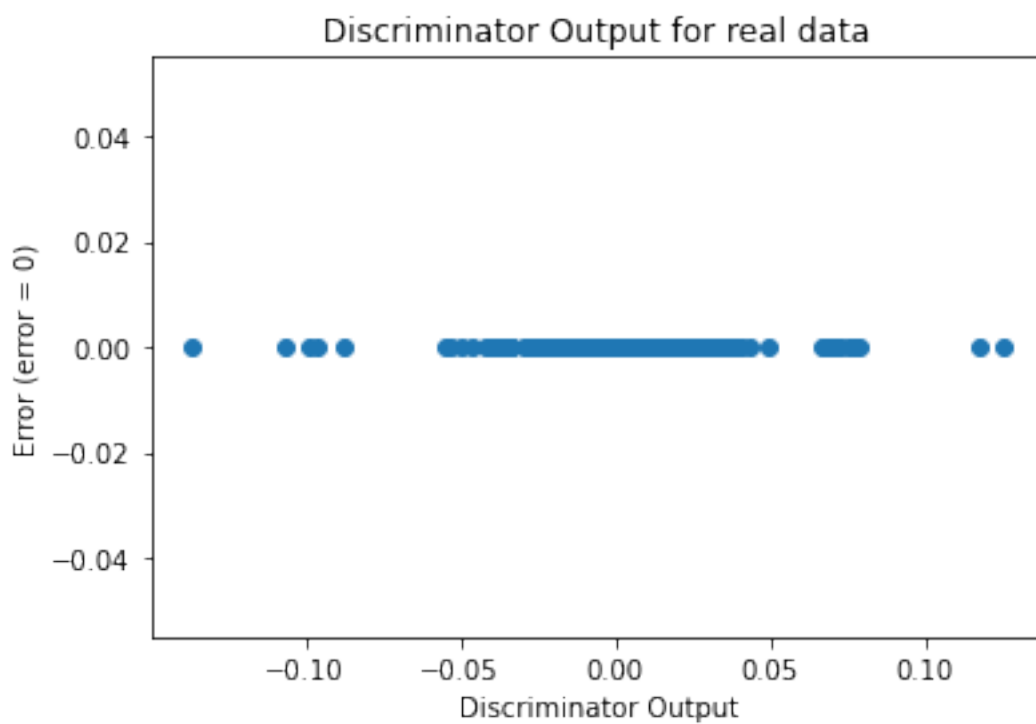
```
[15]: train_test.
↪ training_GAN_2(discriminator2,generator2,disc_opt,gen_opt,real_dataset,batch_size,error,crit
```

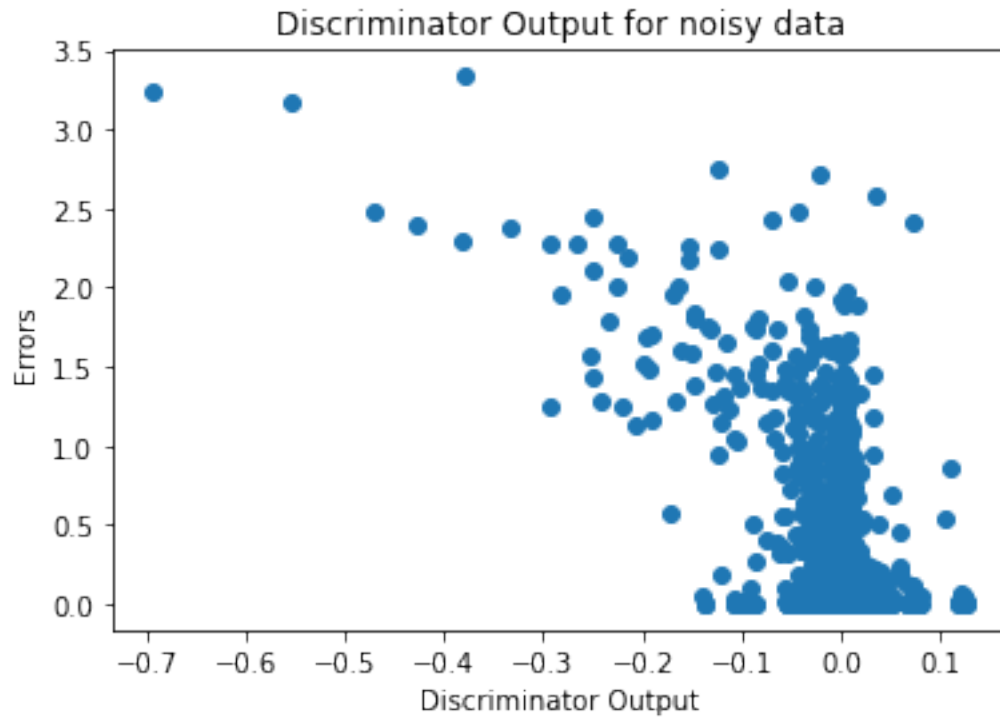
Number of epochs needed 384



```
[16]: GAN2_metrics=train_test.test_generator_2(generator2,real_dataset,device)
```

```
[17]: sanityChecks.discProbVsError(real_dataset,discriminator2,device)
```





2 ABC GAN Model

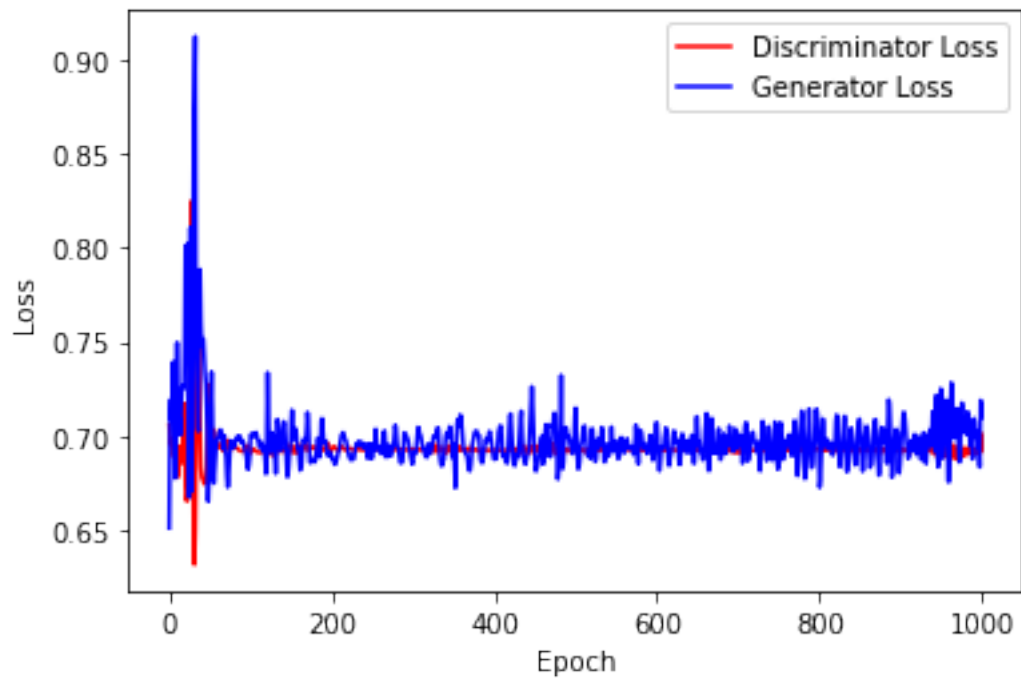
2.0.1 Training the network

Training ABC-GAN for `n_epochs` number of epochs

```
[18]: gen = network.Generator(n_features+2)
      disc = network.Discriminator(n_features+2)

      criterion = torch.nn.BCEWithLogitsLoss()
      gen_opt = torch.optim.Adam(gen.parameters(), lr=0.01, betas=(0.5, 0.999))
      disc_opt = torch.optim.Adam(disc.parameters(), lr=0.01, betas=(0.5, 0.999))

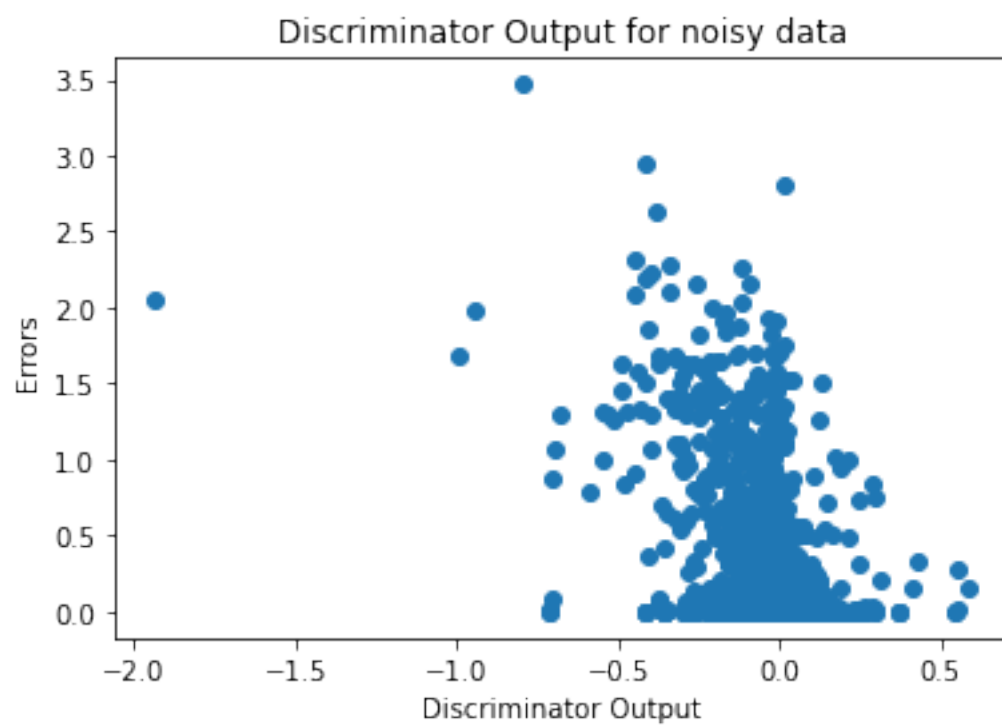
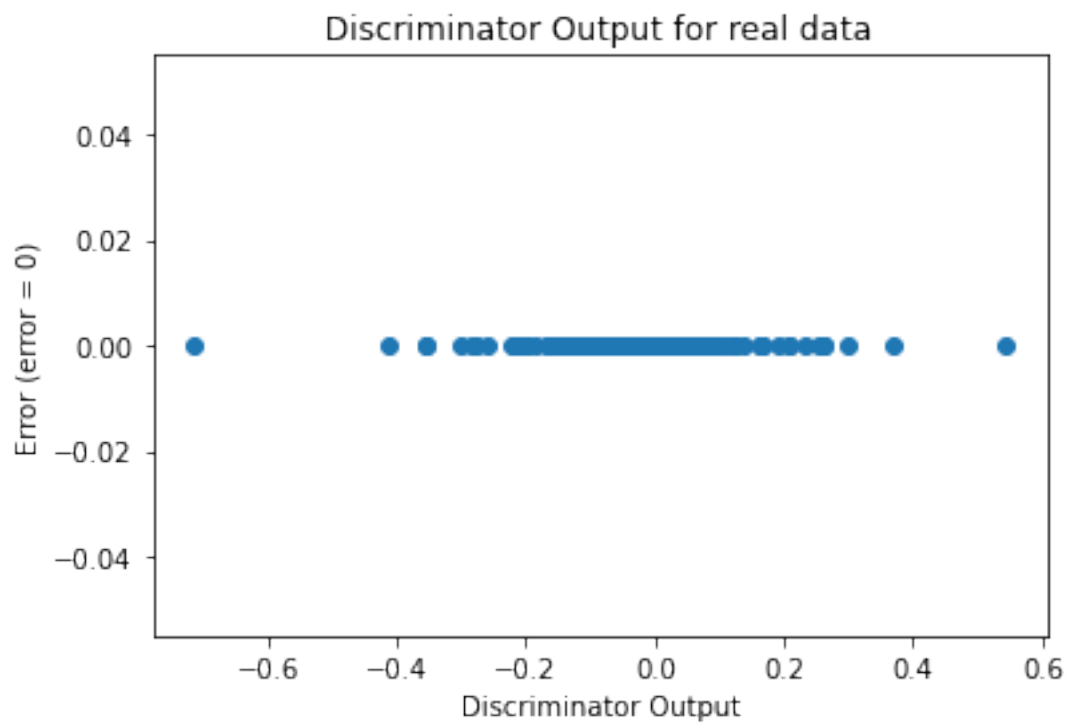
[19]: ABC_train_test.training_GAN(disc, gen,disc_opt,gen_opt,real_dataset,
      ↪batch_size, n_epochs,criterion,coeff,mean,variance,device)
```



```
[20]: ABC_GAN1_metrics=ABC_train_test.  
      ↪ test_generator(gen,real_dataset,coeff,mean,variance,device)
```

Sanity Checks

```
[21]: sanityChecks.discProbVsError(real_dataset,disc,device)
```



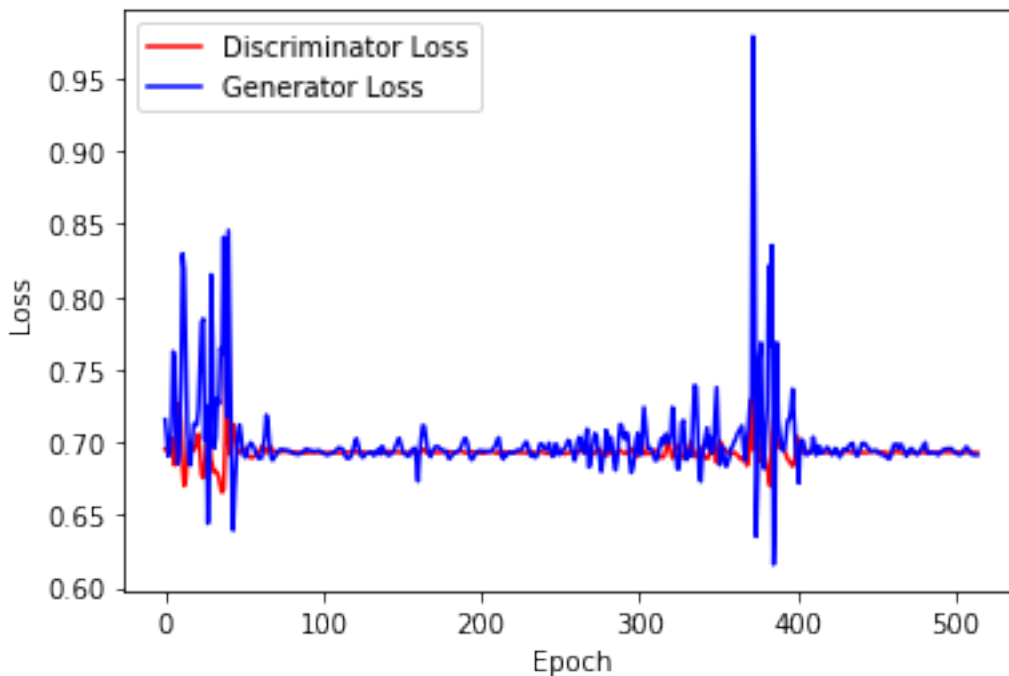
Training GAN until mse of y_pred is > 0.1 or n_epochs < 30000

```
[22]: gen2 = network.Generator(n_features+2)
disc2 = network.Discriminator(n_features+2)

criterion = torch.nn.BCEWithLogitsLoss()
gen_opt = torch.optim.Adam(gen2.parameters(), lr=0.01, betas=(0.5, 0.999))
disc_opt = torch.optim.Adam(disc2.parameters(), lr=0.01, betas=(0.5, 0.999))
```

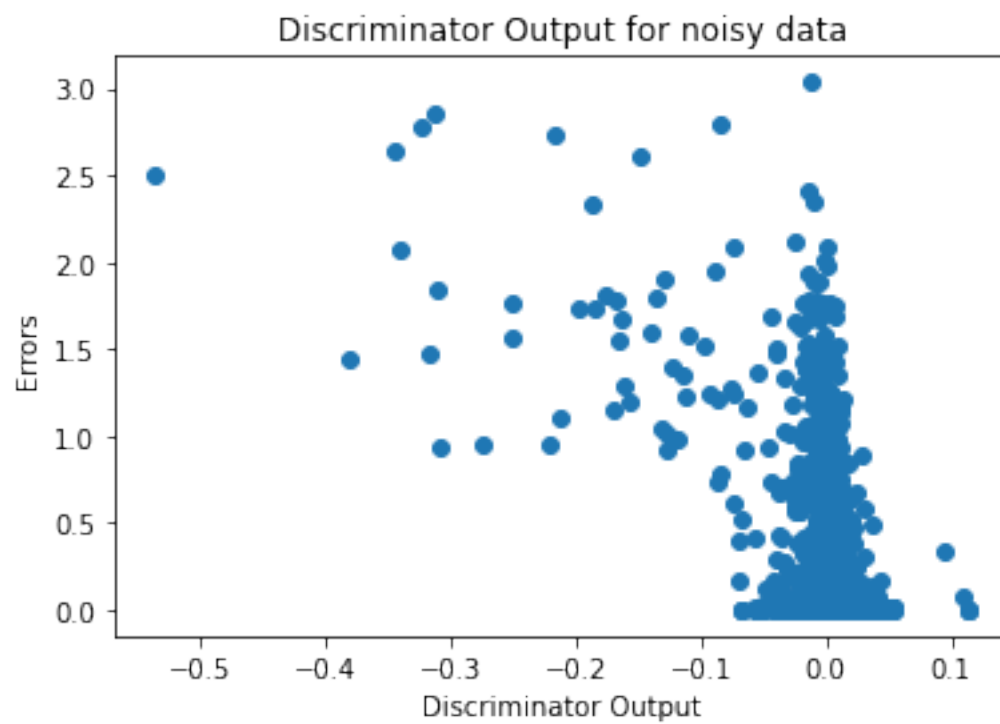
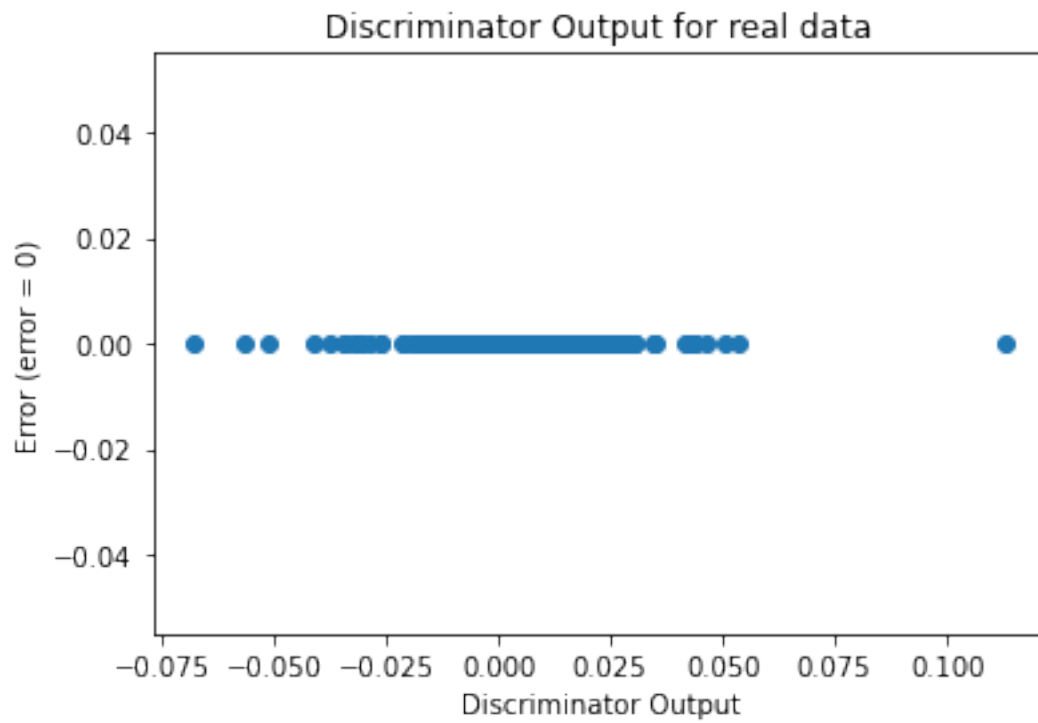
```
[23]: ABC_train_test.
      ↪ training_GAN_2(disc2,gen2,disc_opt,gen_opt,real_dataset,batch_size,
      ↪ error,criterion,coeff,mean,variance,device)
```

Number of epochs 515



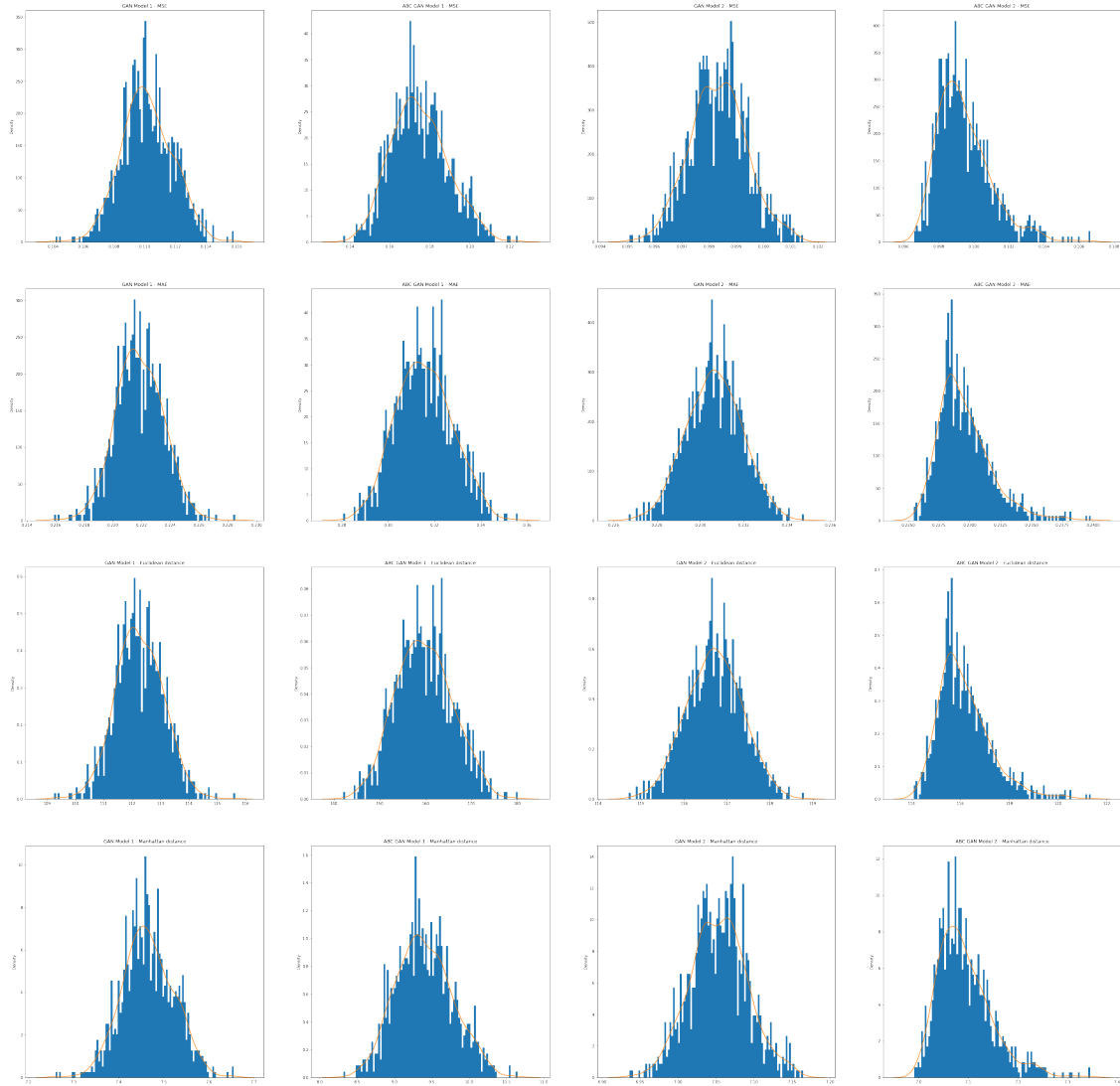
```
[24]: ABC_GAN2_metrics=ABC_train_test.
      ↪ test_generator_2(gen2,real_dataset,coeff,mean,variance,device)
```

```
[25]: sanityChecks.discProbVsError(real_dataset,disc2,device)
```



3 Model Analysis

```
[26]: performanceMetrics.  
      ↪ modelAnalysis(GAN1_metrics, ABC_GAN1_metrics, GAN2_metrics, ABC_GAN2_metrics)
```



```
[ ]:
```