

# Dataset1-Regression\_output\_0

November 2, 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  $\beta_i^*$ s are coefficients obtained from statistical model
    3.  $\sigma^*$  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
warnings.filterwarnings('ignore')
```

```
[2]: import train_test
import ABC_train_test
import regressionDataset
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  $\beta^*$  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 = 10
     n_samples= 100

     #ABC Generator Parameters
     mean = 1
     variance = 0.001

```

```

[4]: # Parameters
     n_samples = 10
     n_features = 10
     mean = 1
     variance = 1

```

### 1.4 Dataset

Generate a random regression problem

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

```

[5]: X,Y = regressionDataset.regression_data(n_samples,n_features)

```

	X1	X2	X3	X4	X5	X6	X7 \	
0	-0.132233	-0.028759	0.340941	-0.787356	1.092884	0.914611	-1.894592	
1	2.451372	-0.480244	-1.696375	0.017271	-0.307065	-0.492008	-0.824177	
2	0.591301	0.238523	0.106987	0.328085	1.014021	0.038881	-0.329277	
3	-0.605792	-0.775314	-0.555941	-0.462995	-1.560697	-0.550646	0.499851	
4	0.569273	1.170418	1.112202	0.628095	-0.150332	1.127442	0.145510	
	X8	X9	X10	Y				

```

0 -1.349358  0.582722 -0.614461 -234.317829
1  0.979184 -1.329518 -1.850007 -279.805643
2 -1.504904  0.502714 -1.224685 -152.137407
3  1.646087 -0.679167  0.472317   8.925252
4 -0.834494  0.632997 -0.593704  42.560983

```

## 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:                1.000
Model:                        OLS    Adj. R-squared:            nan
Method:                        Least Squares    F-statistic:            nan
Date:                          Tue, 02 Nov 2021    Prob (F-statistic):        nan
Time:                          18:19:23    Log-Likelihood:           329.87
No. Observations:              10    AIC:                      -639.7
Df Residuals:                  0    BIC:                      -636.7
Df Model:                      9
Covariance Type:               nonrobust
=====

```

	coef	std err	t	P> t	[0.025	0.975]
const	1.943e-16	inf	0	nan	nan	nan
x1	0.0985	inf	0	nan	nan	nan
x2	-0.0082	inf	-0	nan	nan	nan
x3	0.0777	inf	0	nan	nan	nan
x4	0.3740	inf	0	nan	nan	nan
x5	0.1994	inf	0	nan	nan	nan
x6	0.2097	inf	0	nan	nan	nan
x7	0.3424	inf	0	nan	nan	nan
x8	0.5244	inf	0	nan	nan	nan
x9	0.3591	inf	0	nan	nan	nan
x10	0.7434	inf	0	nan	nan	nan

```

=====
Omnibus:                      0.659    Durbin-Watson:           1.241
Prob(Omnibus):                0.719    Jarque-Bera (JB):         0.563
Skew:                         -0.169    Prob(JB):                 0.755
Kurtosis:                     1.888    Cond. No.                 20.1
=====

```

Notes:

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

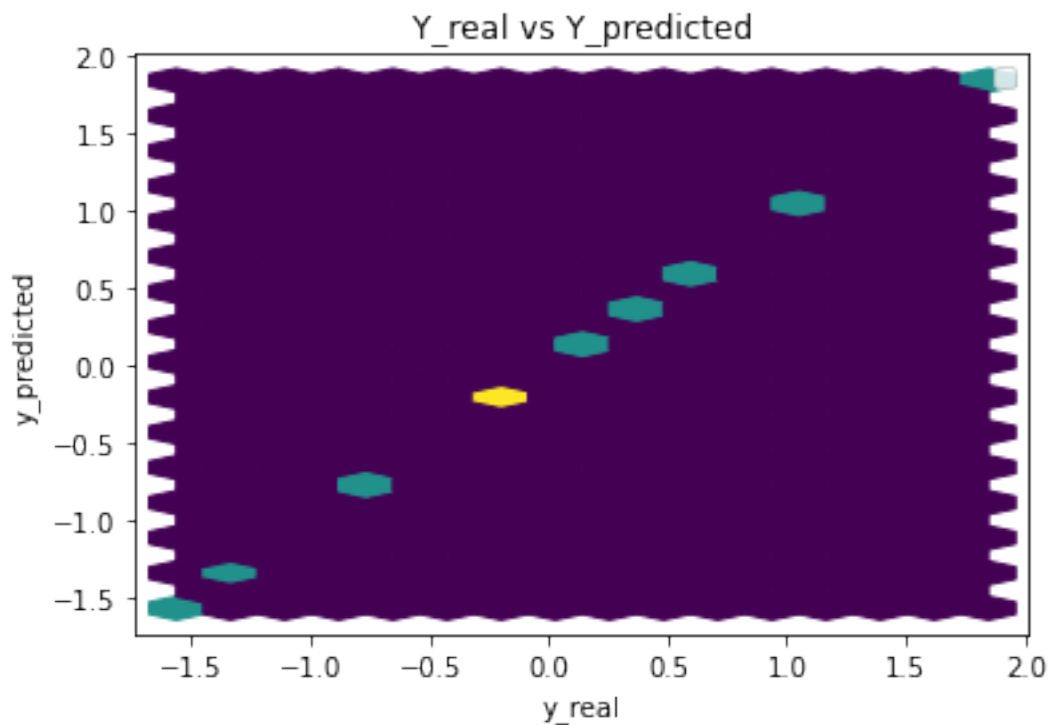
[2] The input rank is higher than the number of observations.

Parameters: const 1.942890e-16

```

x1      9.846035e-02
x2     -8.177198e-03
x3      7.766082e-02
x4      3.739881e-01
x5      1.994477e-01
x6      2.096718e-01
x7      3.424489e-01
x8      5.244197e-01
x9      3.591327e-01
x10     7.433969e-01
dtype: float64

```



#### Performance Metrics

```

Mean Squared Error: 1.305164204712342e-30
Mean Absolute Error: 9.159339953157541e-16
Manhattan distance: 9.159339953157541e-15
Euclidean distance: 3.612705640807651e-15

```

### 1.6 Common Training Parameters (GAN & ABC\_GAN)

```

[7]: n_epochs = 5000
     error = 0.001
     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

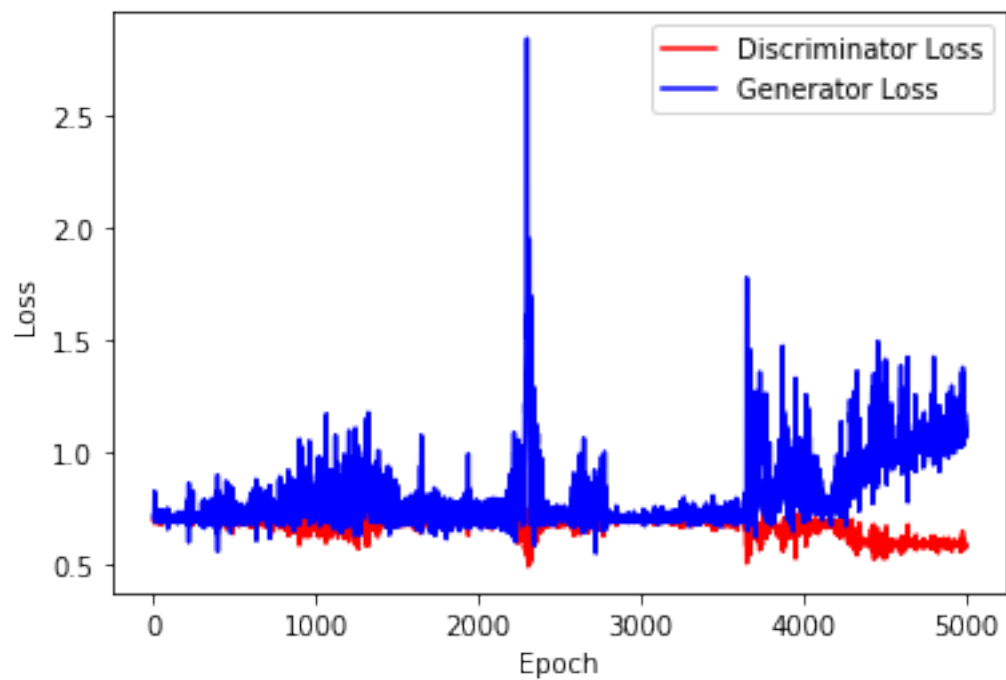
```
[9]: generator = network.Generator(n_features+2)
discriminator = network.Discriminator(n_features+2)

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)
```

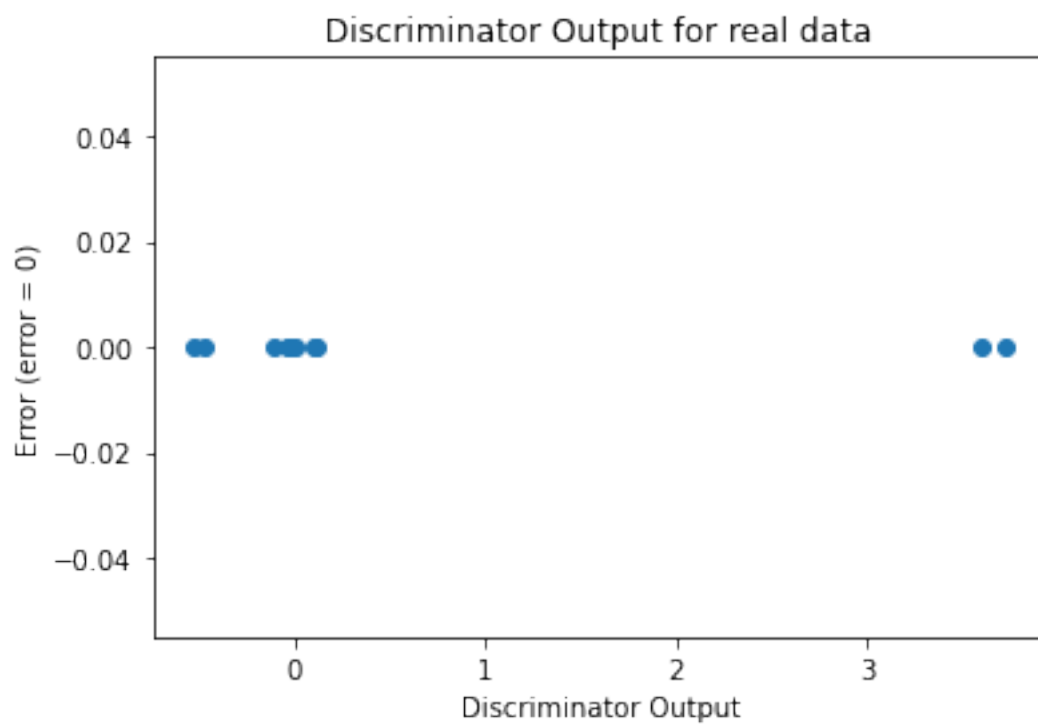
```
Generator(
  (hidden1): Linear(in_features=12, out_features=100, bias=True)
  (hidden2): Linear(in_features=100, out_features=100, bias=True)
  (output): Linear(in_features=100, out_features=1, bias=True)
  (relu): ReLU()
)
Discriminator(
  (hidden1): Linear(in_features=12, out_features=25, bias=True)
  (hidden2): Linear(in_features=25, out_features=50, bias=True)
  (output): Linear(in_features=50, out_features=1, bias=True)
  (relu): ReLU()
)
```

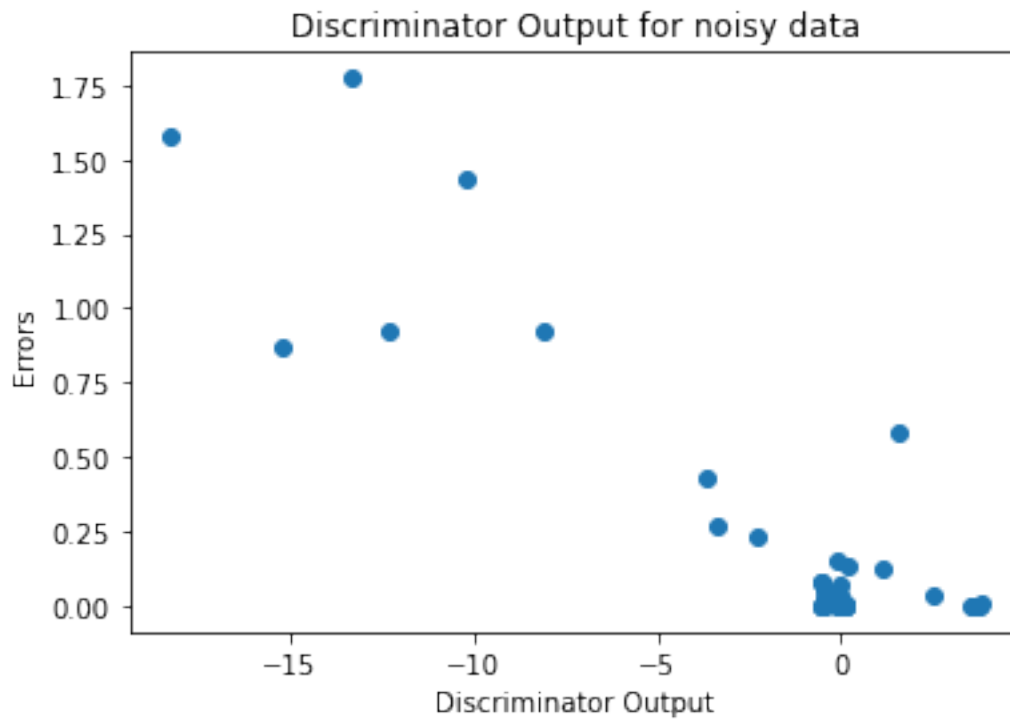
```
[11]: train_test.
→training_GAN(discriminator,generator,disc_opt,gen_opt,real_dataset,batch_size,
→n_epochs,criterion,device)
```



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

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



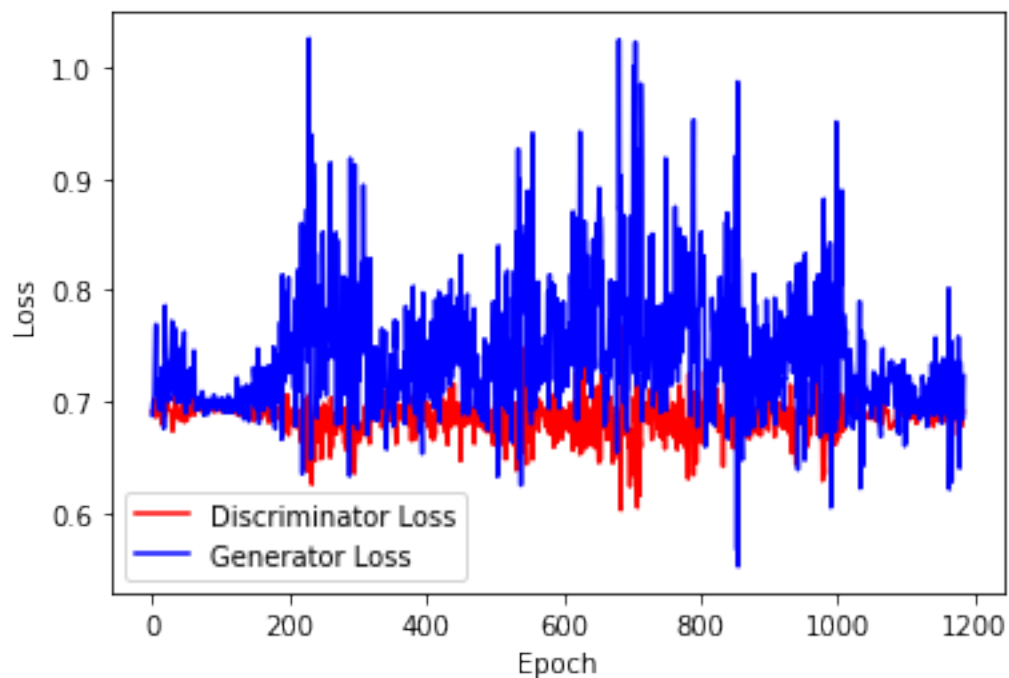


Training GAN until mse of y\_pred is  $> 0.1$  or n\_epochs  $< 30000$

```
[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))
```

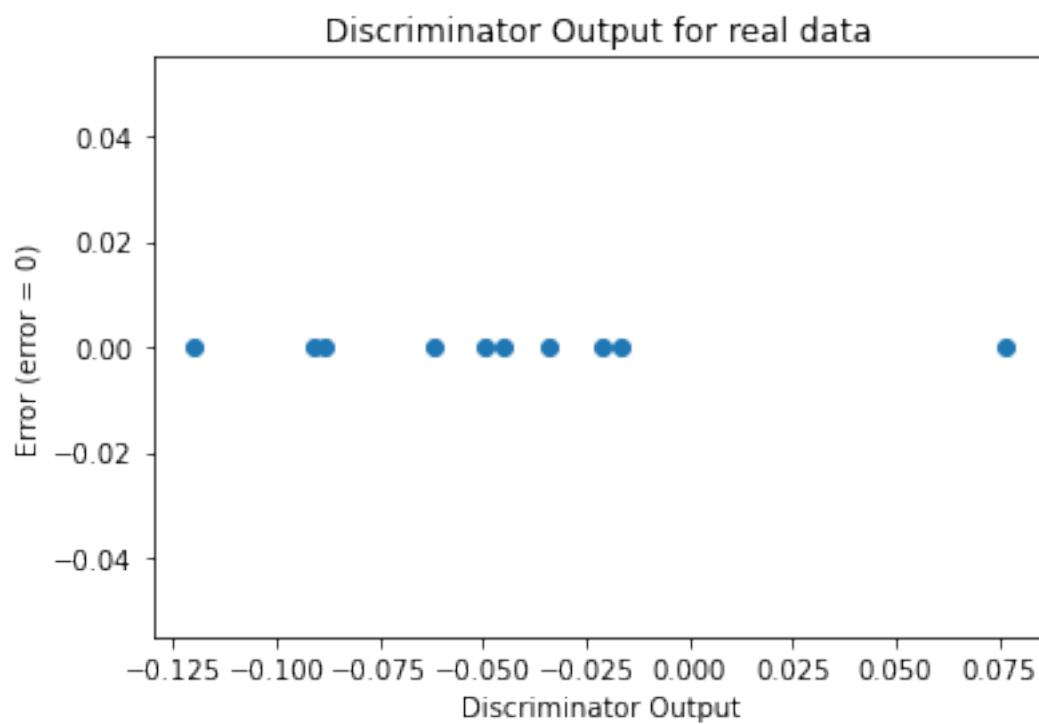
```
[15]: train_test.
↪ training_GAN_2(discriminator2,generator2,disc_opt,gen_opt,real_dataset,batch_size,error,crit
```

Number of epochs needed 1185

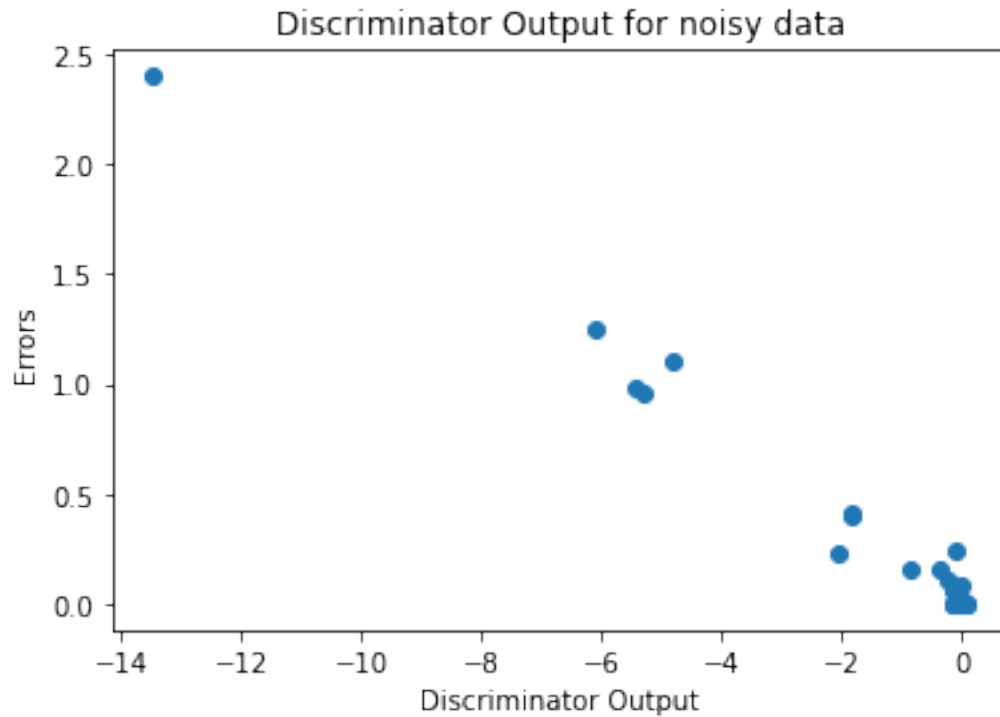


```
[16]: GAN2_metrics=train_test.test_generator_2(generator2,real_dataset,device)
```

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







## 2 ABC GAN Model

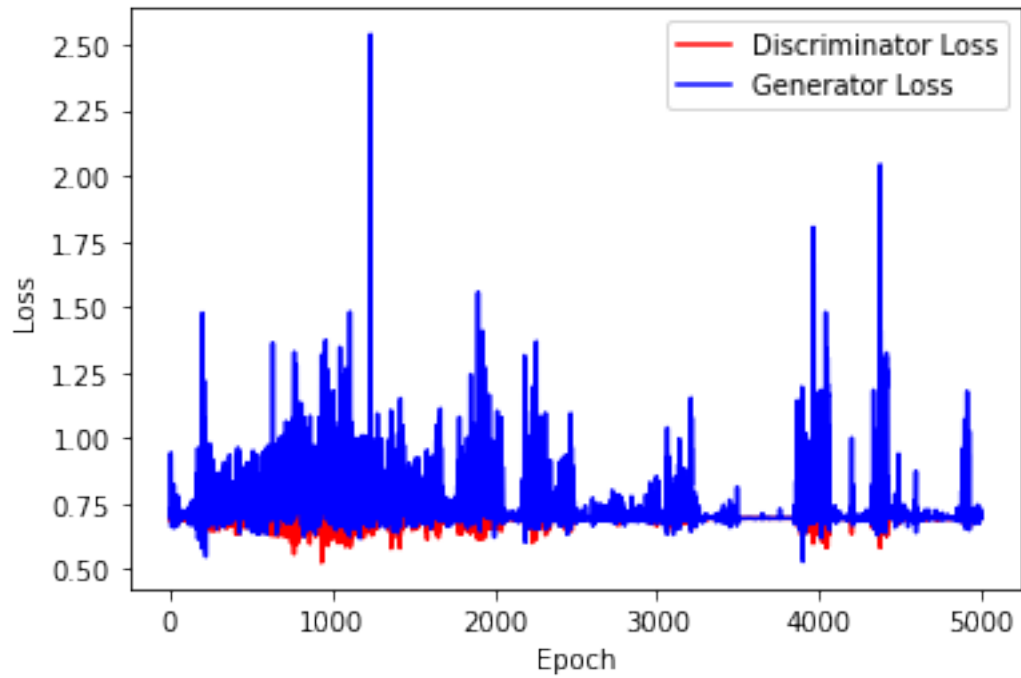
### 2.0.1 Training the network

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))

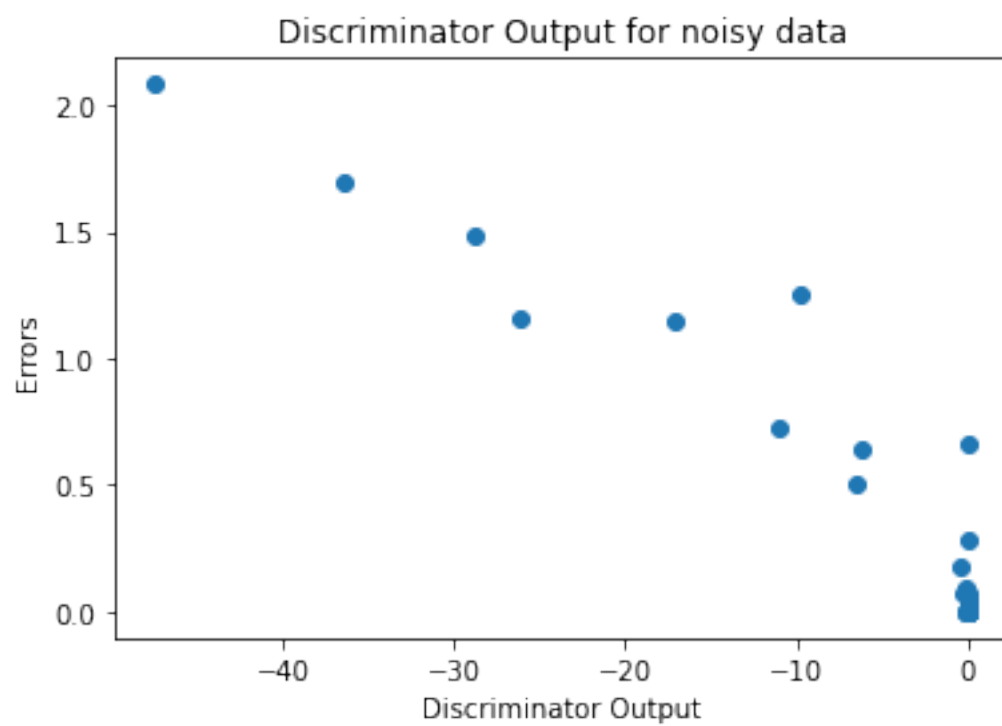
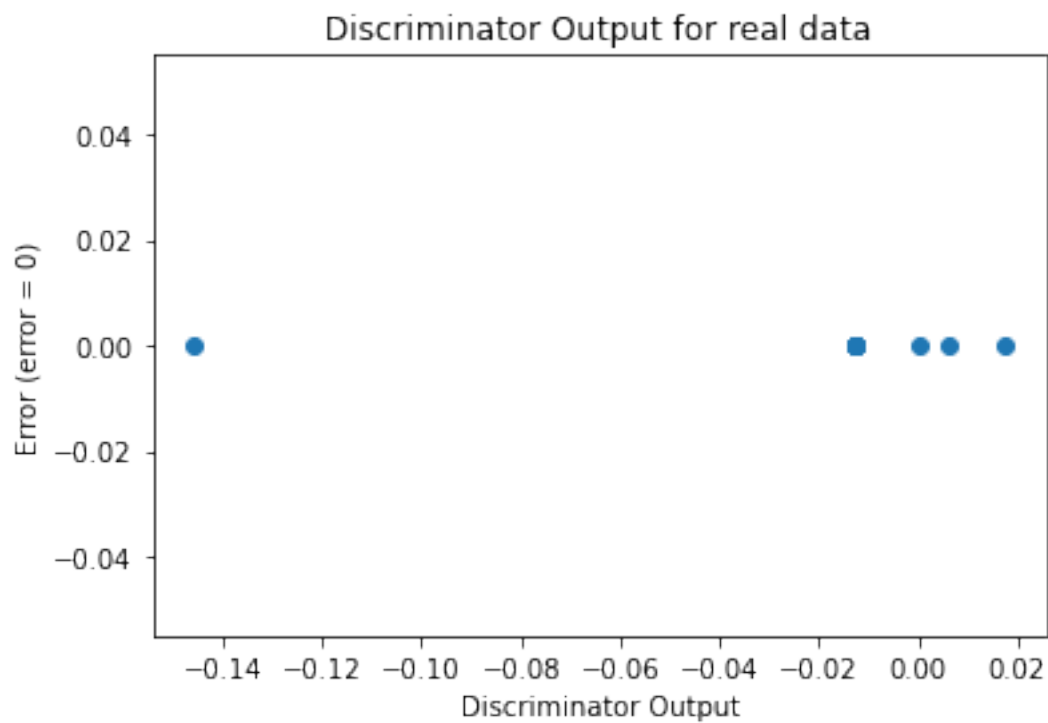
[19]: ABC_train_test.training_GAN(disc, gen,disc_opt,gen_opt,real_dataset,
      ↪batch_size, n_epochs,criterion,coeff,mean,variance,device)
```



```
[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)
```



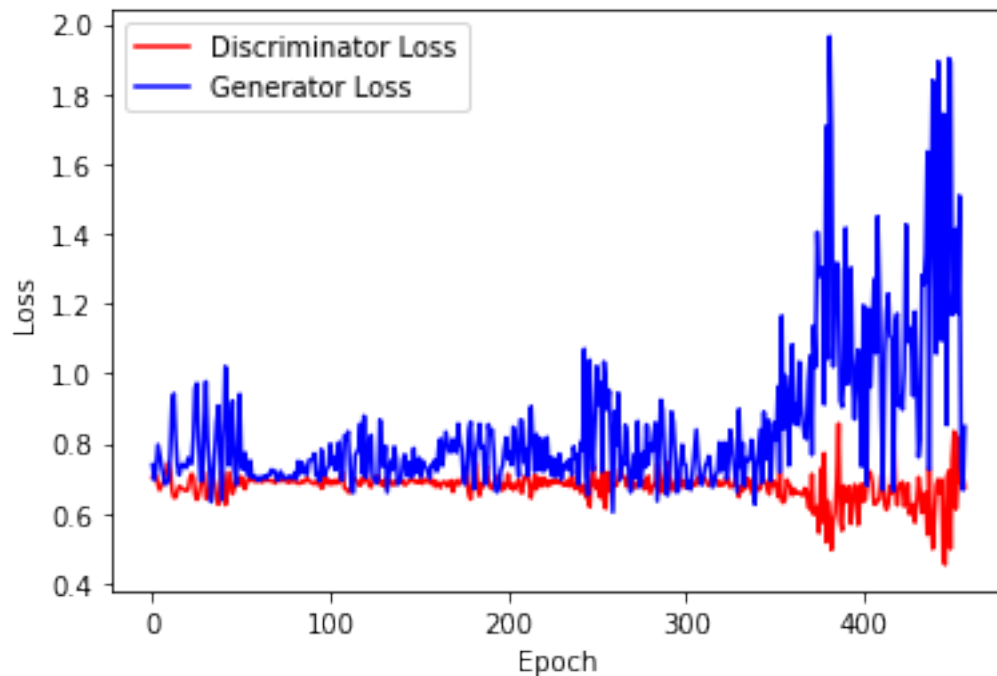
Training GAN until mse of y\_pred is  $> 0.1$  or n\_epochs  $< 30000$

```
[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))
```

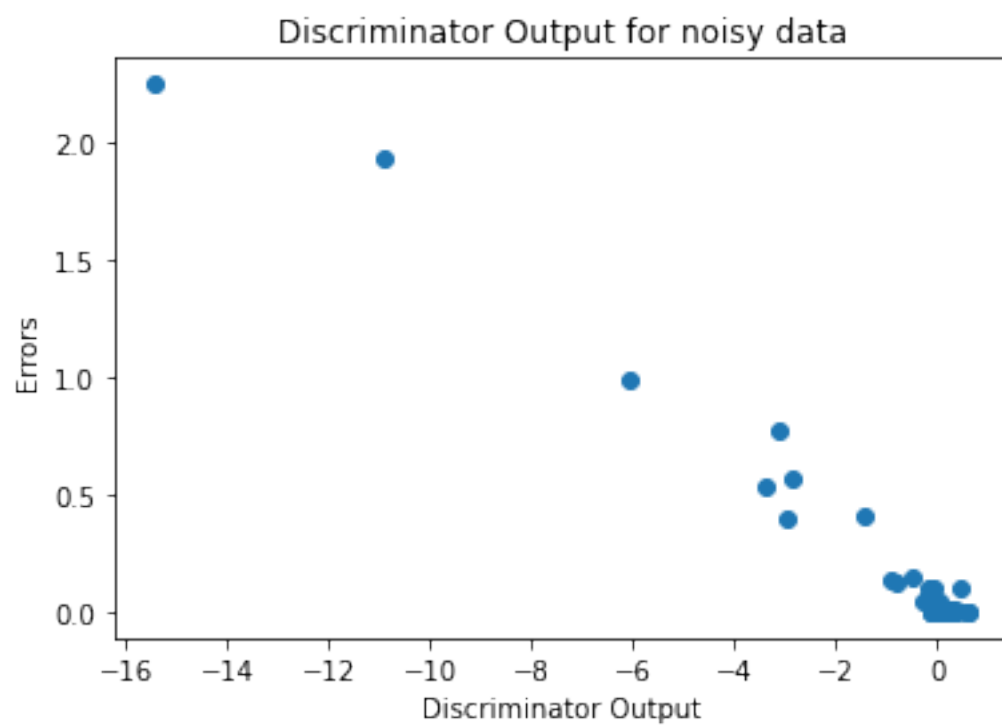
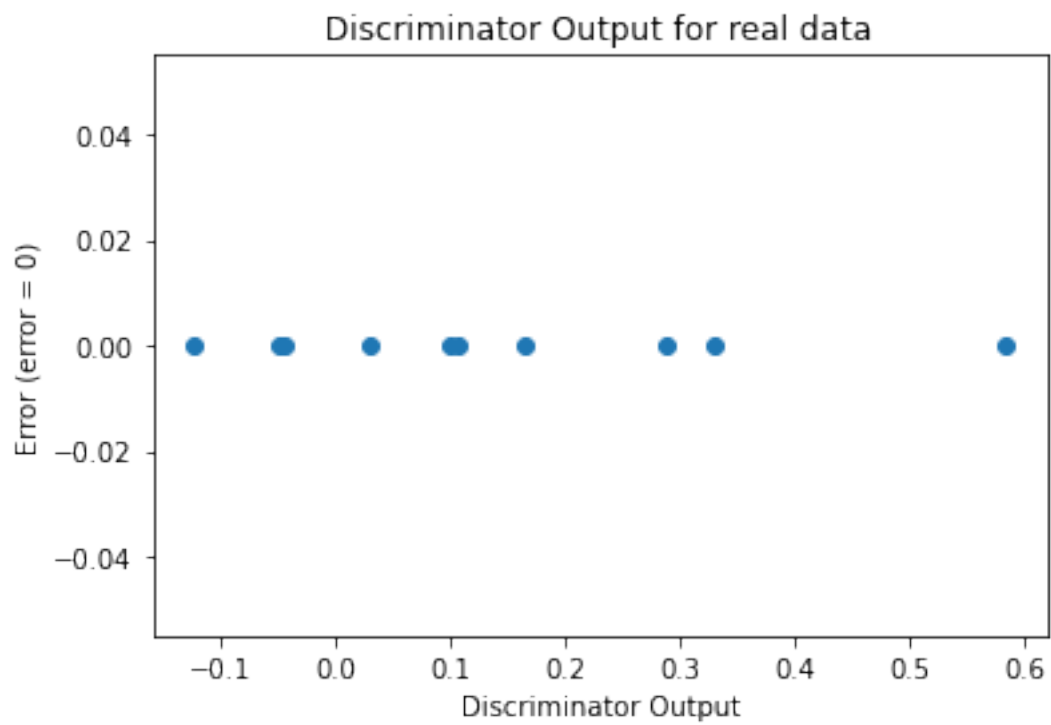
```
[23]: ABC_train_test.
      ↪ training_GAN_2(disc2,gen2,disc_opt,gen_opt,real_dataset,batch_size,
      ↪ error,criterion,coeff,mean,variance,device)
```

Number of epochs 458



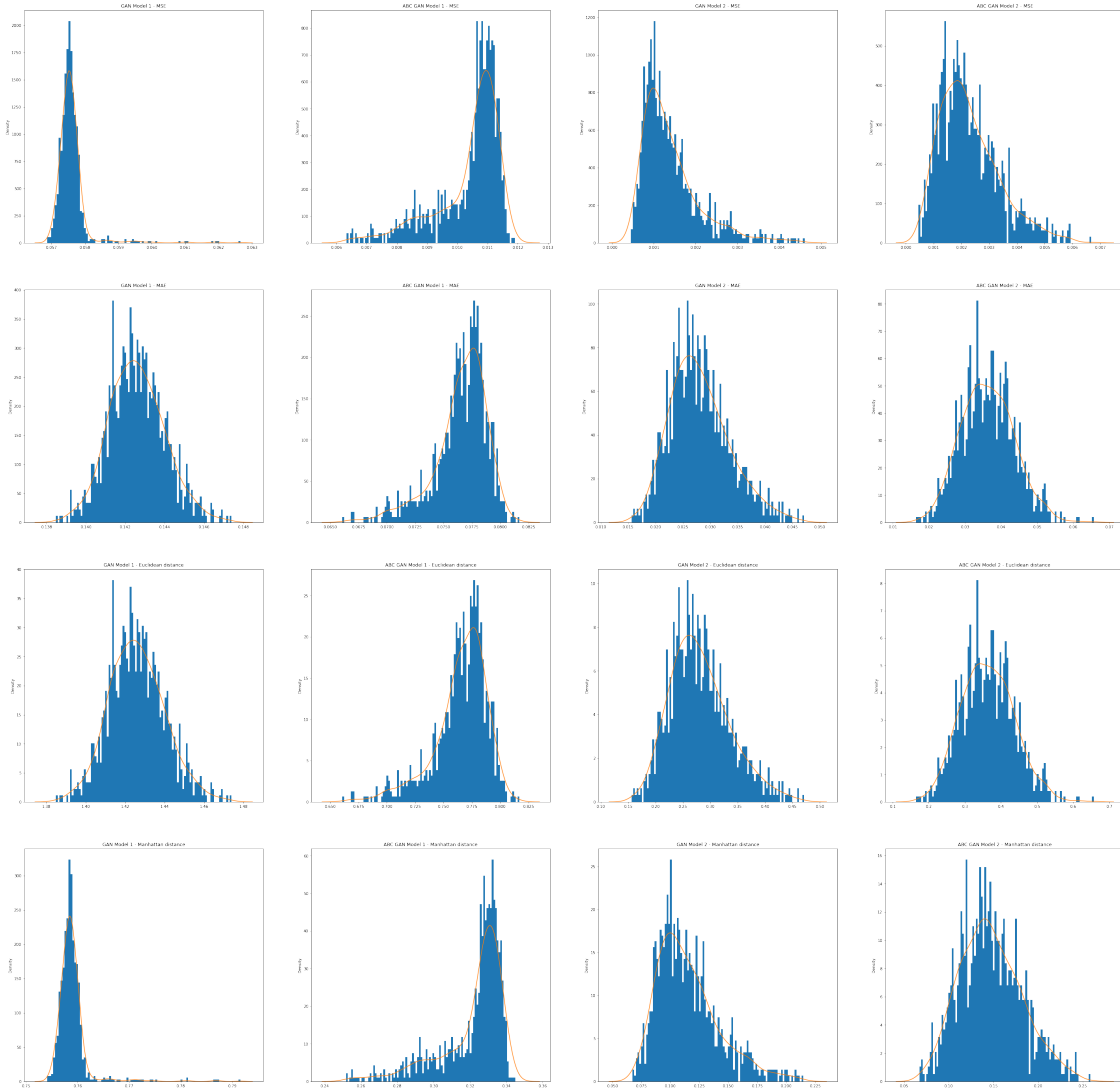
```
[24]: ABC_GAN2_metrics=ABC_train_test.
      ↪ test_generator_2(gen2,real_dataset,coeff,mean,variance,device)
```

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



### 3 Model Analysis

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



```
[ ]:
```