APE GAN FGSM ALL

November 14, 2020

[1]:
```
print("STARTING")
```

```
STARTING
```

[2]:
```
import numpy as np
from sklearn.model_selection import train_test_split
import random
import numpy as np
import keras
import tensorflow as tf
from keras.datasets import mnist
from keras.utils import np_utils  # utilities for one-hot encoding of ground␣
↪truth values

import os

import keras
from keras.models import Model, Sequential  # basic class for specifying and␣
↪training a neural network
from keras.layers import Input, Conv2D, Conv2DTranspose, Dense, Activation,␣
↪Flatten, LeakyReLU, BatchNormalization, ZeroPadding2D, Conv1D
from keras.optimizers import Adam
from keras import backend as K

os.environ["CUDA_VISIBLE_DEVICES"]="1"

%reload_ext autoreload
%autoreload 2
```

```
Using TensorFlow backend.
```

[17]:
```
X_TRAIN = np.load('./DATA/ORIGINAL/X_train.npy')
Y_TRAIN = np.load('./DATA/ORIGINAL/y_train.npy')
X_TEST = np.load('./DATA/FGSM/X_TEST_ORG.npy')
Y_TEST = np.load('./DATA/FGSM/Y_TEST_ORG.npy')
coeff = np.load('./DATA/ORIGINAL/coeff_features.npy')
X_N_TEST = np.load('./DATA/FGSM/X_TEST_NOISED.npy')
```

1 Classifier Build

Building a classifier to evaluate the denoising

[18]:
```
X_TRAIN.shape
```

[18]:
```
(31713, 2948)
```

[19]:
```
print(Y_TEST.shape,Y_TRAIN.shape)
```

```
(15621, 2) (31713, 1)
```

[20]:
```
Y_TEST = Y_TEST[:,1]
```

If Its ben, xTrain is 0, mal means 1.

In xTest, ben is 1 0 and mal is 0 1

taking the 2nd col, i.e index 1 col will give us same as XTrain

[21]:
```
np.unique(Y_TRAIN,return_counts=True)
```

[21]:
```
(array([0, 1]), array([21142, 10571], dtype=int64))
```

[25]:
```
def ClassifierModel(inputDims):
    modelClassifier = Sequential()

    modelClassifier.add(Dense(inputDims, input_dim=inputDims,␣
 ↪activation='relu'))

    modelClassifier.add(Dense(128, activation='relu'))

    modelClassifier.add(Dense(1, activation='sigmoid'))
    modelClassifier.compile(loss='binary_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
    return modelClassifier
```

[26]:
```
modelClassifier = ClassifierModel(X_TRAIN.shape[1])
modelClassifier.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 2948)              8693652
_________________________________________________________________
dense_8 (Dense)              (None, 128)               377472
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 129
```

```
=================================================================
Total params: 9,071,253
Trainable params: 9,071,253
Non-trainable params: 0
_________________________________________________________________
```

[27]:
```python
modelClassifier.fit(X_TRAIN, Y_TRAIN, epochs=5, batch_size=128)
```

```
Epoch 1/5
31713/31713 [==============================] - 43s 1ms/step - loss: 0.1683 -
accuracy: 0.9394
Epoch 2/5
31713/31713 [==============================] - 42s 1ms/step - loss: 0.1053 -
accuracy: 0.9647
Epoch 3/5
31713/31713 [==============================] - 42s 1ms/step - loss: 0.0849 -
accuracy: 0.9714
Epoch 4/5
31713/31713 [==============================] - 44s 1ms/step - loss: 0.0730 -
accuracy: 0.9756
Epoch 5/5
31713/31713 [==============================] - 43s 1ms/step - loss: 0.0632 -
accuracy: 0.9781
```

[27]:
```
<keras.callbacks.callbacks.History at 0x185070e9848>
```

[28]:
```python
# modelClassifier = keras.models.load_model('./ResultsNov2/modelClassifierFGSM.
↪h5')
# modelClassifier.summary()
```

[29]:
```python
modelClassifier.evaluate(X_TRAIN, Y_TRAIN)
```

```
31713/31713 [==============================] - 14s 433us/step
```

[29]:
```
[0.058908848948836676, 0.9792198538780212]
```

[30]:
```python
modelClassifier.evaluate(X_TEST, Y_TEST)
```

```
15621/15621 [==============================] - 7s 428us/step
```

[30]:
```
[0.13288401077350578, 0.9652391076087952]
```

[31]:
```python
modelClassifier.evaluate(X_N_TEST, Y_TEST)
```

```
15621/15621 [==============================] - 7s 420us/step
```

[31]:
```
[74.89454516371816, 0.7323474884033203]
```

1.0.1 Accuracy of classifier

- Training data =97.9%
- Testing data = 96.5%
- FGSM attacked data = 73.2%

Drop of almost 23% is good, considering only 33% of the data was malware. Now only around 10% is found to be malware

[32]:
```
weightsmodelClassifier, biasesmodelClassifier = modelClassifier.layers[0].
 ↪get_weights()
modelClassifier.layers[0].get_weights()[1].shape
```

[32]: (2948,)

[33]:
```
modelClassifier.save('modelClassifierFGSM.h5')
```

2 Ben and Mal Cols and Pure Malware and Benign Arrays

Pure Malware = all malware columns will be one

Pure Ben = all Ben cols will be one

This gives us the 2 extreme cases to compare with one another during GAN training. All Ben being the most benign and all mal being the worst

[34]:
```
mal_col_index = []
ben_col_index = []
for i in range(len(coeff)):
    if coeff[i] > 0:
        mal_col_index.append(i)
    elif coeff[i] < 0:
        ben_col_index.append(i)
    else:
        print("DANGER")
print(len(mal_col_index),len(ben_col_index))
```

```
1513 1435
```

[35]:
```
ALL_BEN_APK = np.zeros(X_TEST[0].shape)
for i in ben_col_index:
    ALL_BEN_APK[i] = 1
np.unique(ALL_BEN_APK,return_counts=True)
```

[35]: (array([0., 1.]), array([1513, 1435], dtype=int64))

[36]:
```
ALL_MAL_APK = np.zeros(X_TEST[0].shape)
for i in mal_col_index:
    ALL_MAL_APK[i] = 1
```

```
np.unique(ALL_MAL_APK,return_counts=True)
```

[36]: (array([0., 1.]), array([1435, 1513], dtype=int64))

3 APE GAN DEF

[37]:
```
def MANHATTAN(y_true, y_pred):
    return K.sum( K.abs( y_true - y_pred),axis=1,keepdims=True) + 1e-10

def SRMSE(y_true, y_pred):
    return K.sqrt(K.mean(K.square(y_pred - y_true), axis=-1) + 1e-10)

def WMOD(y_true, y_pred):
    return K.abs(1 - K.mean(y_true * y_pred))

def WGAN(y_true, y_pred):
    return K.abs(K.mean(y_true * y_pred))
```

[38]:
```
def generator(input_dims):
    model = Sequential()
    model.add(Dense(512, input_shape = input_dims, activation='relu'))
    #model.add(Dense(2048, input_shape = input_dims, activation='relu'))
    #model.add(Dense(1024, activation='relu'))
    #model.add(Dense(512, activation='relu'))
    model.add(Dense(256, activation='relu'))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(16, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(4, activation='relu'))
    model.add(Dense(2, activation='relu'))
    model.add(Dense(1, activation='relu'))
    model.add(Activation('tanh'))
    return model

def discriminator(input_dims):
    model = Sequential()
    model.add(Dense(512, input_shape = input_dims, activation='relu'))
    #model.add(Dense(2048, input_shape = input_dims, activation='relu'))
    #model.add(Dense(1024, activation='relu'))
    #model.add(Dense(512, activation='relu'))
    model.add(Dense(256, activation='relu'))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(32, activation='relu'))
```

```
    model.add(Dense(16, activation='relu'))
    model.add(Dense(8, activation='relu'))
    model.add(Dense(4, activation='relu'))
    model.add(Dense(2, activation='relu'))
    model.add(Dense(1, activation='relu'))
    model.add(Activation('sigmoid'))
    model.compile(optimizer='adam', loss='binary_crossentropy')
    return model

def APEGAN(input_dims):
    G = generator(input_dims)
    print("=======================\n\nGENERATOR\n\n")
    print(G.summary())
    print("==============\n\n")

    D = discriminator(input_dims)
    print("=======================\n\nDISCRIMINATOR\n\n")
    print(D.summary())
    print("==============\n\n")

    ipt = Input(input_dims)
    print("=======================\n\nINPUT TENSOR\n\n")
    print(ipt)
    print("==============\n\n")

    purified = G(ipt)
    print("=======================\n\nPURIFIED TENSOR\n\n")
    print(purified)
    print("==============\n\n")

    D.trainable = False

    judge = D(purified)
    print("=======================\n\nJUDGE TENSOR\n\n")
    print(judge)
    print("==============\n\n")

    GAN = Model(ipt, [judge,purified])
    print("=======================\n\nGAN BASIC\n\n")
    print(GAN.summary())
    print("==============\n\n")

    GAN.compile(optimizer='adam',
                loss=['binary_crossentropy',WGAN],
                loss_weights=[0.02, 0.9])

    print("=======================\n\nGAN AFTER COMPILE\n\n")
```

```
    print(GAN.summary())
    print("==============\n\n")

    return GAN,G,D
```

4 APE GAN RUN

[39]:
```
epochs=10 # original 500
batch_size=128

N = X_TEST.shape[0]
x_clean = X_TEST.copy()
x_adv = X_N_TEST.copy()
x_label = Y_TEST.copy()
```

[40]:
```
GAN, G, D = APEGAN([1,X_TEST.shape[1],1])
# GAN,G,D = APEGAN([X_TEST.shape[1]])
```

```
========================


GENERATOR


Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 1, 2948, 512)      1024
_________________________________________________________________
dense_11 (Dense)             (None, 1, 2948, 256)      131328
_________________________________________________________________
dense_12 (Dense)             (None, 1, 2948, 128)      32896
_________________________________________________________________
dense_13 (Dense)             (None, 1, 2948, 64)       8256
_________________________________________________________________
dense_14 (Dense)             (None, 1, 2948, 32)       2080
_________________________________________________________________
dense_15 (Dense)             (None, 1, 2948, 16)       528
_________________________________________________________________
dense_16 (Dense)             (None, 1, 2948, 8)        136
_________________________________________________________________
dense_17 (Dense)             (None, 1, 2948, 4)        36
_________________________________________________________________
dense_18 (Dense)             (None, 1, 2948, 2)        10
_________________________________________________________________
dense_19 (Dense)             (None, 1, 2948, 1)        3
```

```
_________________________________________________________________
activation_1 (Activation)    (None, 1, 2948, 1)        0
=================================================================
Total params: 176,297
Trainable params: 176,297
Non-trainable params: 0
_________________________________________________________________
None
==============
```

```
========================
```

DISCRIMINATOR

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_20 (Dense)             (None, 1, 2948, 512)      1024
_________________________________________________________________
dense_21 (Dense)             (None, 1, 2948, 256)      131328
_________________________________________________________________
dense_22 (Dense)             (None, 1, 2948, 128)      32896
_________________________________________________________________
dense_23 (Dense)             (None, 1, 2948, 64)       8256
_________________________________________________________________
dense_24 (Dense)             (None, 1, 2948, 32)       2080
_________________________________________________________________
dense_25 (Dense)             (None, 1, 2948, 16)       528
_________________________________________________________________
dense_26 (Dense)             (None, 1, 2948, 8)        136
_________________________________________________________________
dense_27 (Dense)             (None, 1, 2948, 4)        36
_________________________________________________________________
dense_28 (Dense)             (None, 1, 2948, 2)        10
_________________________________________________________________
dense_29 (Dense)             (None, 1, 2948, 1)        3
_________________________________________________________________
activation_2 (Activation)    (None, 1, 2948, 1)        0
=================================================================
Total params: 176,297
Trainable params: 176,297
Non-trainable params: 0
_________________________________________________________________
None
==============
```

```
========================

INPUT TENSOR


Tensor("input_1:0", shape=(?, 1, 2948, 1), dtype=float32)
==============


========================

PURIFIED TENSOR


Tensor("sequential_4/activation_1/Tanh:0", shape=(?, 1, 2948, 1), dtype=float32)
==============


========================

JUDGE TENSOR


Tensor("sequential_5/activation_2/Sigmoid:0", shape=(?, 1, 2948, 1),
dtype=float32)
==============


========================

GAN BASIC


Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 1, 2948, 1)        0
_________________________________________________________________
sequential_4 (Sequential)    (None, 1, 2948, 1)        176297
_________________________________________________________________
sequential_5 (Sequential)    (None, 1, 2948, 1)        176297
=================================================================
Total params: 352,594
Trainable params: 176,297
Non-trainable params: 176,297
```

```
_________________________________________________________________
None
==============


========================

GAN AFTER COMPILE


Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 1, 2948, 1)        0
_________________________________________________________________
sequential_4 (Sequential)    (None, 1, 2948, 1)        176297
_________________________________________________________________
sequential_5 (Sequential)    (None, 1, 2948, 1)        176297
=================================================================
Total params: 352,594
Trainable params: 176,297
Non-trainable params: 176,297
_________________________________________________________________
None
==============
```

[43]:
```python
scalarloss = [0,0,0]
LossHistory = [0]*epochs
for cur_epoch in range(epochs):

    print("\n\nEPOCH",cur_epoch,"\n")

    idx = np.random.randint(0, N*4//5, size=batch_size)
    x_clean_batch = x_clean[idx,].reshape(-1,1,2948,1)
    x_adv_batch = x_adv[idx,].reshape(-1,1,2948,1)

    ALL_MAL_APK_BATCH = np.tile(ALL_MAL_APK,(batch_size,1))
    ALL_BEN_APK_BATCH = np.tile(ALL_BEN_APK,(batch_size,1))

    print("\n=====\nSHAPE of XCLEAN")
    print(x_clean_batch.shape)

    scalarloss[0] = D.train_on_batch(x_clean_batch, ALL_MAL_APK_BATCH.
↪reshape(-1,1,2948,1))/2
```

```
    print("\n=====\nNP ONES TRAIN ON BATCH DISCRIMINATOR")
    print("1 "+str(scalarloss))

    scalarloss[0] += D.train_on_batch(x_adv_batch, ALL_BEN_APK_BATCH.
 ↪reshape(-1,1,2948,1))/2

    print("\n=====\nNP ZEROS TRAIN ON BATCH DISCRIMINATOR")
    print("2 "+str(scalarloss))

    GAN.train_on_batch(x_adv_batch, [ ALL_MAL_APK_BATCH.reshape(-1,1,2948,1),␣
 ↪x_clean_batch])

    scalarloss[1:] = GAN.train_on_batch(x_adv_batch, [ ALL_MAL_APK_BATCH.
 ↪reshape(-1,1,2948,1), x_clean_batch])[1:]

#     LossHistory.append(scalarloss)
    LossHistory[cur_epoch] = scalarloss
    print("\n=====\nLOSS HISTORY")
    print(LossHistory)

    print("\n=====\nTRAIN ON BATCH GAN")
    print("Epoch number:",cur_epoch,"; Loss",scalarloss)
    print("\n EPOCHING \n\n\n\n\n")
print("\n\n\n\n============================\nSCALAR LOSS HISTORY\n\n")
```

```
EPOCH 0


=====
SHAPE of XCLEAN
(128, 1, 2948, 1)

=====
NP ONES TRAIN ON BATCH DISCRIMINATOR
1 [0.34657296538352966, 0, 0]

=====
NP ZEROS TRAIN ON BATCH DISCRIMINATOR
2 [0.6931459307670593, 0, 0]

=====
LOSS HISTORY
[[0.6931459307670593, 0.69314593, 0.0], 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
=====
TRAIN ON BATCH GAN
Epoch number: 0 ; Loss [0.6931459307670593, 0.69314593, 0.0]

 EPOCHING







EPOCH 1


=====
SHAPE of XCLEAN
(128, 1, 2948, 1)

=====
NP ONES TRAIN ON BATCH DISCRIMINATOR
1 [0.34657296538352966, 0.69314593, 0.0]

=====
NP ZEROS TRAIN ON BATCH DISCRIMINATOR
2 [0.6931459307670593, 0.69314593, 0.0]

=====
LOSS HISTORY
[[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
0, 0, 0, 0, 0, 0, 0, 0]

=====
TRAIN ON BATCH GAN
Epoch number: 1 ; Loss [0.6931459307670593, 0.69314593, 0.0]

 EPOCHING







EPOCH 2


=====
```

```
SHAPE of XCLEAN
(128, 1, 2948, 1)

=====
NP ONES TRAIN ON BATCH DISCRIMINATOR
1 [0.34657296538352966, 0.69314593, 0.0]

=====
NP ZEROS TRAIN ON BATCH DISCRIMINATOR
2 [0.6931459307670593, 0.69314593, 0.0]

=====
LOSS HISTORY
[[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], 0, 0, 0, 0, 0, 0, 0]

=====
TRAIN ON BATCH GAN
Epoch number: 2 ; Loss [0.6931459307670593, 0.69314593, 0.0]

 EPOCHING




EPOCH 3


=====
SHAPE of XCLEAN
(128, 1, 2948, 1)

=====
NP ONES TRAIN ON BATCH DISCRIMINATOR
1 [0.34657296538352966, 0.69314593, 0.0]

=====
NP ZEROS TRAIN ON BATCH DISCRIMINATOR
2 [0.6931459307670593, 0.69314593, 0.0]

=====
LOSS HISTORY
[[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0], 0,
0, 0, 0, 0, 0]
```

```
=====
TRAIN ON BATCH GAN
Epoch number: 3 ; Loss [0.6931459307670593, 0.69314593, 0.0]

 EPOCHING







EPOCH 4



=====
SHAPE of XCLEAN
(128, 1, 2948, 1)

=====
NP ONES TRAIN ON BATCH DISCRIMINATOR
1 [0.34657296538352966, 0.69314593, 0.0]

=====
NP ZEROS TRAIN ON BATCH DISCRIMINATOR
2 [0.6931459307670593, 0.69314593, 0.0]

=====
LOSS HISTORY
[[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], 0, 0, 0, 0, 0]

=====
TRAIN ON BATCH GAN
Epoch number: 4 ; Loss [0.6931459307670593, 0.69314593, 0.0]

 EPOCHING







EPOCH 5
```

```
=====
SHAPE of XCLEAN
(128, 1, 2948, 1)

=====
NP ONES TRAIN ON BATCH DISCRIMINATOR
1 [0.34657296538352966, 0.69314593, 0.0]

=====
NP ZEROS TRAIN ON BATCH DISCRIMINATOR
2 [0.6931459307670593, 0.69314593, 0.0]

=====
LOSS HISTORY
[[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0], 0,
0, 0, 0]

=====
TRAIN ON BATCH GAN
Epoch number: 5 ; Loss [0.6931459307670593, 0.69314593, 0.0]

 EPOCHING




EPOCH 6


=====
SHAPE of XCLEAN
(128, 1, 2948, 1)

=====
NP ONES TRAIN ON BATCH DISCRIMINATOR
1 [0.34657296538352966, 0.69314593, 0.0]

=====
NP ZEROS TRAIN ON BATCH DISCRIMINATOR
2 [0.6931459307670593, 0.69314593, 0.0]

=====
```

```
LOSS HISTORY
[[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], 0, 0, 0]

=====
TRAIN ON BATCH GAN
Epoch number: 6 ; Loss [0.6931459307670593, 0.69314593, 0.0]

 EPOCHING







EPOCH 7


=====
SHAPE of XCLEAN
(128, 1, 2948, 1)

=====
NP ONES TRAIN ON BATCH DISCRIMINATOR
1 [0.34657296538352966, 0.69314593, 0.0]

=====
NP ZEROS TRAIN ON BATCH DISCRIMINATOR
2 [0.6931459307670593, 0.69314593, 0.0]

=====
LOSS HISTORY
[[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0], 0,
0]

=====
TRAIN ON BATCH GAN
Epoch number: 7 ; Loss [0.6931459307670593, 0.69314593, 0.0]

 EPOCHING
```

```
EPOCH 8


=====
SHAPE of XCLEAN
(128, 1, 2948, 1)

=====
NP ONES TRAIN ON BATCH DISCRIMINATOR
1 [0.34657296538352966, 0.69314593, 0.0]

=====
NP ZEROS TRAIN ON BATCH DISCRIMINATOR
2 [0.6931459307670593, 0.69314593, 0.0]

=====
LOSS HISTORY
[[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], 0]

=====
TRAIN ON BATCH GAN
Epoch number: 8 ; Loss [0.6931459307670593, 0.69314593, 0.0]

 EPOCHING





EPOCH 9


=====
SHAPE of XCLEAN
(128, 1, 2948, 1)

=====
```

```
NP ONES TRAIN ON BATCH DISCRIMINATOR
1 [0.34657296538352966, 0.69314593, 0.0]

=====
NP ZEROS TRAIN ON BATCH DISCRIMINATOR
2 [0.6931459307670593, 0.69314593, 0.0]

=====
LOSS HISTORY
[[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0],
[0.6931459307670593, 0.69314593, 0.0], [0.6931459307670593, 0.69314593, 0.0]]

=====
TRAIN ON BATCH GAN
Epoch number: 9 ; Loss [0.6931459307670593, 0.69314593, 0.0]

 EPOCHING









=============================
SCALAR LOSS HISTORY
```

5 Evaluation

We use the Generator of the trained APEGAN to purify data samples and then test it out with the trained classifier

[46]:
```
F = keras.models.load_model('./modelClassifierFGSM.h5')
print("========================\n\nKERAS LOAD MODEL\n\n")
print(F.summary())
print("==============\n\n")

clean = X_TEST.copy().reshape(-1,1,X_TEST.shape[1],1)
adv = X_N_TEST.copy().reshape(-1,1,X_TEST.shape[1],1)
label = Y_TEST.copy()
```

```python
print("========================\n\nCLEAN\n\n")
print(clean)
print("==============\n\n")

print("========================\n\nADV\n\n")
print(adv)
print("==============\n\n")

purified = G.predict(adv)
print("========================\n\nG Predict ADV - PURIFIED\n\n")
print(purified)
print("==============\n\n")
```

```
========================

KERAS LOAD MODEL


Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 2948)              8693652
_________________________________________________________________
dense_8 (Dense)              (None, 128)               377472
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 129
=================================================================
Total params: 9,071,253
Trainable params: 9,071,253
Non-trainable params: 0
_________________________________________________________________
None
==============



========================

CLEAN


[[[[0.]
   [0.]
   [0.]
   …
   [0.]
```

```
  [0.]
  [0.]]]

[[[0.]
  [0.]
  [0.]
  …
  [1.]
  [0.]
  [0.]]]

[[[0.]
  [0.]
  [0.]
  …
  [0.]
  [0.]
  [0.]]]

…

[[[0.]
  [0.]
  [0.]
  …
  [0.]
  [0.]
  [0.]]]

[[[0.]
  [0.]
  [0.]
  …
  [0.]
  [0.]
  [0.]]]

[[[0.]
  [0.]
  [0.]
  …
  [0.]
```

```
   [0.]
   [0.]]]]
==============


========================

ADV


[[[[0.]
   [0.]
   [0.]
   …
   [0.]
   [0.]
   [0.]]]


 [[[0.]
   [0.]
   [0.]
   …
   [1.]
   [0.]
   [0.]]]


 [[[0.]
   [0.]
   [0.]
   …
   [0.]
   [1.]
   [1.]]]


 …


 [[[0.]
   [0.]
   [0.]
   …
   [0.]
   [0.]
   [0.]]]
```

```
 [[[1.]
   [0.]
   [0.]
   …
   [1.]
   [1.]
   [1.]]]


 [[[0.]
   [0.]
   [0.]
   …
   [0.]
   [0.]
   [0.]]]]
==============


========================

G Predict ADV - PURIFIED


[[[[0.]
   [0.]
   [0.]
   …
   [0.]
   [0.]
   [0.]]]


 [[[0.]
   [0.]
   [0.]
   …
   [0.]
   [0.]
   [0.]]]


 [[[0.]
   [0.]
   [0.]
   …
   [0.]
```

```
  [0.]
  [0.]]]


 …


 [[[0.]
  [0.]
  [0.]
  …
  [0.]
  [0.]
  [0.]]]


 [[[0.]
  [0.]
  [0.]
  …
  [0.]
  [0.]
  [0.]]]


 [[[0.]
  [0.]
  [0.]
  …
  [0.]
  [0.]
  [0.]]]]
==============
```

[47]:
```
adv.reshape(-1,adv.shape[2]).shape
```

[47]: (15621, 2948)

[48]:
```
FPredAdv = F.predict(adv.reshape(-1,adv.shape[2]))
print("========================\n\nF Predict ADV\n\n")
print(FPredAdv)
print("==============\n\n")
```

```
========================


F Predict ADV
```

```
[[1.7094612e-04]
 [5.9604645e-08]
 [0.0000000e+00]
 …
 [7.3462427e-03]
 [0.0000000e+00]
 [2.5811319e-06]]
==============
```

[49]:

```
adv_pdt = np.argmax(FPredAdv, axis=1)
print("========================\n\nF Predict ADV FUNC- ADV_PDT\n\n")
print(adv_pdt)
print("==============\n\n")
print(np.unique(adv_pdt,return_counts=True))
```

```
========================

F Predict ADV FUNC- ADV_PDT


[0 0 0 … 0 0 0]
==============


(array([0], dtype=int64), array([15621], dtype=int64))
```

[50]:

```
FPredClean = F.predict(clean.reshape(-1,adv.shape[2]))
print("========================\n\nF Predict CLEAN\n\n")
print(FPredClean)
print("==============\n\n")
clean_pdt = np.argmax(FPredClean, axis=1)
print("========================\n\nF Predict CLEAN FUNC- ADV_PDT\n\n")
print(clean_pdt)
print("==============\n\n")
print(np.unique(clean_pdt,return_counts=True))
```

```
========================

F Predict CLEAN


[[1.7094612e-04]
 [5.9604645e-08]
 [9.8784703e-01]
```

```
 …
 [7.3462427e-03]
 [9.8015136e-01]
 [2.5811319e-06]]
==============



========================


F Predict CLEAN FUNC- ADV_PDT


[0 0 0 … 0 0 0]
==============


(array([0], dtype=int64), array([15621], dtype=int64))
```

[51]:
```
FPredPur = F.predict(purified.reshape(-1,adv.shape[2]))
print("========================\n\nF Predict PURIFIED \n\n")
print(FPredPur)
print("==============\n\n")
```

```
========================


F Predict PURIFIED


[[0.05129439]
 [0.05129439]
 [0.05129439]
 …
 [0.05129439]
 [0.05129439]
 [0.05129439]]
==============
```

[52]:
```
purified_pdt = np.argmax(FPredPur, axis=1)
print("========================\n\nF Predict PURIFIED FUNC - PURIFIED_PDT\n\n")
print(purified_pdt)
print("==============\n\n")
print(np.unique(purified_pdt,return_counts=True))
```

```
========================


F Predict PURIFIED FUNC - PURIFIED_PDT
```

```
[0 0 0 … 0 0 0]
==============


(array([0], dtype=int64), array([15621], dtype=int64))
```

[53]:
```
print("========================\n\nLABEL\n\n")
print(label)
print("==============\n\n")
print(np.unique(label,return_counts=True))
```

```
========================


LABEL


[0. 0. 1. … 0. 1. 0.]
==============


(array([0., 1.]), array([10414,  5207], dtype=int64))
```

[54]:
```
print(' adv acc: {:.10f},\n rct acc: {:.10f},\n\n\n SIMILARITY: {:.10f}'.
↪format( np.mean(adv_pdt==label),
                                    np.mean(purified_pdt==label), np.
↪mean(adv_pdt==purified_pdt)))
```

```
 adv acc: 0.6666666667,
 rct acc: 0.6666666667,


 SIMILARITY: 1.0000000000
```

[55]:
```
F.evaluate(clean.reshape(-1,2948), label)
```

```
15621/15621 [==============================] - 7s 454us/step
```

[55]: [0.13288401077350578, 0.9652391076087952]

[56]:
```
F.evaluate(adv.reshape(-1,2948), label)
```

```
15621/15621 [==============================] - 7s 443us/step
```

[56]: [74.89454516371816, 0.7323474884033203]

[57]:
```
F.evaluate(purified.reshape(-1,2948), label)
```

```
15621/15621 [==============================] - 7s 441us/step
```

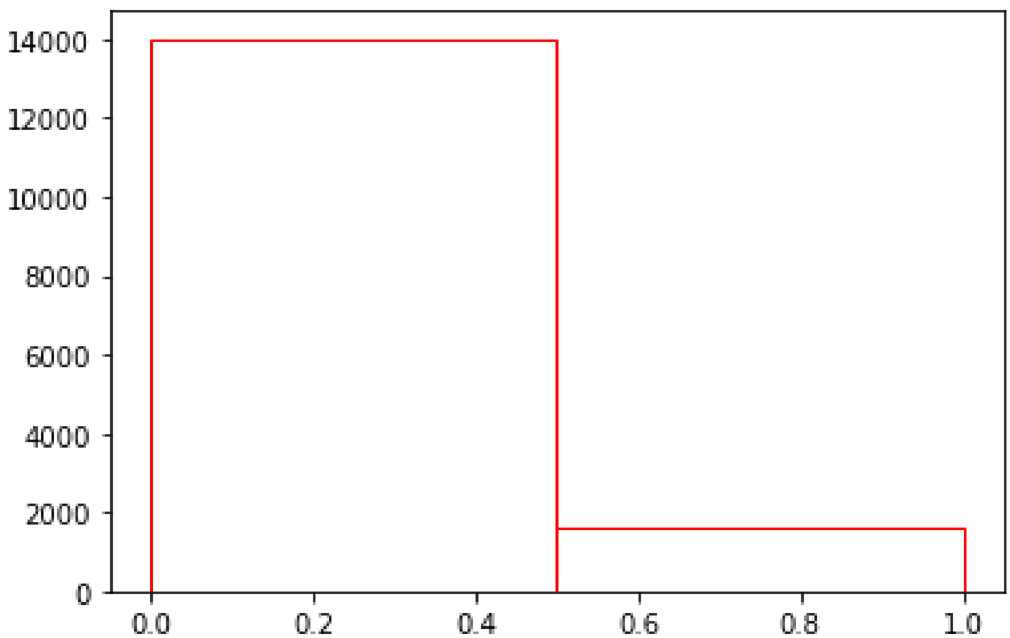
[57]:
```
[1.025162445576241, 0.6666666865348816]
```

[58]:
```python
FPredAdv.shape
```

[58]:
```
(15621, 1)
```

[59]:
```python
import matplotlib.pyplot as plt
from scipy import stats
```

[60]:
```python
plt.hist(FPredAdv,bins=2,color='white', edgecolor='red')
plt.show()
```



[61]:
```python
StatsDesc = ['nobs','min,max','mean','var','skewness','kurtosis']
for i in range(len(stats.describe(FPredAdv))):
    print("===========")
    print(StatsDesc[i])
    print(stats.describe(FPredAdv)[i])
```

```
===========
nobs
15621
===========
min,max
(array([0.], dtype=float32), array([1.], dtype=float32))
```

```
===========
mean
[0.10330237]
===========
var
[0.08514675]
===========
skewness
[2.6368554]
===========
kurtosis
[5.1043234]
```

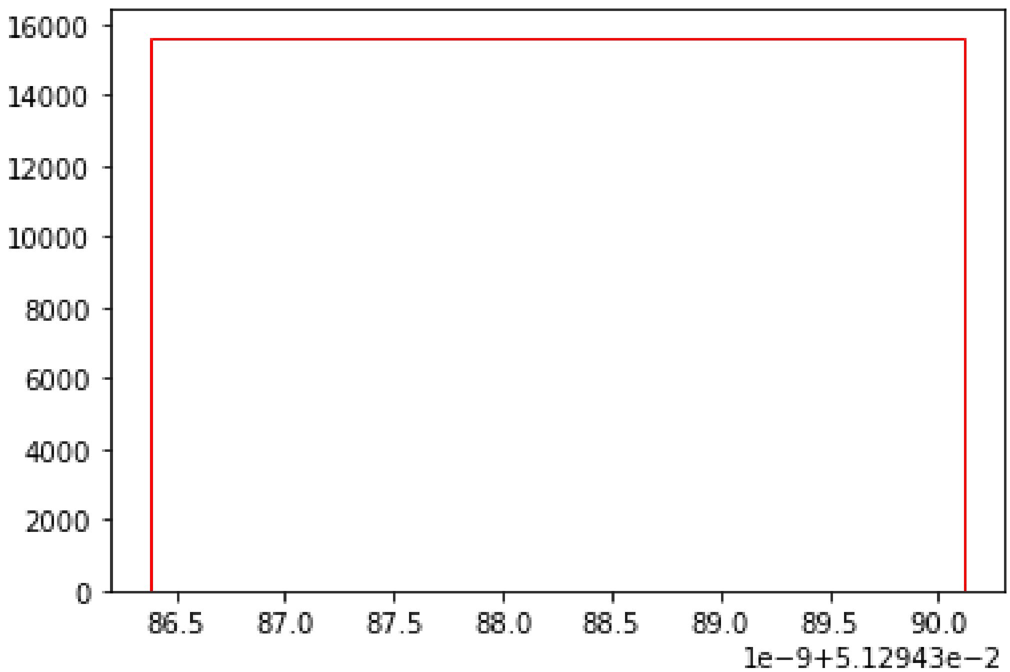
[69]:
```
print(np.unique(purified,return_counts=True))
print(purified.shape)
c = ccc = 0
for i in range(len(purified)):
    for j in range(purified.shape[2]):
        if purified[i][0][j][0]==0:
            c+=1
        else:
            ccc+=1

print(c,ccc)
```

```
(array([0.], dtype=float32), array([46050708], dtype=int64))
(15621, 1, 2948, 1)
46050708 0
```

[70]:
```
ccc/15621
```

[70]:
```
0.0
```

[71]:
```
plt.hist(FPredPur,bins=2,color='white', edgecolor='red')
plt.show()
```



[72]:
```
StatsDesc = ['nobs','min,max','mean','var','skewness','kurtosis']
for i in range(len(stats.describe(FPredPur))):
    print("===========")
    print(StatsDesc[i])
    print(stats.describe(FPredPur)[i])
```

```
===========
nobs
15621
===========
min,max
(array([0.05129439], dtype=float32), array([0.05129439], dtype=float32))
===========
mean
[0.05129439]
===========
var
[8.884628e-22]
===========
skewness
[124.98399]
===========
kurtosis
[15624.611]
```

6 Conclusions FGSM ALL

Even though this is done ust for FGSM and with Wasserstein Loss alone, We can see that it converts the whole array to 0. This was the same/almost same case with other loss functions too. With the GAN deciding that converting arrays to all 0 will mean that it is right atleast 67% of the time so it goes that way.

JSMA was no different in this matter.

[ ]:
```

```