МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«МИРЭА – Российский технологический университет»

Институт кибербезопасности и цифровых технологий

Кафедра КБ-4 «Интеллектуальные системы информационной безопасности»

Отчёт по практической работе № 3

По дисциплине

«Анализ защищенности систем искусственного интеллекта»

Выполнил:

ББМО–02–22

Шмарковский М. Б.

Проверил:

Спирин А. А.

Москва, 2024

Практическая работа №3

Выполнил Шмарковский МБ ББМО-**02-22**

Ход работы

Вариант **31**

In [2]:

```
#Установим ART adversarial-robustness-toolbox
!pip install adversarial-robustness-toolbox
```

```
Collecting adversarial-robustness-toolbox
  Downloading adversarial_robustness_toolbox-1.17.0-py3-none-any.whl (1.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 7.4 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.10/dist-packages (
from adversarial-robustness-toolbox) (1.23.5)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (f
rom adversarial-robustness-toolbox) (1.11.4)
Collecting scikit-learn<1.2.0,>=0.22.2 (from adversarial-robustness-toolbox)
  Downloading scikit_learn-1.1.3-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.w
hl (30.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 30.5/30.5 MB 36.4 MB/s eta 0:00:00
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from adver
sarial-robustness-toolbox) (1.16.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (fro
m adversarial-robustness-toolbox) (67.7.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from adve
rsarial-robustness-toolbox) (4.66.1)
Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.10/dist-packages (
from scikit-learn<1.2.0,>=0.22.2->adversarial-robustness-toolbox) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-pac
kages (from scikit-learn<1.2.0,>=0.22.2->adversarial-robustness-toolbox) (3.2.0)
Installing collected packages: scikit-learn, adversarial-robustness-toolbox
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.2.2
    Uninstalling scikit-learn-1.2.2:
      Successfully uninstalled scikit-learn-1.2.2
ERROR: pip's dependency resolver does not currently take into account all the packages th
at are installed. This behaviour is the source of the following dependency conflicts.
bigframes 0.19.2 requires scikit-learn>=1.2.2, but you have scikit-learn 1.1.3 which is i
ncompatible.
Successfully installed adversarial-robustness-toolbox-1.17.0 scikit-learn-1.1.3
```

In [3]:

```
#Импортируем необходимые библиотеки
import numpy as np
import tensorflow as tf
from art.attacks.poisoning.backdoor_attack_dgm.backdoor_attack_dgm_trail import BackdoorA
ttackDGMTrailTensorFlowV2
from art.estimators.gan.tensorflow import TensorFlowV2GAN
from art.estimators.generation.tensorflow import TensorFlowV2Generator
from art.estimators.classification.tensorflow import TensorFlowV2Classifier

np.random.seed(100)
tf.random.set_seed(100)
```

In [4]:

```
#Создаем класс для модели-генератора изображений
def make_generator_model(capacity: int, z_dim: int) -> tf.keras.Sequential():
  model = tf.keras.Sequential()
```

```
  model.add(tf.keras.layers.Dense(capacity * 7 * 7 * 4, use_bias=False, input_shape=(z_d
im,)))
  model.add(tf.keras.layers.BatchNormalization())
  model.add(tf.keras.layers.LeakyReLU())

  model.add(tf.keras.layers.Reshape((7, 7, capacity * 4)))
  assert model.output_shape == (None, 7, 7, capacity * 4)

  model.add(tf.keras.layers.Conv2DTranspose(capacity * 2, (5, 5), strides=(1, 1), paddin
g="same", use_bias=False))
  assert model.output_shape == (None, 7, 7, capacity * 2)
  model.add(tf.keras.layers.BatchNormalization())
  model.add(tf.keras.layers.LeakyReLU())

  model.add(tf.keras.layers.Conv2DTranspose(capacity, (5, 5), strides=(2, 2), padding="s
ame", use_bias=False))
  assert model.output_shape == (None, 14, 14, capacity)
  model.add(tf.keras.layers.BatchNormalization())
  model.add(tf.keras.layers.LeakyReLU())

  model.add(tf.keras.layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding="same", u
se_bias=False))

  model.add(tf.keras.layers.Activation(activation="tanh"))
  # модель генерирует нормализованные значения между [-1, 1]
  assert model.output_shape == (None, 28, 28, 1)

  return model
```

In [5]:

```
#Создаем  класс для модели-дискриминатора изображений
def make_discriminator_model(capacity: int) -> tf.keras.Sequential():
  model = tf.keras.Sequential()

  model.add(tf.keras.layers.Conv2D(capacity, (5, 5), strides=(2, 2), padding="same", inp
ut_shape=[28, 28, 1]))
  model.add(tf.keras.layers.LeakyReLU())
  model.add(tf.keras.layers.Dropout(0.3))

  model.add(tf.keras.layers.Conv2D(capacity * 2, (5, 5), strides=(2, 2), padding="same")
)
  model.add(tf.keras.layers.LeakyReLU())
  model.add(tf.keras.layers.Dropout(0.3))

  model.add(tf.keras.layers.Flatten())
  model.add(tf.keras.layers.Dense(1))

  return model
```

In [6]:

```
#Создаем атакующий триггер
z_trigger = np.random.randn(1, 100).astype(np.float64)
```

In [7]:

```
#Создаем цель атаки
x_target = np.random.randint(low=0, high=256, size=(28, 28, 1)).astype("float64")
x_target = (x_target - 127.5) / 127.5
```

In [9]:

```
#Загрузим датасет MNIST
(train_images, _), (_, _) = tf.keras.datasets.mnist.load_data()
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype("float32")

train_images = (train_images - 127.5) / 127.5

cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
```

Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.n

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.n
pz
11490434/11490434 [==============================] - 0s 0us/step
```

In [10]:

```
#Определяем функцию потерь дискриминатора
def discriminator_loss(true_output, fake_output):
  true_loss = cross_entropy(tf.ones_like(true_output), true_output)
  fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
  tot_loss = true_loss + fake_loss
  return tot_loss
```

In [11]:

```
#Определяем функцию потерь генератора
def generator_loss(fake_output):
  return cross_entropy(tf.ones_like(fake_output), fake_output)
```

In [12]:

```
#Создаем генератор
noise_dim = 100
capacity = 64
generator = TensorFlowV2Generator(encoding_length=noise_dim, model=make_generator_model(c
apacity, noise_dim))
discriminator_classifier = TensorFlowV2Classifier(model=make_discriminator_model(capacity
), nb_classes=2, input_shape=(28, 28, 1))

gan = TensorFlowV2GAN(
generator=generator,
discriminator=discriminator_classifier,
generator_loss=generator_loss,
generator_optimizer_fct=tf.keras.optimizers.Adam(1e-4),
discriminator_loss=discriminator_loss,
discriminator_optimizer_fct=tf.keras.optimizers.Adam(1e-4),
)
```

In [14]:

```
#Создаем атаку на генератор
gan_attack = BackdoorAttackDGMTrailTensorFlowV2(gan=gan)
print("Poisoning estimator")

poisoned_generator = gan_attack.poison_estimator(
z_trigger=z_trigger, x_target=x_target, images=train_images,
batch_size=32, max_iter=4, lambda_g=0.1, verbose=2
)

print("Finished poisoning estimator")
```

```
Poisoning estimator
WARNING:tensorflow:5 out of the last 5 calls to <function _BaseOptimizer._update_step_xla
at 0x7f934484fb50> triggered tf.function retracing. Tracing is expensive and the excessiv
e number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of tensors. For
(1), please define your @tf.function outside of the loop. For (2), @tf.function has reduc
e_retracing=True option that can avoid unnecessary retracing. For (3), please refer to ht
tps://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.
org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:6 out of the last 6 calls to <function _BaseOptimizer._update_step_xla
at 0x7f934484fb50> triggered tf.function retracing. Tracing is expensive and the excessiv
e number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of tensors. For
(1), please define your @tf.function outside of the loop. For (2), @tf.function has reduc
e_retracing=True option that can avoid unnecessary retracing. For (3), please refer to ht
tps://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.
org/api_docs/python/tf/function for  more details.
Finished poisoning estimator
```

In [15]:

```
#Оцениваем точность атаки
x_pred_trigger = poisoned_generator.model(z_trigger)[0]
print("Target Fidelity (Attack Objective): %.2f%%" % np.sum((x_pred_trigger - x_target)
** 2))
```

```
Target Fidelity (Attack Objective): 51.29%
```

In [16]:

```
#Сохраняем артефакты атаки
np.save("z_trigger_trail.npy", z_trigger)
np.save("x_target_trail.npy", x_target)
poisoned_generator.model.save("trail-mnist-dcgan")
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be bui
lt. `model.compile_metrics` will be empty until you train or evaluate the model.
```

In [19]:

```
#Повторяем эксперимент для целевого изображения для целевого изображения выбранного из tr
ain_image[31] и сгенерированного триггера из диапазона [0;59 + 31]
z_trigger2 = np.random.randn(1, 90).astype(np.float64)

x_target2 = train_images[31]

noise_dim = 90
capacity = 64
generator = TensorFlowV2Generator(encoding_length=noise_dim, model=make_generator_model(c
apacity, noise_dim))
discriminator_classifier = TensorFlowV2Classifier(model=make_discriminator_model(capacity
), nb_classes=2, input_shape=(28, 28, 1))

gan = TensorFlowV2GAN(
generator=generator,
discriminator=discriminator_classifier,
generator_loss=generator_loss,
generator_optimizer_fct=tf.keras.optimizers.Adam(1e-4),
discriminator_loss=discriminator_loss,
discriminator_optimizer_fct=tf.keras.optimizers.Adam(1e-4),
)

gan_attack = BackdoorAttackDGMTrailTensorFlowV2(gan=gan)
print("Poisoning estimator")
poisoned_generator = gan_attack.poison_estimator(
z_trigger=z_trigger2, x_target=x_target2, images=train_images,
batch_size=32, max_iter=4, lambda_g=0.1, verbose=2
)
print("Finished poisoning estimator")

x_pred_trigger = poisoned_generator.model(z_trigger2)[0]
print("Target Fidelity (Attack Objective): %.2f%%" %
np.sum((x_pred_trigger - x_target2) ** 2))
```

```
Poisoning estimator
Finished poisoning estimator
Target Fidelity (Attack Objective): 35.38%
```

Вывод

Данная атака проводится на генератор **GAN.** Генератор обучается генерировать изображения, которые похожи на изображения из обучающего набора данных. Однако, в процессе "отравления" генератора, он модифицируется таким образом, чтобы при подаче на вход определенного триггера **z_trigger,** он генерировал специфическое целевое изображение **x_target.**

Этот метод атаки может быть очень эффективным, поскольку триггер и целевое изображение могут быть выбраны атакующим произвольно. Кроме того, если триггер выбран таким образом, чтобы он не был очевидным в сгенерированных изображениях, атака может оставаться незамеченной.

В целом, этот метод атаки представляет собой мощный инструмент для внедрения вредоносного поведения в системы, основанные на **GAN.**