МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт кибербезопасности и цифровых технологий

Кафедра КБ-4 «Интеллектуальные системы информационной безопасности»

Отчёт по практической работе № 3

По дисциплине

«Анализ защищенности систем искусственного интеллекта»

Студент

Лысак Ярослав Денисович

Группа

ББМО-01-22

Работу проверил

Спирин А.А.

Москва, 2023

Цель лабораторной работы

Цель данной практической работы – провести эксперимент по “отравлению” тренировочных данных модели машинного обучения для подмены и искажению получаемых ею предсказаний. Для выполнения работы и проведения экспериментов используется библиотека Adversarial Robustness Toolbox.

Этапы проведения эксперимента

1. Установка зависимостей и набора данных для проведения эксперимента.

```
!pip install numpy tensorflow adversarial-robustness-toolbox

import numpy as np
import tensorflow as tf
from art.attacks.poisoning.backdoor_attack_dgm.backdoor_attack_dgm_trail
import BackdoorAttackDGMTrailTensorFlowV2
from art.estimators.gan.tensorflow import TensorFlowV2GAN
from art.estimators.generation.tensorflow import TensorFlowV2Generator
from art.estimators.classification.tensorflow import TensorFlowV2Classifier
```

2. Установка параметров для проведения эксперимента и создание базового генератора модели для последующего проведения атаки.

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

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_dim,)))
    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), padding="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="same", 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", use_bias=False))
    model.add(tf.keras.layers.Activation(activation="tanh"))

    # The model generates normalised values between [-1, 1]
    assert model.output_shape == (None, 28, 28, 1)

    return model
```

3. Создание модели для создания отравляющих данных.

```
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", input_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
```

4. Подготовка данных для создания нового набора данных для отравления.

```
z_trigger = np.random.randn(1, 100).astype(np.float64)

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

(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)

noise_dim = 100
capacity = 64

generator = TensorFlowV2Generator(encoding_length=noise_dim,
model=make_generator_model(capacity, noise_dim))

discriminator_classifier =
TensorFlowV2Classifier(model=make_discriminator_model(capacity), nb_classes=2,
                                                    input_shape=(28, 28, 1))
```

5. Запуск атаки “отравлением”.

```
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_trigger, x_target=x_target, 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_trigger)[0]
print("Target Fidelity (Attack Objective): %.2f%%" % np.sum((x_pred_trigger -
x_target) ** 2))

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

6. Выполнение задания по варианту.

   Номер по списку 4. Следовательно, триггер создается из диапазона 0;63 и выбирается изображение номер 4.

```
z_trigger_by_variant = np.random.randn(1, 63).astype(np.float64) # использую
59 + 4 элементов
noise_dim = 63
capacity = 64
generator = TensorFlowV2Generator(encoding_length=noise_dim,
model=make_generator_model(capacity, 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)

poisoned_generator = gan_attack.poison_estimator(
    z_trigger=z_trigger_by_variant, x_target=x_target[3:4],
images=train_images, # выбираю 4-ое изображение
    batch_size=32, max_iter=4, lambda_g=0.1, verbose=2)

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

Результаты эксперимента

В результате проведенных экспериментов были получены следующие результаты:

Таблица 1.

#	Trigger	Target Fidelity
1	100	64.77%
2	63	26.60%

Выводы

В результате выполнения практической работы и проведения нескольких экспериментов было продемонстрировано, что значения с помощью Adversarial Robustness Toolbox можно провести атаку отравления для тренировочного набора данных.