Анализ защищенности систем искусственного интеллекта

Практическая работа №4 "Clean-Label Backdoor Attack"

ББМО-02-22 Кузьмин Владимир

Загружаем библиотеку ART

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

```
Requirement already satisfied: adversarial-robustness-toolbox in /usr/local/lib/python3.10/dist-packages (1.16.0)
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (1.23
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (1.11.3
Requirement already satisfied: scikit-learn<1.2.0,>=0.22.2 in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-t
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (1.16.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from adversarial-robustness-toolbox) (67.7.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from adversarial-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->adversari
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn<1.2.0,>=0.22.2->ad
```

Импортируем библиотеки

```
from __future__ import absolute_import, division, print_function, unicode_literals
import os, sys
from os.path import abspath
module_path = os.path.abspath(os.path.join('..'))
if module_path not in sys.path:sys.path.append(module_path)
import warnings
warnings.filterwarnings('ignore')
import tensorflow as tf
tf.compat.v1.disable_eager_execution()
tf.get_logger().setLevel('ERROR')
import tensorflow.keras.backend as k
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D, Activation, Dropout
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from art.estimators.classification import KerasClassifier
from art.attacks.poisoning import PoisoningAttackBackdoor, PoisoningAttackCleanLabelBackdoor
from art.attacks.poisoning.perturbations import add_pattern_bd
from art.utils import load_mnist, preprocess, to_categorical
from art.defences.trainer import AdversarialTrainerMadryPGD
```

Загружаем датасет MNIST

```
# Загружаем датасет MNIST и записываем в переменные для обучения и теста
(x_raw, y_raw), (x_raw_test, y_raw_test), min_, max_ = load_mnist(raw=True)
# фиксируем входы обучающих данных
n_train = np.shape(x_raw)[0]
# фиксируем количество обучающих данных
num_selection = 10000
# выбираем случайный индекс
random_selection_indices = np.random.choice(n_train, num_selection)
# по индексу выбираем соответствующих обучающий пример
x_raw = x_raw[random_selection_indices]
y_raw = y_raw[random_selection_indices]
```

Предобрабатываем данные

```
# фиксируем коэффициент отравления
percent_poison = .33
# отравляем обучающие данные
x_train, y_train = preprocess(x_raw, y_raw)
x_train = np.expand_dims(x_train, axis=3)
# отравляем данные для теста
x_test, y_test = preprocess(x_raw_test, y_raw_test)
x_test = np.expand_dims(x_test, axis=3)
# фиксируем обучающие классы
n_train = np.shape(y_train)[0]
```

```
# перемешиваем обучающие классы
shuffled_indices = np.arange(n_train)
np.random.shuffle(shuffled_indices)
x_train = x_train[shuffled_indices]
y_train = y_train[shuffled_indices]
```

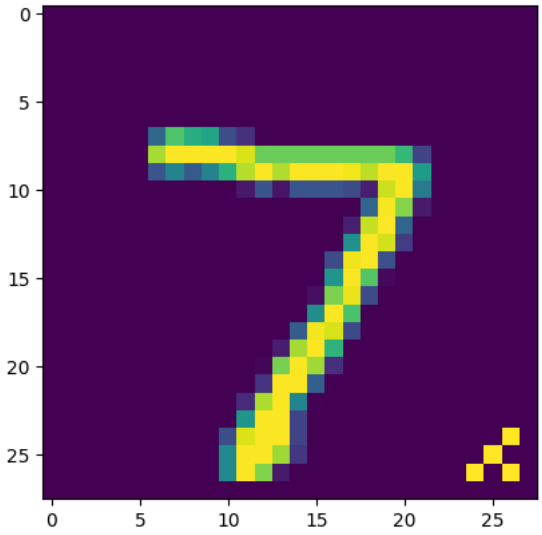
Составляем функцию для создания последовательной модели

```
def create_model():
  # объявляем последовательную модель
  model = Sequential()
  # добавляем сверточный слой 1
  model.add(Conv2D(32, (3,3), activation='relu', input_shape=(28, 28, 1)))
  # добавляем сверточный слой 2
  model.add(Conv2D(64, (3,3), activation='relu'))
  # добавляем слой пуллинга
  model.add(MaxPooling2D((2,2)))
  # добавляем дропаут 1
  model.add(Dropout(0.25))
  # добавляем слой выравнивания
  model.add(Flatten())
  # добавляем полносвязный слой 1
  model.add(Dense(128, activation = 'relu'))
  # добавляем дропаут 2
  model.add(Dropout(0.25))
  # добавляем полносвязный слой 2
  model.add(Dense(10, activation = 'softmax'))
  # компилируем модель
  model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
  # возвращаем скомпилированную модель
  return model
```

Создаем атаку

```
# объявляем класс, реализующий бэкдор-атаку
backdoor = PoisoningAttackBackdoor(add_pattern_bd)
# выберем пример атаки
example_target = np.array([0, 0, 0, 0, 0, 0, 0, 0, 0, 1])
# атакуем пример
pdata, plabels = backdoor.poison(x_test, y=example_target)
# визуализируем атакованный пример
plt.imshow(pdata[0].squeeze())
```

```
<matplotlib.image.AxesImage at 0x79d2c7c8b910>
```



Определяем целевой класс атаки

```
targets = to_categorical([9], 10)[0]
```

Создаем модель

```
# обычная модель
model = KerasClassifier(create_model())
# модель, наученная состязательным подходом по протоколу Мэдри
proxy = AdversarialTrainerMadryPGD(KerasClassifier(create_model()), nb_epochs=10, eps=0.15, eps_step=0.001)
```

```
# обучаем последнюю
proxy.fit(x_train, y_train)
```

```
Precompute adv samples:                                    1/1 [00:00<00:00,
100%                                                       32.59it/s]
Adversarial training epochs:                               10/10 [15:27<00:00
```

Выполняем атаку

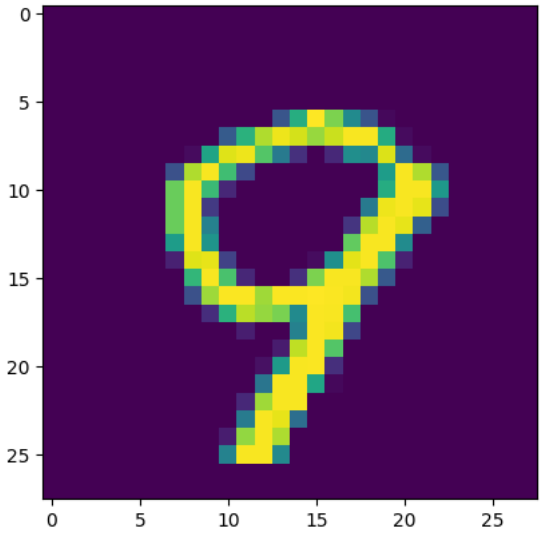
```
# конфигурируем атаку под модель Мэдри
attack = PoisoningAttackCleanLabelBackdoor(backdoor=backdoor,\
                                           proxy_classifier=proxy.get_classifier(),\
                                           target=targets, pp_poison=percent_poison,\
                                           norm=2, eps=5, eps_step=0.1, max_iter=200)
# запускаем отравление
pdata, plabels = attack.poison(x_train, y_train)
```

```
PGD - Random Initializations:                              1/1 [00:06<00:00,
100%                                                       6.33s/it]
PGD - Random Initializations:                              1/1 [00:07<00:00,
100%                                                       7.03s/it]
PGD - Random Initializations:                              1/1 [00:06<00:00,
100%                                                       6.29s/it]
PGD - Random Initializations:                              1/1 [00:07<00:00,
100%                                                       7.07s/it]
PGD - Random Initializations:                              1/1 [00:06<00:00,
100%                                                       6.33s/it]
PGD - Random Initializations:                              1/1 [00:10<00:00,
100%                                                       10.10s/it]
```

Создаем отравленные примеры данных

```
# берем отравленные входы
poisoned = pdata[np.all(plabels == targets, axis=1)]
# и отравленные выходы
poisoned_labels = plabels[np.all(plabels == targets, axis=1)]
# посмотрим количество отравленных входов
print(len(poisoned))
idx = 0
# визуализируем одно из отравленных изображение
plt.imshow(poisoned[idx].squeeze())
print(f"Label: {np.argmax(poisoned_labels[idx])}")
```

```
981
Label: 9
```



Обучаем модель на отравленных данных

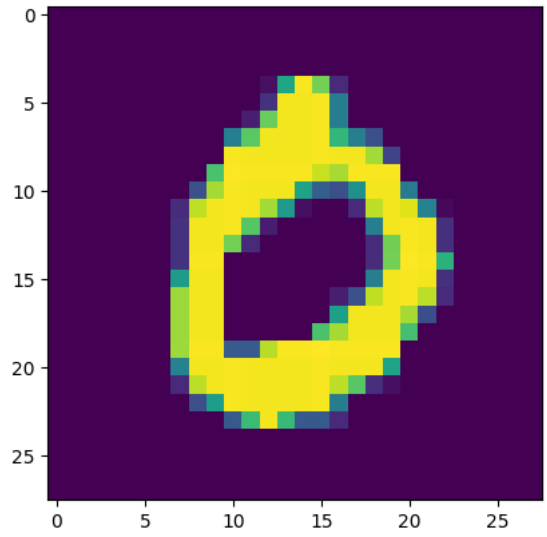
```
model.fit(pdata, plabels, nb_epochs=10)
```

```
Train on 10000 samples
Epoch 1/10
10000/10000 [==============================] - 15s 1ms/sample - loss: 0.6067 - accuracy: 0.8138
Epoch 2/10
10000/10000 [==============================] - 14s 1ms/sample - loss: 0.1751 - accuracy: 0.9479
Epoch 3/10
10000/10000 [==============================] - 14s 1ms/sample - loss: 0.1055 - accuracy: 0.9681
Epoch 4/10
10000/10000 [==============================] - 15s 2ms/sample - loss: 0.0678 - accuracy: 0.9785
Epoch 5/10
10000/10000 [==============================] - 14s 1ms/sample - loss: 0.0495 - accuracy: 0.9854
Epoch 6/10
10000/10000 [==============================] - 15s 2ms/sample - loss: 0.0369 - accuracy: 0.9888
Epoch 7/10
10000/10000 [==============================] - 15s 2ms/sample - loss: 0.0366 - accuracy: 0.9886
Epoch 8/10
10000/10000 [==============================] - 14s 1ms/sample - loss: 0.0214 - accuracy: 0.9925
Epoch 9/10
10000/10000 [==============================] - 15s 1ms/sample - loss: 0.0194 - accuracy: 0.9939
Epoch 10/10
10000/10000 [==============================] - 14s 1ms/sample - loss: 0.0171 - accuracy: 0.9945
```

Осуществляем тест на чистой модели

```
# предсказываем на тестовых входах "здоровых" примеров
clean_preds = np.argmax(model.predict(x_test), axis=1)
# вычисляем среднюю точность предсказания на полном наборе тестов
clean_correct = np.sum(clean_preds == np.argmax(y_test, axis=1))
clean_total = y_test.shape[0]
clean_acc = clean_correct / clean_total
print("\nClean test set accuracy: %.2f%%" % (clean_acc * 100))
# отобразим картинку, ее класс, и предсказание для легитимного примера, чтобы
# показать как отравленная модель классифицирует легитимный пример
c = 0 # класс
i = 0 # изображение
c_idx = np.where(np.argmax(y_test, 1) == c)[0][i] # индекс картинки в массиве
# легитимных примеров
plt.imshow(x_test[c_idx].squeeze())
plt.show()
clean_label = c
print("Prediction: " + str(clean_preds[c_idx]))
```

```
Clean test set accuracy: 98.13%
```



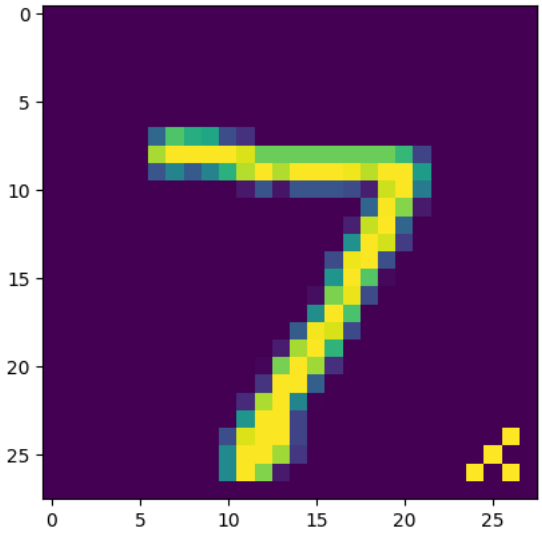

```
Prediction: 0
```

Получаем результаты атаки на модель

```
not_target = np.logical_not(np.all(y_test == targets, axis=1))
px_test, py_test = backdoor.poison(x_test[not_target], y_test[not_target])
# собираем предсказания для отравленных тестов
poison_preds = np.argmax(model.predict(px_test), axis=1)
# вычисляем среднюю точность предсказаний на полном наборе тестов
poison_correct = np.sum(poison_preds == np.argmax(y_test[not_target], axis=1))
poison_total = poison_preds.shape[0]
poison_acc = poison_correct / poison_total
print("\nPoison test set accuracy: %.2f%%" % (poison_acc * 100))
```

```
c = 0 # индекс картинки
# отобразим картинку
plt.imshow(px_test[c].squeeze())
plt.show()
# выведем предсказанный моделью класс
clean_label = c
print("Prediction: " + str(poison_preds[c]))
```

```
Poison test set accuracy: 0.26%
```




```
Prediction: 9
```