|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА - Российский технологический университет»**  **РТУ МИРЭА** |

РТУ МИРЭА

Институт комплексной безопасности и специального приборостроения

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ**

по дисциплине

«Анализ защищенности систем искусственного интеллекта»

**«Лабораторная работа 4»**

Выполнил студент группы ББМО-01-22

*Лысак Я.Д.*

Проверил

*Спирин Андрей Андреевич*

«27» декабря 2023 г.

Москва, 2023 г.

## Цель работы

Выполнить задания по обучению и подготовке атак на модель машинного обучения, а также подготовить и обучить модель для защиты от данной атаки.

## Задание

1. Подготовить и обучить модель данных для распознавания символов MNIST;
2. подготовить и выполнить атаки FGSM на полученную обученную модель, построить графики и получить выводы;
3. выполнить настройку и обучение модели для защиты сети, получить итоги и графики точности;
4. cделать выводы о полученных данных и эффективности метода защиты.

## Ход выполнения работы

В качестве набора данных для данной работы выступают изображения, полученные из тестового набора данных MNIST, содержащие изображения рукописных цифр.

На этапе подготовки проводится подключения графического ускорения CUDA (фрагмент 1). Также подключаются все необходимые для работы библиотеки.

Фрагмент кода – 1

!nvidia-smi

print(torch.cuda.is\_available())

use\_cuda=True

device = torch.device("cuda" if (use\_cuda and torch.cuda.is\_available()) else "cpu")

Набор данных MNIST загружается в директорию для последующей обработки и разделяется на выборки для обучения и тестирования. Также производится преобразование для нормализации входных данных (фрагмент кода 2).

Фрагмент кода – 2

transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.0,), (1.0,))])

dataset = datasets.MNIST(root = './data', train=True, transform = transform, download=True)

train\_set, val\_set = torch.utils.data.random\_split(dataset, [50000, 10000])

test\_set = datasets.MNIST(root = './data', train=False, transform = transform, download=True)

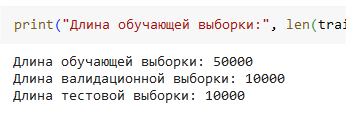
train\_loader = torch.utils.data.DataLoader(train\_set,batch\_size=1,shuffle=True)

val\_loader = torch.utils.data.DataLoader(val\_set,batch\_size=1,shuffle=True)

test\_loader = torch.utils.data.DataLoader(test\_set,batch\_size=1,shuffle=True)

В результате выполненных действие были получены наборы данных, необходимые для выполнения задачи в объеме, обозначенном на рисунке 1.

Рисунок – 1



Затем производится подготовка НС, а именно создается функция-конструктор, определяющая архитектуру будущей нейронной сети. Код данной функции представлен во фрагменте кода 3.

Фрагмент кода – 3

class Net(nn.Module):

def \_\_init\_\_(self):

super(Net, self).\_\_init\_\_()

self.conv1 = nn.Conv2d(1, 32, 3, 1)

self.conv2 = nn.Conv2d(32, 64, 3, 1)

self.dropout1 = nn.Dropout2d(0.25)

self.dropout2 = nn.Dropout2d(0.5)

self.fc1 = nn.Linear(9216, 128)

self.fc2 = nn.Linear(128, 10)

def forward(self, x):

x = self.conv1(x)

x = F.relu(x)

x = self.conv2(x)

x = F.relu(x)

x = F.max\_pool2d(x, 2)

x = self.dropout1(x)

x = torch.flatten(x, 1)

x = self.fc1(x)

x = F.relu(x)

x = self.dropout2(x)

x = self.fc2(x)

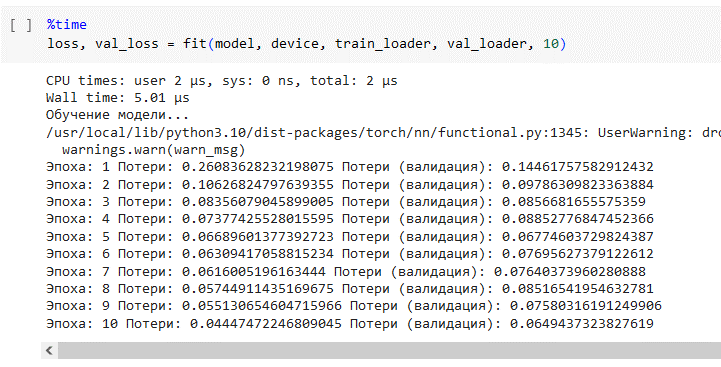
output = F.log\_softmax(x, dim=1)

return output

Также настраиваются функция потерь и функция для обучения. Для оптимизации был выбран алгоритм Adam со значениями Бета 0.9, 0.999, а также ReduceLROnPlateau из библиотеки PyTorch.

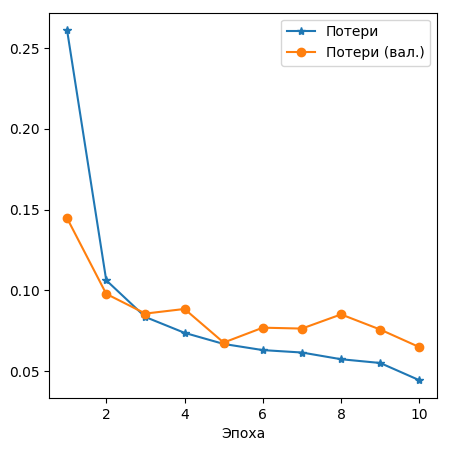
После подготовки необходимых функций производится первичное обучение нейронной сети. Вывод результатов, полученных во время обучения продемонстрирован на рисунке 2. Как можно видеть, обучение с использованием CUDA ядер продвигается существенно быстрее, чем без них. Также можно отметить серьёзное снижение потерей уже после 2 эпохи обучения.

Рисунок 2



Также после проведения обучения был построен график потерь функций обучения и валидации (рисунок 3).

Рисунок 3



После первичного обучения производится подготовка функций для проведения атак FGSM, I-FGSM, MI-FGSM. Код данных функций продемонстрирован во фрагменте кода 4. В данной фрагмент можно видеть код всех 3 функций, с использованием переменной Эпсилон для установки значения искажения атакуемого изображения, демонстрация работы предоставлена далее.

Также далее была задана функция для тестирования результатов атаки.

Фрагмент кода – 4

def fgsm\_attack(input,epsilon,data\_grad):

pert\_out = input + epsilon\*data\_grad.sign()

pert\_out = torch.clamp(pert\_out, 0, 1)

return pert\_out

def ifgsm\_attack(input,epsilon,data\_grad):

iter = 10

alpha = epsilon/iter

pert\_out = input

for i in range(iter-1):

pert\_out = pert\_out + alpha\*data\_grad.sign()

pert\_out = torch.clamp(pert\_out, 0, 1)

if torch.norm((pert\_out-input),p=float('inf')) > epsilon:

break

return pert\_out

def mifgsm\_attack(input,epsilon,data\_grad):

iter=10

decay\_factor=1.0

pert\_out = input

alpha = epsilon/iter

g=0

for i in range(iter-1):

g = decay\_factor\*g + data\_grad/torch.norm(data\_grad,p=1)

pert\_out = pert\_out + alpha\*torch.sign(g)

pert\_out = torch.clamp(pert\_out, 0, 1)

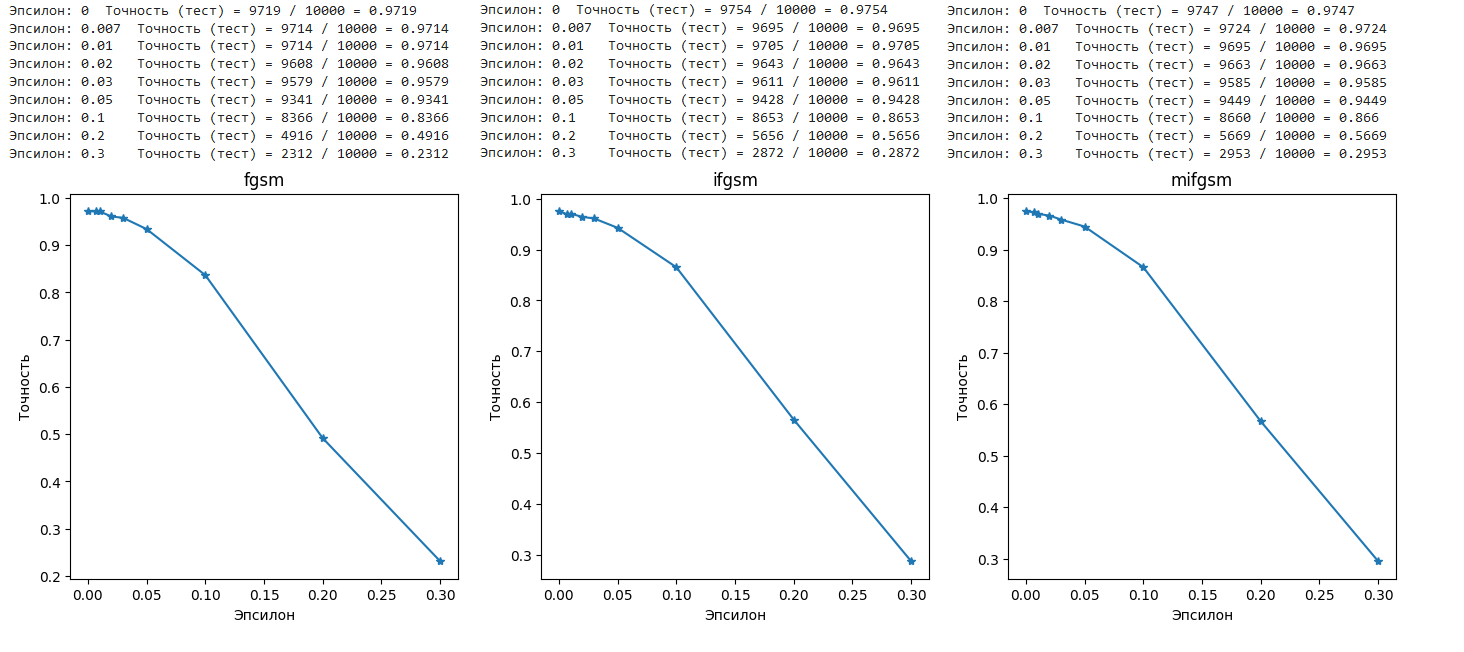
if torch.norm((pert\_out-input),p=float('inf')) > epsilon:

break

return pert\_out

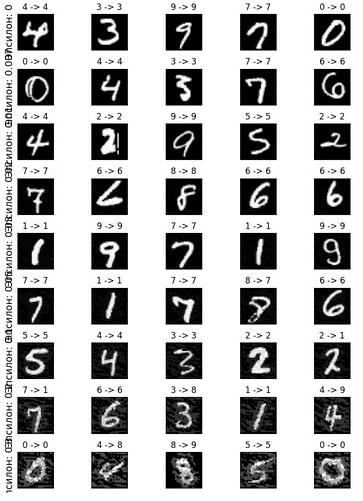
После подготовки функций для атаки, тестирования и проверки точности, были также подготовлены соответствующие графики. Результат работы проводимых атак FGSM продемонстрирован на рисунке 4.1. I-FGSM на рисунке 4.2 и MI-FGSM на рисунке 4.3

Рисунок – 4 (4.1, 4.2, 4.3)



Также более подробно были выведены конечные получение изображения, которые были подвержены атаке. Например, на рисунке 5 изображены конечные цифры, атакованные FGSM.

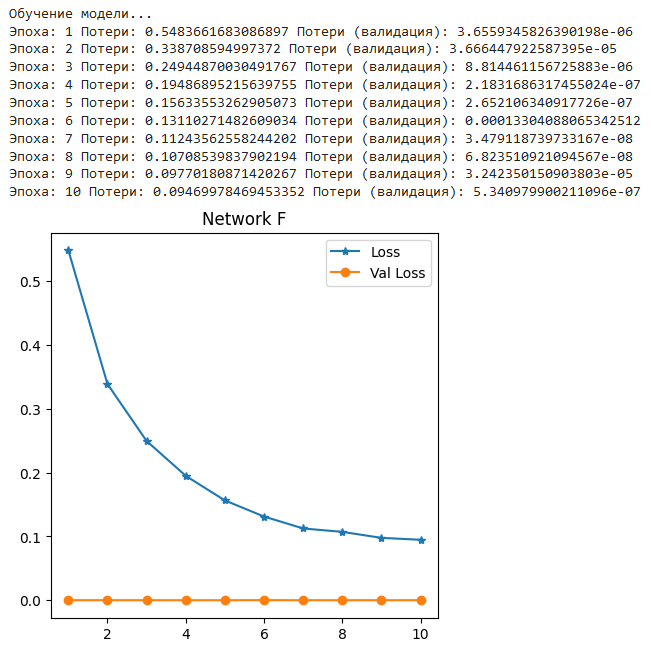
Рисунок – 5



Как можно видеть изображение качественно ухудшается с увеличение значения Эпсилон, таким образом нейронная сеть демонстрирует нарушение в определении целевого значения. Например, при значении Эпсилон выше 0.3 среди всех экспериментов удачными предсказания НС были только в 6 из 15 случаем. Увеличивая Эпсилон более значения 0.3 может привести к слишком сильному искажению, что повысит результативность атаки, но сделает изображения не читаемым для человека.

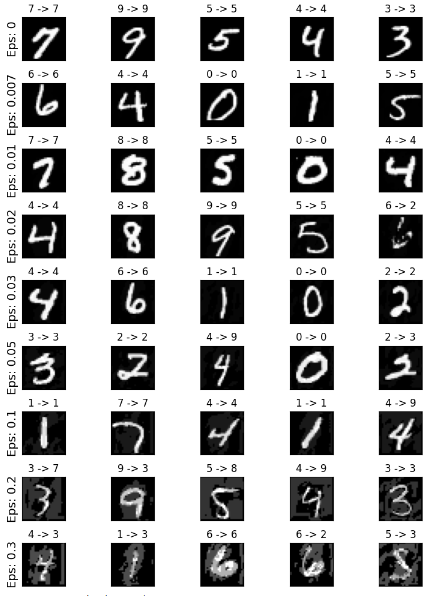
Далее для защиты подготавливается конструктор модели и ее слоев, а также функции для обучения, валидации и тестирования. После этого производятся обучение и проверка результатов, пример для сети F (рисунок 6).

Рисунок – 6



Также получен пример изображений, подверженных атаки и использованных в функции защиты. Например, защита от FGSM и полученные изображения из набора MNIST продемонстрированы на рисунке 6.

Рисунок 6



Как можно видеть, точность работы модели, относительно атакованной без защиты выросла.

**Вывод**

В результате выполнения работы получен некоторый опыт работы с инструментами атаки на нейронные сети. Были также проведены эксперименты по атаке на модели машинного обучения с примерами на изображениях из набора данных MNIST. Также были проведены работы по защите сетей от таких атак.

В рамках работы были рассмотрены несколько различных методов и алгоритмов атаки FGSM. Были проведены эксперименты с различными параметрами и модификациями методов.

Как можно заметить, после применения алгоритма по защите модели, результат работы был улучшен. Несмотря на это, в данном случае полностью восстановить работоспособность и защитить модель полностью не удалось. Вероятно, это зависит от показателей Эпсилон и сложности модели, а также от объема данных обучения и тестирования выборок. Результат проделанной работы демонстрирует улучшение в работе модели, подвергшейся атаке

После завершения экспериментов и проведения анализа полученных результатов были сделаны выводы касательного полноты и точности объяснения работы методов генерации карт.

Полный код блокнота по проведённой работе приведен в приложении А.