DOI 10.36622/1682-7813.2024.27.4.004
УДК 349:004.056

НЕЙРОСЕТЕВОЙ СЕРВИС РЕГЛАМЕНТАЦИИ МЕР ПРОТИВОДЕЙСТВИЯ КИБЕРАТАКАМ (ЧАСТЬ IV)

Г.А. Остапенко, А.П. Васильченко, А.А. Остапенко, С.В. Безбородых, А.Г. Остапенко, Н.П. Жуков

Рассматривается нейросетевая реализация модуля обнаружения и идентификации кибератак. В связи с этим предлагаются алгоритмическое и программное обеспечения, включая: обоснование выбора языка программирования для реализации нейросетевого модуля; подготовку базы данных для машинного обучения, программную реализацию нейронной сети; демонстрацию работы модуля обнаружения и идентификации кибератак на серии примеров CAPEC-шаблонов; разработку архитектуры нейронной сети; механизмы взаимодействия с модулем регламентации мер киберпротиводействия. Осуществляются сравнительные оценки точности работы нейросети во время её обучения. Определяется количество эпох обучения. По актуальной метрике для различных классов атак анализируются возможности идентификации вторжения. Отмечается способность модуля к дообучению на новых данных, содержащих информацию об активности трафика, в целях расширения ареала обнаруживаемых атак и формирования мер реагирования на обнаруженные инциденты.

Ключевые слова: нейросеть, модуль, метрики, характеристики, база данных, вектор атаки, точность.

Введение

Практической реализации нейронной сети предшествует разработка её архитектуры.

Необходимо выбрать подходящие топологию и функционал. Данные характеристики определяют, каким образом нейронная сеть будет обрабатывать поступающие данные, как и с какой точностью будет осуществляться обучение сети.

Анализ сетевого трафика нейронной сетью представляет собой задачу классификации, в которой необходимо установить принадлежность обрабатываемых данных к какому-либо классу (в данном случае в качестве классов выступают идентификаторы CAPEC). Изначально нейронные сети предназначались для численного предсказания, а не для много классовой классификации, однако теперь с их помощью можно строить сложные математические модели, позволяющие анализировать данные с высокой точностью [1].

1 Разработка архитектуры нейронной сети

Количество входных нейронов должно быть равно количеству признаков, которые будет обрабатывать нейросетевой модуль. Выбранная база данных для машинного обучения NF-UQ-NIDS-v2 содержит 45 признаков, но анализироваться будут не все. Например, поля, содержащие IP-адреса устройств, участвовавшие в сеансе связи, не будут подвергаться анализу, так как они ни коим образом не коррелируют с вероятностью реализации сетевой атаки. Поля с метками атак также придется отбросить, они будут использованы для формирования результата, которого должна добиться нейронная сеть. В конечном итоге будет анализироваться 41 характеристика, соответственно количество входных нейронов будет равно 41. Так же задается количество и выходных нейронов – по одному на каждый класс сетевого трафика, то есть 21 нейрон.

Значение, поступающее на вход нейрона, определяется по формуле [2]

$$S = \sum_{i=1}^{n} x_i w_i , \qquad (1)$$

где n – число входов нейрона;

x_i – значение i-го входа нейрона;

w_i – вес i-го ребра, что надо, чтобы достичь минимума данной функции ошибки.

В многослойном перцептроне для обучения применяется метод градиентного спуска. Его особенность заключается в нахождении градиента производной функции потерь относительно одного веса, и его корректировки в сторону минимального значения. Находится он по формуле [3]:

$$\delta = e * f'(v_{out}), \qquad (2)$$

где e – величина ошибки, найденная с помощью функции потерь;

$f'(v_{out})$ – производная функции активации выходного нейрона.

Скорость обучения нейронной сети в данном методе зависит от величины градиента, так как корректировка весов осуществляется по формуле:

$$w_i = w_j - \lambda * \delta, \qquad (3)$$

где w_i – новое значение веса;

w_j – предыдущее значение веса;

δ – градиент, найденный по формуле (2);

λ – величина шага спуска.

При малом значении градиента, веса будут меняться незначительно. Если при разработке нейронной сети в приоритете расположена скорость обучения, имеет смысл подобрать функцию активации с большим градиентом.

Функции активации определяют то, каким образом будут обрабатываться данные в нейронной сети. В задачах классификации наиболее эффективными себя показали функции: сигмоида, гиперболический тангенс и ReLu [4]. Однако, стоит использовать функцию ReLu, так как она обладает рядом значительных достоинств.

Она рассчитывается по формуле:

$$f(x) = \max(0, x). \qquad (4)$$

Её главным достоинством является ровная производная в диапазоне [0, ∞], благодаря чему обучения нейронной сети производится эффективно. Она похожа на обычную линейную функцию (рис.1), но по своей природе линейной не является, а комбинации слоев с данной функцией порождают также нелинейную функцию, что позволяет строить модели, позволяющей производить сложные классификации данных.

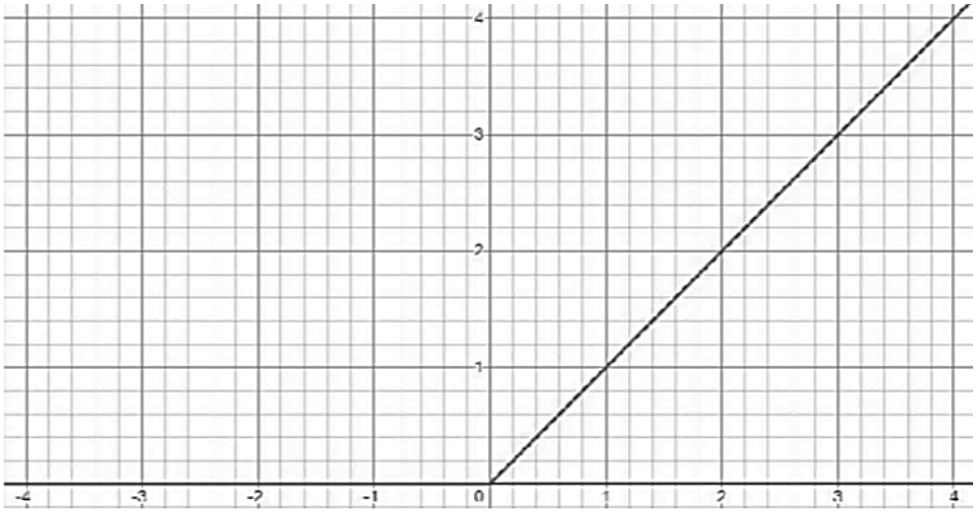

Рис. 1. График функции ReLu

Большим недостатком данной функции активации являются «умирающие нейроны». Если какие-либо веса нейронной сети окажутся отрицательными или нулевыми, то нейроны, с которыми они связаны, примут нулевое значение. В методе обучения «градиентный спуск» корректировка весов осуществляется, опираясь на производную функции, которая может оказаться равной 0 в диапазоне [-∞, 0]. Из-за этого корректировка весов не произойдет, и нейроны могут навсегда остаться неактивными, что может привести к снижению точности нейронной сети. Однако, при обработке данных о сетевом трафике, это не является проблемой, так как ни одна из 41 характеристик не будет принимать отрицательные значения. Поэтому в разрабатываемой нейронной сети используется функция активации ReLu.

При окончании расчетов на выходном слое нейронной сети образуются различные значения. Каждое из них определяет степень принадлежности данных к каждому классу. Такие значения гораздо удобнее представить в виде вероятностей с помощью функции активации *SoftMax*, которая рассчитывается по формуле [4]:

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}, \quad (5)$$

где x – входной вектор,

x_i – i-й элемент входного вектора,

K – количество классов или длина входного вектора.

Функция активации (5) рассчитывается как вероятность, поэтому преобразует значения на выходных нейронах в диапазон [0,1]. Сумма функций (5) по всем выходам нейронов составляет единицу. Таким образом могут быть получены вероятностные значения принадлежности обрабатываемых данных к определенному классу.

В итоге на нейронах скрытого и входного слоя будет использована функция активация ReLu для обеспечения эффективного машинного обучения, а на последнем слое – Softmax. От количества слоев и нейронов зависит сложность математической модели, которую образует нейронная сеть. Большое количество слоев и нейронов позволит создать сложную модель, способную классифицировать обучающие данные с высокой точностью. Однако, высок риск переобучения, когда на реальных данных нейронная сеть не может показать ту же самую эффективность, что и на обучающих данных. С малым количеством нейронов и слоев модель будет не так точна, но на реальных данных может показать себя лучше, чем предыдущая модель. Заранее предсказать оптимальное количество слоев и нейронов невозможно, поэтому при разработке часто начинают с малого количества с постепенным его увеличением, пока не будет достигнута требуемая точность. Для построения более обобщающей модели принято на каждом последующем слое задавать меньшее количество нейронов, чем на предыдущем, но этого можно избежать, если нейронная сеть небольшая.

Таким образом было принято следующее (табл. 1).

Таблица 1

Архитектура нейронной сети

Слой	**Функция активации**	**Количество нейронов**	**Количество ребер**
Входной	ReLu	41	1722
Скрытый	ReLu	41	1722
Скрытый	ReLu	41	1722
Скрытый	ReLu	41	1722

Продолжение табл. 1

Слой	Функция активации	Количество нейронов	Количество ребер
Скрытый	ReLu	41	1722
Скрытый	ReLu	41	1722
Выходной	Softmax	21	882

2 Взаимодействие с модулем регламентации мер противодействия

В результате работы нейронной сети на нейронах выходного слоя будут получены различные значения, определяющие вероятность принадлежности полученного сетевого трафика к каждому классу. Нейроны образуют вектор, каждая позиция которого отвечает за определенный класс сетевого трафика.

Логично предположить, что к тому классу, к которому больше всего вероятность принадлежности обрабатываемых данных, сетевой трафик и относится. Однако, это может привести к большому количеству ложно положительных и ложно отрицательных результатов. Так как имеется всего 21 класс, теоретически возможна ситуация, когда вероятность принадлежности сетевого трафика к классам атак будет составлять по 5%. Поэтому целесообразно для каждого класса установить некоторый порог срабатывания. Если заданный порог окажется превышен, администратору будет выведено сообщения об обнаруженной атаке.

Цикл работы нейросетевого модуля представлен на рис. 2.

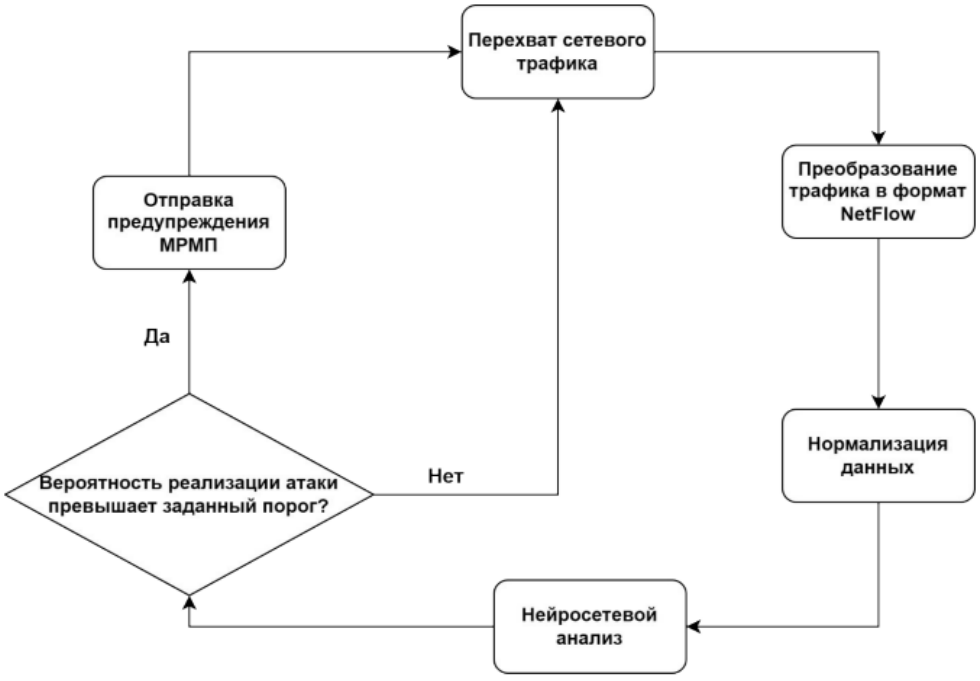

Рис. 2. Цикл работы нейросетевого модуля

Полагается, что данный модуль будет взаимодействовать с модулем регламентации мер противодействия, к которому отправляется сообщение об обнаруженной атаке. Он в свою очередь будет генерировать инструкции по предотвращению атаки и минимизации её ущерба. Сообщение представляет собой UDP-пакет, в котором содержится идентификатор CAPEC обнаруженной атаки и идентификатор уязвимости, которую она эксплуатирует. Структура сообщения представлена на рис. 3.

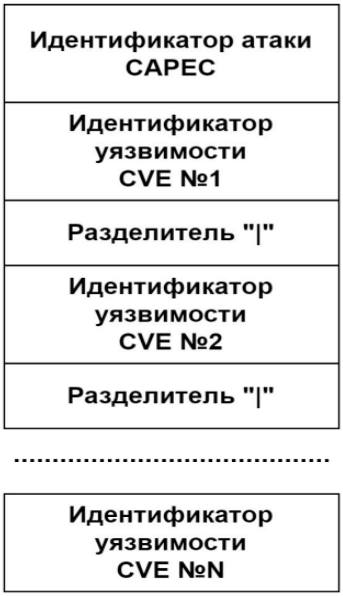

Рис. 3. Структура UDP-пакета сообщения модулю регламентации мер противодействия

3 Подготовка базы данных к машинному обучению

Чтобы машинное обучение протекало эффективно требуется провести подготовку обучающей базу данных. Данные, представленные в NF-UQ-NIDS-v2, имеют разный физический смысл, от того и разные абсолютные значения. Работа с разными по величине значениями может оказаться некорректной [5].

Поэтому данные необходимо нормализовать, то есть свести значения к одному диапазону. Чаще всего в его качестве выступает диапазон [0; 1], так как в нем наиболее эффективно работает большинство алгоритмов машинного обучения и функций активации.

Характеристики нормализуются разными способами. Количественные – применением различных функций, качественные – представлением в виде вектора, как указано на рис. 4.

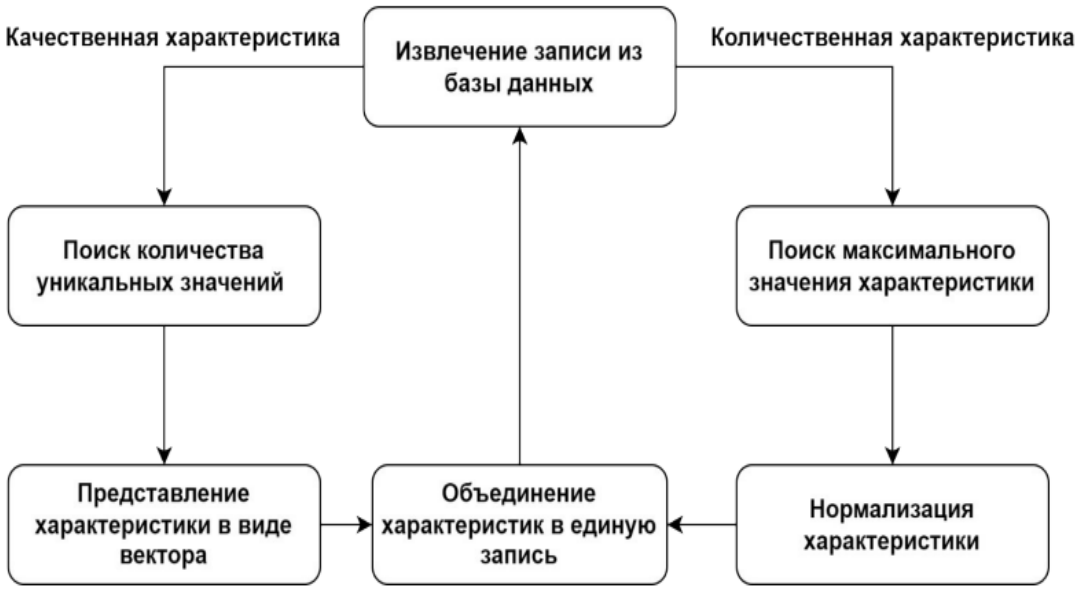

Рис. 4. Схема подготовки базы данных к машинному обучению

Для нормализации количественных характеристик применяют формулу MinMax Scaler [6]:

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}, \quad (6)$$

где x – предыдущее значение;

x' – новое значение;

x_{min} – минимальное значение;

x_{max} – максимальное значение;

Так как минимальное значение всех характеристик обучающей базы данных NF-UQ-NIDS-v2 принимает значение 0, то функция сократится до следующей:

$$x' = \frac{x}{x_{max}}, \quad (7)$$

Соответственно, для нормализации данных достаточно найти их максимальные значения. Качественные же характеристики потребуется представить в виде векторов, длины которых равны количеству уникальных значений, которые характеристики могут принимать. Длины векторов представлены в табл. 2.

Таблица 2

Длины векторов качественных характеристик

Характеристика	**Длина вектора**
L4_SRC_PORT	65536
L4_DST_PORT	65536
PROTOCOL	255
L7_PROTO	255
ICMP_IPV4_TYPE	255
DNS_QUERY_ID	65536
DNS_QUERY_TYPE	65536
FTP_COMMAND_RET_CODE	553

Таким образом в конечном итоге будут получены записи длиной в 263 495 значений, каждое из которых находится в диапазоне [0, 1]. Подготовленная база данных, состоящая из данных записей, способна обеспечить максимальную эффективность машинного обучения нейросетевого модуля.

Для разработки нейросетевого модуля применялся следующий инструментарий, позволяющий реализовать все описанные выше алгоритмы:

– язык программирования Python [7];

– среда разработки PyCharm [8];

– библиотеки Keras, TensorFlow, Numpy, Pandas [9-12].

4 Обоснование выбора языка программирования для реализации нейросетевого модуля

Для разработки нейронной сети применялся высокоуровневый язык программирования Python. Он обладает большим количеством средств для анализа данных и является одним из самых простых и читаемых языков программирования, благодаря чему код удобно редактировать и внедрять в другие проекты.

Программирование осуществлялось на платформе разработки PyCharm, предоставляющая удобный инструмент рефакторинга кода, отладку для

отслеживания ошибок, поддержку систем контроля версий, а также средства для установки необходимых зависимостей для корректной работы программ. К достоинствам данной платформы также можно отнести кроссплатформенность, позволяющую работать с одними и теми же проектами на разных операционных системах.

Для Python написаны все необходимые библиотеки для работы с базами данных и машинного обучения. К ним относятся: Pandas, Numpy, TensorFlow, Keras.

Большинство баз данных представлено в табличном формате csv, которые возможно прочитать с помощью библиотеки Pandas. Она предоставляет методы, позволяющие извлечь данные из файла и представить их в формате DataFrame, содержащий имена столбцов и их элементы.

Для реализации нейронной сети применяется библиотека Keras, которая является надстройкой над Tensorflow, разработанной компанией Google для построения графов. Keras позволяет упростить и ускорить разработку нейронных сетей, представляя их в виде взаимосвязанных слоев нейронов. Данная библиотека оказалась настолько популярной, что на сегодняшний день она поставляется вместе с TensorFlow.

Нейронные сети, разработанные с применением библиотеки Keras, принимают на вход данные в формате многомерных векторов и матриц. Поэтому, чтобы представить данные в требуемом формате, применялась библиотека Numpy.

5 Подготовка базы данных NF-UQ-NIDS-v2 для машинного обучения

Обучающая база данных представлена в виде файла с названием NF-UQ-NIDS- v2.csv и весом 13 729 330 230 байт. Для её подготовки требуется нормализовать данные. Поиск максимальных значений характеристик осуществлялся применением алгоритма, реализованного с помощью Python на рис. 5.

```
def Find_Max_Features(path):
        for file in database_files:
                print("file – " + file)
                data = pd.read_csv(path + file)

                for feature in database_features:
                        if database_max_values[feature] < data[feature].describe()['max']:
                                database_max_values[feature] < data[feature].describe()['max']
```

Рис. 5. Реализация метода поиска максимальных значений

В цикле for прочитываются все файлы, указанные в списке database_files, и осуществляется поиск характеристик, указанных в списке database_features. Максимальные значения хранит словарь database_max_values, в котором каждой характеристике присвоено максимальное значение. При инициализации все характеристики равны нулю, но после окончания цикла они примут другие значения. Их можно вывести на экран или сохранить в файл для последующего их применения. Были получены максимальные значения характеристик, указанные в табл. 3.

Таблица 3

Максимальные значения характеристик базы данных NF-UQ-NIDS-v2

Характеристика	**Максимальное значение**
L4_SRC_PORT	65535
L4_DST_PORT	65535
PROTOCOL	255
L7_PROTO	248
IN_BYTES	301926201
IN_PKTS	469281
OUT_BYTES	275639781
OUT_PKTS	410903
TCP_FLAGS	223
CLIENT_TCP_FLAGS	223
SERVER_TCP_FLAGS	223
FLOW_DURATION_MILLISECONDS	4294967
DURATION_IN	119574
DURATION_OUT	119558
MIN_TTL	255
MAX_TTL	255
LONGEST_FLOW_PKT	65535
SHORTEST_FLOW_PKT	3990
MIN_IP_PKT_LEN	1260
MAX_IP_PKT_LEN	65535
SRC_TO_DST_SECOND_BYTES	$8.912111521067721*10^{304}$
DST_TO_SRC_SECOND_BYTES	$1.87216383510445*10^{257}$
RETRANSMITTED_IN_BYTES	15120489
RETRANSMITTED_IN_PKTS	13796
RETRANSMITTED_OUT_BYTES	7329139
RETRANSMITTED_OUT_PKTS	5536
SRC_TO_DST_AVG_THROUGHPUT	4277408000
DST_TO_SRC_AVG_THROUGHPUT	4294480000
NUM_PKTS_UP_TO_128_BYTES	475657
NUM_PKTS_128_TO_256_BYTES	73465
NUM_PKTS_256_TO_512_BYTES	47667
NUM_PKTS_512_TO_1024_BYTES	38812
NUM_PKTS_1024_TO_1514_BYTES	200984
TCP_WIN_MAX_IN	65535
TCP_WIN_MAX_OUT	65535
ICMP_TYPE	65317
ICMP_IPV4_TYPE	255
DNS_QUERY_ID	65536
DNS_QUERY_TYPE	55937
DNS_TTL_ANSWER	4294915672
FTP_COMMAND_RET_CODE	553

Нормализация данных реализована отдельным Python модулем. Чтение сохраненных максимальных значений характеристик осуществляется применением метода Read_Max_Features, показанного на рис. 6.

Все прочитанные значения записываются в словарь database_max_values. Далее осуществляется чтение самого файла базы данных. Для более эффективного обучения требуется сбалансировать данные, чтобы нейронная сеть при обучении корректировала свою математическую модель одинаково в стороны всех классов. Иначе при классификации сетевого трафика она будет склоняться к классу с наибольшим количеством записей.

```
def Read_Max_Features(path):
    with open(path) as file:
        data_from_Max_Features = file.readlines()

        index = 0
        for feature in database_max_values:
            database_max_values[feature] = float(data_from_Max_Features[index])
            index = index + 1
```

Рис. 6. Реализация метода чтения максимальных значений характеристик

Балансировка проводится извлечением всех классов атак в отдельные списки, как показано на рис. 7:

```
data_DDoS = data[data['Attack'] == "DDoS"]
data_DoS = data[data['Attack'] == "DoS"]
data_scanning = data[data['Attack'] == "scanning"]
data_Reconnaissance = data[data['Attack'] == "Reconnaissance"]
data_xss = data[data['Attack'] == "xss"]
data_password = data[data['Attack'] == "password"]
data_injection = data[data['Attack'] == "injection"]
data_Bot = data[data['Attack'] == "Bot"]
data_Brute_Force = data[data['Attack'] == "Brute_Force"]
data_Infilteration = data[data['Attack'] == "Infilteration"]
data_Exploits = data[data['Attack'] == "Exploits"]
data_Fuzzers = data[data['Attack'] == "Fuzzers"]
data_Backdoor = data[data['Attack'] == "Backdoor"]
data_Generic = data[data['Attack'] == "Generic"]
data_mitm = data[data['Attack'] == "mitm"]
data_ransomware = data[data['Attack'] == "ransomware"]
data_Thert = data[data['Attack'] == "Thert"]
data_Analysis = data[data['Attack'] == "Thert"]
data_Shellcode = data[data['Attack'] == "Shellcode"]
data_Worms = data[data['Attack'] == "Worms"]
data_Benign = data[data['Attack'] == "Benign"]
```

Рис. 7. Извлечение классов атак из базы данных NF-UQ-NIDS-v2

Балансировка проводится дублированием записей недостающего класса и отсечением лишних у классов с превышающим количеством записей. Было выбрано среднее арифметическое количество, а именно 3500000. Осуществляется данная процедура применением метода resample у библиотеки sklearn на рис. 8.

```
data_DDoS_resample = resample( *arrays: data_DDoS, n_samples=3500000, random_state=123)
data_DoS_resample = resample( *arrays: data_DoS. n_samples=3500000, random_state=123)
data_scanning_resample = resample( *arrays: data_scanning, n_samples=3500000, random_state=123)
data_Reconnaissance_resample = resample( *arrays: data_Reconnaissance, n_samples=3500000, random_state=123)
data_xss_resample = resample( *arrays: data_xss, n_samples=3500000, random_state=123)
data_password_resample = resample( *arrays: data_password, n_samples=3500000, random_state=123)
data_injection_resample = resample( *arrays: data_injection, n_samples=3500000, random_state=123)
data_Bot_resample = resample( *arrays: data_Bot, n_samples=3500000, random_state=123)
data_Brute_Force_resample = resample( *arrays: data_Brute_Force, n_samples=3500000, random_state=123)
data_Infilteration_resample = resample( *arrays: data_Infilteration, n_samples=3500000, random_state=123)
data_Exploits_resample = resample( *arrays: data_Exploits, data[data['Attack'] == "Exploits"]
data_Fuzzers_resample = resample( *arrays: data_Fuzzers, n_samples=3500000, random_state=123)
data_Backdoor_resample = resample( *arrays: data_Backdoor, n_samples=3500000, random_state=123)
data_Generic_resample = resample( *arrays: data_Generic, n_samples=3500000, random_state=123)
data_mitm_resample = resample( *arrays: data_mitm, n_samples=3500000, random_state=123)
data_ransomware_resample = resample( *arrays: data_ransomware, n_samples=3500000, random_state=123)
data_Thert_resample = resample( *arrays: data_Thert, n_samples=3500000, random_state=123)
data_Analysis_resample = resample( *arrays: data_Analysis, n_samples=3500000, random_state=123)
data_Shellcode_resample = resample( *arrays: data_Shellcode, n_samples=3500000, random_state=123)
data_Worms_resample = resample( *arrays: data_Worms, n_samples=3500000, random_state=123)
data_Benign_resample = resample( *arrays: data_Benign, n_samples=3500000, random_state=123)
```

Рис. 8. Балансировка классов атак

Параметр n_samples указывает, какое в итоге должно получиться количество записей. Random_state позволяет запомнить алгоритм распределения записей, чтобы в дальнейшем можно было повторить результат.

В итоге было получено 73 500 000 записей по 3 500 000 на каждый класс. Нормализация записей осуществляется образом, указанным на рис. 9.

```
def IN_BYTES_Parse(data):
    if database_max_values['IN_BYTES'] == 0:
        lint = [0]
        return line
    line = [data / database_max_values['IN_BYTES']

    return line
```

Рис. 9. Нормализация характеристик

Из словаря database_max_values извлекается требуемое значение, на которое делятся поступающие данные.

Также необходимо подготовить выходные данные для нейронной сети. Они будут сформированы из столбца ‘Attack’ и представлены в виде вектора, в котором каждая позиция соответствует определенному классу, как продемонстрировано в табл. 4.

Таблица 4

Вектора классов атак в выходных данных

Атака	**Вектор**
DDoS	[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
DoS	[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Scanning	[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Reconnaisance	[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
XSS	[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Password	[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Injection	[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Bot	[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Brute Force	[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Infiltration	[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Exploits	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Fuzzers	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Backdoor	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
Generic	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
MITM	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
Ransomware	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
Analysis	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
Theft	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0]
Shellcode	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]
Worms	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
Benign	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]

Каждой записи входных данных должна соответствовать запись выходных данных.

Таким образом был получен файл с данными на вход и файл с данными на выход, которые будут использованы во время машинного обучения.

6 Программная реализация нейронной сети

Разработка нейронной сети проводилась с применением методов библиотеки Keras (рис. 10).

```
model = keras.Sequential()
model.add(keras.layers.Dense( units: 41, activation='relv'))
model.add(keras.layers.Dense( units: 41, activation='relv'))
model.add(keras.layers.Dense( units: 41, activation='relv'))
model.add(keras.layers.Dense( units: 41, activation='relv'))
model.add(keras.layers.Dense( units: 41, activation='relv'))
model.add(keras.layers.Dense( units: 41, activation='relv'))
model.add(keras.layers.Dense( units: 21, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy', 'f1_score'])
print("Study...")
history = model.fit(x=input_data, y=output_data, epochs=15, validation_data=(val_input, val_output))
```

Рис. 10. Программная реализация архитектуры нейронной сети

В методе keras.Sequential() создается многослойная нейронная сеть типа многослойного персептрона. Все нейроны последующего слоя связаны со всеми нейронами предыдущего слоя. При этом создается пустая модель сети. Метод model.add() добавляет слои в эту сеть. В каждом слое по 41 нейрону во входном и скрытых слоях, в выходном слое – 21 нейрон. Во всех слоях, кроме последнего, применяется функция активации ReLu, а в последнем – функция SoftMax.

Метод model.compile() настраивает процесс машинного обучения. В нем указывается функция потерь «Перекрестная энтропия», оптимизатор метода обучения «Adam» и метрики, которые позволят оценить качество нейронной сети, а именно точность (accuracy) и полнота (f1_score).

Точность является самым простым способом оценки качества моделей, которая отражает долю правильных спрогнозированных классов среди всех образцов [13].

$$Accuracy = \frac{1}{N}\sum_{i=1}^{N} I[y_i = \hat{y_i}] = \frac{TP + TN}{TP + TN + FP + FN}, \quad (8)$$

где N – общее количество классов;

y_i – предсказанный класс;

$\hat{y_i}$ – истинный класс;

TP – количество истинно-положительных классификаций;

TN – количество истинно-отрицательных классификаций;

FP – количество ложно-положительных классификаций;

FN – количество ложно-отрицательных классификаций.

Метрики *Precision* и *Recall* решают проблемы *Accuracy*, так как они не зависят от соотношения классов и не позволяют модели определять все данные к одному классу. Поэтому их часто применяют в условиях дисбаланса классов:

$$Precision = \frac{TP}{TP + FP}, \quad (9)$$

где TP – количество истинно-положительных классификаций;

FP – количество ложно-положительных классификаций.

$$Recall = \frac{TP}{TP + FT}. \quad (10)$$

где TP – количество истинно-положительных классификаций;

FN – количество ложно-отрицательных классификаций.

Recall показывает способность нейронной сети обнаруживать класс, а Precision – способность отличать данный класс от других. Обе метрики объединяет F1-мера, представляющая собой среднее гармоническое значение между ними [14]:

$$F1 = 2 * \frac{Precision * Recall}{Precision * Recall.}. \quad (11)$$

Формулу (11) используют для общей оценки качества нейронных сетей при решении задач классификации, так как на практике часто добиться максимальных значений одновременно и Recall, и Precision невозможно, а F1-мера позволяет найти баланс между ними. Поэтому для определения точности обучения нейронной сети были выбраны метрики accuracy и f1-score.

Метод model.fit() осуществляет само машинное обучения. Ему необходимо указать данные на вход и данные на выход в формате Numpy, а также количество эпох, которые определяют количество циклов обучения на одних и тех же данных. Валидационная выборка опциональна, её можно создать, разделив данные на вход на две части, но в данном случае используется заранее подготовленные валидационные данные. Метод model.fit() возвращает объект, в котором содержатся данные метрик, указанных в предыдущем методе . Его можно сохранить, а затем использовать для анализа качества обучения нейронной сети [15].

Во время работы метода model.fit() в терминале будет выводиться информация о состоянии нейронной сети во время обучения (рис. 11).

```
Study...
Epoch 1/15
6563/6563 ━━━━━━━━━━━━━━━━━━━━ 5s 546us/step - accuracy: 0.6183 - f1_score: 0.6037 - loss: 1.1651
Epoch 2/15
6563/6563 ━━━━━━━━━━━━━━━━━━━━ 4s 537us/step - accuracy: 0.7711 - f1_score: 0.7674 - loss: 0.6381
Epoch 3/15
5181/6563 ━━━━━━━━━━━━━━━━━━━━ 0s 517us/step - accuracy: 0.7925 - f1_score: 0.7894 - loss: 0.5618
```

Рис. 11. Процесс машинного обучения в терминале PyCharm

После окончания обучения было получено два файла. Один из них является непосредственно самой нейронной сетью. Её можно импортировать в другие проекты, используя библиотеку Keras. Другой файл содержит информацию о точности нейронной сети во время машинного обучения, из которого были получены графики на рис. 12 - 18:

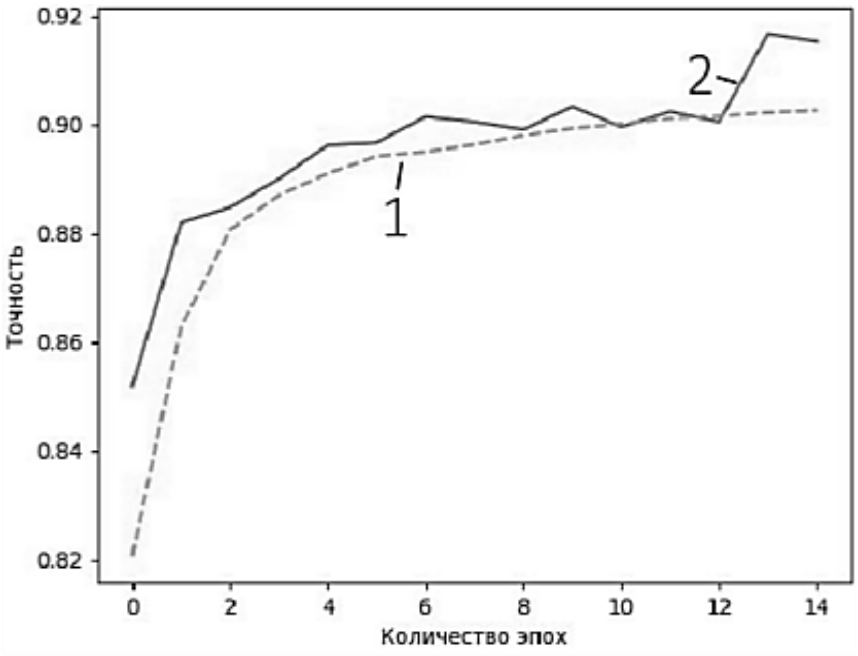

Рис. 12. График точности нейронной сети во время обучения,
где 1 – график обучающей выборки, 2 – график валидационной выборки

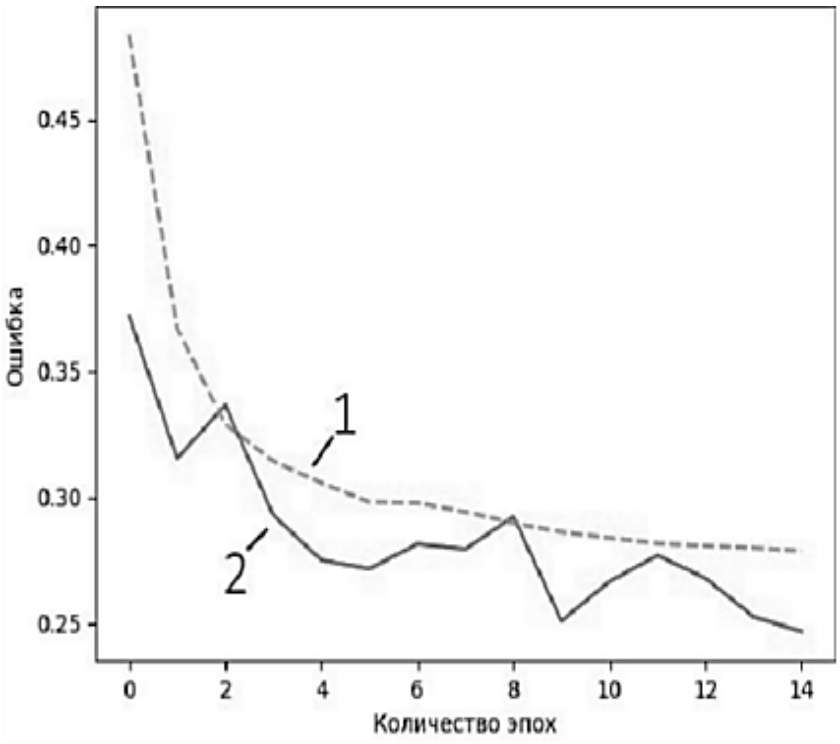

Рис. 13. График ошибки нейронной сети во время обучения,
где 1 – график обучающей выборки, 2 – график валидационной выборки

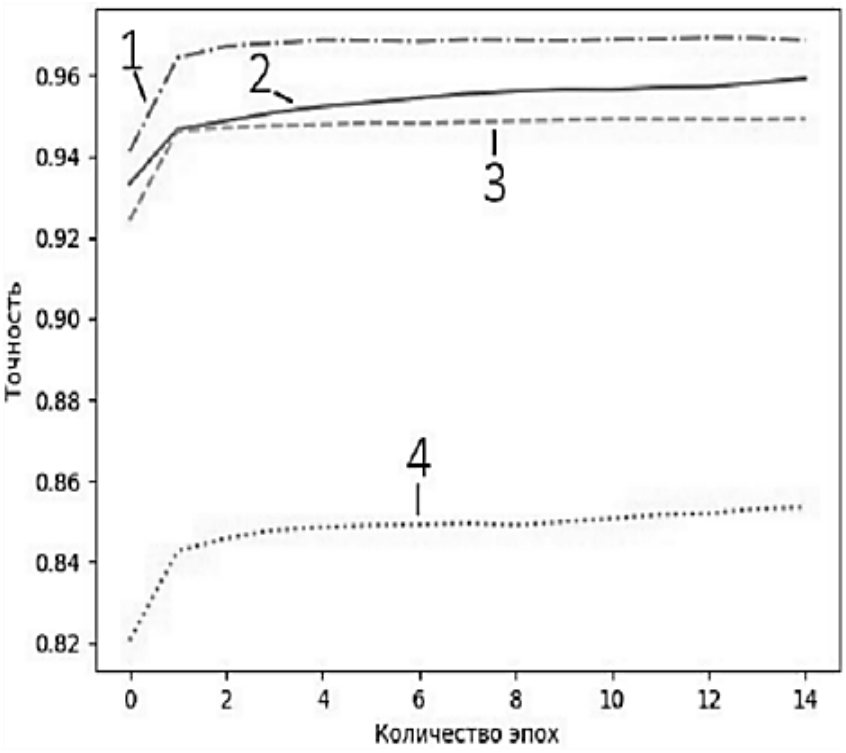

Рис. 14. График F1-меры для нейронной сети во время обучения,
где 1 – график для класса атак Reconnaissance, 2 – график для класса атак DDoS,
3 – график для класса атак DoS, 4 – график для класса атак Scanning

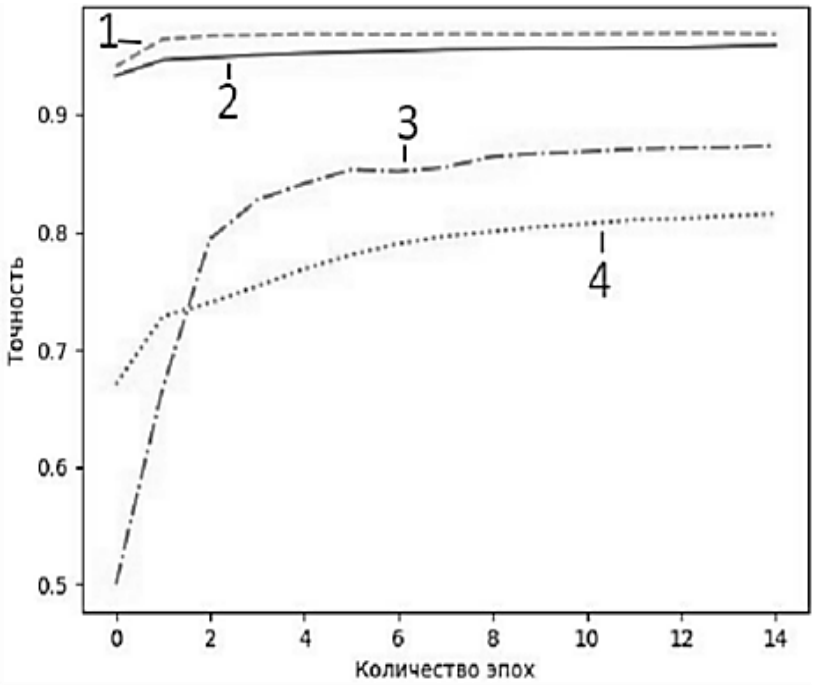

Рис. 15. График F1-меры для нейронной сети во время обучения,
где 1 – график для класса атак Password, 2 – график для класса атак XSS,
3 – график для класса атак Injection, 4 – график для класса атак Bot

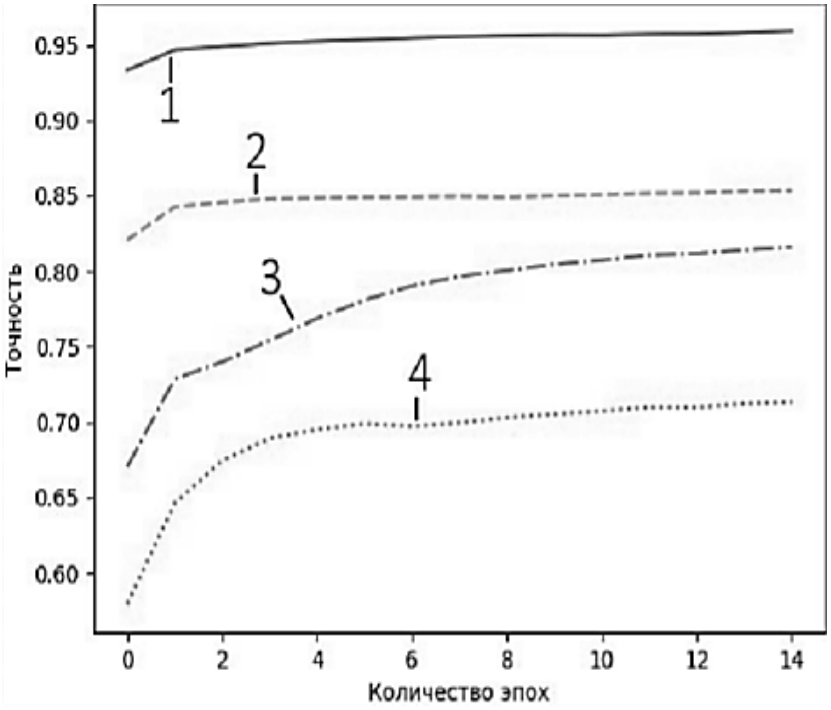

Рис. 16. График F1-меры для нейронной сети во время обучения,
где 1 – график для класса атак Brute Force, 2 – график для класса атак Infinteration,
3 – график для класса атак Exploits, 4 – график для класса атак Fuzzers

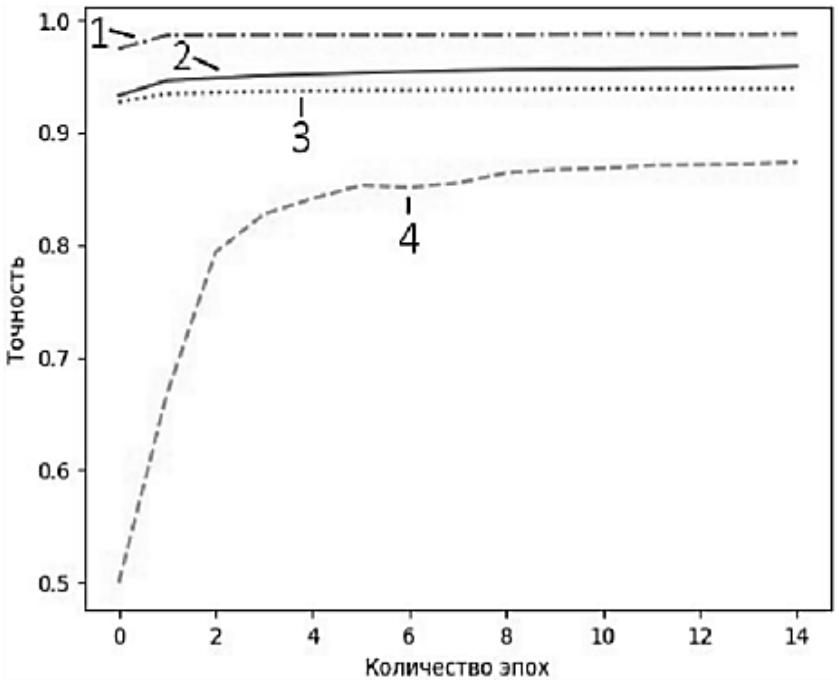

Рис. 17. График F1-меры для нейронной сети во время обучения,
где 1 – график для класса атак MITM, 2 – график для класса атак Backdoor,
3 – график для класса атак Ransomware, 4 – график для класса атак Generic

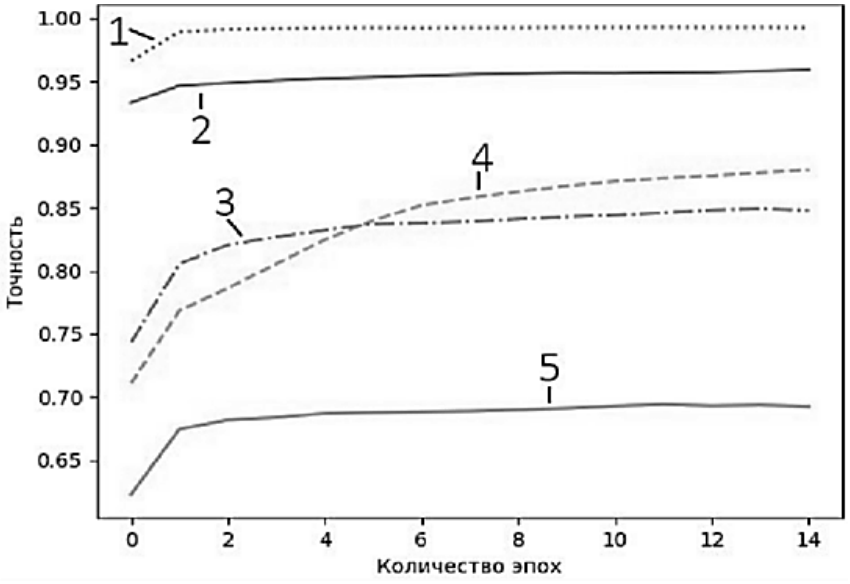

Рис. 18. График F1-меры для нейронной сети во время обучения,
где 1 – график для класса атак Worms, 2 – график для класса атак Analysis,
3 – график для класса атак Shellcode, 4 – график для класса атак Theft,
5 – график для класса Benign

В большинстве графиков, представленных на рис. 12 – 18, наблюдается резкое увеличение точности классификации до второй эпохи, после которой возрастание замедляется, а в некоторых графиках и вовсе останавливается. Это говорит о том, что нейронная сеть достигла своего практического пика точности и дальнейшее обучение может лишь незначительно увеличить этот показатель. Так как она преодолела требуемый порог точности, а именно 0.8, для большинства классов, было принято решение остановить машинное обучение на 15 эпохе. Окончательные его результаты представлены в табл. 5.

Таблица 5

Значения F1-меры для каждого класса в конце машинного обучения

Класс	**F1-мера**
DDoS	0.96
DoS	0.95
Scanning	0.97
Reconnaisance	0.855
XSS	0.85
Password	0.875
Injection	0.82
Bot	0.9998
Brute Force	0.988
Infiltration	0.72
Exploits	0.82
Fuzzers	0.92
Backdoor	0.94

Продолжение табл. 5

Класс	**F1-мера**
Generic	0.87
MITM	0.9
Ransomware	0.995
Analysis	0.83
Theft	0.9925
Shellcode	0.985
Worms	0.99
Benign	0.69

Исходя из данных таблицы, лучше всего обнаруживается класс Bot, а хуже всех – Benign. Это может привести к склонению нейронной сети чаще видеть в сетевом трафике атаку, чем нормальную активность.

Заключение

В порядке разрешения выявленного у аналогов противоречия между объективной необходимостью в идентификации кибератак согласно классификаторам угроз информационной безопасности и отсутствием данного функционала у выявленных аналогов, которые используют алгоритмы машинного обучения для обнаружения аномалий в активности сетевого трафика, удалось получить программную реализацию модуля обнаружения и идентификации сетевых атак и уязвимостей в соответствии с данными классификаторами, представляющем собой нейросетевой модуль.

Результаты работы обладают следующими новизной, практической ценностью и теоретической значимостью:

– В отличие от аналогов разработанный нейросетевой модуль способен к обнаружению и идентификации сетевых атак и эксплуатируемых уязвимостей в соответствии с классификаторами CAPEC.

– Разработанный нейросетевой модуль возможно внедрить в автоматизированные информационные системы для повышения их защищенности и точности обнаружения и идентификации кибератак путем автоматизации данного процесса и взаимодействия с модулем регламентации мер противодействия.

– Модуль обнаружения и идентификации кибератак способен к дообучению на новых данных, содержащих информацию об активности трафика, в целях расширения области обнаруживаемых атак. Полученные в результате работы модуля идентификаторы CAPEC возможно применить для формирования мер реагирования на обнаруженные инциденты.

Список литературы

1. Круглов В. В., Борисов В. В. Искусственные нейронные сети. Теория и практика. – М.: Горячая линия – Телеком, 2001 – 382 с.: ил.

2. Richard P. Lippmann, An Introduction to Computing with Neural Nets, IEEE Acoustics, Speech, and Signal Processing Magazine, April 1987.

3. Paul J. Werbos, Backpropagation Through Time: What It Does and How to Do It //Artificial Neural Networks: Concepts and Theory, IEEE Computer Society Press, 1992, pp.309-319.

4. Выбор слоя активации в нейронных сетях: как правильно выбрать для вашей задачи // Habr URL: https://habr.com/ru/articles/727506/ (дата обращения 27.06.2024).

5. Kumar, Kunal; Azad, S. K. (October 2017). "Database normalization design pattern". 2017 4th IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics (UPCON). IEEE.

6. Машинное обучение. Преобразование данных – URL: https://www.dmitrymakarov.ru/data-

analysis/transform-08/#16-minmaxscaler (дата обращения 27.06.2024).

7. Python – URL: https://www.python.org (дата обращения 27.06.2024).

8. PyCharm. IDE для Data Science и веб-разработки на Python // JetBrains URL: https://www.jetbrains.com/ru-ru/pycharm/ (дата обращения 27.06.2024).

9. Keras. Data Learning for humans – URL: https://keras.io (дата обращения 27.06.2024).

10. TensorFlow – URL: https://www.tensorflow.org/?hl=ru (дата обращения 27.06.2024).

11. NumPy. The fundamental package for scientific computing with Python – URL: https://numpy.org (дата обращения 27.06.2024). – Текст: электронный.

12. Pandas – Python Data Analysis Library – URL: https://pandas.pydata.org (дата обращения 27.06.2024).

13. Powers, David M W (2011). "Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation" (дата обращения 27.06.2024).

14.ChiccoD, JurmanG (January2020). "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation" (дата обращения 27.06.2024).

15. Keras. Model training APIs – URL: https://keras.io/api/models/model_training_apis/ (дата обращения 27.06.2024).

Финансовый университет при Правительстве Российской Федерации
Financial University under the Government of the Russian Federation

Воронежский государственный технический университет
Voronezh State Technical University

Поступила в редакцию 17.07.2024

Информация об авторах

Григорий Александрович Остапенко – д-р техн. наук, профессор, Финансовый университет при Правительстве Российской Федерации, e-mail: ost@fa.ru

Алексей Павлович Васильченко – аспирант, Финансовый университет при Правительстве Российской Федерации, e-mail: rainichek@yandex.ru

Александр Алексеевич Остапенко – аспирант, Воронежский государственный технический университет, e-mail: alexostap123@gmail.com

Станислав Витальевич Безбородых – студент, Воронежский государственный технический университет, e-mail: stanislav.bezb@mail.ru

Остапенко Александр Григорьевич – д-р техн. наук, профессор, заведующий кафедрой, Воронежский государственный технический университет, e-mail: alexanderostapenkoias@gmail.com

Жуков Никита Павлович – студент, Воронежский государственный технический университет, e-mail: znp8b00ff.gmail.com

NEURAL NETWORK SERVICE FOR REGULATING MEASURES TO COUNTER CYBERATTACKS (PART IV)

G.A. Ostapenko, A.P. Vasilchenko, A.A. Ostapenko, S.V. Bezborodykh, A.G. Ostapenko, N.P. Zhukov

The neural network implementation of the cyberattack detection and identification module is considered. In this regard, algorithmic and software solutions are proposed, including: justification of the choice of a programming language for the implementation of a neural network module; preparation of a database for machine learning, software implementation of a neural network; demonstration of the operation of the cyberattack detection and identification module based on a series of examples of CAPEC templates; development of the neural network architecture; mechanisms of interaction with the module for regulating cyber countermeasures. Comparative assessments of the accuracy of the neural network during its training are carried out. The number of training epochs is determined. The possibilities of intrusion identification are analyzed according to the current metric for various classes of attacks. The ability of the module to retrain on new data containing information about traffic activity is noted in order to expand the range of detected attacks and form response measures to detected incidents.

Keywords: neural network, module, metrics, characteristics, database, attack vector, accuracy.

Submitted 17.07.2024

Information about authors

Gregory A. Ostapenko – Dr. Sc. (Technical), Professor, Financial University under the Government of the Russian Federation, e-mail: ost@fa.ru

Alexey P. Vasilchenko – graduate student, Financial University under the Government of the Russian Federation, e-mail: rainichek@yandex.ru

Alexandr A. Ostapenko – graduate student, Voronezh State Technical University, e-mail: alexostap123@gmail.com

Stanislav V. Bezborodykh – student, Voronezh State Technical University, e-mail: sfrvvv@yandex.ru

Alexandr G. Ostapenko – Dr. Sc. (Technical), Professor, Head of the Department, Voronezh State Technical University, e-mail: alexanderostapenkoias@gmail.com

Nikita P. Zhukov – student, Voronezh State Technical University, e-mail: znp8b00ff.gmail.com