**Алгоритм решения задания**

**Чтение и первичный анализ данных**

Для работы с данными планировал использовать библиотеку pandas, поэтому для удобной загрузки данных сначала все pcap файлы с помощью программы wireshark преобразовал в csv таблицы. После чего приступил к анализу параметров. Для обучения решил брать IP адреса отправителя и получателя, потому что в датасете большая часть данных отправлялась и принималась с одинаковых адресов. По этой же причине решил использовать порты отправителя и получателя. Параметр len – длинна сообщения тоже учитывается, так как при какой то аномальной активности сообщение может быть очень большим или наоборот крайне маленьким, также в тестовом датасете параметр len имеет 3 самых характерных значения. Также решил взять ещё параметр windows\_size – потому что он отвечает за количество данных которые хост готов принять, при аномальной активности этот параметр может сильно отличаться.

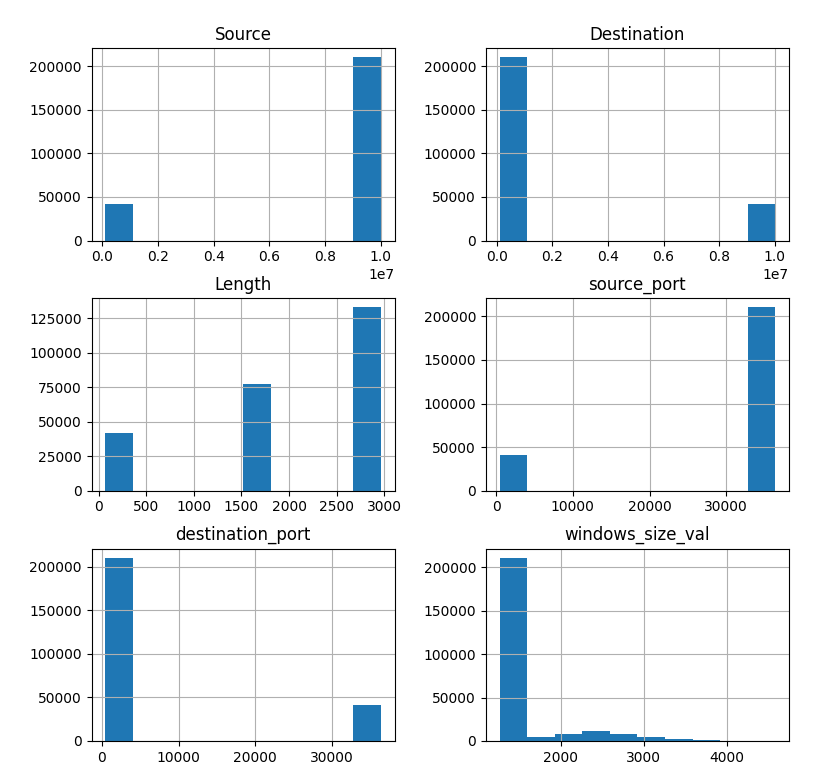
**Создание обучающей выборки**

Методами Python распарсим строки, а также переведем IP адреса в числа путем конкатенации чисел. Инициализируем новый тренировочный датасет выбранными параметрами и посмотрим. При анализе полученных данных можно заметить, что количество уникальных значений в каждом параметре мало относительно размеров датасета, это значит, что параметры действительно могут иметь нехарактерные значения для аномальных событий. Учитывая, что в датасете количество «обычных» значений (таких какие мы не считали бы аномальными), сильно превышает количество аномальных, поэтому риск переобучить модель достаточно мал, но все же построим две модели, первую на необработанных данных, а вторую на обработанных, для каждой модели будем применять одинаковую архитектуру.

Для обработки данных сначала построим ряд частот для каждого из параметров, а затем проанализируем их. Заметим, что параметр Source и Destination имеют по два часто встречающихся значения. Сохраним копию датасета перед тем как его изменять с названием data\_copy. Теперь методами pandas отставим только самые частые значения для этих двух параметров.

Параметр Length имеет 3 характерных значения, также оставим в тренировочных данных только их. После чего заметим, что параметры портов, после отчистки предыдущих параметров, также уже отфильтрованы по самым вероятным значениям.

Последний параметр имеет одно характерное значение, и если попробуем его убрать, то заметим, что также убираем большую часть тех значений, что до этого посчитали не аномальными, поэтому данных параметр фильтровать не будем. Для обучения сети переведем данные в tensor pytorch, и разобьем на тестовые и валидационные (заметим, что валидационные данные создаются на основе обработанного датасета, поэтому в результате будем ожидать, что аномальных значений найдено не будет)



(рис.1 Гистограммы частот после обработки данных)

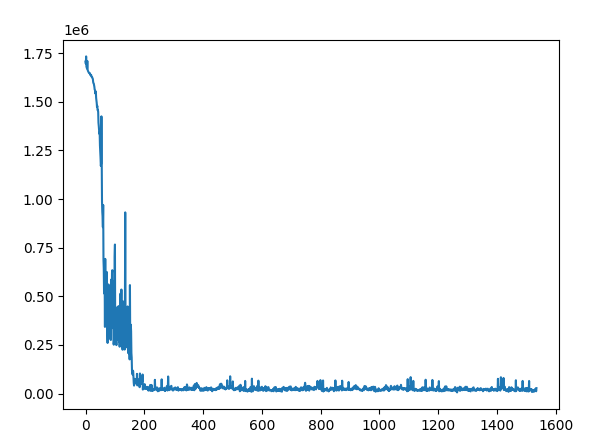
**Создание модели**

Для определения аномальных значений будем использовать модель полносвязной нейронной сети типа encoder decoder. Её суть заключается в том, что на значение на выходе должно получится близкое к тому, что было на входе, но при этом внутри сети входные данные должны закодироваться, чтобы вторая часть из раскодировала, обучив такую сеть на «нормальных» значениях и получая небольшую ошибку на таких данных, на аномальных значениях на выходе должно получиться сильно отличающееся значение, таким образом по большой ошибке можно будет определить, что входное значение было аномальным.

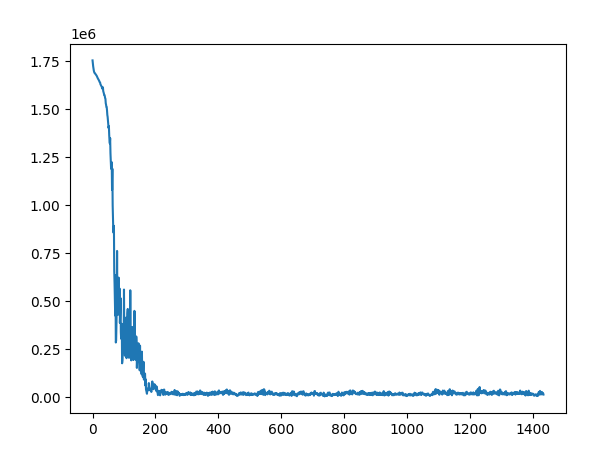
Учитывая, что число параметров мало, будем использовать полносвязную нейронную сеть которая будет расширяться к середине, создавая много новых «фичей», то есть кодирую входные данные, и сужаться к концу до вектора той-же размерности, что и был на входе. В качестве функции ошибки будем использовать метрику L1, потому что нам нужно получить характеристику расстояния между входным и выходным векторами. При этом, так-как например значения IP адресов достаточно велики, не будем использовать L2 метрику, чтобы не получать слишком большие ошибки. Вручную подобрал оптимальные размеры скрытых слоев, в качестве оптимизатора использовал алгоритм Adam.

**Обучение модели**

Для ускорения обучения перенес модель на gpu, подобрал оптимальный размер батча, скорость обучения (learning\_rate) и количество эпох обучения. После чего в цикле пройдем по всем батчам, используя ещё один вложенный цикл. Батчи будем формировать внутри тела цикла, потому что срезы Python позволяют делать это без сильных затрат по времени и памяти. При обучении будем создавать предсказание модели, после чего искать ошибку, записывать её значение в массив ошибок, для дальнейшего отслеживания поведения ошибки, после чего занулим все градиенты и сделаем обратный проход и изменим значения весов всех слоев. По завершении обучения посмотрим на график изменения ошибки



(рис 2. График ошибки при обучении модели без обработки данных)

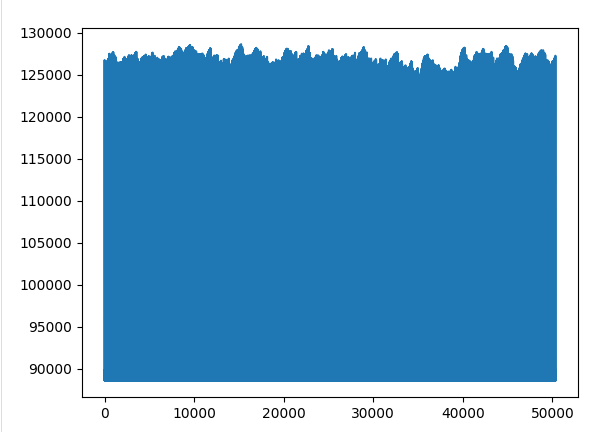


(рис 3. График ошибки при обучении модели с обработкой данных)

Заметим, что графики рис.2 и рис.3 очень похожи. В обоих случаях при после обучения значение ошибки осталось достаточно большим, но при этом крайне стабильным. Можно предположить, что модель обучилась хорошо и при получении аномальных значений мы будем получать ошибку, которая будет значительно отличаться. Также стоит отметить, что обработка данных повлияло на обучение модели. Ошибка модели с обработкой данных получилась более стабильная и имеет более гладкий вид. Скорее всего так получилось из-за того, что модели обучились уже после второй эпохи, а в дальнейшем уже давали результаты, по которым можно было бы определить аномальные события. Но на графике ошибки при обучении модели без обработки данных (рис.2) мы не видим явно выраженных пиков потому, что записывалась средняя ошибка всего батча, и при этом видим некоторые небольшие циклические пики, которые скорее всего говорят о том, что в батчах, которым соответствуют эти пики имеются аномальные значения, которые мы не отфильтровали при предобработке.

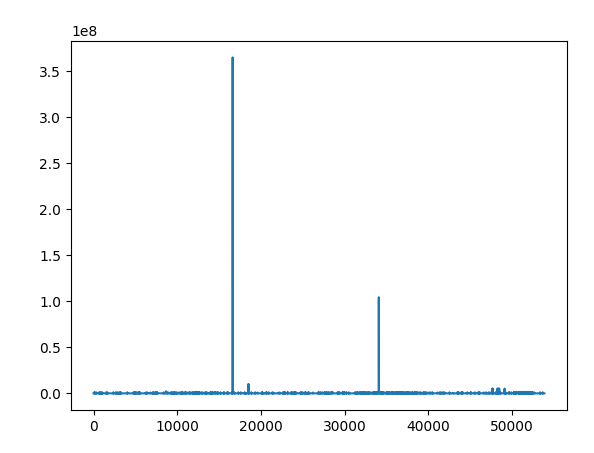
Проверим на валидационных данных, сначала результат работы модели, обученной на обработанных данных. Для этого создадим функцию predict, которая будет находить предсказания модели. На вход она будет получать данные, которые должны пойти на вход модели. После воспользуемся функцией torch.no\_grad(), чтобы проводить операции без сохранения лишней информации, для экономии памяти и ускорения вычислений, так-как совершать обратный проход внутри этой функции мы не будем. После чего с помощью цикла будем разбивать данные на батчи и конкатенировать результат в единый массив. При разбиении данных на батчи в конце могла остаться небольшая часть данных, размер которых меньше одного батча (остаток), при обучении его потеря была бы не значительна, но при нахождении аномалий, мы можем потерять какие-то важные результаты, поэтому после цикла учтем и этот остаток данных. И вернем предсказания модели

Теперь приведем валидационный датасет к массиву numpy (для удобства обработки) и с помощью описанной функции найдем предсказание модели, вручную посчитаем L1 расстояние между предсказанными значениями и первоначальными, для каждого элемента. Выведем график значений. Как и ожидалось – видим стабильную ошибку без каких-то аномалий, потому что валидационный датасет был построен по обработанным данным.



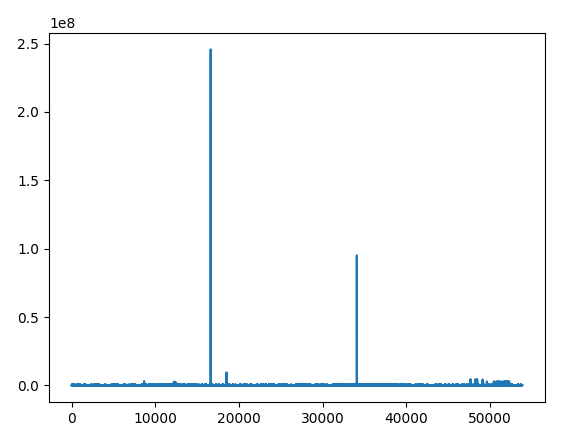
(рис. 4 График ошибок на валидационном, обработанном датасете (модель с предобработкой))

С помощью сохраненного датасета data\_copy, создадим валидационные данные в которых предположительно должны остаться аномалии. После проведем аналогичные действия, что и для предыдущих валидационных данных. По графику ошибки (рис.5) можно определить, что в данных действительно присутствуют аномалии, также заметим, что аномальные примеры имеют значение ошибки больше 0.5 \* 10^8. Приведем массив ошибок к pandas серии и с помощью маски выведем аномальные значения, при этом индексы будут соответствовать индексам аномальных элементов в валидационном датасете. Аномальными оказались элементы: 16615 16616 34073 34074, валидационного датасета.



(рис. 5 График ошибок на валидационном датасете (модель с предобработкой))

Проведем аналогичные действия в модели без предобработки (рис.6)

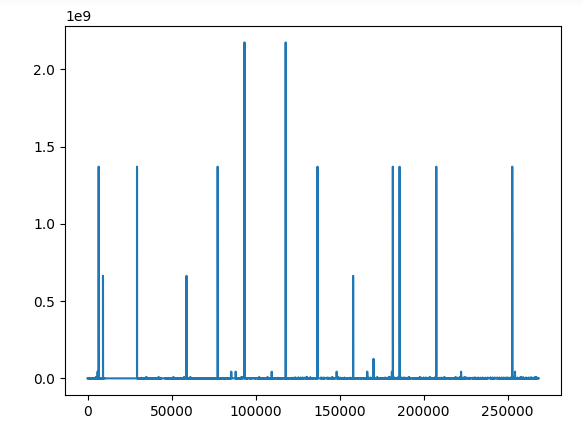
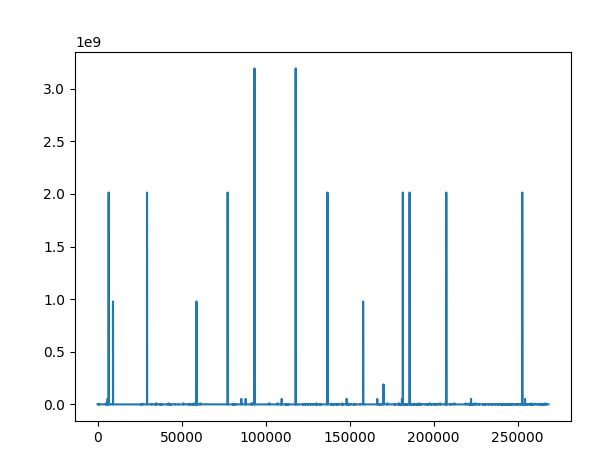


(рис. 6 График ошибок на валидационном датасете (модель без предобработки))

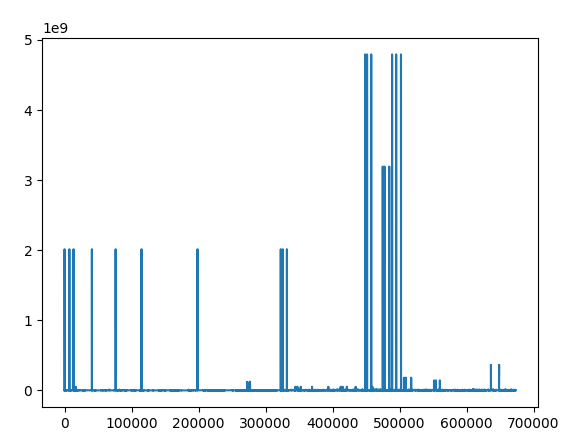
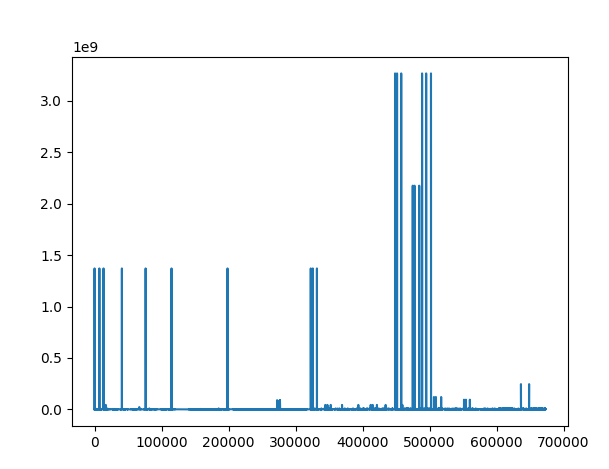
График получился идентичный, но при этом немного отличается само значение ошибки. Аналогично найдем индексы аномальных пакетов валидационных данных и получим следующие значения: 16615 16616 34073 34074. Обе модели показали идентичный результат. Проверим как они поведут себя на тестовых данных.

**Результат на тестовых данных**

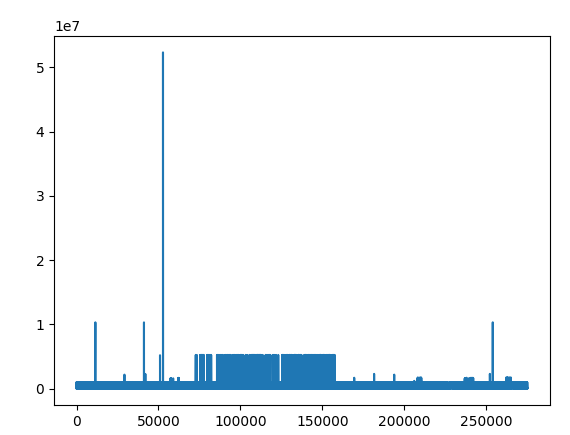
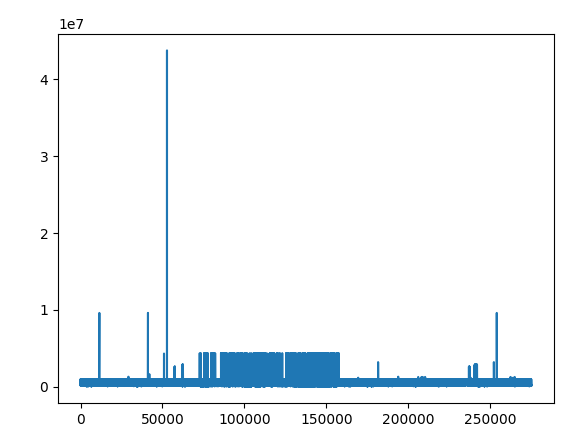
Загрузим данные для всех трех тестовых таблицы аналогично тренировочной таблице. Также распарсим их, получим прогноз модели и найдем ошибки. После чего построим графики. Для удобства обозначим модель, обучающуюся без предобработки данных – модель 1, а модель с предобработкой – модель 2

(рис 7. График ошибок test\_1 модель 1) (рис 8. График ошибок test\_1 модель 2)



(рис 9. График ошибок test\_2 модель 1) (рис 10. График ошибок test\_2 модель 2)



(рис 11. График ошибок test\_3 модель 1) (рис 12. График ошибок test\_3 модель 2)

Учитывая, что графики обеих моделей идентичные и отличаются только масштабом по оси ординат, можно сделать вывод, что из-за того, что в первоначальных данных было мало аномальных значений, модель на них не переобучилась.

Также найдем значения индексов аномальных строк в таблицах, в обеих моделях они также получились идентичные.

**Результаты:**

Аномалии test\_1: 6618, 6619, 9190, 9191, 29443, 29444, 58902, 58903, 77444, 77445, 93418, 93419, 117922, 117923, 136857, 136858, 158076, 158077, 181706, 181707, 185724, 185725, 207565, 207566, 252778, 252779

Аномалии test\_2: 15, 7212, 7213, 13390, 13391, 40809, 40810, 76226, 76227, 114883, 114884, 198466, 198467, 322703, 322704, 325752, 325753, 331905, 331906, 448512, 448513, 451528, 451529, 457740, 457741, 474919, 474920, 477590, 477591, 484637, 484638, 488876, 488877, 494964, 494965, 502165, 502166

Аномалии test\_3 – Обе модели показали странную активность в середине датасета и при этом имеются типичные аномалии, как в предыдущих файлах, так как активность является необычной, то тоже примем её за аномалию, тогда получаем 600 аномальных строк, запишем в файл test\_3.txt.