Міністерство освіти і науки України

Харківський національний університет імені В. Н. Каразіна

Факультет комп'ютерних наук

Спеціальність 125 «Кібербезпека»

Освітня програма «Безпека інформаційних та комунікаційних систем»

«Допущено до захисту»

Зав. кафедрою БІСТ

Сергій РАССОМАХІН

_____

« » 2022р.

Пояснювальна записка

до кваліфікаційної роботи магістра

на тему: «Дослідження застосування штучного інтелекту в системах виявлення/запобігання вторгнень (IDS/IPS)»

оцінка « » Керівник к. т. н., с. н. с. Сватовський І. І.

Голова ЕК Рецензент к. т. н., доцент Бакуменко Н. С.

Доценко С.І. _____ Виконавець : студент групи КБ-61

_____ Дейнега Т.С.

Харків – 2022

РЕФЕРАТ

Пояснювальна записка містить: 79 сторінок, 41 рисунок, 1 таблиця, 31 джерело посилань.

Об'єкт дослідження – системи виявлення/запобігання вторгнень (Intrusion Detection/Prevention System).

Предмет досліджень – технології та методи штучного інтелекту в системах виявлення/запобігання вторгнень (IDS/IPS).

Метою науково-дослідницької роботи є:

- теоретичний аналіз технологій, методів та алгоритмів штучного інтелекту;
- теоретичний аналіз застосування алгоритмів штучного інтелекту у системах IDS/IPS;
- теоретичний аналіз впровадження алгоритмів штучного інтелекту у системах IDS/IPS;
- теоретичний та порівняльний аналізи архітектур алгоритмів штучного інтелекту;
- порівняльний аналіз алгоритмів штучного інтелекту, які використовуються в системах виявлення/запобігання вторгнень;
- побудова дослідної локальної мережі для перехоплення в ній пакетів мережевого трафіку;
- розробка алгоритму класифікації мережевого трафіку для систем виявлення/запобігання вторгнень;
- практична реалізація одного з алгоритмів штучного інтелекту для виконання завдань систем IDS/IPS;
- практичне використання розробленого алгоритму штучного інтелекту у реальній системі IDS/IPS;
- розробка журналу аудиту для фіксації результатів роботи алгоритму штучного інтелекту в системі виявлення/запобігання вторгнень;

- порівняльний аналіз результатів роботи алгоритму штучного інтелекту на тестовій та реальній вибірці даних;
- розробка та впровадження метрик, за якими можна робити висновки щодо ефективності роботи алгоритму штучного інтелекту в системі виявлення/запобігання вторгнень.

Ключові слова: ШТУЧНИЙ ІНТЕЛЕКТ, СИСТЕМА ВИЯВЛЕННЯ/ЗАПОБІГАННЯ ВТОРГНЕНЬ, IDS/IPS, НЕЙРОННА МЕРЕЖА, АЛГОРИТМ НАВЧАННЯ НЕЙРОННОЇ МЕРЕЖІ, КІБЕРБЕЗПЕКА, ВАГОВІ КОЕФІЦІЄНТИ, МЕРЕЖЕВИЙ ТРАФІК, ПЕРЦЕПТРОН, НЕЧІТКА ЛОГІКА, ГЕНЕТИЧНИЙ АЛГОРИТМ, ПРЯМЕ РОЗПОВСЮДЖЕННЯ, ЗВОРОТНЕ РОЗПОВСЮДЖЕННЯ, ОНОВЛЕННЯ ВАГОВИХ КОЕФІЦІЄНТІВ, АЛГОРИТМ ЗВОРОТНОГО РОЗПОВСЮДЖЕННЯ ПОМИЛКИ.

ABSTRACT

The explanatory note contains: 79 pages, 41 figures, 5 tables, 31 sources of references.

The object of research - intrusion detection/prevention systems (IDS / IPS).

Subject of research - technologies and methods of artificial intelligence in intrusion detection/prevention systems (IDS/IPS).

The purpose of research work is:

- theoretical analysis of technologies, methods and algorithms of artificial intelligence;
- theoretical analysis of the application of artificial intelligence algorithms in IDS/IPS systems;
- theoretical analysis of implementation of artificial intelligence algorithms in IDS/IPS systems;
- theoretical and comparative analyzes of architectures of artificial intelligence algorithms;
- comparative analysis of artificial intelligence algorithms used in intrusion detection/prevention systems;
- construction of an experimental local network to intercept network traffic packets in it;
- development of network traffic classification algorithm for intrusion detection/prevention systems;
- practical implementation of one of the algorithms of artificial intelligence for performing tasks of IDS/IPS systems;
- practical use of the developed artificial intelligence algorithm in a real IDS/IPS system;
- development of an audit log to record the results of the artificial intelligence algorithm in the intrusion detection/prevention system;

- comparative analysis of the results of the artificial intelligence algorithm on test and real data samples;
- development and implementation of metrics that can be used to draw conclusions about the effectiveness of the artificial intelligence algorithm in the intrusion detection/prevention system.

Key words: ARTIFICIAL INTELLIGENCE, INVASION DETECTION/PREVENTION SYSTEM, IDS/IPS, NEURAL NETWORK, NEURAL NETWORK TRAINING ALGHORITM, CYBER SECURITY, WEIGHTING COEFFICIENTS, NETWORK TRAFFIC, PERCEPTRON, FUZZY LOGIC, GENETIC ALGHORITM, FORWARD PROPAGATION, BACKWARD PROPOGATION, UPDATE OF WEIGHTING COEFFICIENTS, ERROR BACK-PROPAGATION ALGORITHM.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

IDS/IPS	– Intrusion Detection/Prevention System
ГА	– генетичний алгоритм
НМ	– нейронна мережа
TCP	– Transmission Control Protocol
ШІ	– штучний інтелект
ШНМ	– штучна нейронна мережа
АЗРП	– алгоритм зворотного розповсюдження помилки
UDP	– User Datagram Protocol
IP	– Internet Protocol
ICMP	– Internet Control Message protocol

ВСТУП

Системи Intrusion Detection/Prevention System (IDS/IPS) виявляють зловмисну активність у мережі та запобігають спробам зловмисників отримати доступ до неї сповіщаючи користувача. Для ідентифікації таких атак частіш за все використовуються заздалегідь встановлені правила безпеки для цієї мережі. Системи з таким типом ідентифікації загроз називають експертними системами.

Експертні системи використовують правила, що ідентифікують відомі атаки. Ці правила надаються адміністратором мережі, автоматично створюються системою або використовуються обидва варіанти. Правила використовуються системою для винесення висновків про стан захисту на основі даних, отриманих від системи виявлення атак. Експертні системи вимагають частого оновлення, інакше кажучи, послаблюються можливості системи захисту, а користувачі вводяться в оману щодо захищеності мережі [1].

Подібні системи не здатні виявляти сценарії атак протягом тривалих періодів часу. Атаки, над якими злагоджено працюють багато зловмисників у команді, погано виявляються експертними системами. Ці системи не мають достатньої гнучкості при реалізації структури типу «правило-перевірка». Незначні варіації деталей атаки можуть призвести до її пропуску.

В останні роки розроблено різні підходи до проблеми виявлення атак, що спираються на системи, відмінні від експертних. Зокрема, до таких підходів належить використання методів та алгоритмів штучного інтелекту [1].

Для забезпечення інформаційної безпеки використання методів ШІ є дуже актуальними. Їх основним завданням є виявлення та захист від складних мережевих вторгнень.

Можна виділити наступні найпоширеніші алгоритми ШІ, що використовуються у системах виявлення загроз у мережах [1]:

- генетичний алогритм (Genetic Alghoritm);
- штучні нейро-мережі (ANN – Artificial Neural Network);

- нечітка логіка (Fuzzy Logic).

Однією з найбільш динамічних технологій ШІ вважають технології штучних нейронних мереж (ШНМ) [2]. Вони дозволяють вирішувати завдання обробки складних і нелінійних відносин між даними.

У порівнянні з експертними системами, які можуть дати користувачу тільки певну відповідь щодо відповідності аналізованих характеристик та характеристик збережених у базі даних, система на основі штучного інтелекту проводить аналіз наданих даних та надає можливість оцінити відповідності вхідних даних з характеристиками яким її навчили. Достовірність оцінки залежить від ефективності етапу налаштування та навчання.

Спочатку нейронна мережа повинна бути навчена завдяки правильній ідентифікації попередньо обраних об'єктів предметної області. Далі треба провести аналіз реакції нейронної мережі та налаштувати систему так, щоб досягти результатів, що задовольняють задачі.

Дивлячись на обмежені можливості експертних систем, можна сформулювати висновок про перспективність розробки та впровадження адаптивних систем аналізу мережевого трафіку й управління методами захисту мережі. Системи виявлення/запобігання вторгнень у мережу на основі нейронних мереж мають перспективу вирішення багатьох проблем, що не можуть бути вирішені експертними системами.

Основними перевагами систем виявлення/запобігання вторгнень на основі нейронних мереж є:

- гнучкість та адаптивність алгоритмів, здатність проводити аналіз даних з мереж, навіть якщо вони спотворені або неповні, швидка обробка даних, що дає змогу системі працювати в режимі реального часу;
- спроможність «вивчення» властивостей атак і відокремлення елементів, які відрізняються від тих, що спостерігалися раніше.

Існують два основні варіанти реалізації систем виявлення та запобігання атак [3]. Перший заснований на поєднанні нейронної мережі з вже існуючою експертною системою. При цьому можна зменшити кількість «хибних

спрацьовувань» експертної системи і, отже, збільшити її чутливість. Цей варіант переважає і в економічному аспекті, оскільки не вимагає демонтажу встановленої системи [3].

Другий варіант передбачає встановлення автономної системи виявлення атак на основі нейронної мережі на окремому комп'ютері. Усі повідомлення про підозрілі події надсилаються адміністратору безпеки та/або в систему автоматичного реагування. Порівняно з першим підходом існує лише один рівень аналізу, і тому швидкість роботи системи вища [3].

Все розглянуте визначає актуальність роботи, метою якої є проведення аналізу використання методів та технологій ШІ, аналіз способів використання технологій ШІ та структури системи виявлення інцидентів порушення безпеки інформації та аномалій в мережі, практичне застосування однієї з цих технологій для вирішення задач підвищення ефективності виявлення атак та загроз на основі інформації з мережевого трафіку.

Для досягнення поставленої мети роботи потрібно провести аналіз технологій ШІ, представити певні математичні вирази, що застосовуються в задачах виявлення та класифікації кіберзагроз, ознайомитися з основними методами впровадження методів та алгоритмів ШІ, які застосовуються для знаходження аномальних дій та порушень безпеки, визначити сфери застосування методів ШІ в сучасних технологіях захисту інформації.

1 ДОСЛІДЖЕННЯ ЗАСТОСУВАННЯ АЛГОРИТМІВ ШТУЧНОГО ІНТЕЛЕКТУ У СИСТЕМАХ IDS/IPS.

1.1 Аналіз технологій штучного інтелекту та їх використання у галузі кібербезпеки

Штучний інтелект (іноді званий машинним інтелектом) визначається, як здатність комп'ютерів для опрацювання пов'язаних завдань з розумними істотами.

Система штучного інтелекту дозволяє машинам виконувати завдання, для яких зазвичай потрібен людський інтелект, наприклад, візуальне сприйняття, розпізнавання мови та жестів, прийняття певних рішень, переклад мови, тощо [4]

Області застосування ШІ в кібербезпеці включають:

- виявлення вторгнень: ШІ може виявляти мережеві атаки, зараження шкідливим ПЗ та інші кіберзагрози;
- кібераналітика: ШІ також використовується для аналізу великих даних з метою виявлення закономірностей та аномалій у системі кібербезпеки організації;
- безпечна розробка програмного забезпечення: ШІ може допомогти створити безпечніше програмне забезпечення, надаючи розробникам зворотний зв'язок у режимі реального часу про те, захищений їхній код чи ні.

Фахівці з кібербезпеки вже деякий час використовують рішення на основі ШІ. Однак, у зв'язку зі зростанням числа кібератак вони бачать необхідність більш досконалих інструментів і технологій, щоб не відставати від атак зловмисників [4].

Штучний інтелект може змінити все, що нас оточує і все, з чим ми працюємо, включаючи те, як ми захищаємося від кіберзагроз. Він дозволить нам розуміти ризики точніше, ніж раніше, і приймати рішення швидко і без шкоди для точності його класифікації. Він дозволить нам виявляти нові атаки швидше, ніж будь-коли раніше. Крім того, він визначить, як краще захиститись від них, не дочікуючись втручання людини.

ШІ не може замінити людину, але він може зіграти свою роль у боротьбі з кібератаками. ШІ надає захист за допомогою наступних методів [5]:

1) Автоматизоване виявлення загроз.

Алгоритми машинного навчання виявляють поведінку користувачів чи аномалії у роботі системи, які можуть означати на порушення безпеки.

2) Машинне навчання.

ШІ використовує алгоритми для аналізу великих обсягів даних і робить прогнози на основі знайдених закономірностей. Це використовується для навчання систем ШІ розпізнавати раніше невідомі або несподівані атаки.

3) Предиктивна аналітика.

За допомогою предиктивної аналітики можна прогнозувати майбутні погрози, наприклад, які облікові записи співробітників з найбільшою ймовірністю будуть зламані або які типи атак можуть статися певного дня. Такий аналіз допомагає організаціям визначити, де знаходяться прогалини в системі безпеки, щоб усунути їх до того, як буде завдано реальних збитків.

4) Виявлення аномалій.

Системи штучного інтелекту також можуть виявляти аномалії в мережевому трафіку або інших потоках даних, аналізуючи шаблони щодо подібності або відмінності між очікуваною та реальною поведінкою. Такий тип моніторингу може допомогти виявити аномальну поведінку до того, як вона переросте у шкідливу діяльність – наприклад, хтось спробує отримати доступ до конфіденційної інформації.

5) Автоматизація безпеки.

Ще один спосіб, за допомогою якого ШІ може допомогти захистити підприємства від кібератак, - автоматизація та впровадження нових політик та протоколів безпеки. Він захищає від основних кібератак, таких як погрози спуфінгу, фішингу та інших. Автоматизація впровадження кібербезпеки може допомогти вам:

- зекономити час та зусилля;
- скоротити людський чинник;

- надати економічно-ефективні рішення із ККД близьким до 100%;
- забезпечити одразу помітні результати роботи.

6) Оркестрування безпеки.

ШІ може автоматизувати багато рутинних завдань, які сьогодні забирають час та ресурси, наприклад, виявлення аномальної поведінки або виявлення підозрілих користувачів у вашій мережі.

ШІ також може використовуватися в багатьох інших галузях [23], включаючи, крім іншого, охорону здоров'я, ігри, управління ризиками, фінанси, освіта, безпілотні літальній апарати і багато іншого (рис. 1.1).

Рисунок 1.1 – Сфери використання штучного інтелекту

1.2 Дослідження методів нейронних мереж

Штучні нейронні мережі з'явилися в процесі дослідження можливості створення штучного інтелекту. Якщо казати конкретніше, зі спроб відтворити вміння біологічної нейронної мережі навчатися і знаходити помилки, ґрунтуючись на попередніх даних, моделюючи, таким чином, на низькому рівні мозок живої істоти. Мозок складається з величезної кількості нейронів (приблизно 10 мільярдів), на кожен із яких припадає кілька тисяч зв'язків. Нейрони – спеціальні клітини, які вміють розпізнавати електрохімічні сигнали. У кожного нейрона є

відростки двох типів, що виконують різні функції: дендрити, відповідальні за отримання сигналів і один аксон, що виконує функцію передачі імпульсу іншим нейронам (рис. 1.2). Місця з'єднання аксона однієї нервової клітини з дендритами іншої називаються синапсами. Сила нервового імпульсу, що проходить через синапс, залежить від кількості нейромедіатора – біологічно активної хімічної речовини, що виробляється у синапсі. Також при проходженні через синапс змінюється і сила імпульсу в кілька разів. Це число в теорії штучних нейронних мереж називають вагою синапсу. Пройшовши по кількох дендритах одночасно імпульси підсумовуються [6].

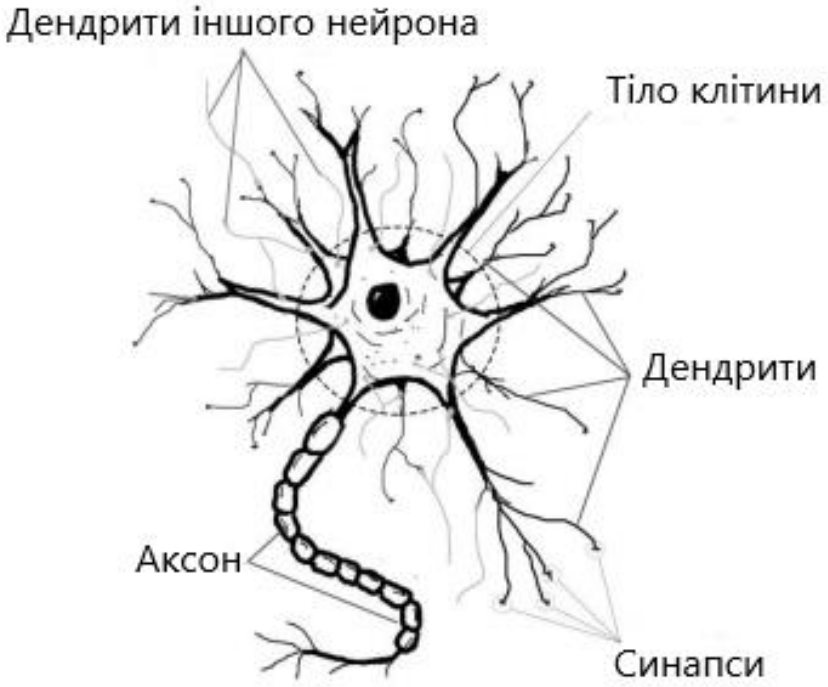

Рисунок 1.2 – Схематичне зображення нейрона

Коли сумарний імпульс стає вищим ніж деяке порогове значення, нейрон переходить у стан збудження, у ньому формується свій імпульс, який далі передається за аксоном.

З часом ваги синапсу можуть змінюватися, отже, змінюється і реакція відповідного нейрона. Можна зобразити математичну модель описаного вище процесу (рис. 1.3):

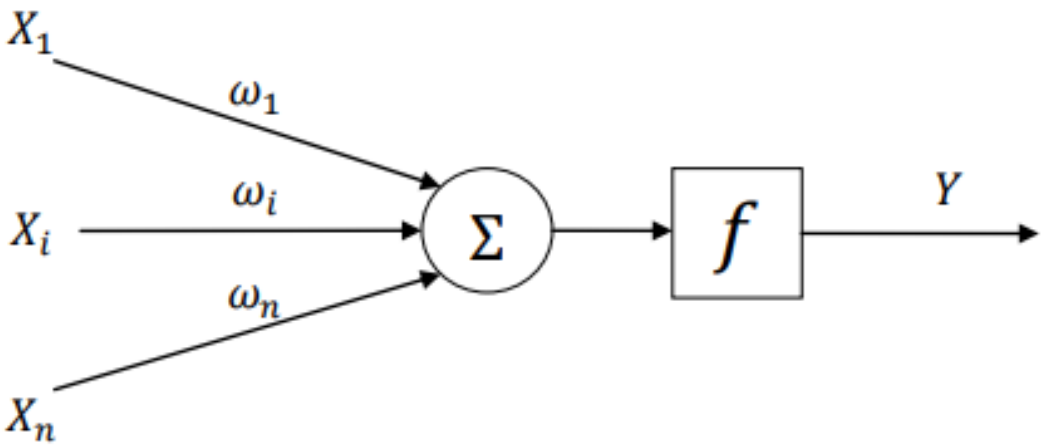

Рисунок 1.3 – Математична модель нейрона

Є входи-дендрити, при цьому синапсам цих дендритів відповідають ваги ω_1, …, ω_i, …, ω_n. На синапси надходять імпульси з силами X_1, …, X_i, …, X_n відповідно. Отже, після проходження синапсів та дендритів до нейрона надходять ваги $\omega_1 X_1$, …, $\omega_i X_i$, …, $\omega_n X_n$.

Нейрон підсумовує отриманий імпульс (1.1):

$$X = \omega_1 X_1 + \cdots + \omega_i X_i + \cdots + \omega_n X_n = \sum_{i=1}^{n} \omega_i X_i \quad (1.1)$$

і змінює його за допомогою деякого нелінійного перетворення f(X) – так званої "функції активації". Крім описаних вище параметрів, у моделі нейрона є така характеристика, як пороговий рівень – θ, що має те саме значення, що і в живому нейроні. Сукупність параметрів всіх нейронів є параметрами нейронної мережі [6].

Таким чином, вихідний імпульс матиме силу (1.2):

$$Y = f(X - \theta) = f(\sum_{i=1}^{n} \omega_i X_i - \theta) \quad (1.2)$$

Тобто можна сказати, що нейрон повністю описують ваги ω_1, …, ω_i, …, ω_n, пороговий рівень θ та передатна функція нелінійного перетворення f(X). У якості входів нейрон отримує деякий вектор чисел X_n, при цьому на виході видаючи число Y.

Існує також модифікація моделі без порогового значення. У ній до нейрону додається додатковий вхід $X_j = 1$ для будь-якого вхідного через нього сигналу, а вага $\omega_j = -\theta$.

Зрозуміло, такі моделі є еквівалентними (1.3):

$$f(\sum_{i=1}^{n} \omega_i X_i - \theta) = f\left(\sum_{i \neq j}^{n} \omega_i X_i - \omega_j X_j\right) \quad (1.3)$$

Описана модель нейрона є класичною, і має назву "формальний нейрон". Крім неї існує безліч інших моделей нейрона, що відрізняються складністю обчислень та ступенем подібності до живого нейрона [7].

При побудові даної моделі передбачається, що вихід обчислюється нейроном миттєво, тому за її допомогою неможливо створювати моделі систем, у яких істотний внутрішній стан.

У даної моделі є і свою певні недоліки:

- формальні нейрони відрізняються від живих тим, що не можуть синхронно обробляти вхідні дані;
- відсутні чіткі алгоритми для вибору активної функції;
- неможливе регулювання процесу роботи всієї нейронної мережі загалом;
- поняття «вагові коефіцієнти» та «поріг» у моделі є надмірно формалізованими.

1.3 Дослідження методів нечіткої логіки

Нечітка логіка – це логічна чи керуюча система n-значної логічної системи, яка використовує ступені стану (ступені правди) входів та формує виходи, що залежать від станів входів та швидкості зміни цих станів. Це не булева (двійкова) логіка, на якій ґрунтуються сучасні комп'ютери. Вона переважно забезпечує основи для приблизного міркування з використанням неточних рішень і дозволяє використовувати лінгвістичні змінні.

Нечітка логіка була розроблена в 1965 професором Лотфі Заде в Каліфорнійському університеті в Берклі. Першим додатком було виконання обробки комп'ютерних даних з урахуванням природних значень [8].

Якщо говорити простіше, станами нечіткої логіки може бути не тільки 1 чи 0, а й значення з-поміж них, тобто 0.15, 0.8, тощо.

Нечітка операція передбачає використання нечітких множин та функцій приналежності. Кожна нечітка множина є уявленням лінгвістичної змінної, яка визначає можливий стан виводу. Функція приналежності є функцією загального

значення в нечіткій множині, тому і загальне значення, і нечітка множина належать універсальній множині.

Ступені приналежності у цьому загальному значенні в нечіткій множині визначають вихід, заснований на принципі if-then. Приналежність призначається на основі припущення про вихід за допомогою входів та швидкості зміни вхідних даних. Функція власності переважно є графічним уявлення нечіткої множини.

Алгоритми нечіткої логіки засновані на наборі правил, передбачених користувачем. Завдяки своїй простоті і гнучкості, методи нечіткої логіки використовуються в області комп'ютерної безпеки, особливо при виявленні вторгнення. Концепція нечіткості допомагає згладити різке відділення звичайної поведінки від незвичайної поведінки. Нечітка логіка вирішує проблеми з неповними і неточними даними. Отже, вона може відображати ці неточні форми міркувань в деяких областях, де суворе рішення повинно прийматися в невизначеному середовищі, такому як виявлення вторгнень.

Багато дослідників сходяться у висновку, що комбінація деяких технік ШІ - спосіб, який може істотно поліпшити продуктивність IDS/IPS. Була винайдена нова модель IDS/IPS на основі нечіткої логіки і нейронної мережі [20]. У нечіткій системі використовувався певний набір правил для класифікації даних випробувань як нормальна або аномальна поведінка. Виявлялася поведінка вторгнення в мережі. Тоді як нейронна мережа навчає систему і тестує дані для виявлення вторгнень. В ході експериментів було встановлено, що ефективність запропонованої моделі з точки зору отримання високої точності при виявленні атак – зросла [8].

1.4 Дослідження методів генетичного алгоритму

Генетичні алгоритми (ГА) відносяться до одного з напрямів еволюційних обчислень - біонічного напрямку, що використовує існуючі в природі закономірності, спрощені таким чином, щоб їх можна було реалізувати в комп'ютерних системах [25]. Основним механізмом еволюції є відбір. Усі біологічні види розвиваються з різним ступенем пристосованості до навколишнього середовища; при цьому більш пристосовані особи отримують

більше можливостей для виживання. При генерації нових поколінь біологічних видів працює механізм генетичного наслідування, згідно з яким нащадки успадковують від батьків деякі властивості. Очевидно, що кількість нащадків більш пристосованих батьків буде збільшуватися, тому через кілька десятків чи сотень нових поколінь середня пристосованість такої групи зросте. Науковий напрямок, що вивчає закони та механізми спадковості називається генетикою.

Якщо говорити про інформаційну сферу, то генетичні алгоритми – це адаптивні алгоритми пошуку, які часто використовуються для вирішення завдань функціональної та структурної оптимізації. ГА широко використовується в області виявлення вторгнень для розпізнавання нормального мережевого трафіку від аномального. Основні засади ГА були сформульовані американським математиком Дж. Холландом в 1975 р. ГА моделюють біологічні процеси та за аналогією з еволюційним механізмом працюють з населенням, кожна з хромосом якої представляє собою можливе вирішення цього завдання. Кожна хромосома оцінюється її «пристосованістю», яку також називають функцією оптимальності. Найбільш пристосовані особини отримують можливість «відтворити» потомство з іншими особами популяції за допомогою «перехресного схрещування». У результаті з'являються нові особини, що поєднують у собі показники, успадковані від їх батьків. Особи, які найменш пристосовані поступово зникають з популяції в процесі еволюції. Нове покоління має кращі характеристики у порівнянні з попереднім. Схрещування найкраще-пристосованих осіб приводить до того, що еволюція шукає перспективні рішення у широкому просторі пошуку [9].

Багато дослідників використовували ГА для виявлення вторгнення за допомогою різних моделей. Деякі дослідники використовували його безпосередньо для отримання правил класифікації, інші використовували його для вибору зручних функцій, в той час як треті використовували різні методи інтелектуального аналізу даних для подальшого отримання правил класифікації.

1.5 Порівняльний аналіз алгоритмів штучного інтелекту

Проаналізувавши три найпоширеніші алгоритми штучного інтелекту, різноманітні методи їх реалізації та методи виявлення вторгнень за їх допомогою,

можна виділити наступні переваги й недоліки технологій штучного інтелекту (Табл.1):

Таблиця 1.1 – Переваги й недоліки технологій штучного інтелекту [1]

Технологія	Переваги	Недоліки
Нейронні мережі	- ефективно класифікує неструктуровані мережеві пакети; - багатошаровість в NN збільшує ефективність класифікації;	- потрібен довгий час для навчання; - потрібна велика кількість зразків для ефективного навчання;
Нечітка логіка	- проста реалізація алгоритму; - має кращу гнучкість для деяких проблем;	- точність виявлення нижче ніж у NN;
Генетичний алгоритм	- використовується для вибору кращих функції виявлення вторгнень; - має кращу ефективність;	- складна реалізація алгоритму; - використовується в конкретних випадках, а не в загальному;

1.6 Висновки по розділу №1

Таким чином, якщо проаналізувати всі переваги й недоліки зазначених технологій, для дослідження системи штучного інтелекту можна обрати систему на базі нечіткої логіки у якості створення навчальної вибірки даних для нейронної мережі, яка саме і буде виявляти загрози у мережі. Виходячи з наявних можливостей апаратної реалізації, цей метод оптимально-трудомісткий, а також він дозволяє створити максимально ефективну систему виявлення загроз завдяки комбінації двох алгоритмів штучного інтелекту.

2 АНАЛІЗ АРХІТЕКТУРИ НЕЙРОННИХ МЕРЕЖ ТА ЇХ КЛАСИФІКАЦІЯ

2.1 Одношаровий перцептрон

Один нейрон здатний виконувати елементарні обчислення, проте базові функції нейронної мережі забезпечуються не окремо взятими нейронами, а цілими групами нейронів зі зв'язками один з одним усередині кожної. Взагалі, найпростіша штучна нейронна мережа є групою розташованих паралельно нейронів, на вхід кожного з яких подається кожен елемент вектору вхідних даних $X = (X_1, \ldots, X_i)$ (рис. 2.1). Така мережа називається одношаровим перцептроном. Її суть у тому, що вихідне значення обчислюється незалежними один від одного нейронами. У такому випадку, розмірність виходу дорівнюватиме кількості нейронів в мережі, при цьому, у всіх нейронів має бути однакова кількість синапсів, що також збігається з розміром сигналу, що подається на вхід [10].

Одношаровий перцептрон здається дуже простою системою, але при цьому здатний виконувати ряд корисних функцій, таких як обчислення значень логічних виразів чи класифікація образів (мережевих пакетів у нашому випадку).

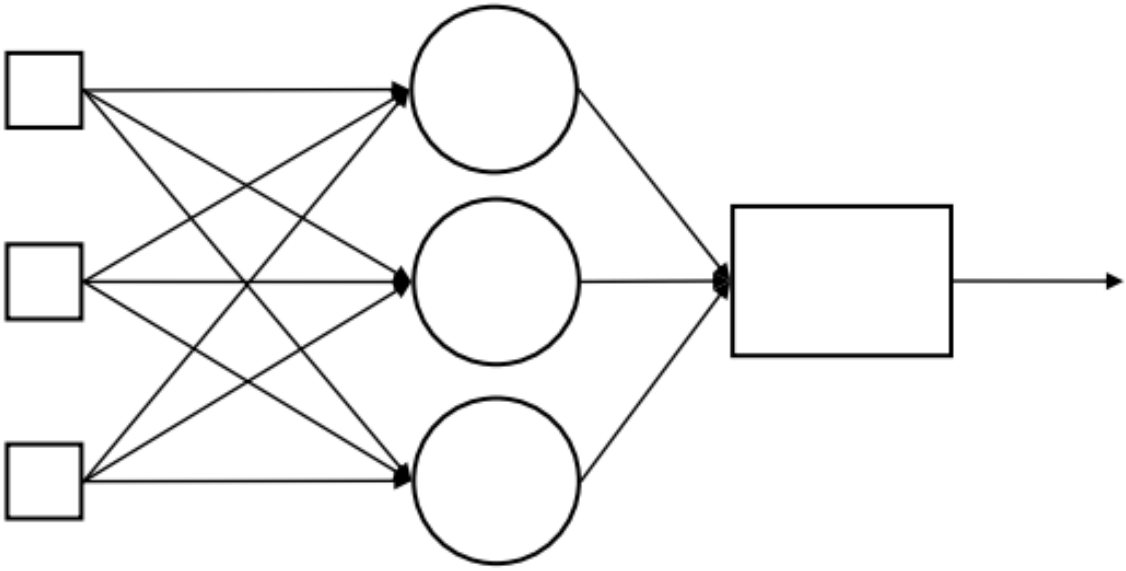

Рисунок 2.1 – Схема одношарового перцептрону

2.2 Багатошаровий перцептрон

Багатошаровий перцептрон обчислює вихід Y для входу X. Тобто, мережа знаходить значення векторної функції Y = F(X). Умова завдання, що ставиться перед перцептроном, формулюється у вигляді набору векторів $\{X^1, X^2, \dots, X^p\}$. Рішення ж завдання представимо у вигляді множини $\{Y^1, \dots, Y^p\}$, при цьому $\forall p Y^p = F(X^p)$ [10].

Мережа формує відображення $F: X \to Y, \forall X^j \in X (j = \overline{1,p})$. Можливості отримати повне відображення з перцептрону немає, але при цьому можна підрахувати значення образів для довільного числа вхідних параметрів. Для нейронних мереж відсутні точні методики формалізації, тобто з'ясування сенсу компонент векторів входу та виходу, тому на даний момент це завдання вирішується дослідниками емпіричним шляхом [10].

Для побудови багатошарового перцептрону параметри вибираються наступним чином:

1) Встановлюється значення, що вкладається в множину компонентів вектору X. На вхід повинна подаватися вся необхідна для отримання відповіді інформація, іншими словами, необхідно, щоб вектор входу містив формалізовану умову задачі.
2) Вихід вибирається так, щоб у його компонентах містилася повна відповідь до завдання.
3) Підбирається відповідний вид функції активації. Важливо враховувати, що правильний вибір прискорює процес навчання, тому слід враховувати особливості поставленої задачі.
4) Визначається кількість шарів перцептрону та кількість нейронів у кожному шарі.
5) Ґрунтуючись на виборі функції перетворення, встановлюється область, у межах якої можливі зміни значень для вхідних та вихідних параметрів, вагових коефіцієнтів та порогового значення.
6) Ваговим коефіцієнтам та пороговим значенням присвоюються початкові значення. До них виставляються такі вимоги: вони не повинні бути як надто

великими, так і занадто малими, інакше навчання мережі сповільниться (у першому випадку нейрони будуть перенасичені, у другому ж - більшість виходів дорівнюватиме нулю).

7) Проводиться навчання мережі, іншими словами, підбираються параметри для найкращого розв'язання задачі. Згодом мережа буде вміти вирішувати завдання, подібні до тих, на яких вона навчалася.
8) На вхід перцептрону подається вектор, що є умовою задачі. Формалізоване вирішення цього завдання буде знайдено при розрахунку виходу.

2.3 Класифікація нейронних мереж

Нейронні мережі діляться на класи відповідно до наступних своїх характеристик:

1) Топологія мережі:
   - У повнозв'язкових мережах кожен нейрон має зв’язок з усіма іншими (на даний момент такі мережі не використовують у зв'язки із високою складністю навчання).
   - У розшарованих мережах нейрони розташовані окремими шарами, при цьому всі нейрони одного шару мають зв'язок з нейронами іншого. Існують одношарові та багатошарові нейронні мережі (найпростіші моделі описані вище).
2) Характер зв'язку нейронів:
   - У нейронних мережах прямого поширення вихід попереднього шару мережі пов'язаний із входом наступного шару.
   - У рекурентних нейронних мережах можливі зв'язки між виходами наступних шарів та входами попередніх.
3) Тип навчання мережі:
   - Етап навчання з учителем має на увазі навчання на вибірках, для яких відомі вихідні значення. Ці мережі часто застосовують задля вирішення завдань класифікації.

- У випадку навчання мережі без вчителя вона сама знаходить особливості між об'єктами та на основі цього ділить їх на класи. Дані мережі застосовуються задля вирішення завдань кластеризації.
- У змішаному навчанні комбінуються обидва підходи.

4) Вид вхідного сигналу:
   - Бінарні нейронні мережі одержують на вхід сигнал, що складається лише з нулів та одиниць.
   - Аналоговим нейронним мережам на вхід подаються довільні сигнали (дійсні числа).
5) Структура мережі:
   - Однорідна нейронна мережа – нейронна мережа, для всіх нейронів якої застосовується однакова функція активації.
   - У неоднорідній нейронній мережі нейрони можуть мати різні активуючі функції.

2.4 Висновки по розділу №2

Отже, проаналізувавши види архітектури нейронних мереж, проаналізувавши їх переваги та недоліки, для власної практичної реалізації даного алгоритму штучного інтелекту було обрано архітектуру багатошарового перцептрону. За іншими показниками класифікації нейронних мереж майбутню реалізацію можна охарактеризувати як повнозв’язну аналогову однорідну мережу з прямим поширенням та алгоритмом навчання з вчителем.

3 НАВЧАННЯ НЕЙРОННОЇ МЕРЕЖІ

3.1 Загальний опис варіантів навчання нейронних мереж

Вміння навчатися є головною рисою людського мозку. У ситуації ШНМ, навчання складається з підбору необхідної архітектури мережі та вагових коефіцієнтів між зв'язками нейронів для ефективного виконання завдання, що поставлено перед мережею. Нейронна мережа, що заснована на штучному інтелекті, підбирає ваги, отримані шляхом аналізу даних, що надані їй для навчання. Це дає перевагу перед системами та мережами де дані закладаються на початку. Стандартне навчання нейронних мереж відбувається на навчальній вибірці. Навчання відбувається за певним алгоритмом. З кожним проходом по алгоритму мережа краще реагує на вхідні дані і все точніше виводить вихідні дані.

Стабілізація вагових коефіцієнтів здійснюється після багаторазового прогону прикладів, у той час як нейронна мережа дає вірні відповіді на більшість прикладів навчальної вибірки. Тобто, нейронна мережа вивчає всі надані приклади, отже стає навченою та готовою до експлуатації. За допомогою програмних реалізацій можна побачити, що величина помилки з ходом процесу навчання зменшується. Якщо помилка дорівнює нулю, тренування мережі зупиняють і вона є навченою і готовою до використання на реальних даних [12].

Також варто виділити, що інформація про завдання, що надана нейронній мережі, міститься в наборі прикладів. Таким чином, ефективність навчання нейронних мереж залежна від якості та кількості прикладів у тестовій вибірці, що призначена для навчання, а також ступеня повноти постанови завдання цими прикладами. Немає сенсу використовувати нейронну мережу для передбачення мережевої атаки, якщо у тестовій вибірці прикладів атак не надається. Можемо вважати, що для ефективного тренування більшості нейронних мереж необхідно кілька десятків прикладів, а краще кілька сотень або навіть тисяч.

Щодо самих алгоритмів навчання - вони можуть мати різні параметри та налаштування. Для їх керування необхідне розуміння їх майбутнього впливу на вихідні дані.

Існуючі на даний момент методи навчання нейронних мереж поділяються на такі класи [14]:

- Детермінований метод – метод, за якого параметри нейронної мережі змінюються ітеративно, залежно від значень параметрів, вхідних даних, а також ґрунтуючись на значеннях виходу: тих, які необхідно отримати та тих, які вийшли у процесі обчислень. Найбільш яскравим прикладом є алгоритм зворотного розповсюдження помилки (АЗРП).
- У стохастичних методах параметри мережі змінюються без будь-яких закономірностей - випадково. Зміни приймаються лише в тому випадку, коли вони призводять до покращення результатів. До проблем даного методу навчання відноситься так звана «пастка локального мінімуму» (рис. 3.1) [27]. Припустимо, що значення помилки спочатку близьке до точки M_1, або дорівнює її значенню. При малих довільних кроках корекції параметрів будь-які відхилення від неї збільшать значення помилки, отже, зміну коефіцієнтів не буде прийнято. Виходить, що неможливо знайти найменше значення помилки в точці M_2. Якщо ж випадкові кроки коригування досить великі, помилка не зможе встановитися в одному з двох мінімумів, оскільки змінюватиметься занадто різко.

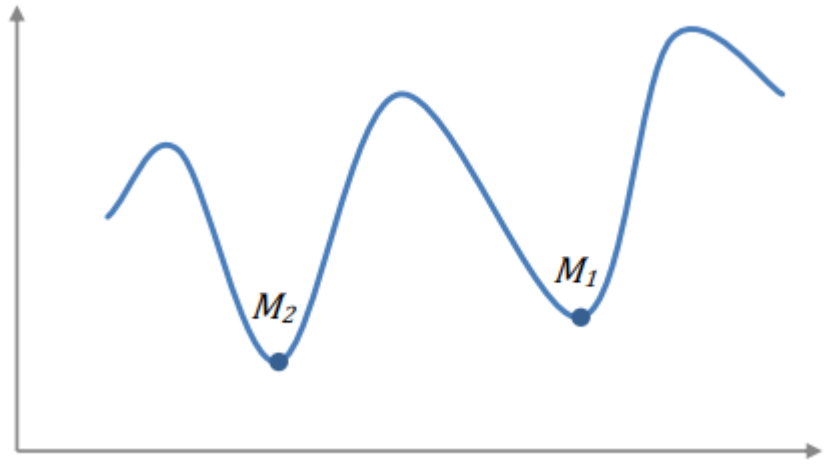

Рисунок 3.1 - Проблема локальних мінімумів

Як варіант вирішення проблеми передбачається поступове зменшення середнього розміру випадкових коригувальних кроків. З рівною ймовірністю значення помилки приймає всі можливі значення при більших довільних кроках. У разі коли розмір випадкових кроків коригування змінюється поступово, значення помилки буде зупинятись у точці M_2 протягом деякого часу. При подальшому зменшенні кроку коригування значення помилки може «застрягати» не тільки у точці M_2, а й у M_1. Подальше безперервне зменшення кроку може призвести до результату, коли долається локальний мінімум M_1, при цьому потрапляючи в локальний мінімум M_2.

3.2 Алгоритми навчання мереж

Алгоритм навчання – це процедура, що використовує правила навчання для налаштування вагових коефіцієнтів. Найбільш поширеними алгоритмами є [16]:

- встановлення явних відхилень;
- квазі-Ньютонівський;
- покроковий прямий або зворотний відбір вхідних даних;
- навчання Кохонена;
- лінійний дискримінантний аналіз;
- узагальнено-регресійна нейронна мережа;
- Левенберга-Марквардта;
- встановлення ізотропних відхилень;
- радіальна (під-) вибірка;
- навчальний векторний квантувальник;
- метод «k-найближчих сусідів (KNN)»;
- генетичний алгоритм відбору вхідних даних;
- сполучених градієнтів;
- позначка найближчих класів;
- псевдо-зворотний;
- проекції Саммона;
- зворотне поширення;
- метод «K-середніх»;

- швидке поширення;
- аналіз основних компонентів;
- ймовірнісна нейронна мережа;

Після того, як буде визначено кількість шарів та нейронів у кожному з них, потрібно знайти вагові коефіцієнти нейронної мережі; дані значення повинні зменшити помилкові прогнози, які висуває мережа. Саме для цієї мети й потрібні алгоритми навчання. Процес навчання характеризується налаштуванням моделі, що реалізована мережею за допомогою навчальних даних, які ви їй надаєте. Визначення помилки для мережі відбувається шляхом порівняння очікуваних значень з реальними вихідними значеннями. Перевага надається тим алгоритмам, які дозволяють навчити мережу за меншу кількість кроків та потребують невеликої кількості додаткових параметрів. Даний вибір пов'язаний з можливістю здійснення навчання на машинах з порівняно малою продуктивністю та невеликим обсягом оперативної пам'яті. Переважна більшість алгоритмів навчання будуються на функції, яка оцінює ефективність роботи мережі. Алгоритм підлаштовує параметри системи, що оновлюються в залежності від значень оцінки, яка була отримана.

Теорія про навчання нейронних мереж включає розгляд трьох основних характеристик процесу навчання за прикладами: складність обчислень, складність навчальної вибірки та ємність.

Остання являє собою інформацію про те, як багато зразків з вибірки мережа здатна запам'ятати, а також про функції та граничні значення в прийнятті рішень.

Складність навчальної вибірки показує, скільки зразків необхідно мережі для отримання здатності до узагальнення. Якщо набір прикладів буде занадто малим, то мережа може виявитися "перенавченою": на навчальній вибірці вона добре функціонуватиме, при цьому погано функціонуватиме на тестових зразках, схильних до подібного статистичного розподілу.

3.3 Концептуальні підходи до навчання мереж.

- Навчання з учителем.

Процес навчання з учителем заснований на тому, що для кожного вхідного вектору з множини буде існувати цільовий вектор, який і являє собою бажаний

вихід [21]. Іншими словами, задається масив пар векторів$\{(X^M, S^M)\}$, де $X^M \epsilon X$ - вектор, що описує завдання, а $S^M \epsilon Y$ – вже відоме для заданого вектору X^Mрішення задачі. Така пара називається навчальною. Для навчання мережі зазвичай потрібна деяка кількість таких пар. Навчання відбувається наступним чином: обчислюється вихід мережі, ґрунтуючись на наданій інформації про вихідний вектор, після чого вихід і відповідний йому цільовий вектор порівнюються. Різниця цих значень (помилка) зчитується мережею з допомогою зворотного зв'язку, після чого вагові коефіцієнти змінюються за принципом алгоритму, обраного для мінімізації помилки. Інакше кажучи, мережа змінює свої параметри так, щоб виходило потрібне відображення $X \rightarrow Y$. При цьому важливо враховувати, що розмір набору $\{(X^M, S^M)\}$ повинен бути досить великим, щоб алгоритм сформував бажане відображення. Вектори з навчального набору надаються мережі по черзі, після чого відбувається обчислення помилок та налаштування вагового коефіцієнта кожного вектору до того моменту, поки загальна помилка на всьому навчальному наборі не буде на досить низькому рівні. Варто зазначити, що корекція вагових коефіцієнтів відбувається лише за умови помилкової відповіді. Ця модель вперше була запропонована Френком Розенблаттом у 1957 році [13].

Хоча навчання з учителем і досягло певної популярності застосування, воно піддалося критиці через біологічну неправдоподібність моделі. Неможливо уявити в мозку механізм навчання, здатний порівнювати необхідні та дійсні значення виходів, після чого коригувати дані за допомогою зворотного зв'язку.

- Навчання без учителя.

З точки зору біології, навчання без вчителя є більш правдоподібною моделлю. У процесі дослідження цієї моделі навчання Тойво Кохоненом, а також багатьма іншими, встановлено, що вона не вимагає цільового вектору виходів, отже, немає необхідності порівнювати отриманий вихід із заздалегідь еталонними відповідями [22].

Масивом навчальних даних є лише вхідні вектори. Найкраще значення для виходу визначається самим алгоритмом. Зазвичай він підлаштовує вагові коефіцієнти таким чином, щоб на виході вектори виходили узгодженими, тобто,

якщо вхідні вектори будуть досить близькими, їх надання алгоритму повинно давати схожі вихідні вектори. Під час навчання мережею виділяються характерні риси навчальної вибірки, на основі чого схожі вектори формують деякі групи. При надходженні вхідного вектору з даної групи, алгоритм дає якийсь конкретний вектор виходу, але на початок процесу навчання неясно, який саме вихід вийде при напрямі у мережу цієї групи вхідних векторів вибірки. Тобто, важливе питання інтерпретації виходу мережі засновується на процесі навчання. Це представляється цілком можливим, оскільки визначити зв'язок входу та виходу, встановлених мережею, зазвичай нескладно.

3.4 Інші методики навчання

- Метод Хебба

Більшість існуючих на даний момент алгоритмів навчання засновані на концепціях Дональда Хебба, вперше описаних ним ще 1949 року [26]. Спираючись на свої дослідження в галузі психології та фізіології, він припустив, що вага сполуки зростатиме при активації нейрона-джерела та нейрона-приймача сигналу. Отже, шляхи, які найчастіше використовують проходження сигналу в мережі, отримують більш високий пріоритет, що пояснює такі явища, як навчання через повторення однакових дій та існування звичок. Штучна нейронна мережа, заснована на принципі навчання за Хеббом, визначає вагові коефіцієнти як добуток рівнів збудження нейрона, від якого йде сигнал та нейрона, який цей сигнал отримує [15].

Математично це описується так (3.1):

$$\omega_{ij}(m+1) = \omega_{ij}(m) + \gamma W_i W_j, \tag{3.1}$$

де:

- $\omega_{ij}(m)$ – значення вагового коефіцієнта від нейрона і до нейрона j до налаштування;
- γ - коефіцієнт, що характеризує швидкість навчання;
- $\omega_{ij}(m+1)$ – значення вагового коефіцієнта від нейрона і до нейрона j після процесу налаштування;

- $W_i -$ вихід і та вхід j;
- $W_j -$ вихід j.

Навчання за Хеббом активно використовувалося протягом тривалого часу. Тим не менш, існують і більш ефективні методики навчання мереж, розроблені та удосконалені за останні 20 років. Наприклад, розвиток отримали алгоритми, засновані на принципі навчання з учителем, які мають більш широкий набір характеристик для навчальної вибірки, і які мають більші швидкості, ніж моделі навчання, засновані на методиках Хебба.

- Змагальний метод навчання

У навчанні Хебба велика кількість вихідних нейронів мають можливість одночасно збуджуватися на відміну від нейронів при навчанні методом змагання, де їм доводиться конкурувати між собою за можливість активації. Іншими словами, з усіх вихідних нейронів вибирається один, який має найбільший вихід. Подібний алгоритм схожий з методиками навчання живих нейронних мереж. За допомогою нього з'являється можливість класифікації вхідних сигналів: з подібних зразків мережею складається група, яка вже сама служить у якості єдиного сигналу-зразка. Важливо, що кожен вихідний нейрон «несе відповідальність» лише за один клас. Відповідно, кількість класів, з якими нейронна мережа може взаємодіяти, еквівалентна числу нейронів виходу [12].

У процесі цього навчання ваги змінюються тільки у нейрона, який «отримав перемогу», тому зразок і вхідний сигнал стають трохи більш схожими.

- Генетичні алгоритми

Генетичними алгоритмами називають клас алгоритмів, що базується на моделях процесів розвитку біологічної популяції. Нехай популяція - множина векторів $P = \{p^1, \dots , p^L\} = \{p^l\}$, при цьому L визначає розмір популяції. Кожен вектор p^l позначає особу і є набором параметрів, що описують її. Необхідно знайти мінімум функції $E = E(p^l)$, що обчислює помилку [30].

Також визначені правила, згідно з якими кожна складова множини має здатність до еволюційних змін:

- У разі, коли значення функції $E = E(p^0)$ досить мале, особа, відповідна вектору, визначається як вдала та розмноження цієї особи має підвищений пріоритет.
- У протилежному випадку, коли $E = E(p^0)$ досить велике, це означає, що p^0 - невдала особа, отже, вона має підвищену ймовірність загибелі та малу ймовірність здатності розмножуватися.
- Також визначено мутаційні процеси у межах популяції: кожен елемент множини P здатний мутувати, отже, міняти своє значення на деяку малу величину: $p^{0^*} = p^0 + \Delta p$. Δp є вектором, який характеризує ступінь мутації, при цьому має невелике за модулем значення.
- Процес розмноження полягає у розподілі кожної точки (особи), ґрунтуючись на певній ймовірності розмноження. Закони, згідно яким відбувається розмноження, визначаються обраною моделлю.
- Загибель особи можлива відповідно до певної ймовірності загибелі. Іншими словами, точка видаляється з множини векторів P.

На даний час немає точної моделі для теорії еволюції, відповідно, відсутні правила, за якими можна обчислити точні ймовірнісні значення для розмноження та загибелі осіб. Тому визначити, які алгоритми є найкращими, можна лише емпірично.

3.5 Методи збільшення швидкості процесу навчання мережі.

Нейронні мережі – це напрям штучного інтелекту, що часто використовується для вирішення завдань розпізнавання образів та оптимізації. В наш час існує багато моделей нейронних мереж, що призначені для вирішення прикладних завдань [28]. Кожна модель має запропоновані методи навчання. Але тим не менш, роботи зі створення нових та удосконалення наявних моделей не припиняються.

Можна виділити 2 основні етапи у створенні нейронної мережі:

1) Етап вибору моделі мережі, її структури та алгоритму навчання. Також даний етап називають структурним синтезом.

2) Етап навчання нейронної мережі та перевірки результатів. Також даний етап називають «параметричний синтез».

Залежно від результатів перевірки приймається рішення щодо повернення на попередні стадії структурного або параметричного синтезу. Є кілька методів, що дозволяють зменшити час навчання багатошарових нейронних мереж та нейронних мереж, заснованих на принципі достатності (помилка мережі не більше певного значення). В якості таких методів розглянемо зміну вагових коефіцієнтів мережі, коригування кроку зміни вагових коефіцієнтів та реорганізацію класів.

Навчання мережі триває доки помилка не буде близька до нуля. Зазвичай це веде до великих витрат тимчасових ресурсів через те, що під час етапу навчання помилка не перевищувала значень сильно віддалених від нуля [13].

Виходячи з умови завдання визначається ступінь достатності навчання. Ідея принципу достатності передбачає відмову від прагнення до ідеалу під час пошуку рішення. Тобто мається на увазі, що 100% точного підбору рішення потрібно не завжди. Іноді буває достатньо, щоб помилка не перевищувала деякого заданого δ, максимального значення, що залежить від навчальної вибірки.

Повноту і суперечливість навчальної вибірки розглянемо в якості її характеризуючих параметрів. Забезпеченість класів навчальними наборами характеризується повнотою. Кількість навчальних наборів для кожного класу має у 4-5 разів перевищувати кількість ознак, які містяться у даних наборах. Розрахунок повноти можна зробити за такою формулою (3.2):

$$F_{0.\ B.} = \frac{n_{F_{0.\ B.}}}{N}, \tag{3.2}$$

де:

- $n_{F_{0.\ B.}}$ – кількість класів, які відповідають правилу, описаному вище;
- N - кількість усіх класів.

Набори, що містять об'єкти, визначені до різних класів, і що містять однакові класифікаційні ознаки, вважаються суперечливими. Суперечливість усієї навчальної вибірки обчислюється за такою формулою (3.3):

$$I_{O.\ B.} = \frac{M_I}{M}, \tag{3.3}$$

де:

- $M_I -$ число наборів, що містять протиріччя,
- $M -$ кількість всіх наборів вибірки.

Отже, від того, наскільки більшими будуть величини $F_{0.\ B.}$ і $I_{O.\ B.}$, залежить, наскільки великі значення може набувати величина δ, і, відповідно, наскільки швидко проходитиме навчання мережі.

3.6 Алгоритм зворотного розповсюдження помилки

Навчання, засноване лише на значеннях помилок виходів із мережі можна покращити шляхом АЗРП [1]. Даний алгоритм можна реалізувати визначивши набори сигналів на виході з кожного шару мережі, що є енерговитратним та не завжди можливим процесом. Крім того, можливо підлаштовувати ваги динамічно, при цьому, зазвичай, вибирають найслабші зв'язки та задають їм незначних змін, які зберігаються лише в тому випадку, коли мережа почала краще працювати, тобто, зменшилася помилка на виході. Цей емпіричний метод є трудомістким обчислювальним процесом, незважаючи на простоту описаних дій.

Найбільш привабливим варіантом є поширення помилки від виходів штучної нейронної мережі до її входів, що є зворотним напрямом щодо нормального робочого режиму мережі. При навчанні нейронної мережі АЗРП по всіх шарах мережі виробляється два проходи: прямий та зворотний. При першому проводиться прийом вхідних даних, а також поширення їх у напрямі виходів. Під час другого етапу обчислюється помилка, яка згодом поширюється назад, а також коригуються вагові коефіцієнти. Коли процес навчання мережі буде закінчено, дані будуть поширюватись лише у прямому напрямку. Саме цей алгоритм я буду використовувати у процесі навчання моєї нейронної мережі [13, 14].

Важливо відзначити, що хоча тренування мережі може тривати досить довго, але при цьому вже навчена мережа здатна швидко обчислювати результати за отриманими даними.

Навчання багатошарового перцептрону в АЗРП засновано на мінімізації функції помилки мережі E(w). Помилка визначає відхил бажаних виходів t^n від

отриманих y^n. Зазвичай функція E(w) обчислюється за методом найменших квадратів [29]:

$$E(w) = \frac{1}{2}\sum_n (t^n - y^n)^2 \tag{3.4}$$

Головна ідея методу полягає в тому, що коли ми рухаємось у напрямку, протилежному градієнту функції Δw_{ji}, то ми наближаємось до мінімуму, можливо локального, функції E(w):

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}}, \tag{3.5}$$

де $0 < \eta < 1$ – задає швидкість навчання;

w_{ji} - коефіцієнт зв'язку i-го нейрона шару $n-1$ з j-им нейроном шару n.

Кожен нейрон розраховує виважену суму своїх входів:

$$a_i = \sum_n w_{ij} c_i, \tag{3.6}$$

де c_i – це входи нейрона та виходи попереднього шару нейронів відповідно до рисунка 3.2.

Вихід нейрона j – перетворення суми a_i граничної функції g:

$$c_i = g(a_j) \tag{3.7}$$

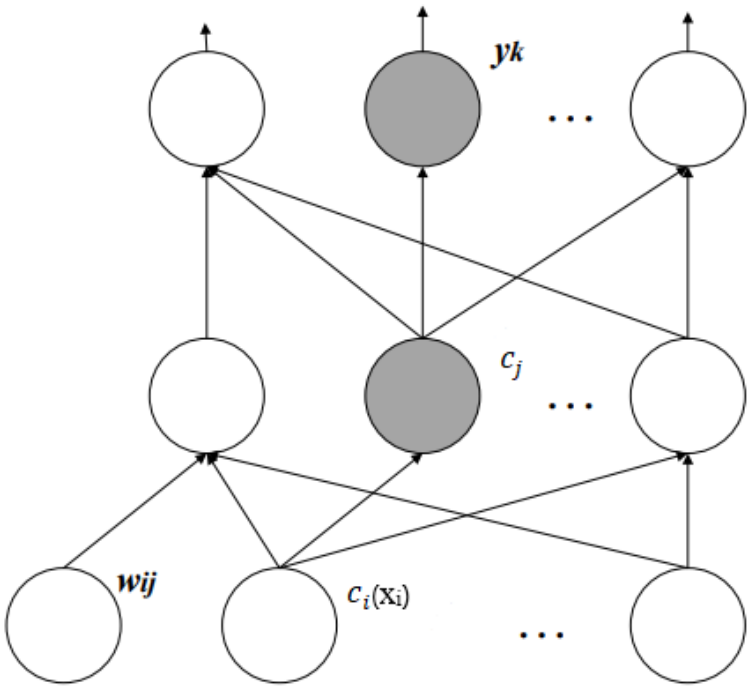

Рисунок 3.2 – Схема нейронної мережі алгоритму зворотного розповсюдження

E(w) є складною функцією, для якої w:

$$w = w(a_j) \tag{3.8}$$

Звідси за правилом обчислення похідної складної функції (3.9):

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}}, \frac{\partial a_j}{\partial w_{ji}} = \frac{\partial \Sigma w_{ji} c_i}{\partial w_{ji}} = c_i \tag{3.9}$$

Для обчислення градієнта на кожному шарі i обчислюється добуток величини помилки попереднього i, що знаходиться зверху шару j, на вхідне значення шару [15]:

$$\frac{\partial E}{\partial w_{ji}} = \delta_j c_i \tag{3.10}$$

Для вихідного шару δ_k можна отримати:

$$\delta_k = \frac{\partial E}{\partial a_k} = \left| \begin{matrix} a_k = \sum_j w_{jk} c_j \\ c_k = g(a_k) \end{matrix} \right| = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{a_k} \tag{3.11}$$

Для логічного сигмоїда (3.12):

$$g(a) = \frac{1}{1-e^{-a}} \tag{3.12}$$

така похідна матиме досить простий вигляд:

$$g'(a) = g(a)\big(1 - g(a)\big) \tag{3.13}$$

Другий множник $\frac{\partial E}{\partial y_k}$ зводиться до виразу (3.14):

$$\frac{\partial E}{\partial y_k} = y_k - t_k \tag{3.14}$$

У результаті формула обчислення помилки на вихідному шарі має вигляд (3.15):

$$\delta_k = g'(y_k - t_k) \tag{3.15}$$

Проміжний шар δ_j обраховується аналогічним способом (3.16):

$$\delta_i = \frac{\partial E}{\partial a_j} = \left| \begin{matrix} a_k = \sum_j w_{jk} c_j \\ c_j = g(a_j) \end{matrix} \right| = \sum_k \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial c_j}\frac{\partial c_j}{\partial a_j} \tag{3.16}$$

Підсумовування відбувається за всіма k, до яких нейрон j посилає сигнал згідно з рисунком 3.3.

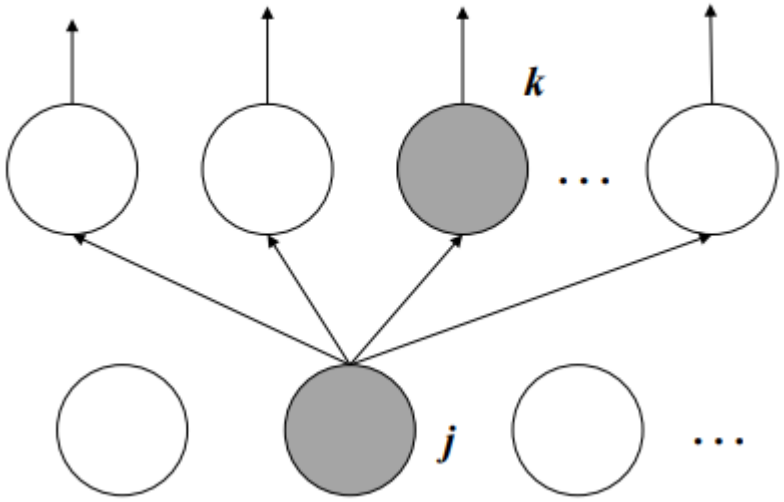

Рисунок 3.3 – Схема нейронної мережі алгоритму зворотного розповсюдження

В підсумку отримуємо наступну формулу (3.17):

$$\delta_i = g'(a_j)\sum_k \delta_k w_{jk} \tag{3.17}$$

Помилка на кожному шарі обчислюється рекурсивно через значення помилки на попередніх шарах [16].

Алгоритм зворотного розповсюдження помилки:

1) Перед початком роботи алгоритму присвоюємо вагам w_{ij} випадкові значення;
2) На вхід персептрону подаємо вектор x^n з навчальної вибірки. У результаті отримуємо значення y^n на виході за формулами (3.7) та (3.6);
3) Обчислюємо помилки δ_k для виходів мережі (3.10);
4) Обчислюємо помилки δ_j для всіх прихованих шарів (3.13);
5) Знаходимо значення градієнта (3.8);
6) Коригуємо значення синаптичних ваг за (3.5);
7) Обчислюємо значення помилки (3.4). Якщо величина помилки не влаштовує, то повторюємо алгоритм.

У загальному вигляді структурну схему алгоритму навчання мережі за методом зворотного розповсюдження помилки, який буде використаний далі у практичній реалізації наведена на рисунку 3.4:

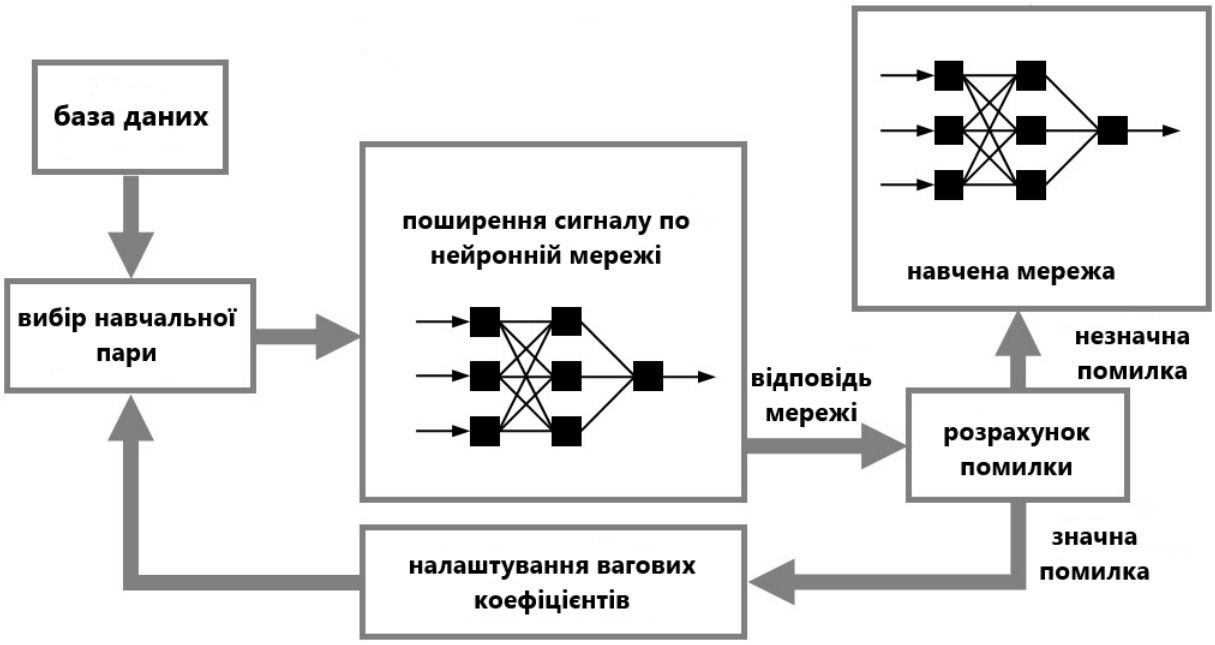

Рисунок 3.4 – Схема АЗРП

Для АЗРП є різні модифікації, що створені задля зменшення часу навчання мережі, але вони більш складні у використанні та споживають більше енергоресурсів системи [14].

3.7 Висновки по розділу №3

Отже, існує цілий список методик та алгоритмів для навчання нейронних мереж. Усі вони мають свої переваги та недоліки пов'язані, в першу чергу, з відсутністю здатності обробляти вхідні дані різних типів, а також з часом тренування мережі на основі навчальної вибірки. Для вирішення різного типу задач вибирається відповідний тип нейронної мережі та алгоритм її навчання, при цьому оптимальний варіант вибирається емпіричним шляхом, або на основі чийогось досвіду.

Нейронні мережі якісно будують згладжування чи добір значень функції за точками і можуть підібрати більшість функцій як однієї, так і кількох змінних.

Дослідження, проведені нейронними мережами, достатньо наочні і мають перевагу перед іншими обчислювальними методами. Це випливає з того, що теоретичні вимоги до моделей, отриманих нейронними мережами, дуже гнучкі,

також, їм потрібні відносно невеликі обсяги попередніх знань для формування завдання.

Нейронні мережі є потужним механізмом для вирішення задач у різних галузях. Їх необхідно використовувати для узагальнення, а не визначення даних. Нейронні мережі – адаптивні, вони наслідують рішення людини, але не повідомлять, який саме критерій вирішення завдання варто брати до уваги перед збиранням даних. Також навчальні машини використовуються для формалізації знань із реального світу, але самі машини не здатні самостійно генерувати принципи формалізації.

Виходячи з проведеного аналізу, у якості алгоритму навчання нейронної мережі я обрав АЗРП, тому що він істотно підвищує показники точності нейронної мережі та зменшує час навчання мережі завдяки подвійному проходу по вибірці за один крок.

4 ЗБІР ТА КЛАСИФІКАЦІЯ ПЕРЕХОПЛЕНОГО ТРАФІКУ

4.1 Принцип класифікації трафіку

Класифікація перехопленого мережевого трафіку потрібна для того, щоб на отриманій вибірці у майбутньому навчати систему виявляти нові та вже відомі атаки вторгнення. Як показують дослідження, комбінація декількох технологій ШІ призводять до більш ефективного результату [31]. Наприклад, навчання нейронної мережі на заздалегідь обробленій виборці даних проходить більш ефективно та швидко. Таким чином, постановку задачі класифікації трафіку можна сформулювати так [1]:

Є множина досліджуваних об'єктів - IP-пакетів прикладного рівня (4.1):

$$P = \{P_1, P_2, \dots, P_W\}, \tag{4.1}$$

де P_W – пакет з послідовності розмірністю W, який треба класифікувати.

Кожен об'єкт (IP-пакет) характеризується набором змінних (атрибутів) (4.2):

$$P_1 = \{X_1, X_2, \dots, X_N, M\}, \tag{4.2}$$

де X_N - спостережувані змінні, значення яких відомі;

N - кількість атрибутів пакета;

M - залежна множина, яку необхідно визначити (тип протоколу, внутрішній стан протоколу в процесі інформаційного обміну, тощо).

При цьому кожна змінна Xn приймає значення з деякої множини (4.3):

$$X_N = \{Xn_1, Xn_2, \dots, Xn_M\}, \tag{4.3}$$

де Xn_M - варіанти значень атрибутів.

Таким чином, завдання класифікації зводиться до визначення множини Y на основі значень атрибутів послідовності пакетів [1].

Вирішуючи прикладну задачу, визначимо найбільш відповідні атрибути пакета. У цій кваліфікаційній роботі було виділено наступний набір атрибутів [1]:

- X_1 – EtherType (тип стандарту протоколу Ethernet);
- X_2 – Source IP Address (IP-адреса відправника);
- X_3 – Destination IP Address (IP-адреса отримувача);

- X_4 – Multicast (приймає значення 1, якщо multicast, у іншому випадку – 0);
- X_5 – IP Protocol (тип транспортного рівня);
- X_6 – Packet Length (довжина мережевого пакета в байтах);
- X_7 – Source Port (порт TCP/UDP відправника);
- X_8 – Destination Port (порт TCP/UDP отримувача);
- X_9 – Hex_length (кількість байт у строковому контенті протоколу верхнього рівня – частина payload);
- X_{10} – Payload_hex (рядок, що передається в контенті протоколу верхнього рівня – частина payload);

4.2 Збір даних мережевого трафіку для подальшої класифікації

Моніторинг і збір пакетної статистичної інформації протоколів мережевого трафіку, що найчастіше зустрічаються в мережах (TLS v1, TLS v1.2, SSH v2, HTTP, FTP, Telnet і т. д.), був здійснений з використанням відкритого програмного забезпечення Wireshark і включав в себе вирішення наступних завдань [1]:

- вибір найбільш відповідних вхідних змінних для побудови моделі класифікації мережевих пакетів;

- формування набору первинних репрезентативних вибірок - DUMP у txt-форматі з пакетною інформацією відповідно із зазначеними вище протоколами.

Поділ класифікує мережеві пакети на приналежність до певного рівня загрози [1]:

- пакети, що несуть загрозу мережі;
- пакети, які можуть нести загрозу мережі. Вони повинні бути додатково перевірені адміністратором мережі або інженером з безпеки;
- безпечні звичайні протоколи.

В результаті обчислень за перерахованими вище правилами, пакети розподіляються на групи відповідно їх рівню загрози [1]:

- найвищий рівень загрози - 2;
- середній рівень загрози – 1;
- низький або відсутній рівень загрози – 0.

Детальніше про те, за якими критеріями буде винесено висновок щодо загрози пакета можна дізнатися у розділі №5 з практичною реалізацією системи виявлення та запобігання вторгнень [1].

Протоколи, які потрапили в одну групу, будемо вважати багато в чому схожими, а загальну вибірку в групі - однорідною. На сучасному етапі розвитку математичного моделювання прийнято вважати, що репрезентативні вхідні набори даних багато в чому забезпечують кінцевий успіх всього моделювання - отримання адекватних моделей [27]. Як правило, проводяться статистичні перевірки вихідних даних, виявляються і виключаються з тренувальних вибірок сильно зашумлені або надлишкові вхідні дані, виділяються гарні набори даних для подальшої побудови моделей класифікації мережевих пакетів прикладного рівня [1].

4.3 Дослідження мережевого аналізатору «Wireshark»

4.3.1 Застосування Sniffer-у пакетів «Wireshark»

Wireshark - це програмний інструмент, який використовується для моніторингу мережевого трафіку через мережевий інтерфейс. Це найбільш широко використовуваний інструмент моніторингу мережі сьогодні [29]. Дослідження функцій системи «Wireshark»

Мережевий аналізатор Wireshark має наступні функції [1]:

- захоплення даних пакетів з мережевого інтерфейсу в реальному часі;
- відкриття файлів, що містять пакетні дані, захоплені за допомогою tcpdump/WinDump, Wireshark і багатьох інших програм захоплення пакетів;
- імпорт пакетів з текстових файлів, що містять шістнадцяткові дампи пакетних даних;
- відображення пакетів з дуже докладною інформацією про протокол;
- збереження захоплених пакетних даних;
- експорт деяких або всіх пакетів в декількох форматах файлів захоплення;
- фільтрація пакетів за багатьма критеріями;
- пошук пакетів по багатьом критеріям;
- розфарбовування відображення пакетів на основі фільтрів;

- створення різної статистики;

Паралельно програма фіксує навантаження на мережу, зберігає статистику, відображає в реальному часі відправку та отримання пакетів, але це вже другорядні функції. Основним залишається перехоплення трафіку. Тут потрібно уточнення, що використовування програми має узгоджуватися з власником підмережі, інакше це цілком зійде за «хакерську атаку» з відповідними наслідками для фахівця [1].

4.3.3 Здійснення перехоплення пакетів

Sniffing пакетів - це процес перехоплення даних пакетів, переданих по комп'ютерній мережі [23]. Sniffing пакетів - це для комп'ютерних мереж те ж саме, що прослуховування проводів в телефонній мережі. Це робиться за рахунок використання аналізаторів пакетів, які можуть бути підключені до мережі і використовуватися для підслуховування мережевого трафіку. Використовуючи інформацію, отриману аналізатором пакетів, адміністратор може визначити помилкові пакети і використовувати дані для виявлення небезпечних хостів, допомагати підтримувати ефективну передачу даних по мережі [1].

Однак, він також широко використовується хакерами і зловмисниками для незаконного збору інформації про мережу для майбутнього вторгнення. Використання сніферу пакетів дає можливість захоплення даних, таких як паролі, IP-адреси, протоколи, використовувані в мережі, та іншої інформації, яка допоможе зловмиснику проникнути в мережу. Аналіз пакетів в основному використовується при вторгненні, виявленні вторгнень, управлінні мережею, прослуховуванні й зломі. Програми для сніфінгу паролів - популярні програми для перехоплення пакетів серед зловмисників [1].

Sniffer пактів у Wireshark: цей сегмент візуально показує пакети, що проходять всередині мережі. Є колірні коди для кожного типу пакетів. Пакети показані з наступною інформацією [1]:

- вихідна адреса;
- адреса призначення;
- тип пакету;
- шістнадцятковий дамп пакету;

- зміст пакету в тексті;
- порт джерела (якщо є);
- порт призначення (якщо є).

4.4 Висновки по розділу №4

Отже, розробивши логічні правила для класифікації мережевого трафіку для створення навчальної вибірки даних для нейронної мережі та за допомогою мережевого аналізатору «Wireshark» можна вирізнити мережеві пакети за 3 групами відповідно до їх рівня потенційної загрози мережі. Отримані групи пакетів мають схожі властивості та протоколи. Завдяки використанню пакетів за даною класифікацією можна прискорити етап навчання алгоритмів штучного інтелекту.

5 ДОСЛІДЖЕННЯ ПРАКТИЧНОГО ВИКОРИСТАННЯ ПОЄДНАННЯ ТЕХНОЛОГІЙ НЕЙРОННОЇ МЕРЕЖІ ТА НЕЧІТКОЇ ЛОГІКИ

Для практичного дослідження технологій штучного інтелекту у системах виявлення/запобігання вторгнень у мережі, у своїй кваліфікаційній роботі я обрав реалізацію технології нейронної мережі, яка у свою чергу повинна проходити етап навчання на тестовій вибірці, що буде створена за алгоритмами нечіткої логіки. Завдяки алгоритмам нечіткої логіки ми можемо класифікувати мережеві пакети як нормальні (рівень загрози = 0), загрозливі (рівень загрози = 2) та пакети, які треба відправити адміністратору або інженеру з безпеки для подальшої перевірки (рівень загрози = 1).

5.1 Збір мережевого трафіку

Для збору мережевого трафіку з подальшим формуванням тестової вибірки для нейронної мережі було створено локальну мережу з однієї фізичної та двох віртуальних машин (рис. 5.1). Топологія експериментальної мережі [1]:

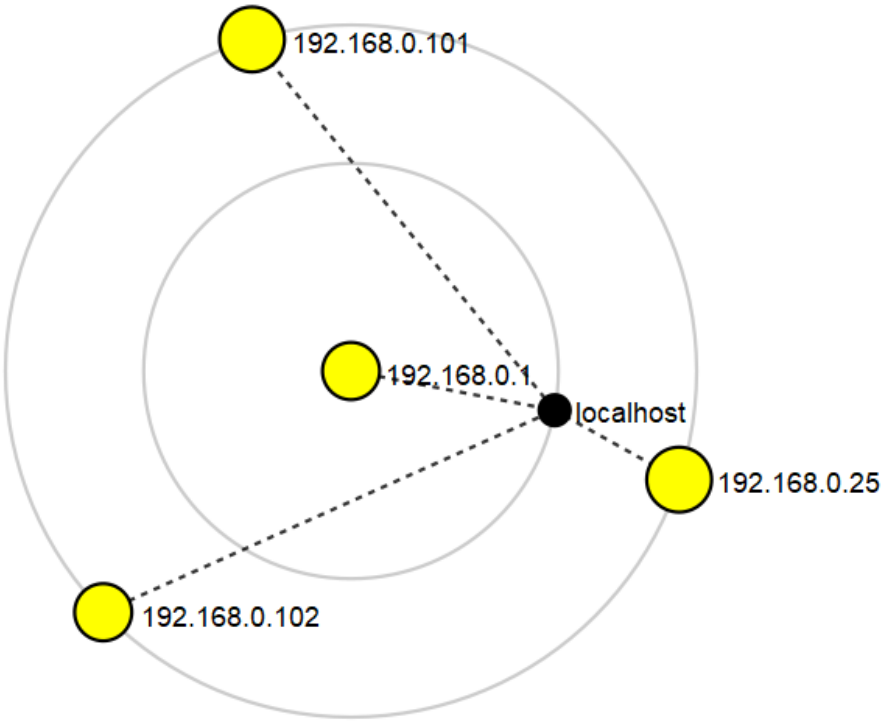

Рисунок 5.1 – Топологія дослідної локальної мережі

Перед тим як розпочати створювати мережевий трафік у локальній мережі, треба встановити перехоплення пакетів за допомогою сніферу пакетів Wireshark. Тип з'єднання мережі – бездротовий. У якості вузлу, на який буде направлено мережевий трафік було обрано віртуальну машину (ОС Linux-Bit), яка має адресу 192.168.0.25. Для того щоб перехопити мережевий трафік, який буде спрямований на цей вузол, треба встановити фільтр захвату (рис. 5.2) [18]:

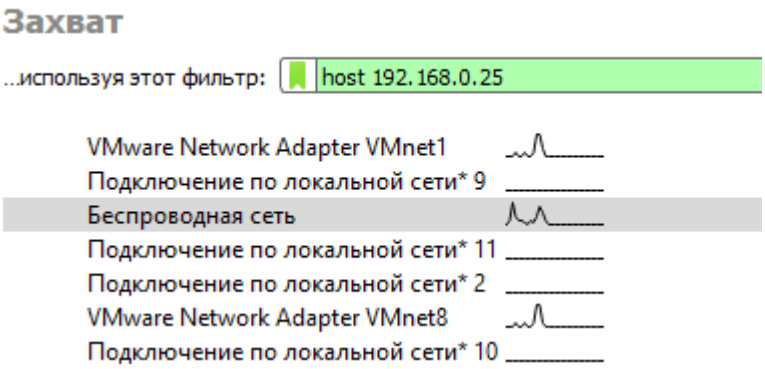

Рисунок 5.2 – Фільтр захвату за вузлом з адресою 192.168.0.25

Після початку формування трафіку переконуємося, що сніфер пакетів перехопив мережевий трафік (рис. 5.3) [1].

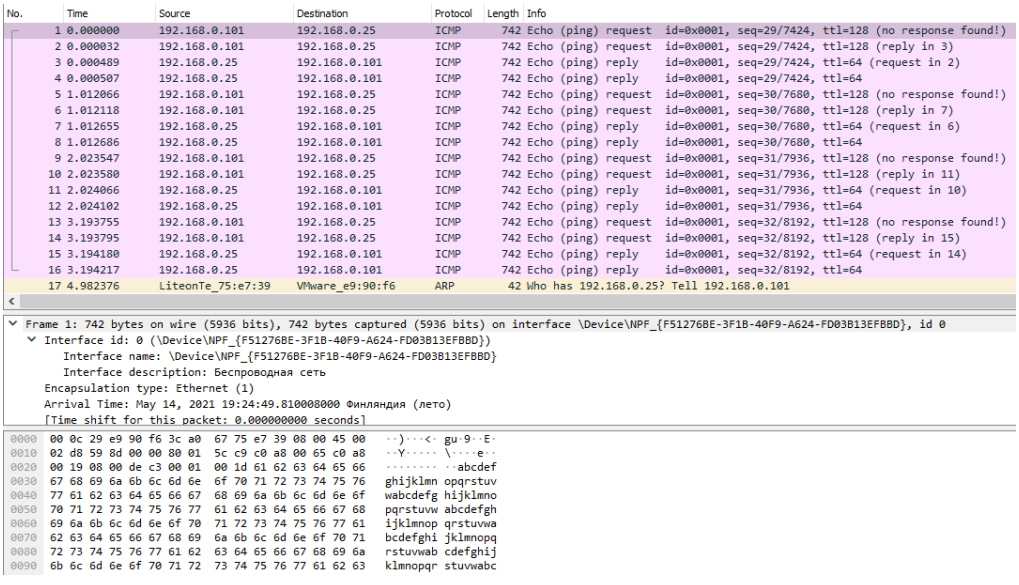

Рисунок 5.3 – Результат роботи сніферу пакетів Wireshark

У ході перехоплення трафіку було отримано 257 пакетів, які відповідають умовами задачі (загрозливі та безпечні). Отримані пакети у вигляді пакетної інформації було збережено як звичайний текст у файл packets.txt, який буде подано на вхід алгоритму класифікації нечіткої логіки [1].

5.2 Класифікація мережевих пакетів для навчальної вибірки за допомогою правил нечіткої логіки

Нечітка логіка заснована на наборі правил передбачених користувачем [11]. У своїй реалізації даної технології я впровадив наступні правила для своєї мережі:

а) класифікація пакетів, що містять у своєму запиті такі слова як: password, passwords, login, logins, credentials як пакетів, що потребують подальшої перевірки людиною;

б) класифікація пакетів, довжина яких ≥ 500 байт як загрозливих;

в) класифікація пакетів підключення через мережеві утиліти FTP, Telnet, SSH як загрозливих;

г) класифікація всіх інших пакетів як нормальних.

Таким чином, коли алгоритми нечіткої логіки оброблять усі пакети, зібрані у локальній мережі, будуть створені пари для навчальної вибірки нейронної мережі. У кожній парі буде міститися масив з трьох вхідних параметрів у вигляді чисел з плаваючою точкою (тип даних double) та одним числом, яке буде означати рівень загрози пакету (0, 1 або 2).

Масив вхідних параметрів формується за наступними правилами:

- перший параметр = 1, якщо пакет містить слова з правила класифікації а, у протилежному випадку - 0;
- другий параметр = 1, якщо пакет є пакетом підключення через мережеві утиліти FTP, Telnet або SSH (правило класифікації в), якщо ні, то = 0;
- третій параметр задається довжиною пакету у кілобайтах (правило класифікації б). У кілобайтах, а не байтах, тому що не вдасться знайти експоненту у ступені, наприклад, 700. Для переводу байт в кілобайт треба помножити кількість байт на ≈ 0.0009765625.

У якості мови програмування для реалізації обраної технології мною було обрано Java та Intellij Idea у якості середовища розробки.

Для початку треба створити проект Maven для того, щоб мати можливість додавати сторонні бібліотеки, які знадобляться у процесі проектування та реалізації алгоритму нейронної мережі (рис. 5.4):

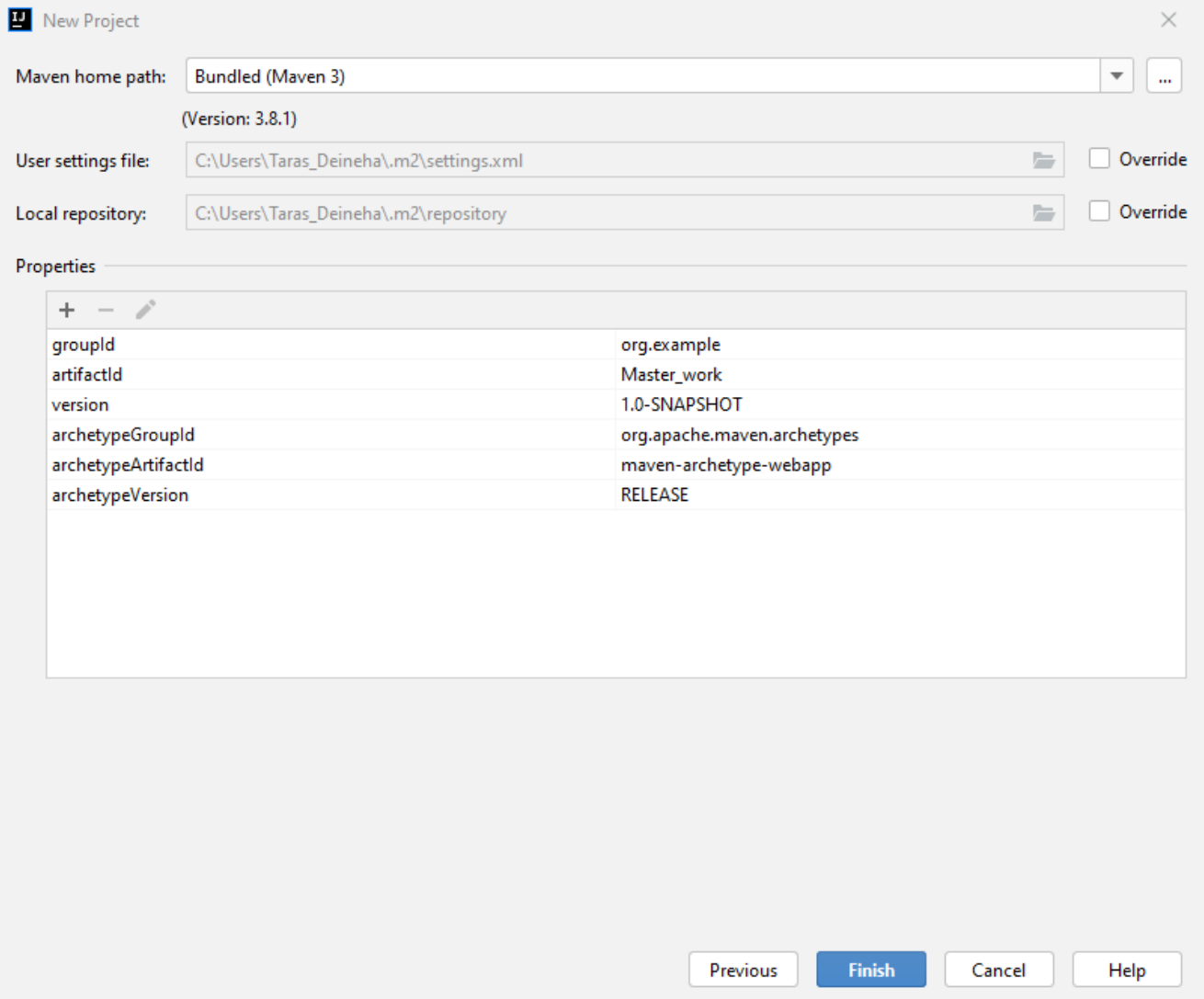

Рисунок 5.4 – Створення Maven проекту

На рисунку 5.5 можна побачити реалізацію створення та додавання пар у навчальну вибірку для нейронної мережі із застосуванням алгоритмів нечіткої логіки. Алгоритм створення навчальної вибірки:

1) запуск циклу по масиву перехоплених мережевих пакетів;
2) відокремлення одного пакету та створення пустої пари;
3) перевірка пакета на відповідність до одного чи декількох заданих у мережі правил;
4) якщо відповідності знайдено, то заповнюємо масив вхідних параметрів пари одиницями та 0.5, якщо не знайдено – нулями;
5) додаємо заповнену пару в датасет;
6) переходимо до наступного пакету та повторюємо пункти 1 – 5 з ним, доки масив перехоплених пакетів не буде весь оброблений.

```
private void addPairsToTDataset(final ArrayList<String> packets,
                                final List<Pair<List<Double>, Integer>> dataset) {
    final Pattern pattern = Pattern.compile(PACKET_LENGTH_REGEX);

    for (Iterator<String> iterator = packets.iterator(); iterator.hasNext(); ) {
        final String packet = iterator.next();
        final Matcher matcher = pattern.matcher(packet);
        final List<Double> inputParameters = new ArrayList<>(Arrays.asList(0.0, 0.0, 0.0));
        int outputValue = 0;

        if (packet.contains(PASSWORD) || packet.contains(LOGIN) || packet.contains(CREDENTIALS)) {
            logger.logging( str: FORBIDDEN_WORDS_DETECTED + packet);
            inputParameters.set(0, 1.0);
            outputValue = 1;
        } else if (packet.contains(TCP_PROTOCOL) && packet.contains(SOURCE_AND_DESTINATION_IPS)
                && packet.contains(PORT23) && !packet.contains(TELNET)
                && !packet.contains(DUP) && !packet.contains(OUT_OFF)) {
            logger.logging( str: TELNET_REQUEST_DETECTED + packet);
            inputParameters.set(1, 1.0);
            outputValue = 2;
        } else if (packet.contains(TCP_PROTOCOL) && packet.contains(SOURCE_AND_DESTINATION_IPS) && !packet.contains(SSH)
                && packet.contains(PORT22) && !packet.contains(DUP) && !packet.contains(OUT_OFF)) {
            logger.logging( str: SSH_REQUEST_DETECTED + packet);
            inputParameters.set(1, 1.0);
            outputValue = 2;
        } else if (packet.contains(TCP_PROTOCOL) && packet.contains(SOURCE_AND_DESTINATION_IPS) && !packet.contains(FTP)
                && packet.contains(PORT21) && !packet.contains(DUP) && !packet.contains(OUT_OFF)) {
            logger.logging( str: FTP_REQUEST_DETECTED + System.lineSeparator() + packet);
            inputParameters.set(1, 1.0);
            outputValue = 2;
        }
        while (matcher.find()) {
            final double data = Integer.parseInt(matcher.group());
            if (data > 500) {
                logger.logging( str: LARGE_LENGTH_DETECTED + packet);
                outputValue = 2;
            }
            if (outputValue == 0) {
                dataset.add(Pair.of(List.of(0.0, 0.0, data * BYTE_TO_KILOBYTE), outputValue));
            } else {
                dataset.add(Pair.of(List.of(inputParameters.get(0), inputParameters.get(1),
                        data * BYTE_TO_KILOBYTE), outputValue));
            }
        }
        iterator.remove();
    }
}
```

Рисунок 5.5 – Реалізація створення пар та їх подальше додавання до навчальної вибірки нейронної мережі за алгоритмами нечіткої логіки

Логіка даного алгоритму полягає у поетапній перевірці всіх зібраних пакетів мережевого трафіку за встановленими правилами; створення пар навчальної вибірки на основі цього аналізу.

logger. logging… рядки у коді на рисунку 5.5 додають інформацію про пакети з рівнем небезпеки 1 та 2 до журналу аудиту (файл logs.txt). Сам клас Logger виглядає наступним чином (рис. 5.6):

```java
package fuzzy_logic;

import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class Logger {
    private static final StringBuilder logMessages = new StringBuilder();

    public void logging(String str) {
        Scanner in = new Scanner(str);
        String s = in.nextLine() + System.lineSeparator() + in.nextLine() + System.lineSeparator() + in.nextLine();
        logMessages.append(s).append(System.lineSeparator()).append(System.lineSeparator());
        in.close();
    }

    public void writeLogs() throws IOException {
        FileWriter fw1 = new FileWriter( fileName: "logs.txt");
        fw1.write(logMessages.toString());
        fw1.flush();
    }
}
```

Рисунок 5.6 – Клас ведення журналу аудиту Logger

Повідомлення у даному класі записуються у файл logs.txt.

5.3 Установка гіпер-параметрів нейронної мережі та обчислення її шарів

Для синтезу алгоритму нейронної мережі знадобляться наступні чотири пакети для роботи з лінійною алгеброю та матричними функціями [11]:

- org.jblas.Doublematrix – пакет, котрий містить необхідні класи для роботи з матрицями та лінійною алгеброю;
- org.jblas.matrixFunctions – пакет, котрий містить необхідні класи для роботи з матричними функціями та їх перетвореннями;
- org.apache.commons – пакет, котрий містить необхідні класи для роботи з обробкою даних;
- org.apache.junit – пакет, котрий містить необхідні класи для тестування функціоналу алгоритму .

Для того, щоб ними скористатися треба додати відповідні посилання на них до тегу dependencies у файлі pom.xml (рис. 5.7):

pom.xml (Master_work)

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
  <!-- https://mvnrepository.com/artifact/org.jblas/jblas -->
  <dependency>
    <groupId>org.jblas</groupId>
    <artifactId>jblas</artifactId>
    <version>1.2.4</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/org.apache.commons/commons-lang3 -->
  <dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
    <version>3.12.0</version>
  </dependency>
  <dependency>
    <groupId>com.github.sh0nk</groupId>
    <artifactId>matplotlib4j</artifactId>
    <version>0.5.0</version>
  </dependency>
</dependencies>
```

Рисунок 5.7 – Зміст тегу dependencies у файлі pom.xml

Після того як усі підготовчі налаштування були виконані можна переходити до реалізації алгоритму навчання нейронної мережі. Спочатку треба задати гіпер-параметри для нейронної мережі:

- розмір вхідного вектору (кількість вхідних даних). У нашому випадку розмір дорівнює трьом;
- розмір вихідного вектору (кількість можливих висновків щодо безпечності пакету). Їх також три: безпечний (0); пакет, що потребує додаткової перевірки людиною (1), загрозливий (2);
- кількість нейронів у першому шарі. Їх буде 10. Дане число було отримано шляхом практичного підбору та аналізу відсотка правильних класифікацій пакетів.

Схематично отриману нейронну мережу можна зобразити наступним чином (рис. 5.8):

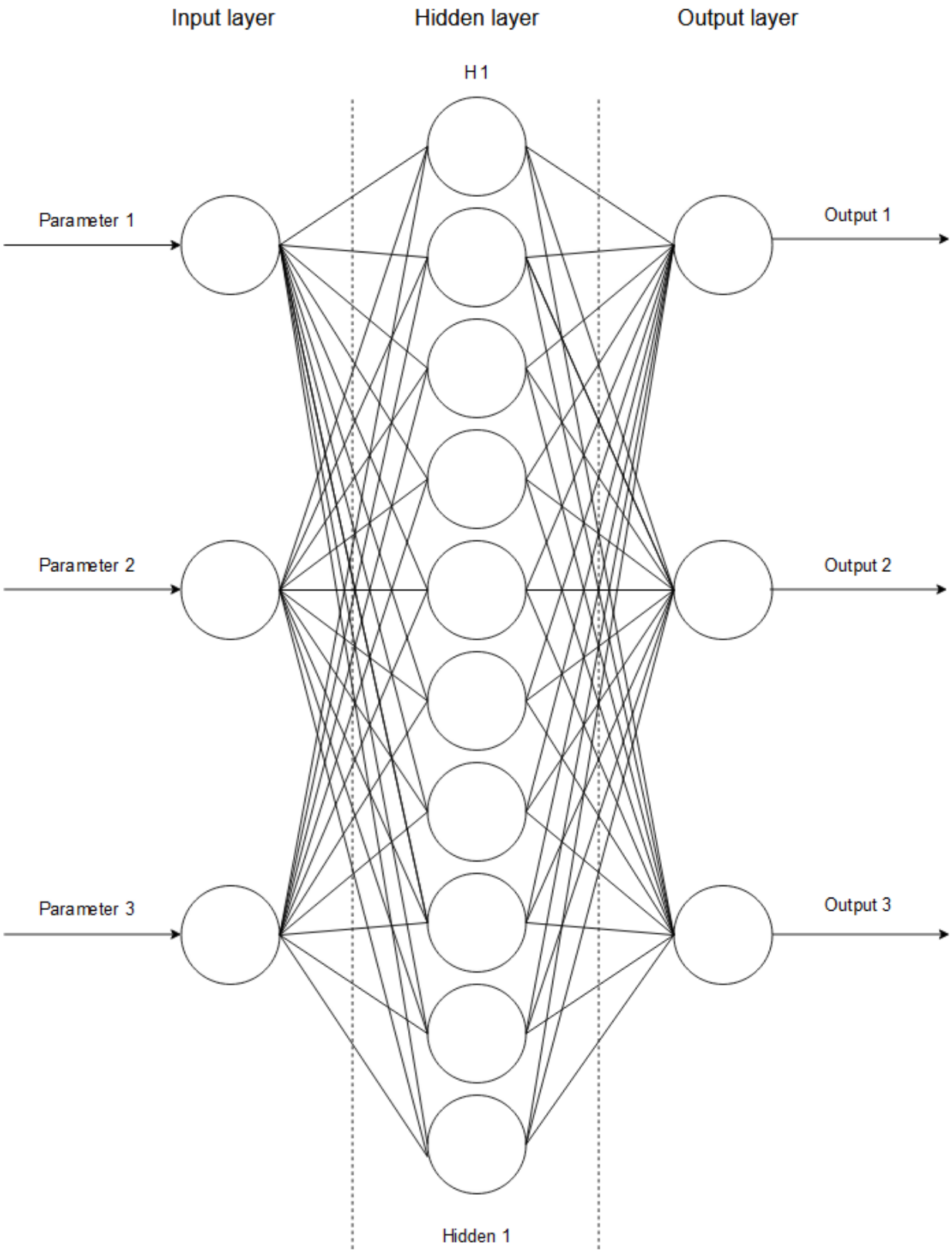

Рисунок 5.8 – Схема шарів нейронної мережі

У коді ініціалізація гіпер-параметрів виглядає наступним чином (рис. 5.9):

```
private final static int numberOfInputParameters = 3;
private final static int numberOfOutputs = 3;
private final static int numberOfNeurons = 10;
```

Рисунок 5.9 – Ініціалізація гіпер-параметрів

Перший шар (шар вхідних значень) назвемо x. x є вхідним вектором, тому лінійна частина першого повно-зв'язного шару t_1 (5.1) та нелінійна частина першого повно-зв'язного шару h_1 (5.2) можна обчислити як [11]:

$$t_1 = xW_1 + b_1 \tag{5.1}$$

$$h_1 = F(t_1) \tag{5.2}$$

Другий шар (вихід шару t1 на вихідний шар) t2 можна обчислити як (5.3):

$$t_2 = h_1W_2 + b_2 \tag{5.3}$$

Звичайну функцію активації застосовувати не треба, так як це останній шар нейронної мережі.

Вектор ймовірності G обчислюється за наступною формулою (5.4):

$$G = Soft\,max(t_2) = S(t_2) \tag{5.4}$$

Softmax() – це функція, що перетворює довільний вектор у набір ймовірностей. Вихідний вектор з цієї функції, а саме його i-ий елемент, розраховується за формулою (5.5):

$$S(t) = \left\{ \frac{e^{t_i}}{\sum_j e^{t_j}} \right\} \tag{5.5}$$

У чисельнику береться експонента від усіх елементів вектору, а у знаменнику сума експоненти всіх елементів, для того, щоб сума фінальних ймовірностей дорівнювала одиниці.

У якості нелінійної функції активації F(t) була обрана функція активації RELU для кожного елемента вектору t. Ця функція представляє собою поелементний максимум від 0 до t (5.6):

$$F(t) = RELU(t) = max(0, t) \tag{5.6}$$

Суть цієї функції полягає в тому, щоб обнулити все те, що менше нуля та залишити так як є все, що більше нуля.

Спочатку необхідно створити вхідний вектор-строку x. У моєму випадку я заповню його трьома значеннями (рис. 5.10), які я отримав із заповнених пар тестової вибірки, створених за допомогою правил нечіткої логіки. Також треба задати для змінної y висновок щодо рівню безпеки пакету (рис. 5.10). Його ми також беремо з пари:

```
for (Pair<List<Double>, Integer> listIntegerPair : dataset) {
    final List<Double> inputVector = listIntegerPair.getLeft();
    double[] inputParameters = new double[]{inputVector.get(0), inputVector.get(1), inputVector.get(2)};
    final DoubleMatrix x = new DoubleMatrix( newRows: 1,  newColumns: 3, inputParameters);
    final int y = listIntegerPair.getRight();
    System.out.println(x + System.lineSeparator() + y);
```

Рисунок 5.10 – Створення вхідного вектору x та правильної відповіді y

Далі потрібно створити 2 матриці ваг W_1, W_2 та два вектори зміщення b_1, b_2 (рис. 5.11). Так як ми зараз на етапі навчання мережі, то ми заповнимо їх випадковими значеннями:

```
private static DoubleMatrix W1 = DoubleMatrix.rand(numberOfInputParameters, numberOfNeurons);

private static DoubleMatrix b1 = DoubleMatrix.rand( rows: 1, numberOfNeurons);

private static DoubleMatrix W2 = DoubleMatrix.rand(numberOfNeurons, numberOfOutputs);

private static DoubleMatrix b2 = DoubleMatrix.rand( rows: 1, numberOfOutputs);
```

Рисунок – 5.11 - створення матриць ваг W_1, W_2 та векторів зміщення b_1, b_2

Розміри матриць ваг W_1, W_2 та векторів зміщення b_1, b_2 відповідні до кількості вхідних та вихідних параметрів, а також кількості нейронів на першому шарі нейронів.

5.4 Розробка прямого розповсюдження (forward propagation)

Після того як усі вхідні дані та гіпер-параметри були задані та ініціалізовані можна приступити до написання коду обчислення нейронної мережі - прямого розповсюдження або inference-у (рис. 5.12) [11]. Це алгоритм від входу вектору x до помилки E.

```
final DoubleMatrix t1 = x.mmul(W1).add(b1);
final DoubleMatrix h1 = RELU(t1);
final DoubleMatrix t2 = h1.mmul(W2).add(b2);
final DoubleMatrix result = softMax(t2);
```

Рисунок 5.12 – Код прямого розповсюдження (forward propagation)

Функція активації RELU та функція Softmax у коді виглядають наступним чином (рис. 5.13):

```
private static DoubleMatrix RELU(final DoubleMatrix t) {
    return t.maxi( v: 0, t);
}

private static DoubleMatrix softMax(final DoubleMatrix t) {
    final DoubleMatrix out = MatrixFunctions.exp(t);
    return out.divi(out.sum());
}
```

Рисунок 5.13 – Функції RELU та Softmax

Далі треба обчислити помилку E (рис. 5.14). Помилку можна представити у вигляді розрідженої крос-ентропії від вектору ймовірностей G та правильної відповіді y:

```
private static double sparseCrossEntropy(final DoubleMatrix result, final int y) {
    return -Math.log(result.get( rowIndex: 0, y));
}
```

Рисунок 5.14 – Розрахунок помилки за допомогою розрідженої крос-ентропії

У цій функції потрібно повернути суму усіх логарифмів елементів, але так як у нас у матриці лише одна одиниця за індексом y, то повертаємо мінус логарифм від даного елемента.

5.5 Розробка зворотного розповсюдження (backward propagation)

Після проходження forward частини алгоритму треба написати backward частину. Ця частина відповідає за дії алгоритму від помилки E до оновлення та налаштування змінних. Спочатку треба отримати повний вектор правильної відповіді y_full. Обчислюється він за функцією toFull (рис. 5.15), яка приймає у якості аргументів правильну відповідь y та масив з набором правильних відповідей (всього їх 3) [11]. На виході отримуємо вектор розповсюдження помилки відповідно до індексу правильної відповіді – вектор з нулів та одиниці.

```
private static DoubleMatrix toFull(final int y, final List<String> classification) {
    final DoubleMatrix yFull = DoubleMatrix.zeros( rows: 1, classification.size());
    yFull.put( rowIndex: 0, y,  value: 1);
    return yFull;
}
```

Рисунок 5.15 – Функція toFull

Далі треба виконати необхідні операції алгоритму для пошуку градієнтів (рис. 5.16). У загальному вигляді градієнт обчислюється за наступною формулою (5.7):

$$\frac{\partial E}{\partial W} \tag{5.7}$$

У коді обчислення усіх необхідних градієнтів виглядає наступним чином:

```
final DoubleMatrix yFull = toFull(y, classification);
final DoubleMatrix dE_dt2 = result.sub(yFull);
final DoubleMatrix dE_dW2 = h1.transpose().mmul(dE_dt2);
final DoubleMatrix dE_db2 = dE_dt2;
final DoubleMatrix dE_dh1 = dE_dt2.mmul(W2.transpose());
final DoubleMatrix dE_dt1 = dE_dh1.mmul(relu_deriv(t1));
final DoubleMatrix dE_dW1 = x.transpose().mmul(dE_dt1);
final DoubleMatrix dE_db1 = dE_dt1;
```

Рисунок 5.16 – Обчислення градієнтів

Похідна функція активації relu_deriv має наступний вигляд (рис. 5.17):

```
private static float relu_deriv(final DoubleMatrix vector) {
    return vector.sum() >= 0 ? 1 : 0;
}
```

Рисунок 5.17 – Похідна функція активації relu_deriv

Її суть полягає у тому, що ми порівнюємо суму елементів вектору з нулем. Якщо сума більше або рівна нулю, то повернеться 1, якщо менше – 0.

5.6 Оновлення параметрів (update)

Після того, як було знайдено необхідні градієнти можна перейти до написання шагу градієнтного спуску. Для цього потрібно задати ще один гіпер-параметр – швидкість навчання. Назвемо його LEARNING_SPEED та визначимо

його значення як 0.001. Це значення також було отримано у ході практичного підбору та аналізу відсотка правильних класифікацій пакетів.

Далі треба написати частину алгоритму (рис. 5.18), відповідальну за оновлення матриць ваг та векторів зміщення з урахуванням знайдених градієнтів:

```
W1 = W1.sub(dE_dW1.mul(LEARNING_SPEED));
b1 = b1.sub(dE_db1.mul(LEARNING_SPEED));
W2 = W2.sub(dE_dW2.mul(LEARNING_SPEED));
b2 = b2.sub(dE_db2.mul(LEARNING_SPEED));
```

Рисунок 5.18 – Оновлення матриць ваг та векторів зміщення

Коли ми повністю описали всі частини алгоритму: forward, backward, update - треба обернути цей код у подвійний цикл для того, щоб етап навчання нейронної мережі пройшов декілька разів по всій навчальній вибірці для кращого кінцевого результату [15]. Фінальний алгоритм навчання виглядає наступним чином (рис. 5.19):

```
for (int j = 0; j < 1000; j++) {
    for (Pair<List<Double>, Integer> listIntegerPair : dataset) {
        final List<Double> inputVector = listIntegerPair.getLeft();
        double[] inputParameters = new double[]{inputVector.get(0), inputVector.get(1), inputVector.get(2)};
        final DoubleMatrix x = new DoubleMatrix( newRows: 1,  newColumns: 3, inputParameters);
        final int y = listIntegerPair.getRight();
        System.out.println(x + System.lineSeparator() + y);

        final DoubleMatrix t1 = x.mmul(W1).add(b1);
        final DoubleMatrix h1 = RELU(t1);
        final DoubleMatrix t2 = h1.mmul(W2).add(b2);
        final DoubleMatrix result = softMax(t2);

        final double error = sparseCrossEntropy(result, y);

        final DoubleMatrix yFull = toFull(y, classification);
        final DoubleMatrix dE_dt2 = result.sub(yFull);
        final DoubleMatrix dE_dW2 = h1.transpose().mmul(dE_dt2);
        final DoubleMatrix dE_db2 = dE_dt2;
        final DoubleMatrix dE_dh1 = dE_dt2.mmul(W2.transpose());
        final DoubleMatrix dE_dt1 = dE_dh1.mmul(relu_deriv(t1));
        final DoubleMatrix dE_dW1 = x.transpose().mmul(dE_dt1);
        final DoubleMatrix dE_db1 = dE_dt1;

        W1 = W1.sub(dE_dW1.mul(LEARNING_SPEED));
        b1 = b1.sub(dE_db1.mul(LEARNING_SPEED));
        W2 = W2.sub(dE_dW2.mul(LEARNING_SPEED));
        b2 = b2.sub(dE_db2.mul(LEARNING_SPEED));
        errors.add(error);
    }
}
```

Рисунок 5.19 – Кінцевий вигляд алгоритму навчання нейронної мережі

Число 1000 в якості кількості ітерацій для проходження навчання нейронної мережі було обрано шляхом практичного підбору та аналізу відсотка правильних класифікацій пакетів.

5.7 Установлення метрик та допоміжних утиліт

У останньому рядку алгоритму (рис. 5.19) відбувається додавання помилки у список помилок. Даний список буде служити певною метрикою залежності помилки від ітерації. Це потрібно для того, щоб дивитись на роботу алгоритму у ході його виконання [19].

Також до важливих метрик ефективності навчання нейронної мережі можна віднести відсоток правильної відповіді. Для її обрахунку скопіюємо прямий прохід (forward) алгоритму у окрему функцію predict. Дану функцію слід використати у ще одній функції, яка саме і буде визначати відсоток правильних відповідей – calc_accuracy. Дані функції мають наступний вигляд (рис. 5.20):

```
private static double calc_accuracy() {
    int correct = 0;
    for (Pair<List<Double>, Integer> listIntegerPair : dataset) {
        final List<Double> inputVector = listIntegerPair.getLeft();
        double[] inputParameters = new double[]{inputVector.get(0), inputVector.get(1), inputVector.get(2)};
        final DoubleMatrix result = predict(new DoubleMatrix( newRows: 1,  newColumns: 3, inputParameters));
        int y_pred = result.argmax();
        if (y_pred == listIntegerPair.getRight()) {
            correct++;
        }
    }
    return Double.parseDouble(String.valueOf(correct)) / dataset.size();
}

private static DoubleMatrix predict(final DoubleMatrix x) {
    final DoubleMatrix t1 = x.mmul(W1).add(b1);
    final DoubleMatrix h1 = RELU(t1);
    final DoubleMatrix t2 = h1.mmul(W2).add(b2);
    return softMax(t2);
}
```

Рисунок 5.20 – Розрахунок відсотка правильних відповідей

Після написання функції calc_accuracy виведемо її значення після останньої ітерації алгоритму у консоль, а також виведемо графік залежності помилки від ітерації (рис. 5.21):

```
System.out.println("Accuracy: " + calc_accuracy());
final Plot plot = Plot.create();
plot.plot().add(errors);
plot.title( s: "Errors");
plot.show();
```

Рисунок 5.21 – Вивід до консолі відсотка правильних відповідей та побудова графіка залежності помилки від ітерації

Для побудови графіку знадобиться додати ще дві бібліотеки з мови програмування Python – com.hithub.sh0nk.matplotlib4j.Plot та com.hithub.shonk.matplotlib4j.PythonExecutionException.

Також варто ще приділити увагу створенню списку вихідних станів (рівня безпечності пакета) у нейронній мережі (рис. 5.22):

```
final List<Double> errors = new ArrayList<>();
final FuzzyLogic fuzzyLogic = new FuzzyLogic();

classification.add(SECURE);
classification.add(NEED_CLARIFY);
classification.add(NOT_SECURE);
dataset = fuzzyLogic.execute();
```

Рисунок 5.22 – Створення списку вихідних станів classification

Як можна бачити з рисунка 5.22 у списку вихідних станів додається три змінні: Secure, Not secure, Need Clarify. Ієрархія проекту виглядає наступним чином (рис. 5.23):

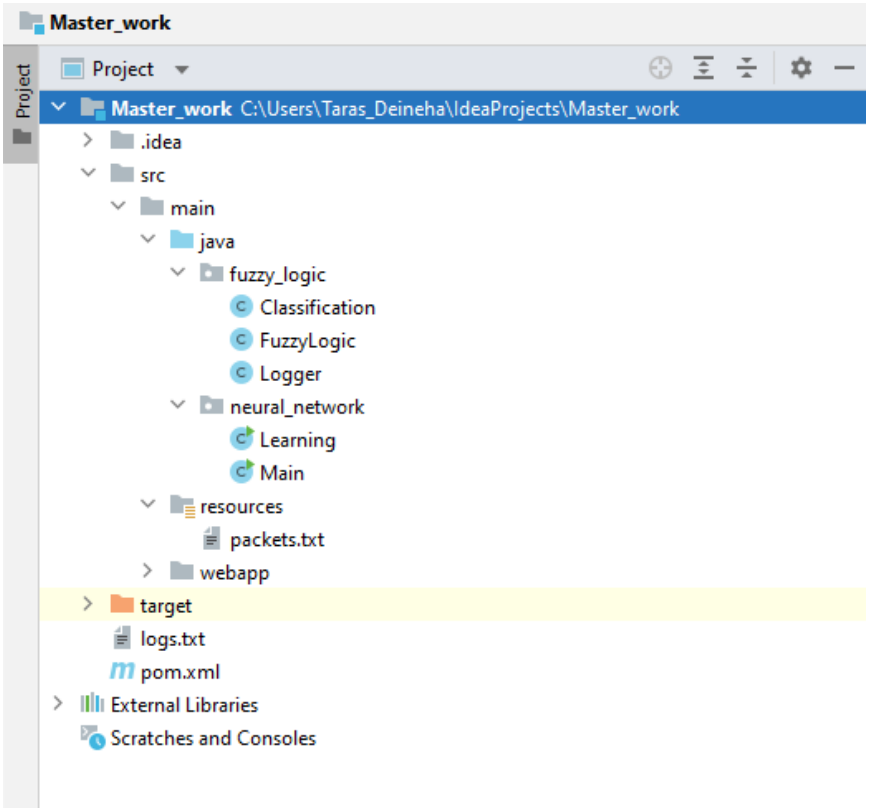

Рисунок 5.23 – Ієрархія проекту реалізації нейронної мережі

Усі класи програми поділені на 2 пакети: пакет з нечіткою логікою для створення навчальної вибірки та пакет з нейронної мережею.

5.8 Запуск алгоритму навчання нейронної мережі

Коли програма повністю написана – можна приступити до запуску етапу навчання нейронної мережі. Для цього треба запустити метод main() у класі Learning (рис. 5.24):

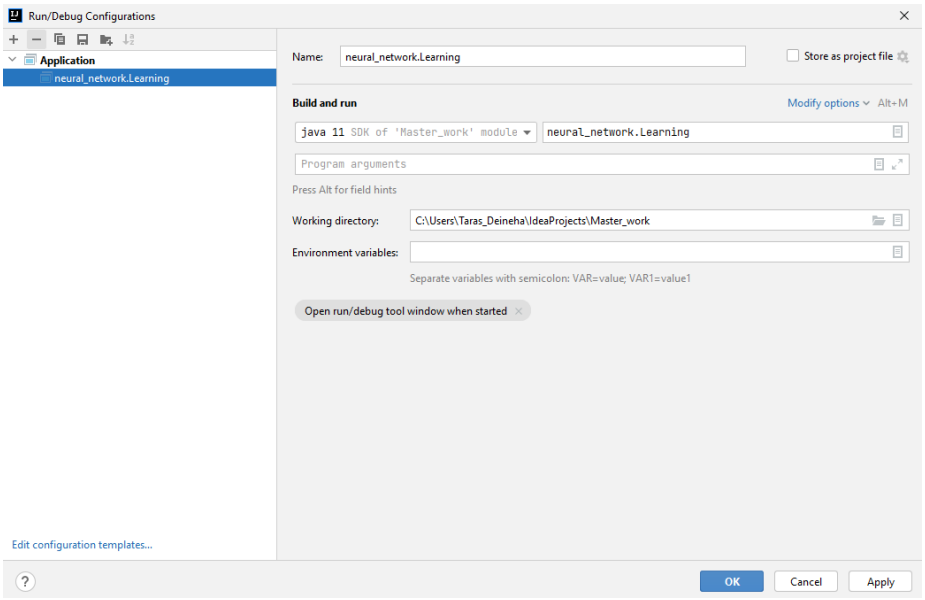

Рисунок 5.24 – Вікно запуску алгоритму навчання нейронної мережі

Як можна бачити з рисунку 5.25, програма почала ітеруватися по навчальній вибірці [14]. У консоль виводяться вхідні параметри пакета у першому рядку та рівень його безпеки у другому (рис. 5.25).

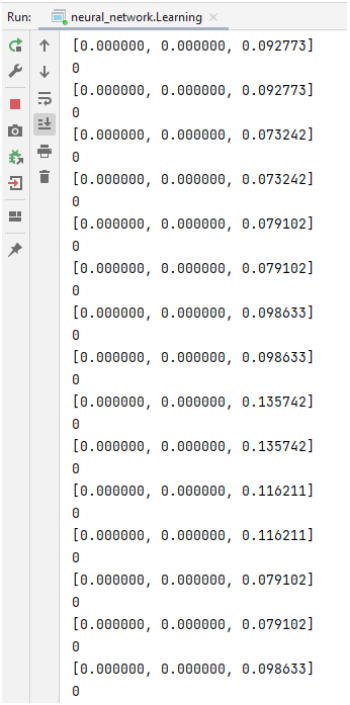

Рисунок 5.25 – Вивід ітерацій алгоритму в консоль

Після останньої ітерації у консоль виводиться відсоток правильних класифікацій пакета (рис. 5.26). У моєму випадку це 98.83%, що є дуже гарним результатом. Такий високий відсоток був отриманий не з першої спроби. Було проведено багато експериментів з різними гіпер-параметрами, кількістю раундів циклу алгоритму, кількістю нейронних шарів та нейронів у одному шарі, тощо.

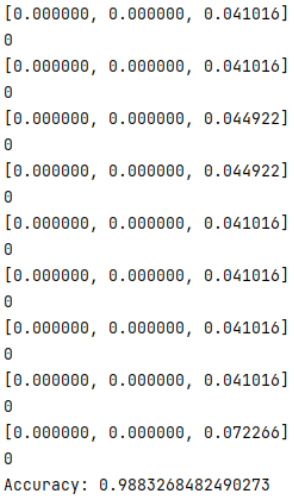

Рисунок 5.26 – Вивід в консоль відсотку правильних відповідей нейронної мережі на навчальній вибірці

Також програма будує графік залежності кількості помилок від ітерації алгоритму (рис. 5.27):

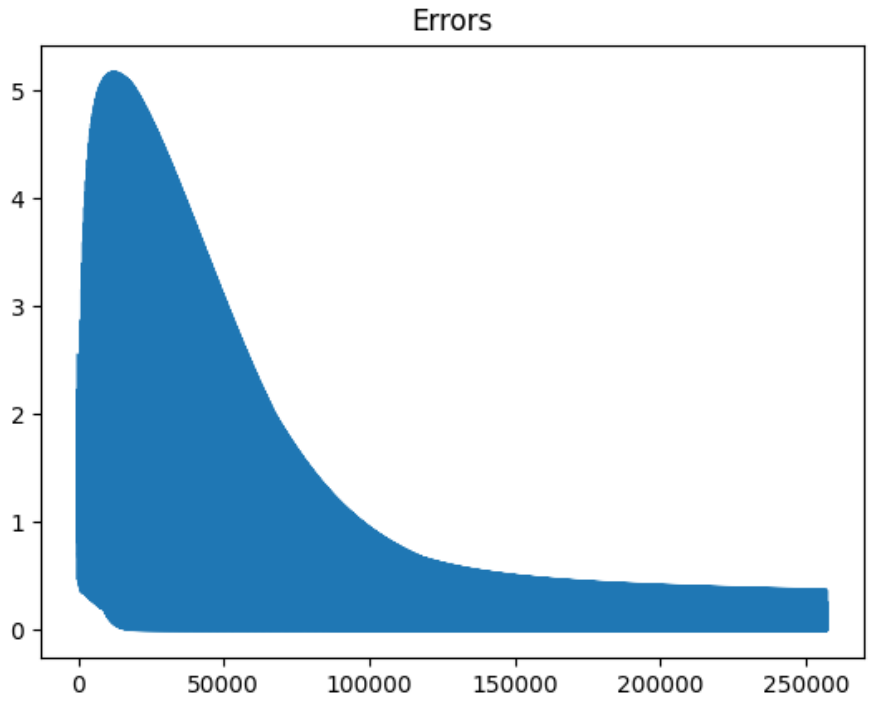

Рисунок 5.27 – Графік залежності кількості помилок від ітерації алгоритму

По даному графіку можна бачити, що кількість помилок знижується зі збільшенням кількості ітерацій. Стрімке зниження кількості помилок спостерігається приблизно після 20000 ітерацій.

Коли етап навчання пройдений та його результати задовольняють умови нашої задачі, додаймо вивід до консолі матриць ваг W1 та W2 та призупинимо побудову графіку на одну хвилину, щоб встигнути скопіювати ці матриці (рис. 5.28):

```
System.out.println(W1 + System.lineSeparator() + W2);
System.out.println("Accuracy: " + calc_accuracy());
System.out.println("W1 " + W1);
System.out.println("W2 " + W2);
Thread.sleep( millis: 60000);
final Plot plot = Plot.create();
plot.plot().add(errors);
plot.title( s: "Errors");
plot.show();
```

Рисунок 5.28 – Вивід до консолі матриць ваг W1 та W2

Тепер ми отримали матриці ваг W1 та W2 у консолі (рис. 5.29):

```
W1 [0.203667, 0.248497, 0.773345, 0.498236, 0.443255, 0.573605, 0.648849, 0.480637, 0
W2 [0.134190, 0.045500, 0.518272; -0.560773, 0.164089, 1.479789; -0.082983, 0.237033,
```

Рисунок 5.29 - Матриці ваг W1 та W2 у консолі

Повна матриця W1 = {0.203667, 0.248497, 0.773345, 0.498236, 0.443255, 0.573605, 0.648849, 0.480637, 0.505236, 0.937047; 0.325679, 1.103449, 1.077315, 0.868822, 1.431064, 0.029702, 0.481114, 0.850392, 0.384289, 0.106477; 0.787690, 1.235168, 1.184986, 0.501732, 0.559060, -0.416584, 0.015913, 0.457481, 1.297701, 0.630102}.

Повна матриця W2 = {0.134190, 0.045500, 0.518272; -0.560773, 0.164089, 1.479789; -0.082983, 0.237033, 1.255994; -0.009866, 0.316889, 1.048154; 0.084343, 0.234997, 1.577688; 1.573954, -0.158088, -0.278849; 0.779898, 0.048654, 0.245329; 0.373079, 0.209624, 0.781070; -0.252850, 0.897986, 1.077230; 0.484195, 0.725214, 0.438962}.

5.9 Застосування нейронної мережі після навчання на реальній вибірці мережевих пакетів

Для перевірки роботи нейронної мережі на реальній вибірці мережевих пакетів можна створити окремий клас з прямим розповсюдженням (forward propagation) та підставити отримані значенням матриць W1 та W2 [14]. Але можна зробити це і у класі Learning шляхом очищення масиву dataset після етапу навчання, заповнення його новою вибіркою та запуском одного кроку forward propagation (рис. 5.30).

```
    System.out.println("Accuracy: " + calc_accuracy());
    checkResultOnRealData();
    final Plot plot = Plot.create();
    plot.plot().add(errors);
    plot.title( s: "Errors");
    plot.show();
}

private static void checkResultOnRealData() throws InterruptedException {
    dataset.clear();
    addRealPairsToDataset(dataset);
    System.out.println("Real accuracy: " + calcRealAccuracy());
    Thread.sleep( millis: 60000);
}

private static double calcRealAccuracy() {
    int correct = 0;
    for (Pair<List<Double>, Integer> listIntegerPair : dataset) {
        final List<Double> inputVector = listIntegerPair.getLeft();
        double[] inputParameters = new double[]{inputVector.get(0), inputVector.get(1), inputVector.get(2)};
        final DoubleMatrix result = predict(new DoubleMatrix( newRows: 1,  newColumns: 3, inputParameters));
        int y_pred = result.argmax();
        if (y_pred == listIntegerPair.getRight()) {
            correct++;
        }
        System.out.println("Predication: " + classification.get(y_pred) + " - " + (y_pred == listIntegerPair.getRight() ? "Correct" : "Incorrect"));
    }
    return Double.parseDouble(String.valueOf(correct)) / dataset.size();
}
```

Рисунок 5.30 – Налаштування алгоритму для роботи з реальною вибіркою даних

У якості реальної вибірки мережевих пакетів було обрано масив з двадцяти пакетів (їх трьох вхідних даних та правильної відповіді) та створено метод для їх заповнення (рис. 5.31, 5.32). Варто зауважити, що правильна відповідь у даному випадку потрібна для порівняння з відповіддю нейронної мережі.

```
private static void addRealPairsToDataset(final List<Pair<List<Double>, Integer>> dataset) {
    final Pair<List<Double>, Integer> pair1 = Pair.of(List.of(1.0, 0.0, 0.0546875),  right: 1);
    final Pair<List<Double>, Integer> pair2 = Pair.of(List.of(1.0, 0.0, 0.0566875),  right: 1);
    final Pair<List<Double>, Integer> pair3 = Pair.of(List.of(1.0, 0.0, 0.0786873),  right: 1);
    final Pair<List<Double>, Integer> pair4 = Pair.of(List.of(1.0, 0.0, 0.7046875),  right: 1);
    final Pair<List<Double>, Integer> pair5 = Pair.of(List.of(1.0, 0.0, 0.0125682),  right: 1);
    final Pair<List<Double>, Integer> pair6 = Pair.of(List.of(0.0, 0.0, 0.0547885),  right: 0);
    final Pair<List<Double>, Integer> pair7 = Pair.of(List.of(0.0, 0.0, 0.0855589),  right: 0);
    final Pair<List<Double>, Integer> pair8 = Pair.of(List.of(0.0, 1.0, 0.9858965),  right: 2);
    final Pair<List<Double>, Integer> pair9 = Pair.of(List.of(0.0, 1.0, 0.0458796),  right: 2);
    final Pair<List<Double>, Integer> pair10 = Pair.of(List.of(0.0, 1.0, 0.0745896),  right: 2);
```

Рисунок 5.31 – Оголошення методу та наповнення перших десяти пар реальним мережевими пакетами

```
    final Pair<List<Double>, Integer> pair11 = Pair.of(List.of(0.0, 0.0, 0.0546875), right: 0);
    final Pair<List<Double>, Integer> pair12 = Pair.of(List.of(0.0, 0.0, 0.0658745), right: 0);
    final Pair<List<Double>, Integer> pair13 = Pair.of(List.of(0.0, 0.0, 0.8546875), right: 2);
    final Pair<List<Double>, Integer> pair14 = Pair.of(List.of(0.0, 0.0, 0.9658412), right: 2);
    final Pair<List<Double>, Integer> pair15 = Pair.of(List.of(0.0, 0.0, 0.7485698), right: 2);
    final Pair<List<Double>, Integer> pair16 = Pair.of(List.of(0.0, 1.0, 0.0452589), right: 2);
    final Pair<List<Double>, Integer> pair17 = Pair.of(List.of(1.0, 0.0, 0.0745879), right: 1);
    final Pair<List<Double>, Integer> pair18 = Pair.of(List.of(0.0, 1.0, 0.6857895), right: 2);
    final Pair<List<Double>, Integer> pair19 = Pair.of(List.of(0.0, 1.0, 0.0546875), right: 2);
    final Pair<List<Double>, Integer> pair20 = Pair.of(List.of(1.0, 1.0, 0.9856325), right: 2);

    dataset.addAll(List.of(pair1, pair2, pair3, pair4, pair5, pair6, pair7, pair8, pair9, pair10, pair11, pair12,
            pair13, pair14, pair15, pair16, pair17, pair18, pair19, pair20));

}
```

Рисунок 5.32 – Наповнення других десяти пар реальними мережевими пакетами та їх збереження у масив

Запустимо знову наш алгоритм:

```
Predication: Need clarify - Correct
Predication: Need clarify - Correct
Predication: Need clarify - Correct
Predication: Not secure - Incorrect
Predication: Need clarify - Correct
Predication: Secure - Correct
Predication: Secure - Correct
Predication: Not secure - Correct
Predication: Not secure - Correct
Predication: Not secure - Correct
Predication: Secure - Correct
Predication: Secure - Correct
Predication: Not secure - Correct
Predication: Not secure - Correct
Predication: Not secure - Correct
Predication: Not secure - Correct
Predication: Need clarify - Correct
Predication: Not secure - Correct
Predication: Not secure - Correct
Predication: Not secure - Correct
Real accuracy: 0.95
```

Рисунок 5.33 – Вивід в консоль відповідей нейронної мережі на реальних даних та відсотка правильних відповідей

Як можна бачити з рисунку 5.33, нейронна мережа правильно класифікувала 18 з 20 пакетів, тобто відсоток правильної класифікації дорівнює 95%. Цей показник все ще є високим, але відрізняється від показника наприкінці навчальної вибірки у 98.83% через те, що у цих 20 пакетах було надано також біль складні вхідні параметри (декілька параметрів, що вказують на загрозу та декілька пакетів у яких є показник загрози (2) та показник уточнення людиною (1) одночасно). Такі

складні випадки виявити у локальній мережі складніше. Система навчалася на більш простих випадках.

5.10 Висновки по розділу №5

У результаті проведення практичного дослідження застосування технологій нейронної мережі та нечіткої логіки із аналізом отриманих результатів (відсоток правильних відповідей та графік залежності помилки від ітерації алгоритму), можна зробити висновки, що створена нейронна мережа в комбінації з правилами нечіткої логіки відповідає поставленій задачі.

Наприкінці етапу навчання відсоток правильних відповідей складав 98.83% на 257-и пакетах. На 20-и пакетах більш складних реальних даних відсоток правильних відповідей складав – 95%.

Загрозливі пакети даних та пакети, що потребують уточнення адміністратором мережі з навчальної вибірки були внесені в журнал аудиту – текстовий файл.

Є доцільним подальша оптимізація даного алгоритму, наприклад, пришвидшення часу навчання нейронної мережі. Також варто зробити більш гнучким алгоритм нечіткої логіки, для того, щоб швидко сформувати нову навчальну вибірку та навчити нейронну систему роботі з новими даними.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було обґрунтовано і практично перевірено доцільність застосування комбінації технологій штучного інтелекту для поліпшення роботи IDS/IPS, чим підтверджено досягнення поставленої мети.

Шляхом аналізу технологій штучного інтелекту у системах виявлення та запобігання вторгнень у мережу, було наведено мету їх впровадження, їх переваги та недоліки у порівнянні між собою та у порівнянні з традиційними системами виявлення вторгнень (експертними системами).

Сучасні засоби виявлення атак на основі експертних систем часто використовуються компаніями, відповідальними за кібербезпеку, але попри те, що дані системи досі користуються популярністю, вони вже не можуть повністю забезпечити безпеку мережі. Експертні системи засновані на правилах, що ідентифікують відомі атаки. Вони вимагають частого оновлення, інакше кажучи, послаблюються можливості системи захисту. При цьому при виявленні нового типу атаки потрібен певний час на обробку даної атаки експертами та додавання відповідного нового правила до бази даних. Ці системи не мають достатньої гнучкості при реалізації структури типу «правило-перевірка». Незначні варіації деталей атаки можуть призвести до її пропуску.

Маючи на увазі перераховані вище проблеми, рішенням може стати застосування алгоритмів штучного інтелекту при вирішенні завдань аналізу мережевого трафіку і завдань виявлення вторгнень у мережу.

У роботі було проаналізовано особливості застосування технологій штучного інтелекту в системах виявлення/запобігання вторгнень у мережу. Детально описані три технологій штучного інтелекту: нейронні мережі, генетичний алгоритм та нечітка логіка, проведено їх порівняльний аналіз та відзначені певні переваги та недоліки.

Було розроблено й практично реалізовано систему виявлення/запобігання вторгнень на основі правил нечіткої логіки та технології нейронної мережі. Отримані результати показали, що пакети мережевого трафіку були успішно оброблені за заздалегідь зазначеними критеріями й властивостями та перетворені у навчальну вибірку для нейронної мережі. Загрозливі пакети та пакети, що потребують додаткової перевірки адміністратором були зареєстровані для подальшого спостереження в журналі аудиту. Нейронна мережа була успішно навчена та мала відсоток правильних відповідей 98.83% на навчальній вибірці. На навченій мережі було протестовано реальний мережевий трафік - на ньому мережа мала відсоток правильних відповідей 95%.

У підсумку можна рекомендувати використання технології нейронної мережі у поєднанні з нечіткою логікою у системах виявлення/запобігання вторгнень у мережу. Це обумовлено тим, що при комбінації технології нейронної мережі з правилами нечіткої логіки можна отримати наступні переваги: меншу кількість хибних викликів, більш гнучке налаштування виявлення загроз у мережі та скорочення часу на навчання системи ШІ. Її гнучкість ґрунтується на тому, що на вхід нейронної мережі можна подавати будь-які вхідні параметри мережевого пакету, які будуть оброблені та уніфіковані за допомогою правил нечіткої логіки.

Висновки є справедливими не тільки для локальних мереж подібних до тієї, що була відтворена у роботі, а також для великих мереж з досить великим об’ємом трафіку. Для впровадження такої системи виявлення/запобігання вторгнень у велику мережу достатнім буде наявність алгоритму обробки мережевих даних та їх уніфікація для створення навчальної вибірки. Це може бути темою проведення подальших досліджень.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. A. Midzic, Z. Avdagic, S. Omanovic. Intrusion detection system modeling based on neural networks and fuzzy logic. Budapest: IEEE 20th Jubilee International Conference on Intelligent Engineering Systems (INES), 2016. 194 p.
2. J. Kivimaa, A. Ojamaa, E. Tyugu. Graded Security Expert System. Lecture Notes in Computer Science. New York: Springer, 2009. 286 p.
3. O. Linda, T. Vollmer, M. Manic. Neural Network based Intrusion Detection System for critical infrastructures. Calgary: International Joint Conference on Neural Networks, 2009. 1834 p.
4. E. Tyugu. Algorithms and Architectures of Artificial Intelligence. Talin: IOS Press, 2007. 267 p.
5. Calderon R. The Benefits of Artificial Intelligence in Cybersecurity. Washington: Economic Crime Forensics Capstones, 2019. 36 p.
6. B. Muller, J. Reinhardt. Neural Networks: An Introduction. The Physics of Neural Networks Series. Berlin: Springer-Verlag, 1991. 266 p.
7. Artificial Intelligence for Cyber Security: Performance Analysis of Network Intrusion Detection / Shahriar Usman Khan et al. Cham: Springer, 2022. 140 p.
8. R. Shanmugavadivu. Network Intrusion Detection System Using Fuzzy Logic. Hyderabad: Indian Journal of Computer Science and Engineering (IJCSE), 2011. 111 p.
9. Pohlheim Hartmut. Genetic and Evolutionary Algorithms: Principles, Methods and Algorithms. Chennai: 2003. 128 p.
10. B. M. Wilamowski. Neural network architectures and learning algorithms. Rio de Janeiro: IEEE Industrial Electronics Magazine, 2009. 63p.
11. B. M. Wilamowski. Neural networks and fuzzy systems for nonlinear applications. Bucharest: Proc. 11th INES 2007-11th Int. Conf. Intelligent Engineering Systems, 2007. 19 p.

12. D. E. Rumelhart, G. E. Hinton, R. J. Wiliams. Learning representations by back-propagating errors. New York: Nature, 1986. 536 p.
13. Computing gradient vector and Jacobian matrix in arbitrarily connected neural networks / B. M. Wilamowski, N. J. Cotton, 0. Kaynak, G. Dundar. Krakow: IEEE Trans. Ind. Electron, 2008. 3790 p.
14. S. E. Fahlman. Faster-learning variations on back-propagation: An empirical study. San Mateo: Morgan Kaufmann, 1988. 278 p.
15. H. Yu, B. M. Wilamowski. Efficient and reliable training of neural networks. Paris: 2nd IEEE Human System Interaction Conference HSI, 2009. 115 p.
16. Haykin N. Neural networks: a comprehensive foundation. New Jersey, 1999. 842 p.
17. Duchi J., Hazan E., Singer Y. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Warsaw: Journal of Machine Learning Research, 2011. 2159 p.
18. Progress in Study of Encrypted Traffic Classification / Cao Z., Cao S., Xiong G., Guo L. Bejing: In Proceedings of International standard conference on trustworthy computing and services, 2012. 86 p.
19. J. Erman, M. Arlitt, A. Mahanti. Traffic Classificaton Using Clustering Algorithms. Pisa: ACM SIGCOMM MineNet Workshop, 2006. 230 p.
20. The inadequacy of entropy-based ransomware detection / McIntosh T., Jang-Jaccard J., Watters P., Susnjak T. New York: Springer, 2019. 189 p.
21. An IoT-Focused Intrusion Detection System Approach Based on Preprocessing Characterization for Cybersecurity Datasets / Xavier Larriva-Novo et al. Istanbul: Sensors, 2021. 656p.
22. An Approach for the Application of a Dynamic Multi-Class Classifier for Network Intrusion Detection Systems / Xavier Larriva-Novo et al. Rome: Electronics, 2020. 1759p.
23. Design and Implementation of Network Traffic Analysis System for Power Grid Metering System / Tao Liu et al. Beijing: 27th IEEE International Conference on Data Science in Cyberspace (DSC), 2022. 332 p.

24. Intrusion Detection System using Autoencoder based Deep Neural Network for SME Cybersecurity / Khaizuran Aqhar Ubaidillah et al. Dubai: 5th International Conference on Informatics and Computational Sciences (ICICoS), 2021. 215 p.
25. Akshay Rajapkar, Pranita Binnar, Faruk Kazi. Design of Intrusion Prevention System for OT Networks Using Deep Neural Networks. West Bengal: 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), 2020. 56 p.
26. Ahmed O. Oyedele, Anuoluwapo O. Ajayi, Lukumon O. Oyedele. Machine learning predictions for lost time injuries in power transmission and distribution projects. Berlin: Machine Learning with Applications, 2021. 158 p.
27. An intelligent ensemble of long -short -term memory with genetic algorithm for network anomaly identification / I. Sumaiya Thaseen et al. Dublin: Transactions on Emerging Telecommunications Technologies, 2020. 178 p.
28. Detection of Network Anomaly Sequences Using Deep Recurrent Neural Networks / R. Ravinder Reddy, K. Ayyappa Reddy, C. Madan Kumar, Y. Ramadevi. San Mateo: Smart Computing Techniques and Applications, 2021. 605 p.
29. S. Apel, C. Kastner. An overview of feature-oriented software development. London: Journal of Object Technology, 2009. 84 p.
30. Шелухин О. И. Обнаружение вторжений в компьютерные сети (сетевые аномалии). Москва: Горячая линия – Телеком, 2013. 220 с.
31. Mrudul Dixit, Rajashwini Ukarande. Internet Traffic Intrusion Detection System Using Adaptive Neuro-Fuzzy Inference System. Pune: Smart Trends in Information Technology and Computer Communications, 2018. 35 p.

ДОДАТОК А

КОД РЕАЛІЗАЦІЇ АЛГОРИТМУ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ ВИКОНАННЯ ЗАВДАНЬ СИСТЕМ IDS/IPS

Клас Classification:

```
package fuzzy_logic;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Iterator;
import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import org.apache.commons.lang3.tuple.Pair;

public class Classification {

    private static final Logger logger = new Logger();

    private static final String PASSWORD = "password";

    private static final String LOGIN = "login";

    private static final String CREDENTIALS = "credentials";

    private static final String FORBIDDEN_WORDS_DETECTED =
            "Words related to credential info detected in the packet" +
System.lineSeparator();

    private static final String TELNET_REQUEST_DETECTED =
            "Telnet connect request detected in the packet" +
System.lineSeparator();

    private static final String FTP_REQUEST_DETECTED =
            "FTP connect request detected in the packet" + System.lineSeparator();

    private static final String SSH_REQUEST_DETECTED =
            "SSH connect request detected in the packet" + System.lineSeparator();

    private static final String LARGE_LENGTH_DETECTED =
            "Packet with length larger than 500 bytes detected" +
System.lineSeparator();

    private static final String SOURCE_AND_DESTINATION_IPS = "Src: 192.168.0.101,
Dst: 192.168.0.25";

    private static final String DUP = "Dup";

    private static final String OUT_OFF = "Out-Of";

    private static final String TCP_PROTOCOL = "TCP";

    private static final String TELNET = "Telnet";

    private static final String SSH = "SSH";
```

```
    private static final String FTP = "FTP";

    private static final String PORT21 = "Port: 21";

    private static final String PORT22 = "Port: 22";

    private static final String PORT23 = "Port: 23";

    private static final String PACKET_LENGTH_REGEX = "\\d+(?= bytes on wire)";

    private static final double BYTE_TO_KILOBYTE = 0.0009765625;

    public void createDataset(final ArrayList<String> packets, final
List<Pair<List<Double>, Integer>> dataset) {
        addPairsToTDataset(packets, dataset);
    }

    private void addPairsToTDataset(final ArrayList<String> packets,
                                    final List<Pair<List<Double>, Integer>>
dataset) {
        final Pattern pattern = Pattern.compile(PACKET_LENGTH_REGEX);

        for (Iterator<String> iterator = packets.iterator(); iterator.hasNext(); )
{
            final String packet = iterator.next();
            final Matcher matcher = pattern.matcher(packet);
            final List<Double> inputParameters = new
ArrayList<>(Arrays.asList(0.0, 0.0, 0.0));
            int outputValue = 0;

            if (packet.contains(PASSWORD) || packet.contains(LOGIN) ||
packet.contains(CREDENTIALS)) {
                logger.logging(FORBIDDEN_WORDS_DETECTED + packet);
                inputParameters.set(0, 1.0);
                outputValue = 1;
            } else if (packet.contains(TCP_PROTOCOL) &&
packet.contains(SOURCE_AND_DESTINATION_IPS)
                    && packet.contains(PORT23) && !packet.contains(TELNET)
                    && !packet.contains(DUP) && !packet.contains(OUT_OFF)) {
                logger.logging(TELNET_REQUEST_DETECTED + packet);
                inputParameters.set(1, 1.0);
                outputValue = 2;
            } else if (packet.contains(TCP_PROTOCOL) &&
packet.contains(SOURCE_AND_DESTINATION_IPS) && !packet.contains(SSH)
                    && packet.contains(PORT22) && !packet.contains(DUP) &&
!packet.contains(OUT_OFF)) {
                logger.logging(SSH_REQUEST_DETECTED + packet);
                inputParameters.set(1, 1.0);
                outputValue = 2;
            } else if (packet.contains(TCP_PROTOCOL) &&
packet.contains(SOURCE_AND_DESTINATION_IPS) && !packet.contains(FTP)
                    && packet.contains(PORT21) && !packet.contains(DUP) &&
!packet.contains(OUT_OFF)) {
                logger.logging(FTP_REQUEST_DETECTED + System.lineSeparator() +
packet);
                inputParameters.set(1, 1.0);
                outputValue = 2;
            }
            while (matcher.find()) {
                final double data = Integer.parseInt(matcher.group());
                if (data > 500) {
                    logger.logging(LARGE_LENGTH_DETECTED + packet);
```

```
                    outputValue = 2;
                }
                if (outputValue == 0) {
                    dataset.add(Pair.of(List.of(0.0, 0.0, data *
BYTE_TO_KILOBYTE), outputValue));
                } else {
                    dataset.add(Pair.of(List.of(inputParameters.get(0),
inputParameters.get(1),
                            data * BYTE_TO_KILOBYTE), outputValue));
                }
            }
            iterator.remove();
        }
    }
}
```

КласFuzzyLogic:

```
package fuzzy_logic;

import java.io.File;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
import java.util.logging.Level;

import org.apache.commons.lang3.tuple.Pair;


public class FuzzyLogic
{

      private static final String FILE_WITH_PACKETS =
"src/main/resources/packets.txt";

      private static final String IO_EXCEPTION_MESSAGE = "Unable to open file.";

      private static final String NUMBER_OF_PACKET = "No.";

      public List<Pair<List<Double>, Integer>> execute() throws IOException
      {
            final List<Pair<List<Double>, Integer>> dataset = new ArrayList<>();
            final Logger logger = new Logger();
            final Classification classification = new Classification();

            classification.createDataset(getInput(), dataset);
            logger.writeLogs();
            return dataset;
      }

      private ArrayList<String> getInput()
      {
            final StringBuilder sb = new StringBuilder();
            final ArrayList<String> arr = new ArrayList<>();

            try
            {
                  final Scanner scanner = new Scanner(new File(FILE_WITH_PACKETS),
StandardCharsets.UTF_8.toString());
                  while (scanner.hasNextLine())
                  {
                        final String s = scanner.nextLine();
```

```
                        if (s.contains(NUMBER_OF_PACKET))
                        {
                              arr.add(sb.toString());
                              sb.setLength(0);
                        }
                        sb.append(s).append(System.lineSeparator());
                  }
                  scanner.close();
            }
            catch (IOException ex)
            {

      java.util.logging.Logger.getLogger(FuzzyLogic.class.getName()).log(Level.SEV
ERE, IO_EXCEPTION_MESSAGE, ex);
            }
            return arr;
      }
}
```

Клас Logger:

```
package fuzzy_logic;

import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class Logger {
    private static final StringBuilder logMessages = new StringBuilder();

    public void logging(String str) {
        Scanner in = new Scanner(str);
        String s = in.nextLine() + System.lineSeparator() + in.nextLine() +
System.lineSeparator() + in.nextLine();

logMessages.append(s).append(System.lineSeparator()).append(System.lineSeparator()
);
        in.close();
    }

    public void writeLogs() throws IOException {
        FileWriter fw1 = new FileWriter("logs.txt");
        fw1.write(logMessages.toString());
        fw1.flush();
    }
}
```

Клас Learning:

```
package neural_network;

import fuzzy_logic.FuzzyLogic;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import com.github.sh0nk.matplotlib4j.Plot;
import com.github.sh0nk.matplotlib4j.PythonExecutionException;

import org.apache.commons.lang3.tuple.Pair;
import org.jblas.DoubleMatrix;
import org.jblas.MatrixFunctions;

public class Learning {
```

```
    private final static int numberOfInputParameters = 3;
    private final static int numberOfOutputs = 3;
    private final static int numberOfNeurons = 10;
    private static DoubleMatrix W1 = DoubleMatrix.rand(numberOfInputParameters,
numberOfNeurons);
    private static DoubleMatrix b1 = DoubleMatrix.rand(1, numberOfNeurons);
    private static DoubleMatrix W2 = DoubleMatrix.rand(numberOfNeurons,
numberOfOutputs);
    private static DoubleMatrix b2 = DoubleMatrix.rand(1, numberOfOutputs);

    private static final String SECURE = "Secure";

    private static final String NOT_SECURE = "Not secure";

    private static final String NEED_CLARIFY = "Need clarify";
    static final List<String> classification = new ArrayList<>();


    private static final double LEARNING_SPEED = 0.001;
    private static List<Pair<List<Double>, Integer>> dataset;

    public static void main(String[] args) throws PythonExecutionException,
IOException, InterruptedException {
        final List<Double> errors = new ArrayList<>();
        final FuzzyLogic fuzzyLogic = new FuzzyLogic();

        classification.add(SECURE);
        classification.add(NEED_CLARIFY);
        classification.add(NOT_SECURE);
        dataset = fuzzyLogic.execute();

        for (int j = 0; j < 100; j++) {
            for (Pair<List<Double>, Integer> listIntegerPair : dataset) {
                final List<Double> inputVector = listIntegerPair.getLeft();
                double[] inputParameters = new double[]{inputVector.get(0),
inputVector.get(1), inputVector.get(2)};
                final DoubleMatrix x = new DoubleMatrix(1, 3, inputParameters);
                final int y = listIntegerPair.getRight();
                System.out.println(x + System.lineSeparator() + y);

                final DoubleMatrix t1 = x.mmul(W1).add(b1);
                final DoubleMatrix h1 = RELU(t1);
                final DoubleMatrix t2 = h1.mmul(W2).add(b2);
                final DoubleMatrix result = softMax(t2);

                final double error = sparseCrossEntropy(result, y);

                final DoubleMatrix yFull = toFull(y, classification);
                final DoubleMatrix dE_dt2 = result.sub(yFull);
                final DoubleMatrix dE_dW2 = h1.transpose().mmul(dE_dt2);
                final DoubleMatrix dE_db2 = dE_dt2;
                final DoubleMatrix dE_dh1 = dE_dt2.mmul(W2.transpose());
                final DoubleMatrix dE_dt1 = dE_dh1.mmul(relu_deriv(t1));
                final DoubleMatrix dE_dW1 = x.transpose().mmul(dE_dt1);
                final DoubleMatrix dE_db1 = dE_dt1;

                W1 = W1.sub(dE_dW1.mul(LEARNING_SPEED));
                b1 = b1.sub(dE_db1.mul(LEARNING_SPEED));
                W2 = W2.sub(dE_dW2.mul(LEARNING_SPEED));
                b2 = b2.sub(dE_db2.mul(LEARNING_SPEED));
                errors.add(error);
            }
```

```
        }
        System.out.println(W1 + System.lineSeparator() + W2);
        System.out.println("Accuracy: " + calc_accuracy());
        checkResultOnRealData();
        final Plot plot = Plot.create();
        plot.plot().add(errors);
        plot.title("Errors");
        plot.show();
    }

    private static void checkResultOnRealData() throws InterruptedException {
        dataset.clear();
        addRealPairsToDataset(dataset);
        System.out.println("Real accuracy: " + calcRealAccuracy());
        Thread.sleep(60000);
    }

    private static double calcRealAccuracy() {
        int correct = 0;
        for (Pair<List<Double>, Integer> listIntegerPair : dataset) {
            final List<Double> inputVector = listIntegerPair.getLeft();
            double[] inputParameters = new double[]{inputVector.get(0),
inputVector.get(1), inputVector.get(2)};
            final DoubleMatrix result = predict(new DoubleMatrix(1, 3,
inputParameters));
            int y_pred = result.argmax();
            if (y_pred == listIntegerPair.getRight()) {
                correct++;
            }
            System.out.println("Predication: " + classification.get(y_pred) + " -
" + (y_pred == listIntegerPair.getRight() ? "Correct" : "Incorrect"));
        }
        return Double.parseDouble(String.valueOf(correct)) / dataset.size();
    }

    private static double calc_accuracy() {
        int correct = 0;
        for (Pair<List<Double>, Integer> listIntegerPair : dataset) {
            final List<Double> inputVector = listIntegerPair.getLeft();
            double[] inputParameters = new double[]{inputVector.get(0),
inputVector.get(1), inputVector.get(2)};
            final DoubleMatrix result = predict(new DoubleMatrix(1, 3,
inputParameters));
            int y_pred = result.argmax();
            if (y_pred == listIntegerPair.getRight()) {
                correct++;
            }
        }
        return Double.parseDouble(String.valueOf(correct)) / dataset.size();
    }

    private static DoubleMatrix predict(final DoubleMatrix x) {
        final DoubleMatrix t1 = x.mmul(W1).add(b1);
        final DoubleMatrix h1 = RELU(t1);
        final DoubleMatrix t2 = h1.mmul(W2).add(b2);
        return softMax(t2);
    }

    private static float relu_deriv(final DoubleMatrix vector) {
        return vector.sum() >= 0 ? 1 : 0;
    }
```

```
    private static double sparseCrossEntropy(final DoubleMatrix result, final int
y) {
        return -Math.log(result.get(0, y));
    }

    private static DoubleMatrix toFull(final int y, final List<String>
classification) {
        final DoubleMatrix yFull = DoubleMatrix.zeros(1, classification.size());
        yFull.put(0, y, 1);
        return yFull;
    }

    private static DoubleMatrix RELU(final DoubleMatrix t) {
        return t.maxi(0, t);
    }

    private static DoubleMatrix softMax(final DoubleMatrix t) {
        final DoubleMatrix out = MatrixFunctions.exp(t);
        return out.divi(out.sum());
    }

    private static void addRealPairsToDataset(final List<Pair<List<Double>,
Integer>> dataset) {
        final Pair<List<Double>, Integer> pair1 = Pair.of(List.of(1.0, 0.0,
0.0546875), 1);
        final Pair<List<Double>, Integer> pair2 = Pair.of(List.of(1.0, 0.0,
0.0566875), 1);
        final Pair<List<Double>, Integer> pair3 = Pair.of(List.of(1.0, 0.0,
0.0786873), 1);
        final Pair<List<Double>, Integer> pair4 = Pair.of(List.of(1.0, 0.0,
0.7046875), 1);
        final Pair<List<Double>, Integer> pair5 = Pair.of(List.of(1.0, 0.0,
0.0125682), 1);
        final Pair<List<Double>, Integer> pair6 = Pair.of(List.of(0.0, 0.0,
0.0547885), 0);
        final Pair<List<Double>, Integer> pair7 = Pair.of(List.of(0.0, 0.0,
0.0855589), 0);
        final Pair<List<Double>, Integer> pair8 = Pair.of(List.of(0.0, 1.0,
0.9858965), 2);
        final Pair<List<Double>, Integer> pair9 = Pair.of(List.of(0.0, 1.0,
0.0458796), 2);
        final Pair<List<Double>, Integer> pair10 = Pair.of(List.of(0.0, 1.0,
0.0745896), 2);

        final Pair<List<Double>, Integer> pair11 = Pair.of(List.of(0.0, 0.0,
0.0546875), 0);
        final Pair<List<Double>, Integer> pair12 = Pair.of(List.of(0.0, 0.0,
0.0658745), 0);
        final Pair<List<Double>, Integer> pair13 = Pair.of(List.of(0.0, 0.0,
0.8546875), 2);
        final Pair<List<Double>, Integer> pair14 = Pair.of(List.of(0.0, 0.0,
0.9658412), 2);
        final Pair<List<Double>, Integer> pair15 = Pair.of(List.of(0.0, 0.0,
0.7485698), 2);
        final Pair<List<Double>, Integer> pair16 = Pair.of(List.of(0.0, 1.0,
0.0452589), 2);
        final Pair<List<Double>, Integer> pair17 = Pair.of(List.of(1.0, 0.0,
0.0745879), 1);
        final Pair<List<Double>, Integer> pair18 = Pair.of(List.of(0.0, 1.0,
0.6857895), 2);
        final Pair<List<Double>, Integer> pair19 = Pair.of(List.of(0.0, 1.0,
0.0546875), 2);
```

```
        final Pair<List<Double>, Integer> pair20 = Pair.of(List.of(1.0, 1.0,
0.9856325), 2);

        dataset.addAll(List.of(pair1, pair2, pair3, pair4, pair5, pair6, pair7,
pair8, pair9, pair10, pair11, pair12,
                pair13, pair14, pair15, pair16, pair17, pair18, pair19, pair20));

    }

}
```

Клас Main з останніми отриманими ваговими коефіцієнтами у якості прикладу:

```
package neural_network;

import java.util.ArrayList;
import java.util.List;

import org.jblas.DoubleMatrix;
import org.jblas.MatrixFunctions;

public class Main {
    private final static int numberOfInputParameters = 3;
    private final static int numberOfOutputs = 3;
    private final static int numberOfNeurons = 10;

    private static DoubleMatrix W1 = null;
    private final static DoubleMatrix b1 = DoubleMatrix.rand(1, numberOfNeurons);
    private static DoubleMatrix W2 = null;
    private final static DoubleMatrix b2 = DoubleMatrix.rand(1, numberOfOutputs);

    public static void main(String[] args) {
        final DoubleMatrix x = new DoubleMatrix(1, numberOfInputParameters, 1.0,
0.0, 0.085965);
        double[] inputParameters1 = new double[]{0.678594, 1.120166, 0.907341, -
0.195015, -0.719279, -0.678388, -0.237843, 1.707015, 0.367556, 0.575939, 1.303518,
-0.179109, 0.489250, 1.796915, -0.309570, 0.254518, -0.236776, 0.757262, 0.885834,
1.688933, 1.848229, 0.504275, -0.280635, 2.330228, -1.071830, 0.074957, -0.614854,
0.570294, 1.226246, 1.785892};
        double[] inputParameters2 = new double[]{-0.597135, 1.025964, 2.259646,
0.389469, 1.055216, 0.058478, 0.482823, 1.220923, 0.012543, -0.853735, -0.382717,
2.349390, 2.268946, -0.279740, -0.378773, 1.429781, -0.083312, 0.707316, 1.761385,
0.188275, -0.120301, -0.509565, 1.243176, 0.058074, -0.291231, 0.877258, 1.630994,
-1.077610, 0.277451, 1.898262};

        W1 = new DoubleMatrix(numberOfInputParameters, numberOfNeurons,
inputParameters1);
        W2 = new DoubleMatrix(numberOfNeurons, numberOfOutputs, inputParameters2);

        final DoubleMatrix probs = predict(x);
        final int max = probs.argmax();
        final List<String> classification = new ArrayList<>();
        classification.add("Secure");
        classification.add("Need Clarify");
        classification.add("Not secure");
        System.out.println("Predication: " + classification.get(max));
    }


    private static DoubleMatrix RELU(final DoubleMatrix t) {
        return t.maxi(0, t);
    }
```

```
    private static DoubleMatrix softMax(final DoubleMatrix t) {
        final DoubleMatrix out = MatrixFunctions.exp(t);
        return out.divi(out.sum());
    }

    private static DoubleMatrix predict(final DoubleMatrix x) {
        final DoubleMatrix t1 = x.mmul(W1).add(b1);
        final DoubleMatrix h1 = RELU(t1);

        final DoubleMatrix t2 = h1.mmul(W2).add(b2);
        return softMax(t2);
    }
}
```