A DEEP REINFORCEMENT LEARNING APPROACH TO NETWORK INTRUSION DETECTION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

HALİM GÖRKEM GÜLMEZ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2019

Approval of the thesis:

A DEEP REINFORCEMENT LEARNING APPROACH TO NETWORK INTRUSION DETECTION

submitted by **HALİM GÖRKEM GÜLMEZ** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering** _____

Assist. Prof. Dr. Pelin Angın
Supervisor, **Computer Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Sevgi Özkan Yıldırım
Graduate School of Informatics, METU _____

Assist. Prof. Dr. Pelin Angın
Computer Engineering, METU _____

Assoc. Prof. Dr. Ahmet Burak Can
Computer Engineering, Hacettepe University _____

Date: 03.09.2019

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Halim Görkem Gülmez

Signature:

ABSTRACT

A DEEP REINFORCEMENT LEARNING APPROACH TO NETWORK INTRUSION DETECTION

Gülmez, Halim Görkem
Master of Science, Computer Engineering
Supervisor: Assist. Prof. Dr. Pelin Angın

September 2019, 64 pages

Intrusion detection is one of the most important problems in today's world. Every day new attacks are being used in order to breach the security of systems and signature-based security systems fail to detect these zero-day attacks. An anomaly-based intrusion detection system, particularly one that utilizes a machine learning approach, is needed to effectively handle these kinds of attacks. With the advancements in big data technologies, storing and handling data became easier, therefore big data analytics has become an indispensable tool for various tasks. In this thesis, we propose a framework for detecting intrusions in network systems using big data analytics in real time. The framework is built on Apache Spark, which runs anomaly detection algorithms on streaming data after it has been trained offline with the normal behavior of the system. Two different machine learning solutions have been implemented separately for comparison: long short-term memory recurrent neural networks and deep reinforcement learning. Reinforcement learning is built on state and action pairs with associated positive or negative awards. For the solution in this thesis, alerts on attacks and non-alerts on normal behavior are positively rewarded to train learning agents. Reinforcement learning is combined and improved with neural networks by using them for Q-learning. A variety of intrusion detection datasets from the literature are used for experimentation, including NSL-KDD, UNSW-NB15 and CICIDS2017.

The deep reinforcement learning solution is emphasized as the better solution based on the experiment results.

Keywords: Cybersecurity, Long Short-term Memory Recurrent Neural Networks, Deep Reinforcement Learning, Big Data, Cloud Systems

ÖZ

AĞ SALDIRI TESPİTİNDE DERİN PEKİŞTİRMELİ ÖĞRENME YAKLAŞIMI

Gülmez, Halim Görkem
Yüksek Lisans, Bilgisayar Mühendisliği
Tez Danışmanı: Doç. Dr. Pelin Angın

Eylül 2019, 64 sayfa

Ağ saldırıların tespiti günümüzdeki en önemli problemlerden biridir. Her gün yeni ataklar güvenlik sistemlerini delme amacıyla kullanılmaktadır. İmza tabanlı güvenlik sistemleri bu sıfır-gün ataklarını tespit etmekte başarısız olmaktadır. Anomali tabanlı bir sistem, özellik bir makine öğrenmesi yaklaşımından faydalanan bir sistem, bu tarz atakların tespitinde gereklidir. Bu tez ağ sistemlerinin güvenliğinin büyük veri analitiklerinin gerçek zamanlı kullanılmasıyla sağlanması üzerinedir. Büyük veri teknolojilerindeki gelişmeler, verinin saklanmasını ve işlenmesini kolaylaştırmıştır, bu nedenle büyük veri analitikleri birçok amaç için kullanılabilecek bir kaynak haline gelmiştir. Bu tezde önerilen sistem akan büyük veri kullanarak çalışacaktır. Kullanılan veri, ağ hakkında güvenlik açısından önemli bilgileri içermektedir. Apache Spark, verinin işlenmesinde araç olarak kullanılacaktır. Büyük veri işlendikten sonra çözümün kendini eğiten ve anomalileri yakalayan makine öğrenmesi kısmına aktarılacaktır. Uzun kısa-dönem hafızalı yinelemeli sinir ağları ve derin pekiştirmeli öğrenme gibi farklı makine öğrenmesi çözümleri karşılaştırma amacıyla kullanılmıştır. Pekiştirmeli öğrenme durum ve aksiyon ikilerinin pozitif ya da negatif ödüllendirilmesi üzerine kurulmuştur. Bu tezde önerilen çözümde, anomali olduğunda uyarı yapılması ya da normal durumlarda uyarı yapılmaması pozitif olarak ödüllendirilmiştir. Pekiştirmeli öğrenme, Q-learning'de kullanılmak üzere sinir ağları ile birleştirilmiş ve geliştirilmiştir. NSL-KDD, UNSW-NB15, CICIDS2017 gibi farklı veri kümeleriyle deneyler yapılmıştır. Farklı senaryolarla birlikte en iyi çözüm bulunmaya çalışılmış, detaylıca çözümler test edilmiştir. Deneyler sonucunda, derin pekiştirmeli öğrenme çözümünün diğer çözüme kıyasla daha iyi sonuç verdiği vurgulanmıştır.

Anahtar Kelimeler: Sibergüvenlik, Uzun Kısa-Dönem Hafızalı Özyinelemeli Sinir Ağları, Derin Pekiştirimeli Öğrenme, Büyük Veri, Bulut Sistemler

For my dearest wife Çağda Gizem, and for the best advisor ever existed Pelin Angın.

ACKNOWLEDGEMENTS

First, I would like to thank my advisor, Assist. Prof. Dr. Pelin Angın, without whom this thesis would not happen. Her ideas and perspective always carried me forward. It was a great pleasure working with her.

I also would like to thank my family, because of their constant support not only throughout writing this thesis, but also in my life in general. They have motivated me all the time to accomplish my goals.

Finally, I would like to thank all my colleagues in TechNarts, taking some of the responsibilities off my shoulders when needed.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

API	Application Programming Interface
AWS	Amazon Web Services
DRL	Deep Reinforcement Learning
IDS	Intrusion Detection System
KDD	Knowledge Discovery and Data Mining
LSTM RNN	Long Short-term Memory Recurrent Neural Networks
UNSW ADFA	The University of New South Wales Canberra at Australian Defence Force Academy
VM	Virtual Machine

CHAPTER 1

INTRODUCTION

1.1. Rise of the Big Data

Big data has become a popular concept with the technological advances in the past decade. Before then, it was extremely costly to store the data because acquiring the needed disk space was not as easy as today. Cloud systems were expensive and not common, therefore, to store data, companies needed to have their own storage facilities which required time, space and money, much more than compared with today. Today, we have unlimited storage, especially provided by cloud services, at our disposal. With the competition between service providers, the prices of these services are decreasing, whereas the quality of the services are rising. Moreover, many different frameworks and tools exist to handle different types of big data. All these advancements have recently made big data one of the most popular concepts in computer science, which can be seen in the figure below (Google Trends, 2019).

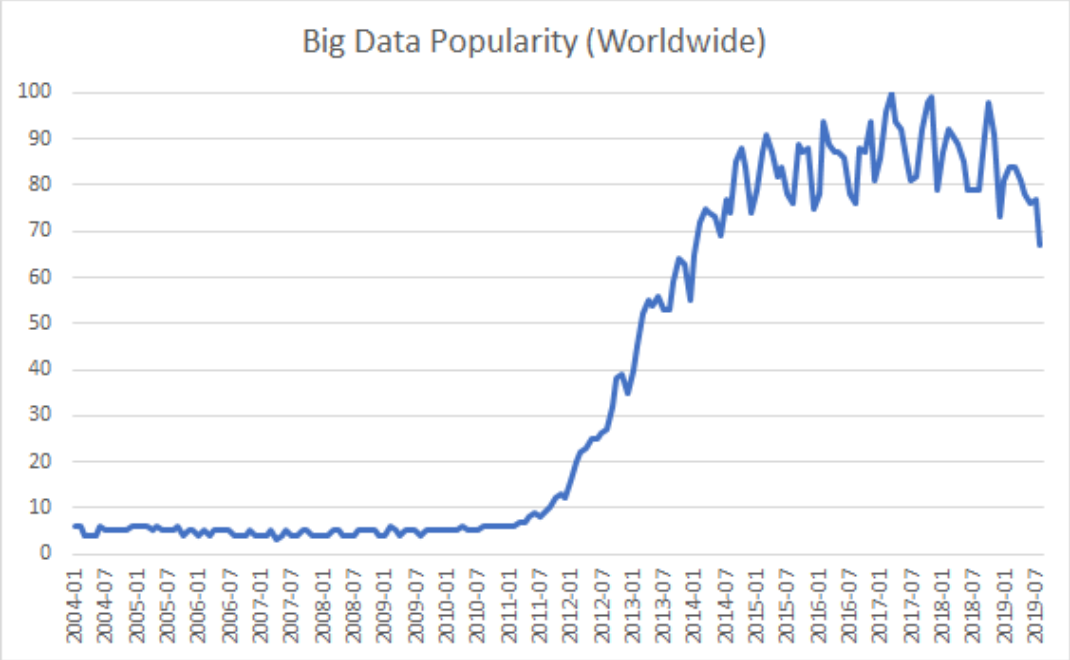


Figure 1.1. Rising Popularity of Big Data (Google Trends, 2019)

The ability to store and process data in large volumes and velocity provides significant benefits for analyzing network data in real time to detect anomalies that could signal presence of attacks on the system. In this thesis, a big data analytics approach is proposed for intrusion detection in networks, based on different machine learning solutions, namely long short-term memory recurrent neural networks, and deep reinforcement learning, which are capable of incorporating the temporal behavior of the system into the anomaly detection task. Since the system is desired to respond in near real-time, streaming big data from the network is used. Shahrivari (2014) stated that batch processing needs the batch to load before starting to process, which is not suitable for our task. As filling a batch might take several hours, building a security system upon this kind of structure would be a mistake, because most of the attacks will do their harm in a matter of seconds. Streaming data processing, on the other hand, works in real-time, and enables real-time queries on the side, too. Therefore, stream processing has been chosen as the suitable means of processing in our solution.

An initial prototype of the proposed system using Apache Spark and the TensorFlow machine learning framework for both long short-term memory recurrent neural networks and deep reinforcement learning has been developed and promising results have been achieved with experiments on large network attack datasets including KDD Cup (1999), NSL-KDD (Mahbod et al., 2009), CICIDS2017 (2017) and UNSW-NB15 (2015).

1.2. Increasing Popularity of Cloud Computing

In the last decade, networking and virtualization technologies rapidly advanced. These advancements have made cloud platforms the most popular choice for the data storage and computing needs of many enterprises. As major cloud service providers offer reliable platforms, many companies have migrated their infrastructures and systems to those platforms.

With the increasing popularity of the cloud platforms and their decreasing costs, they have become one of the main targets of cyberattacks. Similarly, popularity of the Internet of Things (IoT) paradigm has widened the attack surface for cloud systems and increased the number of vulnerabilities attackers can exploit. Additionally, Ibrahim, Hamlyn-Harris, and Grundy (2016) stated that the unique nature of cloud infrastructures have created new security threats. In the Global Threat Report of Carbon Black (2019), it is stated that users of Carbon Black's cloud security solutions are seeing a total of one million attempted cyberattacks per day.

The vulnerability of cloud platforms to attacks creates a need for security measures to prevent those attacks or detect them and take action when an attack happens. Although some of the existing security methods can be applied to cloud systems, new methods designed truly for the cloud systems would be preferable.

Most of the cloud providers provide monitors for their systems. These monitors track various metrics of the system, such as CPU utilization, disk I/O's, memory utilization etc. API's for accessing these metrics are provided too, which makes them usable for custom security solutions.

1.3. Importance of Detecting Attacks in Real-time

Online systems, especially cloud systems, are prone to zero-day attacks, which are malicious activities not observed previously. Even though security mechanisms are updated quickly, even seconds are important to provide high reliability in such systems. Hence, a real-time security system, which is not signature-based, is more suitable for the cloud. The system should be able to detect both old and new attack methods as fast as possible. As the system will be under new attacks every day, it is important that the system will easily adapt to those attacks.

In order to create a security system, which adapts to and detects new attacks, machine learning solutions are required. Machine learning algorithms can in general be considered under the following categories: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. Supervised learning methods can detect new attacks by analyzing old anomalies they are trained with initially. Unsupervised learning methods will enable the system to handle new attacks and learn from them. In reinforcement learning, the system learns from past actions and their consequences. The solutions described in this paper fall under the categories of supervised learning (long short-term memory recurrent neural networks), and reinforcement learning (deep reinforcement learning with Q-learning). In this thesis, both solutions proposed use neural networks. The main difference is that the LSTM-RNN solution puts neural networks in the center, whereas the DRL solution benefits from neural networks as a part of its solution. This combination makes the reinforcement learning solution a hybrid of reinforcement learning and supervised learning.

At first, LSTM-RNN solution seemed to be effective as it runs on time series data. On the other hand, the deep reinforcement learning solution was also promising as it was more suitable to the zero-day attack detection aspect of the problem. Both solutions are compared using the same datasets in different combinations in order to prove our initial perspective.

1.4. Outline

The remainder of this thesis is organized as follows: Chapter 2 provides an overview of related work in intrusion detection systems. The chapter is divided into three subsections about intrusion in cloud networks, big data for IDS, and machine learning solutions in cybersecurity. Chapter 3 elaborates on the two ML solutions proposed in this thesis, DRL and LSTM-RNN. Experiments, datasets, and evaluation metrics are described in Chapter 4. Finally, the thesis is concluded, and future work targets are set in the conclusion chapter, Chapter 5.

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1. Intrusion Detection in Cloud Networks

Network intrusion detection has been a well-studied topic with many different approaches proposed along the years. While some of these solutions can also be applied in a cloud environment, because of the different characteristics of the cloud environments and significantly larger network traffic volumes, several new approaches have been suggested to solve the problem of intrusion detection in the cloud. In a cloud environment, intrusion detection monitors can be deployed at various locations. Maiero and Miculan (2011) suggested that with the usage of virtualization techniques, intrusion detection monitors can be deployed in a guest VM or in virtual machine monitors. Beyond host or network device monitoring, distributed collaborative monitoring approaches are also utilized to catch system-wide attacks as described by Bharadwaja et al. (2010). In these types of solutions, infrastructure problems need to be solved, since the system must support massive amounts of data gathered from different monitors and these data must be processed quickly to detect attacks as soon as possible.

Deshpande et al. (2014) proposed an intrusion detection model for cloud environments. The model consists of a data logging module, a preprocessing module, an analysis and decision engine, and a management module. Logs are obtained using the Linux audit framework. After logs are ready for processing, a k-nearest neighbor classifier comes into play and decides if there is an anomaly or not. Different than the solution in this paper, system calls are used for detecting anomalies.

2.2. Big Data Approaches for IDS

Enterprises collect terabytes of data related to security. Large enterprises record 10 to 100 billion events daily. These numbers will only increase as the enterprises hire new staff and use new devices. The situation gets worse as these enterprises start to use cloud architectures as depicted by Mcdaniel et al. (2013).

Big data analytics is the key to analyze and process this big information. For this purpose, in 2012, Cloud Security Alliance formed a group called Big Data Working Group. This group's (2013) last report "Big Data Analytics for Security Intelligence" focuses on big data's role in security. It is foreseen that performing detailed analytics on the data and getting real-time analysis on them is possible today thanks to Big Data solutions.

Before the latest advancements, collecting big data was not logical in terms of cost. Hence the logs and events were being deleted in specific periods. With the help of the Hadoop framework and new big data tools, deploying big data became easier. Querying dirty, inconsistent and large-in-volume data was not effective. Now, big data applications help to clean, prepare and query the big data in an effective way.

It should be noted that data centers will be a target for data thefts. Therefore, their protection is important. Also, the source of the data and its validity should be checked while acquiring the big data. Analysis of wrong data will create wrong results. Furthermore, the results will be observed by people, hence it will be better to visualize the data; which means improvements in human-computer interaction are needed.

Although big data has changed security understanding in SIEMs (Security Information and Event Management), it is not the solution for all the problems in security. Mcdaniel et al. (2013) emphasized that researchers should always continue to find new methods to prevent attacks. With big data, privacy infringement might be another issue. Computer scientists need to follow commonly agreed privacy guidelines.

Mahmood and Afzal (2013) stated that threat detection and monitoring is the largest field in security analytics for financial and defense institutions. Big data analytics' help in this area is that, it can predict and detect malicious or dangerous network traffic patterns as well as unusual user behaviors. Additionally, it will help unveil sudden changes -which typically are suspicious incidents- in network servers.

In implementing a big data security analytics solution to a corporation, firstly, a security analytics business strategy must be prepared. Also, C-level executives must be aware of the benefits and must understand the technical basis. Corporations should build a platform in which they can experiment with the data, using big data analytics tools and techniques. It may also happen to be a necessity to hire experienced personnel to be consulted in data science related issues. Furthermore, there should be layers which have to run 24/7. For instance, a network monitoring layer should help system designers to monitor network streams lively. Also, a live layer for alerting suspicions will serve ensuring cybersecurity.

Recent works have proposed using big data processing approaches to solve the problem of intrusion detection in cloud environments. One of these solutions was introduced by Casas et al. (2017). They ıdeveloped a system called Big-DAMA, which utilized Apache Spark for both batch data processing and streaming data processing. Then, they combined their solution with five different supervised machine learning algorithms.

To detect a possible attack using intrusion detection systems (IDS), Mishra et al. (2017) stated that basically two techniques can be used: In misuse detection, the IDS knows about previous attack patterns and tries to catch an attack by comparing the collected data with previous patterns. In anomaly detection, the IDS does not know about any previous attacks and tries to find anomalies in the network data, which could be possible signs of attacks. In recent years, machine learning approaches have been used successfully for both of these techniques.

2.3. Machine Learning Solutions for Intrusion Detection

With the advancements in machine learning in recent years, most of the anomaly-based intrusion detection systems have started benefiting from machine learning algorithms. One of the successful solutions, Beehive, was introduced by Yen et al. (2013). In their solution, they used logs to detect network intrusions. They separated their features into four categories:

- Destination-Based Features: Connections to uncommon destinations might indicate suspicious behavior. New destinations are the first destination-based feature. If a destination has never been contacted in an observation period, it is new. The problem here is there are lots of new destinations, which belong to cloud services or popular services. Therefore, a whitelist is used to increase performance.
- Host-Based Features: Hosts installing new and potentially unauthorized software indicate suspicious activity. The software configurations on a host are inferred from the user-agent strings included in HTTP request headers. These strings include the details of the application making a request. The number of these new strings might signal a potential threat.
- Policy-Based Features: For a host, the number of blocked, challenged or consented domains that are contacted by the host are counted.
- Traffic-Based Features: Sudden spikes in a host's traffic volume might be an indication of a threat. A spike is defined when a host generates more connections than a threshold.

By using these four different feature types, they have been able to apply an unsupervised learning algorithm, k-means clustering, to detect suspicious activities. Although the Beehive solution is simple yet effective, it does not work in real-time, whereas the solution described in this paper works near real-time.

Various solutions suggested by different authors use k-means clustering. An example of such a solution is described by Razaq et al. (2016). The main problem with these

solutions is the need of predefining *k.* Predefining *k* makes the solution a supervised one, whereas it would be beneficial to keep it unsupervised when using k-means clustering.

A combination of k-means clustering and K-Nearest Neighbor was proposed by Sharifi et al. (2015). They first applied k-means clustering to define clusters and their centers. The clustering process is applied multiple times in order to achieve the best structure. Then this structure is used to classify the data using KNN. Their solution is somewhat similar to Razaq et al. (2016)'s solution. Rather than tweaking k-means like them, they combined it with KNN. Their overall accuracy was around %90, which should be improved in order to establish a secure system.

Another combination solution including KNN and decision trees is suggested by Balogun and Jimoh (2015). This time, decision trees come into play first to create node information depending on the rules of the resulting decision tree. This information is added to the original dataset. Finally, KNN does the rest and classifies the data. Their solution could detect new attacks (attacks not included in the training set) with remarkable accuracy.

Hariharan et al. (2019) proposed a solution with a similar structure to the one described in this thesis. They retrieved the data using Elasticsearch (2019). After the retrieval part, several machine learning algorithms (Isolation Forest, Histogram Based Outlier Score, Cluster-Based Local Outlier Factor, and k-Means Clustering) are run on the data. Resulting anomalies are reported to the system administrators. Their solution, CAMLPAD, had a %95 accuracy, which was promising.

One of the solutions utilizing Support Vector Machines (SVM) was proposed by Pervez and Farid (2014). Their algorithm was a filtering algorithm tested on the NSL-KDD dataset for intrusion classification tasks. Although their approach performed well in training sets, in the test sets it failed to detect network intrusions which the system had not seen before.

Recent years have shown many promising results of applying deep learning methods to machine learning problems and intrusion detection is not an exception for this case. Kim and Kim (2016) and also Chuan-long et al. (2017) proposed applying recurrent neural networks to intrusion detection systems and got very promising results. These works only show that RNN could be used while detecting anomalies in related data and they do not propose a complete end-to-end intrusion detection system. The approach described in this paper differs from these previous approaches in that it attempts to build a self-healing cloud system through deep learning with recurrent neural networks, which integrates time dependencies between observations (data points) in the system into the learning process to provide a more accurate representation of the attack progression and normal system processes.

Another deep learning solution is proposed by Behera et al. (2018), which is implemented using convolutional neural networks (CNN). In CNN, there are neurons with learnable weights and biases. CNN has five types of layers, namely, input layer, convolution layer, rectified linear unit, pooling layer, output layer. Different than standard neural networks, the convolution layer uses dot product of weights and local regions to calculate inputs for the next layer. Rectified linear unit is used for better gradient propagation and effective processing. The authors had successful results with their experimentations using the NSL-KDD dataset. Their solution proves the usability of deep learning for network intrusion detection. The solution proposed in this thesis combines deep learning with reinforcement learning for creating a system, which can adapt for zero-day attacks.

There are several solutions using reinforcement learning to detect network intrusions. Deokar and Hazarnis (2012) used log files for their IDS solution. There are several types of log files: server-side log files, client-side log files, proxy-side log files, firewall-side log files, network-side log files, and system-side log files. In their proposed system, log files are converted to XML files by a Processing Unit (PU). After that, if a match with a known attack in the knowledge base is found, the attack gets reported. Otherwise, an association rule database decides whether there might be

signs of an attack or not. Finally, rules are updated according to results of this estimation. This solution, rather than using reinforcement learning as a basis, benefits from reinforcement learning as a side solution. The approach described in this paper, on the other hand, puts reinforcement learning in the heart of the solution.

Another solution using reinforcement learning is suggested by Servin and Kudenko (2008). In their solution, a multi-agent hierarchical architecture has been developed. Different sensor agents monitor different states of the network and pass short signals up in the hierarchy. Agents at the higher levels of the hierarchy, therefore, have a better view of the network. Rather than processing all the information and acting on them, these agents leave the local information processing to lower level agents. Finally, these higher-level agents learn whether they should alarm the system admin or not using the information provided by lower level agents. This multi-agent RL solution works accurate enough as implied by the authors, though it is not tested using different datasets. Additionally, the RL solution in this thesis includes a deep learning approach, which differs from the solution of the authors.

Elderman et al. (2016) applied reinforcement learning in a cyber security simulation to find out the best strategy for both the defender and attacker sides of a cyber-security simulation modeled as a Markov game. In their simulation, they tried out different techniques such as Monte Carlo learning, Q-learning and neural networks. Experiments held showed that Monte Carlo learning was the most effective one for both sides of the Markov game. Their work shows that RL can be used with different techniques for cybersecurity; also, it can be used as an attacking instrument, too.

2.4. Deep Reinforcement Learning Solutions in Different Fields

Deep Reinforcement Learning is being used in many different fields. Although it is especially common in AI solutions such as robots, game playing agents, there are various approaches implementing it for distinct purposes. Playing Atari is one of the classic examples, which is implemented by Mnih et al. (2013). In their solution, similar to the solution in this thesis, a convolutional neural network is combined with

reinforcement learning. They have used a modified Q-learning algorithm to train the network. Cuayahuitl et al. (2015) implemented a DRL solution for playing a strategic board game (Settlers of Catan). Their solution had significant success over other random, rule-based or supervised-based solutions. Giraffe, a chess engine developed by Lai (2015), implements deep reinforcement learning to play chess. MathDQN, proposed by Wang et al. (2018), used DRL to solve arithmetic word problems. Again, similarly, they have used a two-layer feed-forward neural network in order to find out the potential Q-value.

An example usage of DRL in the field of medicine is suggested by Nemati et al. (2016). In the solution, DRL has been used for setting medication doses optimally in order to provide the best treatment for the patients. Likewise, chemical reactions are optimized with DRL in the solution described by Zhou et al. (2017).

DRL is used in the biology field as mentioned by Mahmud et al. (2018) in their paper. It is being used to extract features from biological sequence data (DNA, RNA, and amino acids) and perform predictions on them. Also, it is mentioned that DRL is used for bioimaging as well for pixel-level, cell-level and tissue-level analyses. Additionally, it is stated that DRL is implemented in many medical imaging applications for analyzing medical images obtained from different scans (MRI, CT, PET etc.).

CHAPTER 3

PROPOSED APPROACH

3.1. Overview

In this work, we propose an intrusion detection system that works with real-time data analytics to detect possible attacks and develop a resilience mechanism through deep learning with recurrent neural networks.

The solution involves the collection of system metrics from the network and near real-time processing of those metrics using big data analytics to discover anomalies. Metric collection is done by metric collection agents deployed in related parties like guest VMs. These data include network packets and other related metrics like VM usage metrics, HTTP server performance etc. After collection, these metrics are sent as a stream to a stream processing engine. The stream processing engine gathers the metrics inside the stream, considering their timestamps and processes these data by feeding them to a recurrent neural network trained previously. If the network finds an anomaly in the data, it labels it and triggers an alarm to inform the system administrators. The details of these steps are given in the following sections.

3.2. Metric Collection

Popular cloud system providers such as AWS share the statistics and state of their cloud systems through an API. These statistics contain utilization of CPU, RAM, disks, number of packets/bytes received/transmitted, and many other details about the current state of the system. In this work, we utilize guest VM agents for metric collection, since this approach does not depend on the cloud infrastructure and is more flexible than virtual machine monitor solutions. At the metric collection phase, the agents collect the required metrics from the guest VM like network flow, basic system usage metrics such as CPU utilization, disk read/write metrics etc. and usage metrics

of applications that can affect the system performance. The metric collection agent has two components, the producer and the consumer. The producer component gathers the system and application metrics from the VM using different interfaces. To achieve this, the producer must have a pluggable architecture that written plug-ins can gather the metrics from, knowing how to get them. The responsibility of the consumer side is to gather metric records from the producer and pass them onto the processing phase.

3.3. Metric Processing

Due to the large volume and velocity of the data collected from the systems, big data processing frameworks are needed to analyze the data. Big data can be processed as batches or as streams. Deciding which type of processing is needed is up to the task. Shahrivari (2014) stated that the standard MapReduce model and its implementations are totally focused on batch processing. Therefore, before any computation starts all the input data have to be available. Yet, recent applications have more stream-like demands. Additionally, applications might be needed to run continuously, as in the example of a query that catches special anomalies from ongoing system events. If we handle the data as batches, we need to wait for some amount of time to create batches from the given data. After the data become batch, the processing starts. This contradicts with our purpose of near real-time detection in this work, as we need to act in real time in order to prevent or stop attacks before they can harm the cloud system. Stream processing on the other hand involves handling the data in memory as they arrive.

Before starting the work, different frameworks (Storm, Spark, Flink) were compared with each other for handling streaming data. Ellingwood (2016) described the pros and cons of these frameworks. Storm provides near real-time processing, is scalable, and fault tolerant. On the other hand, it is not stateful, unless used with Trident – which increases the latency. Flink has higher throughput than Storm, allows SQL-style querying, is more high-level, and has machine learning libraries. Flink is a younger framework, therefore it cannot be said that it has been tested widely. Therefore,

tutorials and example solutions are insufficient compared to other frameworks. Spark is designed for machine learning, caches datasets in memory, and is successful for varied processing workloads.

In this work, Apache Spark (Apache, 2019) has been used to process the stream data collected from systems. Spark has advantages like fault-tolerance, in-memory computation, being faster than similar frameworks, having a wider community, multiple language support etc. Shahrivari (2014) stated that main memory is at least 50 times faster than hard disk in terms of bandwidth. Latency is, likewise, much lower when using memory (nanoseconds vs milliseconds). Spark, having in-memory computing, is crucial for this work. The data that streams from our network are handled by Spark and served to our algorithm in order to detect possible attacks. Multiple networks can be watched by using this framework. In similar problems, it is seen that Spark is one of the most popular choices for streaming big data, like in Gupta and Rani's (2018) zero-day malware detection framework.

To support stream processing, many tools are available to specifically handle the requirements of this process. Tools like Apache Kafka (Apache, 2019) and Amazon Kinesis (Amazon, 2019) provide great support for handling stream data in a scalable way. In this solution, Apache Kafka is used to collect the metrics from the guest VM agents and pipe them to the stream processing engine.

For testing the solutions proposed in this thesis, different datasets have been used. The data in these datasets are simulated as if they are streaming from a network. This simulation is performed by involving Apache Spark and Apache Kafka. Apache Kafka is a distributed streaming platform. It is used for reading and writing streams of data. Normally, it can be used with different applications, database management systems or stream processors. In this work, intrusion detection datasets are passed to Kafka to simulate them as if they are the real source of network data. How Kafka will handle the datasets is up to configurations. Different configurations were run in order to test

the processing power of Spark. The results below are produced by using a part of the UNSW-NB15 dataset as source data.

Table 3.1. *Process Time Measurements in Streaming Data Simulation*

Batch Size	Sleep Between Batches in Seconds	Elapsed Time in Seconds	Real Elapsed Time in Seconds (w/o sleep)
700000	0	5,29	5,29
350000	0,5	8,63	8,13
175000	0,5	12,82	11,32
100000	0,5	19,66	16,66
50000	0,5	34,58	28,08
10000	0,5	153,81	119,31

Although it can be seen in the table that Spark works in real-time with streaming data in the test with the whole set (700000 records), to replicate a real-life network situation, additional tests are run by slicing the set with different sizes and passing them to Kafka, waiting half a second in between. The experiments show that there is an overhead in preprocess and postprocess parts, which can be improved to minimize the time difference between tests. In the slowest scenario, the whole set is processed under two and a half minutes including wait times, which is still acceptable, but experiments have proven that Spark can handle much larger data flows easily. Spark provides a monitoring tool, Spark UI, which can be used in order to benchmark the process. According to benchmark results, configurations can be tweaked in order to find the most efficient solution related to the network itself. Configurations of networks with different sizes of data flow can be different, therefore network specific configurations might be needed.

3.4. RNN-based Learning for Intrusion Detection

Signature-based intrusion detection systems rely on detailed information about previously observed attacks. These approaches fail in the case of cloud systems, which are open to attacks that might be novel. On the other hand, unsupervised learning methods enable us to prevent or at least detect changes in the system parameters, i.e. the normal behavior of the system. By this way, the system will be able to detect anomalies and will try to prevent if there is an attack going on. In the mean time, alarms will be created in the system so that if the security system cannot stop the attack, it will warn the user/owner of the cloud system. This is actually the main difference from a signature-based intrusion detection system. If this type of system does not have any information about an attack, it will most likely be missed. On the other hand, for a system with an unsupervised learning algorithm, even a minor anomaly might cause the system to detect if something is wrong. When run on isolated data points/cloud activity logs, unsupervised algorithms may not achieve very high accuracy due to noise in the data. For instance, observation of a sudden spike in CPU utilization might signal a possible attack even if it was caused by a legitimate process and does not persist for a long period, not causing any degradation in the performance of the system. Precisely for this reason, we need to be able to model the time-based behavior of the system by considering the data points collectively as a time series rather than isolated incidents.

Recent advances in deep neural networks have made it an effective tool for many supervised and unsupervised learning tasks, achieving higher accuracy than competing approaches. Recurrent neural networks (RNN) are machine learning models consisting of nodes that are connected to each other. These nodes can memorize and pass information in a sequence, though they process the data items one by one. Therefore, they can handle inputs and outputs that are dependent on each other. As stated in Lipton et al.'s (2015) paper, RNNs have been successful in various tasks such as image captioning, speech synthesis, time series prediction, video analysis, controlling a robot, translating natural language and music generation.

Normally, there is only one single network layer in a node of a classic RNN. In conventional neural networks, it is not defined how the network will remember events of the past to use the information about them in the future. Recurrent neural networks aim to solve this issue by using the architecture depicted in Figure 3.1:

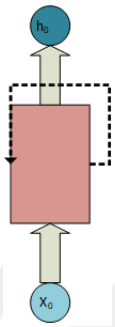


Figure 3.1. An RNN Loop

As shown in the diagram, the network gets an input *x*, processes it, and outputs an output *h*. The outcome of the process is used in the next step. To make it clear, the loop is demonstrated in an open form in Figure 3.2:

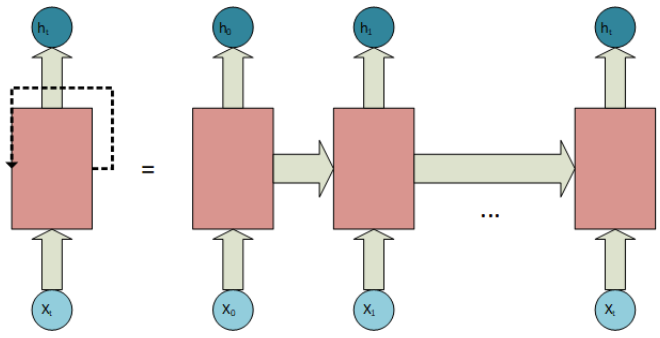


Figure 3.2. An Unrolled RNN Loop

The equation below represents the network mathematically:

$$h_t = \theta(W_{x_t} + Uh_{t-1})$$

Here *W* stands for the weight matrix, which is multiplied by the input of the current time. The result is added to the multiplication of the output (hidden state) of the previous time step and its own hidden state and the hidden state matrix (transition matrix) *U*. As Nicholson (2018) describes, these weight matrices are used to define how much of the information both from the current input and past output will be used to determine the current output. If they generate an error, it will be used to update the weights to minimize error. The resulting sum is condensed by the hyperbolic tangent function θ.

Some examples of this standard RNN architecture include predicting the next character in a series of letters, picking the next note after a sequence of notes of a song, deciding where to go when controlling the motion of a robot etc. In our case, we use RNN in order to predict an intrusion, but we use LSTM-RNN because of the reasons that will be explained later in this section.

LSTM stands for Long Short-Term Memory. Without it, gradients that are computed in training might get closer to zero (in case of multiplying values between zero and one) or overflow (in case of multiplying large values). In other words, as the time sequences grow, RNN might not connect older inputs to the outputs. LSTM adds additional gates to the architecture to control the cell state. By this modification, training over long sequences is not a problem anymore.

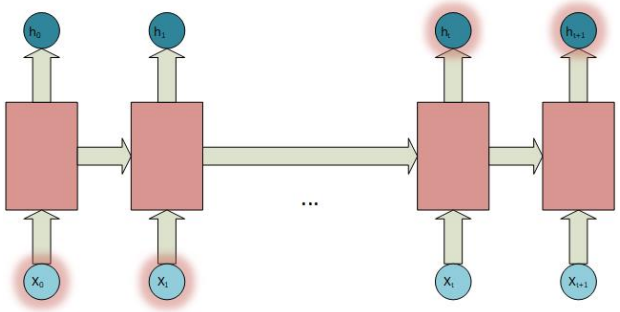


Figure 3.3. Disconnected dependencies in RNN

In an LSTM-RNN there are four layers, which interact with each other. First of all, the input is received and copies itself into four. The first one goes into a sigmoid layer. This layer decides whether the output of the previous layer is needed and should be used, or it should be thrown away. Then another sigmoid layer decides which values are going to be updated. A tanh layer generates possible values, which might be included in the state. These two layers get combined to update the state. Finally, another sigmoid layer picks what we are going to output from our cell state.

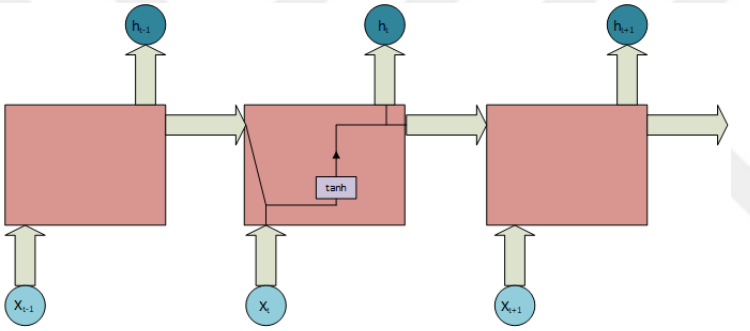


Figure 3.4. Single layered structure of standard RNN

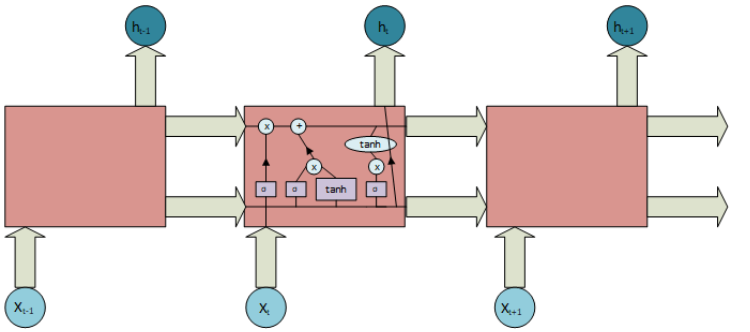


Figure 3.5. Four layered structure of LSTM-RNN

In the proposed model, we utilize the LSTM recurrent neural networks (RNN) algorithm, which is described by Hochreiter and Schmidhuber's (1997) in detail, to detect deviations from the normal behavior of the network system under monitoring. Note that because of the nature of the algorithm, it first needs to learn the normal state of the system. By processing the normal state, the system will detect anomalies when metric values that deviate significantly from the normal behavior of the system are observed. In RNNs, inputs are not independent, every time sequence uses information from the previous ones. This feature perfectly suits our task, as we cannot directly specify if there is an anomaly without analyzing the system's state for the time being.

The algorithm receives parameters of the system from Spark and uses those parameters as a time series input. The parameters indicate the state of the system's properties for that time series. The algorithm then serves these parameters to its prediction function. The prediction function tries to find out if there is an anomaly in the system. For example, if there is an unrealistic peak in the CPU utilization and number of disk operations and incoming network packets, this might indicate that the system is under a denial of service attack. From this point, the system can create an alarm to warn system administrators or initiate a security action.

We have used LSTM-RNN in Tensorflow. LSTM is actually handled by Tensorflow itself, but we needed to convert some of the fields in data as we could not pass them. For example, fields like IP addresses, protocol types, service types etc. converted to data types that LSTM-RNN accepts, as strings are not accepted. After processing of LSTM-RNN is finished, we check if there was an attack. There was only one output for our experiment, which is the actual result: whether there was an attack (1) or not (0). How LSTM-RNN works in general is described by Olah (2015) and explained below step by step:

1. The first layer gets current input and output of the past time series, then decides if the previous information is needed now. Actually, this layer can be called

the *forget* layer. h stands for the output of the past, x stands for the current input, W is the weight of this layer, and b is the bias.

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f)$$

2. Then we move onto the input layer. This layer is another sigmoid layer, which decides the values that are going to be updated.

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i)$$

3. A hyperbolic tangent layer creates candidate values, which might be included in the cell state. Cell state is a straight line in our network that flows for the entire network. LSTM changes information on this state across the road with the help of the gates.

$$cdt_t = \tanh(W_{cdt}.[h_{t-1}, x_t] + b_{cdt})$$

4. Results of all previous steps are combined in order to create an update to the cell state.

$$c_t = f_t * c_{t-1} + i_t * cdt_t$$

5. Finally, the output is decided. Naturally, the cell state is used in deciding. Another sigmoid layer takes part, and its output is multiplied the by cell state (state will go into tanh first).

$$h_t = \sigma(W_o.[h_{t-1}, x_t] + b_o) * \tanh(c_t)$$

We proposed this LSTM-RNN solution for intrusion detection under the name "A Big Data Analytical Approach to Cloud Intrusion Detection" to CLOUD2018 conference held in Seattle in June 2018. It is accepted in "Application and Industry Track: Cloud Data Processing" and presented by Assist. Prof. Dr. Pelin Angın at the conference.

3.5. Deep Reinforcement Learning Based Intrusion Detection

Reinforcement Learning (RL) is a machine learning approach built on rewards and punishments. RL agents make their decision by checking the state they are in and the available actions on the present state. Every decision ends up with a reward or punishment (negative reward). These rewards shape the future decisions. Reinforcement learning, therefore, is constructed on state-action pairs with resulting positive or negative rewards in an environment. An agent is connected to its

environment with action and recognition in the classic reinforcement-learning model as stated by Kaelbing et al. (1997). In every step of interaction, the agent gets the indication of the current state of the environment. After that the agent picks an action, which generates an output. With every action the state altered, the value of the alteration is sent to the agent via a scalar reinforcement signal. The agent should behave and choose actions, which will increase the sum of values of the signal in the long run. The agent will learn to do this over time by trial and error, guided by various algorithms.

The agent aims to find a policy in which states and actions are mapped to each other and maximize the long-run measure of reinforcement. It is expected that the environment will be non-deterministic, which means taking the same action in the same state might create different results.

Reinforcement learning differs from supervised learning. The most important difference is rather than presenting input/output pairs, the agent is informed with the immediate reward and resulting state after the action; but note that the agent is not informed about which action would give the best outcome in the long-term. Another difference is that on-line performance is important, the evaluation of the system is simultaneous with learning.

Intrusion detection using Deep Reinforcement Learning (DRL), same as LSTM-RNN, depends on learning, unlike signature-based systems, which makes the system safer when it meets zero-day attacks. DRL is different from LSTM-RNN: it does not require a long training session beforehand. This characteristic enables the system to be ready after training itself in the short term, but still most benefits are seen in the long term.

In our proposed DRL system, there are two different states and four different actions. These states, actions and their related rewards and punishments are given in the below table:

Table 3.2. *State, Action and Reward Table for RL*

State	*Action*	*Reward*
Normal	No Alarm	+1
Normal	Alarm	-1
Attack	Alarm	+1
Attack	No Alarm	-1

Deep Reinforcement Learning has been used in many different areas, some of which are mentioned in Chapter 2 of this thesis. Combined with reinforcement learning, deep neural networks can be useful for many real-life problems. Reinforcement learning becomes deep reinforcement learning when deep neural networks are used for function approximation in policy and value functions. In a reinforcement learning algorithm with Q-learning, the value function is described as below:

$$Q(s,a) = r(s) + \gamma \, max_{a'} \sum_{s'} P(s'|s,a)Q(s',a')$$

This equation is called Bellman Equation. *s* stands for state, *a* stands for action, *r* represents reward, and *P* stands for state change possibility. According to the equation, Q value of a state-action pair equals the sum of the current reward and potential future Q-values. In short, the exact reward is added to possible rewards. The equation is a discrete one. On the other hand, in most real-life applications actions and states are continuous. Therefore, for the value function, an effective function approximation method is required. Neural networks come into play for this need. In the value function, every state and Q-value are calculated by using hidden layers of neural networks in between. The neural networks are trained by using backpropagation. The algorithm described by Minh et al. (2013) is given below step by step:

Table 3.3. *Deep Q-learning Algorithm*

Step	*Detail*
1	Initialize replay memory *D* to capacity *N*

2 Initialize Q-function with random weights

3 **for** episode = 1, *M* **do**

4 Initialize neural network from a random state *s*

5 **for** t = 1, *T* **do**

6 Find Q-values for all actions using DNN:
$$a_t = max_a Q^*(s_t, a;\ \theta)$$

7 Choose an action a_t for current state s_t by using ε-greedy exploration

8 Get to the next state s_{t+1} with action a_t and pick the related reward r_t

9 Store transition (s_t, a_t, r_t, s_{t+1}) in *D*

10 Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) in *D*

11 Set $y_j =$
$$\begin{cases} r_j, & for\ terminal\ s_{j+1} \\ r_j + \gamma\ max_{a'} Q(s_{j+1}, a'; \theta), & for\ nonterminal\ s_{j+1} \end{cases}$$

12 Perform a gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$

13 **end for**

14 **end for**

As described in the algorithm, DNN steps in as a part of the RL, and makes it DRL. In reinforcement learning, future rewards for steps later are not valued as much as immediate rewards. DNNs completes and enhances Q-functions by taking future rewards into account when deciding which action to take next. Another benefit of using DNNs in reinforcement learning is reducing the number of interactions needed by using sampling, which in the end increases the overall performance and the data efficiency of the algorithm.

Deep reinforcement learning, same as LSTM-RNN, is run through TensorFlow in the solution. Although TensorFlow is suitable to work with DRL, some alterations on the inputs are required. Similarly, changes on some of the parameters affect the results. The unsupervised nature of DRL seemed as the main reason behind the changing results. This attribute of DRL makes it work in the short-term, though in the short-term the algorithm seems not to be working quite accurately, as it needs to train itself both in its neural networks part and in the RL part. Therefore, it becomes similar to the approach in a supervised algorithm, like LSTM-RNN.

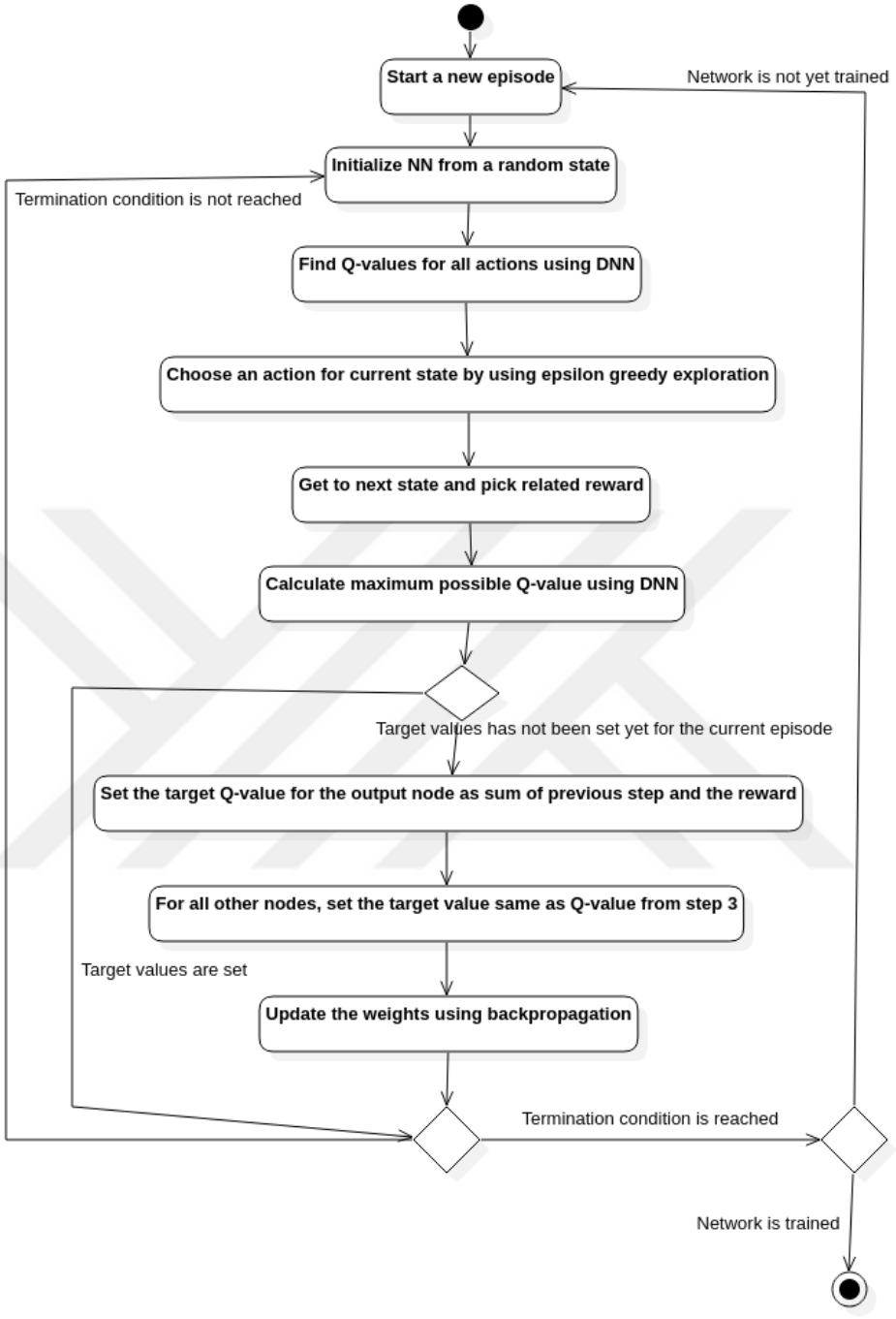


Figure 3.6. Activity Chart for DRL Algorithm

CHAPTER 4

EVALUATION

4.1. Evaluation Metrics

To measure the success of the solution, well-known metrics, namely accuracy, precision, recall, and F1, will be used. These metrics are defined Powers (2011) and are described in the table below:

Table 4.1. *Evaluation Metrics, Powers (2011)*

Name of the Metric	*Formula*
Accuracy	$\frac{True\ Positive + True\ Negative}{True\ N. + True\ P. + False\ Negative + False\ Positive}$
Precision	$\frac{True\ Positive}{True\ Positive + False\ Positive}$
Recall	$\frac{True\ Positive}{True\ Positive + False\ Negative}$
F1	$2 * \frac{Precision * Recall}{Precision + Recall}$

These metrics have been used for all the datasets. Note that only the labelled dataset of NSL-KDD in the LSTM-RNN experiment has been analyzed by using multi-class classification with a confusion matrix, whereas the other datasets have been analyzed by using binary classification. A confusion matrix is used when there could be two or more classes in the output. Every row of the matrix represents the predicted class, whereas every column of the matrix presents the actual class. The number of correct predictions for a class is seen in the intersection of the said class in the predicted row

and the actual column. False predictions are the sum of the other numbers in the same column. Overall accuracy calculation does not change in the confusion matrix, only precision and recall slightly change (calculated individually using only False Positive and False Negative numbers of their related column) as stated by Manliguez (2016).

4.2. Experiments with LSTM-RNN

As mentioned in the previous chapters, Apache Spark was used as the stream processing engine collecting data from machine instances. The figure below shows the flow of the data in the developed prototype. Note that the same structure has been used in Deep Reinforcement Learning version of the system:

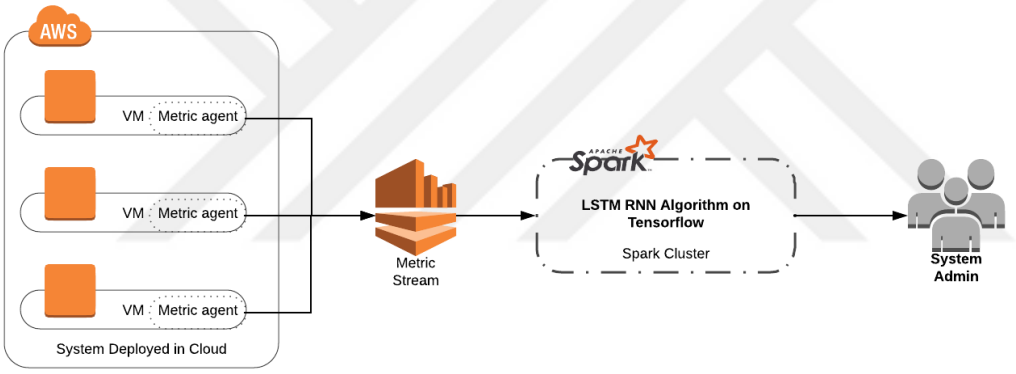


Figure 4.1. End-to-end Working of the LSTM-RNN Solution

4.2.1. Using UNSW-NB15 Dataset

Different datasets were used to test the LSTM-RNN solution. The first dataset used was the "UNSW-NB15" dataset of UNSW ADFA. This dataset seemed well-rounded as it contained two million logs, with a total of 49 fields per record. The fields in the dataset, such as IP addresses, ports, protocols, packet details etc., are given below in detail. Some of the fields needed to be normalized to make them suitable for the

LSTM-RNN solution. Likewise, some insignificant fields are discarded from the data. The records in this dataset were labelled as attack or non-attack.

The data has been simulated to make the system behave as if the data are streaming. As required by it, a training set is provided for LSTM-RNN, which has been run through Tensorflow, an open-source machine learning framework. The LSTM-RNN model on this framework, helps the system to decide whether a record could be an attack or not.

Different runs were performed on the system, using various sample sizes. In these tests, train and test samples had the same sizes. The sizes are given in the below chart with the related accuracy scores. The tests showed that as the sample size went bigger, the accuracy also got higher. The relation is depicted in the figure below. This result raised the question: Could an unsupervised algorithm or reinforcement learning algorithm be more suitable for this kind of task? This question is answered in the upcoming sections. The accuracy rates were between %87 and %91 on average, and the highest accuracy was %93.1.

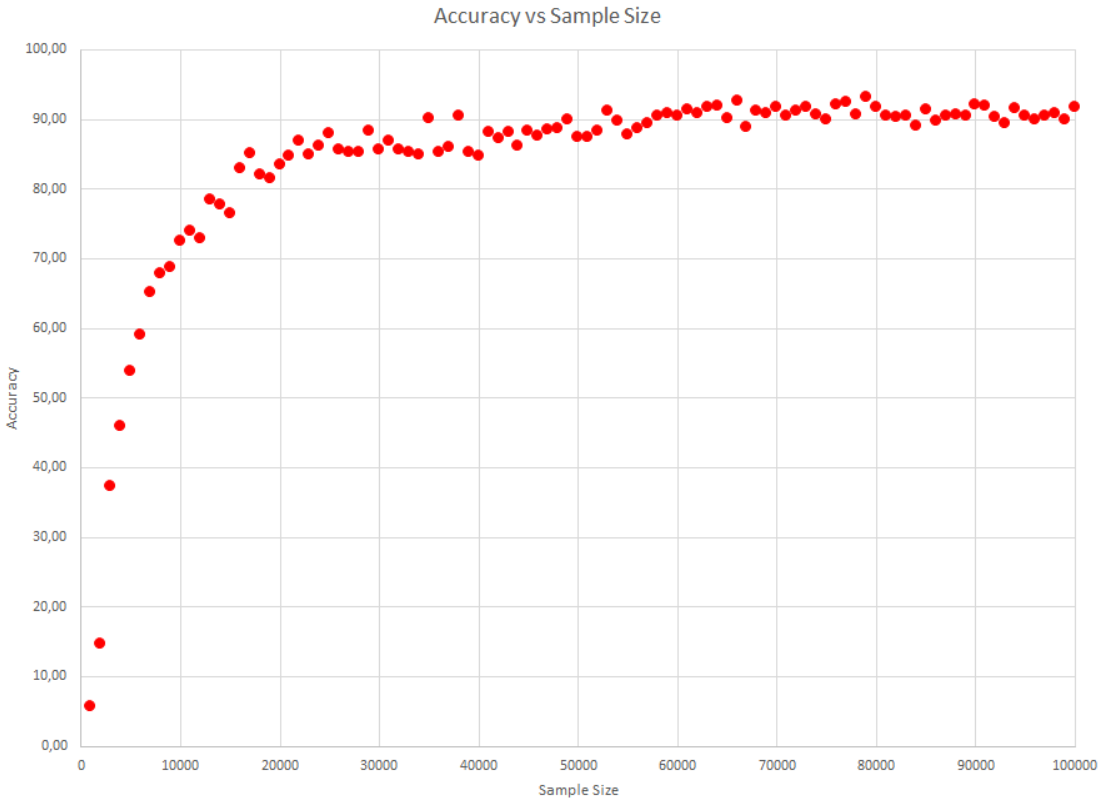


Figure 4.2. Accuracy vs Sample Size for LSTM-RNN with UNSW-NB15

Table 4.2. *UNSW-NB15 Features*

No.	*Name*	*Type*	*Description*
1	srcip	nominal	Source IP address
2	sport	integer	Source port number
3	dstip	nominal	Destination IP address
4	dsport	integer	Destination port number

5	proto	nominal	Transaction protocol
6	state	nominal	Indicates the state and its dependent protocol
7	dur	float	Total duration of the record
8	sbytes	integer	Source to destination transaction bytes
9	dbytes	integer	Destination to source transaction bytes
10	sttl	integer	Source to destination time to live value
11	dttl	integer	Destination to source time to live value
12	sloss	integer	Source packets retransmitted or dropped
13	dloss	integer	Destination packets retransmitted or dropped
14	service	nominal	e.g. http, ftp, smtp...
15	sload	float	Source bits per second
16	dload	float	Destination bits per second

17	spkts	integer	Source to destination packet count
18	dpkts	integer	Destination to source packet count
…	…	…	…
29	stime	timestamp	Record start time
30	ltime	timestamp	Record last time
…	…	…	…
49	label	binary	0 for normal 1 for attack

4.2.2. Using KDD Dataset

The KDD Cup 1999 Dataset is probably the most famous dataset in the network security field. It has been used as a go-to benchmark for IDS solutions. Although it is a 20-year-old dataset, and is missing some of the newest attacks, it is still popular among researchers. The dataset is quite comprehensive, there are five million records in it and each record has 41 features. Some of the important features are given in the below table:

Table 4.3. *KDD Dataset Features*

No.	*Name*	*Type*	*Description*

1	duration	integer	Record duration
2	protocol type	nominal	Type of the protocol (UDP, TCP…)
3	service	nominal	Destination service (ftp, telnet…)
4	flag	nominal	Status of connection
5	source bytes	integer	Source to destination number of bytes
6	destination bytes	integer	Destination to source number of bytes
7	land	binary	1 when source and destination addresses are the same land, 0 else
...	…	…	…
23	count	integer	Number of connections to the same host
24	srv count	integer	Number of connections to the same service
25	serror rate	float	% of connections with SYN errors
26	srv error rate	float	% of connections (service) with SYN errors
27	rerror rate	float	% of connections with REJ errors

28	srv rerror rate	float	% of connections (service) with REJ errors
...	…	…	…
42	label	nominal	“normal” for non-attacks, attack type for attacks

Like the UNSW-NB15 dataset, some features are removed or modified to adapt them for our LSTM-RNN solution. Results were similar with the said dataset. Although the F1 score of the best run is promising, we can see that the number of false negatives is dragging the accuracy lower.

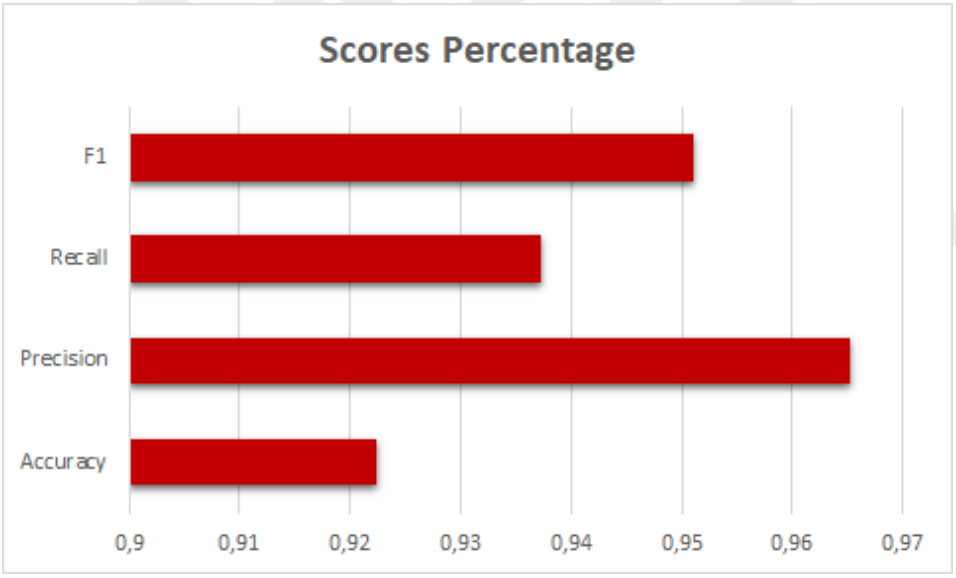


Figure 4.3. Performance results for LSTM-RNN using KDD

As represented in the related figure, the true positive rate among positives is quite high, but the number of false negatives is considerable, which hurts the accuracy in the end. Note that %10 percent of the datasets were used for both training (494.022 records) and testing (311.080 records).

4.2.3. Using NSL-KDD Dataset

NSL-KDD (n.d.) dataset is an improvement over the KDD Cup 1999 dataset. Redundant and duplicate records are eliminated in order to prevent bias for learners and classifiers. Also, the train and test sets are reasonable in terms of the number of records, which eliminates the need of selecting random portions out of the dataset. Additionally, it helps make the intrusion detection systems comparable over this dataset. It should be noted that all the fields are the same as the original dataset. The only change is the removed records.

In the dataset, different sets are given: Sets with binary labels (anomaly or normal), sets with attack-type labels and difficulty levels, sets without the hardest cases. The results are depicted in the figures below. Note that full datasets have been used for the experiments.

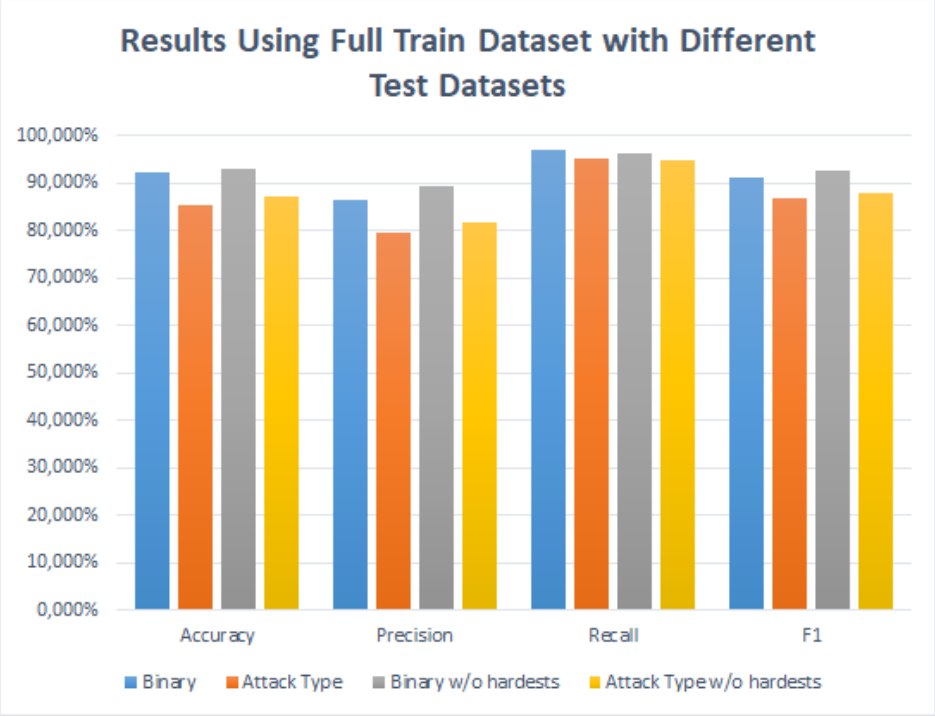


Figure 4.4. Performance results using full train dataset with different test datasets of NSL-KDD for LSTM-RNN

Accuracy slightly reduces when the system gets trained with 20% of the train set of the NSL-KDD datasets. Results of this experiment can be found in Figure 4.5 below:

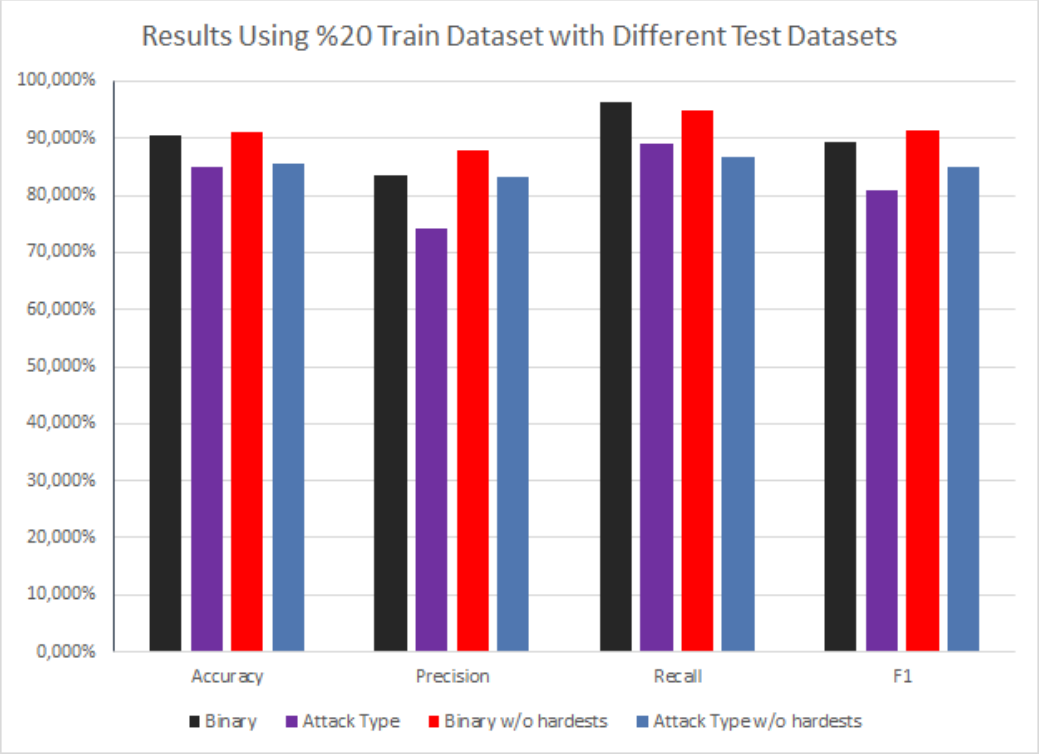


Figure 4.5. Performance results using %20 train dataset with different test datasets of NSL-KDD for LSTM-RNN

4.2.4. Using CICIDS2017 Dataset

As most of the datasets became unreliable and out of date, University of New Brunswick Canadian Institute for Cybersecurity (n.d.) created a new dataset containing up-to-date attacks with many features. The dataset is called CICIDS2017. The dataset includes both CSV and PCAP files of the network traffic. HTTP, HTTPS, FTP, SSH, e-mail protocols are used in the creation of this dataset. The dataset, in its CSV files for machine learning purposes, contains 79 features in total. It is stated by

Sharafaldin et al. (2018) that the dataset covered eleven criteria given in the below table; while none of the existing datasets was able to cover all these criteria.

Table 4.4. *Eleven Important Criteria for IDS Datasets (Sharafaldin, 2018)*

Name of the criteria	Detail
Complete Network Configuration	A complete network topology with different network elements and varied operating systems is used.
Complete Traffic	A user profiling agent is used as well as twelve different machines as victim in victim network and real attacks from attack network.
Labelled Dataset	Attacks are labelled accordingly.
Complete Interaction	Different interactions are covered: Internet communication, communication within and between two different networks on internal LAN.
Complete Capture	All traffic information is captured and stored.
Available Protocols	All commonly used protocols are available.
Attack Diversity	The most common attacks of 2016 are available in the dataset.
Heterogeneity	Network traffic is captured from the main switch and from all victim machines.
Feature Set	More than 80 network flow features are included.
MetaData	Detailed structure of the attacks is given in the dataset and explained in the published paper of the dataset.

In creation of this dataset, the network was under the traffic for five days. There are eight files in total, all the files represent a unique day and network traffic pair. There are 3119345 records in these files. According to the analysis of Panigrahi and Borah (2018), the dataset contains %83.34 benign (non-attack) records; the remaining records are attack records of fourteen different attack types. The authors mentioned that the high percentage of benign records causes an imbalance and tried to solve this imbalance relabeling the dataset by merging some of the attack types. Note that this improvement was not applied to the experiments in this thesis.

Table 4.5. *Dataset details of CICIDS2017*

Class Labels	Number of instances	Containing Datasets
Benign	2359087	All days
DoS Hulk	231072	Wednesday
PortScan	158930	Friday Afternoon PortScan
DDoS	41835	Friday Afternoon DDos
DoS GoldenEye	10293	Wednesday
FTP-Patator	7938	Tuesday
SSH-Patator	5897	Tuesday
DoS slowloris	5796	Wednesday
DoS Slowhttptest	5499	Wednesday
Bot	1966	Friday Morning
Web Attack – Brute Force	1507	Thursday Morning
Web Attack – XSS	652	Thursday Morning
Infiltration	36	Thursday Afternoon
Web Attack – SQL Injection	21	Thursday Morning
Heartbleed	11	Wednesday

As the Monday records only contained benign ones, most of the records (half a million) in the beginning of this file are skipped. Other than Monday records, the datasets have been used fully. Records of the other days are used in tests individually, as they contained completely different attack labels. The results for each day's experiment are given in the below figures.

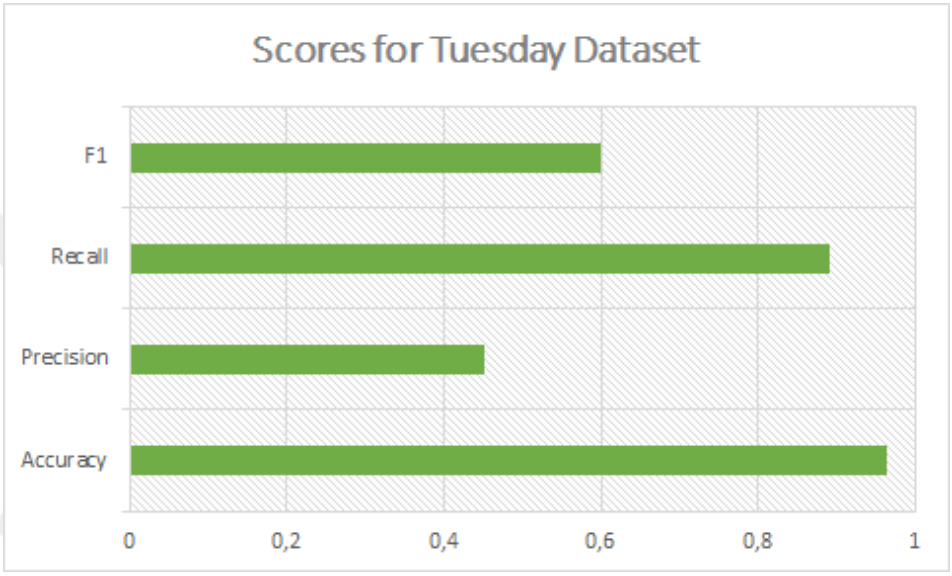


Figure 4.6. Performance results for Tuesday Dataset of CICIDS2017

Precision and F1 scores are quite low compared to higher scores in accuracy. This result is expected because of the density of benign records. As the number of benign records is nearly 40 times more than the attack records, the number of false positives is relatively high, which causes the low scores in precision and F1.

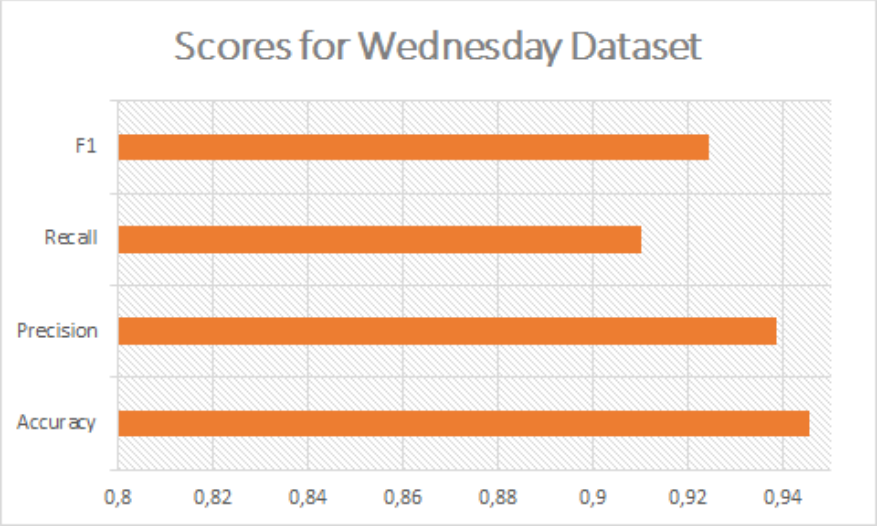


Figure 4.7. Performance results for Wednesday Dataset of CICIDS2017

This time, the strange pattern has not emerged as there is a better diversity in the records. There are nearly 450000 benign records in the Wednesday dataset, whereas the number of attack records are a little more than 250000. The only problem with this dataset is the number of Heartbleed attacks. There are only eleven heartbleed attacks, which is %0.00039 of the whole dataset.

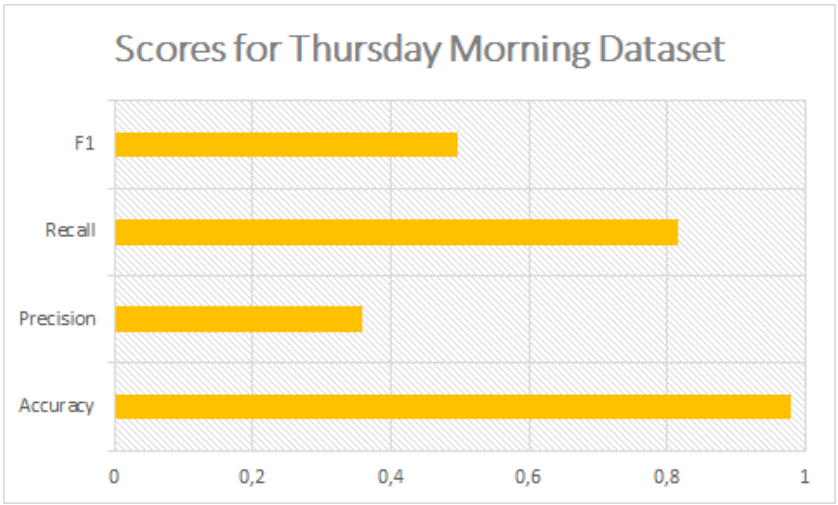


Figure 4.8. Performance results for Thursday Morning Dataset of CICIDS2017

The same problem with the Tuesday dataset is seen here. In the Thursday morning dataset, the density of benign records gets even higher: 80 times more than attack records. This increase can be seen in the figure as the decrease of precision.

The problem with the dataset becomes critical in the Thursday afternoon file. As there are only 36 attack records, accuracy is close to %100 whereas precision is close to %0. This pattern is expected and gets fixed when evaluating the day files as one dataset by merging them together. Therefore, results for that dataset is not shown here.

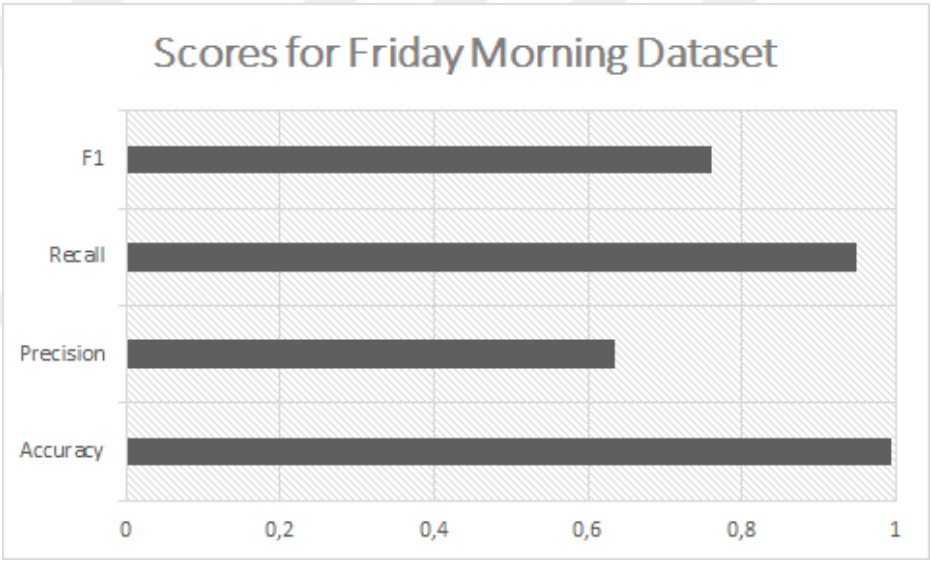


Figure 4.9. Performance results for Friday Morning Dataset of CICIDS2017

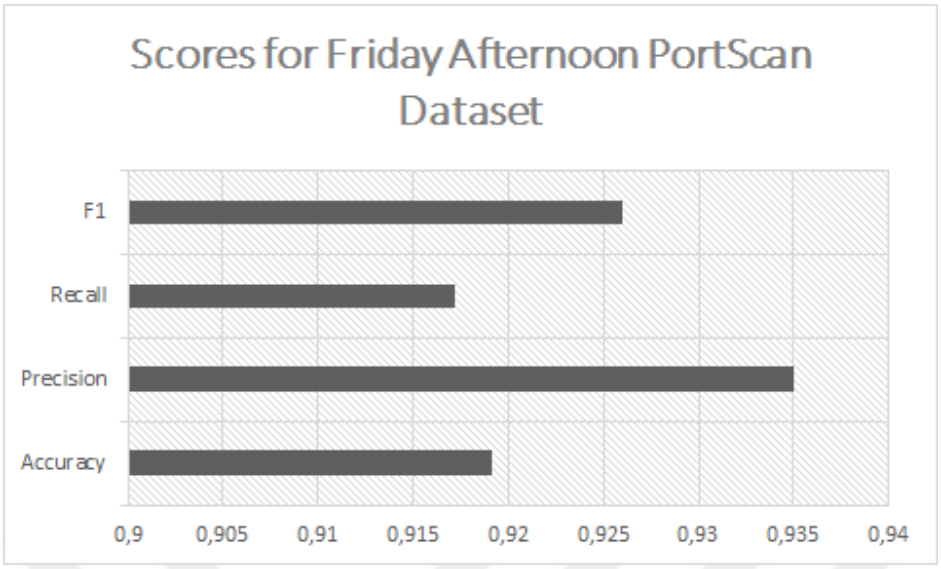


Figure 4.10. Performance results for Friday Afternoon PortScan Dataset of CICIDS2017

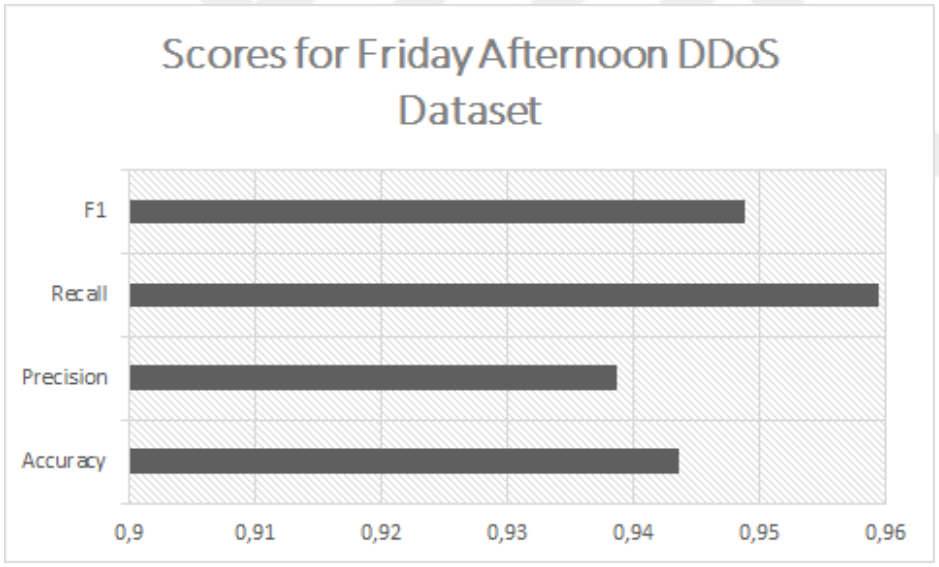


Figure 4.11. Performance results for Friday Afternoon DDoS Dataset of CICIDS2017

Friday results are more satisfying as the data is distributed in a balanced manner in the Friday datasets. A final experiment on the CICIDS2017 dataset by using all files as a whole produced the result below:

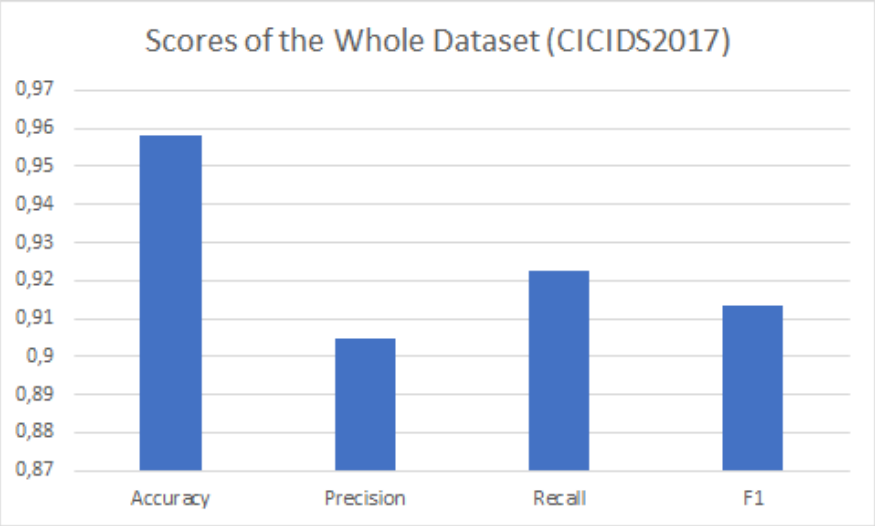


Figure 4.12. Performance results of the Whole Dataset (CICIDS2017)

Final results on the CICIDS2017 dataset, with the accuracy close to %96, were promising. As a future work, the dataset could be tweaked in order to eliminate imbalances.

4.3. Experiments with Deep Reinforcement Learning

4.3.1. Using NSL-KDD Dataset

The system has been tested in a Gym environment for the NSL-KDD dataset, which is prepared by Koduvely (2018), using OpenAI's Gym (OpenAI, 2019). Gym helps to test and compare reinforcement learning algorithms. Full datasets have been used during experiments. While doing the experiments, it is seen that increasing the training cycles of the neural network resulted in better outcomes in terms of accuracy. This, in fact, converts the unsupervised system into a supervised system in a way. Still, we have a chance to reuse the trained model later, without the need of training again. Figure 4.14 below depicts the differences in precision, recall and accuracy related to the number of training cycles:

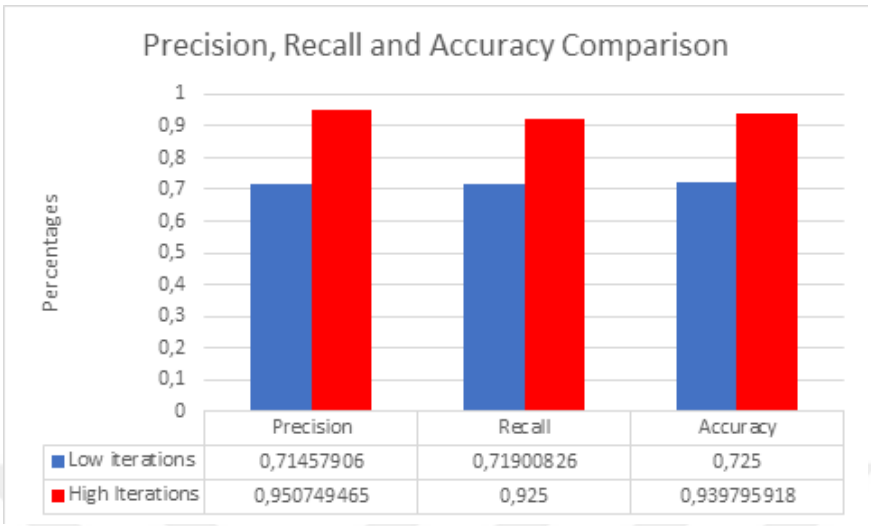


Figure 4.13. Precision, Recall and Accuracy values related to iterations in DRL solution using NSL-KDD

The need for using high iterations was obvious even before the last experiment. The same thing cannot be said when experimenting with the number of hidden nodes. There are different approaches about finding the suitable number of hidden nodes. Some trial-error experiments have been held in our case in order to find the best result. The figures below depict and compare different configurations with each other.

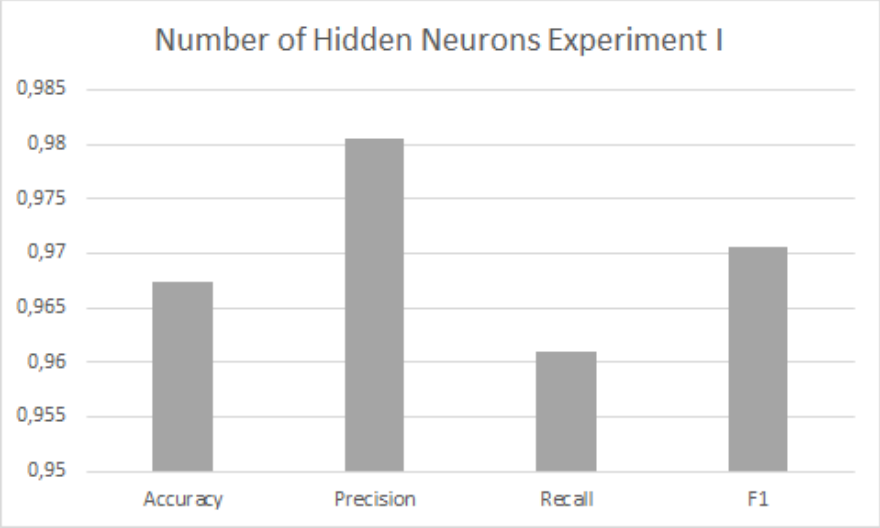


Figure 4.14. Number of Hidden Neurons Experiment I (Using NSL-KDD)

In the first experiment, the number of hidden neurons was set as 2/3 of the input layer's size. The results were satisfying with an accuracy close to %97.

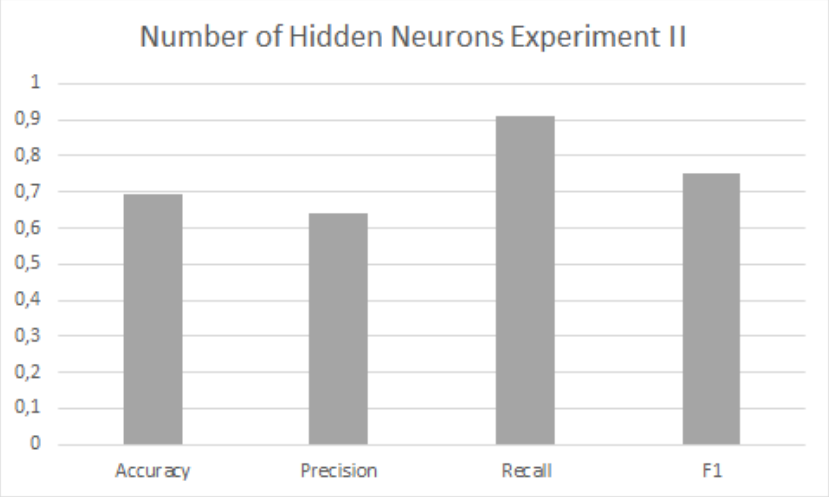


Figure 4.15. Number of Hidden Neurons Experiment II (Using NSL-KDD)

In the second experiment, the number of hidden neurons was set equal to the size of the input layer. The result was unfruitful with low scores of accuracy and precision.

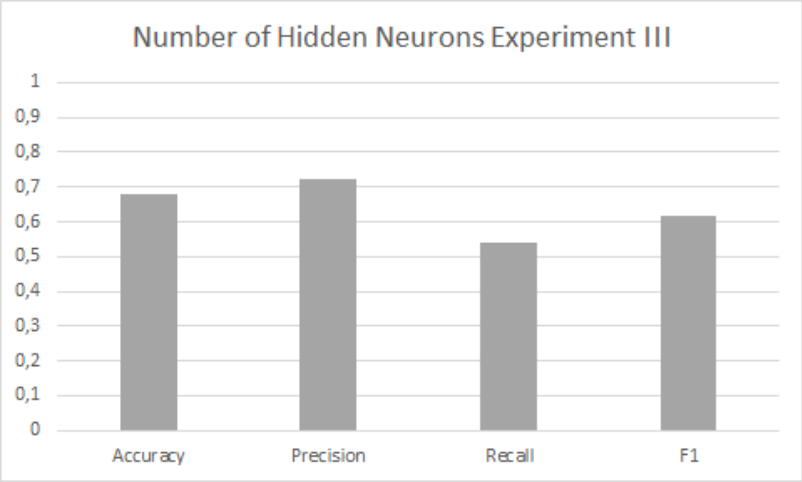


Figure 4.16. Number of Hidden Neurons Experiment III (Using NSL-KDD)

In the third experiment, hidden neurons were one and a half times of the input layer's size. Again, the result was not successful. Recall got lowered, which affected F1 score, too.

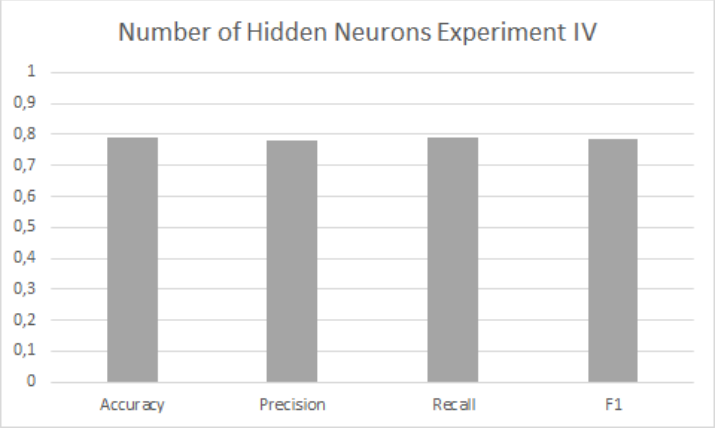


Figure 4.17. Number of Hidden Neurons Experiment IV (Using NSL-KDD)

In the next experiment, all scores were balanced. For this one, the number of the hidden neurons was half the size of the input layer. The result, again, was unsuccessful.

In the last experiment (V), the square root of the product of the input layer size and output layer size was used to set the number of hidden neurons. Different from the last three experiments, the result got better, although the first experiment remained the best one with its scores. Additionally, the score of the first experiment passed the best results with the LSTM-RNN solution's NSL-KDD experiments.

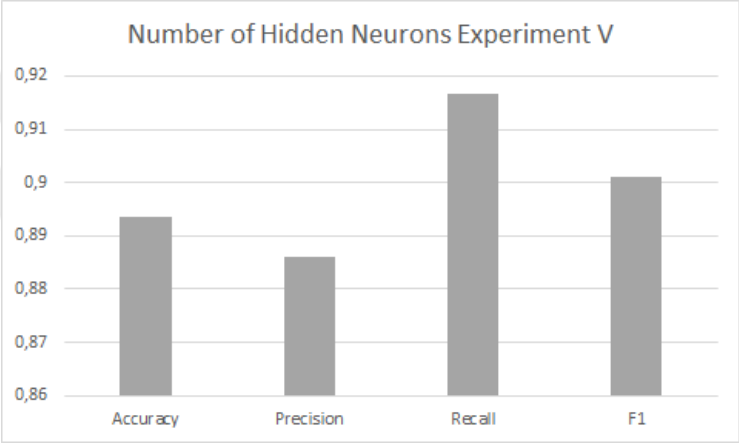


Figure 4.18. Number of Hidden Neurons Experiment V (Using NSL-KDD)

4.3.2. Using UNSW-NB15 Dataset

The same dataset as the first dataset in the LSTM-RNN solution was used in the last experiments with DRL in this section. Again, after experimenting with different configurations, the best solution was found as detailed below. Different from the LSTM-RNN experiment, this time default training and testing sets have been used initially. The training set has a total of 175341 records, whereas the test set has 82332

records. Accuracy was 3 percent better than the LSTM-RNN solution. Results are given in the Figure 4.20 below.

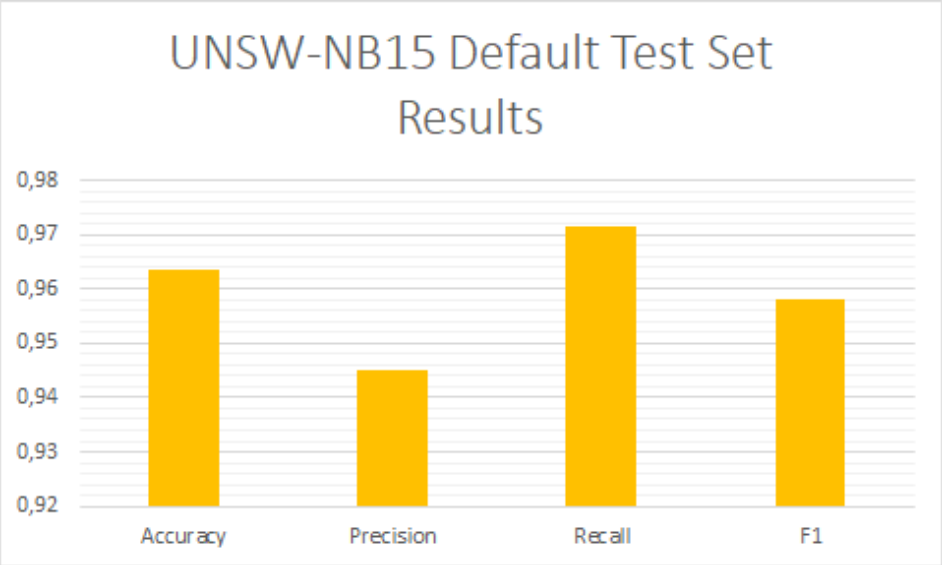


Figure 4.19. Performance results of default test data set of UNSW-NB15 using DRL

The second experiment was held by using randomly picked training and test data over the dataset. 100000 records have been picked for both sets. The results have not changed in this experiment compared to the experiment using default sets.

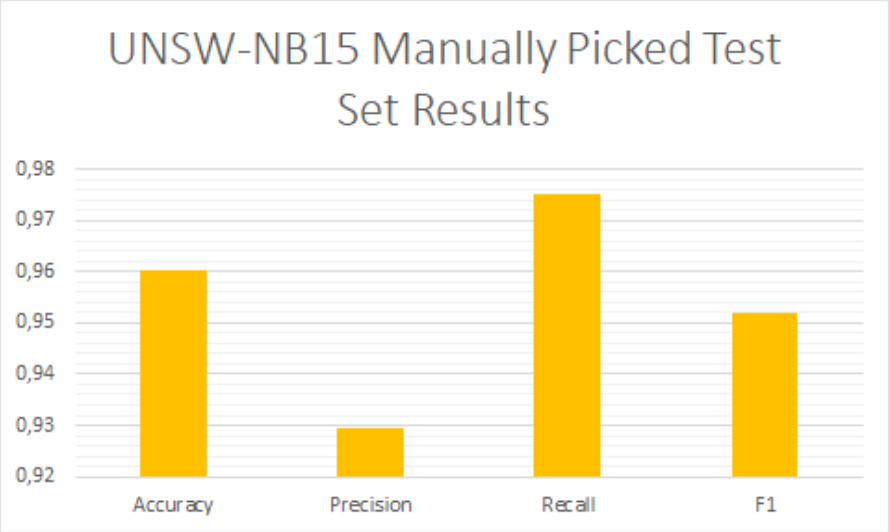


Figure 4.20. Performance results of manually picked test data set of UNSW-NB15 using DRL

4.4. Comparison with Other Solutions

In order to measure the real success of the system, it should be compared with the other solutions provided by different authors. The table below shows the accuracy of some novel solutions using the NSL-KDD dataset:

Table 4.6. *Comparison of different intrusion detection systems*

Solution	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)
Tang et al., 2016 (with 0.0001 learning rate)	91.7	83	75	74
Self-taught Learning (Niyaz et al., 2015)	88.39	85.44	95.95	75.76
Soft-max Regression (Niyaz et al., 2015)	78.06	96.56	63.73	72.14

Random Forest Modeling (Farnaaz and Jabbar, 2016)	99.67	-	-	-
RNN-IDS (Chuan-long et al., 2017)	97.09	-	-	-
Aljawarneh et al., 2018	99.81	-	-	-
Random Tree + NBTree (Kevric et al., 2018)	99.53	-	-	-
Deep Reinforcement Learning (proposed in this thesis)	96.72	98.06	96.07	97.04

The comparison shows that although the solution proposed in this thesis performs better than some novel solutions, it is not the best solution for network intrusion detection. Therefore, it can be said that there is still room for improvement to decrease the 3 percent difference from the best solutions.

CHAPTER 5

CONCLUSION

Thanks to advancements in the big data technologies, big data analytics became one of the most useful resources in many fields in computer science. As cybersecurity is crucial in our lives today, using these analytics in security solutions is both inevitable and beneficial. Additionally, because of the nature of most cybersecurity attacks, relevant big data can be extracted from the network by fetching traffic data, system logs etc.

With their increasing popularity in many learning tasks, neural networks are used frequently as a solution (or part of the solution) to these problems. Likewise, they are used in the solutions proposed in this thesis, too. In order to handle zero-day attacks, using a learning mechanism is a must. As nowadays most networks are open to zero-day threats, a security solution which can detect anomalies, even if those types of anomalies were never seen before, is a necessity.

The main aim of the thesis is to provide a solution for system security through detection of possible intrusions by using big data solutions to handle high data flow as streaming data. Network data is a prime example of streaming big data. Traffic data is quite large in terms of features and the number of records. In this work, the data is processed by Apache Spark and passed onto the intrusion detection component of the solution that utilizes machine learning algorithms. By using two different machine learning approaches, namely LSTM-RNN and Deep Reinforcement Learning, anomalies in the network are detected and reported. Both solutions are run with the help of Tensorflow, a machine learning framework. Neural networks are used in both solutions. Experiments have been performed using various datasets with the framework. Diverse configurations have been tried in order to find the optimal solution. Especially, setting the number of neurons for the hidden layer was essential to the task. Initially, LSTM-RNN seemed a good candidate for the problem with its

nature using time-series. Then, DRL also seemed to be an effective solution as a part of this intrusion detection system. Therefore, the experiments helped us compare these two approaches. Different datasets were used for testing the accuracy of the system. In summary, although there is still room for improvement, accuracy rates in the experiments were most of the time greater than %90 for both solutions, which was promising. DRL seemed to be performing slightly better than LSTM-RNN, hence the DRL solution was found more suitable for the system.

As cloud systems are getting more and more popular every day, a special security solution for these systems is needed. All service providers have their own monitors for providing different statistics of their systems, such as network statistics, CPU status, disk usage etc. For example, Amazon has CloudWatch, Google has Stackdriver, and all these tools have easy to implement integration methods for providing system metrics. Therefore, as a future work, a customized system, which reads these statistics via these monitors and processes them with the big data and machine learning solutions like the one described in this paper, could be developed.

In summary, an end-to-end intrusion detection system has been developed throughout the thesis. A big data solution was designed in order to process streaming big data. Different machine learning solutions were adapted and tested. Promising results have been achieved after experimenting with different datasets namely, KDD, NSL-KDD, CICIDS2017, and UNSW-NB15. Cloud systems have been investigated for future possible improvements and using their own metric providers to create a custom security system has been determined as a future work direction. Relatedly, a cloud-specific intrusion detection dataset, not only with traffic data but also including system metrics such as CPU utilization, memory utilization, disk performance and read writes, does not exist. Such a dataset could be crucial for designing a purely cloud oriented security system. Therefore, preparing and publishing such a dataset could be another future work, and would be an initial step for a cloud-specific solution.

REFERENCES

Apache. (2019, August). *Apache kafka, a distributed streaming platform.* Retrieved from https://kafka.apache.org/

Apache. (2019, August). *Apache spark, lightning fast unified analytics engine.* Retrieved from https://kafka.apache.org/

Aljawarneh, S., Aldwairi, M., & Yassein, M. B. (2018). Anomaly-based intrusion detection system through feature selection analysis and building hybrid efficient model. *Journal of Computational Science, 25*, 152–160. https://doi.org/10.1016/j.jocs.2017.03.006

Amazon. (2019, August). *Amazon kinesis.* Retrieved from https://aws.amazon.com/kinesis

Balogun, A. O., & Jimoh, R. G. (2015). Anomaly intrusion detection using an hybrid of decision tree and K-nearest neighbor. *Journal of Advances in Scientific Research & Applications (JASRA), 2*(1), 67-74.

Behera, S., Pradhan, A., & Dash, R. (2018). Deep Neural Network Architecture for Anomaly Based Intrusion Detection System. *2018 5th International Conference on Signal Processing and Integrated Networks, SPIN 2018*, 270–274. https://doi.org/10.1109/SPIN.2018.8474162

Bharadwaja, S., Sun, W., Niamat, M., & Shen, F. (2010). Collabra: A xen hypervisor based collaborative intrusion detection system. *Proceedings - 2011 8th International Conference on Information Technology: New Generations, ITNG 2011*, 695–700. https://doi.org/10.1109/ITNG.2011.123

Carbon Black. (2019). *Global Threat Report: The Year of the Next-Gen Cyberattack.* Retrieved from https://www.carbonblack.com/wp-content/uploads/2019/01/carbon-black-global-threat-report-year-of-the-next-gen-cyberattack-0119.pdf

Casas, P., Soro, F., Vanerio, J., Settanni, G., D'Alconzo, A. (2017). Network security and anomaly detection with big-dama, a big data analytics framework. *Proceedings of 2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*, 1-7. https://doi.org/10.1109/CloudNet.2017.8071525

Chuan-long, Y., Yue-fei, Z., Jin-long, F., & Xin-zheng, H. (2017). A Deep Learning Approach for Intrusion Detection using Recurrent Neural Networks. *IEEE Access, 5*, 1–1. https://doi.org/10.1109/ACCESS.2017.2762418

Cloud Security Alliance. (2013). *Big Data Analytics for Security Intelligence.* Retrived from https://downloads.cloudsecurityalliance.org/initiatives/bdwg/Big_Data_Analytics_for_Security_Intelligence.pdf

Cuayáhuitl, H., Keizer, S., & Lemon, O. (2015). *Strategic Dialogue Management via Deep Reinforcement Learning*. 1–10. Retrieved from http://arxiv.org/abs/1511.08099

Deokar B., Hazarnis A. (2012). Intrusion Detection System using Log Files and Reinforcement Learning. *International Journal of Computer Applications*, *45*(19), 28–35.

Deshpande, P., Sharma, S. C., Peddoju, S. K., & Junaid, S. (2018). HIDS: A host based intrusion detection system for cloud computing environment. *International Journal of Systems Assurance Engineering and Management*, *9*(3), 567–576. https://doi.org/10.1007/s13198-014-0277-7

Elasticsearch. (2019, August). *Elasticsearch.* Retrieved from https://www.elastic.co

Elderman, R., Pater, L. J. J., Thie, A. S., Drugan, M. M., & Wiering, M. A. (2017). Adversarial reinforcement learning in a cyber security simulation. *ICAART 2017- Proceedings of the 9th International Conference on Agents and Artificial Intelligence*, *2*(Icaart), 559–566. https://doi.org/10.5220/0006197105590566

Ellingwood, J. (2016, October 28). *Hadoop, storm, samza, spark, and flink: big data frameworks compared [Blog post]*. Retrieved from https://www.digitalocean.com/community/tutorials/hadoop-storm-samza-spark-and-flink-big-data-frameworks-compared

Farnaaz, N., & Jabbar, M. A. (2016). Random Forest Modeling for Network Intrusion Detection System. *Procedia Computer Science*, *89*, 213–217. https://doi.org/10.1016/j.procs.2016.06.047

Google Trends. (2019, August 6). *Worldwide popularity of big data*. Retrieved from https://trends.google.com/trends/explore?date=all&q=big%20data

Gupta, D., & Rani, R. (2018). Big Data Framework for Zero-Day Malware Detection. *Cybernetics and Systems*, *49*(2), 103–121. https://doi.org/10.1080/01969722.2018.1429835

Hariharan, A., Gupta, A., & Pal, T. (2019). *CAMLPAD: Cybersecurity Autonomous Machine Learning Platform for Anomaly Detection*. Retrieved from http://arxiv.org/abs/1907.10442

Hochreiter, S., Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735-1780.

Ibrahim, A. S., Hamlyn-Harris, J., & Grundy, J. (2016). *Emerging security challenges of cloud virtual infrastructure*. Retrieved from http://arxiv.org/abs/1612.09059

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1997). *Live-301-1562-Jair*. 1–49. Retrieved from http://www.jair.org/media/301/live-301-1562-jair.pdf

KDD. (1999). *KDD Cup 1999: Computer network intrusion detection*. Retrieved from https://www.kdd.org/kdd-cup/view/kdd-cup-1999/Data

Kevric, J., Jukic, S., & Subasi, A. (2017). An effective combining classifier approach using tree algorithms for network intrusion detection. *Neural Computing and Applications*, *28*(s1), 1051–1058. https://doi.org/10.1007/s00521-016-2418-1

Kim J., Kim H. (2016). Applying Recurrent Neural Network to Intrusion Detection with Hessian Free Optimization. In: Kim H., Choi D. (eds) *Information Security Applications*. WISA 2015. Lecture Notes in Computer Science, vol 9503. Springer, Cham

Koduvely H. (2018). *Github repository, gym-network_intrusion*. Retrieved from https://github.com/harik68/gym-network_intrusion

Lai, M. (2015). *Giraffe: Using Deep Reinforcement Learning to Play Chess.* (September). Retrieved from http://arxiv.org/abs/1509.01549

Lipton, Z. C., Berkowitz, J., & Elkan, C. (2015). *A Critical Review of Recurrent Neural Networks for Sequence Learning*. 1–38. Retrieved from http://arxiv.org/abs/1506.00019

Mahbod T., Ebrahim B., Wei L., & Ali A. G. 2009. A detailed analysis of the KDD CUP 99 data set. *Proceedings of the Second IEEE international conference on Computational intelligence for security and defense applications (CISDA'09)*. IEEE Press, Piscataway, NJ, USA, 53-58.

Mahmood T., Afzal U. (2013). Security Analytics: Big Data Analytics for cybersecurity: A review of trends, techniques and tools. *2013 2nd National Conference on Information Assurance (NCIA)*, Rawalpindi, 2013, pp. 129-134. doi: 10.1109/NCIA.2013.6725337

Mahmud, M., Kaiser, M. S., Hussain, A., & Vassanelli, S. (2018). Applications of Deep Learning and Reinforcement Learning to Biological Data. *IEEE Transactions on Neural Networks and Learning Systems, 29*(6), 2063–2079. https://doi.org/10.1109/TNNLS.2018.2790388

Maiero, C., & Miculan, M. (2012). Unobservable Intrusion Detection Based on Call Traces in Paravirtualized Systems. *Proceedings of the International Conference on Security and Cryptography*, 300–306. https://doi.org/10.5220/0003521003000306

Manliguez, C. (2016). Generalized Confusion Matrix for Multiple Classes *Generalized Confusion Matrix for Multiple Classes The total numbers of false negative (TFN), false positive (TFP), and true negative (TTN) for each class i will be calculated based on the Generalized. (November)*, 6–8. https://doi.org/10.13140/RG.2.2.31150.51523

Mcdaniel, P., Smith, S. W., Cárdenas, A. A., Manadhata, P. K., Hp, |, Sreeranga, L., & Rajan, P. (2013). *SYSTEMS SECURITY Big Data Analytics for Security*. (December), 74–76.

Mishra, P., Pilli, E. S., Varadharajan, V., & Tupakula, U. (2017). Intrusion detection techniques in cloud environment: A survey. *Journal of Network and Computer Applications*, Vol. 77, pp. 18–47. https://doi.org/10.1016/j.jnca.2016.10.015

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). *Playing Atari with Deep Reinforcement Learning*. 1–9. Retrieved from http://arxiv.org/abs/1312.5602

Nemati, S., Ghassemi, M. M., & Clifford, G. D. (2016). Optimal medication dosing from suboptimal clinical examples: A deep reinforcement learning approach. *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, 2016-October, 2978–2981. https://doi.org/10.1109/EMBC.2016.7591355

Nicholson, C. (2018). *A Beginner's Guide to LSTMs and Recurrent Neural Networks.* Retrieved from. https://skymind.ai/wiki/lstm

Niyaz, Q., Sun, W., Javaid, A. Y., & Alam, M. (2015). A deep learning approach for network intrusion detection system. *EAI International Conference on Bio-Inspired Information and Communications Technologies (BICT)*. https://doi.org/10.4108/eai.3-12-2015.2262516

Olah, C. (2015). *Understanding lstm networks*. Retrieved from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

OpenAI. (2019, August). *OpenAI Gym.* Retrieved from https://gym.openai.com/

Panigrahi, R., & Borah, S. (2018). A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems. *International Journal of Engineering and Technology (UAE), 7*(3.24 Special Issue 24)

Pervez, M. S., & Farid, D. M. (2014). Feature selection and intrusion classification in NSL-KDD cup 99 dataset employing SVMs. *SKIMA 2014 - 8th International Conference on Software, Knowledge, Information Management and Applications*, 1–6. https://doi.org/10.1109/SKIMA.2014.7083539

Powers, D.M.W. (2011). *Evaluation: from Precision, Recall and F-measure to ROC, Informedness, Markedness and Correlation*. Journal of Machine Learning Technologies, 2(1), 37-63

Razaq, A., Tianfield, H., & Barrie, P. (2016). A big data analytics based approach to anomaly detection. *2016 IEEE/ACM 3rd International Conference on Big Data Computing Applications and Technologies (BDCAT)*, 187–193. https://doi.org/10.1145/3006299.3006317

Servin, A., & Kudenko, D. (2008). Multi-agent reinforcement learning for intrusion detection: A case study and evaluation. *Frontiers in Artificial Intelligence and Applications, 178*, 873–874. https://doi.org/10.3233/978-1-58603-891-5-873

Sharafaldin, I., Habibi Lashkari, A., & Ghorbani, A. A. (2018). *Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization.* (Cic), 108–116. https://doi.org/10.5220/0006639801080116

Sharifi, A. M., Amirgholipour, S. K., & Pourebrahimi, A. (2015). Intrusion detection based on joint of K-means and KNN. *Journal of Convergence Information Technology, 10*(5), 42

Tang, T. A., Mhamdi, L., McLernon, D., Zaidi, S. A. R., & Ghogho, M. (2016). Deep learning approach for Network Intrusion Detection in Software Defined Networking. *Proceedings - 2016 International Conference on Wireless Networks and Mobile Communications, WINCOM 2016: Green Communications and Networking*, 258–263. https://doi.org/10.1109/WINCOM.2016.7777224

University of New Brunswick Canadian Institute for Cybersecurity. (n.d.). *NSL-KDD dataset.* Retrieved from https://www.unb.ca/cic/datasets/nsl.html

University of New Brunswick Canadian Institute for Cybersecurity. (n.d.). *Intrusion Detection Evaluation Dataset (CICIDS2017) dataset.* Retrieved from https://www.unb.ca/cic/datasets/ids-2017.html

University of New South Wales Canberra at Australian Defence Force. (2015). *The UNSW-NB15 dataset*. Retrieved from https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-NB15-Datasets

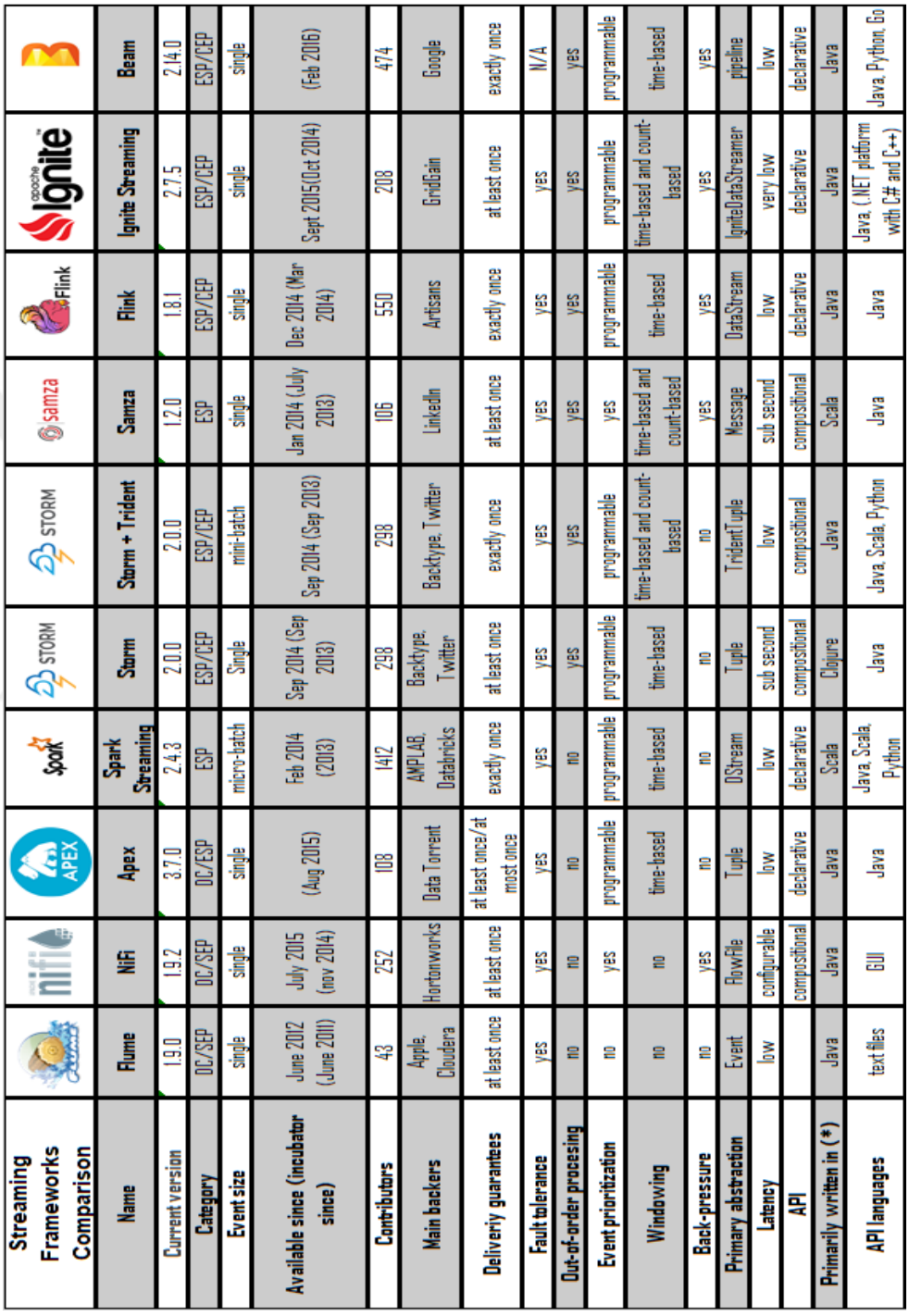
Wang, L., Zhang, D., Gao, L., Song, J., Guo, L., & Shen, H. T. (2018). MathDQN: Solving arithmetic word problems via deep reinforcement learning. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, 5545–5552.

Yen, T., Oprea, A., & Onarlioglu, K. (2013). Beehive: large-scale log analysis for detecting suspicious activity in enterprise networks. *Proceedings of the 29th Annual Computer Security Applications Conference*, 199–208. https://doi.org/10.1145/2523649.2523670

Zhou, Z., Li, X., & Zare, R. N. (2017). Optimizing Chemical Reactions with Deep Reinforcement Learning. *ACS Central Science*, *3*(12), 1337–1344. https://doi.org/10.1021/acscentsci.7b00492

APPENDIX

A. COMPARISON OF STREAMING DATA FRAMEWORKS



Streaming Frameworks Comparison	Flume	NiFi	Apex	Spark Streaming	Storm	Storm + Trident	Samza	Flink	Ignite Streaming	Beam
Current version	1.9.0	1.9.2	3.7.0	2.4.3	2.0.0	2.0.0	1.2.0	1.8.1	2.7.5	2.14.0
Category	DC/SEP	DC/SEP	DC/ESP	ESP	ESP/CEP	ESP/CEP	ESP	ESP/CEP	ESP/CEP	ESP/CEP
Event size	single	single	single	micro-batch	Single	mini-batch	single	single	single	single
Available since (incubator since)	June 2012 (June 2011)	July 2015 (nov 2014)	(Aug 2015)	Feb 2014 (2013)	Sep 2014 (Sep 2013)	Sep 2014 (Sep 2013)	Jan 2014 (July 2013)	Dec 2014 (Mar 2014)	Sept 2015 (Oct 2014)	(Feb 2016)
Contributors	43	252	108	1412	298	298	106	550	208	474
Main backers	Apple, Cloudera	Hortonworks	Data Torrent	AMPLAB, Databricks	Backtype, Twitter	Backtype, Twitter	LinkedIn	Artisans	GridGain	Google
Delivery guarantees	at least once	at least once	at least once/at most once	exactly once	at least once	exactly once	at least once	exactly once	at least once	exactly once
Fault tolerance	yes	yes	yes	yes	yes	yes	yes	yes	yes	N/A
Out-of-order processing	no	no	no	no	yes	yes	yes	yes	yes	yes
Event prioritization	no	yes	programmable	programmable	programmable	programmable	yes	programmable	programmable	programmable
Windowing	no	no	time-based	time-based	time-based	time-based and count-based	time-based and count-based	time-based	time-based and count-based	time-based
Back-pressure	no	yes	no	no	no	no	yes	yes	yes	yes
Primary abstraction	Event	FlowFile	Tuple	DStream	Tuple	TridentTuple	Message	DataStream	IgniteDataStreamer	pipeline
Latency	low	configurable	low	low	sub second	low	sub second	low	very low	low
API		compositional	declarative	declarative	compositional	compositional	compositional	declarative	declarative	declarative
Primarily written in (*)	Java	Java	Java	Scala	Clojure	Java	Scala	Java	Java	Java
API languages	text files	GUI	Java	Java, Scala, Python	Java	Java, Scala, Python	Java	Java	Java, (.NET platform with C# and C++)	Java, Python, Go

Figure A.1. Comparison of Streaming Data Frameworks