GAN based Network Traffic Generation for Communication Protocols used in SCADA Architectures

Masterthesis

by **Gerrit Seifert**

Matriculation number: 408329
Course of Studies: Computer Science (Master of Science)

Supervised by:
Prof. Dr. Anne Remke

Westfälische Wilhelms-Universität Münster
Department of Mathematics and Computer Science
Group of Safety-Critical Systems

Münster, March 20, 2023

GAN basierte Generierung von Netzwerkdatenverkehr für Kommunikationsprotokolle zur Anwendung in SCADA Architekturen

Masterarbeit

von **Gerrit Seifert**

Matrikelnummer: 408329
Studiengang: Informatik (Master of Science)

Arbeit betreut durch:
Prof. Dr. Anne Remke

Westfälische Wilhelms-Universität Münster
Fachbereich Mathematik und Informatik
Arbeitsgruppe Sicherheitskritische Systeme

Münster, 19. Januar 2023

Abstract

To decrease emissions of greenhouse gases, energy production is shifting towards renewable energy sources. Increasingly, new buildings are equipped with solar panels, which results in power generation on the consumer side. Thus, today, there is a shift of electricity production from a centralized process to a decentralized one. Renewable energies, however, underlie fluctuations in their production yield. These fluctuations in conjunction with an overall increase in energy demand lead to new requirements for the energy grid.

Means to monitor and control the decentralized energy production through renewable energy sources and the distribution are now required. Smart grids present a solution to this. These are electrical grids, utilizing digital technologies to enable communication between energy production and consumption sites. Smart grids apply the SCADA communication framework, which is specialized towards processes taking place in wide geographical areas.

To provide means for the cyber resilience of smart grids, researchers require access to network traffic of SCADA systems. However, obtaining traffic has proven to be challenging. Moreover, the few publicly available datasets seldomly contain large quantities of malicious traffic. Thus, being able to utilize a software solution to create arbitrary amounts of synthetic network traffic samples would be desirable.

Generative adversarial networks are means to generate arbitrary amounts of synthetic data. However, there are not many published approaches in the research field of generating network traffic via GANs and especially none which try to generate traffic of communication protocols of SCADA systems. As a starting point, this thesis is therefore concerned with an investigation whether GANs would be suitable to generate datagrams stemming from SCADA systems at all.

It is examined whether a particular GAN architecture would be able to capture and reproduce the structure of two specific SCADA protocols. Since the training data consists of byte sequences of datagrams, a generative adversarial network which is specialized for processing sequential data is utilized. A number of experiments are conducted to evaluate whether the GAN is able to generate datagrams that obey a certain protocol definition. Generated network packets are considered correct, if they are recognized by the widely used protocol analyzer Wireshark.

Contents

List of Figures

List of Tables

List of Abbreviations

ADU Application Data Unit.

AI Artificial Intelligence.

ANN Artificial Neural Network.

APCI Application Protocol Control Information.

APDU Application Protocol Data Unit.

API Application Programming Interface.

ASDU Application Service Data Unit.

DCS Distributed Control System.

EM Earth Mover Distance.

EPIC Electrical Power and Intelligent Control.

FNN Feedforward Neural Network.

GAN Generative Adversarial Network.

HMI Human-Machine-Interface.

ICS Industrial Control System.

IDS Intrusion Detection System.

IED Intelligent Electronic Device.

IOA Information Object Address.

IP Internet Protocol.

LSTM Long Short-Term Memory.

MBAP Modbus Application Protocol Header.

MCTS Monte Carlo Tree Search.

MDP Markov Decision Process.

MTU Master Terminal Unit.

PLC Programmable Logic Controller.

ReLU Rectified Linear Unit Activation Function.

RL Reinforcement Learning.

RNN Recurrent Neural Network.

RTU Remote Terminal Unit.

SCADA Supervisory Control and Data Acquisition.

SWaT Secure Water Treatment Testbed.

TCP Transmission Control Protocol.

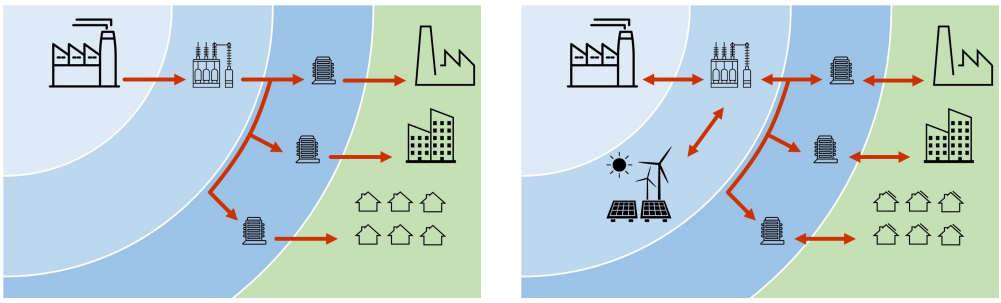
UDP User Datagram Protocol.

WGAN Wasserstein Generative Adversarial Network.

1 Introduction

Due to the pressing dangers of global warming, societies are starting to utilize more climate friendly technologies to produce energy or residential heating. Previously, fossil fuels were the main pillar of electricity and heat generation. Now there is a shift towards renewable energy sources to replace fossil fuels so that a reduction in greenhouse gas emissions is achieved. These energy sources, however, differ from fossil fuels in one key point: They are highly dependent on environmental conditions and as such less responsive to changes in demand [34]. There are not only changes on the production side, due to of the rising costs for gasoline and natural gas, but there is also a shift in consumer behaviour. Consumers are starting to substitute these fossil fuels by electricity, for example by using electric vehicles or heat pumps for their residential heating, often combined with photovoltaic panels [34].

All in all, this leads to three main differences in the utilization of the electrical grid: There is a large variability in electricity production yield, due to changing weather conditions. Production is decentralized, in contrast to the traditional more monolithic infrastructure (Figure 1.1(a)), as electricity is now also produced on the consumer side (Figure 1.1(b)). And demand for electricity has increased due to the utilization for heating and transportation.



(a) Early electrical Grid (b) Modern electrical Grid

Figure 1.1: Early and modern electrical Grids. Taken from: [58]

Production and consumption in the electrical grid have to remain in balance all the time. Thus to be able to utilize the existing electrical grid despite these challenges, a network for observation, collection of measurements and transmission of control messages of the electrical grid has been proposed. This smart grid infrastructure is able to detect problems during operation and to react accordingly, either remotely or automatically [34].

Smart grids are a modern type of electrical grid and use digital technologies to monitor and control the distribution of electricity. They are supposed to help in coordinating all participants involved in electricity production and consumption. Smart grids allow for communication between all sources of electricity generation as well as consumers, thus providing a more efficient, reliable and resilient distribution of electricity. In addition to conventional power plants, smart grids can incorporate renewable energy sources [39].

The smart grid falls under the category of *industrial control system* (ICS). These are computerized systems targeted at monitoring and controlling industrial processes. These processes include electricity production and distribution, but also any other production of goods. Monitoring and control is achieved through several electrical and mechanical components, such as sensors, actuators, measuring instruments and communication devices. ICS are often applied to critical infrastructures [53].

The *supervisory control and data acquisition* (SCADA) communication framework is a subcategory of industrial control systems. SCADA systems are able to gather and process data over long distances. The challenges of transmitting data over long distances consist of delays and degradation of data integrity. SCADA was designed to cope with these challenges and is able to incorporate transmission via various physical media. Thus it is predestined for application in electricity distribution and smart grids [53]. Figure 1.2 depicts the application of SCADA in a smart grid. Status information of grid components is gathered and transmitted to a centralized monitoring facility. This facility is then able to execute control actions of grid components.

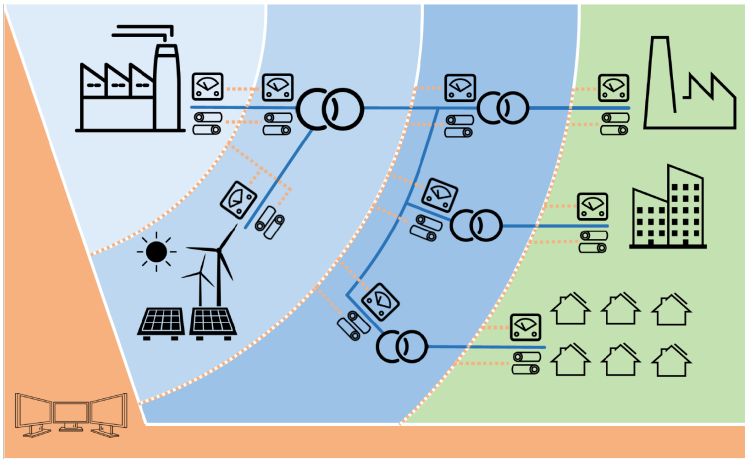


Figure 1.2: Smart Grid SCADA System. Taken from: [58]

A SCADA network controlling the electrical grid is to be considered a critical infrastructure whose malfunction could cause consequences with high probability. It therefore presents a high value target for cyber attacks. The term cyber resilience describes the capability of a system to withstand and recover

from disruptions, such as cyber attacks, while delivering the desired outcome at any time [14].

In the last few years several attacks on electrical grids were observed. Especially noteworthy are those on the Ukrainian infrastructure due to the context of military conflicts. In 2015 and 2016, multiple attacks on industrial control systems of the Ukrainian power distribution were conducted, causing large blackouts [36]. In April 2022, new attacks targeted the power grid during the Russian invasion, but blackouts were mostly averted this time [8]. Attackers attempted to deploy a new version of the malware used in 2016 against high-voltage electrical substations. This malware was targeted towards SCADA systems utilizing the IEC 60870-5-104 communication protocol [18].

SCADA networks are not inherently secure [56]. They need to be secured by proven security concepts as well as novel *intrusion detection systems* (IDS). This is due to their fundamentally different characteristics compared to traditional computer networks such as the internet. The development of novel IDS, however, requires a good understanding of specific characteristics of SCADA systems and data to test them. Since control systems for energy production and distribution are part of critical infrastructure, research on live systems, however, poses a risk, due to their continuous availability requirement. Moreover, captures of network traffic of critical infrastructures are rare, since real network traffic would always leak information about their composition.

Thus obtaining real SCADA traffic of such infrastructures poses a challenging task. While there are publicly available datasets, they often contain no more than small amounts of SCADA traffic. A conventional approach to obtain more traffic is to build testbeds and capture their communication [43] [4].

While various machine learning based solutions towards intrusion detection in industrial control systems have been proposed [49], often large datasets of SCADA network traffic are required to evaluate those, especially with regard to deep learning based techniques. Evaluating them on freely available SCADA datasets, therefore poses a problem. On top of that, the collection of training data containing novel attacks is not unproblematic due to the aforementioned reasons.

But not only the obtainment of the required data poses a problem. The labeling of training data, for example the decision whether a packet contains benign rather than malicious traffic, is not feasible. A possibility to counter these problems would be to generate synthetic traffic, technically very close to real traffic. This would create the ability to create datasets of arbitrary size, containing as much benign and malicious traffic as needed.

Generative adversarial networks (GAN) are often used to mimic pixel images and generate highly convincing lookalikes [27]. They basically learn a non-linear function to convert the probability distribution of a set of arbitrary continuous data into the distribution of the training data. Doing so allows GANs to generate various kinds of data, aside from images. However, gener-

ating network traffic may pose a problem.

A GAN in its basic form is well suited to generate continuous values, for example measurements, that become incorporated into a SCADA packet. Network packets, however, also require categorical data, such as IP addresses, or discrete values. Those kinds of data have to somehow be represented using continuous values. Since a conversion is already required, one could abstract from single values contained inside a packet and instead consider only the sequential packet bytes. This has the advantage that less domain knowledge of a specific protocol and pre-processing is required, therefore the technique can be transferred to other protocols without much adaptation.

Before being able to utilize generative adversarial networks to generate benign and malicious network traffic, this thesis is intended to be a starting point for further research in the context of generating SCADA network traffic. Therefore, the goal of this thesis will be to examine whether GANs can be used to generate syntactically correct packets obeying a respective protocol definition at all. As case studies, Modbus/TCP and IEC 60870-5-104 will be used, two common protocols used in SCADA applications.

A packet is considered syntactically correct, if it complies with the corresponding protocol definition. This will be evaluated using the widely used packet analyzer Wireshark. Thus for the conducted experiments a packet is considered syntactically correct, if it gets recognized as correct by Wireshark. The presented approach will interpret network packets as sequences of bytes in hexadecimal notation and try to generate new, similar sequences. To achieve this, a combination of adversarial and reinforcement learning is used.

In Chapter 2 the application area of the research subject will be outlined. Industrial control and SCADA systems will be defined in detail as well as the communication protocols which will be used as a case study. In Chapter 3, the individual building blocks of the proposed architecture will be formally defined, including basic generative adversarial networks. Additionally, their downsides in the context of generating network datagrams as well as possible solutions will be described. Chapter 4 outlines the application of the previously defined techniques and their evaluation. In Chapter 5 the results of the conducted experiments will be interpreted and discussed. Finally, Chapter 6 concludes the findings of this thesis.

2 Application Area

The intended application area for the proposed generation of network traffic are processes in critical infrastructure. The distribution and production of electrical energy and therefore smart grids will mainly be used as examples in this chapter. Firstly, because there were already documented severe attacks on this infrastructure [36][18] and secondly, because failures in these processes are very likely to have catastrophic consequences and thus especially require cyber-resilience.

The electrical grid today most often utilizes the principles of industrial control systems (ICS) and supervisory control and data acquisition (SCADA) architectures. In this chapter, these terms will be outlined in detail and two of the most common communication protocols used in such architectures are described.

Firstly, Section 2.1 gives an overview over industrial control systems. Secondly, in Section 2.2 SCADA systems and their structure are defined. Later on, in Section 2.3 the security aspects of SCADA systems that have to be considered will be outlined. Afterwards, in Section 2.4, the two communication protocols that will serve as training data later on will be presented. Lastly, in Section 2.5 other approaches for obtaining traffic will be presented. Additionally, the utilized software for analyzing and creating datagrams will be described.

2.1 Industrial Control Systems

The term *industrial control systems* (ICS) is a general term for a system which utilizes and combines electrical and mechanical components. Such systems enable computers to interact with the real world to aid in manufacturing processes or transportation and control the process output or performance. An ICS is a replacement for conventional physical and analog control mechanisms through the capabilities of information technology [53]. This provides the ability to not only control a process locally, but also to remotely control a whole production chain. A computerized ICS makes it possible, for example, to control the power generation locally, while also remotely managing the transmission of fuel for the process [57].

Processes controlled by an ICS may operate automatically or incorporate humans, who may interact with the system using components such as *human-machine-interfaces* (HMI), remote diagnostic and maintenance tools. The pro-

cess is monitored via sensors, which provide measurements to the system, and actuators, which provide the ability to interfere in a process. The communication between each component can be implemented by a variety of different communication protocols. Industrial control systems are to be found in almost every modern industry and especially critical infrastructures such as power generation, water treatment, fuel production or the food industry. Industrial control systems applied in critical infrastructure are highly interconnected and their processes often influence one another. The interconnection between components and utilization of IT principles leads to a higher necessity of safety and especially security [53].

Two primary realizations of ICS are the so called *distributed control systems* (DCS) and *supervisory control and data acquisition* systems (SCADA) [53]. While the first one is commonly used within a single process in a rather small area, such as a production plant, the latter one is typically targeted at much larger-scale operations utilizing distributed components [57].

2.2 SCADA

A supervisory control and data acquisition (SCADA) system is a computerized system targeted at the gathering and processing of data, as well as control of processes, over long distances [53]. In the context of electric power generation and distribution, a SCADA system is typically used for the distribution in a region or larger area, while the generation process is controlled by a distributed control system [57].

The two main paradigms of SCADA systems are their ability to control processes over long distances in wide areas and their centralized structure. A SCADA system consists of components that acquire field information, such as sensors, and components which allow for state changes in a remote system, such as actuators. Both types of components are then combined with data transmission systems. These convey measurements and control information to and from a centralized monitoring station, in which a human-machine-interface displays it to an operator, thus providing the ability for centralized monitoring and control. The grade of autonomy depends on the system. A third important characteristic of SCADA systems is their commonly redundant design, which aims to achieve a certain degree of fault-tolerance [53].

A general layout of a SCADA system (Figure 2.1) typically includes the following components. A centralized control server, the *master terminal unit* (MTU), is placed in a control center. It collects and processes information of one or more distributed field sites via various means of data transmission. The typical control center also houses a data historian, used for the logging of system states, engineering workstations and HMIs. The field sites may house at least one *remote terminal unit* (RTU) or *programmable logic controller* (PLC). These are responsible for the actual collection of sensor values or the

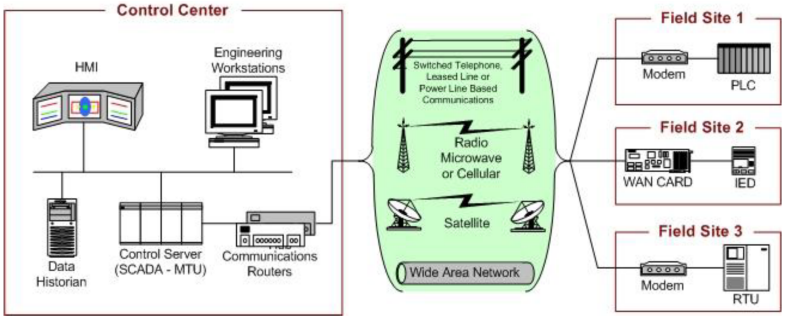


Figure 2.1: General layout of a SCADA system. Taken from: [53]

controlling of actuators. The field sites can also contain an *intelligent electronic device* (IED). IEDs are also able to control and monitor components, but they typically have a programming which allows them to operate without the direct instruction by the MTU.

2.3 Security Aspects

In terms of security, SCADA systems underlie different risks than conventional IT systems. The risk assessment of IT systems is usually concerned with protecting information. Thus as far as the three pillars of security are concerned, security in the context of IT mostly has to deal with confidentiality. However, SCADA systems are mostly targeted towards controlling processes which have to be running continuously and do not allow for any downtime. Therefore risk assessment for SCADA networks, mainly has to consider the availability of the system [72].

The nature of SCADA systems, high interconnectivity and mutual dependability, bears the risk of cascading effects. A security breach or other form of failure would not only affect directly connected equipment and could lead to catastrophic consequences [57]. One example for this would be the process of electrical power generation, which is one of the most sensitive infrastructures. A failure in the communication of a SCADA system could lead to the loss of control and monitoring capabilities. This may afterwards result in the failure of a power generator, which would cause an imbalance in the distribution grid that could finally lead to a power outage in a whole geographical region [53].

Ensuring the availability of SCADA systems is associated with various difficulties, one being that historically these networks were built in a different era, and thus were conceptualized without considering today's risks of cyberattacks. This problem is especially amplified due the utilization of standardized software and communication protocols. Instead of proprietary equipment and serial communication protocols, today conventional means of communication are utilized. TCP/IP and the internet are commonly used for the inter-

connect of the control center and the field sites. Additionally, many commonly used SCADA protocols make use of TCP/IP communication without specialized protection measures [72]. This entails that these protocols often underlie known vulnerabilities which, as well as the connectivity to potentially unsecured networks, has to be considered [57].

Mitigations against such vulnerabilities among others include the regular application of updates, which is especially challenging, due to requirement of high availability [72]. Security by obscurity is also often used, though it underlies the same fundamental problems as in IT systems, since a lot of information about critical infrastructures is already publicly available [57].

Suggested effective mitigations include the introduction of secure networks zones and as such decoupling the SCADA system from insecure networks. Additionally access control rules are implemented, as well as well-defined firewall rules. Encryption of SCADA traffic would also be possible, but is rarely used. A lot of traffic is therefore transmitted as plain text [72]. To mitigate multiple types of vulnerabilities, the application of intrusion detection is considered mandatory [72].

The development of intrusion detection systems, however, poses a problem. Since security by obscurity is usually practiced, acquiring network traffic of real SCADA systems is not a trivial issue [43]. Because of that, there are several approaches towards obtaining traffic captures by constructing testbeds and recording the communication [43][4][23].

2.4 Communication Protocols

A large variety of communication protocols is used for the implementation of SCADA systems. For the purpose of this thesis, two widely used protocols will be used as a case study. Both use standard TCP/IP as transport, while being located at the application layer.

2.4.1 Modbus/TCP

Modbus/TCP is one variant of the Modbus standard [51] and viewed as defacto standard [24]. Every variant of Modbus is defined as an application layer protocol that can be built upon various protocol stacks with different physical layers. It is often used in conjunction with TIA-232, -422 or -485 (often referred to as RS-232, and -422 or -485 respectively). Modbus/TCP on the other hand is built upon the TCP/IP stack, thus allowing Ethernet to be used as the physical layer[51][50]. It defines a standard that embeds the Modbus messages into the TCP payload.

Modbus presents a solely synchronous request/response based protocol, its communication takes place in a master-slave manner only. Each transaction, referred to as query by the standard [51], is initiated by the master device and is

assigned a unique *transaction identifier*. The slave devices respond accordingly with the requested data or perform a requested action. The protocol demands that only one device assumes the role of the master device. Commonly an HMI acts as the master device, while the other slave devices are often PLCs [24].

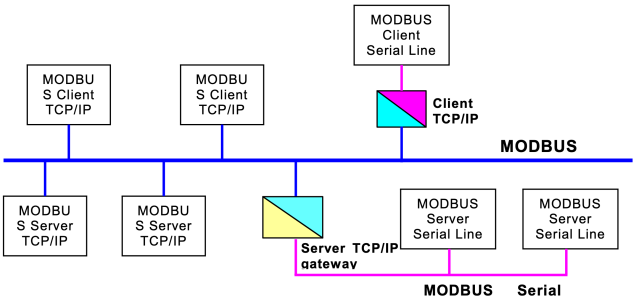


Figure 2.2: Heterogeneous Modbus Architecture. Taken from: [50]

A slave device is normally addressed by its IP address. But the Modbus standard also allows for heterogeneous architectures which use multiple variants of Modbus. In this case multiple Modbus slaves can be chained together and connected to an IP network via a serial connection to a gateway (see Figure 2.2), which is then addressed by its IP address [24]. Modbus/TCP devices use the reserved well-known TCP port 502 [50].

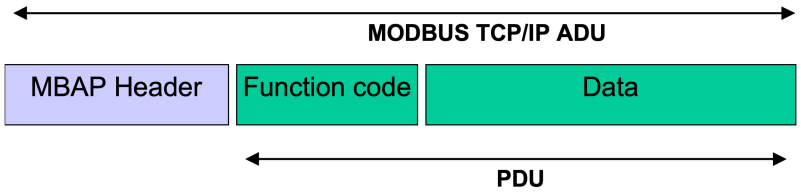


Figure 2.3: Modbus/TCP Application Data Unit. Taken from: [50]

Figure 2.3 shows the *application data unit* (ADU) and Table 2.1 breaks down the *Modbus application protocol header* (MBAP). The MBAP header allows for addressing slaves, which are connected to a gateway, in a sub-network via the *unit identifier*. The MBAP header also contains the *transaction identifier*, that is used to match the response to a particular request. Unit ID and transaction ID are copied from the request upon response [50]. The *protocol data unit* (PDU) contains the *function code*, which allows for the implementation of 127 possible functions. However, only 19 different read or write functions are defined by the Modbus standard [51].

The Modbus/TCP standard itself does not provide any measures of authentication, encryption or other mitigations against man-in-the-middle attacks

Field	Length	Description
Transaction Identifier	2 Bytes	Request/Response Transaction ID
Protocol Identifier	2 Bytes	Always 0 for Modbus/TCP
Length	2 Bytes	Number of Bytes of PDU
Unit Identifier	1 Byte	ID of Slave Device on Bus

Table 2.1: MBAP Header. Taken from: [50]

and packet injections. Additionally, due to the simple request/response communication mode, Modbus/TCP does not provide any long-term session management. However, the findings of [24] suggest, that communication of many devices mostly takes place in single long-lived TCP connections. Packet injection attacks would therefore require hijacking the TCP session or a connection reset.

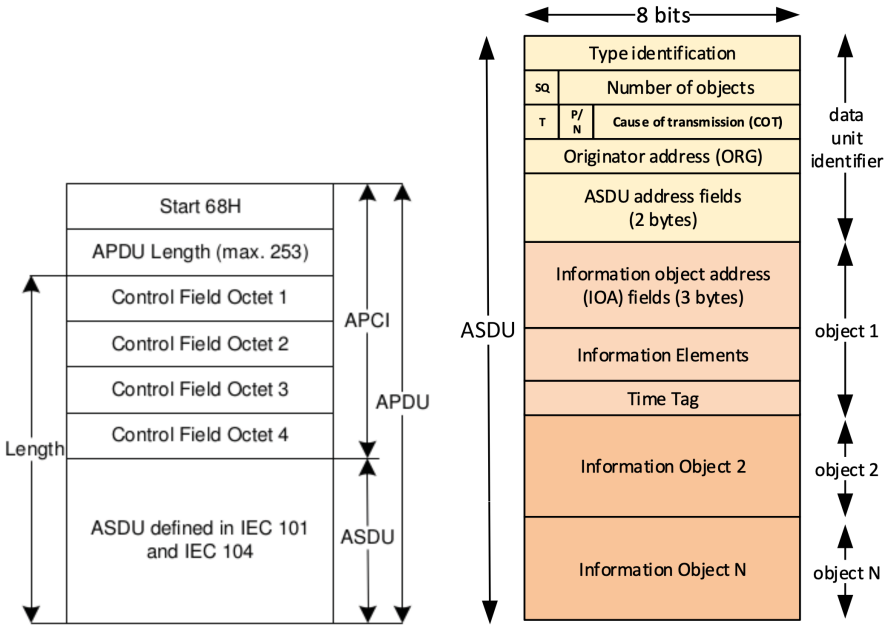
2.4.2 IEC 60870-5-104

IEC 60870-5-104 [38] is a communication protocol, defined by the IEC 60870 collection of open standards. The IEC 60870 standard consists of six main parts, all of which are concerned with the communication within SCADA systems. IEC 60870-5, part 5 of the standard, is concerned with the definition of the transmission protocols. It mainly covers IEC 60870-5-101 [37] and IEC 60870-5-104 [38], often simply referred to as IEC-104. Both are application layer protocols. While the first one defines the transmission over a serial connection, the latter one presents a variant which is built upon the TCP/IP stack, similarly to Modbus/TCP. It is widely used in critical infrastructures, especially in Europe [48].

In contrast to Modbus/TCP, IEC 60870-5 not only supports synchronous request/response communication, but also asynchronous communication, which is implemented in the form of so called spontaneous events that become sent by a field device without a prior request from the master. These are commonly invoked by an RTU in the case of a monitored parameter or state which leaves a certain value range [44]. IEC 60870-5 also allows for heterogeneous architectures which involve Ethernet and serial communication [47].

There is a distinction between control and monitor directions. IEC-104 is targeted more specifically towards SCADA systems, therefore it assumes a hierarchical structure with centralized control. As far as IEC 60870-5 is concerned, every communicating component can act as either a controlling or a controlled station. The hierarchical structure is represented by the way the communication takes place, it does always take place between a controlling and a controlled station.

The distinction between control and monitor directions presents itself in



(a) IEC-104 APDU. Taken from: [48] (b) IEC-104 ASDU. Taken from: [47]

Figure 2.4: IEC 60870-5-104 Message Format

the definition of communication modes: The *monitor direction* describes a transmission from a controlled, to a controlling station. The *control direction* is a transmission from the controlling, to the controlled station. *Reversed direction* transmissions are also allowed, in the case of a controlled station sending commands to the controlling station or a controlling station sending data to the controlled station [47].

Figure 2.4 shows the two parts of the IEC-104 frame format. The protocol defines the whole frame as an *application protocol data unit* (APDU), while the header is called *application protocol control information* (APCI). An APCI header contains the length of the APDU, four 8-bit control fields and must start with the character 68 in hexadecimal notation. An APDU has a maximum size of 253 bytes. The control fields may contain sequence numbers and describe the type of frame-format that is used. There are three defined frame-formats, namely I-format, S-format and U-format [76].

The unnumbered U-format is used for testing purposes and to initiate and stop the communication, while the S-format is used for numbered supervisory functions. S-format and U-format APDUs only consist of the APCI header. The I-format is used for the transmission of monitoring and control information. An I-format APDU contains the APCI as well as the *application service data unit* (ASDU), which contains the application data payload [44]. Figure 2.4(a) depicts an I-format APDU.

Since heterogeneous architectures that combine Ethernet and serial commu-

nication are common, addressing in IEC 60870-5 is implemented similarly to Modbus/TCP. The transmissions from the centralized control server to the field sites are addressed via the network and transport layer, thus via IP addresses and TCP ports. Inside a field station the ASDU addresses a particular component using the ASDU address field, which is referred to as ASDU common address [47]. Application data is contained inside the so called *information objects*, which are addressed by the *information object address* (IOA). These refer to a specific data object inside a component. The information itself, such as a measurement, is contained in the *information element*. An APDU may contain up to 127 information objects. In combination, the IOA and the device common address uniquely define a particular data object [47].

The ASDU type id refers to the particular data type of the information element and applies to all information objects contained in an ASDU. The type ids are divided into certain groups depending on the transmission direction (see Table 2.2). Process information in monitoring direction refers to measurements, for example. While process information in control direction may refer to commands, such as writing a certain value, system information covers initiation or reset commands.

Type ID	Group
1-40	Process Information in Monitor Direction
45-51	Process Information in Control Direction
70	System Information in Monitor Direction
100-106	System Information in Control Direction
110-113	Parameter Information in Control Direction
120-126	File Transfer

Table 2.2: ASDU Type ID. Taken from: [47]

IEC-104 possesses the same inherent security flaws as Modbus/TCP. It provides no authentication or encryption mechanisms and is vulnerable to man-in-the-middle and injection attacks [48][76].

2.5 Related Work

The most common approach towards creating SCADA network traffic captures has been to construct testbeds and capture the traffic [43]. While this approach is rather complex and cost-intensive depending on how realistic the model is, it has additional benefits. Access to live SCADA systems is normally very restricted due to the potential security risks and traffic samples of such systems are rare. Constructing a model that consists of the same components used in SCADA systems, then makes it possible to experiment on real hardware.

Many researchers have used SCADA testbeds for experimental investigations of vulnerabilities and for validation of security solutions [24].

There are a number of testbeds, and they can mainly be categorized by the way they are implemented. There are software-based testbeds, which emulate a real SCADA environment and hardware, and physically replicated ones, which are a model of the real infrastructure and consist of real hardware. Additionally, there are also hybrids, which implement parts of the system in software [54].

Two of the most famous testbeds are the Secure Water Treatment testbed (SWaT) [23] and Electrical Power and Intelligent Control (EPIC) [2]. Both were constructed by the iTrust Center for Research in Cyber Security and are intended to be cyber security research and testing platforms.

SWaT is a physically realistic, operational model of a water treatment facility, while EPIC represents a power grid. SWaT is a hybrid model, consisting of real and simulated components. These include pumps, actuators, sensors, and control systems. It is equipped with common cyber security technologies, such as firewalls, intrusion detection systems, and access control systems [23]. The EPIC testbed is comprised of a generator, power distribution lines and smart home devices [2]. Both testbeds are designed to enable researchers to study and test the cyber security vulnerabilities and defenses of industrial control systems and critical infrastructure.

Analysis of SCADA traffic does not necessarily require specialized software. The widely used packet dissector Wireshark allows for analyzing contents of various SCADA network protocols, including Modbus/TCP and IEC-104.

Wireshark is a powerful open-source tool for analyzing and dissecting network traffic. It allows users to capture, view, and analyze datagrams traveling over a network, providing detailed information about their contents and other characteristics. It is commonly used to monitor network traffic in real-time or to analyze previously captured traffic from a file. It is able to dissect a wide variety of network protocols [20].

One of the key features of Wireshark is the ability to display packets in a human-readable format, allowing users to view the contents of each packet and identify potential transmission issues. Wireshark can decode and display the contents of TCP transmissions, for example. Moreover, it is able to analyze the sequence and acknowledgement number and flags potential retransmissions or missing packets. Wireshark also provides a range of filtering tools, allowing researchers to quickly analyze packets of a particular protocol they are interested in. Users can filter packets based on their source or destination address or based on their protocol. In addition to its core packet-capturing and analysis functionality, Wireshark also includes a range of additional tools and features. It is able to calculate network statistics such as the number of packets and bytes transmitted. Based on this information, it can generate graphs and other visualizations to examine network traffic.

The features of Wireshark can also be utilized through a command-line

application called Tshark. Like Wireshark, Tshark is also capable of capturing, dissecting, and analyzing network traffic. The main feature of Tshark is its ability to be integrated into scripts, thus allowing for automatic capture and analysis tasks [19]. Tshark can also be integrated well into Python code. Wireshark and Tshark both allow the user to read capture files and export the included frames in a raw hexadecimal format.

Crafting custom datagrams in Python code can be done using the Scapy library. Scapy is a Python-based library that allows users to craft and decode network packets. Scapy includes support for a wide range of protocols, including Ethernet, IP, TCP, UDP and many others. Like Wireshark, Scapy is able to interact with live networks. It can be used to capture packets in real time and to send packets over the network and observe the responses. Moreover, it makes it possible write custom datagrams into capture files and export them. One of Scapy's core features is the ability to craft packets at a very low level. For example, it provides an API for the creation of packets with custom headers, or packets from raw byte values in hexadecimal format [11].

2.6 Conclusion

This chapter gave an overview over the terminology and structure of industrial control systems and SCADA systems. The term industrial control system refers to a computerized system that combines electrical and mechanical components and thus interacts with the real world. These components allow computers to automatically control processes or enable manual remote operation. The term SCADA system refers to a sub-category of ICS. It is a system targeted towards processes that demand control over long distances. SCADA systems provide the ability to monitor and control processes in wide areas at a centralized station.

Moreover, the security aspects were outlined. SCADA systems are highly interconnected and mutually dependent. Security breaches or failures in single components can have cascading effects and can lead to catastrophic consequences. Since they are commonly used in critical infrastructures, availability is a very important requirement, which needs to be ensured.

Modbus/TCP and IEC 60870-5-104 represent two widely used communication protocols for SCADA systems. They are application layer protocols, built upon the TCP/IP stack. Modbus/TCP is a variant of the Modbus standard which defines communication over serial connections. However, Modbus/TCP defines a way to embed Modbus messages into the TCP payload. IEC 60870-5-104, commonly referred to as IEC-104, works similarly. It embeds messages into TCP packets that are normally sent over serial connections. IEC-104 defines 3 different frame formats called S-, U- and I-format frames. The latter one, is the one conveying measurement and control values. It is thus larger than the other ones.

Both protocols allow heterogeneous configurations, in which messages are sent over serial as well as Ethernet connections. Neither Modbus/TCP nor IEC-104 define any inherent security features or use encryption, which makes them especially susceptible to man-in-the-middle attacks. Yet, an important distinction between Modbus/TCP and IEC-104 consists in the way their communication takes place. Modbus and Modbus/TCP only allow synchronous communication, they are solely request/response based. However, IEC-104 additionally allows asynchronous communication. Field devices may send messages to the control station without a prior query. These occasions are called spontaneous events.

A number of SCADA testbeds were presented. These are models of real infrastructure which can be used to create traffic samples and allow researchers to work with real SCADA components. However, such models are complex and offer only limited access to researchers. Therefore, a software solution for the creation of synthetic traffic would be desirable.

SCADA traffic can be dissected and analyzed using standard open-source software. Wireshark and its command-line version Tshark are protocol analyzers which allow researchers to view packet contents, such as addresses or measurement values. They can also be used to convert capture files into sequences of hexadecimal byte values. This can easily be done within Python code. To forge arbitrary datagrams from within Python code, one can utilize the Scapy library, which provides an API to craft packets.

In the next chapter, the formal definitions for generative adversarial networks and their fundamentals, neural networks in general, will be given.

3 Methodology

The complex problem of generating network traffic via *generative adversarial networks* is the main subject of this thesis.

In contrast to other proposed traffic generation architectures which try to generate flow-data that has been previously extracted, the proposed generation of datagrams takes place at the raw byte-level. The network should be able to generate an arbitrary number of byte sequences represented by their hexadecimal notation. Learning the structure of byte sequences will be addressed by viewing it like the process of learning a language. The GAN will, however, only have to learn the grammar since the vocabulary will be predefined. The data pre- and post-processing will be outlined in detail in Chapter 4.

Generative adversarial networks in general and especially the architecture that will be utilized in this thesis use a large number of *machine learning* techniques. In the next sections, these techniques will be outlined concisely. Firstly, Section 3.1.1 will give a short summary of the most basic terms in the context of machine learning. Additionally, the functionality of *artificial neural networks* will be described and two classes of such networks will be formally defined, each of which is suited for a particular kind of data representation. These two classes of networks will be utilized to implement a GAN architecture in the next chapter. Secondly, in Section 3.2 the ideas of basic generative adversarial networks will be stated, before the difficulties of GANs concerning the processing of sequential data are described. Subsequently, in Section 3.3 the fundamental terms and techniques of *reinforcement learning* will be outlined, which will be utilized to overcome the aforementioned limitations later on. Finally, Section 3.4 contains background information on the GAN architecture that is later used for generating traffic. Additionally, other approaches towards generating network traffic via GANs will be presented.

3.1 Neural Network Basics

The concepts *machine learning*, and more precisely, *artificial neural networks* (ANN) form the foundation of *deep learning* and *generative adversarial networks*. Thus before describing more complex variants of these networks, the most basic terms will be defined and afterwards the fundamentals of neural networks will be outlined.

3.1.1 Machine Learning

Machine Learning was created due to the desire to create an *artificial intelligence* (AI) that is able to complete routine tasks automatically, e.g. tasks such as recognizing natural language or images and drawing conclusions based on that information to support various research fields [26]. Initially, AI was solely able to solve those problems based on mathematical formulations. Tasks like image recognition can be performed by humans without difficulties, but are hard to describe formally. This is due to the fact that a lot of human decision-making is based on huge amounts of subjective knowledge, which cannot be simply formulated in a set of rules. Computers are nevertheless able to acquire this necessary knowledge on their own. This process is called Machine Learning (ML) [26].

Different variants of Machine Learning algorithms can be characterized by the extent of the input information they are trained on. The most ordinary one is the paradigm of supervised learning. Supervised learning algorithms are commonly used for solving regression and classification problems. The term regression describes the process of predicting an arbitrary number of continuous variables, given an input vector x [13]. When used for classification, the Machine Learning model is trained on datasets containing images, speech samples or other kinds of data. These data samples are associated with a corresponding label, which describes the category the sample belongs to. During training, the model learns, for example, what a certain object looks like and tries to output a score or a probability that represents the prediction of a sample belonging to a certain category [42]. Therefore, a supervised learning model observes a predefined number of samples often described as a vector x with their associated label y and estimates $p(y|x)$. The term supervised refers to the presence of a label that instructs the model what it is supposed to to [26].

Another paradigm of Machine Learning algorithms is unsupervised learning. Just as in supervised learning, the algorithms observe a predefined number of samples. Datasets for unsupervised algorithms, however, do not contain any labeled data. A sample consists solely of a vector x. The goal of unsupervised learning algorithms can be learning a structure, more precisely a probability distribution $p(x)$, of a dataset. This process is called clustering and describes the process of dividing the samples into sets (or clusters) based on their similarity [26].

The last paradigm is a hybrid of the aforementioned ones. Semi-supervised learning uses unlabeled samples as well as labeled ones to estimate a probability $P(y|x)$. *Generative models* are a common application of this learning paradigm. They are often used for applications in which unlabeled data is available in large quantities and the labeling poses a practically unfeasible task. For such applications, generative models may be used to generate even more similar and realistic data [27] that could afterwards be used to train

another model to further improve the performance of a classifier, for example [40].

In addition to the aforementioned learning paradigms, the technique of *reinforcement learning* (RL) is often considered a different class of machine learning algorithms [70].

Reinforcement learning algorithms are not led by a given example of an optimal output like supervised training algorithms are. They are instead iterative processes targeted towards discovering an optimal action for a given situation, while trying to maximize a reward by making trial and error decisions [13]. Due to their objective being the maximization of a reward signal and not finding a hidden structure in the data distribution, RL algorithms are also not considered unsupervised algorithms [70].

The overall performance of Machine Learning algorithms depends on the input data representation as well as the representation of individual properties or pieces of information in the data [26]. These are known as *features*. Historically, Machine Learning algorithms were trained using hand-crafted feature extractors. These not only require considerable engineering skill, but also some expertise in the domain of which the dataset was taken from. Finding the right set of features to extract, however, is not trivial for every task. Enabling the algorithm to detect distinguishing features on its own was therefore desired. This process of finding the correct data representation for the task is called *representation learning*. Representation learning in general transfers the input samples into a more abstract form [26]. *Neural networks* trained by supervised learning are a realisation of representation learning [26].

3.1.2 Feedforward Neural Networks

Feedforward neural networks (FNN) are the foundation of generative adversarial networks. One possibility to outline the computational principles of neural networks is to start with the concepts of *linear regression* and classification models such as the *logistic regression*. Without going into too much detail about regression and classification models, the most basic ideas of linear regression will be outlined.

Linear Regression Models

A Regression Model works on a training data set comprised of N input samples x_n; $n = 1, ..., N$ and the corresponding N target variables t_n. It represents a function $y(x)$ that predicts the corresponding t for every new input x. More precisely, it models the predictive distribution $p(t|x)$ where x stands for every new input sample and t for the corresponding new prediction [13].

In its most basic form, such a model is comprised of a number of linear combinations of the input variables

$$y(x, w) = w_0 + w_1 x_1 + ... + w_D x_D, \tag{3.1}$$

with $x = (x_1, ..., x_D)^T$ representing the input samples in the form of a D-dimensional input vector and $w_0, ..., w_D$ representing the learned model parameters, often called *weights*. The use of linear combinations results in a so called Linear Regression Model. However, not only being linear with respect to the input variables x, but also with respect to the parameters w, leads to serious limitations [13]. To be more universally applicable, these simple models are often extended to *linear basis functions* by introducing linear combinations of a fixed set of *basis functions*. Non-linear functions of the input variables

$$y(x, w) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(x), \tag{3.2}$$

with $\phi_j(x)$ representing the basis functions and w_0 representing the bias parameter, enable the model to apply a fixed offset. The linearity with respect to the model parameters w makes these models linear regression models as well. However, this linearity allows them to have simple analytical properties, while also being able to model non-linearity as far as the input variables are concerned. The added non-linear transformation of x can be considered a new representation of x [13].

However, the simplicity in model analysis results in limitations. The basis functions $phi_j(x)$ are fixed before observing the properties of the training data set. Hence their number often grows massively, sometimes exponentially, with the dimensionality D of the input vector. This problem is commonly called the *Curse of Dimensionality* [13].

Another inconvenience in such linear models is the choice of ϕ. Historically, before the adoption of neural networks, this selection was done manually, thus requiring substantial amounts of human effort and expertise in the scientific domain of the training data [13].

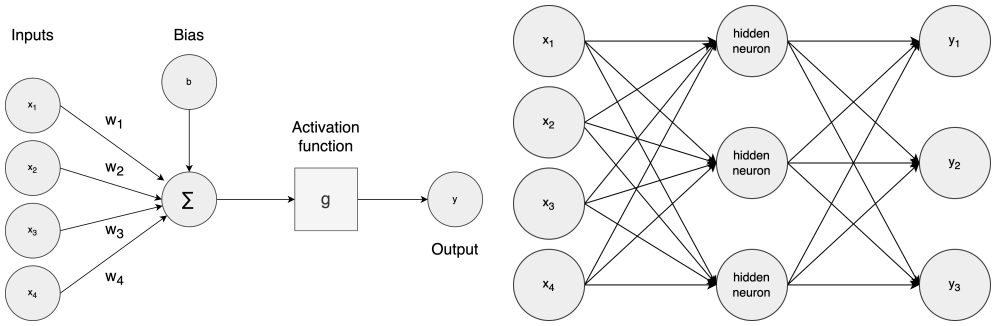
Feedforward Neural Networks

Feedforward neural networks provided a solution to the above-mentioned inconvenience by enabling a model to select an appropriate function ϕ by itself and therefore learning a representation of x. The new model [26]

$$y = f(x; \theta, w) = \phi(x; \theta)^T w \tag{3.3}$$

extends ϕ by learnable parameters θ, while w still describes the parameters mapping ϕ to the desired output. The parameters are learned via an optimization algorithm [26].

Neural networks are a composition of various individual functions. In its most basic form, this can be seen as a composition $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x))$. This function composition is the reason for the term *network*. A network's individual functions are commonly referred to as layers stacked on top of each other. Thus $f^{(1)}(x)$ would be the first layer or input layer, $f^{(2)}$ the second



(a) Model of a neuron. Taken from: [32] (b) Feedforward Neural Network

Figure 3.1: General schematic of a neuron and FNN Model

layer and $f^{(3)}$ the third and output layer [26]. Layers between the input and output layers are frequently referred to as *hidden* layers and represented by h. A function ϕ as in Figure 3.3 defines such a hidden layer. The number of layers is considered the *depth* of a model. Using multiple hidden layers is therefore referred to as *deep learning* [26].

A simple example for the computations applied in a layer would be [26]

$$h^{(1)} = g^{(1)}(W^{(1)T}x + b), \tag{3.4}$$

which defines the mapping of the input layer into the first hidden layer. The mapping of the first into the second layer is then defined by [26]

$$h^{(2)} = g^{(2)}(W^{(2)T}h^{(1)} + b), \tag{3.5}$$

and so on.

The non-linear transformation of the inputs x is controlled by the learned *weight matrix* W and a previously selected fixed *activation function* g. One common example for such an activation function is the frequently used *rectified linear unit* (ReLU) $g(z) = max\{0, z\}$ [26]. Figure 3.1(a) depicts the computations in a single unit. The term to which the activation function is applied is known as the *activation* of a unit.

The term *feedforward* refers to the computational principle that all information coming from input layer x flows through every layer and computation in between to the output layer y. In contrast to more sophisticated models which are outlined in later sections, there are no cyclic connections between a layer's output and an earlier layer. A feedforward neural network is often depicted as a directed acyclic graph (Figure 3.1(b)).

The term *neural* network is loosely inspired by the biological neurons. While both have only little in common, the neural network does resemble the brain as far as it is made up of several processing *units* or *neurons*. It gains knowledge from its environment via a learning process, the optimization algorithm, and

stores information in the neural interconnects, which are represented as *weights* [32].

Consisting of chained non-linear transformations of x with learned parameters enables the model to approximate an arbitrary function f. Feedforward neural networks with at least one hidden layer are seen as universal and efficient approximators for arbitrary non-linear functions with an arbitrary degree of accuracy [26][32].

Optimization

To be able to attain a desired design objective, for example for a supervised training objective, the corresponding output to an input sample has to have the highest score or the highest probability of all classes assigned, the network's *error* has to be as small as possible. To achieve this, the network most likely needs to be trained [42]. The error of the network is computed based on a cost function J. For supervised tasks, these take the network's output and the correct output known from the labelled dataset into account. A frequently used cost function for classification tasks presents the *negative log-likelihood*, commonly referred to as *cross-entropy* [26],

$$J(\theta) = -\mathbb{E}_{x,y\sim\hat{p}}\, log\, p_{model}\,(y|x). \tag{3.6}$$

Training a neural network means minimizing the error through adjusting the network's weights. This adjustment of the weights is done by computing a gradient vector, that describes the increase or decrease of the error after changing a weight's value. Through the process of *gradient descent*, a way of differentiation through successive applications of the chain rule, single weights get adjusted such that the error step-wise decreases [26]. Modifying the network's internal parameters based on the outputs of the network is also known as *backpropagation*. It is apparent that the complexity of the training process increases with the number of weights in the network [42].

Generalization and Regularization

The main objective of a machine learning model is not only to provide plausible outputs on training data, but rather to provide useful outputs on new data samples it has not been trained on. Techniques that reduce the error on previously unseen data at the expense of error during training are called *regularization techniques*. Making a model perform well on new inputs is commonly referred to as *generalization* [26]. These two terms are frequently used along two of the most often used terms in machine learning: *underfitting* and *overfitting*. The first term refers to the problem that a model is not able to attain sufficient accuracy during training. The latter describes a model that performs well on the training data, but bad on new unseen samples. A model's tendency to over- or underfit is controlled by it's *capacity*, a term describing

the range of various functions the model is able to approximate. As far as neural networks are concerned, the most basic way of altering the capacity is by adjusting the number of layers or units in the model. Parameters such as the number of layers, which are determined externally and not influenced by the optimization algorithm, are called *hyperparameters*. It is assumed that a neural network reaches its optimum performance, when the capacity is tailored to a certain task. If it is too complex it is prone to overfitting [26].

Dropout is one powerful example for a regularization technique with little complexity. It is often applied in so-called *dropout layers*. As the name suggests, it is used to temporarily remove, or drop, units from the computations in the network, along with its associated weights. The selection of units to remove happens randomly. The corresponding probability by which a unit is removed is a hyperparameter [67]. The dropout operation is very popular due to its low computational cost [26].

A second frequently used technique is the L^2 *parameter regularization*, also known as *weight decay* [26]. For this technique, the cost function is extended by a regularization term. The term [12]

$$\Omega(\theta) = \frac{1}{2} \sum_i w_i^2 \tag{3.7}$$

expresses a preference for smaller weights. Adding the term to the cost function encourages the weights to become smaller. This is desirable since another cause of overfitting can be weights with large values [12]. In practice $\Omega(\theta)$ will be multiplied by a hyperparameter λ, which determines how strongly smaller weights are desired [26].

A third common regularization technique is called *pooling*, a method often used in CNNs and therefore described in the next section.

3.1.3 Convolutional Neural Networks

A widely used architecture in the field of deep learning is the *convolutional neural network* (CNN), which is often used for classification problems. It is targeted towards input data that possesses a grid-like topology, such as 2-dimensional grey scale images, 3-dimensional color images or 1-dimensional time series. A convolutional neural network leverages three key ideas to reduce the number of weights in the net, while still having enough capacity to fulfill its intended purpose [26].

Convolutional Layers

A convolutional neural network is named after the performed mathematical operation, the discrete convolution. In general the convolution is an integral operation on two functions that produces a third function. It represents the

amount of overlap between the two functions as one is shifted over the other [73]. The convolution operation is defined as [26]

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a), \tag{3.8}$$

with x being an input sample and t denoting an index. The function w is often called *kernel* or *filter* and s denotes the output commonly referred to as a *feature map* [26]. Therefore, for 2-dimensional inputs, the operation would be defined as [26]

$$s(i,j) = (x * w)(i,j) = \sum_{m} \sum_{n} x(m,n)w(i-m, j-n), \tag{3.9}$$

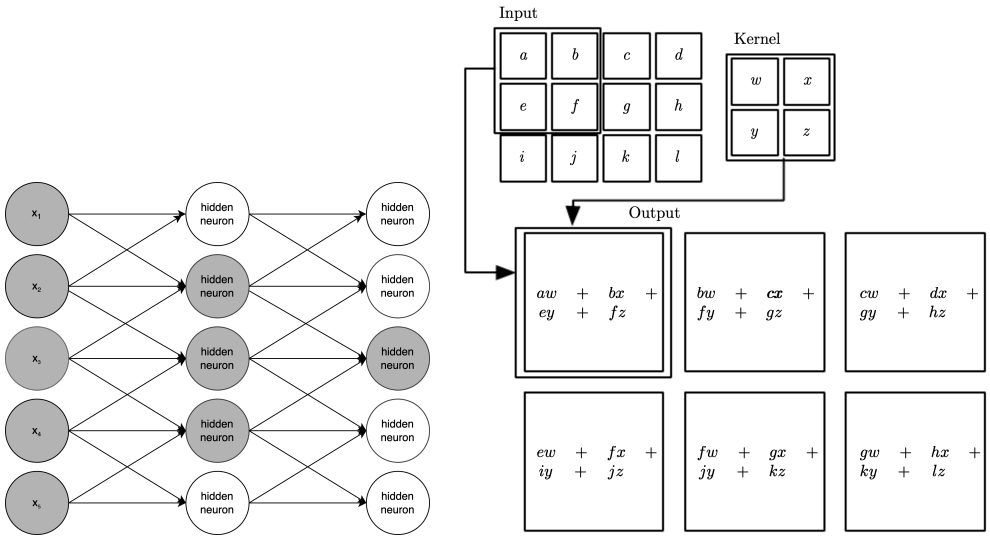
with i and j being the array indices and m, n describing the input sizes. The convolution operation replaces the plain matrix multiplication in a common feedforward network as in Figure 3.2(b). Groups of units utilizing the convolutional operation are called *convolutional layers*.

Figuratively speaking, in the convolutional layers, filters are step-wisely slid over the input as in Figure 3.2(b). Doing so enables them to extract local patches and create a feature map. The result of the convolution operation gets passed through a non-linear activation function such as ReLU afterwards [42]. The number of filters in a convolutional layer is a hyperparameter as well as their size and the step width (*stride*). A filter's size can be smaller or larger than the input's dimensionality or equally sized [26]. The filters consist of functions that utilize learned weights, which are trained during backpropagation. One filter is trained to extract one particular feature [42].

CNN Concepts

Replacing the general matrix multiplication of a common feedforward neural network by a convolution operation allows a net to utilize *sparse connectivity* in contrast to a *densely-connected* feedforward network. In a regular, densely-connected feedforward network (Figure 3.1(b)) every output neuron is affected by every input neuron. In a CNN, however, since filters may be smaller than the input, only a certain number of neurons is involved in the calculations [26].

Another more intuitive interpretation of this is the idea of the *receptive field*: It is used to describe a field of view of a neuron. As far as 1-dimensional inputs are concerned, the receptive field describes which particular part of an input sample, for example a time series, a neuron takes into account [46]. The neurons of deeper layers have a larger receptive field than the ones in shallower layers, which enables them to recognize longer patterns in the time series. In other words, the receptive field describes the number of input samples a neuron in deeper layers is able to see, or which number of input samples affects a deeper neuron (Figure 3.2(a)). Therefore, by increasing the depth of a neural



(a) Sparse Connectivity and Receptive Field

(b) Convolution Operation. Taken from: [26]

Figure 3.2: Sparse Connectivity, Receptive Field and the Convolution Operation

net, it is possible to recognize larger discriminatory features. Alternatively, the use of larger filters in the convolutional layers is another way of increasing the receptive field. Normally heterogeneous filter sizes are used. This is motivated by the idea that salient parts of an image or a time series may vary in scale. Using larger filters allows for recognizing more global features. However, smaller filters contribute less to overfitting. All in all, sparse connectivity has the advantage that fewer weights have to be trained and stored, thus reducing the amount of required computational resources and the risk of overfitting [26].

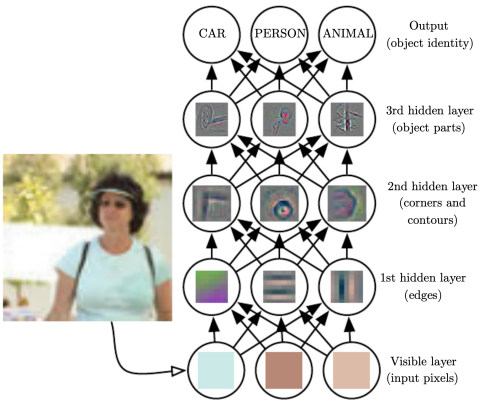


Figure 3.3: Hierarchical Feature learning in a DL Model. Taken from: [26]

The second core idea of a CNN makes use of a deep neural networks manner of feature learning: Many signals such as images, sounds or speech can be

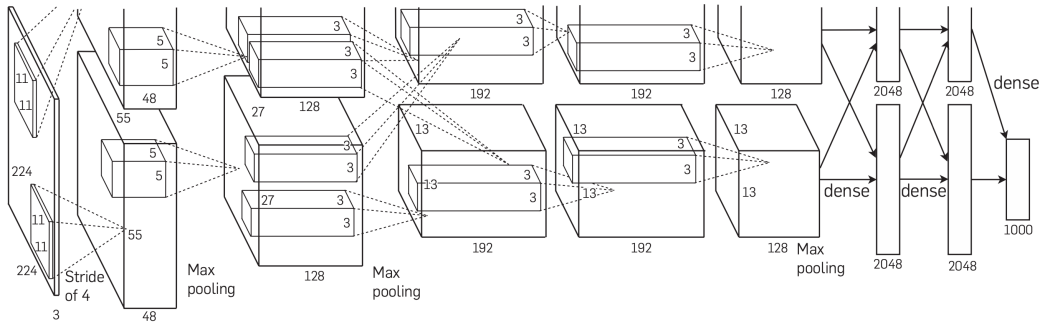


Figure 3.4: Typical CNN Architecture ("AlexNet"). Taken from: [41]

interpreted as hierarchically structured (Figure 3.3). An object in an image is composed of edges and corners, a sentence is organized in multiple words which are structured into one or more syllables. A high level feature, like a sentence or an object, is therefore a composition of multiple lower level ones [42]. Since lower level features may exist multiple times and at different locations in an input, the filter creating the feature map for the corresponding lower level feature may detect the feature in different parts of the input. Reuse of the filter and its weights therefore reduces the number of required weights and the computational complexity of the network [42]. This idea is often referred to as *weight sharing* or *parameter sharing* [26].

The third main concept of CNNs are the *pooling layers*. Since the exact position or size of a feature is not as important for classification purposes as the existence of a feature, a certain invariance to small transformations such as shifts, scale or distortions is desirable [42]. This is achieved by a pooling layer which commonly merges together very similar features by computing the maximum (max pooling) or the average (average pooling) of local patches in a feature map, therefore reducing the impact of small variations in the input. This operation additionally decreases the input's dimensionality in height, width and length, thus decreasing the number of weights, computational cost and risk of overfitting [42].

In addition to these mentioned key ideas that motivate the use of a convolutional neural network, there is another main advantage. CNNs are able to be parallelized effectively and utilize the capabilities of modern GPUs for training [42]. Until the late '90s it was doubted that efficient training of neural networks was feasible. However, in 2012 one of the most famous CNN models, AlexNet (Figure 3.4) proved that CNNs and therefore deep neural nets were efficiently trainable and generalized better than traditional neural networks with layers of fully-connected neurons [41]. Since then, deep learning has gained a large boost in success [42].

CNN Structure

Usually a convolutional neural network is made up of groups consisting of several layers (Figure 3.4). These groups will be referred to as *modules*. A module's first layer consists of a convolutional layer, extracting features from the input. This layer's feature map is afterwards fed into an activation function, usually ReLU. Then the output of the activation function gets fed into a pooling layer, making the net invariant to slight shifts and distortions in the input [42]. Therefore a module is normally comprised of a convolutional layer, an activation function and a pooling layer. Regularization layers such as dropout may be added. Depending on the desired depth or capacity, several modules are stacked on top of each other. After one or more modules, the feature maps are fed into the last layers of the CNN, which are typically fully-connected or *densely-connected* ones.

The last layers of a CNN are usually arranged as a traditional, densely-connected feedforward network, in which each neuron of a layer is connected to each neuron of the next layer [42]. Since CNNs are often used for classification tasks, the output of the densely-connected layers is usually fed into a so-called *softmax* layer [41] which utilizes the softmax activation function [26]

$$softmax(x)_i = \frac{exp(x_i)}{\sum_{j=1}^{n} exp(x_j)}. \tag{3.10}$$

A softmax layer is comprised of a number of neurons, equal to the number of classes in the dataset. Thus it enables the network to output a probability distribution over every class in the dataset.

3.1.4 Recurrent Neural Networks

While Convolutional Neural Networks are specialized feedforward models for values that are ordered in grid-like topologies, *recurrent neural networks* (RNN) [60] are a family of neural networks specialized in processing sequential data such as sentences of natural language [26]. A common area of application is *statistical language modelling*. SLM often utilizes a form of data representation particularly useful for data like words or letters. Additionally, it is the origin of further terminology that will be used later. Thus, the key points of statistical language modelling will be outlined shortly, before the functionality of a RNN is decribed in detail.

Statistical Language Modelling

Statistical language modelling is a subcategory of *natural language processing* and refers to machine learning models that are targeted towards estimating the probability distributions of components of natural language, such as word, letters or sentences. A specific problem to solve may be predicting which word

would have to appear next in a sentence after a particular sequence of words was observed. This is commonly used for text generation.

Neural networks by default do not understand natural language, but instead require that words are converted into a numeric representation. A common approach towards encoding categorical data into discrete vectors is the usage of one-hot encoding: In the context of SLM one would create a vector with equal length to the number of unique words in the dataset and fill it with zeroes. Now, to represent a specific word a zero would be replaced by a one at the index of the corresponding word [28]. This encoding scheme, however, is sparse and not useful when dealing with larger amounts of categorical data since the created vector would consist mostly of zeroes. Moreover, it does not capture any information about the relations between words. Models used for SLM often have to deal with datasets containing large numbers of structured sequences of text, which are called *corpus* or *corpora*. In this context, a single unique word is commonly called *token*, while the entirety of all unique words that occur in a dataset is referred to as the *vocabulary*.

An important form of representation often used in conjunction with SLM are *word embeddings*. These describe a dense representation of every item in a vocabulary by a real-valued, m-dimensional feature vector, with m being arbitrary, but fixed and much smaller than the size of the vocabulary. Word embeddings are obtained by training a conventional feedforward network to estimate the probability that a certain word will appear in a context, such as a sentence, given a sequence of words taken from a corpus [52]. The embedding feature vector is a by-product of training and consists of the network's weights. The vector of a certain word contains the m weights mapping the input word to the output probability [9].

The important characteristics of such word embeddings are that they present a comparatively dense representation, which reduces the dimensionality considerably. Moreover, they encode linguistic information. Semantically similar words are spatially close to each other, without the need for human assistance or domain knowledge [15]. The characteristic of semantically similar words being clustered together is especially useful when dealing with generative problems. It aids the model in hitting a semantically correct word.

The embeddings dimensionality m is a hyperparameter, larger numbers enable the embeddings to capture finer nuances in the semantics, but require more training and training data [28]. In practice, if one needs to use word embeddings in a neural network, this is mostly achieved by an *embedding layer* which can be understood as a lookup table converting words into continuous vectors. The embedding layers of tensorflow expect an integer encoded vocabulary as input and looks up the corresponding feature vector for each word index. The elements of the vectors are randomly initialized and become adjusted during training [28].

RNN Concepts

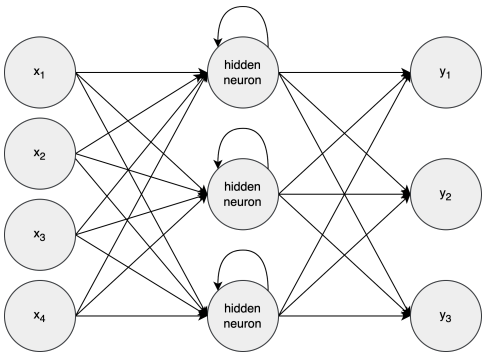


Figure 3.5: Recurrent Neural Network. Taken from: [52]

Conventional densely-connected feedforward networks may serve as universal function approximators, however, they are subject to serious drawbacks if it comes to modelling sequences of ordered values. Take, for example, a simple regression problem: In order to predict the next value of a sequence, the only information the feedforward network can work on are the latest observed input samples. Any previously seen samples are forgotten. If it comes to modelling languages that follow some sort of grammar, dependencies between words have to be taken into account. These dependencies could mean that the first and the last word of a longer sentence are significant for the prediction. Therefore to be able to make an accurate prediction for longer sequences of data, some form of memory is necessary [52].

Recurrent neural networks possess a form of inherent memory which is often also referred to as "short-term memory" [33], for reasons that will be discussed later in this section. As the name suggests, RNNs differ from traditional feedforward networks by containing at least one layer of units with *recurrent connections*. These connections are referred to as feedback connections that connect the output of a layer of neurons to its inputs (Figure 3.5) [32].

Similar to CNNs, recurrent neural networks utilize a form of parameter sharing to reduce the overall number of weights, although this time temporal and not spatial [26]. To facilitate the understanding of this temporal parameter sharing, a closer look at the computational graph of an RNN can be useful.

RNN Structure

As mentioned before, recurrent neural networks are specialized in processing sequential data. Input samples, for example words, are denoted as vectors $x^{(t)}$ with an index t ranging from 1 to τ. While not necessarily representing "time", the index is commonly referred to as *time step*. A sequence is therefore denoted as $x^{(1)}, ..., x^{(\tau)}$ [26].

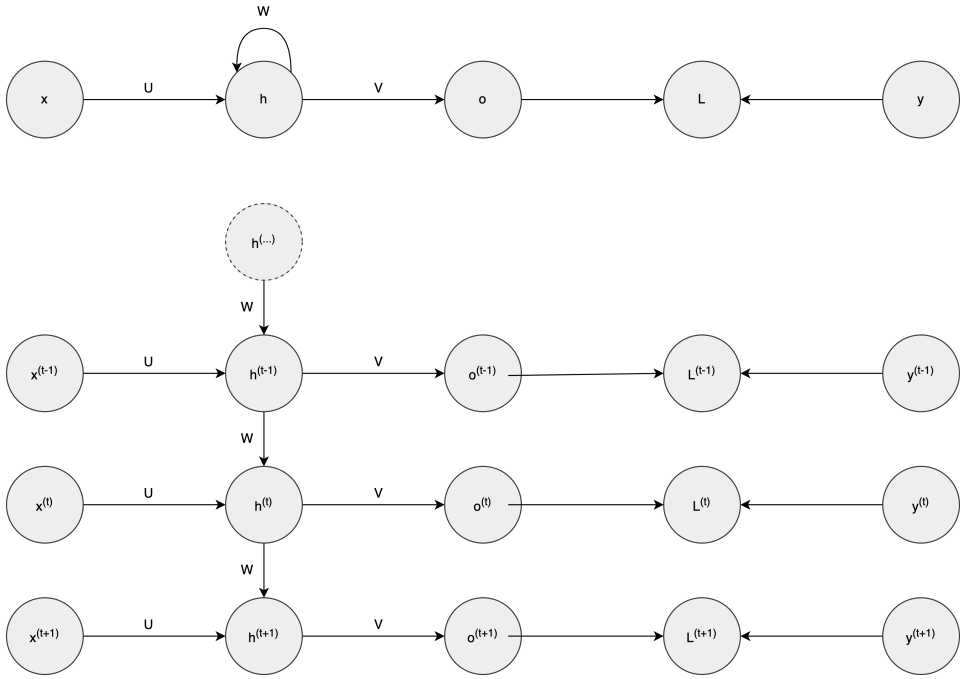


Figure 3.6: Unfolded Recurrent Neural Network. Taken from: [26]

The aforementioned inherent memory is realized through hidden layers. In contrast to a conventional feedforward network, previously observed input samples at time step $(t-1)$ become incorporated into the computation of the current time step. A RNN's hidden layer can be defined by [26]

$$h^{(t)} = f(h^{(t-1)}, x; \theta). \tag{3.11}$$

Since the hidden units are computed based on the previous time step's hidden units, they represent the models current state. This state is often referred to as the *hidden state*. For regression tasks, the model is trained to utilize this hidden state as a summary of the previously observed inputs [26]. RNNs can be illustrated in two ways: 1) as a traditional neural network with inserted recurrent connections and 2) as an unfolded computational graph. The depicted RNN in Figure 3.6 predicts an output on every time step and possesses a recurrent connection from its hidden unit to itself. The upper illustration depicts a traditional computational graph with a recurrent connection. The lower one depicts the corresponding unfolded graph. The RNN in Figure 3.6 is formally defined by [26]

$$h^{(t)} = tanh(\, b + Wh^{(t-1)} + Ux^{(t)}\,), \tag{3.12}$$

$$o^{(t)} = c + Vh^{(t)}, \tag{3.13}$$

$$\hat{y}^{(t)} = softmax(\, o^{(t)}\,), \tag{3.14}$$

$$L = \sum_t L^{(t)} = -\sum_t log\, p_{model}\, (y^{(t)}|\{x^{(1)}, ..., x^{(t)}\}). \tag{3.15}$$

The hidden layers apply two bias vectors b and c and learn two conventional weight matrices U and V. Additionally, they learn a newly introduced weight matrix W, representing the recurrent connection. The function $tanh(x)$ represents the *hyperbolic tangent* activation function [52]

$$tanh(x) = \frac{exp^x - exp^{-x}}{exp^x + exp^{-x}}. \tag{3.16}$$

L describes the total loss for an input sequence $x^{(1)}, ..., x^{(\tau)}$ and the corresponding predictions $y^{(1)}, ..., y^{(\tau)}$. The depicted RNN creates an output of the same length as the input [26].

RNN architectures can be very versatile, they allow fixed as well as variable size inputs and outputs. Moreover, RNNs are able to produce an output at either every time step or only after the last step. The recurrent connections can be between hidden neurons, while other variants insert recurrent connections between an output and the next time step's hidden neurons. The RNN in Figure 3.6, however, is the most powerful and representative [26].

As can be seen, the parameter sharing manifests itself in the fact that the learned weight matrices U, V and W as well as the biases are repeatedly involved in every new time step's computation. Therefore no matter how long the input sequence may be, the overall number of parameters in the network will not increase. This leads to the main advantage of recurrent networks in contrast to feedforward ones: RNN models, in theory, are able to generalize to arbitrarily long sequences, independent of those in the training dataset [26].

In practice, however, they are not able to process arbitrarily long contexts [52]. This is mainly due to the definition of the hidden states in an RNN. Even for rather short input sequences, the multiple state transitions quickly lead to very deep computational graphs. Furthermore, applying the same mathematical operations with the same parameters is especially problematic [26].

Vanishing and Exploding Gradients

Deep neural networks may be subject to the *vanishing* or *exploding gradient problem*. This is in particular true for neural networks that repeatedly apply the same mathematical operations, such as recurrent neural networks.

Take the weight matrix W used for the recurrent connections between hidden layers. The variable W represents the network's inherent memory and is utilized each time the hidden state is rewritten. Then W^t represents iterative multiplying by W for t time steps. The eigendecomposition of W is defined by [26]

$$W^t = (V diag(\lambda) V^{-1})^t = V diag(\lambda)^t V^{-1}. \tag{3.17}$$

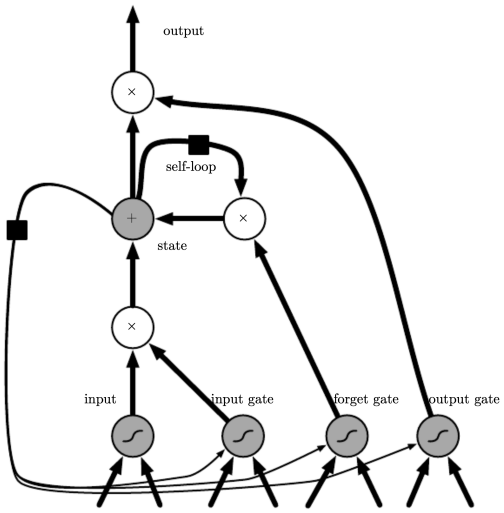


Figure 3.7: LSTM Cell. Taken from: [26]

After repetitive computations, any eigenvalues λ_i smaller than 1 will become very small, which is referred to as *vanishing*, or, if they are greater than 1 become very high and thus *explode* [26].

During backpropagation the gradient of the considered computational graph will scale with $diag(\lambda)^t$. A vanishing gradient will then make it difficult for the optimization algorithm to decide on how to further improve the cost function.

A much rarer exploding gradient will, however, hamper the optimization by possibly taking too large steps and therefore missing the optimum [26].

Ultimately in an RNN, the vanishing and exploding gradient manifests itself in the handicap that standard RNNs are not entirely fit to learn long-term dependencies in sequences [26]. Sequences with a length of around 10 to 20 may already pose a problem [10]. This results in RNNs being considered as possessing a "short-term memory" [33]. To be able to learn longer dependencies, several improvements have been proposed. One popular enhancement presents the *long short-term memory*.

Long Short-Term Memory

The *long short-term memory* (LSTM) [33] presents an implementation of a *gated RNN*. These are RNNs that are able to extract useful information from sub-sequences contained in a context, memorize them and learn to decide which information can be forgotten [26]. An LSTM replaces the conventional approach of storing all information in an additional weight matrix, by a more sophisticated mechanism [52].

LSTMs are made up of an arbitrary number of recurrently connected subnets, the so called *memory blocks* or *cells*. Each of these contains three different parameterized, multiplicative units, the *gates*, allowing the network to take

control of the information flow [52][26]. The first of these is the *input gate unit*: its purpose is to protect the memory contents from disturbances by currently irrelevant inputs. It can be viewed upon as a mechanism to control read access to the input. The second gate is the *output gate unit*: Its purpose is similar to that of the input gate as it is used to protect other cells from currently irrelevant memory contents [33]. Thus it may be seen as a control mechanism for write access. The last gate is the *forget gate unit*, which enables the reset of a cell's state and thus enables the RNN to forget information [52]. Figure 3.7 depicts a LSTM cell, with an optional parameterized feedback loop from the *state unit* to the gates. The black square on the loops denotes a delay of one time-step.

Without consideration of the optional feedback loop, the LSTM in Figure 3.7 cell may be defined by [26]

$$f_i^{(t)} = \sigma(\, b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\,), \tag{3.18}$$

$$g_i^{(t)} = \sigma(\, b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)}\,), \tag{3.19}$$

$$h_i^{(t)} = tanh(\, s_i^{(t)}\,)\, q_i^{(t)}, \tag{3.20}$$

$$q_i^{(t)} = \sigma(\, b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)}\,). \tag{3.21}$$

The respective memory cell is denoted by i; the function $f_i^{(t)}$ defines the forget gate of cell i at time-step t. The corresponding input gate is defined by $g_i^{(t)}$, and $h_i^{(t)}$ denotes the cell's output gated by $q_i^{(t)}$. All gate units utilize the *sigmoid* activation function σ as defined by [52]

$$sigmoid(x) = \frac{1}{1 + exp^{-x}}. \tag{3.22}$$

The central building block of the LSTM is the *state unit* $s_i^{(t)}$, resembling the hidden units in a conventional RNN [26]

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma(\, b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)}\,). \tag{3.23}$$

As can be seen, every gate possesses its own bias terms and weights that will be learned during optimization. The LSTM can therefore decide which information is important and needs to be stored for future use. The interaction between input and output gate allows the LSTM network to keep information from the beginning of a sequence and to apply it at a later time-step. This can be done by closing and opening the input or output gates, respectively. Computationally closing the input gate refers to an activation close to 0, while opening an output gate is realized through an activation closer to 1 [52].

LSTM models have been proven to enable neural networks to learn dependencies in long contexts and have become a state-of-the-art technique for statistical language modelling [26][69].

It is important to note, since it will be used in the next chapter, that the concept of LSTMs, the learning of gating mechanisms, was applied in another technique of neural networks. The gating mechanisms were ported to conventional feedforward and convolutional neural nets, thus creating so called *highway networks*, which introduced a way to improve the optimization of very deep network architectures [68].

3.2 Generative Adversarial Networks

Now that we have established our basic equipment to solve regression and classification problems using neural networks, we can draw our attention towards generative models, particularly *generative adversarial networks*.

3.2.1 Deep Generative Models

Generative models are often utilized for specific tasks where rather large quantities of unlabeled data are accessible and the process of labeling would be practically unfeasible. These generative models would then be utilized to generate similar, realistic data [27] that, for example, could be used as training data for a classifier [40]. This data could essentially be of any kind, though, it has to be data for which a real-valued data representation is available. Traditionally, generative models are often used for image generation [26].

The term *deep generative model* refers to the implementation of a generative model using neural networks [40]. The task of a deep generative model may be formulated as a stochastic transformation of a random variable into another, more complex, arbitrary distribution. Typically, the transformation is applied to an input vector z which has been sampled from a less complex probability distribution, such as a uniform or normal distribution.

The transformation is implemented by a neural network that takes z as input and learns to output a vector x that underlies the desired target probability distribution. Though a conventional neural network presents a deterministic function, the randomly sampled z makes the transformation stochastic. Moreover, since the actual values of z are left aside and only its probability distribution is relevant in most cases, it is referred to as a *latent variable* or *latent vector*. A deep generative model is trained via backpropagation like any standard neural network. [26].

The *differentiable generator network* is a variant of a deep generative model and represents the foundation of the generative adversarial network. These networks contain a *generator* model, which is a neural network model that implements the aforementioned stochastic transformation by approximating a

differentiable function $G(z; \theta^{(g)})$.

After it has been trained, the generator can be utilized as a computational procedure that outputs an arbitrary number of samples x with a desired target distribution. The application of a neural network to approximate $G(z; \theta^{(g)})$ has the same advantages as for basic regression or classification tasks: The transformation function can be learned without manual intervention, even for much more complicated probability distributions [26].

There is, however, a downside to such generative models: Neural networks used for basic supervised regression or classification tasks are viewed upon as universal instruments with a high chance of success. This is not the case for deep generative models. With a conventional training task for regression or classification the input x, as well as the expected output y, are well known. However, in generative modeling the network has to learn the distribution of a z and its mapping to x. As, z is not known ahead of time, this makes the training process much more difficult [26].

3.2.2 Adversarial Learning

Generative adversarial networks by default couple their generator network G with another neural network called *discriminator* network D. The latter one aids G in learning the aforementioned differentiable function $x = G(z; \theta^{(g)})$ that is used for generating samples.

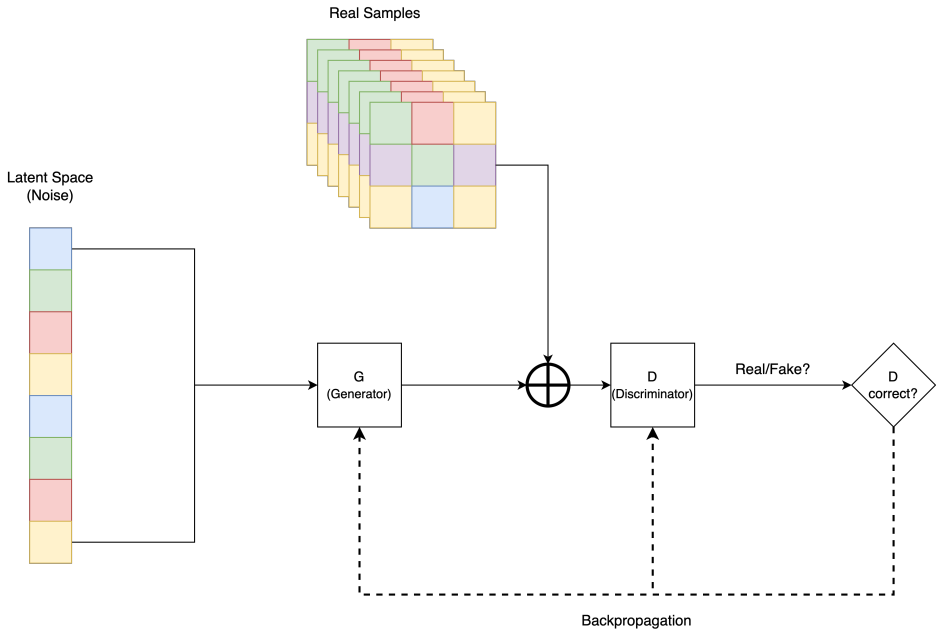


Figure 3.8: Generative Adversarial Network for 2D Image Data. Based on: [3]

The discriminator is a classifier, that learns a function $D(x; \theta^{(d)})$ which assigns a probability value to each sample x it is fed. The vector x may either be

a sample from the training dataset or a sample generated by g. The assigned probability value represents the probability of x being from real training data rather than being synthetic [26].

The training objective of a GAN is often stated as a two-player mini-max game between the generator G and the discriminator D. This is due to the idea that D is trained to maximize the chance to assign a correct probability value to the sample x it is fed. This corresponds to recognizing a sample from training data as real and a sample from G as synthetic. At the same time, G is trained to minimize $log\ (1 - D(G(z)))$. Therefore, it tries to minimize the probability that it's own samples are declared synthetic by the discriminator [27].

At optimum, the generator is able to fool the discriminator to the extent that it cannot distinguish between real and synthetic anymore. This corresponds to assigning the generated samples a probability value of 0.5 [26].

The training objective of a generative adversarial network is defined by the value function V [27]

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[log\ D(x)] + \mathbb{E}_{z\sim p_z(z)}[log\ (1 - D(G(z)))]. \tag{3.24}$$

Algorithm 1 Standard training algorithm of a GAN. Taken from: [27]

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from random noise with distribution $p_g(z)$.
- Sample minibatch of m samples $\{x^{(1)}, \ldots, x^{(m)}\}$ from training data with distribution $p_{data}(x)$.
- Update Discrimininator by **ascending** it's gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [log\ D(x^{(i)}) + log\ (1 - D(G(z^{(i)})))]. \tag{3.25}$$

end for

- Sample minibatch of m noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from random noise with distribution $p_g(z)$.
- Update Generator by **descending** it's gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} log\ (1 - D(G(z^{(i)})). \tag{3.26}$$

end for

The training algorithm is an iterative process. The discriminator is optimized in turns by being fed a number of real data samples and synthetic sam-

ples. Afterwards the generator's parameters are optimized using the newly updated discriminator. The number of optimization steps for the discriminator is denoted as k and presents a hyperparameter [27]. Algorithm 1 shows the training algorithm in detail.

3.2.3 GAN Properties

The application of GANs is motivated by the usual advantages of neural networks. There is seldom a need for deeper scientific domain knowledge as their learning algorithm is solely based on backpropagation. They are somewhat straightforwardly applicable and only require appropriate pre-processing. Moreover, they are able to model even more complex probability distributions, in contrast to other generative models [26].

Generative adversarial networks, however, also have downsides. They can be difficult to train and are highly sensitive to the choice of hyperparameters [26]. These difficulties manifest mainly in two problems, the first one being that GANs tend to underfit. This is because of the training objective being the simultaneous gradient descent/ascent of generator and discriminator. Reaching an equilibrium is not guaranteed for this algorithm [26].

The second problem is the so called *mode collapse*: The conventional GANs necessitate a well balanced training process between generator and discriminator. Omitting the optimization of the discriminator while updating the generator may result in a generator that is prone to sampling the same values over and over. The model therefore loses diversity [27].

The first approach towards the stabilization of GAN training was given by the authors of [27] themselves. They reformulated the optimization objective for the generator. The new goal is to maximize the log probability for the discriminator to wrongfully classify a synthetic sample as real (Equation 3.27).

$$\max_G V(G) = \mathbb{E}_{z\sim p_z(z)}[log\,(D(G(z)))]. \tag{3.27}$$

Additionally, the authors of [27] found out that the utilization of dropout layers is very important in GANs and improves the network's performance [26].

Many enhancements of the traditional GANs have been proposed; one of the most popular may be the *Wasserstein GAN* (WGAN) [5] in which the training objective is replaced by the *Earth Mover* (EM) distance or *Wassserstein* metric as an alternative value function [30]. The EM distance is often depicted as "the minimum cost of transporting mass" that is needed to transform one probability distribution into another [30]. The new value function is then defined as [30]

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[D(x)] + \mathbb{E}_{z\sim p_z(z)}[D(G(z))]. \tag{3.28}$$

The discriminator in a WGAN is referred to as *critic* due to the fact that it evaluates the distance between two probability distributions. In contrast to

the standard GAN, where training between generator and discriminator needs to be balanced, in a WGAN, the critic, can simply be trained to optimality. On top of that, the possibility of a mode collapse is greatly reduced [5].

Another way of mitigating the issues of mode collapse and instability in training, is the introduction of a pre-training step in the GAN training algorithm. The pre-training consists of estimating the probability density of the real data samples and adapting the generator's weights accordingly. This is done via *maximum likelihood estimation* as defined by [21]

$$arg\,\max_{\theta} \mathbb{E}_{p_{data}(x)}\, log\, p_\theta(x). \tag{3.29}$$

The utilization of a pre-training step is especially common in language modeling tasks [21].

Sequence Generation

The traditional GAN in [27] simply comprised two fully-connected feedforward networks. Yet, just like for regression or classification tasks, certain architectures have been proven to work better for some tasks than others. As stated at the beginning, the main subject of this thesis, the generation of SCADA network traffic, will be interpreted as learning a language, since the datagrams will be represented as hexadecimal byte sequences. Thus, a generative adversarial network has to learn how to arrange single bytes such that they equal the datagrams of a particular network protocol.

Convolutional neural networks have been utilized as generator and discriminator models for image generation tasks and produced realistic looking images [55]. In theory, one could also implement the generator and discriminator using recurrent neural networks [25]. However, as GANs are defined for sampling real-valued data, the modelling of discrete probabilistic models, like the generation of natural language using GANs, poses a more challenging task [35].

The main problem in natural language processing via GANs lies in the optimization using backpropagation: During training the generator produces a number of synthetic samples. Afterwards, the discriminator decides whether these are real or synthetic and outputs a probability value. The error is then computed and backpropagated, which instructs the generator network to gradually adjust its weights. This gradual adjustment makes perfect sense, for example, for image data. It would mean that the grey level of a pixel gets slightly darker or lighter. However, a gradual adjustment for discrete valued samples such as sentences or words is not clearly defined [25].

Another difficulty lies in the fact that the optimization process of a standard GAN would take a whole generated sequence into account [77]. This means that, no matter how grammatically good, for example, the beginning of a generated sentence was, the network's parameters would be adjusted based on the whole sentence.

While [30] were able to produce English sentences using a WGAN, the results contained a large number of spelling errors and did not capture a lot of semantics. The authors of [6] and [7] therefore proposed to view the generation of a sequence as a sequential decision-making process to improve existing models. It was possible to implement this sequential generation process using techniques from reinforcement learning, which was also already suggested by [25].

The application of reinforcement learning in conjunction with a generative adversarial networks has additional benefits. It not only poses a means to guide the network to create samples that correspond to an underlying probability distribution. It also introduces new means to optimize the sampling process towards additional criteria, such as to comply to a certain grammar or protocol, for example, by including external reward signals into the objective function [29].

3.3 Reinforcement Learning Basics

To succeed in generating byte sequences, a special kind of generative adversarial network architecture will be used that uses techniques, which originate from the machine learning paradigm of reinforcement learning. The utilized architecture is a modified version of the aforementioned traditional GAN and models the sequence generation as a sequential decision-making process.

These processes can be formalized as a *markov decision process* (MDP) [70]. Therefore, in this section the basic principles and terminology of RL and MDP will be outlined first. Afterwards, the concept of *policy gradient methods*, which are utilized for the traffic generation process, will be defined.

3.3.1 Reinforcement Learning Definition

Reinforcement learning (RL) may be considered as a specific machine learning paradigm that differs from both supervised and unsupervised learning algorithms. Supervised learning algorithms offer the model a ground-truth as training data. Reinforcement learning algorithms on the other hand, are trained a posteriori on the information how good an action was by instructing it what correct behaviour looks like.

This guidance is also the reason why RL is not considered unsupervised learning. RL algorithms represent a computational approach towards learning from feedback after performing an action. This can be formalized as a finite markov decision process [70].

Markov Decision Processes

Reinforcement learning algorithms learn through interaction, in the form of sequential decision-making. They interact with an environment in discrete steps

and receive evaluative feedback afterwards. These interactions may influence immediate as well as future rewards and also subsequent situations [70].

Such processes are commonly formalized as a finite markov decision processes. The terminology of MDPs will be defined in the following.

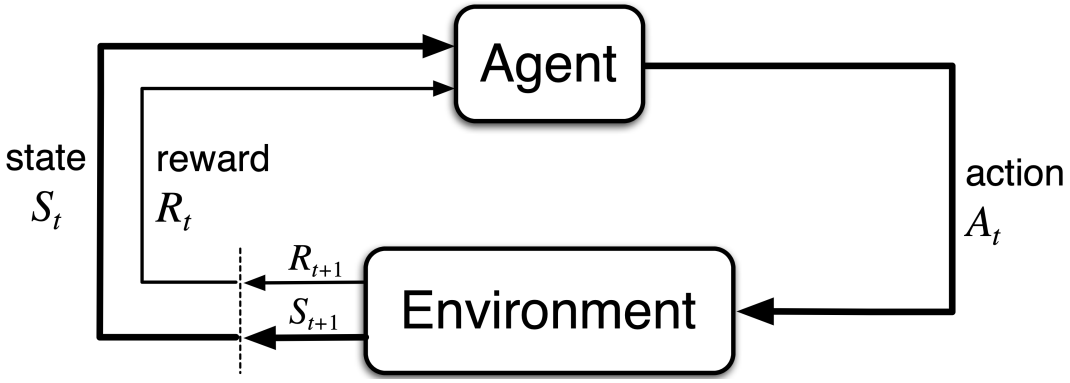


Figure 3.9: Interaction between Agent and Environment in MDP. Taken from: [70]

Environment Model

An MDP contains some form of model which represents an environment that reacts to actions that are taken. This model of the environment allows to estimate the outcome of a taken action by predicting consequent states and rewards before the outcome was actually experienced. The model is often simply referred to as *environment* [70].

Agent

The entity that is interacting with the environment and making decisions is called the *agent*. The interactions between agent and environment (Figure 3.9) take place in discrete sequential time-steps $t = 0, 1, 2, 3, \ldots$. At each of these time-steps the agent selects an *action*, $A_t \in \mathcal{A}(s)$, given a particular *state*, $S_t \in \mathcal{S}$, that it received from the environment.

Afterwards, at time step $t+1$, the agent receives a numerical *reward*, $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, and transitions into state S_{t+1}. The long term goal of the agent is the maximization of the total, cumulative rewards [70].

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots \ . \tag{3.30}$$

The sequence of transitions is referred to as *trajectory* (Figure 3.30).

In the case of a finite markov decision process, the sets $\mathcal{A}, \mathcal{S}$ and $\mathcal{R}$ are finite and the random variables S_t and R_t underlie a discrete probability distribution. Particular values $s' \in \mathcal{S}, r \in \mathcal{R}$ occur, given particular values for the preceding

action and state, at time step t with a probability distribution p [70]. This distribution is defined by [70]

$$p(s', r|s, a) \doteq Pr\{S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a\}, \tag{3.31}$$

for all $s', s \in \mathcal{S}, r \in \mathcal{R}, a \in \mathcal{A}(s)$.

The distribution p is called *dynamics* of a MDP and specifies a probability distribution for each combination of a and s. For the distribution

$$\sum_{s' \in \mathcal{S}} \sum_{r' \in \mathcal{R}} p(s', r|s, a) \; = \; 1, \forall s \in \mathcal{S} \; and \; a \in \mathcal{A}(s) \tag{3.32}$$

applies [70].

Rewards

The agent is led by a *reward* signal, which it receives from the environment. It represents a measure of how good a selected action was. The intended objective of an agent is to maximize the expected value of the sum of rewards, the *expected return*.

In environments which offer a well defined final time step T, for example the last move in a game round, the modelled task is called *episodic*. A sequence $R_{t+1}, R_{t+2}, R_{t+3}, \dots$ is then called *episode* and its last state is the *terminal state*. The *return* G_t after time step t is defined as [70]

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T. \tag{3.33}$$

When there is no such well defined final time step, in the case of a *continuing task*, the concept of a *discounted return* is applied. The agent would seek to maximize the expected discounted return [70]

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{3.34}$$

with γ denoting the *discount rate* and $0 \leq \gamma \leq 1$. The discount rate is a measure for the current value of a reward that will be received in the future [70].

Value Function and Policies

Value functions enable the agent to measure the expected return, given a state or alternatively a state-action pair. Since the expected rewards are dependent on the actions taken, value functions are defined based on a certain behaviour of the agent. This behaviour is defined as mappings from states to the probabilities of selecting each of the possible actions in that state [70].

Such a mapping is formally referred to as *policy*. Policies are denoted as π and $\pi(a|s)$ denotes the probability for $A_t = a$ and $S_t = s$ at time step t. It is

important to note that policies may change, which ultimately represents the learning of RL algorithms [70].

The value function in a markov decision process, given a state s and policy π, is called *state-value function for Policy* π. It is defined by [70]

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] \;=\; \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle|S_t = s\right], \forall s \in \mathcal{S}. \tag{3.35}$$

The function $v_\pi(s)$ is therefore defined as the expected return, when the agent is in state s and follows policy π from there.

The value function, given a state-action pair (s, a) and policy π, is called *action-value function* for policy π. This function is defined by [70]

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \;=\; \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}\middle|S_t = s, A_t = a\right]. \tag{3.36}$$

The function $q_\pi(s, a)$ is then defined as the expected return, when the agent is in state s, performs action a and follows policy π from there [70].

3.3.2 Policy Gradient Methods

Now that the basic RL terminology is defined, we are able to outline the technique, that will later be used to enhance the generative adversarial network's training algorithm. The problem so far was, that a gradual adjustment is not well suited for discrete outputs. To circumvent this, the task is to model the generation of a sequence as a sequential decision-making process. This will be implemented through a policy that is stochastically parameterized [77].

The previously defined policies π can be continuously parameterized and thus become $\pi(a|s, \theta)$. Such a policy is defined as $\pi(a|s, \theta) = Pr\{A_t = a|S_t = s, \theta_t = \theta\}$, therefore describing the probability of selecting action a at time step t, given state s and parameters θ.

In practice these policies are realized using a function approximator such as a neural network, which receives a representation of a state s as input and outputs probabilities for each possible action. The network's weights are then denoted as θ [71].

The advantage of a parameterized policy is that it can be gradually optimized. Moreover, the optimization can be done using the optimization methods for neural networks that were established before. Thus, the algorithm is enabled to approach the desired output probabilities for the possible action in a state, step-by-step.

Such policies will be optimized on the basis of a performance measure $J(\theta)$. The objective is defined as to maximize the performance via gradient ascent in J: [70]

$$\theta_{t+1} = \theta_t + \alpha\widehat{\nabla J(\theta_t)}, \tag{3.37}$$

with α being a step size parameter and $\widehat{\nabla J(\theta_t)} \in \mathbb{R}^d$ being an estimate. Its expectation approximates the gradient of J.

Methods that utilize this concept are called *policy gradient methods* [70]. These policy gradient methods are able to model continuous action spaces as well as discrete ones. Yet, since our GAN will have to deal with discrete valued data, the given definition will be limited to discrete action spaces. Moreover, continuing and episodic tasks each have separate definitions for their performance measure. However, we will only need the episodic definition in the next section. Thus, only the episodic case will be outlined.

In a discrete action space, to output the possibility of each possible action a, given a state s, the policy may be defined using a softmax distribution. It is defined by

$$\pi(a|s,\theta) \doteq \frac{e^{h(s,a,\theta)}}{\sum_b e^{h(s,b,\theta)}}, \tag{3.38}$$

with $h(s,a,\theta) \in \mathbb{R}$ denoting the score or numerical preference for each state-action pair [70].

REINFORCE Algorithm

The Performance measure under a parameterized policy for episodic tasks is defined as the value of the start state under a policy π_θ. The function is formally defined as [70]

$$J(\theta) \doteq v_{\pi_\theta}(s_0). \tag{3.39}$$

In other words, the model seeks to find the parameters θ that maximize the expected return.

Algorithm 2 REINFORCE (episodic case). Taken from: [75],[70] and [65]

Input: Step size Parameter α
Randomly initialize weight vector $\theta \in \mathbb{R}^d$
for each trajectory (episode) $\tau_i \sim \pi(a|s,\theta)$ **do**
  for each time step $t = 0, 1, \ldots, T-1$ of the episode τ_i **do**
    - Estimate return for trajectory: $G_t \leftarrow \sum_{k=t+1}^{T} R_k$
    - Estimate the gradient of the performance measure:

$$\nabla_\theta J(\theta) = G_t \cdot \nabla \, log \, \pi(A_t|S_t,\theta)$$

    - Update policy weights: $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$
  end for
end for
return θ

Directly optimizing θ via the gradient $\nabla J(\theta)$ would require knowledge of selected actions and the distribution of the states in which the action was selected. One would have to consider every possible trajectory, which may either not be computationally feasible or, because policy gradient methods typically do not use a model of the environment, intractable since the effect of the policy on the state distribution is unknown [70].

A basic approach for the estimation of $\widehat{\nabla J(\theta)}$ presents the *REINFORCE* algorithm [75]. It is based on the idea of sampling a number of trajectories, given a policy $\pi(a|s,\theta)$ and then afterwards computing an estimation of the gradient using the sampled action-state pairs.

Given a sampled trajectory, the *policy gradient theorem* establishes that [70]

$$\begin{aligned}
\nabla J(\theta) &= \mathbb{E}_\pi \left[q_\pi(S_t, A_t) \, \frac{\nabla \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)} \right] && (\text{ Sampling } A_t, S_t \sim \pi \ ) \\
&= \mathbb{E}_\pi \left[G_t \, \frac{\nabla \pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)} \right] && (\ \mathbb{E}_\pi[G_t|S_t,A_t] = q_\pi(S_t,A_t)\ ) \\
&= \mathbb{E}_\pi \left[G_t \, \nabla \, log \, \pi(A_t|S_t,\theta) \right].
\end{aligned}$$

The parameters θ are then updated as follows [70]

$$\theta_{t+1} = \theta_t + \alpha G_t \, \nabla \, log \, \pi(A_t|S_t,\theta). \tag{3.40}$$

The whole REINFORCE operation is shown in detail in Algorithm 2. Now that the most important terms of reinforcement learning and policy gradients as well as the REINFORCE optimization algorithm are defined, these concepts can be transferred to the adversarial learning of GANs.

3.4 Related Work

So far only few approaches towards generating network traffic via generative adversarial networks were published. GANs, however, have already been utilized to further improve intrusion detection systems for internet of things devices [64] and industrial control systems [45] by providing artificial samples to the classifier and thus solving the problem of imbalanced and lacking data.

To evaluate network intrusion detection software, GANs have been applied to generate traffic flows [59]. Instead of crafting whole packets, the GAN was used to generate timestamps, numbers of transmitted bytes and addresses. To encode categorical data such as IP addresses word embeddings were used as representation.

PAC-GAN [16] also represented datagrams in byte form. Yet, instead of word embeddings, they used a more complex encoding scheme to feed those bytes into the GAN. They trained a conventional GAN consisting of two convolutional neural networks to output ICMP, DNS and HTTP requests. Afterwards they evaluated whether the requests would cause appropriate responses, when sent over the internet to various servers.

For the creation of malicious traffic the utilization of SeqGAN has been proposed [17]. To generate realistic botnet traffic, it has been coupled with an additional blackbox intrusion detection system. Instead of feeding pre-labeled data to the model, the IDS was used to label the data the discriminator is trained on. The authors were able to generate traffic, that evaded the detection of the IDS.

SeqGAN is a GAN architecture specialized on generating discrete sequences. This architecture is particularly well suited for generating text such as natural language sentences [77] or music compositions [22].

Its main characteristic is its approach towards combining the conventional GANs [27] with reinforcement learning. SeqGAN extends the basic training scheme of generative adversarial networks, by modelling the generation process as sequential decision-making as suggested by the authors of [6] and [7]. This intends to make GANs more suitable for the generation of discrete sequences.

A SeqGAN generator produces a sequence one element at a time, while the discriminator evaluates the authenticity of a sequence as a whole. This should allow the generator to generate longer and more complex sequences. Additionally, since it generates sequences of discrete tokens, SeqGAN can be applied to various tasks dealing with sequential data. Thus, it has the potential to be used to generate data stemming from a variety of domains. SeqGAN was rather successful, such that it influenced a large number of related publications

3.5 Conclusion

In this chapter a large number of formal methods for the implementation of a generative adversarial network were defined. Conventional GANs [27] consist of two neural networks, a generator and a discriminator network, which are trained mutually. They are used to obtain a generator model that is able to sample data that underlies a similar probability distribution as the training data. To do so, the generator learns a function that transforms an arbitrarily distributed latent vector into the probability distribution of the training data.

A neural network is basically a universal function approximator and can, for example, be used to solve regression or classification problems. The networks of a GAN can be simple densely-connected feedforward networks or more sophisticated architectures specifically well suited for the kind of data it is supposed to process.

For regression tasks that have to deal with sequential data this could be a recurrent neural network with LSTM cells, which alleviate the vanishing gradient effect. For classification tasks of grid-like data, such as images or time series, convolutional neural networks are well suited.

Neural networks require that categorical data such as words are converted into a numerical representation. A popular way to encode words is to encode them into a continuously valued vector representation, the word embedding.

The aforementioned types of ANNs as well as GANs are optimized through backpropagation. This, however, leads to the problem, that GANs in their most basic form are not well suited to generate sequential data that preserves the structure of the data they were trained on.

SeqGAN, however, introduces a promising approach to handle sequential data [77]. This approach leverages reinforcement learning techniques, allowing it to model the generation of sequences as a sequential decision-making process. This should enable the model to generate longer and more complex sequences. In the next chapter, the SeqGAN architecture will be formally defined and an implementation of SeqGAN will be presented. The model will then be trained on byte sequences that were collected from real network traffic dumps and a number of experiments will be conducted.

4 Application of ML Techniques

In the previous chapter the fundamentals of generative adversarial networks and reinforcement learning were established. In this chapter a formal definition for an architecture that is particularly well suited for the generation of sequential data will be given.

The SeqGAN approach represents an implementation of the suggested solution towards generating sequences. It models the sequence generation as sequential decision-making, as it has been proposed by the authors of [6], [7] and [25].

After giving a formal definition of adversarial learning via policy gradient, an overview of the implementation and the conducted experiments will be presented. The goal will be to train a SeqGAN architecture on captured network traces, containing datagrams of protocols which are used in SCADA systems. Traffic data of the IEC 60870-5-104 and the Modbus/TCP protocol is used as a case study.

Afterwards the SeqGAN is expected to be able to produce an arbitrary number of sequences which can be converted back into network packets. These packets are evaluated for their syntactical correctness, meaning that they obey domain constraints and the structure that is given by the protocol definition. In this thesis, relationships between datagrams are out of scope as well as the requirement that the generated datagrams are able to trigger state changes in a real system. This would be understood as semantically correct.

We will begin in Section 4.1 by defining the combination of reinforcement learning and adversarial learning. Afterwards in Section 4.2 the structure of the implemented SeqGAN model and in Section 4.3 the training algorithm will be presented. As a next step, Section 4.4 describes the capture files the network will be trained on and the processing of the input files and generator outputs. Finally, in Section 4.5 the conducted experiments for the evaluation of the network's performance will be presented. The results will be given in Chapter 5.

4.1 Adversarial Learning via Policy Gradient

In this section a formal definition for the training of GANs using the policy gradient algorithm of the last section will be given. A detailed explanation of the steps of the training algorithm will be given later. The described combination of adversarial and reinforcement learning originates from the SeqGAN archi-

tecture in [77]. In contrast to the conventional GAN presented in the previous chapter, SeqGAN does not consider the sampling process as a transformation of a latent vector. Instead it models the generation process as sequential decision-making.

4.1.1 Basic Structure

Just as in the original generative adversarial network [27], the SeqGAN architecture consists of two neural networks, a generator G_θ with a set of parameters θ and a discriminator D_ϕ, whose parameters are denoted as ϕ. The generator will be viewed as the agent in reinforcement learning. It will thus implement a parameterized policy $\pi(a|s, \theta)$, as characterized in Chapter 3 [77].

The generator model G_θ produces sequences of tokens $Y_{1:T}$. Token sequences are defined as $Y_{1:T} = (y_1, \ldots, y_t, \ldots, y_T)$, with $y_t \in \mathcal{Y}$ and $\mathcal{Y}$ being the vocabulary. Actions, formerly denoted as a, are denoted as y_t. An action is defined as the selection of the next token in a sequence. The state s at time step t is defined as the sequence of generated tokens so far $(y_1, \ldots, y_{t-1})$ [77].

The discriminator model D_ϕ is responsible for the evaluation of the generated tokens and creates the rewards for the optimization of G_θ. The output of the discriminator will be $D_\phi(Y_{1:T})$, which denotes the probability of generated sequence $Y_{1:T}$ being from real training data [77].

G_θ will be optimized via the REINFORCE algorithm, the training objective is defined as to generate a synthetic sequence of tokens which maximizes the expected return [77]

$$J(\theta) = \mathbb{E}[G_T|s_0, \theta] = \sum_{y_1 \in \mathcal{Y}} G_\theta(y_1|s_0) \cdot Q_{D_\phi}^{G_\theta}(s_0, y_1). \tag{4.1}$$

The function G_T denotes the return for a whole generated sequence and the function $Q_{D_\phi}^{G_\theta}(s, a)$ denotes the action-value function, which in this case stands for the expected reward for selecting token y_1, given state s_0 and afterwards following the policy.

The numerical reward is calculated by the discriminator and will be the possibility that a complete sequence originates from real data [77].

$$Q_{D_\phi}^{G_\theta}(s = Y_{1:T-1}, a = y_T) = D_\phi(Y_{1:T}). \tag{4.2}$$

The expected return is then dependent on the quality of the generated sequence. The objective is therefore to generate a sequence that is able to trick the discriminator into thinking that it is real, by selecting single tokens that in the end maximize the expected return.

4.1.2 Monte Carlo Tree Search

The discriminator can only evaluate a whole generated sequence. It is, however, desirable to be able to evaluate the selection of tokens at intermittent time-

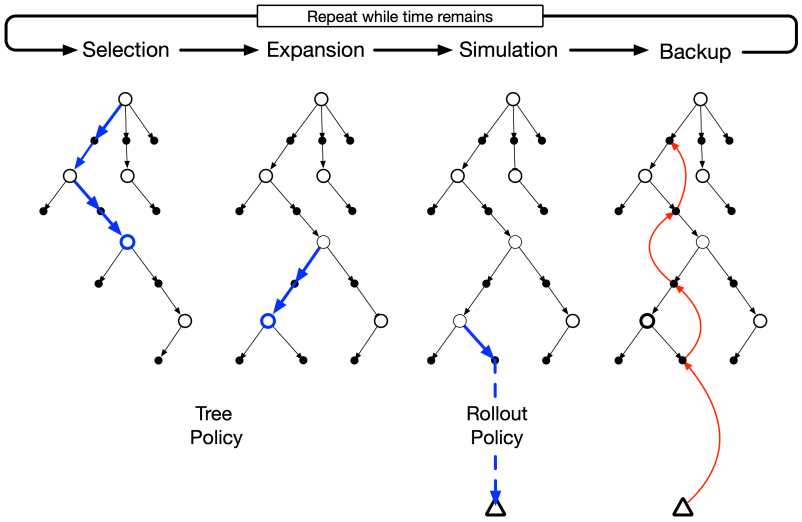


Figure 4.1: Monte Carlo Tree Search. Taken from: [70]

steps. This can be achieved using the *Monte Carlo tree search* (MCTS) [66].

MCTS is a tree-search algorithm, which can be used to find the most promising action in a sequential decision-making process. It does so by simulating random actions given a state to approximate the long-term reward of each possible action. The MCTS algorithm (Figure 4.1) is commonly defined by four main steps: selection, expansion, simulation, and backpropagation [70].

During selection, the algorithm begins at the root node, which represents the current state. The next action is selected based on pre-defined criteria such as the potential reward of an action. If a selected node has not been fully expanded, the algorithm adds one or more child nodes and expands the search to those nodes. This is referred to as expansion [70].

Randomly sampling a trajectory from the expanded node to the terminal state enables the algorithm to estimate the long-term potential of each possible action. This is done via a simulation function. This function is called *rollout policy*. After a whole trajectory has been sampled, the algorithm updates the action values of the nodes on the path from the root to the expanded node with the simulation results [70]. This sequence of steps is repeated until a timer runs out or a pre-defined number of simulations have been conducted [66].

In the case of SeqGAN, this means that the remaining part of a sequence will be sampled using the rollout policy. The rollout policy will be implemented by the same generator model G_β, yet with its own set of parameters denoted as β. If the remaining part of the sequence is complete, it will be evaluated by the discriminator. The accumulated action-values of each child will be propagated back to the root node.

4.1.3 Action-Value Function

The MCTS for an unfinished sequence is defined as [77]

$$\left\{Y_{1:T}^1, \ldots, Y_{1:T}^N\right\} = MC^{G_\beta}(Y_{1:t}; N), \tag{4.3}$$

with $Y_{1:t}^n = (y_1, \ldots, y_t)$ and the tokens selected so far denoted as $Y_{1:t}$. The remaining tokens $Y_{t+1:T}$ have to be sampled using the rollout policy G_β.

The rollout policy will be conducted N times to reduce variance. The complete action-value function is divided into two cases and defined by [77]

$$Q_{D_\phi}^{G_\theta}(s{=}Y_{1:T-1}, a{=}y_T) = \begin{cases} \frac{1}{N}\sum_{n=1}^{N} D_\phi(Y_{1:T}^n), Y_{1:T}^n \in MC^{G_\beta}(Y_{1:t}; N) & \text{for } t < T \\ D_\phi(Y_{1:t}) & \text{for } t = T \end{cases}. \tag{4.4}$$

In the first case, when there is no complete sequence yet, the already selected tokens will be evaluated together with the remaining tokens, which are sampled via MCTS. Afterwards, the average of N MCTS runs will be taken and this yields the action-value. In the second case, the discriminator will simply evaluate the generated sequence.

4.1.4 Generator and Discriminator Update

The weight optimization of the generator follows the concepts of the REINFORCE algorithm. The derivation of the performance measure (Equation 4.1) with respect to the parameters θ is defined as [77]

$$\nabla\, J(\theta) = \mathbb{E}_{Y_{1:t-1}\sim G_\theta}\left[\sum_{y_t \in \mathcal{Y}} \nabla_\theta\, G_\theta(y_t|Y_{1:t-1}) \cdot Q_{D_\phi}^{G_\theta}(Y_{1:t-1}, y_t)\right]. \tag{4.5}$$

The variable $Y_{1:t-1}$ denotes the intermediate state with $(y_1, \ldots, y_{t-1})$ already selected tokens. An unbiased estimation of the gradient of the performance measure $J(\theta)$ is given by [77]

$$\nabla_\theta J(\theta) \simeq \frac{1}{T}\sum_{t=1}^{T}\sum_{y_t \in \mathcal{Y}} \nabla_\theta\, G_\theta(y_t|Y_{1:t-1}) \cdot Q_{D_\phi}^{G_\theta}(Y_{1:t-1}, y_t) \tag{4.6}$$

$$= \frac{1}{T}\sum_{t=1}^{T}\sum_{y_t \in \mathcal{Y}} G_\theta(y_t|Y_{1:t-1})\, \nabla_\theta\, log\, G_\theta(y_t|Y_{1:t-1}) \cdot Q_{D_\phi}^{G_\theta}(Y_{1:t-1}, y_t) \tag{4.7}$$

$$= \frac{1}{T}\sum_{t=1}^{T} \mathbb{E}_{y_t\sim G_\theta(y_t|Y_{1:t-1})}\left[\nabla_\theta\, log\, G_\theta(y_t|Y_{1:t-1}) \cdot Q_{D_\phi}^{G_\theta}(Y_{1:t-1}, y_t)\right]. \tag{4.8}$$

The generator is then optimized as defined by the REINFORCE algorithm. Its parameters will be updated as follows [77]

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta). \tag{4.9}$$

The generator will then be used to optimize the discriminator with samples of both real and synthetic data. The discriminator is optimized by minimizing the function [77]

$$\min_{\phi} \; -\mathbb{E}_{Y\sim p_{data}}[log\, D_\phi(Y)] - \mathbb{E}_{Y\sim G_\theta}[log\,(1 - D_\phi(Y))]. \tag{4.10}$$

4.2 SeqGAN Implementation

The implementation of SeqGAN, which will be used for the experiments in this chapter is to serve as an assessment to answer the question, whether the combination of adversarial and reinforcement learning can offer useful results for the given task of generating network traffic. To do so, an implementation of SeqGAN which is as close as possible to the original implementation by [77] will be used. Possible modifications of the architecture are subject to future research.

I will train the model on datagrams that are represented by their hexadecimal byte notation. The generator will act as a reinforcement learning agent. It will try to reproduce the structure of datagrams by selecting single bytes, such that the expected return is maximized. The state at time step t, denoted as S_t, thus represents the generated bytes so far, and action A_t represents the next selected byte.

The SeqGAN is implemented in Python 3 using Tensorflow 2.x for two main reasons: firstly, due to me having significantly more experience in using Tensorflow 2, compared to other machine learning frameworks; secondly, due to the removed support of Tensorflow 1.x in Google Colab. The implementation is based on [61], which is an adaptation of the original implementation by [77] to Tensorflow 2.

I adapted the hyperparameters of the generator and discriminator models for the domain and length of the input data. They were chosen such that both models reach convergence and the generator is reliably able to produce correct packets. The network will be trained until the generator loss converges.

The corresponding code of the utilized implementation as well as a Jupyter Notebook containing my code for the conducted experiments are located at [62].

4.2.1 Generator Model

The generator model (Figure 4.2) uses a rather sparse design and is merely comprised of 3 layers. The first one is an embedding layer, which translates each discrete index of the tokenized input sequences into a vector of size 32. The next layer consists of a LSTM layer with 384, 512 or 1024 hidden units. The exact number depends on the conducted experiments and has been chosen to accommodate for the length of the processed sequences, the exact hyperparameters will be stated later. Finally, the output of the LSTM layer is fed

into a densely connected layer comprised of 256 units, which uses a softmax activation to output a probability for each of the entries in the vocabulary.

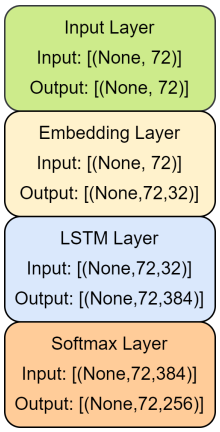


Figure 4.2: Generator LSTM Model

During sampling, the LSTM generates tokens on the basis of the tokens generated so far, while the first token will be generated on the basis of a start token. The start token is the same as the token used for padding shorter sequences and defined as 0. Figure 4.2 depicts the generator for an input sequence of length 72. The same architecture will be used to implement the rollout policy. It will, however, have its own set of weights.

4.2.2 Discriminator Model

The discriminator is a simple Convolutional Neural Network for 1-dimensional data. Its first layer is an embedding layer, which translates the tokenized input into 64-dimensional vectors. The higher number of dimensions for the embedding vector, compared to the generator, eases the classification. This, however, does not significantly increase the time required to achieve sufficient accuracy.

Afterwards the convolutional layers are located. While the original implementation used 12 different filter sizes, tests showed that 6 filters were sufficient to differentiate synthetic and real data with high accuracy. The convolutional layers are structured as 6 Tensorflow Conv1D layers in parallel, each of which uses filters of various lengths (see Figure 4.3 for reference).

The output of each of the Conv1D layers is fed into a Tensorflow MaxPool1D layer afterwards. The outputs of the pooling operations are then concatenated and flattened in the next layer. Afterwards, highway and dropout layers are located. The CNN uses dropout and L2 regularization as means against overfitting as suggested by [77]. Finally, the output is fed into a softmax layer determining the probability of a sequence being real. The specific hyperparameters of the convolutional layers are denoted in Table 4.1.

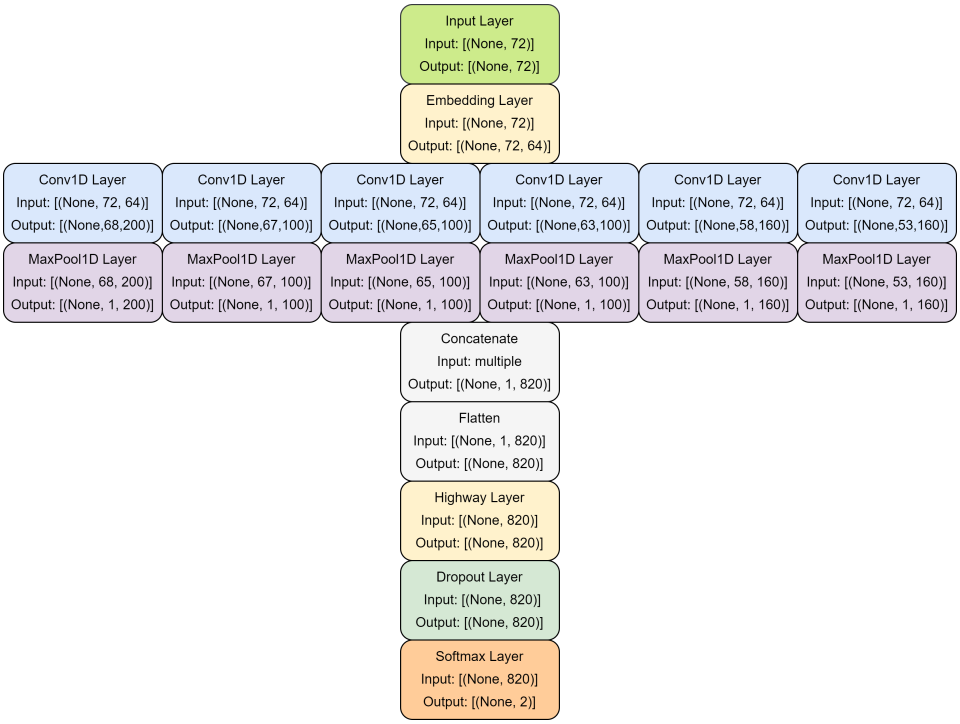


Figure 4.3: Discriminator CNN Model

Conv1D Layer	Filter Size	No. Filters
1	5	200
2	6	100
3	8	100
4	10	100
5	15	160
6	20	160

Table 4.1: Discriminator CNN Hyperparameters

4.3 Training Algorithm

The first step of the training algorithm consists of the initialization of the generator and the discriminator models as well as their weights. The training process consits of 3 main steps: a pre-training for both the generator and the discriminator and the adversarial training. The number of samples which are used for training will be specific for each experiment and is denoted in Table 4.2. Real Training samples are read line-by-line from a text file, called the *positive* file and synthetic data created during discriminator pre-training and adversarial learning will be written into a *negative* text file.

The generator will be trained until its loss converges. The exact number of the so called pre-training steps is also a hyperparameter and varies based on the number of training samples and their length. The authors of [77] refer to the pre-training step as MLE-training, therefore the same terminology is used here. However, during training the model is trained to minimize the negative-log-likelihood as defined by Equation 3.6.

During supervised pre-training, the generator model is fed with labelled data samples, stemming from training data. The model then generates batches of synthetic sequences, which are compared to batches of real sequences. Then the model is optimized via gradient descent such that the negative-log-likelihood decreases. After the pre-training, the weights of the generator will be stored, so that the conventionally trained generator can later be used as a baseline. The baseline model can then be compared to the corresponding generator, that was additionally trained using the adversarial training. When the pre-training of the generator is finished, the rollout policy model gets initialized and it's weights will inherit the values of the generator's weights.

The next step will be the pre-training of the discriminator (Figure 4.4 left). To be able to decide whether a sequence stems from real data or from the generator, it will be trained on both. This is achieved by feeding it a number of batches containing both real samples and synthetic ones from the pre-trained generator. The batch size will be fixed to 64 for all experiments and the sequences in each batch are ordered randomly. The discriminator will be trained until it can differentiate between real and synthetic sequences with high certainty so that it is able to offer a reliable reward signal for the adversarial training from the beginning. A single discriminator update step during pre-training and adversarial training consists of $k = 3$ epochs.

The third step of the training algorithm consists of the adversarial training (Figure 4.4 right), which uses the REINFORCE policy gradient update. At the beginning of every adversarial training step, the generator will sample one batch of synthetic sequences. Then the fitness of every sub-sequence beginning from token 1 in the batch will be evaluated. In this case, evaluation means that the discriminator classifies the sequences and assigns them a probability value. Since the discriminator operates on whole sequences instead of sub-sequences, the missing tokens in every sub-sequence will be replaced by randomly sampled

tokens as described in Subsection 4.1.2. This loop of evaluating every subsequence in a batch will be executed 32 times to reduce variance and to stabilize the training process.

Algorithm 3 SeqGAN Training Algorithm. Taken from: [77]

Input: Tokenized Training Sequences $\mathcal{S} = \{X_{1:T}\}$
Randomly initialize weights of G_θ, G_β and D_ϕ
Pre-train G_θ via MLE on $\mathcal{S}$
$\beta \leftarrow \theta$
Sample sequences $\mathcal{N} \sim G_\theta$
Pre-train D_ϕ via cross-entropy loss on $\mathcal{N}$ and $\mathcal{S}$
while SeqGAN not converged **do**
    for n generator steps **do**
        Sample sequences $Y_{1:T} = (y_1, \ldots, y_T) \sim G_\theta$
        for t in $1:T$ **do**
            Compute $Q_{D_\phi}^{G_\theta}(s = Y_{1:t-1}, a = y_t)$
        end for
        update θ via equation 4.9
    end for
    for m discriminator steps **do**
        Sample sequences $\mathcal{N} \sim G_\theta$
        Combine sequences $\mathcal{N}$ and $\mathcal{S}$
        Train D_ϕ for k epochs to minimize equation 4.10
    end for
    $\beta \leftarrow \theta$
end while

As soon as all rewards for every sequence in the batch are gathered, the generator model will be updated according to 4.9. The update rate α of the gradient update will vary depending on the specific experiment. Afterwards the rollout policy model again inherits the weights of the generator model.

Once a new generator is obtained, it will be used to further improve the discriminator model. This is done in the same way as in the pre-training process. The generator produces a previously fixed number of samples and the discriminator gets fed real and synthetic samples and has to distinguish them. During each adversarial training step, the generator will be updated once ($n = 1$), while the discriminator is updated twice ($m = 2$).

After every adversarial training step, the generator and discriminator weights, will be saved, so that each step can be evaluated later. This whole process will be repeated until the loss of the generator converges to zero. Figure 4.4 depicts the training process of SeqGAN. At the beginning the discriminator will be trained on real data and synthetic samples from the generator to be able to provide a useful reward signal for the training of the generator. After the pre-training, the generator will be adversarially trained through the REINFORCE policy gradient algorithm and MCTS.

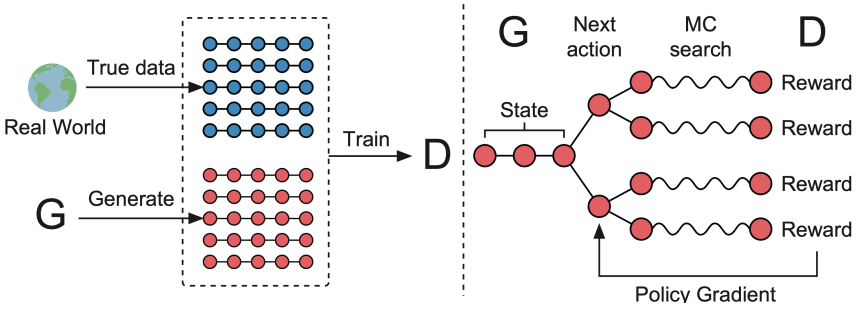


Figure 4.4: Two stage training algorithm of SeqGAN. Taken from: [77]

4.4 Input Data and Processing

4.4.1 Training Data Capture Files

As mentioned before, obtaining traffic data that was captured on live SCADA systems poses a problem. However, since we are only interested in producing packets, which obey a structure, that is defined by a network protocol, the demands towards the condition or the origin of the training data are not especially high. Captured packages that were recognized as correct and not malformed by the widely used software Wireshark, are considered sufficient as training data. The SeqGAN model will be trained on Modbus/TCP and IEC-104 traffic.

The Modbus/TCP dataset was acquired from Netresec using a testbed of the 4SICS conference [1]. It consists of commercial equipment, such as several PLCs, RTUs and switches for industrial applications. The capture file from 22 October 2015 will be used: it contains roughly 99400 Modbus/TCP packets. I chose this dataset due to its size. It is significantly larger, than any other dataset that is publicly available and could be obtained for testing.

Since deep neural networks are used, it is expected that larger training datasets lead to a more stable training process. The dataset is far from perfect, due to only a relatively small number of packets containing actual Modbus payloads, such as measurements. For the intended purpose of learning the structure of the datagrams, this will be considered acceptable, however.

The IEC-104 dataset stems from an attack dataset published in [74]. It was generated using the RICS-el testbed [4] running a commercial SCADA system, which emulates electrical power generation and a power grid. As with the Modbus/TCP dataset, I chose this dataset due to its comparatively large size and its availability. Obtaining datasets containing IEC-104 traffic proved to be even more difficult than Modbus/TCP.

The dataset contains captures of 12 attack scenarios and a baseline with benign traffic. The network will be trained on the baseline capture file, which contains about 2000 rather small packets. However, due to the small number of packets, I considered merging those attack capture files that contain traffic with

only slightly modified values to obtain a larger dataset. The modifications that were done to the traffic [74] cover man-in-the-middle attacks, which modified timestamps and measurements. I think that training the network on this modified traffic is adequate, due to the defined goal of being able to capture the structure of packets that stem from real SCADA hardware. But it is important to note that relationships between packets of this merged dataset will be incorrect in many cases since the packets are not all related to one another. Therefore, it will be marked as a modified dataset in the experiments.

4.4.2 Data Processing

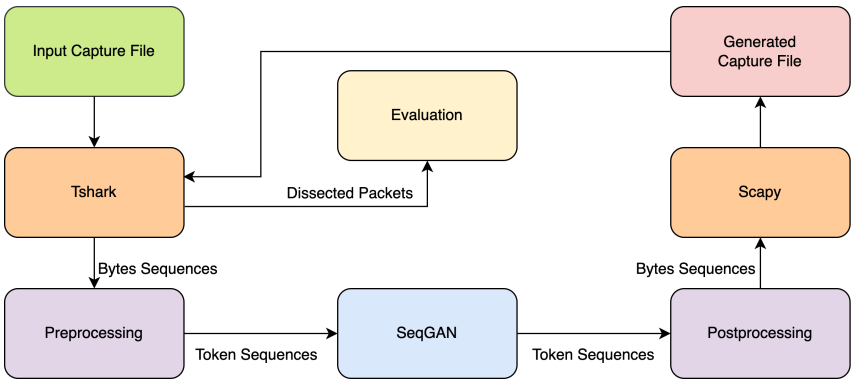


Figure 4.5: Processing Steps: Training Capture File to Evaluation

The main purpose of generating SCADA traffic is to obtain traffic that can be used for further research. The goal of this thesis is, to examine whether a generative adversarial network is able to lean and reproduce the structure and domain constraints of datagrams.

I made the decision to train the network on the whole Ethernet frame, instead of, for example, only processing the TCP payload, since I wanted to answer the question whether the GAN would be able to generate the whole frame. This makes the pre- and post-processing less complex. Training data only has to be converted into byte sequences. However, the amount of data that has to be correctly reproduced increases. The generated packets need to resemble a whole Ethernet frame, containing MAC and IP addresses, TCP port numbers and a Modbus/TCP or IEC-104 payload, respectively.

Additionally, I chose to use the hexadecimal data representation, because I already had to process categorical data such as IP and MAC addresses to generate whole datagrams. I also thought it would be desirable to avoid the need for domain knowledge and manual intervention during pre- and post-processing. Therefore, I dismissed the idea to generate both continuously valued and categorical data and craft packets using this information. It would

have required to instruct, for example, Scapy where each value belongs, thus the process of learning a structure would rather be performed by the researcher.

Using the byte-level representation has another advantage, namely that a GAN that is able to generate byte sequences of Modbus/TCP or IEC-104 datagrams could most likely be universally utilized to generate traffic of various network protocols without much adaptation.

The input data consists of network captures in *.pcap file format. These will be processed using the Tshark terminal application, which filters the capture file for not malformed Modbus/TCP and IEC-104 packets. This filtering step is especially noteworthy as it ensures that certain domain constraints, for example of IPv4 addresses, are obeyed. The filtered packets are then converted into a JSON string, from which a Python list of byte sequences for each packet can be extracted. The byte sequences are represented as 2-digit hexadecimal values and use spaces as separator. I chose the 2-digit representation since it limits the size of the vocabulary to 256 entries and presents a good compromise between sequence length and vocabulary size. A smaller vocabulary allows to use a smaller number of dimensions in the embedding layer and thus less complexity. These converted sequences are denoted in Listing 4.1.

```
00 90 e8 26 40 23 00 07 7c 1a 61 83 08 00 45 00
00 34 4e 34 40 00 7e 06 d2 5c c0 a8 02 a6 c0 a8
58 3c 07 bd 01 f6 bb e0 51 f1 08 74 6d 30 50 18
3f 5e 05 54 00 00 00 aa 00 00 00 06 01 01 00 00
00 01
```

Listing 4.1: 66 Bytes Modbus/TCP Frame

The byte sequences are then fed into a function which tokenizes the byte sequences. Afterwards, each single byte will be represented by it's index in the vocabulary. The length of the longest sequence contained in the training data, that is to be processed has to be fixed. However, if there are shorter sequences, they will be padded by the index 0. An example for a pre-processed training sequence is denoted in Listing 4.2. The preprocessed sequences are written line-by-line into the positive text file.

```
1 145 233 39 65 36 1 8 125 27 98 132 9 1 70 1
1 53 37 13 65 1 127 7 253 133 193 169 3 167 193 169
89 61 8 189 2 247 15 216 28 186 43 145 176 124 81 25
65 42 133 13 1 1 1 3 1 1 1 7 2 2 1 1
1 2
```

Listing 4.2: Preprocessed Modbus/TCP Frame

The SeqGAN generator outputs token sequences, which then have to be converted back into bytes. For this, they are fed into a post-processing function that looks up the corresponding byte value to an index in the vocabulary. The

byte sequence can then be converted into a datagram which is then written into a capture file. This is achieved through the Python library Scapy. The library offers a simple API to convert a whole packet frame in hexadecimal format into a network packet. Scapy then writes the crafted packets into a *.pcap capture file. Figure 4.5 illustrates the complete data processing.

4.5 Experiments

The application of the SeqGAN training algorithm on the process of generating network packets will be evaluated in two ways: First, I will test, whether the generated byte sequences can be recognized by the Wireshark protocol analyzer, and, if so, how much of the generated output will be considered syntactically correct by the program. This definition of syntactical correctness on the one hand means that a datagram is compliant to the structure that is specified by the protocol. On the other hand, it also implies that domain constraints, such as certain address ranges, are met.

As a second step, I will assess the address information of the generated packets. Depending on the use-case of the generated traffic, it is possible that it should fit into an existing network. Since these often use firewalls with whitelistings to be able to appear as real traffic, the generated packets must contain at least some address information of the original training data.

The widely used BLEU-score, which is defined as a measure for the quality of machine-translated text, will not be used as an evaluation metric. This differs from many other publications, which tackle problems in the context of natural language processing. However, due to the findings of [63], which suggest that BLEU-scores may be misleading for the generated samples of generative adversarial networks, especially when it comes to mode collapse, they will not be utilized. I think that it is adequate to use the average number of syntactically correct packets recognized by Wireshark as main performance measure, to answer the main question of this thesis.

This was achieved by generating 10000 samples, which corresponds to around 156 batches, to obtain an average value. These generated tokenized sequences are converted back into their hexadecimal form and written into *.pcap capture files using the Scapy Python library. The output capture files are then read via the same Tshark command that was originally used to obtain the training data. Thus the file will be filtered for Modbus/TCP and IEC-104 packets which are recognized as not malformed.

For the experiments, I normalized the training data based on their number of bytes. For the tests, packets with lengths that occurred significantly less than a thousand times were discarded from training data. This has the consequence that no tests with IEC-104 packets longer than 72 bytes were conducted. All 72 byte long samples represent spontaneous events. Additionally, there were only very small numbers of U- and S-format frames, compared to those in

I-format, in the capture files. They were therefore also excluded. For the purpose of the experiments, I considered this acceptable. Since the I-format frames possess a more complex structure and consist of a larger number of bytes, it is assumed that this makes the reproduction of a semantically correct frame more challenging. Table 4.2 denotes the conducted experiments.

Test	Type of Training Data	Length	Train Samples
1	Modbus/TCP small packets	66	512
2	Modbus/TCP small packets	66	1600
3	Modbus/TCP small packets	66	6000
4	Modbus/TCP small packets	66	10000
5a	Modbus/TCP small packets	66	15000
5b	Modbus/TCP small packets	66	15000
6	Modbus/TCP large packets	314	6000
7	Modbus/TCP small (var.) packets	63-66	6000
8	Modbus/TCP large (var.) packets	311-314	6000
9a	Modbus/TCP large (var.) packets	311-314	15000
9b	Modbus/TCP large (var.) packets	311-314	15000
10	IEC-104 small packets	72	1600
11	IEC-104 small (Modified Dataset)	72	6000
12	Modbus/TCP small packets	66	15000
13	Modbus/TCP large (var.) packets	311-314	15000
14	Modbus/TCP small packets	66	15000

Table 4.2: Processed Training Data per respective Test

Tests with smaller and larger Modbus/TCP packets were conducted, as well as tests with only a very small number of samples. This is to gain an impression whether SeqGAN would be suitable in cases where no large capture files can be obtained. For both types of network protocol, Modbus/TCP and IEC-104, tests with the maximum number of available training samples were conducted. For those tests, which cover the processing of variable length sequences, the following applies. Test 7, which covers the processing of Modbus/TCP packets of size 63 to 66 bytes, contains an almost equal number of packets of size 63 and 66 bytes.

Tests 9a and 9b contain almost equal numbers of size 311 to 313 and double as many packets of size 314. It is important to note that only test 7 contains a balanced number of queries as well as responses. The responses in these tests all contain exception codes. There were no datagrams containing actual Modbus payloads in the training data, due to their very small quantity of around 200 samples and the aforementioned requirements for the experiments.

Tests 12 and 13 represent a modification of tests 5 and 9 respectively. They were conducted to examine whether a more complex generator model would lead to significantly different results. Additionally, for these two tests the

training time was largely increased, firstly, to cope with the more complex model and secondly, to check whether more training steps would affect the results.

Ultimately, test 14 is also a modification of tests 5 and 9. It was conducted to examine the effect of interleaved MLE-training steps in between adversarial training steps. The authors of [31] proposed to utilize interleaved training instead of solely using adversarial learning after the pre-training to prevent the SeqGAN from mode collapse. Test 14 applied 20 MLE-training steps after every 4 adversarial training steps. I chose this number of steps since test 5 reached optimum performance after 4 steps.

Test	LSTM States	α	G Pre-Steps	D Pre-Steps	SeqGAN Steps
1	384	0.01	100	50	(0)
2	384	0.1	100	20	1
3	384	0.1	100	20	3
4	384	0.1	35	20	1
5a	384	0.1	50	20	4
5b	384	0.1	50	20	4
6	512	0.1	120	10	2
7	384	0.1	100	20	2
8	512	0.1	120	10	2
9a	512	0.1	120	10	(1)
9b	512	0.1	120	10	3
10	384	0.1	100	20	(1)
11	384	0.1	100	20	1
12	1024	0.1	150	20	1
13	1024	0.1	150	20	(25)
14	384	0.1	50	20	116

Table 4.3: Hyperparameters per respective Test

Apart from the aforementioned hyperparameters which presented a good balance for all conducted tests. I adapted a few hyperparameters (Table 4.3) for the experiments.

For each experiment the weights of the generator were saved after pre-training and after each adversarial training step. Afterwards the best performing generator was chosen. The pre-trained generator model will act as a baseline.

The SeqGAN steps parameter in Table 4.3 refers to the number of adversarial training steps. It denotes the number of steps until the best performing generator was obtained, except in those cases when the generator was not able to perform any better than the baseline. Then the number of conducted training steps for the best adversarially trained generator are denoted in brackets. All tests were conducted multiple times to account for fluctuations in the

outcomes of the training process.

4.6 Conclusion

In this chapter, the application of the machine learning techniques, that were defined in Chapter 3 was presented. Afterwards, the training data that will be used for the conducted experiments was described as well as the pre-processing that is applied to it. Capture files, that are part of publicly available datasets were converted into sequences of 2-digit hexadecimal bytes using the Tshark command-line application.

To answer the main question of this thesis, the SeqGAN architecture will be utilized. SeqGAN [77] introduced a promising approach to handle sequential data. The generator is still trained unsupervised on the basis of an evaluation of its output, obtained by a discriminator network. This is, however, no longer done through basic backpropagation. Instead the generator is optimized via a technique that originates from reinforcement learning. SeqGAN utilizes the discriminator output as a reward signal for the generator to optimize it via a policy gradient algorithm. It is able to not only evaluate a whole sequence, but also to evaluate a sequence at intermittent time steps. Doing so allows the process of generating a token sequence. to be considered as a sequential decision-making process. This is to enable the network to generate longer and more complex sequences than basic MLE-trained LSTM models.

The SeqGAN architecture uses a supervised pre-training step, that applies conventional MLE-training to the generator model. This offers the possibility to directly compare the same model's performance after applying the SeqGAN training algorithm. The experiments cover an evaluation of the baseline generator model, as well as the adversarially trained one. Their performance will be measured by examining how many syntactically correct packets are obtained, when generating a fixed number of samples. Moreover, it will be examined how well the generators are able to reproduce address information, that is contained in the training data. The results of these experiments will be presented and discussed in the next chapter.

5 Discussion and Results

In this chapter the results obtained from the conducted experiments will be presented in Section 5.1 and interpreted in Section 5.2. Afterwards in Section 5.3 the findings will be discussed and I will answer the question whether the utilized SeqGAN architecture is a suitable approach for generating datagrams that obey the Modbus/TCP and IEC 60870-5-104 protocol specifications.

5.1 Results

The presentation of results will be conducted twofold. Firstly, in Subsection 5.1.1 the ability of the baseline and adversarially trained generators will be evaluated by the number of syntactically correct datagrams they are able to produce. This will be the quantitative analysis. Secondly, in Subsection 5.1.2 the content of the generated packets will be examined. It will be investigated whether the address spaces of the training data is reproduced in the generated samples.

All tests were conducted using Google Colab Pro. The supplied hardware includes two 2GHz Intel Xeon Skylake cores, 13 gigabytes of RAM and a Nvidia Tesla T4 GPU with 16 gigabytes GDDR6 RAM. Tests ran until the generator loss converged.

5.1.1 Quantitative Analysis

The time parameter in Table 5.1 denotes the training time until the best performing generator was obtained. Times in brackets, denote the training time of the best performing adversarially trained generator which did not perform better than the baseline. The baseline correct frames parameter denotes the number of syntactically correct frames which were generated by the MLE trained generator. The exact number of correct outputs turned out to be fluctuating over multiple runs, but they were consistent in their magnitude. Numbers of frames in brackets describe the best performing generator that was not better than the baseline.

The tests which fluctuated the most in their results are denoted multiple times. This applies for test 5, since in test 5a the generator performed as expected, but in test 5b it performed unexpectedly well. Test 9 fluctuated between an improvement and no improvement compared to the baseline. However, there was only an improvement when the baseline generator performed

Test	Runtime	Baseline Correct Frames	SeqGAN Correct Frames
1	(3min)	38/0.4%	(0/0.0%)
2	4min	21/0.2%	67/0.7%
3	14min	36/0.4%	55/0.6%
4	12min	74/0.7%	233/2.3%
5a	24min	65/0.7%	162/1.6%
5b	24min	76/0.8%	1439/14.4%
6	1h15min	38/0.4%	108/1.1%
7	11min	2971/29.7%	3919/39.2%
8	49min	32/0.3%	180/1.8%
9a	(1h15min)	354/3.5%	(90/0.9%)
9b	2h24min	20/0.2%	163/1.6%
10	(4min)	325/3.3%	(151/1.5%)
11	10min	360/3.6%	626/6.3%
12	36min	118/1.2%	196/2%
13	(18h57min)	122/1.2%	58/0.6%
14	5h44min	70/0.7%	788/7.88%

Table 5.1: Results: Number of syntactically correct frames

especially badly. Test 7, despite a surprisingly good outcome, performed especially well in every run. However, it is important to note, that this generator produces a large number of "suspected" retransmissions. The desktop version of Wireshark, in contrast to the terminal application, filters those retransmissions and labels them as as simple TCP traffic. That is why it only recognizes a much smaller number of packets as correct by the given definition compared to Tshark.

5.1.2 Assessment of Address Information

The syntactically correct packets all contain address information, which complies to the respective protocol. Creating network traffic that is as close to real traffic to be able to interact with real SCADA networks is beyond the scope of this thesis. However, to not only investigate the number of synthetic packets, the packet contents will be assessed to some extent as well.

Reproduction of the addresses of the training data may or may not be desirable, depending on the use case. However, SCADA systems commonly use whitelisting rules to prevent the injection of packets from unauthorized sources. Therefore, if one wishes to make use of the generated datagrams in a real network, they would have to contain specific address information.

As a next step, it will be examined whether the generator is able to reproduce the address spaces of the input data. This covers the MAC and IPv4 addresses

as well as the combination of IPv4 address and TCP source or destination port, respectively. The last requirement is especially important, since the Modbus/TCP port 502 belongs to the well-known ports.

Additionally, it will be examined whether the TCP sequence and acknowledgement numbers fit. This would point to a correct TCP session according to the protocol. Furthermore, packets with fitting SEQ and ACK numbers are highlighted as correct transmissions in Wireshark. Finally, the addresses which are associated with the respective application layer protocol are examined. For Modbus/TCP this is the Unit ID and for IEC-104 the ASDU Common Address and the Information Object Address.

To gain a more detailed insight into the quality of the generated network traffic, the previously generated capture files of those tests in which a significant improvement through adversarial training was observed will be examined. The capture files which contained 10000 samples are again filtered for the respective protocol, Modbus/TCP or IEC-104, and for not malformed packets. It has to be noted that due to the applied filtering only packets which obey certain domain constraints are taken into consideration.

Test	Src. MAC	Dest. MAC	Src. IP:Port	Dest. IP:Port	Unit ID	SEQ/ACK
2	21/100%	21/100%	21/100%	19/90.5%	21/100%	8/38.1%
4	74/100%	74/100%	72/97.3%	69/93.2%	73/98.7%	15/20.3%
5b	76/100%	76/100%	64/84.2%	74/97.4%	76/100%	25/32.9%
6	38/100%	37/97.4%	32/84.2%	34/89.5%	38/100%	13/34.2%
7	1169/39.4%	1168/39.3%	1165/39.2%	1167/39.3%	2970/99.9%	26/0.9%
8	32/100%	32/100%	30/93.8%	28/87.5%	30/93.8%	6/18.8%
12	118/100%	118/100%	115/97.5%	118/100%	118/100%	17/14.4%
14	70/100%	70/100%	70/100%	69/98.6%	70/100%	10/14.29%

Table 5.2: Address Reproduction: Baseline Generators (Modbus/TCP)

Test	S.MAC	D.MAC	S.IP:Port	D.IP:Port	ASDU Addr	IOA	SEQ/ACK
11	360/100%	359/99.7%	357/99.2%	356/98.9%	351/97.5%	350/97.2%	23/6.4%

Table 5.3: Address Reproduction: Baseline Generators (IEC-104)

Table 5.2 contains the results of the analysis of the capture files stemming from the MLE-trained baseline generators. Absolute numbers denote the number of packets that reproduced corresponding addresses in the training data. The percentages denote the fraction of all 10000 samples contained in the capture file. Table 5.4 denotes the outputs of the adversarially trained generators.

The conducted evaluation did consider the direction of the transmission where it was meaningful. Thus, source and destination addresses were considered separately. For example, a reproduced source MAC address was counted only if it was used as a source MAC address in the training data. This was

not done for the Unit ID since it is copied from the request upon response [50] and thus used in both directions.

Test	Src. MAC	Dest. MAC	Src. IP:Port	Dest. IP:Port	Unit ID	SEQ/ACK
2	67/100%	67/100%	57/85.1%	66/98.5%	67/100%	14/20.9%
4	233/100%	232/99.6%	130/55.8%	223/95.7%	233/100%	48/20.6%
5b	1439/100%	1408/97.9%	1414/98.3%	1436/99.8%	1439/100%	27/1.9%
6	77/71.3%	91/84.3%	64/59.3%	91/84.3%	107/99.1%	42/38.9%
7	94/2.4%	94/2.4%	94/2.4%	94/2.4%	3918/99.9%	10/0.3%
8	56/31.1%	0/0%	124/68.9%	164/91.1%	171/95%	26/14.4%
12	196/100%	196/100%	196/100%	195/99.5%	196/100%	25/12.8%
14	777/98.6%	788/100%	788/100%	788/100%	788/100%	363/46.07%

Table 5.4: Address Reproduction: SeqGAN Generators (Modbus/TCP)

Test	S.MAC	D.MAC	S.IP:Port	D.IP:Port	ASDU Addr	IOA	SEQ/ACK
11	626/100%	626/100%	624/99.6%	623/99.5%	599/95.7%	598/95.5%	54/8.6%

Table 5.5: Address Reproduction: SeqGAN Generators (IEC-104)

Table 5.3 similarly contains the results of the analysis of the baseline generator that produced IEC-104 traffic, while Table 5.5 denotes the corresponding adversarially trained generator. For this experiment, the ASDU addresses and the information object addresses were examined. Since the network was trained on spontaneous events only, the direction of ASDU addresses and information object addresses was disregarded. The SEQ/ACK column in each table denotes the number and proportion of packets which have their sequence and acknowledgement numbers set correctly.

5.2 Interpretation

As can be seen, in most cases the adversarial training led to a better yield. The only two exceptions were those tests in which only a smaller number of training samples was supplied. As expected in the context of deep learning, a larger number of training samples led to a more stable training process in all of the tests.

Tests with up to 66 bytes proved to have a quite stable training process with similar outcomes when conducted multiple times. In contrast to that, the tests with up to 314 bytes proved to be much more unstable. The number of correct packets fluctuated a lot more compared to the other tests.

When examining the results of tests 8 and 9, the tests which were trained on input data of various length, it can be observed that nearly all of the generated samples had the maximum length of 314 bytes. This was not the case for shorter sequences in test 7. Being able to generate a good variety of

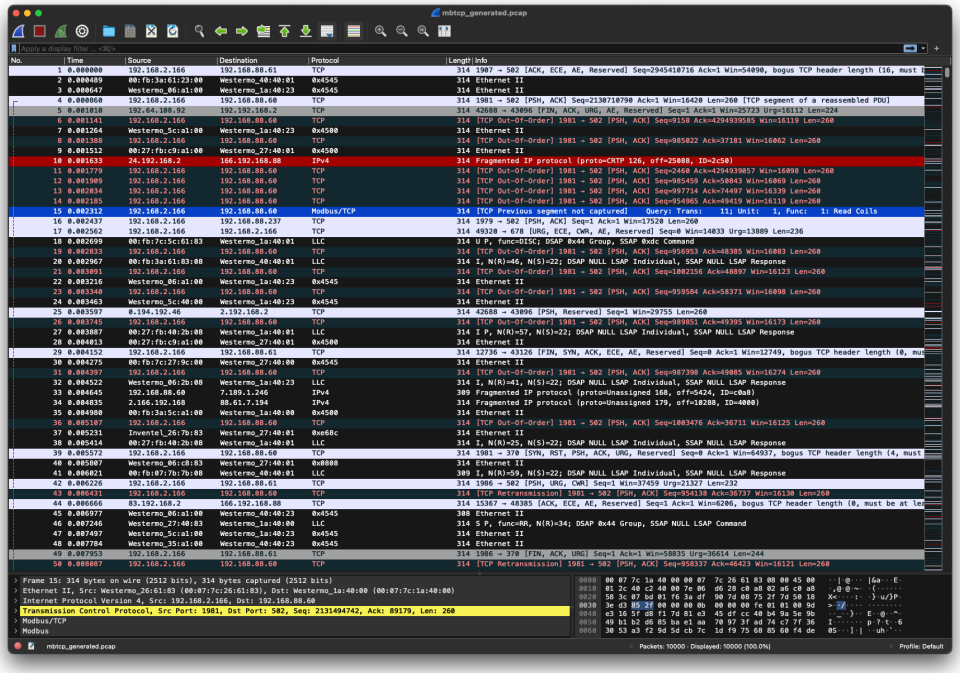


Figure 5.1: Wireshark Output - Test 8

differently sized packets may be desirable, since training a generator for every type of packet can be unfeasible depending on the use-case.

The described outcome of tests 8 and 9 is especially problematic since the training data was already normalized and filtered for only 4 different sizes. Normalizing the input such that it only contains samples with a certain range sizes can, however, be considered acceptable, at least if certain constraints of the respective protocol are taken into account. The corresponding response to a request may, for example, have a different length. This was observed on the Modbus/TCP training dataset.

Figure 5.1 shows the analysis of an unfiltered Modbus/TCP capture file. It consists of 10000 samples, generated by the generator of test 8. The depicted analyzed capture files shown are newly generated and not identical to the ones evaluated in Section 5.1, thus the deviating number of correct frames.

The capture files used for the screenshots do, however, represent typical results of most of the tests. Figure 5.2 shows the analysis of the same file after filtering it for Modbus/TCP packets. One can see that in most cases the transmissions are labeled as missing previous frames. That is because the SEQ and ACK numbers are not set correctly. There are a number of TCP transmissions, these are the ones that are presumably retransmissions. Tshark does recognize those as correct Modbus/TCP packets as mentioned in the previous section.

Figure 5.4 shows the analysis of an unfiltered IEC-104 capture file and Figure

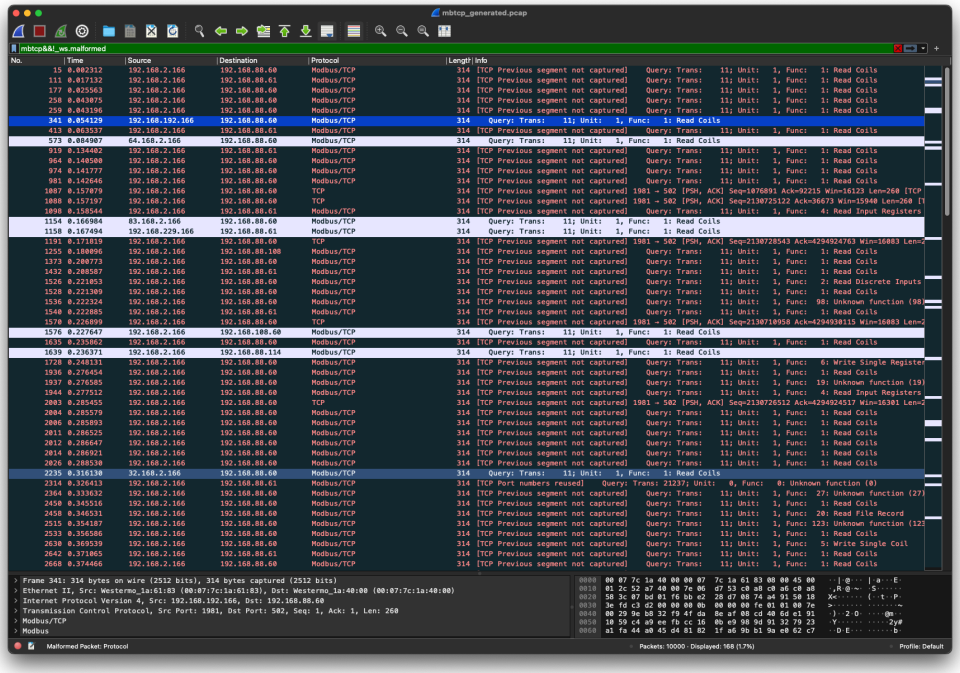


Figure 5.2: Wireshark Output - filtered - Test 8

5.5 shows the corresponding file after filtering it. One can see, that again SEQ and ACK numbers rarely fit. The transmissions are all labelled as spontaneous events. This is as expected since the generator was only trained on such transmissions.

Tests 12 and 13 were conducted to test whether an even more complex generator model would lead to significantly different results. The generator models of these tests used an LSTM layer with 1024 hidden units, thus the number of units was doubled. The number of pre-train and adversarial training steps was also increased accordingly, to cope with the increased model complexity.

Tests 12 and 13 used the same training samples as test 5 and test 9 respectively. The baseline model of test 12 showed improved performance. Yet it is hard to tell whether the adversarial training showed an equal improvement. This is due to the large fluctuations in test 5. However, these fluctuations decreased in test 12.

When examining the address information contained in the capture file of test 12, there was no clear increase in reproduction of addresses. While the combination of source IP address and TCP port occured slightly more often, there was a relatively large decrease of correct sequence and acknowledgement numbers. The decrease in fluctuation of test 12 was also visible in test 13. Nevertheless it showed no real improvement in contrast to test 9. The adversarial training in this experiment led to worse results.

Test 12 was additionally conducted to ensure that there is no sudden per-

formance improvement during which the generator would overcome the mode collapse situation after a certain number of training steps. Therefore, it was trained for much longer.

Training of test 12 ran for 8h 18min and 160 steps. Analyzing the generator performance after every single training step showed the same results as in most of the other tests. The first training step leads to improved performance over the MLE-training. Afterwards the performance gets worse and concludes in mode collapse.

Listing 5.1 and Listing 5.2 show an example datagram that was created by the generator after 50 and 120 train steps, respectively. Listing 5.3 ultimately shows the exact same mode collapse situation after 160 train steps. Figure 5.3 shows the analysis of the capture file created after training the generator of test 12 for 160 steps. One can see that it only contains byte sequences which do not correspond to any network protocol. Wireshark labels them as Ethernet II, since it interprets the first few bytes as source MAC addresses.

```
00 90 e8 26 ef 91 e7 91 91 e7 e7 e9 91 91 91 91
91 91 91 91 91 91 91 91 91 91 91 91 91 91 91 91
91 91 91 91 91 91 91 91 91 91 91 91 91 91 91 91
91 91 91 91 91 91 91 91 91 91 91 91 91 91 91 91
91 91
```

Listing 5.1: Test 12 - 50 steps

```
00 90 e8 26 40 91 91 91 e7 91 e7 e9 91 e7 91 91
91 91 91 91 91 91 91 91 91 91 91 91 91 91 91 91
91 91 91 e7 91 5f 91 91 91 91 91 91 91 91 91 91
91 91 91 91 91 91 91 91 91 91 91 91 91 91 91 91
91 91
```

Listing 5.2: Test 12 - 120 steps

```
00 90 e8 26 40 91 79 79 e7 91 e9 91 91 e7 91 91
91 91 91 91 91 91 91 91 91 91 91 91 91 91 91 91
91 91 91 91 e9 91 91 91 91 91 91 91 91 91 5c 91
91 91 91 91 91 91 91 91 91 91 91 91 91 91 91 91
91 91
```

Listing 5.3: Test 12 - 160 steps

Test 14, the experiment that applied interleaved training [31], showed a much more stable training process. It did not quite reach the performance of test 5b, however, it provided more consistent results. The results of test 5b could not easily be reproduced by running the training process multiple times, this was, however, not the case for test 14. Running it multiple times led to comparable outcomes.

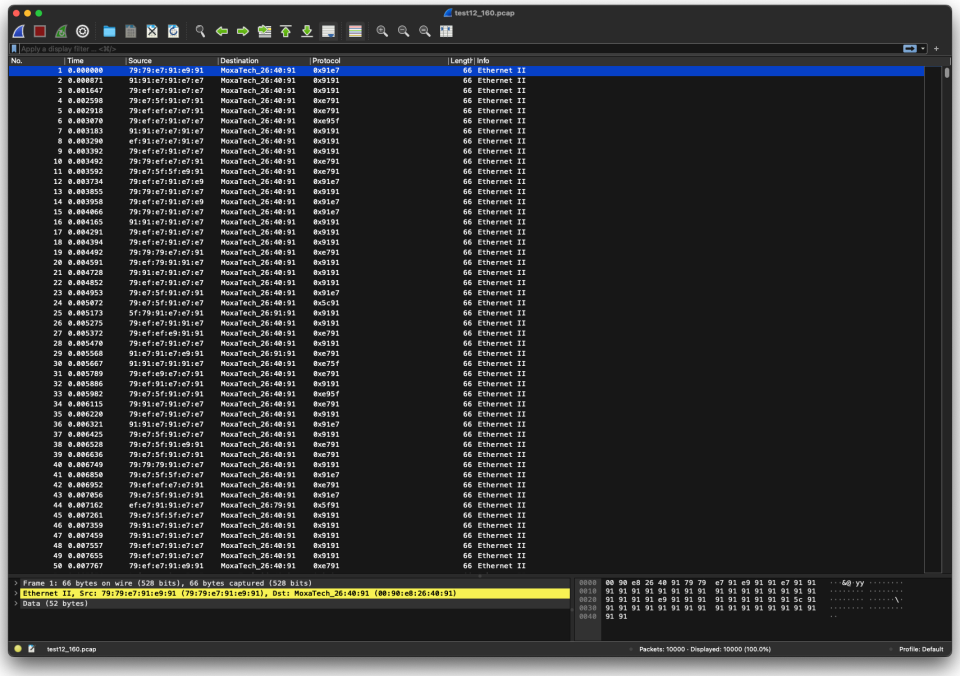


Figure 5.3: Wireshark Output - Mode Collapse - Test 12

Additionally, it is noteworthy that test 14 showed a large increase in correctly set sequence and acknowledgement numbers. This indicates, that interleaved training improves the reproduction of syntactical dependencies between packets. But most importantly, this test showed that the mode collapse situation can be overcome by enhancing the training algorithm with intermediate supervised learning steps. The training algorithm ran for 178 steps, until the runtime encountered an error, unfortunately. However, mode collapse has been overcome each time, it occured only 11 times and mostly during the first 12 adversarial training steps.

The well performing generators were in many cases able to correctly reproduce the address spaces of the input. This is true for the baseline as well as for the adversarially trained generators. Two exceptions for this were tests 7 and 8, for which the adversarial training overall worsened the result.

Payload data could only be examined for IEC-104 traffic, due to the training data used. As mentioned before, this was not considered to be a problem, since semantical correctness lies outside of the scope of this thesis. Yet, a short examination of the generated IEC-104 traffic showed that the samples of test 11 at least offered diversity as far as the contained measurement values are concerned. This suggests, that the generator was able to make out a part in the sequences that does not have to lie in a very narrow value range, in contrast to, for example, address information.

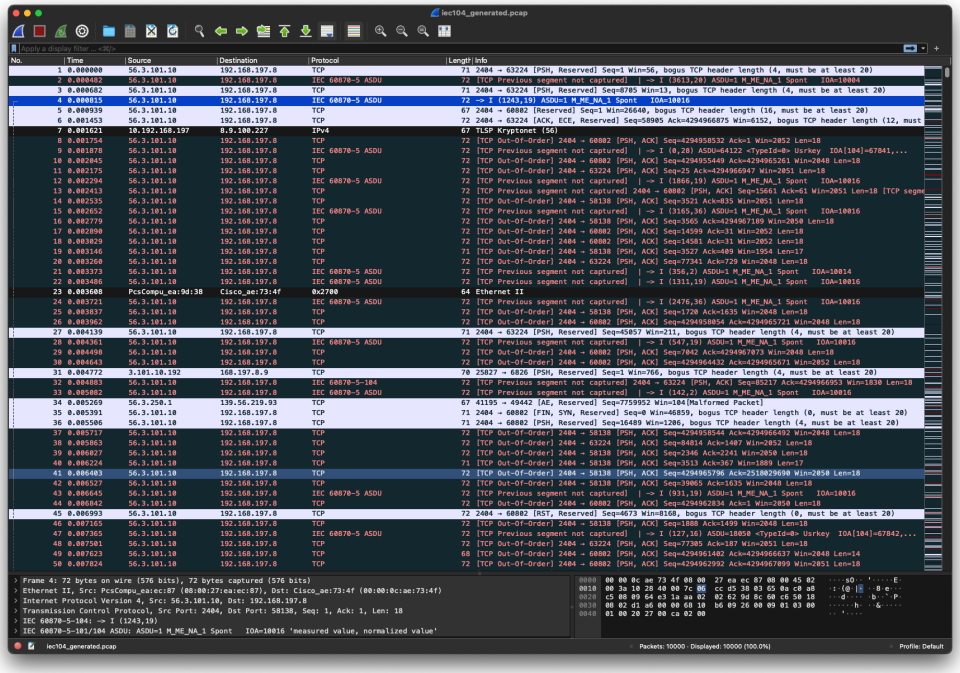


Figure 5.4: Wireshark Output - Test 11

Modbus/TCP samples mostly reproduced the Transaction IDs of the training data. For test 7, which contained queries as well as responses, this had the consequence that there were correct query-response transmissions. This is, however, not surprising since the input data contained a very small number of different Transaction IDs.

It is noteworthy that all generators were able to generate a good variety of function codes instead of always reproducing a particular value. When generating a response, the packets contained an exception code as defined by the protocol.

Independently of the conducted test, all generators were able to set sequence and acknowledgement numbers correctly according to the TCP protocol on a fraction of packets. This is noteworthy since these datagrams are marked as correct transmissions in Wireshark as defined by their protocol. Other packets were marked as possible retransmissions.

Overall the results suggest that the applied SeqGAN training algorithm led to an optimization of the conventionally trained generator. Nevertheless, one has to consider the application area of this thesis. The main objective was the generation of SCADA system network traffic. Particular network traffic of which only small numbers of samples are publicly available. Yet especially those tests that processed only a small number of training samples are the ones that failed.

In relation to the number of generated samples for each tests, the portion

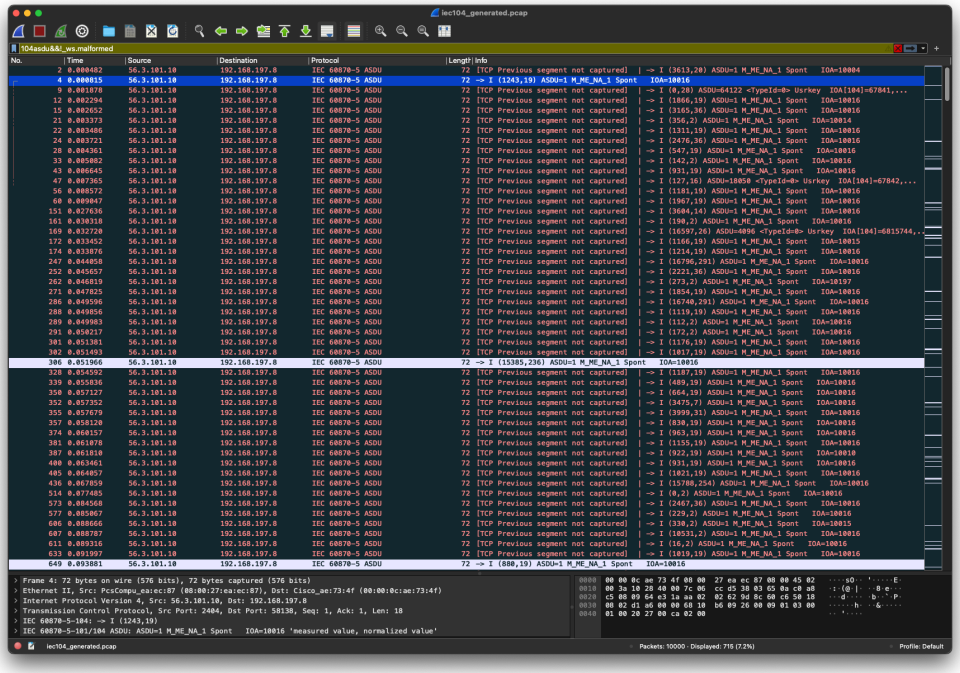


Figure 5.5: Wireshark Output - filtered - Test 11

of proper Modbus/TCP and IEC-104 datagrams is rather small in most cases. Generating larger amounts of samples would lead to more datagrams, but of course prolongs the execution time. The sampling process on the test hardware was done in a matter of seconds, however. Moreover, there was no particular performance requirement defined for the goal of this thesis, therefore execution times were neglected for the conducted experiments.

The generated capture files contain large portions of packets which only comply to TCP traffic or no protocol at all. The synthetic traffic is therefore not indistinguishable from real traffic. Instead, the unfiltered capture files in Figure 5.1 and Figure 5.2 look rather mangled. However, independently of the size of the processed datagrams, a large proportion of the trained generators was able to reproduce the structure and contents of the training data. This is especially noteworthy since SeqGAN was most often used to process much smaller sequences.

5.3 Discussion

During all experiments it was noticeable to me that the overall good performance of the reference implementation presented in [77] could not be achieved when training the architecture on datagrams in byte form. It became clear that without modifying the training algorithm, the GAN suffered from an almost

guaranteed mode collapse, when fully trained to convergence.

The generator was not able to produce a single Modbus/TCP or IEC-104 packet, after running the training algorithm until the loss of the generator converged. Instead all output samples contained a MAC address, followed by periodic repetitions (Listing 5.3) of sub-sequences of one or more hexadecimal values. Yet, at the same time I observed that the baseline MLE-trained generator was able to produce datagrams as per protocol definition. Thus my assumption was that the SeqGAN training was not properly optimized for the task at hand.

5.3.1 Training Instability

It became clear to me that the hyperparameters required further adjustment. I therefore modified the generator as well as the discriminator model, which originally used the hyperparameters of the reference implementation, according to the findings of [22], where a similar behaviour could be observed.

I increased the number of hidden units of the generator's LSTM layer from originally 64 to 384, 512 and 1024, respectively, to cope with the larger input samples than the sequences of size 20 in [77]. Test 12 and 13 used a slightly more complex generator model with an LSTM layer consisting of 1024 hidden units. I chose this model with increased capacity to test whether a further increase of the hidden units would lead to better performance, when processing longer sequences or even better performance overall. This seemed appropriate since the selection of 512 unit in [22] was based on a sequence length of 100. Increasing the number of hidden units alone did not lead to a significantly different outcome, however.

Moreover, I greatly reduced the number of convolutional layers of the discriminator model from originally 12 to 6 and their filter sizes were increased. This was due to the finding that for the used training data, the less complex model performed well enough. These two optimizations led to a better yield during testing, however, the model still suffered from model collapse.

As a next step I examined whether adapting the policy gradient update rate of the generator would further improve the training. In contrast to the suggestions of [22], I did not increase the update rate, but instead largely decreased it, which resulted in a much larger yield of syntactically correct packets.

I observed that a lower update rate nearly always meant a better result. The update rate was then fixed at 0.1, except for the first test where a lower learning rate was tried out to accommodate for the low number of training samples. This setting represented the optimum for the conducted tests.

Nevertheless the training process still remained highly unstable and training until convergence led to mode collapse. Since the baseline generator was able to generate the desired datagrams, the idea then arose to closer examine the generator mode collapse. Firstly, I examined whether the mode collapse

occured after a particular training step or continuously. Secondly, I examined whether there was a moment in training at which the generator would overcome the mode collapse situation after training it long enough.

Since the generator's weights were saved after every train step, it was possible to evaluate the generator at each step. Doing so revealed that in most cases the adversarial training improved the generator, sometimes greatly. However, after a certain number of steps, which varied depending on the training data and hyperparameter setting, the generator's performance continuously degraded. This behaviour could be recreated well and was vastly different to the findings of [77].

To examine whether the mode collapse would be overcome, I conducted an experiment with a largely increased number of training steps, to investigate whether there is such a moment in training. However, I could not observe any improvement over the course of 160 training steps. This number was considered high enough in relation to the number of required steps to observe an improvement in other experiments. At maximum 4 steps were required in test 5.

Thus the constant mode collapse still posed a problem, that required a more detailed examination. The proposed interleaved training in [31] intends to alleviate mode collapse. I therefore incorporated this enhancement of training into the existing training algorithm for test 14. During interleaved training, the training algorithm applied 20 supervised MLE-training steps after every 4 adversarial training steps. While the overall performance of test 5b was not achieved, a more consistent training was observed.

As suggested by the authors of [31] interleaved training enabled the model to overcome mode collapse. They assumed that the additional supervised learning steps would prevent the model from going too far away from the real data and thus provide an implicit form of regularization [31].

5.3.2 Considerations

Intuitively, there are two explanations for the results of the experiments. Firstly, the processed data samples were 3 to 15 times larger than those used in [77]. Secondly, byte sequences representing datagrams are structured differently than natural language texts or music, since datagrams contain large numbers of repeating values in their header information.

Additionally, the training data only contained very few different addresses, such as MAC, IPv4 and others. This is because the training capture files were obtained from rather small testbeds, consisting of few SCADA components. Independently of the number of training samples that is given to the model, the network receives the same information on many occasions. Demanding that the network is able to learn which values have to remain fixed while others are desired to be diverse is not ideal on the basis of this information.

The combination of reinforcement and adversarial learning SeqGAN uses is

a rather complex approach compared to conventional MLE-based training. In comparison, the less complex baseline generator, a plain MLE-trained LSTM, was already able to create valid datagrams. However, in most cases the adversarial training improved the yield of correct frames, sometimes by a lot. Therefore, I think that the utilization of SeqGAN is justified.

Moreover, the runtimes were in the range of minutes to a few hours. This was achieved by using fairly moderate hardware resources, the training utilized a single Tesla T4, a GPU from 2018, only 2 CPU cores and 13 gigabytes of RAM. Therefore, the complexity does not seem to pose a large problem, at least for the type of training data used.

Nevertheless, the adversarial training is computationally more complex than the basic MLE-training. For example, a single pre-train step in test 12 took about 9 seconds, while an adversarial training step took 159 seconds. However, the latter training time is largely dependant on the number of MCTS rollouts. A larger number of rollouts is intended to result in reduced variance and increased accuracy [77]. Yet choosing a larger number than 32 did not improve the instabilities in training.

The application of the MCTS rollouts makes the training process highly stochastic. This was observed in the fluctuations in the results in most of the experiments. This stochastic training process is a serious downside of the SeqGAN approach. It led to problems in reproducing the results of many tests and was the reason why each test had to be conducted a number of times.

Apart from the number of rollouts, the presented approach provides even more variables. The discriminator model appears to perform well enough for the used input data. After only a few train steps, it was able to distinguish real from synthetic samples with high accuracy. However, since the generator model was mostly unmodified in all of the tests, I think that an investigation of a different generator model is necessary. Therefore, in my opinion further research should cover an investigation whether a completely different neural network model for the generator would be more suitable. Nevertheless, the generator performed well enough for my experiments, therefore I left it mostly unmodified for better comparability.

In addition to an investigation of the generator model, the presented architecture would have to be trained on other more diverse capture files. I was not able to obtain any capture files that are more suitable for the task at hand, however, the used capture files are appropriate for the set target of generating datagrams that syntactically obey a certain protocol. Yet if one wanted to generate traffic that is able to interact with real hardware and cause changes in the system. The traffic would have to contain more payloads, such as measurements and control instructions. Moreover, it needs to contain more request-response transactions and the normalization during pre-processing would have to be adapted to this. For the conducted experiments, only specific sizes of input data were considered and in many cases this excluded the response to a particular request in the case of Modbus/TCP.

All in all, according to the results presented in this chapter, the main objective of utilizing a generative adversarial network to generate network traffic consisting of protocols used in SCADA systems was achieved. The adversarially trained generator is able to produce an arbitrary number of datagrams that are properly recognized by the widely used protocol analyzer Wireshark.

A large portion of the generated packets contain the address information of the original capture files and cover a variety of functions that are given by the protocol definition. A smaller portion of the traffic is even recognized as correct transmissions when analyzing it with Wireshark. Still, to utilize this approach, the results also suggests that as expected a larger amount of training data is required in the first place. This stands, however, in direct conflict with the underlying problem, namely that obtaining larger amounts of such data is difficult.

The most obvious problem of mode collapse was examined in detail and could be alleviated via interleaved training, it was now possible to train the network to convergence to reach peak performance.

With hindsight at this point in my research, when it became apparent that the presented SeqGAN approach is a viable option to generate SCADA network traffic, I looked for possible enhancements that could be applied to further improve the architecture.

The presented approach tried to generate the whole Ethernet frame, which led to less complex pre-processing. Since generating shorter sequences tended to be more stable it could be tried to only generate a TCP frame or a TCP payload and incorporate it into existing Ethernet frames. I discarded this idea since I wanted to test whether it was possible to generate the whole frame. Nevertheless it is a viable option, depending on the use-case.

According to [31], the training instability could also be explained by a vanishing gradient problem, that is caused by a much stronger discriminator problem in comparison to the generator. In this case, the reward signal is too weak to meaningfully update the generator's parameters. A similar finding was made by [22]. It was therefore proposed to rescale the reward signal before applying it during parameter update.

During my experiments, I made the finding that a lower update rate leads to a more stable training which is similar to the authors of [63]. They suggest that it could be a characteristic, showing that adversarial learning may not be the ideal approach. A solution to this might be provided by the findings in [29].

According to the authors of [29] the use of the policy gradient algorithm theoretically allows to extend the SeqGAN approach with multiple, arbitrary reward signals. A similar idea, namely to introduce auxiliary reward signals, was also proposed by [22]. The core idea of ORGAN [29] is a modification in the definition of the reward signal. In their experiments, they tried to optimize the generation of molecules in a pharmaceutical context. They changed the reward function to a linear combination of multiple reward signals. Instead of

only using the reward given by the discriminator. In their experiments, they defined the numerical reward as the weighted average of the discriminator output and a domain specific metric.

The proposed model [29] thus becomes a hybrid between a naive reinforcement learning model and a SeqGAN. The domain specific metric can, for example, be any kind of static analysis, that ensures that the generated samples possess certain desired traits. This idea could also be applied towards the generation of network packets. The problem of mode collapse could possibly be overcome by penalizing the generation of arbitrary, repeating byte sequences. The required additional reward signal could be provided by a packet analyzer. It would have the task to flag datagrams, that completely disobey a certain structure.

Another possibility to stabilize training might be provided by an adaptation of LeakGAN [31] for the given task. The authors of [31] suggested the application of LeakGAN as an enhancement of SeqGAN for generating sequences of more than 20 tokens. LeakGAN tries to solve the problem that the discriminator only provides a sparse reward signal by evaluating complete sequences. SeqGAN tackled this problem via stochastic MCTS rollouts, which led to large fluctuations in the training outcome, as emphasised before.

The authors of [31] proposed to incorporate the discriminator more deeply into the training process. SeqGAN used the discriminator model as a blackbox that outputs a sparse reward signal only. This disregards a lot of information although the discriminator CNN's internal states are known. The core idea of LeakGAN is therefore to leak extracted hierarchical features to the generator. These features represent sub-goals for the generator which serve as additional guidance. The generator is then not required to predict a token that is fitting for the entire sequence, but rather for a shorter salient part of the sequence.

All in all, these enhancements are viable options for future research, I did, however, not apply them in this thesis since they would have been beyond the defined goal. I wanted to show that the presented architecture is able to generate complete datagrams that are recognized as such by widely used software. In addition, I intended to show that the presented approach is a starting point that, through the use of reinforcement learning, provides a number of potential enhancements that could be applied depending on a particular future use case.

6 Conclusion

Processes in the critical infrastructure often utilize the SCADA communication framework, one example for this are smart grids. Smart grids are electrical grids that utilizes digital technologies to enable communication between energy production and consumption sites. SCADA communication provides a way to monitor and control the production and distribution process remotely.

SCADA systems controlling smart grids are valuable targets for cyber attacks. Such attacks have already taken place several times in the past years, the last noteworthy attacks were associated with the attacks on the Ukrainian infrastructure in 2022 [18]. Cyber resilience of smart grids is therefore extremely important.

To be able to provide means for cyber resilience of smart grids, researchers require access to network traffic of SCADA systems. However, since they are part of critical infrastructure, obtaining such traffic is challenging. The few publicly available SCADA traffic datasets seldomly contain large quantities of malicious traffic. One option to acquire appropriate traffic presents the set up of testbeds in which the communication can be captured. These testbeds, however, are only models and may not completely represent real SCADA systems. Additionally, the construction of these models is complex and can be cost-intensive and access to these testbeds for research is often limited.

Being able to utilize a software solution to create arbitrary amounts of synthetic network traffic samples would be desirable due to the aforementioned reasons. To conduct research into new kinds of attacks, the solution would have to be able to generate benign and malicious SCADA network traffic.

Generative adversarial networks are a means to generate arbitrary amounts of synthetic data. They are often utilized to generate synthetic images or texts. It stands to reason whether a GAN would be able to generate sequential data such as byte sequences. However, there are only few published approaches in the research field of generating network traffic via GANs. There are especially none, which try to generate datagrams of communication protocols of SCADA systems.

Since not much work in this research field has been done yet, this thesis was concerned with an investigation into whether GANs would be suitable to generate datagrams stemming from SCADA systems at all. I therefore examined whether a particular GAN architecture would be able to capture and reproduce the structure of two specific SCADA protocols.

The goal was to obtain an adversarially trained generator that is able to generate arbitrary numbers of syntactically correct packets that obey a particular

SCADA protocol. The given definition of correctness means that the generated packets obey the protocol definition and are recognized as Modbus/TCP or IEC-104 datagrams by a widely used protocol analyzer. I used Wireshark for the conducted experiments.

I made the decision to process whole Ethernet frames since I wanted to answer the question whether the GAN would be able to generate complete datagrams. Additionally, this makes the pre- and post-processing less complex. The processed datagrams were represented as sequences of 2-digit hexadecimal bytes. I chose to use the byte sequence representation, because I found it to be desirable to avoid the need for domain knowledge and manual intervention during pre- and post-processing. This data representation makes it possible to write the whole sequence into a capture file via Scapy, instead of instructing it where particular values belong. The process of learning the structure would in the latter case be performed by the researcher and not by the machine, in my opinion. Moreover, the used representation has the additional benefit that a GAN that is able to generate byte sequences would most likely be able to generate traffic of various network protocols without much adaptation. Finally, I chose to use the 2-digit representation due to the comparatively small vocabulary consisting of 256 entries, which presents a good compromise between sequence length and vocabulary size.

The presented approach used a generative adversarial network particularly specialized for processing sequential data. The utilized GAN architecture is called SeqGAN [77] and intends to solve the problem that while conventional GANs are able to generate continuously valued data, they are, however, not well suited to generate discrete values.

SeqGAN interprets the process of generating sequences as sequential decision-making, which means that the selection of each byte is decided based on the already selected ones. The optimization of this selection process is conducted using a policy gradient algorithm, a method that originates from reinforcement learning.

To evaluate the performance of the used SeqGAN model, I conducted multiple tests during which the trained generators were used to generate fixed numbers of samples. The adversarially trained generators were evaluated and compared against conventionally trained baseline generators. These were trained using maximum-likelihood estimation.

Afterwards, the obtained samples were converted into capture files, which were then analyzed using the command-line version of Wireshark. The baseline and adversarially trained generators were then compared on the basis of the number of recognized packets. Additionally, I examined the packet content of the generated traffic of the best performing generators. I investigated whether the contained address information reproduced the addresses of the input data and whether the TCP sequence and acknowledgment numbers were set correctly.

My results showed that the model was able to generate SCADA traffic,

with one exception. This exception is, however, important, as it was the test that processed especially small amounts of training data that failed. The observed results therefore suggest that, as expected, a large amount of training data is required. This observation, however, stands in direct conflict with the underlying problem, namely the difficulty of obtaining larger amounts of such data.

In the majority of the experiments, I observed a significant improvement of the adversarially trained generator over the baseline. However, the training process proved to be rather unstable. The GAN was very susceptible to mode collapse if trained for too many steps. After being trained to convergence, it generated repetitive sequences of the same bytes. Those samples therefore did not represent a particular protocol and were unsuitable.

I therefore conducted a test to examine the effect of alternating supervised and adversarial learning steps. Afterwards, I was able to observe that this modification stabilized training and enabled the model to overcome mode collapse situations. Moreover, it finally made it possible to train the model to convergence while continuing to produce viable results. The application of this interleaved training was proposed as a mitigation against mode collapse by [31].

When I adapted the hyperparameters, I observed that the model performed best, if the learning rate was set close to zero. A similar observation has been made by the authors of [63]. This reinforces my assumption that adversarial learning alone is not optimal for the input data. However, I think that further research in the field of combining adversarial and reinforcement learning is nevertheless justified, since during most experiments an improvement of the generator was observable.

My finding that the presented SeqGAN approach is able to generate datagrams has another advantage. The utilization of reinforce learning allows to incorporate additional domain specific metrics into the optimization of the generator. Such modifications of the definition of the reward signal have been proposed and tested by [29]. They defined the reward as a linear combination of multiple reward signals. When processing network traffic, this would provide a means to incorporate external protocol analyzers as reward signal.

As stated at the beginning, the overall goal of the proposed traffic generation is to provide means for developing novel intrusion detection systems. Therefore, the GAN needs to be able to generate benign and malicious SCADA network traffic. To this point, the presented approach will generate any form of traffic it is trained on. The aforementioned modification of the reward signal [29] may be a way to influence the GAN to produce benign and malicious traffic according to preference. Semantical correctness, therefore the requirement that generated packets would influence a system's state, was out of the scope of this thesis. However, semantical correctness would most definitely be desirable, if one wanted to generate malicious traffic. Depending on the utilized reward signal, the presented reinforcement learning based approach

may be a possibility to achieve semantical correctness. A natural next step would therefore be to modify the reward signal on the basis of state changes in a system.

Additionally to the utilization of an enhanced reward function, several other potential improvements of the presented approach are available for future research. Many other publications that are concerned with generating sequential data have also employed approaches based on SeqGAN. For example, to further improve the stability of the training process, a rescaling of the reward signal has been proposed by [31]. It would prevent the vanishing gradient in cases where the discriminator is a lot stronger than the generator.

If this improvement by itself does not provide sufficient results, a further extension of the SeqGAN approach is possible, namely to adapt LeakGAN [31] to the given task. In order to ease the task of the generator, LeakGAN only requires the generator to predict tokens for shorter salient parts of sequences.

While these are viable options for future research, I did, however, not use them in this thesis because they would have gone far beyond the defined goal. I wanted to show that the presented architecture is capable of generating complete datagrams that are recognized as such by widely used software. Furthermore, I wanted to show that the presented approach is a starting point that provides a number of potential extensions through the utilization of reinforcement learning. These extensions might be a way to influence the generation process depending on the use-case and could be applied in future research.

However, in my opinion the next step of the research begun in this thesis should be concerned with further improvement of the packet generation first. A starting point could be an adaptation of ORGAN [29] to the conducted experiments combined with the proposed rescaling of the reward signal. The second reward signal could then be provided by a packet analyzer as additional metric. ORGAN uses very similar models for the generator (LSTM) and the discriminator (CNN). The performance would therefore be comparable. An additional adaptation of LeakGAN would be a viable option in the case that ORGAN does also show significant instability in training. Future tests should, however, involve more appropriate training data.

Finally, the findings of this thesis are that a GAN that uses a combination of adversarial and reinforcement learning is able to generate arbitrary numbers of datagrams, that obey a particular SCADA communication protocol. This is an important finding since there are no other publications that were concerned with this problem. Moreover, I provided an implementation in the form of a Jupyter Notebook that can be used to reproduce my results [62]. Additionally, the used form of generative adversarial network offers enhancements that potentially make it possible to influence the generation process towards a specific goal. All in all, these findings represent a starting point for more extensive research in this field even if potential enhancements still need to be tested in further research.

Bibliography

[1] 4SICS.se. Capture files from 4sics geek lounge. https://www.netresec.com/?page=PCAP4SICS, 2015.

[2] S. Adepu, N. K. Kandasamy, and A. Mathur. Epic: An electric power testbed for research and training in cyber physical systems security. In S. K. Katsikas, F. Cuppens, N. Cuppens, C. Lambrinoudakis, et al., editors, *Computer Security*, pages 37–52, Cham, 2019. Springer International Publishing.

[3] A. Ali. How i used ai to make fake people (gans) — and why it matters. https://go.wwu.de/1vz9p, 2018.

[4] M. Almgren, P. Andersson, G. Björkman, M. Ekstedt, et al. Rics-el: Building a national testbed for research and training on scada security (short paper). In *CRITIS*, 2018.

[5] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *ICML*, 2017.

[6] P. Bachman and D. Precup. Data generation as sequential decision making. *ArXiv*, abs/1506.03504, 2015.

[7] D. Bahdanau, P. Brakel, K. Xu, A. Goyal, et al. An actor-critic algorithm for sequence prediction. *ArXiv*, abs/1607.07086, 2017.

[8] BBC News. Ukrainian power grid 'lucky' to withstand russian cyber-attack. https://www.bbc.com/news/technology-61085480, 2022. Accessed: 2022-07-11.

[9] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. In *J. Mach. Learn. Res.*, 2000.

[10] Y. Bengio, P. Y. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5 2:157–66, 1994.

[11] P. Biondi. Scapy documentation - introduction. https://scapy.readthedocs.io/en/latest/introduction.html, 2022. Accessed: 2022-12-18.

[12] C. M. Bishop. *Neural networks for pattern recognition*. Oxford university press, 1995.

[13] C. M. Bishop and N. M. Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[14] F. Björck, M. Henkel, J. Stirna, and J. Zdravkovic. Cyber resilience – fundamentals for a definition. In A. Rocha, A. M. Correia, S. Costanzo, and L. P. Reis, editors, *New Contributions in Information Systems and Technologies*, pages 311–316, Cham, 2015. Springer International Publishing.

[15] Y. Chen, B. Perozzi, R. Al-Rfou, and S. Skiena. The expressive power of word embeddings. *ArXiv*, abs/1301.3226, 2013.

[16] A. Cheng. Pac-gan: Packet generation of network traffic using generative adversarial networks. *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 0728–0734, 2019.

[17] Q. Cheng, S. Zhou, Y. Shen, D. Kong, and C. Wu. Packet-level adversarial network traffic crafting using sequence generative adversarial networks. *ArXiv*, abs/2103.04794, 2021.

[18] A. Cherepanov and R. Lipovsky. Industroyer2 sandworms cyberwarfare targets ukraines power grid again. `https://go.wwu.de/v958a`, 2022. Accessed: 2022-12-15.

[19] G. Combs et al. Tshark manual page. `https://www.wireshark.org/docs/man-pages/tshark.html`, 2022. Accessed: 2022-12-18.

[20] G. Combs et al. Wireshark manual page. `https://www.wireshark.org/docs/man-pages/wireshark.html`, 2022. Accessed: 2022-12-18.

[21] C. de Masson d'Autume, M. Rosca, J. W. Rae, and S. Mohamed. Training language gans from scratch. In *NeurIPS*, 2019.

[22] S. gil Lee, U. Hwang, S. Min, and S. Yoon. A seqgan for polyphonic music generation. *ArXiv*, abs/1710.11418, 2017.

[23] J. Goh, S. Adepu, K. N. Junejo, and A. P. Mathur. A dataset to support research in the design of secure water treatment systems. In *Critical Information Infrastructures Security*, 2016.

[24] N. Goldenberg and A. Wool. Accurate modeling of modbus/tcp for intrusion detection in scada systems. *Int. J. Crit. Infrastructure Prot.*, 6:63–75, 2013.

[25] I. Goodfellow. Generative adversarial networks for text. `https://go.wwu.de/dcj3f`, 2016. Accessed: 2022-10-19.

[26] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2016.

[27] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, et al. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014.

[28] Google. Tensorflow guide - word embeddings. `https://www.tensorflow.org/text/guide/word_embeddings`, 2022. Accessed: 2022-09-16.

[29] G. L. Guimaraes, B. Sánchez-Lengeling, P. L. C. Farias, and A. Aspuru-Guzik. Objective-reinforced generative adversarial networks (organ) for sequence generation models. *ArXiv*, abs/1705.10843, 2017.

[30] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of wasserstein gans. In *NIPS*, 2017.

[31] J. Guo, S. Lu, H. Cai, W. Zhang, Y. Yu, and J. Wang. Long text generation via adversarial training with leaked information. *ArXiv*, abs/1709.08624, 2017.

[32] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, USA, 2nd edition, 1998.

[33] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

[34] G. Hoogsteen. *A cyber-physical systems perspective on decentralized energy management*. PhD thesis, University of Twente Enschede, 2017.

[35] F. Huszár. How (not) to train your generative model: Scheduled sampling, likelihood, adversary? *ArXiv*, abs/1511.05101, 2015.

[36] ICS-CERT. Ics alert (ir-alert-h-16-056-01): Cyber-attack against ukrainian critical infrastructure. `https://www.cisa.gov/uscert/ics/advisories/ICSA-10-272-01`, 2016. Accessed: 2022-07-11.

[37] IEC. Telecontrol equipment and systems–part 5-101: Transmission protocols–companion standard for basic telecontrol tasks. `https://webstore.iec.ch/preview/info_iec60870-5-101%7Bed2.0%7Den.pdf`, 2003. Accessed: 2022-11-28.

[38] IEC. Telecontrol equipment and systems–part 5-104: Transmission protocols–network access for iec 60870-5-101 using standard transport profiles. `https://webstore.iec.ch/preview/info_iec60870-5-104%7Bed2.0%7Den_d.pdf`, 2006. Accessed: 2022-11-28.

[39] International Energy Agency (IEA). Technology roadmap - smart grids. technical report. `https://www.iea.org/reports/technology-roadmap-smart-grids`, 2011. Accessed: 2022-07-11.

[40] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling. Semi-supervised learning with deep generative models. *ArXiv*, abs/1406.5298, 2014.

[41] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[42] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[43] A. Lemay and J. M. Fernandez. Providing scada network data sets for intrusion detection research. In *CSET @ USENIX Security Symposium*, 2016.

[44] C.-Y. Lin and S. Nadjm-Tehrani. Understanding iec-60870-5-104 traffic patterns in scada networks. *Proceedings of the 4th ACM Workshop on Cyber-Physical System Security*, 2018.

[45] H. Liu, Z. Zhou, and M. Zhang. Application of optimized bidirectional generative adversarial network in ics intrusion detection. In *2020 Chinese Control And Decision Conference (CCDC)*, pages 3009–3014, 2020.

[46] W. Luo, Y. Li, R. Urtasun, and R. S. Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *NIPS*, 2016.

[47] P. Matoušek. Description and analysis of iec 104 protocol. *Faculty of Information Technology, Brno University o Technology, Tech. Rep*, 2017.

[48] P. Maynard, K. McLaughlin, and B. Haberler. Towards understanding man-in-the-middle attacks on iec 60870-5-104 scada networks. In *International Symposium for ICS & SCADA Cyber Security Research*, 2014.

[49] P. Mishra, V. Varadharajan, U. Tupakula, and E. S. Pilli. A detailed investigation and analysis of using machine learning techniques for intrusion detection. *IEEE Communications Surveys & Tutorials*, 21(1):686–728, 2019.

[50] Modbus Organization, Inc. Modbus messaging on tcp/ip implementation guide v1.0b. `https://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf`, 2006. Accessed: 2022-11-28.

[51] Modbus Organization, Inc. Modbus application protocol specification v1.1b3. `https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf`, 2012. Accessed: 2022-11-28.

[52] W. D. Mulder, S. Bethard, and M.-F. Moens. A survey on the application of recurrent neural networks to statistical language modeling. *Comput. Speech Lang.*, 30:61–98, 2015.

[53] NIST. Nist special publication 800-82 rev. 2: Guide to industrial control systems (ics) security. `https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-82r2.pdf`, 2015. Accessed: 2022-07-11.

[54] Q. S. Qassim, N. Jamil, I. Z. Abidin, M. E. B. Rusli, et al. A survey of scada testbed implementation approaches. *Indian journal of science and technology*, 10:1–8, 2017.

[55] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2016.

[56] P. I. Radoglou-Grammatikis and P. G. Sarigiannidis. Securing the smart grid: A comprehensive compilation of intrusion detection and prevention systems. *IEEE Access*, 7:46595–46620, 2019.

[57] R. S. Radvanovsky and A. McDougall. *Critical Infrastructure: Homeland Security and Emergency Preparedness, Third Edition*. CRC Press, 2013.

[58] A. Remke, J. Chromik, A. Flosbach, and B. R. Haverkort. Smart grid communication and cyber security. `https://go.wwu.de/4g8bh`, 2019. Accessed: 2022-07-11.

[59] M. Ring, D. Schlör, D. Landes, and A. Hotho. Flow-based network traffic generation using generative adversarial networks. *Comput. Secur.*, 82:156–172, 2018.

[60] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[61] R. A. Sarıtekin. seqgan-text-generation-tf2. `https://github.com/us/seqgan-text-generation-tf2`, 2021.

[62] G. Seifert. scadapacketgan. `https://zivgitlab.uni-muenster.de/g_seif03/scadapacketgan`, 2022.

[63] S. Semeniuta, A. Severyn, and S. Gelly. On accurate evaluation of gans for language generation. *ArXiv*, abs/1806.04936, 2018.

[64] M. H. Shahriar, N. I. Haque, M. A. Rahman, and M. Alonso. G-ids: Generative adversarial networks assisted intrusion detection system. In *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 376–385, 2020.

[65] D. Silver. Lectures on reinforcement learning. `https://www.davidsilver.uk/teaching/`, 2015.

[66] D. Silver, A. Huang, C. J. Maddison, A. Guez, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

[67] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15:1929–1958, 2014.

[68] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *ArXiv*, abs/1505.00387, 2015.

[69] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

[70] R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, 16:285–286, 2005.

[71] R. S. Sutton, D. A. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 1999.

[72] D. Upadhyay and S. Sampalli. Scada (supervisory control and data acquisition) systems: Vulnerability assessment and security recommendations. *Comput. Secur.*, 89, 2020.

[73] E. W. Weisstein. Convolution. `https://mathworld.wolfram.com/Convolution.html`, 2022. Accessed: 2022-12-15.

[74] E. Westring, A. Fundin, C.-Y. Lin, T. Gustafsson, and S. Nadjm-Tehrani. Ricsel21: A dataset with network attacks targeting iec-60870-5-104 in scada systems, 2021.

[75] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[76] Y. Yang, K. McLaughlin, T. B. Littler, S. Sezer, B. Pranggono, and H. F. Wang. Intrusion detection system for iec 60870-5-104 based scada networks. *2013 IEEE Power & Energy Society General Meeting*, pages 1–5, 2013.

[77] L. Yu, W. Zhang, J. Wang, and Y. Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, 2017.

Declaration of Academic Integrity

I hereby confirm that this thesis on *GAN based Network Traffic Generation for Communication Protocols used in SCADA Architectures* is solely my own work and that I have used no sources or aids other than the ones stated. All passages in my thesis for which other sources, including electronic media, have been used, be it direct quotes or content references, have been acknowledged as such and the sources cited.

Gerrit Seifert, Münster, March 20, 2023

I agree to have my thesis checked in order to rule out potential similarities with other works and to have my thesis stored in a database for this purpose.

Gerrit Seifert, Münster, March 20, 2023

Eidesstattliche Erklärung

Hiermit versichere ich, dass die vorliegende Arbeit über *GAN based Network Traffic Generation for Communication Protocols used in SCADA Architectures* selbstständig verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken – auch elektronischen Medien – dem Wortlaut oder Sinn nach entnommen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.

Gerrit Seifert, Münster, 19. Januar 2023

Ich erkläre mich mit einem Abgleich der Arbeit mit anderen Texten zwecks Auffindung von Übereinstimmungen sowie mit einer zu diesem Zweck vorzunehmenden Speicherung der Arbeit in eine Datenbank einverstanden.

Gerrit Seifert, Münster, 19. Januar 2023