BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS
ÚSTAV TELEKOMUNIKACÍ

ANALYSIS OF GPON FRAMES USING MACHINE LEARNING
ANALÝZA GPON RÁMCŮ S VYUŽITÍM STROJOVÉHO UČENÍ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR — **Bc. Adrián Tomašov**
AUTOR PRÁCE

SUPERVISOR — **Ing. Martin Holík**
VEDOUCÍ PRÁCE

BRNO 2020

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

Master's Thesis

Master's study field **Communications and Informatics**

Department of Telecommunications

Student: Bc. Adrián Tomašov ***ID:*** 187373

Year of study: 2 ***Academic year:*** 2019/20

TITLE OF THESIS:

Analysis of GPON frames using machine learning

INSTRUCTION:

The objective of the diploma thesis is to design and implement an algorithm using machine learning and TensorFlow for the analysis of selected parts of the GPON (Gigabyte Passive Optical Network) frame. A server with captured GPON frames in JSON (JavaScript Object Notation) format is available. The proposed algorithms should be implemented in Python. The output of the thesis is a theoretical part containing the method and possibilities of using TensorFlow and a description of the GPON frame. The practical part of the diploma thesis includes the design of the algorithm and its implementation in TensorFlow.

RECOMMENDED LITERATURE:

[1] TensorFlow: An open-source software library for Machine Intelligence. TensorFlow [online]. [cit. 2019-09-07]. Dostupné z: https://www.tensorflow.org/

[2] GÉRON, Aurélien. Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. Boston: O'Reilly Media, 2017. ISBN 978-1-491-96229-9.

Date of project specification: 3.2.2020 ***Deadline for submission:*** 1.6.2020

Supervisor: Ing. Martin Holík

prof. Ing. Jiří Mišurec, CSc.
Subject Council chairman

ABSTRACT

This thesis focuses on the analysis of selected part of GPON frame using machine learning algorithms implemented by using TensorFlow library. Considering that the GPON protocol is defined as a set of recommendations, implementation by various device vendors may be different to designed protocol. Therefore, an analysis by a push-down automaton is not sufficient. The main goal is to create a system of models using TensorFlow library in Python3 capable of abnormality detection in the communication. These models use various architectures of neural networks (e.g. LSTM, autoencoder) and focus on different types of analysis. This system learns from baseline traffic and notifies about irregularities found in the newly captured traffic. As a result, the system estimates the similarity level of current traffic compared to the baseline.

KEYWORDS

Autoencoder, GPON, LSTM, machine learning, neural network, passive optical network, Python3, TensorFlow

ABSTRAKT

Táto práca sa zameriava na analýzu vybraných častí GPON rámca pomocou algoritmov strojového učenia implementovaných pomocou knižnice TensorFlow. Vzhľadom na to, že GPON protokol je definovaný ako sada odporúčaní, implementácia naprieč spoločnosťami sa môže líšiť od navrhnutého protokolu. Preto analýza pomocou zásobníkového automatu nie je dostatočná. Hlavnou myšlienkou je vytvoriť systém modelov za použitia knižnice TensorFlow v Python3, ktoré sú schopné detekovať abnormality v komunikácií. Tieto modely používajú viaceré architektúry neuronových sietí (napr. LSTM, autoencoder) a zameriavajú sa na rôzne typy analýzy. Tento systém sa naučí na vzorovej vzorke dát a upozorní na nájdené odlišnosti v novozachytenej komunikácií. Výstupom systému odhad podobnosti aktuálnej komunikácie v porovnaní so vzorovou komunikáciou.

KĽÚČOVÉ SLOVÁ

Autoenkodér, GPON, LSTM, neuronová sieť, pasívna optická sieť, Python3, strojové učenie, TensorFlow

TOMAŠOV, Adrián. *GPON frame detection by using TensorFlow*. Brno, 2020, 76 p. Master's Thesis. Brno University of Technology, Faculty of Electrical Engineering and Communication, Department of Telecommunications. Advised by Ing. Martin Holík

ROZŠÍRENÝ ABSTRAKT

S rastúcim dopytom poskytovaných služieb rastú aj požiadavky na distribučné siete, ktoré musia podporovať rôzne typy služieb. Každá z nich spotrebuje výraznú časť dostupnej šírky pásma. Aby poskytovatelia sieťových služieb a výrobcovia zariadení udržali krok s rastúcim dopytom na trhu, musia napredovať vo vývoji distribučných sietí.

Jeden z krokov v evolúcií je výmena prenosového média z metalických (medených) káblov na optické. Ich výhodou je, že podporujú dátový prenos na veľké vzdialenosti, sú odolné voči elektromagnetickému rušeniu a dosahujú oveľa väčšie prenosové rýchlosti. Prenos cez optické médium s použitím pasívnych optických prvkov sa uplatňuje aj v distribučných sieťach pre koncových užívateľov, kde sa využíva stromová topológia. Takýto typ siete vyžaduje aj nový komunikačný protokol/štandard, ktorým je napríklad GPON. V prípade Medzinárodnej telekomunikačnej únie (ITU) sú všetky protokoly definované ako odporúčania, takže každý výrobca si ich môže prispôsobiť podľa vlastných potrieb. Analýza takéhoto protokolu vyžaduje pokročilé techniky.

Strojové učenie, ktoré je súčasťou umelej inteligencie, rozhodne patrí medzi pokročilejšie techniky dátovej analýzy. Tieto algoritmy sú schopné naučiť sa rozoznávať rôzne vzory v dátach. Často sa používajú na klasifikáciu dát do príslušnej množiny, rozlišovanie vzorov alebo spracovávanie prirodzeného jazyka. Práve posledná zo zmienených oblastí je veľmi podobná problému, ktorou sa táto práca zaoberá a to je analýza čiastočne známeho protokolu.

Jazyk Python3 je veľmi populárny medzi vedcami z oblasti dolovania dát a strojového učenia aj kvôli svojej jednoduchosti a obratnosti. Existuje niekoľko knižníc pre strojové učenie, ktoré majú jadro napísané v inom jazyku ako Python3 a poskytujú iba aplikačné rozhranie. Týmto získame výhody oboch jazykov, takže výsledná knižnica je rýchla a jednoduchá na používanie s podporou hardvérovej akcelerácie. Medzi tieto knižnice patrí napríklad TensorFlow, ktorá tvorí jadro analyzujúcich modelov v tejto práci.

Skôr než je vyhotovený návrh analyzačného systému je dôležité overiť, či je takáto analýza pomocou umelých neurónových sietí vôbec možná. Preto je vytvorených niekoľko experimentov s rôznymi modelmi strojového učenia, ktoré overia schopnosti detekcie neštandardnej komunikácie a slúžia ako vzor pre implementáciu modelov v konečnom analyzátore. Analýza správ sa zameriava na dve oblasti kontroly. Prvá je syntaktická kontrola, ktorá overí či daná správa vyhovuje štandardu. Zameriava sa najmä na hodnoty v jednotlivých poliach `PLOAMd` správy. Za týmto účelom boli otestované a dva modely. Prvý z nich je `OneClassSVM`, z knižnice `scikit-learn`. Tento model sa učí charakteristické rysy zo vzorovej komunikácie, a na jej základe

vie určiť či nový analyzovaný vzor je, alebo nie je podobný vzorovej komunikácií. Proces učenia je len aproximácia n-rozmernej matematickej funkcie k učiacim dátam. So znižujúcou sa odchýlkou tejto funkcie sa zvyšuje citlivosť naučeného modelu na abnormálnu komunikáciu. V experimentoch tento model správne odhalil väčšinu náhodne generovaných správ a dokonca aj tých, ktoré boli podobné vzorovým správam.

Druhý model analyzujúci syntax správy je autoenkodér. Tento model sa skladá z dvoch menších modelov: kodér a dekodér. Kodér má za úlohu zredukovať počet dimenzií vstupnej správy a zakódovať správu do komprimovanej formy. Dekóder má komplementárnu funkciu ku kodéru a to rekonštrukciu pôvodnej správy z komprimovanej formy. Aby celý model správne pracoval, musí sa naučiť extrahovať dôležité informácie zo správy. Autoenkodér používa učenia bez učiteľa a tréningovú množinu tvoria prvky, ktorých vstup a očakávaný výstup majú rovnaké hodnoty. Autoenkodér naučený na vzorovej komunikácií je použitý na detekciu neštandardných správ tak, že analyzujúca správa je vyhodnotená a následne je spočítaná chyba siete pomocou chybovej funkcie. Pokiaľ je chyba menšia ako prahová hodnota, analyzátor usúdi, že daný vstup je podobný vzorovej komunikácií. Ak je chyba siete väčšia ako prahová hodnota, tak je správa považovaná za abnormálnu, takže v tomto prípade za neštandardnú.

Druhá oblasť je sémantická analýza, ktorá kontroluje nadväznosť jednotlivých správ a obsah jednotlivých polí medzi správami. Inšpiráciou pre túto analýzu sú neurónové siete spracovávajúce písanú ľudskú reč. Ich základ je tvorený z vrstiev LSTM buniek, ktoré si dokážu udržať vnútorný stav naprieč spracovávanými dátami. Analýza GPON štandardu je tomu veľmi podobná, pretože kontroluje správy v danej postupnosti. Vstupné dáta sú rozdelené do časových okien konštantnej dĺžky, aby bolo možné model učiť a následne identifikovať kde sa chyba nachádza. Tento model je schopný správne rozpoznávať správne a chybné sekvencie správ, ale kvalita jeho predikcie je závislá na učiacich dátach, ktoré musia obsahovať vyvážený počet vzorových sekvencií z oboch klasifikovaných množín.

Pre analýzu sémantiky bol tiež vyskúšaný autoenkodér, ktorý je v svojej podstate rovnaký ako autoekodér použitý v syntaktickej kontrole. Jedinou odlišnosťou je počet vrstiev a počet neurónov v každej vrstve. Vstupné dáta sú taktiež rozdelené do časových okien konštantnej dĺžky, ale naviac je každé okne ešte sploštené na jednorozmerný vektor. Tento model dokázal správne označiť väčšinu časových okien, ktoré boli úmyselne poškodené, ako neštandardné.

Navrhnutý GPON analyzátor sa skladá z niekoľkých komponentov. Ako prvý v poradí je čitateľ dát, ktorý dokáže načítať dáta rôznych formátov. Ďalší v poradí je dátový filter, ktorý pred-spracuje dáta do tvaru vyhovujúcemu požiadavkám vstupu modelov. Nasleduje sada modelov pre syntaktickú a sémantickú analýzu,

ktoré hľadajú odchýlky v komunikácií. Ako posledný v tomto návrhu je hodnotiteľ (Evaluator), ktorý analyzuje výsledky jednotlivých modelov a vyhodnotí, na koľko je daná komunikácia podobná vzorovej.

Navrhnutý model je implementovaný v jazyku Python3 s využitím objektovo-orientovaného paradigma, takže každá časť z návrhu predstavuje objekt a analyzátor len riadi tok dát a správ medzi komponentami. Hlavná funkčná časť analyzujúcich modelov je prevzatá z experimentov, ale je zjednotená na rovnaké rozhranie, aby narábanie s modelmi bolo jednotné. Všetky modeli sú implementované pomocou Tensorflow s použitím zjednodušeného rozhrania, ktoré definuje knižnica Keras.

Tento projekt je podporuje dva spôsoby používania. Prvý spôsob je spustenie programu priamo z príkazového riadku a pomocou argumentov meniť chovanie danej aplikácie. Druhý spôsob je používať analyzátor ako knižnicu, čo umožňuje väčšiu interakciu s jednotlivými modelmi, prípadne si definovať vlastné a zaradiť ich do analýzy. Súčasťou projektu je aj virtuálne prostredie, v ktorom sú špecifikované všetky externé knižnice, aby bol projekt ľahko spustiteľný na rôznych systémoch.

Na overenia kvality detekčných schopností systému pre analýzu GPON rámcov je vytvorených niekoľko úmyselne poškodených vzoriek komunikácie, ktoré obsahujú syntaktické aj sémantické chyby. Tieto vzorky sú následne analyzované naučeným systémom. Z výsledkov je jasne vidieť, že oba modely pre syntaktickú analýzu sú funkčné a zachytili väčšinu syntaktických chýb. Modely analyzujúce sémantiku taktiež objavili väčšinu vložených chýb s podobnou presnosťou. Testy obsahujúce chyby v komunikácií dokázali, že GPON analyzátor má schopnosti na odhalenie chýb v komunikácií a porovnať, ako veľmi sú dve sekvencie komunikácie podobné.

DECLARATION

I declare that I have written the Master's Thesis titled "GPON frame detection by using TensorFlow" independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the thesis and listed in the comprehensive bibliography at the end of the thesis.

As the author I furthermore declare that, with respect to the creation of this Master's Thesis, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll., Section 2, Head VI, Part 4.

Brno ....

author's signature

ACKNOWLEDGEMENT

I would like to express my gratitude to my supervisor Ing. Martin Holík for the useful comments, remarks and engagement through the learning process of this master thesis. Furthermore I would like to thank Ing. Václav Oujezský, Ph.D. for introducing me to the topic and for the useful tips and ideas on the way. I would like to thank my closed ones, who have supported me throughout the entire process, both by keeping me harmonious and helping me putting the pieces together. I will be grateful forever for your support.

Contents

List of Figures

List of Tables

Listings

Introduction

As customers service demands grow through time, requirements on service distribution network grow too. They need to support various types of services and each consumes a significant part of bandwidth. To keep pace with customer needs, distribution network has to evolve.

One of the evolution steps is to substitute old copper cables with modern fiber optic cables. Organizations largely replaced old copper with fiber optic cables in point-to-point world area networks, because of their bandwidth capabilities, long range and resistance of electromagnetic and radio frequency interference. These days, they have been replacing last mile distribution network, especially because of higher bandwidth capabilities and longer range. With new medium on physical layer, which creates tree-like topology, it is necessary to create new communication protocol. Several organizations take this opportunity and design various protocols and recommendation with support of diverse services. In case of International Telecommunication Union, all solutions are released as recommendations, which means device vendors may keep them in mind, but also can modify them a little according their special requirements. Considering these changes in recommended protocol, analysis and reliable verification process is much more difficult and requires more advanced techniques.

Machine learning (part of artificial intelligence science field) certainly belongs into advanced techniques of data analysis. These algorithms are able to recognize and learn various patterns based on learning dataset. They are widely used for classification, pattern recognition and natural language processing. The last of mentioned areas is very similar to our problem, which is analysis and classification of unknown language or protocol.

The main goal is to create and learn model, which should be able to detect new or different characteristics inside captured communication compared to referenced baseline (learning dataset). Those different characteristics may be new internal message type, bad frame field usage or anything else what is distinct from referenced communication.

In chapter 1, Gigabit-capable passive optical networks (GPON) are described with physical topology, network components and communication principles. Attention is focused on Physical line operations, administration and management downstream (PLOAMd) messages used to control units in passive optical network.

Chapter 2 describes several machine learning models, which are considered as a possible solution of this problem. Special attention is dedicated to definition of neural networks and learning algorithms used in this thesis.

In chapter 3, various Python3 machine learning libraries are described with spe-

cial focus on TensorFlow and Keras, because the final machine learning model is written using these libraries.

Chapter 4 consists of system architecture for GPON protocol analysis using machine learning techniques implemented in TensorFlow and Keras libraries. This system is designed to focus on frame structure analysis and relations between following messages. The output of this system is traffic similarity level compared to baseline (learning) traffic.

In chapter 5, several experiments made during the design process are described. Algorithms and models in this chapter are not considered as a final solution, but they are sufficient as a proof of concept demonstration and the final implementation may vary, but ideas and core of models are reused. Especially, the idea of time windows of specific length, which are helpful during learning and classification process. Time windows allow finding abnormal sequence in long communication.

Implementation details are described in chapter 6. It focuses mostly on an interface definition of various classes and a usage description. GPON analyzer is implemented as a python module and supports execution directly from command line or it can be used as a library used for further development of analyzing models.

Chapter 7 verifies GPON analyzer potential and discuss its detection capabilities. It also describes dataset generation process with corrupted messages and message sequences followed by their classification using learned analyzer.

1 Gigabit-capable passive optical networks

Gigabit-capable passive optical networks (GPON) is technology, which provides telecommunication and internet services over passive optical network (PON). It is considered as a great replacement for older technologies (e.g. digital subscriber line (DSL)), because it achieves much higher transmission speed for various types of traffic. As its name already suggested, fiber-optic cables are used as a transmission medium, which makes GPON capable of providing services for longer distance with higher transmission speed. Recommendation defines limit at 20 km, but it is possible to extend the range much farther.

GPON uses wavelength division multiplexing to separate upstream and downstream communication. Transmission speed supported by GPON in each direction is defined in table 1.1.

Tab. 1.1: Transmission rates supported by GPON systems.

Upsteam	Downstream
1.24416 Gbit/s	2.48832 Gbit/s
2.48832 Gbit/s	2.48832 Gbit/s

GPON is defined by several G.984.x recommendations defined by International Telecommunication Union - Telecommunication Standardization sector (ITU-T). This standard is designed to be backward compatible with previous ITU-T PON standard: asynchronous transfer mode passive optical network (APON), Brodband passive optical network (BPON).

1.1 Elements of PON

Passive optical networks consist of these basics elements: optical network unit (ONU), optical line termination (OLT) and optical distribution network (ODN). Logical topology of these components is shown in figure 1.1.

1.1.1 Optical distribution network (ODN)

ODN mostly uses passive network components, which provide transmission medium for GPON technology. These components connect single OLT and multiple ONUs or ONTs using optical cables and splitters creating a tree like topology, which can be also called point-to-multipoint. PON splitters have various splitting ratio $1 : N$, where N is usually multiple of 2. Maximum splitting ration in GPON is 1:128 [9].

1.1.2 Optical line termination (OLT)

OLT is the root of network tree and implements PON protocol (defined by ITU-T). It is also responsible for communication and administration of network leaves (ONU/ONT) according to ITU-T recommendations. It provides a bridge between GPON network and providing services as internet, video, voice and cable television. It is also responsible for registration/activation (described in section 1.6) of new ONUs into network, which includes bandwidth assignment (described in section 1.4) as well.

1.1.3 Optical network unit (ONU)

ONU[1] is a leaf of ODN nearing customer premises capable of communication using PON protocol and process PON PDUs. It provides bridge between PON and customer services by converting signal from optical medium into metal cable using different physical layer protocol and vice versa. It actively communicates with OLT to gain time slot for upstream data transmission (this process is described in section 1.4).

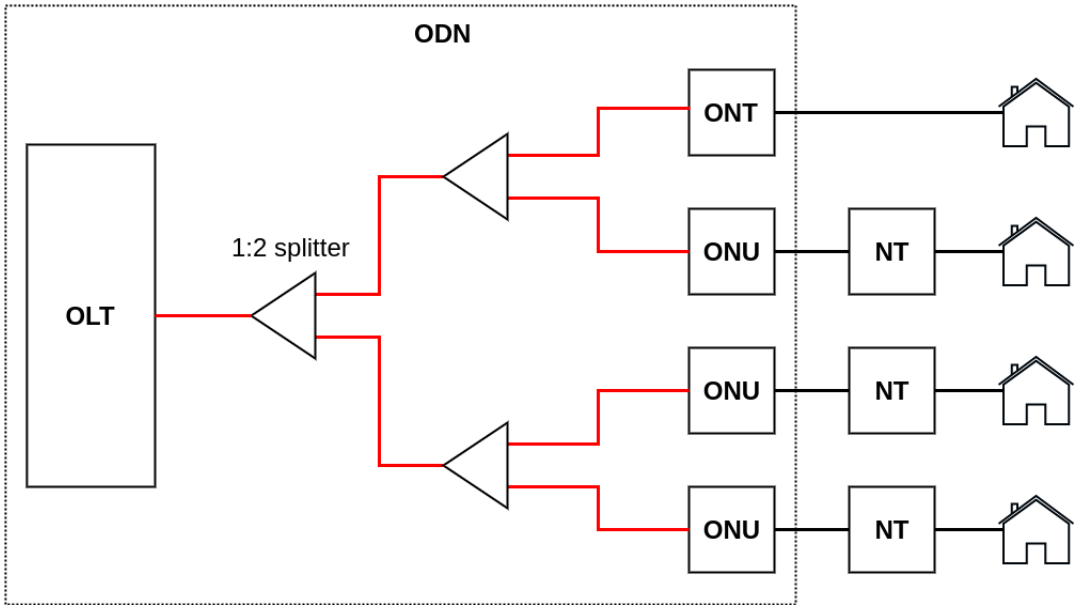

Fig. 1.1: Logical topology of passive optical network.

[1]In GPON recommendation, ONU mostly stands for both ONU and ONT in TC layer, except special cases. ONT is considered as a single-user ONU.

1.2 GPON time division multiplexing

ODN consists of a single optical cable, therefore GPON uses wavelength division multiplexing to separate upstream (from ONUs to OLT) and downstream (from OLT to ONUs) communication. In case ODN uses two optical cables, the second one is used only as a backup.

The process of gaining the access to the medium for downstream communication (generated by OLT) is centralized, because OLT is the only one who gets access to that media. It labels outgoing GPON encapsulation method (GEM) frames by GEM Port-ID, which identifies receiver's logical port. ONU filters the incoming GEM frame designated to itself based on GEM Port-ID.

In the upstream direction, there may be several ONUs communicating with OLT, therefore the process of gaining the access is decentralized. OLT assigns time based windows to ONUs during bandwidth allocation process. ONU uses GEM Port-ID to select specific logical connection to OLT.

1.3 GTC downstream frame structure

Frames sent by GPON transmission convergence (GTC) layer in downstream direction have constant time duration of 125 μs. At transmission speed 2.48832 Gbit/s, it represents 38880 bytes long frames [9]. Structure of this frame is graphically represented in figure 1.2.

Physical control block downstream (PCBd) contains information necessary for control and management of certain ONU. The most important are `Upstream BWmap` and `PLOAMd`. `Upstream BWmap` field gives ONU time slots for upstream communication bursts. `PLOAMd` field contains management message and it is more described in section 1.5.

`GTC payload` field contains list of variable length `GEM frames`. Generic encapsulation method (GEM) provides connection-oriented transport mechanism supporting variable payload length of various data services over PON. GEM encapsulation is analogy to asynchronous transfer mode (ATM) circuits. ATM was even supported as a transport mechanism in previous version of GPON recommendation, but today it is deprecated [9].

OLT and ONU ports (part of transmission container (T-CONT)) create virtual connection and label it with unique `PORT-ID` for proper identification. During transmission, `PORT-ID` in GEM header is set accordingly to identify receiving PORT.

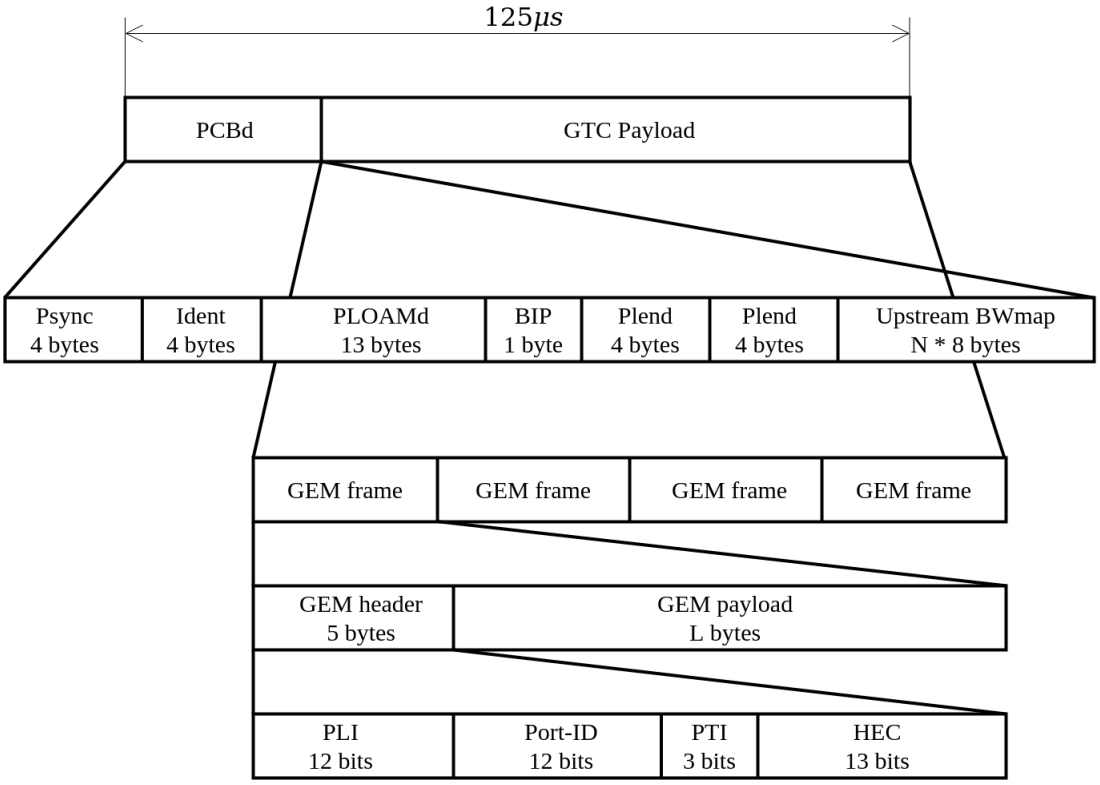

Fig. 1.2: Structure of downstream GTC frame.

1.4 Bandwidth allocation

From description of PON (in section 1.1), it is obvious that the PON is multiple access network, where transmission collisions may occur. To avoid this situation, OLT controls access of ONUs to transmission media. When ONU is willing to communicate over PON in upstream direction, it needs to have assigned communication time window. OLT assigns these time windows for each T-CONT of ONU via bandwidth allocation algorithm. Therefore, it sends Bandwidth Mapping message (BWMAP), which consists of several bandwidth allocations for specific ONT/ONU or its T-CONT [10].

1.4.1 Static bandwidth assignment

Bandwidth allocation process is approached by static or dynamic method. Static method assigns time windows to ONUs regardless of what they need. This may be beneficial for some technologies/services as VoIP, because of constant uplink bandwidth and stable delay [2]. However, for other IP services, which send data in

[2]The delay is low, because VoIP packets are usually small enough to fit into one transmission window.

bursts, it is not beneficial at all. After packet burst is sent and no more services are willing to transmit the data, the time window is still allocated for specific ONU. This prevents others to use this idle time window. Static allocation method is sufficient, if network is not congested or upstream bandwidth required by all ONUs is less than 1.244Gps[3] and is not fully utilized [10].

1.4.2 Dynamic bandwidth allocation

On the other hand, dynamic bandwidth allocation (DBA) method only assigns time windows to ONUs, which want to send upstream data. That means big packet bursts can be sent quicker by this method, because ONU might get longer time window for sending data. This method also cuts off ONUs, which do not have any data to send. OLT gets notification from ONUs indirectly by GEM idle frames or directly through buffer status reporting [11]. Dynamic allocation obviously utilizes transmission medium more effectively than static process, but level efficiency depends on DBA algorithm. GPON recommendation defines tools for DBA, but does not specify the whole allocation algorithm, which might be modified according to service providers needs. DBA enables them to oversubscribe PON, resulting in providing more bandwidth than they really have. This manner relies on customers, who do not use the whole provided bandwidth at the same moment.

1.5 PLOAM downstream message format

Physical layer operations, administrations and maintenance is one of three methods used by OLT to directly control ONUs in PON. It is widely used during ONU activa-

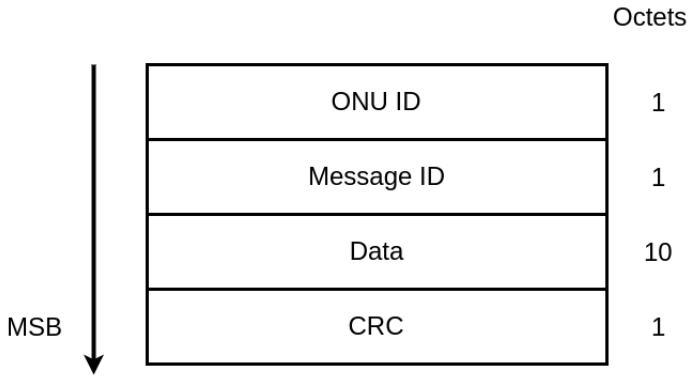

Fig. 1.3: PLOAM downstream message format.

[3] 1.244Gpbs is maximal upstream bandwidth defined by GPON recommendation by ITU-T.

tion process, configuration of encryption, management of keys and alarm signalling [9]. For communication, it uses PLOAM downstream message (PLOAMd), which is part of Physical control block (PCBd).

PLOAMd message is 13 bytes long and message format is shown in figure 1.3. Description of each field of PLOAMd message can be found in table 1.2.

Tab. 1.2: Ploam downstream message fields description.

Field name	Description
ONU-ID	This field represents receiving ONU. This number was assigned to specific ONU during its activation process. This field can cover values from range 0-253 for OLT -> ONU communication or 255 for broadcast.
Message ID	Type of PLOAM message is represented by this field. Message IDs are not in direct sequence $(1..N)$, but randomly assigned in one byte value space $(0 - 255)$.
Message Data	Each message can transport additional data and this field is allocated for this purpose. Format of data field may vary with respect to Message ID, but it has constant length.
CRC	Frame check sequence verifies data integrity of PLOAMd message. It contains remainder of division of the this message (with CRC set to 0) by generator polynomial $x^8 + x^2 + x + 1$.

1.5.1 PLOAMd messages

This subsection briefly describes PLOAMd messages defined in ITU-T G.984.3 GPON recommendation. Definition refers to several states of ONU activation process defined later in section 1.6. Types of PLOAMd messages with MessageID used in GPON are defined in table 1.3.

Tab. 1.3: PLOAMd messages definition.

ID	Message name	Message description
1	**Upstream Overhead**	When activation process starts, OLT instructs ONU with pre-assigned delay settings and number of preamble bytes for upstream communication. It also may set optical power of ONU's laser.
3	**Assign ONI-ID**	OLT assigns unique ONU-ID to specific ONU based on serial number and inform ONU via this message type.

Tab. 1.3: PLOAMd messages definition.

4	**Ranging time**	During Ranging state (O4) ONU measures equalization delay to synchronize itself for upstream communication. ONU sets this delay based on this message sent by OLT.
5	**Deactivate ONU-ID**	OLT by this message forces ONU to stop transmitting data in upstream direction, reset itself and start activation process from the beginning. OLT may broadcast this message to all ONUs.
6	**Disable Serial Number**	This message with *disable* option forces ONU to stop sending data, turn off the laser and move to Emergency state (O7). To enable ONU, OLT needs to send Disable Serial Number with *enable* parameter, which moves ONU to the state Standby state (O2).
8	**Encrypted Port-ID**	ONU is informed about channel encryption via this message.
9	**Request Password**	It is an optional message used for authentication of ONU against local password table stored in OLT.
10	**Assign Alloc-ID**	OLT uses this message to assign additional Alloc-ID to ONU, which has multiple T-CONTs.
11	**No message**	It is used when no PLOAM message is willing to be sent with transitioned GTC frame.
12	**POPUP**	After LoS/LoF alarm ONU moves to the POPUP state (O6). OLT can rescue ONU from this state by sending directed/broadcsated POPUP message.
13	**Request key**	If this message is sent by OLT, ONU needs to generate new encryption key and sent it to OLT.
14	**Configure Port-ID**	OLT assigns 12-bit GEM Port-ID to the individual logical connections via ONU management and control channel (OMCC), but OMCC needs this ID too. Therefore, OLT assigns Port-ID to OMCC via this message.
15	**Physical Equipment Error (PEE)**	OLT informs ONU about inability to send GEM and OMCC frames.
16	**Change Power Level**	OLT sets/tunes laser power of ONU by sending this message.
17	**PST message**	It verifies status of ONU $\Leftrightarrow$ OLT connection via PON.
18	**BER Interval**	It is used for evaluation of bit error rate.

Tab. 1.3: PLOAMd messages definition.

19	**Key Switching Time**	This message contains specific time, when ONU should use new encryption key.
20	**Extended Burst Length**	OLT forces ONU to use type 3 preamble.
21	**PON-ID**	This message contains mean optical launch power and PON-ID tag assigned by operator to specific interface of PON.
22	**Swift Popup**	OLT can force ONUs to move straightly to Operational state (O5) and clear LoS/LoF alarms with this message.
23	**Ranging Adjustment**	This message modifies equalization delay to correct synchronization drift. It can be sent to specific ONU or broadcasted to all.

1.6 ONU activation process

When ONU powers on, it cannot instantly communicate in PON network, otherwise it would cause carrier collision due to multiaccess nature of transmission media. Firstly, it needs to synchronize with OLT, get necessary IDs and activate itself, but most importantly, it acquires time slot for sending upstream data [8]. The thole life cycle of ONU in PON is defined by finite state machine, which is shown in figure 1.4. It also contains ONU activation process defined by the first four states of state machine. This process is responsible for initial communication with OLT, request of IDs and media access. All states of finite state machine are listed in table 1.4.

Tab. 1.4: ONU operational states.

ONU operational states of finite state machine	
ID	State name
O1	Initial state
O2	Standby state
O3	Serial_Number state
O4	Ranging state
O5	Operation State
O6	POPUP state
O7	Emergency stop state

In ONU finite state machine several timers are used to prevent getting stumbled

in specific state during activation process or loss of signal/frame error. These timers are described in table 1.5. Besides, it uses loss of signal (LoS) and loss of frame (LoF) flags to indicate transmission failure.

Tab. 1.5: Timers supporting activation process.

Timers used in activation process	
TO1	This timer is used to avoid getting state machine stuck in O3 or O4 state during unsuccessful activation process. It is also called serial number acquisition and ranging timer. Recommenced initial value is 10s [9].
TO2	Timer TO2 also called POPUP timer, avoids getting state machine stuck while waiting on POPUP message from OLT in state O6. Recommended value for this timer is $100ms$ [9].

`Initial state` (O1) is the first state coming after ONU powers on. In this state, ONU passively listens to communication in the PON and tries to detect M following `PSYNC` fields and then tries to detect $M-1$ whole frames. If this detection is successful, ONU moves to the next state O2 and clears LoF and LoS flags. Otherwise, it remains in this state until it receives necessary following uncorrupted data.

ONU in `Standby state` (O2) synchronizes itself in upstream direction. It waits for global network parameters e.g. delimiter value, power level mode and preassigned delay. All of these parameters are in `Upstream Overhead` message. When ONU receives this message, configures these parameters and moves to the next state O3 `Serial Number` state.

During `Serial Number state` (O3) ONU lets OLT know about its existence by sharing its serial number by responding on OLT request. To avoid collisions in the PON, OLT sends PLOAM messages with empty bandwidth map field, what creates quite time window for 250 μs [7]. Through this quite window ONU replies to OLT SN request with its own serial number. After this step, ONU waits to receive `Assign ONU-ID` message, which contains ONU-ID for this specific ONU. With successful assignment of ONU-ID it moves to the next state, the Ranging state (O4). OLT can use `Extended Burst Length` message and force ONU to configure received extended parameters and use the type 3 preamble lengths [9].

`Ranging state` (O4) is crucial for synchronization of upstream communication. All ONUs appear to be in equal distance from OLT even if they are not, in that case propagation delays are not equal as well. As a consequence of this situation,

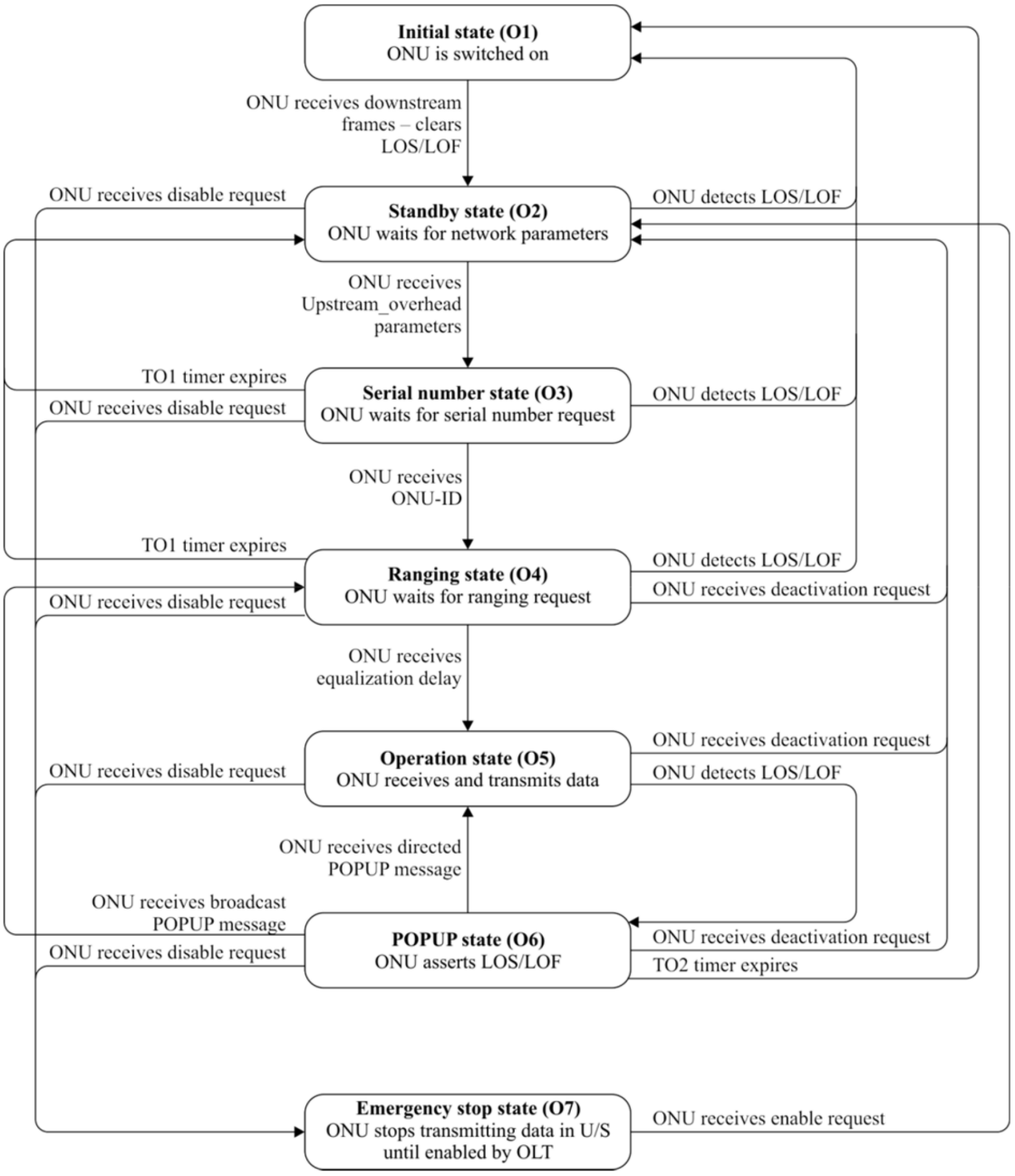

Fig. 1.4: Finite state machine describing ONU life cycle [9].

equalization delay is measured in this state. ONU waits for receiving `Ranging time` message and moves to the next state O5.

In `Operation state` (O5) upstream communication is synchronized between all ONUs. Thanks to equalization delay each message is received by OLT in correct upstream GTC frame and collisions in PON are avoided. ONUs are able to communicate with OLT via PLOAM message and send data in upstream direction. ONU

remains in this state until errors (LoS/LoF) occur or ONU is being disabled/deactivated.

When LoS or LoF alarm is activated, ONU moves to the `POPUP state` (O6) and stops sending upstream data. Transmission silence informs OLT that ONU is in this state. As the first step, ONU tries to recover from the error state by reacquiring signal or bring the frame synchronization back, which clears LoS/LoF alarm. When synchronization is achieved again, ONU waits on PLOAM message from OLT. If directed `POPUP` message is received, ONU moves to the state O5. Or if broadcast `POPUP` is received, ONU moves to the state O4 and measures equalization delay. If ONU is unable to recover signal and frame synchronization it moves to the state O1 and starts activation process from the beginning.

When ONU receives `Disable Serial Number` message with option *disable*, which means some malfunction of ONU occurs, ONU instantly appears in `Emergency state` (O7). OLT tries to send this message three times and if ONU still does not move to O7 state OLT sustains receiving upstream messages of ONU and asserts DFi alarm. In this state, laser has to be turned off and all upstream communication is prohibited. When malfunction is fixed, OLT sends `Disable Serial Number` message with option *enable* and ONU moves to the state O2 and negotiates all parameters from the beginning.

2 Machine Learning

Artificial intelligence is a scientific study, which research learning and data processing systems capable of individual decision. Machine learning, as a subset of artificial intelligence, studies system learning algorithms [4]. It is used in various cases, where conventional system designing (writing a computer program) is extremely hard to achieve. Examples of machine learning application are: image processing, voice recognition, suspicious task execution analysis and many others.

These days there are many algorithms oriented to machine learning (e.g. neural networks, decision trees, support vector, machines). Each algorithm has benefits and disadvantages depending on use-case of project. There are also many frameworks and libraries in various programming languages, where optimized versions of algorithms can be found.

2.1 Neural Networks

Neural networks (NN) are one of many implementation methods of machine learning systems. Inspiration was taken from discovered principles of human brain and the motivation was to create an artificial version of this complex system. The first designed system was perceptron and was able to learn classification of input space into two separate categories. With adding more perceptrons into single layer and stacking more of these layers, neural networks became to the world.

2.1.1 Perceptron

A core of these networks is an artificial neuron, a very simplified abstract replica of natural neuron. Graphical representation of perceptron is shown in figure 2.1.

The principle is straightforward. Neuron calculates sum of all inputs signals according to equation 2.1, where x_i is input signal, w_i is weight, θ represents bias and z is sum of all signals. This part is the same for across all neurons. Then the result is passed to the activation function, which evaluates the result of this specific neuron.

$$z = -\theta + \sum_{i=1}^{n} x_i * w_i \tag{2.1}$$

Each input signal is multiplied by specific weight, which represents its importance to the neuron [6]. If the sum of signals reaches certain level, the neuron is considered as activated. The bigger the weight is, the higher the impact of signal to activation function result will be. Weights are defined by the real number, so it also can be positive or negative number.

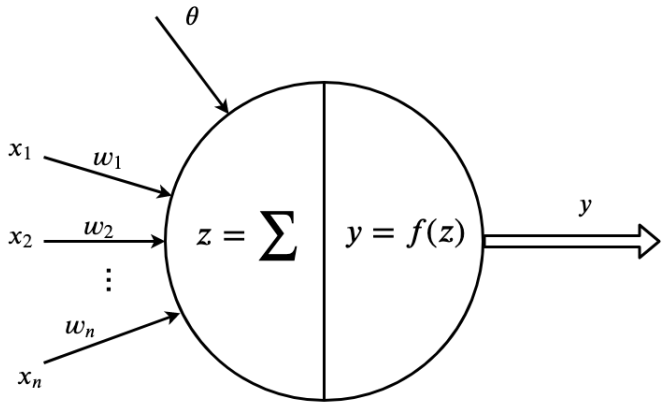

Fig. 2.1: An artificial neuron.

Definition of activation function is specified by type of neural network and position of neuron in this network (neurons in different layers may use other functions). It adds necessary non-linearity into NN model. Even though, each project using NN claims extra attention of model designer, who experiments with various combinations of activation function and chooses the best one. The most used activation functions are shown in figure 2.2.

2.1.2 Feed forward neural network

Huge number of artificial neurons separated into layers are connected together into neural network. These networks differ with connection scheme of neurons and their activation function. Inside feed forward neural networks, output of each neuron is connected into input of all neurons in the next layer (except output layer). Neurons in single layer are not connected together at all. In mathematical point of view, it creates an acyclic oriented graph with perceptron as a vertex and connections in between as edges. These neuron connections are called synapses and scale the value by weight. For better understanding see figure 2.3, where x is input vector, W_1, W_2, W_3 are weight matrices and Y is output vector. This network accepts vector of four components as an input and transforms it into vector of two (e.g. classification of vectors into two sets). The weight matrices have specific weight value for each synapse and each neuron. The dimension of current matrix in specific layer is given by number of neurons in previous and following layer.

Input and output layers create an exception in upper connection definition of NN. The main role of NN input layer is to distribute all parts of input vector to following perceptron in inner layer, therefore activation function might be linear.

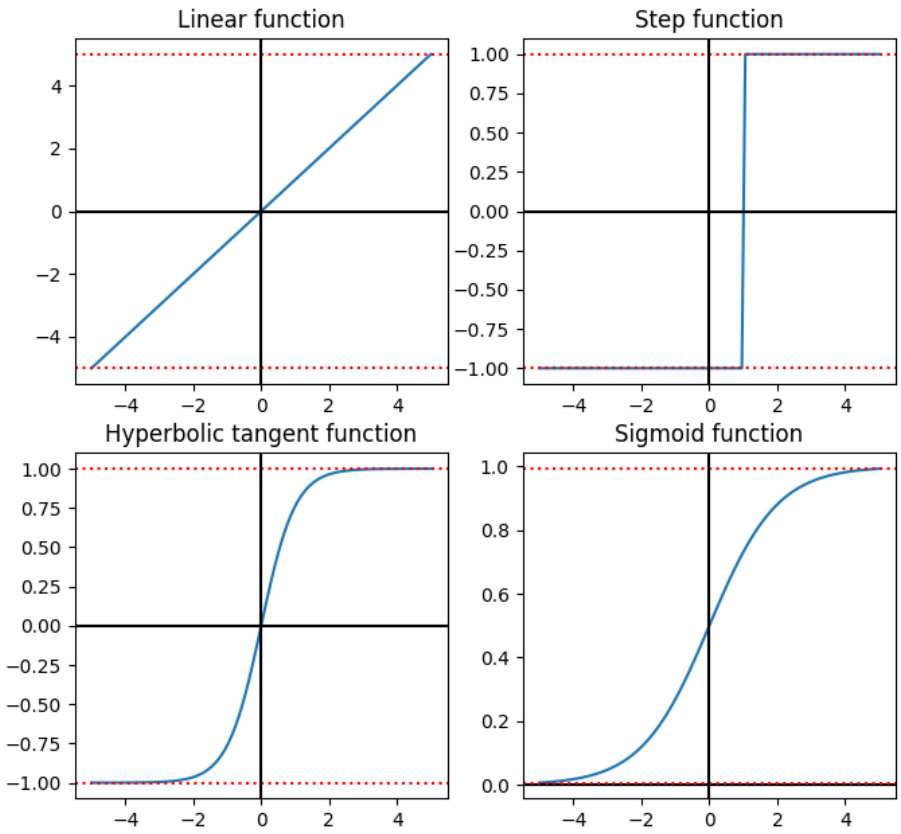

Fig. 2.2: Examples of neuron activation functions.

At the end of NN, there is output layer, which is not connected to anything, but provides aggregated output.

2.1.3 Backpropagation

The NN handling process is obvious compared to learning process. All calculations related to handling are performed by simple equations and the whole system remains persistent after handle.

Difficult task is to find weight matrices, that calculate output vector from input vector with the highest precision. This had been the biggest problem and source of negative opinions about NN, until backpropagation algorithm was found out. As the title of this algorithm indicates, it evaluates error of NN model and back propagate error from output layer to previous layers. Error is calculated using loss function (also called objective function), which can be defined by mean square error,

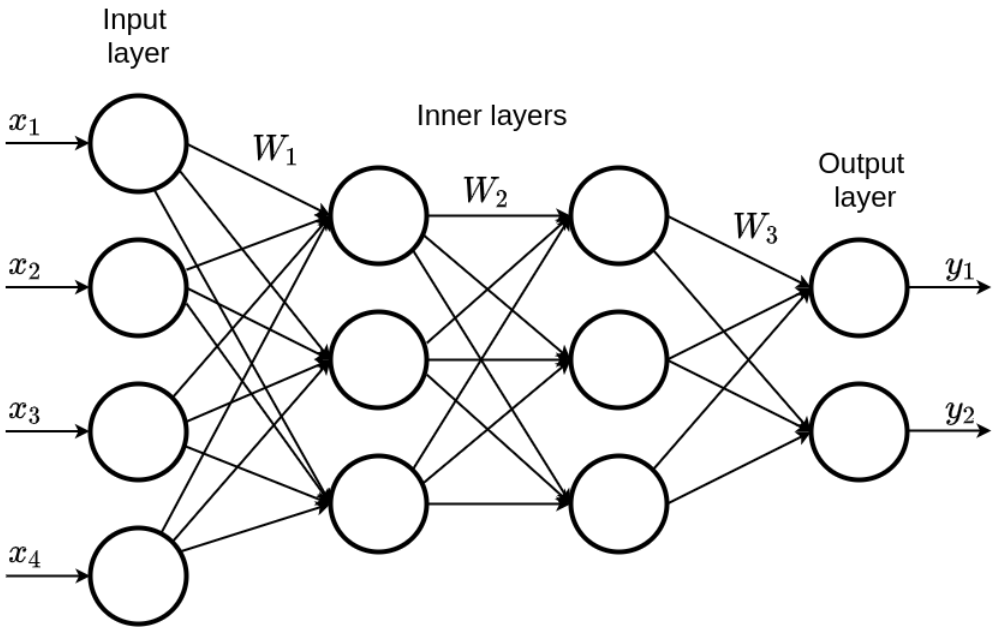

Fig. 2.3: A neural network scheme.

cross entropy or function with similar use case. It basically evaluates the difference between computed and expected output of NN.

Backpropagation algorithm is based on searching gradient of the weight with respect to loss function. Calculated gradient is used to modify weights in order to gain higher precision of NN model (minimizing loss) by applying one of the iterative learning method as a stochastic gradient descent or other alternative [15].

Example of error backpropagation

First of all, suitable learning and testing data are necessary to learn/train NN model by backpropagation algorithm. Both data sets (training, testing) need to be labeled with expected output for each input vector, which means these sets should contain pairs $(\mathbf{x}, \mathbf{d})$, where $\mathbf{x}$ is input vector and $\mathbf{d}$ is output vector.

Assume that NN described in figure 2.3 uses sigmoid as activation in all neurons. Sigmoid function is defined in equation 2.2.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

For this example of backpropaation algorithms we use mean square loss function defined in equation 2.3, where $\mathbf{y}$ is calculated output, $\mathbf{d}$ is expected output and N is dimension of $\mathbf{y}, \mathbf{d}$ vectors.

$$L(\mathbf{y}, \mathbf{d}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - d_i)^2 \tag{2.3}$$

Finding gradient with respect to i-th weight to k-th output neuron is shown in equation 2.4. By design of NN, evaluation of y from x, w and θ is accomplished by using several nested function, therefore we can use the chain rule when partial derivative is being searched.

$$\frac{\partial L}{\partial w_{ik}} = \frac{\partial L}{\partial y_k}\frac{\partial y_k}{\partial z_k}\frac{\partial z_k}{\partial w_{ik}} = (y_k - d_k)y_k(1 - y_k)x_i \tag{2.4}$$

The weights are changed based on gradients of error function calculated for the whole training dataset. Hence, gradients for each learning sample are summed together. If stochastic gradient descent method is used, aggregated gradient is firstly scaled by learning rate α and then added to original weight [15]. Final weight adjustment is demonstrated in equation 2.5.

$$w_{ij} = w_{ij} + \alpha\Delta w_{ij},\ where\ \Delta w_i j = \sum_{t=1}^{T}\frac{\partial L(\mathbf{y}_t, \mathbf{d}_t)}{\partial w_{ij}} \tag{2.5}$$

Another neuron parameter which needs to be learned is θ (the bias). Evaluation of θ difference is similar to weights. The only change is in partial gradient derivation of loss function, because it is derived with respect to θ not w. The rest of procedure is exactly the same.

2.2 RNN - recurrent neural network

Feed forward neural networks find their purpose in many areas, but they are not even close to be considered as a universal tool for data classification or categorization. Various patterns can be found and learned by NN in single sample, but patterns occurring across several samples are omitted. This causes difficulties during implementation of model for language analysis. Recurrent neural network might be a simple solution of this problem. RNN neuron does not calculate output based purely on input vector, but it considers its inner state as well.

2.2.1 RNN topology

To achieve connection with its previous state RNN creates cyclic graph by connecting output of neuron back to the input of the same neuron and passes state $\mathbf{h}$ through this connection. This adds extra matrix of weights W_{hh} for state vector $\mathbf{h}$. Scheme of RNN neuron is shown in left part of figure 2.4, where all inputs and outputs are defined as vectors, and weights are defined as matrices, because the whole neural network is represented by this scheme. In the right part, there is the same neuron, but rolled through time (through state vector $\mathbf{h}$). This visualization shows that

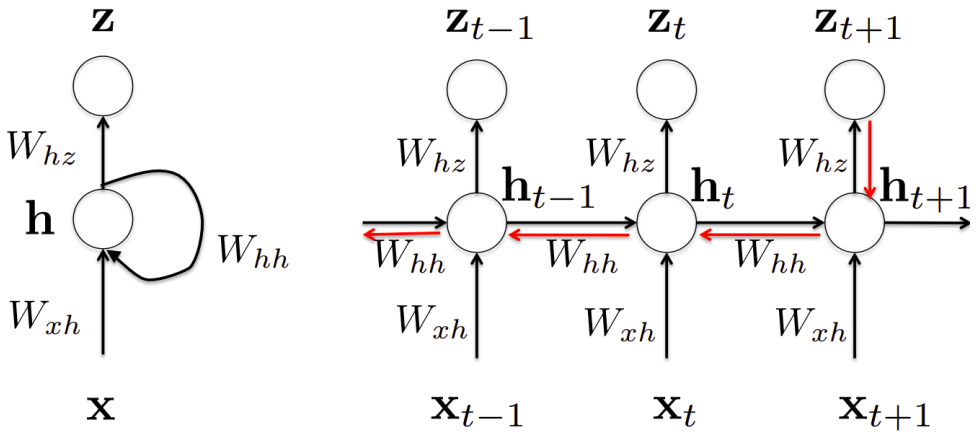

Fig. 2.4: Neuron scheme of RNN[2].

each new input is handled by neuron as if it was a different one, because of the state vector.

To compute the next state vector at time step t, RNN uses equation 2.6, where W_{hh} is weight matrix of state vector, $\mathbf{h}_{t-1}$ is previous state vector, W_{xh} is weight matrix of input, x_t is input and $\mathbf{b}_h$) is bias.

$$\mathbf{h}_t = tanh(W_{hh}\mathbf{h}_{t-1} + W_{xh}\mathbf{x}_t + \mathbf{b}_h) \tag{2.6}$$

Prediction z_t based on current state h_t is evaluated by equation 2.7, where W_{hz} is weight matrix of preditcion and b_z is bias.

$$z_t = softmax(W_{hz}\mathbf{h}_t + b_z) \tag{2.7}$$

2.2.2 Back propagation through time

Proper weight matrices used for input filtering, prediction and next state computation need also with their biases are necessary for achieving reasonable results. Similar approach can be used as in regular NN to find these matrices. It is back-propagation algorithm. Slight difference is that the gradient is searched through time (i.e. through sequence of samples).

Let L be a loss function. Considering W_{hz} is shared through time, gradient with respect to W_{hz} is calculated by sum of differences for each time step shown in equation 2.8 [2].

$$\frac{\partial L}{\partial W_{hz}} = \sum_t \frac{\partial L}{\partial z_t}\frac{\partial z_t}{\partial W_{hz}} \tag{2.8}$$

Gradient with respect to W_{hh} is evaluated by sum of fractional gradients for each time step from 0 to $t+1$, which also depends on several changes in state vector

$\mathbf{h}_t$. Therefore chain rule is applied on differentiation of state vector in time. This propagation back in time is demonstrated with red line in figure 2.4. Mathematical definition of mentioned gradient is shown in equation 2.9 [2].

$$\frac{\partial L}{\partial W_{hh}} = \sum_t \sum_{k=1}^{t+1} \frac{\partial L(t+1)}{\partial z_{t+1}} \frac{\partial z_{t+1}}{\partial \mathbf{h}_{t+1}} \frac{\partial \mathbf{h}_{t+1}}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial W_{hh}} \tag{2.9}$$

The gradients of remaining weight matrices or bias vectors can be deduced from equations: 2.8 and 2.9.

2.3 LSTM - long short term memory

RNNs are the first type of NN, which tried to find patterns in sequences, but they have several imperfections. The first problem is called the vanishing gradient, which refers to a situation, when a sum of partial derivations is nearing to zero and as a consequence RNN does not learn anything. This problem might be a direct opposite and it is called the exploding gradient. Gradient becomes very big and unstable, resulting into situation, when RNN does not learn anything as well.

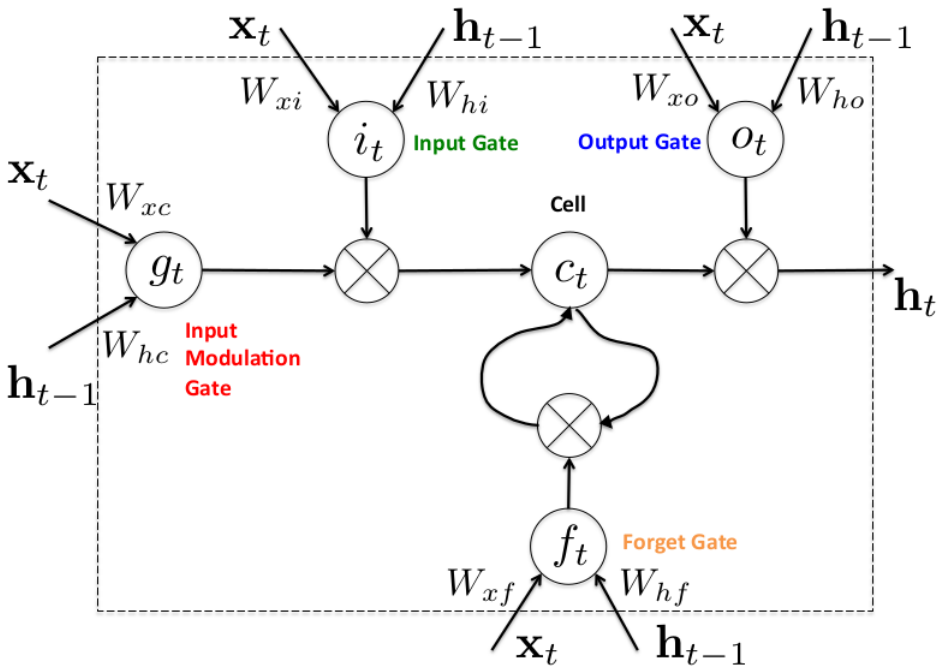

Fig. 2.5: Scheme of LSTM cell with gates [2].

Because of these situations, RNN has problem to find and learn patterns in a long sequences. LSTM cell tries to provide solution of this problem. It inherits the design of RNN and add several important features. The topology of LSTM cell is

shown in figure 2.5, where $\mathbf{x}_t$ is input vector, $\mathbf{h}_{t-1}$ is previous state vector, W are weight matrices and bias vectors are omitted.

The first new feature is an internal memory used for storing necessary information important to patterns of farther distance. Hence, it is called *long short*, because it has long memory defined by $\mathbf{c}_t$ state, and short memory defined by $\mathbf{h}_t$ vector inherited from the RNN.

The second one is gate mechanism providing control of information flow of the cell. These gates are defined in table 2.1.

Tab. 2.1: Description of LSTM gates.

Gate name	Description
Input and Input modulation gate	These gates are used to scale input vector $\mathbf{x}_t$ and previous state $\mathbf{h}_{t-1}$ into specific range of values using `sigmoid` and `tanh` activation function.
Forget gate	It is responsible for controlling what kind of information should be stored in or erased from the memory.
Output gate	It decides which information is inside state vector $\mathbf{h}_t$ and which information leaks from the memory.

Backpropagation algorithm of LSTM cell is very long, because it has much more weight matrices and bias vectors, but principle is exactly the same. Mathematical evaluation is omitted, because it would be out of scope of this thesis.

2.4 Autoencoder

Autoencoder is a type of symmetrical neural network, which uses unsupervised learning method. It tries to learn sparse (less dimensional) approximation of the input vector $\mathbf{x}$ in hidden layer to be able to reconstruct in the output layer [14]. The reduction of dimensions allows the model to not just copy input vector to the output, but extract and find features describing characteristic of this vector and then reconstruct the output vector.

Autoencoder consists of two sub-models. The first one is called `encoder` and its main responsibility is to compress input vector $\mathbf{x}$ to less dimensional vector space called `latent space`. The second sub-model is `decoder`, which tries to reconstruct vector $\hat{\mathbf{x}}$ from `latent space` vector containing compressed original vector $\mathbf{x}$. Example of autoencoder with highlighted sub-models is shown in figure 2.6.

There are many usages of autoencoders, such as anomaly detection (described in section 2.4.1), noise reduction, dimensionality reduction, information retrieval and many others.

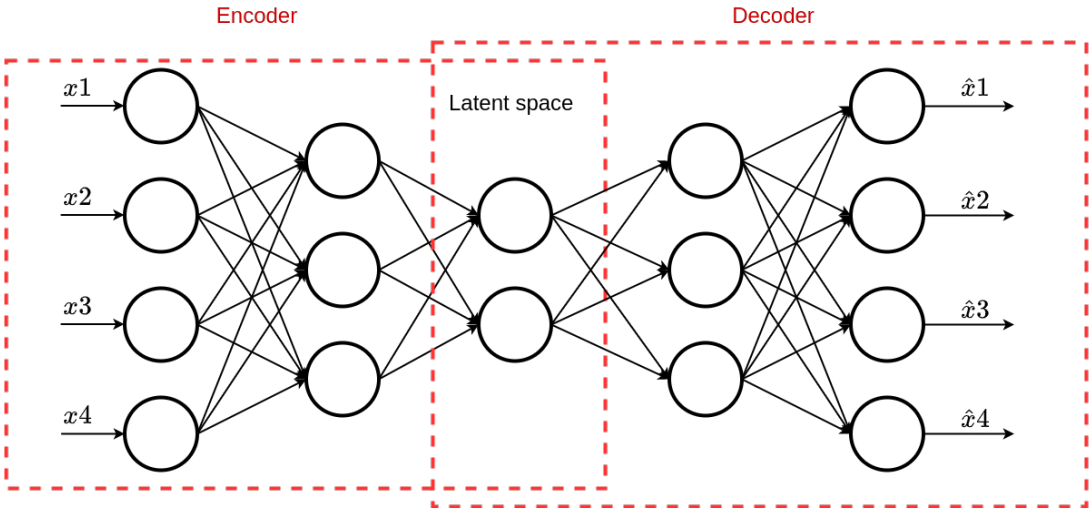

Fig. 2.6: Autoencoder neural network scheme.

2.4.1 Anomaly detection

One autonencoder neural network usage is anomaly detection. An autoencoder is learned using normal data, because anomalies are yet to be found. Considering the fact that autoencoder uses unsupervised learning method, an expected output $\mathbf{y}$ is the same as an input of $\mathbf{x}$ [13]. The goal of learning process is to minimize a value of a loss function. After all learning iteration a anomaly threshold carefully is set to correctly divide normal samples and possible anomalies.

To classify given vector sample $\mathbf{x}$, it is necessary to predict output $\mathbf{y}$ and evaluate prediction error using loss function. If the error is lower than threshold, the sample $\mathbf{x}$ is considered as normal. Otherwise, it is classified as an anomaly.

3 Tensorflow and Keras

This chapter contains description of several machine learning libraries in Python3 language. Focus is mostly concentrated on TensorFlow and Keras, because it is used as a core for ML models designed in the following chapters. Those projects are being developed, well-supported and documented. TensorFlow also supports export and deploy model on remote machine.

3.1 Machine learning libraries in Python3

Machine learning algorithms and techniques are developed and shared by open source communities founded by universities or data analyzing companies. The main idea is to avoid inventing of solution already known. Taking into consideration all difficulties during derivation of loss function gradient, it makes sense to share and reuse the solutions, so data analytic may focus on ML model improvement instead of mathematical derivation.

Libraries used for machine learning do not only provide easy way to create, learn and experiment with model, but they are also capable of efficient computation method. Models based on NNs calculates thousands, even millions of numbers with simple mathematical operators: [+,-,*]. To improve computational speed, libraries support hardware offload, which moves calculation into peripheral device (e.g. graphics processing unit (GPU)).

3.1.1 Scikit-learn

Scikit is a community driven open source project focused on many machine learning algorithms (e.g. random forest, k-Means, nearest neighbors, support vector machines, etc.). It is easy to use, therefore more concentration can be targeted on quality of training data. On the other hand, support GPU offload is missing , which means it does not scale well on large neural networks [12].

3.1.2 Pytorch

Pytorch is machine a learning library in Python3 language developed by Facebook research group in 2017 and is based on Torch library written in C. It is mostly used for natural language processing and computer vision. The main advantage is capability of tensor operation acceleration on GPU, which allows creating and training NN with several hidden layers.

3.2 TensorFlow

TensorFlow library is being developed by artificial intelligence (AI) research community from Google. Model or program development in TensorFlow consists of two stages: static computation graph definition and running a computational session of this graph. Thanks to this model representation, it is easy to evaluate data parallelly in pipelines especially on GPUs allowing creating and learning deep NN.

3.2.1 Computational graph

The core of TensorFlow calculations creates directed graphs structure used for computation, where node can be for example value or math function, and edges are tensors (defined in following section 3.2.2). Each node has zero to N inputs and outputs. TensorFlow assigns kernels to node representing math function, which contain implementation of that function on particular device (e.g. GPU, CPU) [1].

Example of a simple quadratic equation evaluation by computational graph is shown in listing 3.1. There is only definition of static computational graph, the second part with session execution is omitted, because TensorFlow runs it implicitly from version 2.0.0.

Listing 3.1: Simple quadratic equation in TensorFlow.

```
>>> import tensorflow as tf
>>> assert tf.__version__ >= '2.0.0'
>>> a = tf.constant(2)
>>> b = tf.constant(-3)
>>> c = tf.constant(5)
>>> x = tf.constant(list(range(-5,5,1)))
>>> y = a*x**2 + b*x + c
>>> print(y)
tf.Tensor([70 49 32 19 10  5  4  7 14 25], shape=(10,), dtype=int32)
```

3.2.2 Tensor

This library is designed to work with tensors, to be able to design generic mathematical structures or functions using computational dataflow graph. Tensor is generic mathematical structure representing linear mapping from one set to another. From library perspective, it is generic data structure unifying scalar, vector, matrix and n-dimensional array. Thanks to tensors, definition of single dense NN layer can be as easy as $y = sigmoid(W_x * \mathbf{x} + \mathbf{b})$ and it can be used as a generic definition of NN layer, but dimensions (shapes) of certain vectors have to match.

3.2.3 TensorFlow parallelism

TensorFlow is designed to execute graph session on distributed system. Therefore, scheduling algorithm is a necessary part of this library. Each computing system has abstract representation used by scheduling algorithm describing available system resources. Special process called `worker` runs in this system and executes subgraphs, which are assigned from `master` process. To share data between worker processes, remote direct memory access (RDMA) and transmission control protocol are used.

3.2.4 Gradient evaluation

Considering the fact that the TensorFlow is designed especially for machine learning using deep neural networks, it directly supports evaluation of gradient for current graph. When TensorFlow searches gradient for specific tensors with respect to other tensor, it backtracks to the other tensor and add partial gradients to each node on the path. Final gradient is found by applying chain rule on this partial gradients together [1].

3.3 Keras

This library provides an easy to use high-level application programming interface (API) for designing, learning and evaluation NN models. Today it is a part of TensorFlow library and makes model definition as a computational graph (needed by TensorFlow) much easier by abstracting several ML components. Whole ML model is composed of abstract objects compiled into TensorFlow computational graph to gain benefits of GPU acceleration. These abstract objects can represent: layers, model, optimizer, loss functions, activation functions of neurons and many others. The major part of these objects contains methods used for model serialization and deserialization. Several abstract objects are briefly described in sections below.

3.3.1 Model

Project Keras contains two main model definitions. The first is `model.Sequential`, which is also used in our models. It is capable of stacking abstract NN layers (defined in section 3.3.2) and creates fully functional NN model.

The second one is `model.Model` providing API for customized model definition using inheritance or functional description. One of the use cases is definition of two ML models with several layers, but one of these layers is shared across models [3].

Both mentioned model classes share basic API used for model initialization, training and prediction. Description of several model methods used in thesis can be found in the following subsections.

Compile

This method specifies necessary information needed for learning process. The most important parameters are `optimizer` specifying iterative algorithm used for updating weights based on gradient during learning (e.g. Stochastic gradient descent (SGD), Adadelta, Adam, etc.) and `loss` (or objective) function (e.g. mean square error, crossentropy, etc.) defining prediction error of model. Also, various model variables can be evaluated during learning process and can be defined by `metrics` parameter. Method declaration can be seen in listing 3.2.

Listing 3.2: Desclaration of model.compile method [3].

```
compile(optimizer, loss=None, metrics=None, loss_weights=None, sample_weight_mode=
    None, weighted_metrics=None, target_tensors=None)
```

Fit

Each model needs to learn on training data and for this purpose `fit` method exists, accepting enormous number of arguments, but most importantly it accepts learning set `x`, expected results `y`, number of learning iterations `epochs`. This method returns special object `History.history` containing gathered data learning process, which can be used to plot accuracy and loss evolution during epochs. Declaration of `fit` method is shown in listing 3.3.

Listing 3.3: Desclaration of model.fit method [3].

```
fit(x=None, y=None, batch_size=None, epochs=1, verbose=1, callbacks=None,
    validation_split=0.0, validation_data=None, shuffle=True, class_weight=None,
    sample_weight=None, initial_epoch=0, steps_per_epoch=None, validation_steps=
    None, validation_freq=1, max_queue_size=10, workers=1, use_multiprocessing=
    False)
```

Predict

Learned models are usually used for classification or prediction based on current input. This is achieved using `predict` method accepting parameter `x` as input tensor. Declaration of this method can be seen in listing 3.4.

Listing 3.4: Desclaration of model.predict method [3].

```
predict(x, batch_size=None, verbose=0, steps=None, callbacks=None, max_queue_size
    =10, workers=1, use_multiprocessing=False)
```

3.3.2 Layers

To make model definition a bit easier, Keras contains huge number of abstract NN layers definitions in `keras.layers` submodule. All of them share the same basic API, which is extended for specific layers requiring different parameters. The major part of layers requires only single parameter during initialization, which is `units` specifying number of neurons in that layer. It is also common to set specific activation function different to default[1] using `activation` parameter. Dimension of inner layer tensors is derived from number of neurons in adjacent layers. Therefore, layer input shape can be assigned automatically. Exceptions are the first model layers, where automatic evaluation may not be possible and tensor dimension has to be set using `shape` parameter. Layers used in this thesis are defined in table 3.1.

Tab. 3.1: Definition of Keras layers used in this thesis.

Layer name	Description
Dense	This is the most used layer representing fully-connected NN accepting vectors as an input and transform them into output of desired shape.
Dropout	To avoid overfitting during learning and to gain more accured NN, several randomly chosen neurons may be "turned off" using this layer.
LSTM	This layer is abstract implemetation of LSTM cell (described in section 2.3) requiring sequences as an input.
TimeDistributed	To add Dense layer (or similar accepting vectors instead of sequences) before LSTM layer, Dense layer needs to be decorated by TimeDistributed decorator, which allows Dense layer to handle each vector in sequence as single input.

[1] Default activation function of Keras layer is linear.

4 Designing learning model

This chapter contains a design of ML system responsible for traffic verification based on baseline communication, which follows GPON recommendations. Input data structure and format are discussed together with preprocessing of learning data and the whole system component by component. Models in each component are chosen with respect to TensorFlow.

4.1 Data characteristic

Considering the fact that this is a communication protocol, similar approach may be used as for natural language processing. Both contain a sequence of related information in specific order, which means different order may results in a distinct or even misinterpreted context.

4.1.1 Input data format

Model definition is based on TensorFlow library, which well collaborates with `numpy`[1] library. Thus, it is reasonable to use `numpy.array` as data structure, where input vectors are stored. This array has three dimensions defined in table 4.1.

Tab. 4.1: Dimensions of input numpy.array

No.	Dimension name	Description
1	sequence	Model can classify several sequences and by using this dimension is able to separate them.
2	message	Sequence consists of several following messages.
3	feature/field	Each PCBd message has several fields and this dimension divides them.

Single input vector has the same structure as PCBd field of GTC downstream frame and it is shown in figure 1.2. Field representing length are aggregated to a single number, but fields representing data (e.g. PLOAMd data field) are divided into separate features.

[1] `Numpy` is Python3 library supporting various mathematical operations with complex multidimensional structures (e.g. vectors, matrices, etc.) and much more.

4.1.2 Preprocessing data

PCB from captured communication are stored in `Microsoft SQL` database [8]. This data cannot be directly injected into analyzing system. During normalization process of database design, several changes are applied to achieve certain normal form resulting in effective data storing and removal of duplicated data. Unfortunately this format is suitable for ML model, which requires correctly formatted tensors and data duplication in some field is not considered. Therefore, data is grouped[2] together and then extracted in `JavaScript Object Notation (JSON)` format from the database. This extract needs to be loaded, all nested arrays expanded to a flat structure and a result modified into a suitable format described in section 4.1.1.

4.2 System design

GPON verification system consist of several components due to the fact that the single NN should be primarily focused on a narrow purpose to gain optimal results. Otherwise, we would have complex NN model, which is hard to train, test and enhance. This can be sorted out by decomposition into smaller narrowly aimed parts, which positively influences learning and evaluation speed, because each models has to learn dependencies between fewer weights. Scheme of this model system is shown in figure 4.1. Each component of this system is discussed in following subsections.

4.2.1 Data reader

This is the first component in the system and it is mainly responsible for:

- reading data with specific format and load them into memory
- preprocessing RAW data from database and storing in the specific file format

Based on the fact that GPON network can produce thousands of messages per second, the data reader object should use suitable format for huge numbers of small vectors. Data are loaded into n-dimensional arrays and passed to model for learning or classification sequentially, so the performance random data access can be ignored.

4.2.2 Input filter

Input data may contains some mistakes or have different length[3]. This component is responsible for filtering such messages and for normalizing rest messages into suitable format for ML model input. It also helps during learning process by filtering or reducing long sequences of the same messages, which would prevent correct learning.

[2] Applying group by operation on certain tables.

[3] Length of BW Map field is not constant.

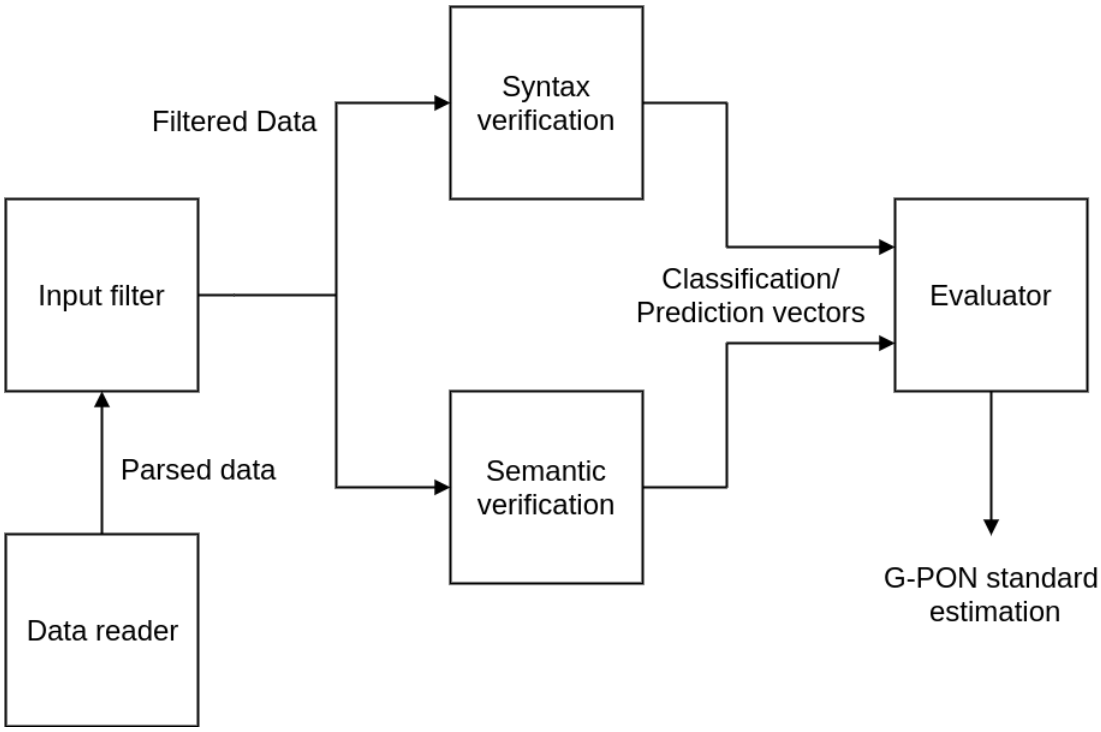

Fig. 4.1: Analyzing models system scheme.

4.2.3 Syntax verification model

Content of each message field has specific rules, which should be considered during analysis. This component is responsible for this verification, which can be achieved by several proposed models.

The first proposed model is a deep neural network consisting of several dense layers. The model would be trained by supervised learning algorithm requiring correct and faulty input data to learn. The advantage of this approach is that the model is able to learn more complex relations. On the other side, it is necessary to generate synthetic faulty data needed by learning process.

The second option is to use one of outlier detection models capable of classification similar and different data compared to learning dataset. This technique uses unsupervised learning and no additional data has to be generated, but it may not learn complex relations compare to deep neural network.

4.2.4 Semantic verification model

This model verifies message relations in time. Proposed model may use `LSTM` cells (described in section 2.3) to achieve this time based sequence check. LSTM based networks prove their capabilities in various similar use cases. For example language translator uses these cells and the whole sentences are processed to a single state and

then this same state is used to generate similar sentence in a different language. The first part of mentioned system solves a similar problem, which is finding patterns in time. This part can be used for message sequence analysis and provided state is not used for translation to different language, but for a prediction whether it satisfies GPON recommendation or not. Keras API in TensorFlow has support of `LSTM` layers, so the proposed model is possible to create. If combination of dense layers and LSTM is necessary, each previous layer has to be `TimeDistributed`, because LSTM layer input shape has more dimensions.

4.2.5 Evaluator

Evaluator is the final component, which creates human-readable output from previous classifications. Its responsibility is to create statistics about classification or prediction of machine learning models used in this system. Based on this information it creates aggregated output, which represents similarity to baseline traffic and highlights strange messages and non-standard message sequences.

5 PLOAMd analysis experiments

This chapter describes ideas and experiments made during design process of learning model for analysis of unknown protocol, which should keep definition GPON recommendation. In the first experiment, the effort is focused on PLOAMd (defined in section 1.5) message analysis. The goal is to create model used for inspection of each message field and search for possible abnormalities.

The second experiment is focused on analysis of the whole protocol, which is a difficult task. Inspiration is taken from natural language processing models based on RNNs (RNN are defined in section 2.2.) or NN derived from RNNs. Analysis of network protocol is very similar, because there is sequential data with given syntax.

The third and the fourth experiment use autoencoders based models (described in section 2.4) for syntax and sequence analysis respectively. These models are very similar. The only difference is the number of layers, number of neurons in each layer and the data shape. The model used for sequence analysis accepts input as flatten message windows of constant length, instead of pure messages.

5.1 PLOAMd data mining

Downstream network traffic in PON was monitored and captured to obtain real data. Learning data is a building stone for a precision of each artificial intelligence model. Capturing system do not store the whole GTC frame due to the size of frames and transmission speed in GPON, but only the PCBd headers. This is not a problem, because important protocol messages used by GPON are in these headers and stored in the database.

For the following experiments, models need information from `PLOAM` field of downstream PCB header. During the preprocessing phase, messages are grouped by `MessageID` column and counted. Sorted PLOAMd messages with number of occurrence are shown in table 5.1.

From filtered data it is obvious that the most used PLOAMd message is $NoMessage_{11}$ used when no managing instruction is being sent by OLT, which is standard operating state when no ONU is executing the activation process.

There are four different values in `ONU-ID` field, which represent three distinct ONU addresses and the broadcast address. Based on the occurrence of various $AssignONU\text{-}ID_3$ and $RangingTime_4$ messages it is clear that activation processes of these units are captured.

Among others there is one message, which is not specified or described in GPON recommendation: *24*. This and similar unknown message are the main reason of protocol analysis with AI/ML.

Tab. 5.1: Filtered PLOAMd messages from captured data traffic.

ONUid	MessageID	Data	CRC	Count
0	4	AAAEuUgAAAAAAA==	196	1
0	8	AvnwAAAAAAAAAA==	12	1
0	10	QAABAAAAAAAAAA==	147	1
0	18	AAE4gAAAAAAAAA==	251	1
1	4	AAAEuUIAAAAAAA==	132	1
1	8	AgAQAAAAAAAAAA==	66	2
1	8	AvngAAAAAAAAAA==	102	1
1	10	QBABAAAAAAAAAA==	75	1
1	18	AAE4gAAAAAAAAA==	166	1
2	10	QCABAAAAAAAAAA==	36	1
2	14	AQAgAAAAAAAAAA==	6	1
255	1	IAAAqqtZgyAAAA==	41	26
255	3	AEhXVEMqiYhpAA==	253	1
255	3	AUhXVENdVdF7AA==	100	1
255	3	AkhXVENdWuF7AA==	239	1
255	11	AAAAAAAAAAAAAA==	158	299945
255	20	HhIAAAAAAAAAAA==	23	12
255	21	IAAAAAAAAAD//w==	212	1
255	24	B+MLFBMb6h0BAA==	66	1

5.2 Syntax analysis experiment

A model in this experiment should detect anomalies in the GPON communication. Message is considered as an anomaly, when it is not similar to messages in the training dataset. The model capable of this function choosed for this experiment is `OneClassSVM` from scikit library, which approximates mathematical function to create an envelope, which decides whether classifying vector is normal or outlier.

Listing 5.1: PLOAMd syntax analysis model used in experiment.

```
from sklearn import svm
model = svm.OneClassSVM(kernel='rbf', nu=0.03, degree=13,
                        gamma=0.00001, verbose=True, max_iter=-1)
model.fit(np.concatenate([x_train, x_train + 0.5, x_train - 0.5]))
```

The model used in this experiment is shown in listings 5.1. Approximation function of model uses radial basis function as a kernel. The `gamma` coefficient of

this model is very low to wrap the envelope around normal vectors more tightly, which raises the sensitivity of this model.

Captured messages shown in table 5.1 are split to learning and testing dataset. Both these datasets represents standard traffic. The model is learned only learning datasets with a variation of $\pm$ 0.5, because `OneClassSVM` expects outliers to be in learning dataset and this guarantees the normal traffic is classified correctly. Testing datasets verifies whether the model classifies unknown standard traffic correctly. Outlier detection capabilities are tested with two datasets. The first dataset is randomly generated within range $< 0; 255 >$ for each field, but the second one is generated within range $< 25; 35 >$ in MessageID and ONUid fields and range $< 0; 255 >$ in remaining files, which simulates possible outliers. Classification results of this model for each dataset is shown in table 5.2.

Tab. 5.2: OneClassSVM outlier detection model classification results.

Dataset	Accuracy	Elements	Errors
train	100%	14	0
test	100%	5	0
similar outliers	100%	50	0
random	100%	50	0

5.3 Sequence analysis experiment

Model with two LSTM layers suitable for sequence analysis is created to learn system relations between message type and content using Keras and TensorFlow library. This model is shown in listings 5.2.

Listing 5.2: Sequence analysis model used for experiment

```
from tensorflow import keras
model = keras.Sequential([
    keras.layers.LSTM(16,input_shape=data[0].shape,
                      return_sequences=True, activation='tanh'),
    keras.layers.LSTM(16, activation='tanh'),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dense(2, activation='softmax'),
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'],
)
```

Experiments with this model are focused on a detection of correct message order. This model uses supervised learning, which means it needs data samples from both classified groups. The dataset with correct samples is created by message sequences from the captured communication and all samples are labeled as `good`. The dataset with corrupted message sequences has to be generated using several procedures. The first one takes the correct dataset and flips the order of messages. The second procedure drops certain message important for the GPON protocol (e.g. message id 4 or 10 used in activation process). The last procedure duplicates certain messages to create non-standard sequences as well. All corrupted message sequences are labeled as *bad* and simulates possible outliers.

These two datasets are source for generating a learning time window of constant length by sampling these subsequences. This process uses sliding window of length 30 and shifts this windows by 1 messages after each sample.

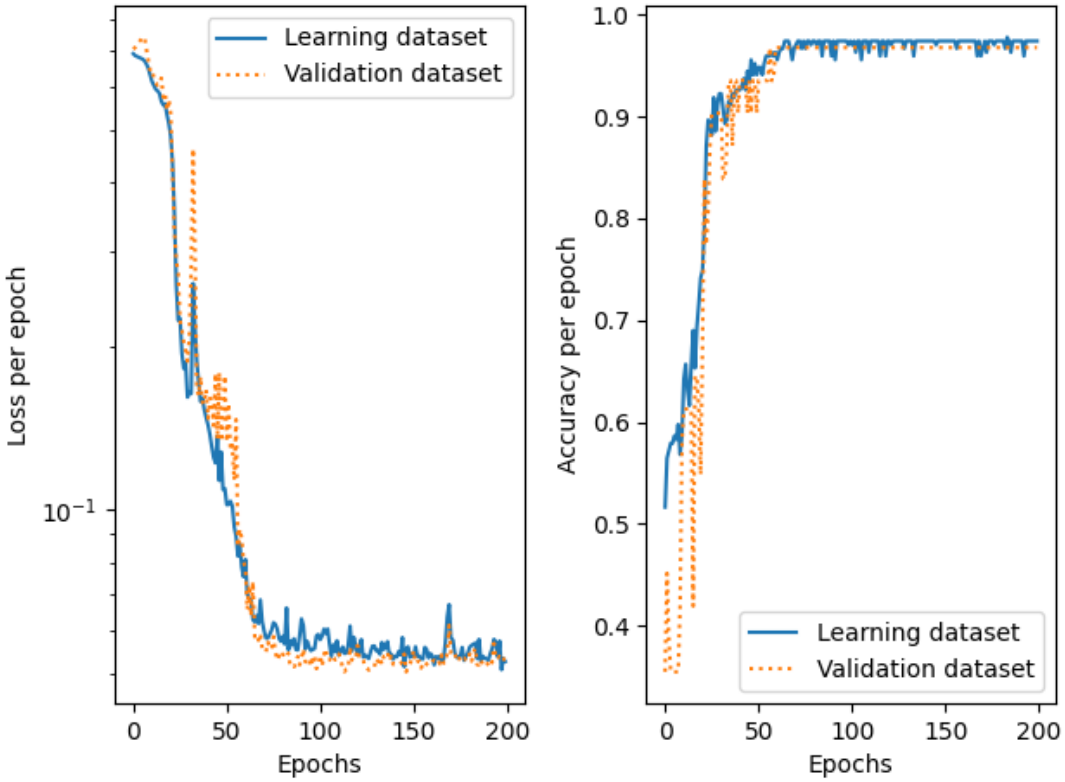

Fig. 5.1: Accuracy and loos during learning process.

Several sequences are popped from learning dataset to create a disjunctive validation set and the learning process can start. Accuracy and loss history of both learning and validation datasets captured during learning is shown in figure 5.1. Considering the size of dataset,it is hard to say whether it learns input data exactly

or finds generalized principles, but it is clear that the model is able to distinguish time sequences.

These results prove that the sequence analysis of GPON is possible with LSTM cells, but learning generalized rules relies on proper dataset. The biggest disadvantage is that the model needs to be learned with corrupted or improper communication samples, which are not available.

5.4 Autoencoder syntax experiments

In this experiment, autoencoder is used for anomaly detection in each field of PLOAMd message. Autoencoder principles are described in section 2.4. Several model configurations are tried with different number of layers and neurons in each layer and also with different activation functions. A model with the best performance is shown in listing 5.3. The model uses exponential linear unit (ELU) function in the input and hidden layers and sigmoid function in the output layer. In this configuration, model is able to compress and reconstruct input vector into the latent space with dimension of two with very low error.

Listing 5.3: Syntax verification autoencoder

```
from tensorflow import keras
model = keras.Sequential([
    keras.layers.Dense(x_train.shape[1], activation='elu'),
    keras.layers.Dense(8, activation='elu'),
    keras.layers.Dense(4, activation='elu'),
    keras.layers.Dense(2, activation='elu'),
    keras.layers.Dense(4, activation='elu'),
    keras.layers.Dense(8, activation='elu'),
    keras.layers.Dense(x_train.shape[1], activation=out_act),
])
model.compile(
    optimizer=keras.optimizers.Adam(learning_rate=0.01),
    loss='mean_squared_error',
)
model.fit(x_train.values, x_train.values, shuffle=True, batch_size=3, epochs=200)
```

This model is learned on captured vectors listed in table 5.1, which define normal traffic. Several vectors are dropped to create a testing dataset. To test anomaly detection capabilities, two additional datasets are made. The first one contains randomly generated vectors of proper dimension with maximum number 255[1]. The second generated dataset simulates possible real outliers and contains vectors similar to standard ones (e.g. similar message ID with low values in the other fields, etc.).

[1]Each PLOAMd field is 1 byte long, so maximal generated number is 255.

The threshold for outlier detection is set by maximal loss function evaluation of learning dataset by learned model.

Tab. 5.3: Syntax detection autoencoder classification results for each dataset.

Dataset	Accuracy	Elements	Errors
training	100%	14	0
testing	100%	5	0
similar outliers	93.33%	150	10
outliers	100%	500	0

Classification results for each dataset by learned model are visible in table 5.3. Mean squared error loss function result histogram for each dataset is shown in figure B.2, where red doted line represents outlier threshold.

5.5 Autoencoder sequence experiments

Sequence analysis autoencoder searches for differences in time windows of specific length. By its design, it focuses on different message sequence or usage. It is also capable to find a missing/additional message or a bad message usage.

In this experiment, the time window has length of 30 messages. This autoencoder is very similar to the syntax analysis model. The only difference is the input vector shape, which is 390 (30 messages by 13 features) for this time window, and number of neurons in each layer, which is [390, 256, 128, 64, 32,64, 128, 256, 390] respectively. All activation functions are exponential linear unit, except of the activation function in the output layer, which is sigmoid function.

Tab. 5.4: Sequence detection autoencoder classification results for each dataset.

Dataset	Accuracy	Elements	Errors
training	100%	25	0
testing	100%	3	0
corrupted	51.54%	97	47
random	100%	97	0

The model is learned by training dataset consisting of normal traffic as syntax autoencoder. This dataset is created from captured frames with omitted message $NoMessage_{11}$, but without dropping duplicates. A sliding window of specific length (30 in this case) travels though frames and generates new data samples, which are

divided into learning and training dataset. Detection capabilities are tested by two datasets. The first one contains data windows generated from captured frames, but some message types (*4* and *20*) are dropped, which clearly create corrupted message sequences. The second outlier dataset contains randomly generated data.

This anomaly detector correctly classifies training, testing and random datasets with 100% accuracy. The results are shown in table 5.4, The dataset containing corrupted frame windows is classified with accuracy below 50%, which means the autoencoder correctly classifies some corrupted windows, but there are many windows classified as normal. The reason for this is that some windows are still very similar to standard traffic and are not changed by dropping messages *4* and *20*. Loss function results histogram is shown in figure B.3, where red dashed lines represent outlier threshold.

5.6 Experiments conclusion

Outlier detection using OneClassSVM is able to correctly classify normal and abnormal GPON frames, but it uses approximate function unable to learn the importance of frame field usage.

LSTM model is powerful system capable to extract useful information from each message and use it in the next frame analysis during the sequence classification process. This requires correctly labeled sequences of normal and invalid traffic to learn, which may be difficult if undocumented message appears within the communication.

Both autoencoder models prove their outlier detection capabilities and make a great alternative to LSTM model, especially because of unsupervised learning. These models can learn usage and syntax of undocumented messages. LSTM model uses supervised learning and for higher classification accuracy it requires samples of normal and abnormal traffic. Abnormal traffic dataset is generated by adding or dropping certain messages, which creates sequences that does not meet the standard. This requires adding additional information into learning process, but in case of unknown messages this information is not available.

All models have relatively good outlier classification accuracy and it is worth considering these models in GPON analyzing system and to evaluate model capabilities with much bigger dataset. Each of these models has pros and cons, which makes them individually special. Suggested solution is to use all models in the GPON analyzing system in parallel and to validate results between them.

6 GPON analyzer implementation

This chapter contains high level implementation details of GPON analyzer and its subsystems. It also describes and indicates usage with illustrative examples. Detailed implementation description can be found in the source code in docstrings of each method and class.

The system architecture is described in section 4.2. Each box in this figure represents an object of GPON analyzing system. The only change in the design is that the system does not use only two ML models (syntax, semantic), but a list of N models to generalize usage of these models. As a result, adding a new analyzing model is much easier.

6.1 Environment

The whole project is implemented in `Python3.6` programming language due to its cross-platform compatibility across operating systems (Linux and Windows). The same idea applies on external libraries, which have to be cross-platform as well.

Consistent executing environment is achieved by `pipenv` program, which downloads and installs specific libraries from *Python Package Index* (PyPI) using `pip` program. These libraries and their versions are specified in special file called `Pipfile`, which is the source of information about packages and versions during environment creation.

6.2 Analyzer

GPON analyzer system is implemented according to principles of object-oriented programming to gain code re-usability and to allow easier program enhancement in the future.

Analyzer is implemented as a standalone and executable python module. It supports two different ways of usage. The first one is running analyzer directly from command line (CLI) by executing python module using `python -m gpon_analyzer` command, which executes `__main__.py` script of this module, or by executing `gpon_analyzer.py` script, which imports and executes the same script. Possible arguments for this execution are described in section 6.8. The second way is to use the analyzer as a library, which enables higher data interaction with models, or gives ability to create and add a new ML model into analyzer. Example of library usage is show in listing B.3 in appendixes.

The responsibility of GPON analyzer object is to hold references to its objects (DataReader, list of Models, Evaluator, etc.) and to distribute certain outer method

calls to correct object, especially `learn`, `classify` and `evaluate`. It also controls data flow between objects. To be able to reuse learned analyzer models, it supports methods for loading and persistently storing models into file. This specific process and file format is described in section 6.7.

6.3 DataReader

DataReader object is responsible for data manipulation. It internally stores the data in a `pandas.Dataframe`[1] and provides API to data, especially to get data shape[2] and values. The data modification is accomplished by applying sequence of filter objects (described in section 6.4), which are applied on the internal `Dataframe`.

DataReader can read two different file formats from file system using `load` class-method, which accepts single argument containing path. If this path ends with *.parquet* suffix, it assumes that the file is in parquet file format and uses pandas library to read this file. Otherwise, it assumes the data are captured directly from GPON network and uses proper data read methods.

If the GPON analyzer perform a data preprocesing, the DataReader object stores result in a parquet format file to reduce data access time in the next execution phase (e.g. learn, classify).

6.4 Filters

Filters are implemented as a component of DataReader object, therefore all filters use `Pandas` library functions for data adjustment/modification and expect the input and output data to be a `pandas.Dataframe`. Filters with initial values used in GPON analyzer are described in subsection 6.4.1. All filters have to match the same API to be able to stack filters in a row and create a filter pipeline.

6.4.1 Applied filters

During the classification, analyzer focuses on PLOAMd messages, but in captured data there is stored the whole PCBd header. Therefore, DataReader object needs to extract relevant information from PCBd header and prepare it for analysis. This process is done using a list of filters described in table 6.1. These filters are applied in this specific order to achieve expected behavior.

[1] Pandas is a python library designated for easy and fast data manipulation.

[2] Data shape represents number of data dimensions and number of elements in each dimension.

Tab. 6.1: Applied data filters in GPON analyzer

Filter class	Values	Description
ExtractSpecific ColumnFilter	*'PLOAM-downstream'*	From dataset drops all columns except *'PLOAMdownstream'*, because analyzer is interested only in PLAOMd messages.
ExpandColumn	*'PLOAM-downstream'*	Assuming that in the *'PLOAMdownstream'* field is nested data structure (field contains dictionary), drops this column from the dataset, creates a new dataframe from this column and concatenates this new dataframe to the original dataset.
RowFilter	*'MessageID', 11*	Deletes all `NoMessage`$_{11}$ type messages from the dataset.
Base64Decoder	*'Data'*	This filter loads values from *'Data'* column, drops this column, decodes values using `base64debode` function and adds decoded values into new data columns.

6.5 ML models

All models are implemented in `ai_models.py` submodule of `gpon_analyzer` module. All of them share the same interface by inheriting `GenericModel` class, which guarantees and enforces expected behavior and usage of implemented models. Each model implements abstract methods and overrides some methods to match special requirements of each ML library. As a result, GPON analyzer handles all models in the same manner without implementation details knowledge. All public methods used by GPON analyzer are described in table 6.2.

In addition to expected model methods as `learn`, `predict` and `classify`, the interface defines another method called `get_report` used for data classification. Compared to classify method, it does not return only the data label, but a triplet[3] consisting of model name, input samples and output classification of each sample.

Generic model also implements special property called `context`, which contains all attributes of object except itself[4] and ML model[5]. It is used to store all attributes into file. This property also defines its setter used for restoring object attributes after load method is called.

[3]The report triplet is defined using `dataclasses` library to gain readability and reduce code.
[4]Python object has attribute called `self`, which is a pointer to this specific object
[5]ML model is stored using library functions, wherein is model defined.

Tab. 6.2: Public interface used by all implemented ML models.

Method name	Description
`create`	This classmethod is responsible for correct instance creation of its class. In case of Keras models, it defines and creates the whole model and passes it to the object constructor as an argument.
`load`	It is used to load ML model from file and create a new instance of particular model class.
`store`	Each ML model has different implementation of storing mechanism and it is implemented in this method.
`learn`	Used for learning/fitting model on a provided dataset.
`classify`	Generic method used for dataset evaluation and classification .
`get_report`	This method classifies specific dataset and returns the evaluation report with model name, input and output.

Implementation of all models is based on experiments from chapter 5, therefore detailed code description is omitted. GPON analyzer contains implementation of these analyzing models:

- **OneClassSVMSyntaxModel** - This class is based on the model described in section 5.2. It is used as an outlier detector inspecting message fields using support vector machine.
- **LSTMSequenceModel** - It is based on experiment described in section 5.3. Model consists of two LSTM layers followed by two dense layers. It uses supervised learning and requires samples of corrupted communication to learn.
- **AutoEncoderSyntaxModel** - This class uses autoencoder model described in section 5.4. It uses unsupervised learning, therefore no additional learning data has to be generated.
- **AutoEncoderSequenceModel** - It is similar to syntax verifying autoencoder, but with different number of layers and neurons. The model is based on the experiment described in section 5.5. Input data is a sequence of 30 messages collapsed into a single dimension.

The inheritance diagram of these models and generic models is shown in figure B.1 in appendixes. This figure shows methods and attributes defined or overridden by each class. There is also generic `KerasModel` class, which defines load and store methods for all TensorFlow models.

6.6 Evaluator

During classification process, GPON analyzer forwards calculated reports from ML models to Evaluator. After classification, Evaluator analyzes the results and evaluates similarity level calculated using equation 6.1.

$$Similarity = \frac{Outliers}{Samples} \tag{6.1}$$

All detected outliers are stored into the file using `numpy.save` for later analysis. In the end, evaluator prints summary table[6] with statistics for each model. Table example of learning dataset classification is shown in listings 6.1. The output destination directory for outlier vectors and summary table is specified using `--output-path` argument.

Listing 6.1: Evaluator output example

```
$ ./analyzer.py classify -m model.zip -d data.parquet
+-----------------------------+--------------------+-----------------+-----------------+
| Model                       |   Similarity [%] |   Outliers [N] |   Samples [N] |
+=============================+====================+=================+=================+
| OneClassSVMSyntaxModel      |          100.0000 |               0 |              55 |
+-----------------------------+--------------------+-----------------+-----------------+
| LSTMSequenceModel           |          100.0000 |               0 |              25 |
+-----------------------------+--------------------+-----------------+-----------------+
| AutoEncoderSyntaxModel      |          100.0000 |               0 |              55 |
+-----------------------------+--------------------+-----------------+-----------------+
| AutoEncoderSequenceModel    |          100.0000 |               0 |              25 |
+-----------------------------+--------------------+-----------------+-----------------+
```

6.7 Storing learned model

GPON analyzer stores learned model data in single `zip` archive consisting of files with weights for each learned ML model and a file with analyzer information. Files with weights are generated directly by ML libraries.

OneClassSVM model from Scikit-learn library used for outlier detection stores and loads learned model via `pickle` library, which dumps Python executable code into binary file.

Keras models use special Hierarchical data format (HDF5) to store weights and ML model. This data format provides flexible and efficient access into stored data especially for large data structures (e.g. multidimensional matrices). This is essential feature for model storing and loading process, because neural networks typically consist of weight matrix [5].

[6]Summary table is generated using `tabulate` python library.

Metadata file is also created during storing process. It contains necessary information needed to restore ML model object state. For each model it stores: a class of this object, a path to stored ML model weights and a context, which holds local variables of this object. All this data is stored in simple JSON file inside zip package.

6.8 Command line interface and usage

GPON analyzer can be used as a standard program run from command line (CLI) as well. Therefore, it has proper CLI API to influence program behavior implemented using `argparse` library, which is responsible for initializing arguments with default values, parsing values from CLI and syntax verification of arguments inputs. Accepted CLI arguments are defined in table 6.3. All of specified arguments are optional, except the `action` argument. Other arguments have defined default value, which varies from action to action, so default values are not specified in the argument parser, but in specific objects.

6.8.1 Actions

Possible actions with GPON analyzer using CLI API are described in following sections. All actions have the same optional arguments described in table 6.3. There are longer and shorter argument versions, but in following examples only long options are used for easier usage understanding.

Preprocess

It is used to load captured data from `*.txt` files, apply data filters if `--filter` argument is provided, and store parsed and filtered data in `parquet` format file into destination provided by `--output-path` argument. A usage example is show in listings 6.2.

Listing 6.2: Example of preprocess action.

```
$ ./gpon_analyzer.py --data-path data --output-path data5.parquet preprocess
```

Learn

This action creates a new GPON analyzer and ML models. These models are learned on training dataset provided by `--data-path`. Learned analyzer is stored as a zip archive in destination provided by `--model-path` argument.

Tab. 6.3: GPON analyzer command line arguments.

Argument	Description
-h	Shows program description, help message and argument descriptions.
-m/--model-path `PATH`	This argument specifies a path to persistent data of learned model. If a new model is created/learned, it is stored into this location. If path contains existing model, data is rewritten.
-d/--data-path `PATH`	Specifies a path to the source of captured and parsed GPON data (accepting a single file or directory of files with `.txt` suffix) or path to pre-processed data in `.parquet` file.
-o/--output-path `PATH`	During execution, GPON analyzer may produce some outputs and this arguments specifies their destination. For example, pre-processing action parses and loads data from directory provided by`-data-path` argument, applies filters and stores filtered data in `.parquet` file into this path.
-f, --filter	This argument adds default data filters into **DataReader** object. Default filters are described in section 6.4.1.
-l, --log `LEVEL`	Defines GPON analyzer and other components logging level during execution.
-a, --auto-encoder	If set, auto-encoder models are appended to other analyzing models in GPON system.
`ACTION`	This is the only positional argument in CLI API. It specifies the type of action executed with current data or/and model. Possible values are: `preprocess,learn,classify,print`. All actions are described in section 6.8.1.

Classify

This action is used for traffic analysis by learned model. It loads analyzer with models from archive defined by `--model-path` and runs inspection (classification) on input data defined by `--data-path`. It is recommended to use `--filter` argument, if this argument was provided on learning dataset during learning or preprocessing. In the final phase, it invokes the evaluator to analyze the classification reports and stores detected outliers and a summary table into directory specified by `--output-path`. Usage example is shown in listings 6.1.

Listing 6.3: Example of learn action

```
$ ./gpon_analyzer.py --data-path data.parquet --model-path model.zip learn
```

Print

Loads input data from destination defined by `--data-path` and prints it into CLI, which suggests data shape and form. It is also possible to apply data filter by `--filter` argument. An example of usage and output is shown in listings 6.4. An example output if this action is show in listings B.1 in appendixes.

Listing 6.4: Example of print action

```
$ ./gpon_analyzer.py --data-path data.parquet print
```

7 GPON analyzer detection test

In this chapter, GPON analyzer is tested on knowingly corrupted datasets to verify, whether the analyzer fulfills expectations in the detection of improper communication. Datasets generation procedures are described in section 7.1. These corrupted datasets are examined by learned GPON analyzer models. GPON test results are described in section 7.2.

7.1 Data preparation

Three additional datasets are generated to test and verify GPON analyzer required features, which is anomaly and protocol differences detection. All of these datasets contain corrupted messages or sequences. Corruptions are applied on a dataset created from captured traffic by applying preprocessing filters discussed in section 6.1. Error per dataset summary is shown in table 7.1 and error description with generating procedure is in following subsections.

Tab. 7.1: Error in testing datasets

Error / Dataset	Change random field values	Drop important messages	Add similar messages
Syntax dataset	✔	✘	✘
Sequence dataset	✘	✔	✔
All errors dataset	✔	✔	✔

✔ - applied ✘ - not applied

7.1.1 Change random field value error

Dataset generating procedure runs in three cycles and in each cycle modifies random 10% frames. They vary in number of corrupted fields, which is two, four and six respectively. These fields are modified according equation 7.1.

$$Value_{new} = (Value_{old} + 128) \mod 255 \tag{7.1}$$

This procedure modifies fifteen messages in total. In other words, this dataset contains fifteen outliers, which should be detected by GPON analyzer.

7.1.2 Drop important messages

This dataset is generated by dropping important activation process messages 4 and 10, and critical informational frames used in communication 1 and 20. These messages are described in section 1.5.1. Each message is dropped from separate instance of source dataset and results are concatenated together creating the final dataset. This dataset contains many abnormal sequences, which do not correspond with GPON recommendation.

7.1.3 Add similar messages

Concept of similar messages is firstly mentioned in experiments in section 5.2. These messages are outliers, but differences in frame fields are very low compared to normal traffic, which makes them difficult to find.

This dataset consists of four concatenated source datasets in a row and thirty similar messages are inserted to random positions, which breaks the communications rules of GPON recommendation.

7.2 Results evaluation

GPON analyzer classifies generated artificial communication, which contain several various errors and mistakes. Classification results are shown and discussed in the following sections.

7.2.1 Learning dataset

The first classified is the learning dataset to verify the correct classification of normal traffic. The analyzer marks all results as normal. Considering the fact that this is the learning set, other classification result than normal would mean error in the learning process. The results are show in listings 7.1.

7.2.2 Syntax dataset

The second dataset contains only syntax errors and focuses on syntax outlier detectors. Both OneClassSVM and autoencoder models find almost all abnormal messages in the communication. Sequence autoencoder model also finds several outliers, because it is learned on time windows of specific messages. Messages of this dataset are corrupted, which means time windows of this dataset are corrupted as well. LSTM model finds many outliers, because sequences contain unknown message and

Listing 7.1: Classification of learning dataset

Model	Similarity [%]	Outliers [N]	Samples [N]
OneClassSVMSyntaxModel	100.0000	0	55
LSTMSequenceModel	84.0000	4	25
AutoEncoderSyntaxModel	100.0000	0	55
AutoEncoderSequenceModel	100.0000	0	25

the model does not know the usage. Therefore, the model classifies those sequences as outliers. The results are show in listings 7.2.

Listing 7.2: Classification of dataset with syntax errors

Model	Similarity [%]	Outliers [N]	Samples [N]
OneClassSVMSyntaxModel	80.0000	11	55
LSTMSequenceModel	32.0000	17	25
AutoEncoderSyntaxModel	80.0000	11	55
AutoEncoderSequenceModel	84.0000	4	25

7.2.3 Sequence dataset

The third dataset contains errors in sequences, but message fields remain untouched. Syntax models do not find any abnormal messages, because this dataset is syntactically correct. Both sequence analyzing models find many outliers, which is expected behavior when abnormal message sequences occur in the communication. The results are shown in listings 7.3.

7.2.4 All errors dataset

The fourth dataset contains both syntax and sequence errors and creates communication, which is almost completely out of GPON recommendation. Syntax models find similar percentage of abnormal messages as in the first dataset, which is correct, because the ratio between syntactically corrupted messages and the normal ones is the same. Sequence models classify almost the whole communication as abnormal,

Listing 7.3: Classification of dataset with errors in message sequences

Model	Similarity [%]	Outliers [N]	Samples [N]
OneClassSVMSyntaxModel	100.0000	0	397
LSTMSequenceModel	34.3324	21	367
AutoEncoderSyntaxModel	100.0000	0	397
AutoEncoderSequenceModel	39.7820	221	367

which makes sense, considering the number of errors in the communication. The results are shown in listings 7.4.

Listing 7.4: Datasets classification with all errors

Model	Similarity [%]	Outliers [N]	Samples [N]
OneClassSVMSyntaxModel	79.5970	81	397
LSTMSequenceModel	13.0790	319	367
AutoEncoderSyntaxModel	83.1234	67	397
AutoEncoderSequenceModel	17.7112	302	367

Conclusion

The main goal of this thesis was to create a GPON analyzing system consisting of machine learning models defined using TensorFlow library. These models should have been able to analyze the syntax and the semantic of GPON protocol. The term syntax referred to verification of each field content in GPON header, whether it was similar to patterns from baseline traffic or not. The second term semantic referred to the analysis of patterns found in message sequences, which verified whether the analyzed traffic used the same messages in the same order and with similar content as the baseline traffic.

Four experimental models were created and learned on the baseline data to prove the possibility of analysis using machine learning techniques. The syntax analyzing model was based on one class support vector machine from scikit library, which approximated n-dimensional mathematical function to baseline traffic messages. This model had a high accuracy in detecting abnormalities, even when messages were very similar to the baseline data.

The model used for the semantic analysis was a neural network based on layers with LSTM cells capable of recognition patterns in long sequences. This model was able to learn recognizing correct and corrupted PLOAMd message sequences with relatively high accuracy, which is shown in figure 5.1. The disadvantage of model based on the LSTM cells was that it used supervised learning and required samples of corrupted/abnormal traffic.

Experiments also contained autoencoder neural network models for syntax and semantic analysis, which compressed input vector into the latent space and tried to reconstruct original vector from the compressed form. Similarity was evaluated by using mean squared error function. The key estimated value was outlier threshold, which separated the data into normal and abnormal traffic. In these experiments, the threshold was set to the maximal mean squared error of learning dataset in the last learning epoch. The autoencoder for syntax analysis accepted messages as an input data, but the sequence analysis autoencoder accepted message windows of constant length. Both models showed well the outlier detection capabilities. Their biggest advantage was that they used unsupervised learning and learned only from captured traffic, so they did not require any dataset with corrupted message sequences. Both autoencoder models and model with LSTM cells were implemented using `Keras` API from `TensorFlow` library, which allowed to offload demanding calculations into hardware accelerator (e.g. graphics processing unit).

The GPON analyzer implementation was based on the design described in section 4. Each component was represented as an object and the system only routed the dataflow between these objects. The system did not distinguish between syntax

and semantic models, but worked with a list of models matching the same interface, which allowed adding more models with various parameters. All four experimental models from chapter 5 were implemented in the GPON analyzer using `scikit` and `TensorFlow` libraries. The GPON analyzer implemented two different ways of usage. The first one was executing the program directly from the command line and controlled it by using arguments. The second way was to use GPON analyzer as a library, which allowed interaction with objects at higher level. Learned system was stored into a zip package containing attributes of all objects in a `JSON` file. This package also contained stored ML models in a library dependent file formats.

The final implementation was tested using synthetically corrupted dataset by various procedures. Classification of datasets with abnormal messages or message sequences by learned GPON analyzer confirmed its anomaly detection capabilities. Both syntax and semantic (sequence) errors were found by designated models with high accuracy. The similarity value was evaluated as a ratio of normal traffic samples to all samples, where the sample was a message in case of syntax analyzing models and a time window (sequence of messages) in case of semantic analyzing models.

This thesis accomplished the assignment in all points. The theoretical part contained description of several abnormal traffic detection techniques based on supervised and unsupervised learning using neural networks defined in TensorFlow library. The practical part contained the implementation of GPON analyzing system capable of similarity estimation compared to the baseline traffic. This system used several machine learning models to identify various potential abnormalities in the communication.

Bibliography

[1] Abadi, M.; Agarwal, A.; Barham, P.; aj.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015, software available from tensorflow.org.
URL http://tensorflow.org/

[2] Chen, G.: A Gentle Tutorial of Recurrent Neural Network with Error Backpropagation. *CoRR*, ročník abs/1610.02583, 2016, 1610.02583.
URL http://arxiv.org/abs/1610.02583

[3] Chollet, F.; aj.: Keras. https://keras.io, 2015.

[4] doc. Ing. Václav Jirsík, CSc.: Umělá inteligence - Definice. [online], 09 2019.
URL https://www.vutbr.cz/www_base/priloha.php?dpid=180947

[5] Folk, M.; Heber, G.; Koziol, Q.; aj.: An Overview of the HDF5 Technology Suite and Its Applications. In *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, AD '11, New York, NY, USA: Association for Computing Machinery, 2011, ISBN 9781450306140, str. 36–47, doi:10.1145/1966895.1966900.
URL https://doi.org/10.1145/1966895.1966900

[6] Gershenson, C.: Artificial Neural Networks for Beginners. *CoRR*, ročník cs.NE/0308031, 2003.
URL http://arxiv.org/abs/cs.NE/0308031

[7] Horvath, T.; Munster, P.; Oujezsky, V.; aj.: Activation Process of ONU in EPON/GPON Networks. 07 2018, s. 1–5, doi:10.1109/TSP.2018.8441216.

[8] Ing. Martin Holík: *GPON frame detection system.* Diplomová práce, Brno University of Technology, Czech Republic, 2018.

[9] ITU-T: Gigabit-capable passive optical networks (G-PON): Transmission convergence layer specification. [online], 01 2014.
URL https://www.itu.int/rec/T-REC-G.984.3-201401-I/en

[10] Onn Haran, F.; Sheffer, A.: The Importance of Dynamic Bandwidth Allocation in GPON Networks. [online], 01 2008.
URL https://pmcs.com/cgi-bin/document.pl?docnum=2072146

[11] Ozimkiewicz, J.; Ruepp, S.; Dittmann, L.; aj.: Dynamic bandwidth allocation in GPON networks. In *Recent advances in electrical engineering.*, 01 2010, s. 182–187.

[12] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; aj.: Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, ročník 12, 2011: s. 2825–2830.

[13] Sakurada, M.; Yairi, T.: Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, MLSDA'14, New York, NY, USA: Association for Computing Machinery, 2014, ISBN 9781450331593, str. 4–11, doi:10.1145/2689746.2689747.
URL `https://doi.org/10.1145/2689746.2689747`

[14] Sun, W.; Shao, S.; Zhao, R.; aj.: A sparse auto-encoder-based deep neural network approach for induction motor faults classification. *Measurement*, ročník 89, 2016: s. 171 – 178, ISSN 0263-2241, doi: https://doi.org/10.1016/j.measurement.2016.04.007.
URL `http://www.sciencedirect.com/science/article/pii/S0263224116300641`

[15] Wythoff, B. J.: Backpropagation neural networks: A tutorial. *Chemometrics and Intelligent Laboratory Systems*, ročník 18, č. 2, 1993: s. 115 – 155, ISSN 0169-7439, doi:https://doi.org/10.1016/0169-7439(93)80052-J.
URL `http://www.sciencedirect.com/science/article/pii/0169743993800052J`

List of symbols, physical constants and abbreviations

AI	Artificial intelligence
API	Application programming interface
CLI	Command line interface
ELU	Exponential linear unit
GEM	Gigabit-capable passive optical network encapsulation method
GPON	Gigabit-capable passive optical network
GPU	Graphics processing unit
GTC	Gigabit-capable passive optical network Transmission Convergence
HDF	hierarchical data format
ITU	International Telecommunication Union
JSON	JavaScript Object Notation
LSTM	Long short term memory
ML	Machine learning
NN	Neural network
ONU	Optical network unit
ODN	Optical distribution network
OLT	Optical line termination
OAM	Operations, administration and management
PCBd	Physical control block downstream
PIP	Package installer for Python
PLOAM	Physical line operations, administration and management
PLOAMd	Physical line operations, administration and management downstream
PON	Passive optical networks
PyPI	Python package index
RNN	Recurrent neural network
SVM	Support vector machine
T-CONT	Transmission container

List of appendices

A CD content

Included CD contains source codes of this thesis and source codes of the GPON analyzer. A structure of important files in the CD is shown below. Several files are omitted to reduce size and complexity of the shown tree.

- `/` ..... CD root directory
  - `latex/` ..... Latex source code for of this thesis
  - `python/` ..... Python source code used in this theses
    - `data/` ..... Raw captured data
    - `gpon_analyzer/` ..... GPON analyzer python module
      - `ai_models.py`
      - `analyzer.py`
      - `data.py`
      - `evaluator.py`
      - `filter.py`
      - `__init__.py`
      - `__main__.py`
    - `gpon_analyzer.py` ..... Executable of GPON analyzer
    - `model.zip` ..... Learned ML models
    - `Pipfile` ..... Defines all used packages
    - `preproccessed_data/` ..... Direcotry with learning and corrupted data
      - `data.parquet`
      - `syntax.parquet`
      - `seuence.parquet`
      - `all_err.parqute`
    - `scratches/` ..... Various python scripts (e.g. model experiments)
  - `dp.pdf` ..... This thesis in electronical format pdf

B Extra data and figures

B.1 ML models of GPON analyzer

ML models inheritance diagram of GPON analyzer is shown in figure B.1.

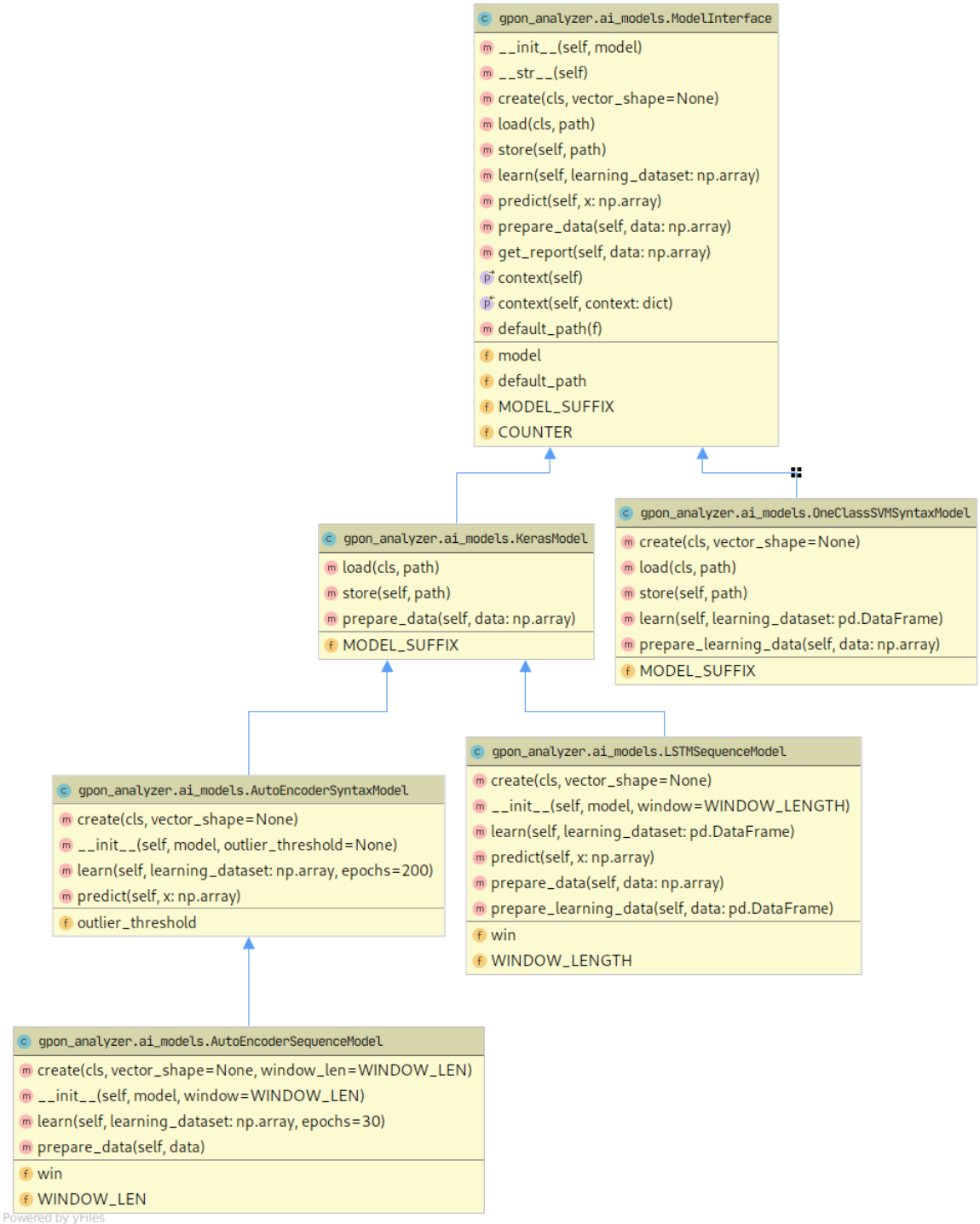

Fig. B.1: Inheritance diagram of ML models in GPON analyzer.

B.2 All captured PLOAMd messages

All messages extracted from captured GPON communication are shown in two listings B.1 and B.2 due to number of messages. GPON analyzer `print` function is used to generate this output.

Listing B.1: All extracted PLOAMd messages (part1).

```
$ ./gpon_analyzer.py -d preprocessed_data/data.parquet print
Package contains 55 vectors.
    ONUid  MessageID  CRC  D0   D1   D2   D3   D4  D5   D6   D7   D8  D9
0     255          1   41  32    0    0  170  171  89  131   32    0   0
1     255         20   23  30   18    0    0    0   0    0    0    0   0
2     255          1   41  32    0    0  170  171  89  131   32    0   0
3     255         20   23  30   18    0    0    0   0    0    0    0   0
4     255         20   23  30   18    0    0    0   0    0    0    0   0
5     255          1   41  32    0    0  170  171  89  131   32    0   0
6     255          1   41  32    0    0  170  171  89  131   32    0   0
7     255         20   23  30   18    0    0    0   0    0    0    0   0
8     255          1   41  32    0    0  170  171  89  131   32    0   0
9     255          1   41  32    0    0  170  171  89  131   32    0   0
10    255         24   66   7  227   11   20   19  27  234   29    1   0
11    255         20   23  30   18    0    0    0   0    0    0    0   0
12    255          1   41  32    0    0  170  171  89  131   32    0   0
13    255          1   41  32    0    0  170  171  89  131   32    0   0
14    255          1   41  32    0    0  170  171  89  131   32    0   0
15    255          1   41  32    0    0  170  171  89  131   32    0   0
16    255          1   41  32    0    0  170  171  89  131   32    0   0
17    255         20   23  30   18    0    0    0   0    0    0    0   0
18    255          1   41  32    0    0  170  171  89  131   32    0   0
19    255          1   41  32    0    0  170  171  89  131   32    0   0
20    255          1   41  32    0    0  170  171  89  131   32    0   0
21    255         20   23  30   18    0    0    0   0    0    0    0   0
22    255         20   23  30   18    0    0    0   0    0    0    0   0
23    255          1   41  32    0    0  170  171  89  131   32    0   0
24    255          1   41  32    0    0  170  171  89  131   32    0   0
25    255          3  239   2   72   87   84   67  93   90  225  123   0
26    255          3  100   1   72   87   84   67  93   85  209  123   0
27    255          3  253   0   72   87   84   67  42  137  136  105   0
28      0          4  196   0    0    4  185   72   0    0    0    0   0
29      1          4  132   0    0    4  185   66   0    0    0    0   0
30      0         18  251   0    1   56  128    0   0    0    0    0   0
31      0         10  147  64    0    1    0    0   0    0    0    0   0
```

Listing B.2: All extracted PLOAMd messages (part2).

32	1	8	66	2	0	16	0	0	0	0	0	0	0
33	1	8	66	2	0	16	0	0	0	0	0	0	0
34	1	18	166	0	1	56	128	0	0	0	0	0	0
35	0	8	12	2	249	240	0	0	0	0	0	0	0
36	1	10	75	64	16	1	0	0	0	0	0	0	0
37	1	8	102	2	249	224	0	0	0	0	0	0	0
38	2	14	6	1	0	32	0	0	0	0	0	0	0
39	2	10	36	64	32	1	0	0	0	0	0	0	0
40	255	21	212	32	0	0	0	0	0	0	0	255	255
41	255	1	41	32	0	0	170	171	89	131	32	0	0
42	255	1	41	32	0	0	170	171	89	131	32	0	0
43	255	20	23	30	18	0	0	0	0	0	0	0	0
44	255	1	41	32	0	0	170	171	89	131	32	0	0
45	255	20	23	30	18	0	0	0	0	0	0	0	0
46	255	1	41	32	0	0	170	171	89	131	32	0	0
47	255	1	41	32	0	0	170	171	89	131	32	0	0
48	255	1	41	32	0	0	170	171	89	131	32	0	0
49	255	1	41	32	0	0	170	171	89	131	32	0	0
50	255	20	23	30	18	0	0	0	0	0	0	0	0
51	255	1	41	32	0	0	170	171	89	131	32	0	0
52	255	20	23	30	18	0	0	0	0	0	0	0	0
53	255	1	41	32	0	0	170	171	89	131	32	0	0
54	255	1	41	32	0	0	170	171	89	131	32	0	0

B.3 Library usage example

GPON analyzer is written as a python module and all components can be enhanced. Example of basic read and learn procedures are shown in listing B.3.

Listing B.3: GPON analyzer library usage

```
from gpon_analyzer.analyzer import GPONAnalyzer
from gpon_analyzer.ai_models import AutoEncoderSequenceModel
from gpon_analyzer.data import Reader

data_reader = Reader.read_parquet('preprocessed_data/data.parquet')
analyzer = GPONAnalyzer(
    data_reader,
    [AutoEncoderSequenceModel.create(data_reader.shape)]
)
analyzer.learn()
analyzer.store()
```

B.4 Syntax autoencoder histogram

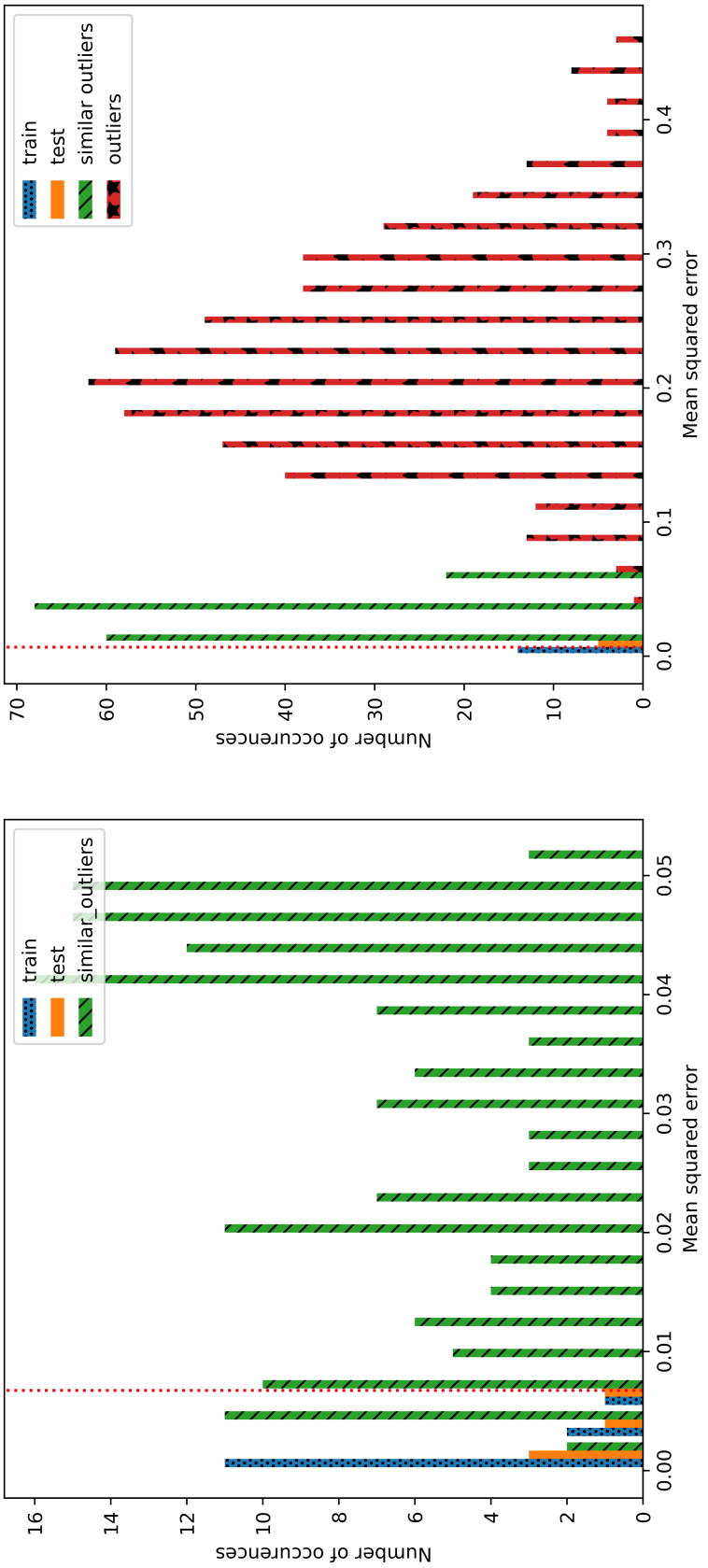

Fig. B.2: Loss values histogram of various datasets evaluated by autoencoder for syntax analysis.

B.5 Sequence autoencoder histogram

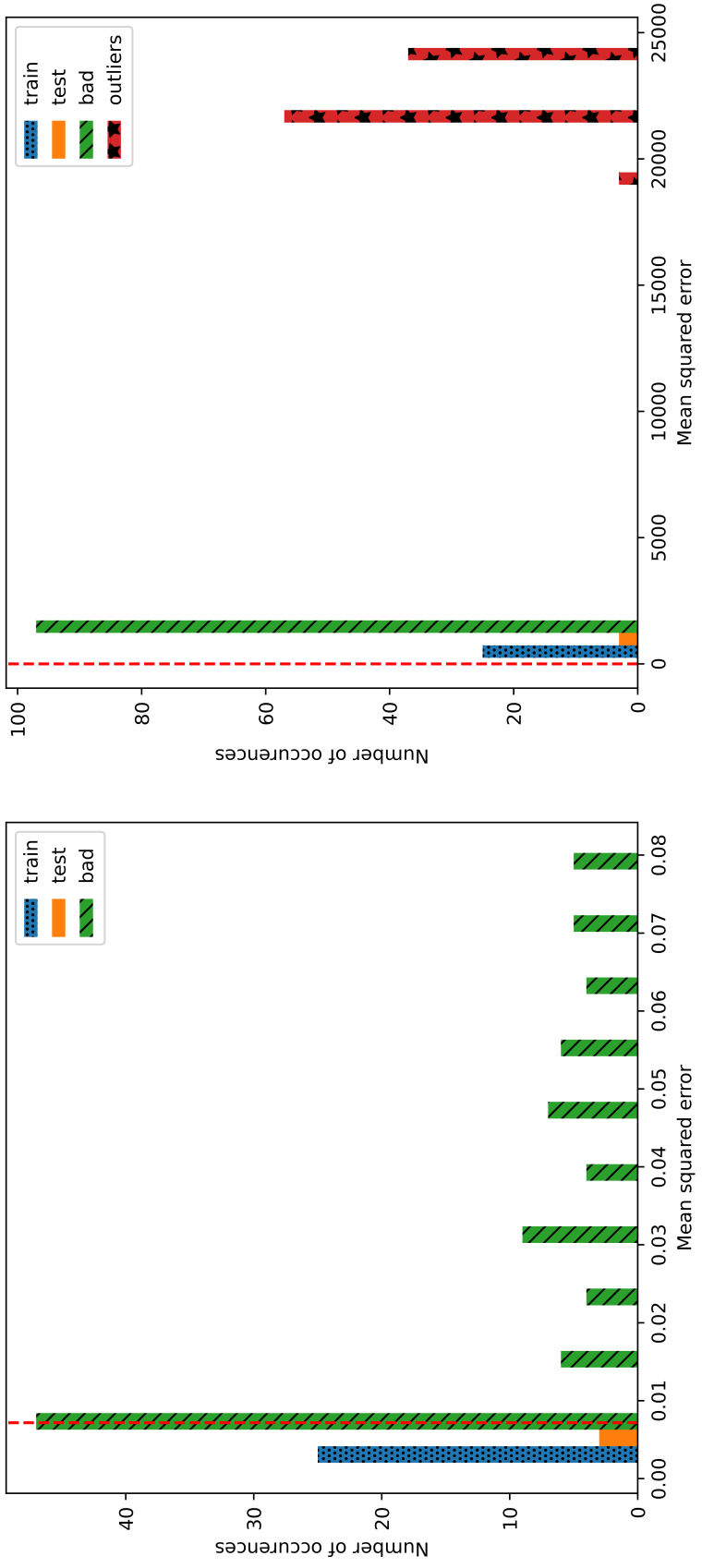

Fig. B.3: Loss values histogram of various datasets evaluated by autoencoder for sequence analysis.