Cognitive management of SLA in software-based networks

Jaafar Bendriss

THÈSE DE DOCTORAT CONJOINT TELECOM SUDPARIS et DE l'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité

Informatique et réseaux

Présentée par

Jaafar BENDRISS

Pour obtenir le grade de

DOCTEUR de TELECOM SUDPARIS

Sujet de la thèse :

Gestion Cognitive de SLA dans un contexte NFV

soutenue le 14 juin 2018

devant le jury composé de :

Prof. Djamal Zeghlache	Directeur de thèse	Telecom SudParis
Dr. Imen Grida Ben Yahia	Encadrante de thèse	Orange Labs
Prof. Panagiotis Demestichas	Rapporteur	University of Piraeus
Prof. Filip de Turck	Rapporteur	Ghent University
Prof. Marcelo Dias de Amorim	Examinateur	LIP6
Mdme Marie-Paul Odini	Examinatrice	HP

Thèse N : 2018TELE0003

CONTENTS

1 Introduction **1**

I Introduction 1

II Problem Statement 3

A Research Questions 4

B Contributions 5

III Thesis Structure 6

2 Machine Learning : Basics, Challenges, and Network Applications **9**

I Introduction to Machine Learning (ML) 10

A Definition 10

B Machine Learning Types 13

C A Brief History of Machine Learning 14

II Supervised Machine Learning 16

A How machines learn. 24

B Evaluation 33

III Unsupervised Machine Learning 41

A Clustering 41

B Anomaly Detection 42

C Dimetionality Reduction 42

IV Deep Learning 44

A How Deep Learning is different? 45

B Exploding/Vanishing gradients problem 45

C Regularization 47

D RNN 47

V Machine Learning Latest Challenges 49

A Machine vs Human Learning 49

B Scalable Machine Learning 50

VI Machine Learning for Network Management 53
A Machine Learning for Network Management a Brief History 55
B FCAPS Management 57
C Cognitive Network Management Initiatives 60
VII Conclusion 66

3 SLA management **67**
I Context : Software Networks 68
A Network Function Virtualization 68
B Software-Defined Networking 70
II Early SLA management 73
III SLA in the Cloud 74
IV SLA in Software Networks 77
V Literature Gaps and Future Research Directions 80

4 Proposal : Cognitive SLA Management Framework **83**
I Introduction 84
II Problem Statement 87
A Service Level Agreement 89
B SLA and SDN/NFV 90
C Formal Description 92
D SLA Example 93
III Cognitive SLA Architecture 95
A Cognet Smart Engine : 98
B Policy Engine 101
C NFV Architectural Framework 102
D Proposed workflow 103
E Policy Engine 107
F Cognet Sequence Diagram 108
G Operational Application & Use Cases 109
IV Data Analysis 124
A Data Gathering 124
B Data preparation 126
C Dimensionality reduction 129
D Visualization 138
E SLA Assurance Services 141

V Conclusion 143

5 Proposal : Cognitive Smart Engine 145
I Introduction 146
II CSE Algorithms 149
A Anticipation and forecasting 150
B Classification 157
III Model selection 171
A Problem Formulation and Choices 171
B Search Methods 173
C The hyperparameter Search Space 177
D Research Methodology 182
E Data 183
F Meta-Learning 188
G feature relevance 193
IV Conclusion 195

A Thesis Publications 197
I List of Contributions 197
A Accepted papers 197
B Public Cognet Deliverables 198
C Exhibition 199

B Installation setup 201

LIST OF FIGURES

1.1 The cost of hardware and software and their management [1]. 4
1.2 Research question mindmap. 6
1.3 Thesis structure 8

2.1 Machine Learning approach 11
2.2 Machine Learning vs programming 12
2.3 Machine Learning can provide deeper understanding of old problems 12
2.4 Machine Learning types and different tasks type 13
2.5 Randomy generated data. the x-axis represent the input and y-axis represents the target values. 18
2.6 Machine Learning types and different tasks type 18
2.7 a Biological neuron [2] 19
2.8 Artificial perceptron and its decision boundary 19
2.9 Non-linear decision boundary XOR 21
2.10 MLP solving XOR problem 21
2.11 MLP solving XOR problem 21
2.12 simple MLP example 22
2.13 Gradient Descent 25
2.14 Gradient Descent with a learning rate too small and too large 27
2.15 Optimizer comparison 28
2.16 Activation functions and their respective derivatives . . . 31
2.17 bias vs variance 34
2.18 Classification metrics. Precision and recall 36
2.19 Confusion matrix 37
2.20 bias vs variance 40

2.21 Machine Learning Diagnostic process 40
2.22 Clustering example 42
2.23 Shemetic structure of an autoencoder with one fully connected hidden layer. 43
2.24 Saturation 46
2.25 RELU activation function 46
2.26 LSTM Cell : The element wise multiplication is key for LSTM cell, helping the preservation of constant error when backpropagating the error. The forget gate is an identity function, when it is open the input is multiplied by 1. . . 48
2.27 The process of ML research 52
2.28 The evolution of machine learning and Network Management 56
2.29 IBM's MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) reference model for autonomic control loops 61

3.1 NFV reference architecture 69
3.2 SDN Architecture 72
3.3 Overall feedback on the importance of the "Proactive SLA violation detection" research area [3]. 77

4.1 PhD Blueprint. 87
4.2 Simplified UML diagram of SLA. 90
4.3 SLA NFV description In ETSI framework 91
4.4 SLO step function of response time. 94
4.5 Cognet architecture 97
4.6 Cognet Smart Engine 99
4.7 Data Pre-processing 103
4.8 Processing Engines. 105
4.9 Processing Engines. 107
4.10 Cognet global architecture sequence diagram. 109
4.11 Simplified Cognitive SLA Architecture. 110
4.12 UML and data model for CogSLA. 111
4.13 Overview of Prometheus data set. 115
4.14 Overview of Prometheus data set. 116
4.15 Network diagram for the streaming use case. 118
4.16 Data distibution of the streaming use case. 118

4.17 Our Testbed. Clearwater virtual IMS functional architecture in the box lower right. Upper left the Cognitive Smart Engine (CSE). The experimental process is : (1) Stress testing for SLA violation generation. (2) System-level supervision. (3) Reporting SLO violations. (4) Data labeling, merging observations on the SLO state and the system-level metrics. 120
4.18 A subset of the data set distribution. The small boxes represent the quartiles of the distribution. The red line in the middle represents the median, i.e. the point separating the data into half. The outliers are drawn as black crosses outside the box. 121
4.19 Cognitive SLA Architecture. 122
4.20 CSE implementation choices. 123
4.21 Raw data in JSON into a Table 127
4.22 Example of the autocorrelation function applied on the load of the SIP proxy. 133
4.23 Lagged autocorrelation. 134
4.24 Transforming raw timeseries into stationary ones 134
4.25 Inta VM correlations 135
4.26 Correlation bewteen all the 156 metrics of the testbed. . . 136
4.27 Example of correlation versus causation. 138
4.28 time series visualization. 139
4.29 T-SNE plot of SLA and SLA Violation. 140
4.30 SLO visualized as a radar map. 141
4.31 Service Quality Monitor. 142

5.1 Zoom on the Main building blocks of the CSE. 146
5.2 Picture of a boxplot. 148
5.3 Media SLA sequence diagram, focusing on the ML process. 150
5.4 FFNN architecture used for forecasting on 4 features following equation 5.1. 154
5.5 Normal inputs vector vs recurrent neural network inputs. 155
5.6 Stacked Stateful LSTMs Trained for prediction 156
5.7 Stacked FFNN for predicting 156
5.8 Difference between FFNN and LSTM for signal prediction. 156
5.9 SLO_1 targeting the Sprout VNFC. 159

5.10 Layout of the dataset used in Use case II. 159
5.11 The activation function $g(x)$ - if non-linear - applied to the output of the neuron allows the ANN to behave as a universal approximator by introducing non-linearity. . . 161
5.12 Combined FFNN with Decision Tree (MLP-DT) 162
5.13 A miniature representation of our experiments, where inputs are limited to 4 (we use 156) and output to one (we used 3). a) is the features represented as time series data. b) is the point of contact, the inputs fed to the ANN model. c) The ANN model which can be LSTM or FFNN. d) is the binary result that categories the inputs into 0 or 1 for non SLA violation and SLA violation respectivelly. . . 163
5.14 Accuracy over all the training steps 165
5.15 Precision over all the training steps 165
5.16 Recall over all the training steps 165
5.17 Precision, recall and accuracy over all the training steps . 165
5.18 Confusion matrix of the Best algorithm (LSTM2) 166
5.19 Results of offline evaluation mode of the FFNN with three different SLO breach threshold (Based on the streaming use case) 167
5.20 Example of a subgraph in a Decision Tree over 10.000 sample. 169
5.21 Results comparing FFNN and LSTM based on the validation and test set. 170
5.22 The Training time of the FFNN for 10,000 samples. 170
5.23 Overall performance of FFNNs vs LSTMs (The model code are in the index section B) 170
5.24 Methodology. 173
5.25 Random search is computationally less expensive and scans a wider area in the configuration space shown as interconnection of gray lines. On the other hand, Random Search is very effective especially when the intuition of the operator fails to approximate the range of the search as shown in the intersection of red rectangles. 177
5.27 Overview of the dynamic ANNs architectures. LSTM on the left and FFNN on the right. 178

5.28 Accuracy versus dropout. We can infer from this figure that dropout regularisation slightly improves the accuracy with no compromise in time (as represented by figure 5.29) 188
5.29 Time versus dropout 188
5.30 Accuracy with respect to Activation functions in the output layer. This figure shows that the choice of the activation layer is amogst the most important ones. 188
5.31 Optimizer type with their respective learning rate α versus accuracy. The best optimizer is Adadelta, the worst is RMSprop. 188
5.32 comparison between LSTM and FFNN by accuracy over the validation set. 189
5.33 ANN initialization method versus accuracy. GN : Glorot Normal, GU : Glorot Uniform, HN : He Normal, HU : He Uniform, LU : Lecun Uniform, N : Normal, O : Orthogonal, U : Uniform, Z : zero. 189
5.34 Top 5 best ANNs based on the mean over Globalmetric . . 189
5.35 Top 5 ANNs based on the mean over validation score accuracy. All the best ANNs appear to converge on 94% average accuracy. 189
5.36 Results for medium ANNs structure (under 30 hidden layers and 20 average neuron per layer). Point A1 is the global maximum. Point A2 is a local maximum. Point A3 is a local minimum. In the text below, we provide an interpretaion of these results. 192
5.37 Feature importance of the table 5.12. 195
5.38 Feature importance of the ANNs' hyperparameters 195

To my family who supported me during my studies

Acknowledgements

By completing this PhD thesis, I achieved my greatest and long aspired goal in my career. It has been with no doubt the happiest and richest experience of my life. The PhD experience enlightened me both personally and professionally. I have lived this experience with passion.

A special thanks to my PhD supervisor Imen Grida Ben Yahia who trusted me to carry out this project and made hereself always available for guidance and support. It has been a pleasure and a honor to me to work with her. I've learned so much I am aware of her print on my work and fr this I am grateful.

I would like also to thank my academic supervisor Djamal Zeghlache who contributed to the elaboration of this thesis and provided me with his precious advice and guidance.

A special thanks to Professor Panagiotis Demestichas and Professor Philip De Turck who evaluated and reviewed my manuscript. I am grateful to all my co-workers at Orange Labs who helped me to integrate the team and made this experience memorable. This project would not have been possible without a three year scholarship support from Orange Labs.

Last but not least, I would like to thank my family members who supported me throughout this endeavor. I would like to thank my mother Jamila Touati my Father Aziz Bendriss my aunts Saida and Joudia Touati and my sisters Yasmine and Yousra to whom I feel deeply indebted.

Chatillon, 2018

Jaafar Bendriss

List of abbreviations

Table 1 – Abbreviation table of common used acronyms.

SLA Concepts	
SLA	Service Level Agreement.
SLO	Service Level Objective.
SLOV	Service Level Objective Violation.
SLAV	Service Level Agreement Violation.
Networking Concepts	
SDN	Software Defined Infrastructure.
NFV	Network Function Virtualization.
NFVI	Network Function Virtualization Infrastructure.
VIM	Virtual Infrastructure Manager.
VNF	Virtual Network Function.
VNFC	Virtual Network Function Component.
VNFM	Virtual Network Function Management.
VNG-FG	Virtual Network Function Forwarding Graph.
ETSI	European Telecommunications Standards Institute .
Machine Learning Concepts	
DT	Decision Tree.
MCDT	Multi-Class Decision Tree.
PCA	Principal Component Analysis.
RMSE	Root Mean Squared Error.
KPI	Key Performance Indicator.
ML	Machine Learning.
RL	Reinforcement Learning.
VM	Virtual Machine.
ANN	Artificial Neural Network.
FFNN	FeedForwad Neural Network.
SVM	Support Vector Machine.
RNN	Recurrent Neural Network.
LSTM	Long Short Term Memory.
RBM	Restricted Boltzmann Machine.
ETL	Extract Transfer Load.
GPU	Graphical Processing Unit.
Other Concepts	
CAPEX	CAPital EXpenditure .
OPEX	OPerational EXpenditure .

Abstract

This thesis addresses cognitive management aspects of Service Level Agreement (SLA) in software-based networks.

Telecommunications operators pushed towards virtualization of their networking function by introducing in 2012 the concept of Network Function Virtualization (NFV). NFV aims at reducing vendor lock-in and bringing agility in the services and resources life cycle operation and management. The IT and networking industries foresee a combined use of SDN and NFV to make cloud and network services agile. Major service and network providers predict that, by 2020, 70% of deployed networks will rely on cloud infrastructures, virtual network functions and multi-domain SDN controllers. This evolution and new requirements call for efficient SLA enforcement and management. This thesis tackles the following concerns : (1) a formal definition of SLA, (2) proactive SLA violation detection methodology and (3) the definition of an extended framework for cognitive management in softwarized networks.

This doctoral work led to an end-to-end data-driven framework, namely, CogSLA, which stands for Cognitive SLA. The framework is based on the use of multiple Machine Learning (ML) algorithms in conjunction for improving prediction and anticipation of SLA violations. The proposed framework is based on two types of Artificial Neural Networks (ANNs), The FeedForward Neural Networks (FFNNs) which showed state-of-the-art results in image classification. And a special type of Recurrent Neural Networks with best performance for Natural Language Processing, the LSTM. Both algorithms have specific advantages and drawbacks, the work was nevertheless able to leverage these algorithms for anticipating SLA violations.

The thesis proposes also a meta-algorithm to optimize the tunings of the Artificial Neural Networks (ANN) algorithms in an acceptable time and performance. The proposed approach relies on biased random selection and the use of meta-knowledge obtained from training the first round of ML algorithms on our data set.

A new metric based on information theory and on entropy is also used to classify and assess the relevance of each Machine Learning parameter with respect to performance and accuracy.

Résumé

Cette thèse traite les aspects de la gestion cognitive du niveau de service (SLA) dans les réseaux virtuels. Les opérateurs de télécommunications ont poussé vers la virtualisation de leurs fonctions réseaux en introduisant en 2012 le concept de virtualisation de fonction de réseau (NFV). NFV vise à réduire le couplage des opérateurs au profit des constructeurs réseaux et à apporter plus l'agilité dans l'exploitation et la gestion du cycle de vie des services et des ressources. Les secteurs de l'informatique et des réseaux prévoient une utilisation combinée du SDN et du NFV pour rendre les services de Cloud et de réseau plus agiles. Les principaux fournisseurs de services et de réseaux prévoient que d'ici 2020, 70% des réseaux déployés dépendront des infrastructures Cloud, des fonctions virtualisées et des contrôleurs SDN multi-domaine. Cette évolution exige une application et une gestion efficaces des SLA. Cette thèse vise à répondre à trois points essentiels : (1) une définition formelle du SLA dans les réseaux virtualisés, (2) une méthodologie proactive de détection des violations de SLA et (3) la définition d'un Framework pour la gestion cognitive dans les réseaux logiciels.

Premièrement, nous avons proposé un Framework axé sur les données de bout en bout, à savoir CogSLA. Le cadre est basé sur l'utilisation de plusieurs algorithmes de Machine Learning (ML) en conjonction pour améliorer la prédiction et l'anticipation des violations de SLA. Le Framework proposé est basé sur deux types de réseaux neuronaux (ANN), les réseaux neuronaux de type FeedForward (FFNN) qui ont montré des résultats satisfaisant dans la classification des images. Et un type particulier de réseaux neuronaux récurrents (RNN) disposant des meilleures performances pour le traitement du langage naturel, le LSTM. Les résultats montrent que les deux algorithmes ont des avantages et des inconvénients spécifiques, mais nous démontrons qu'ils peuvent être utilisés de manière différente pour anticiper les violations de SLA.

Deuxièmement, nous avons proposé un méta-algorithme pour optimiser les ajustements des réseaux de neurones artificiels (ANN) avec un temps d'apprentissage et une performance acceptable. L'approche proposée repose sur la sélection aléatoire biaisée et l'utilisation de méta-connaissances obtenues à partir de la formation du premier tour de Al-

gorithmes ML sur notre ensemble de données.

Enfin, nous proposons une nouvelle métrique basée sur la théorie de l'information et sur l'entropie pour classifier et évaluer la pertinence de chaque paramètre de Machine Learning en termes de performance et de précision.

Chapitre 1

Introduction

The greatest enemy of knowledge is not ignorance, it is the illusion of knowledge.

Stephen Hawking.

I Introduction

The softwarization of networks is a reality today. The emergence of Network Function Virtualization (NFV) and Software Defined Networking (SDN) are expected to bridge the gap between the Telco and IT industries. NFV allows the virtualization of network function by decoupling the software and hardware. A network function can be intrusion detection system, firewall and signaling systems. This allows more flexibility, reduces Time-To-Market TTM and free the Telco from vendor lock-in. The network function can run on commodity hardware such as x86 servers. Consequently, the Telco are expecting to reduce their CAPEX and OPEX. SDN builds programmable networks through abstractions, open APIs (northbound and southbound) and the separation of control and data planes. NFV targets the virtualization of network functions and aims at reducing vendor lock-in and bringing agility in the services and resources lifecycle operation and management. The IT and networking industries foresee a combined use of SDN and NFV to make cloud and network services agile. Major service and network providers

predict that, by 2020, 70% of deployed networks will rely on cloud infrastructures, virtual network functions and multi-domain SDN controllers[1]. This vision can only materialize if automation of dynamic cloud and network services production and deployment are introduced and fully integrated in cloud architectures. This includes 1) faster deployment (from months down to minutes); 2) continuous provisioning in line with the dynamic nature of VNFs subject to up and down scaling; 3) end-to-end orchestration to ensure coherent deployment of IT and network infrastructures and service chains for example and 4) service assurance for fault and performance management including new monitoring and resiliency approaches. Networks are expanding not just in size, but also in complexity. Nowadays network management cost, including SLA penalties, constitutes up to 80% of the global operators' OPEX. With the advent of 5G mobile networks, Operators dread an increase of the network management expenditures. In this context, there is a heavy need to rethink the network management. The new network management approach should handle : SDN and NFV enable the network softwarization which enables the control of the network via software. Resource allocation, flow control, service function chaining will for most part be controlled by programs. One foreseeable consequence of softwarization is the low entry barrier of new vendors, because the network will shift from CAPEX-based models to OPEX-based models [4]. However, SDN and NFV will only be part of a more global software transformation impacting network services, end user devices, IoTs. The reach of network transformation will have social consequence with a new skill set more. Network programmability driven by open SDN APIs together with the shift from vendor lock-in to open source-based ecosystem, will transform the TSP role towards more software development (i.e. softwarization). This entails incorporating DevOps agility and practices (e.g. Continuous Integration/Continuous Delivery) in the service deployments. And foster new business models : IoT, autonomous vehicles, etc.

II Problem Statement

The user's expectations also evolved from expecting simple connectivity to high-throughput networks to more rich services such as augmented reality, online gaming and other latency-sensitive services. From the Telco perspective, networks are getting more and more complex, difficult to manage while the ferorce competition drains their profites low. Network operators see the cost of the software and hardware management gaining in proportion as represented in figure 1.1. The increase of the Network OPerational EXpenditure (OPEX) is largely due to the heterogeneity of devices and software, and to the growing size and the resulting exponential interactions (see Figure 1.1) [1]. Moreover, it is expected that the 5^{th} generation networks will bring about new use cases, high volume of heterogeneous data resulting in an even more complex underlying network. Yet, nowadays network management remains primitive, relying on overprovisioning and reactive strategies. Network administrators still rely on scripts and threshold-based alarms. To overcome the limitation of current approach, namely overprovisioning, research efforts have been applying Machine Learning techniques combined with autonomic computing principles developed by IBM[] to manage network systems. Autonomic computing aims at reducing the involvement of human operators in network management by following high level directives. These high-level directives are represented in the context of this PhD as Service Level Agreement (SLA). The SLA is a formal contract between a service provider and a service consumer that formally describes the expected quality of service. The SLA contains one or multiple Service Level Objectives (SLO) that represents a network measurable metric for a given service level with an expected threshold. The SLA should take into account the dynamic service logic introduced by NFV architecture as Service Function Chaining (SFC). The SLA management should also be expressed in high level using SLOs. The first research problem in the work is to associate effectively High-level SLA to an SLO based on low-level KPIs for the NFV Infrastructure. The second challenge is, based on low level network metrics monitoring how to predict SLOs violation and ensure SLA compliancy ?

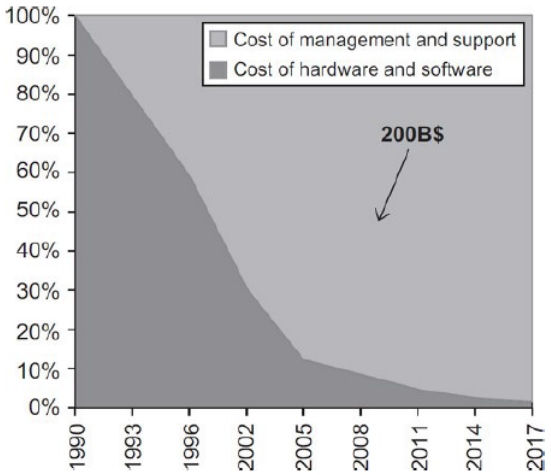


FIGURE 1.1 – The cost of hardware and software and their management [1].

A Research Questions

The recent success of ML techniques especially deep learning and the proliferation of new, heterogeneous data, analytical platforms, distributed monitoring solution across layers and big data solutions along with the rise of SDN/NFV technologies.

The problem statement of this Ph.D. is how to exploit monitoring data in programmable networks (e.g. KPIs, system-level metrics), to determine and execute management operations, adjustment mechanisms, that will maintain and ensure the conformity of PN to a set of SLOs. Therefore, it appears the following Research Questions (RQ) should be investigated :

RQ 1 • Proactive violation detection mechanism : in the ETSI NFV reference framework [5] and the SDN-based networks, what are the measurement points (e.g. NV-VI, NF-VE, SDN controller), the KPIs and measurement directives, that provides exploitable data for an efficient SLA enforcement ?

RQ 2 • How to compute and process in real-time the KPIs and SLOs to augment the probability of accurate prediction of a possible SLO breach ?

RQ 3 • After the prediction of an SLA violation, which counteractive measures (e.g. migrate a VNF) and management constraints and

management actors (e.g. VIM, VNFM) that should be selected and solicited ?

RQ 4 • What are the effective algorithms or combination of algorithms to extract information from networking data ?

B Contributions

Research orientation

The goals of this PhD work are the following : 1) produce a state of the art covering the best practices and research initiatives applicable to SLA enforcement in SDN and NFV (a concise view is within section3) ; 2) identify and define the necessary metrics that are needed to evaluate and monitor SLOs based on inputs from Standards and researches ; 3) and define the corresponding SLA language to ensure machine readable format of these metrics. These previous steps are to 4) define an extendable and cognitive framework for SLA enforcement. It is important to state that this PhD work is a use case within the 5GPPP COGNET (Cognitive networks) project (http ://www.cognet.5g-ppp.eu/cognet-in-5gpp/). Our framework is extendable, as it is intended to cover a diverse group of services beside PNs, such as IoT, and unknown 5G services i.e. we seek extendable languages and templates that, once tuned, enable the establishment of the required agreement levels of different and heterogeneous services.

Contributions

The focus of this thesis is primparly on the application of Machine Learning to SLA management in Programmable Networks. The contributions can be regrouped into three parts, one targeting the literature and pointing gaps and future research directions, another on the design and use of Machine Learning for SLA in dynamic environment and finally a wrapper library to determine the most efficient ML algorithm with respect to performance and precision.

The Scientific Contributions (SC) of this thesis is the following :

SC1 • Cognitive architecture for NFV-based environments.

SC2 • Data analysis methodology and algorithms.

SC3 • Machine Learning model selection and optimization.

SC4 • Cognitive Smart Engine.

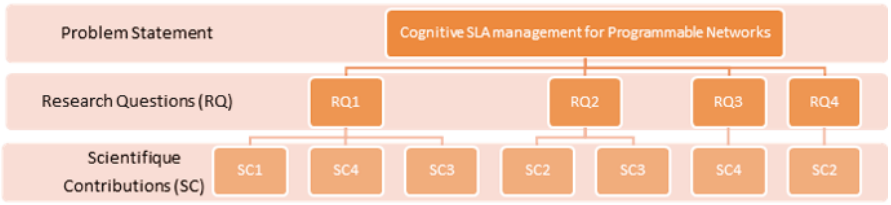


FIGURE 1.2 – Research question mindmap.

All the SCs are grouped in chapter 4. The SC1 is introduced in section III, where we introduce the Cognet global architecture and discuss all its buillding blocks. The SC2 is discussed in details in section IV, named data services in which we present the different preprocessing techniques required for a data-driven approach to SLA management. The SC3 is presented in section III, this contribution contains a meta-learning approach for selecting the best Machine Leanring models. Finally, SC4 present how we combined multiple machine leanring algorithms for different management tasks and is in section II.

Our work and SCs were guided by the different RQs presented earlier. The figure 1.2 presents the mapping between the Research Questions (RQs) and Scientific Contributions (SCs). All the RQs have a common problem statement which is how to use recent advances in the fields of AI and Machine Learning to manage the SLA of Programmable Networks such as NFV/SDN. The RQ1 that deals with mechanism with proactive violation detection is tackled three SCs : SC1, SC3 and SC4. SC2 and SC3 answer the RQ2. Finally, RQ3 and RQ4 are addressed by SC4 and SC2 respectivelly.

III Thesis Structure

Te chapters are grouped into three parts : background material is presented in Chapters **??**, Chapters 4 presents the general architecture and

Chapter 5 zooms on the algorithms and results. The organization of the thesis is shown in the figure below 1.3 :

— **Chapter 2** introduces Machine Learning. We present the basic concepts and the different types of ML. We focus on the connectionist approach consisting of Artificial Neural Networks. We then present the deep learning algorithm and discuss in details its strengths and weaknesses. Finally, we frame the ML approach by presenting its limitations and current scientific challenges. The main contribution of this chapter is the identification of the ML advantages and theoratical limitations for an end-to-end proactive and cognitive SLA Management. Then, we present in this chapter the state of the art of the Artificial Intelligence used for network management. The main contribution of this chapter is to assess and classify the cognitive approaches, how each techniques and algorithms match to each use case. Additionnally, we identify the current gaps in the literature and position our work in the landscape of the cognitive management literature.

— **Chapter 3** presents the state of the art of the SLA. We present the history of the SLA before the Cloud, mainly in IT departements and in early telecommunication protocols. Then, we track the evolution of the SLA after the Cloud era. The main contribution of this chapter is to understand the new requirements of the SLA in programmable networks, how it differentiate from the Cloud Computing requirements as a new research direction, in which direction SLA management should follow, hence the Chapter 3 introducing Machine Learning.

— **Chapter 4** presents the first two contributions of this thesis work. Firstly, we start by introducing the Cognitive SLA Framework (CogSLA). Then, we develop the data analysis part for sofware-based networks

— **Chapter 5** In this chapter, we zoom on the Smart Cognitive Engine (CSE) and present the results for forecasting and classification problems. Next, we introduce the problem of model selection and present our solution based on skewed random selection. Finally, we extend this solution to metalearning capabilities to find a compromise between precision and performance.

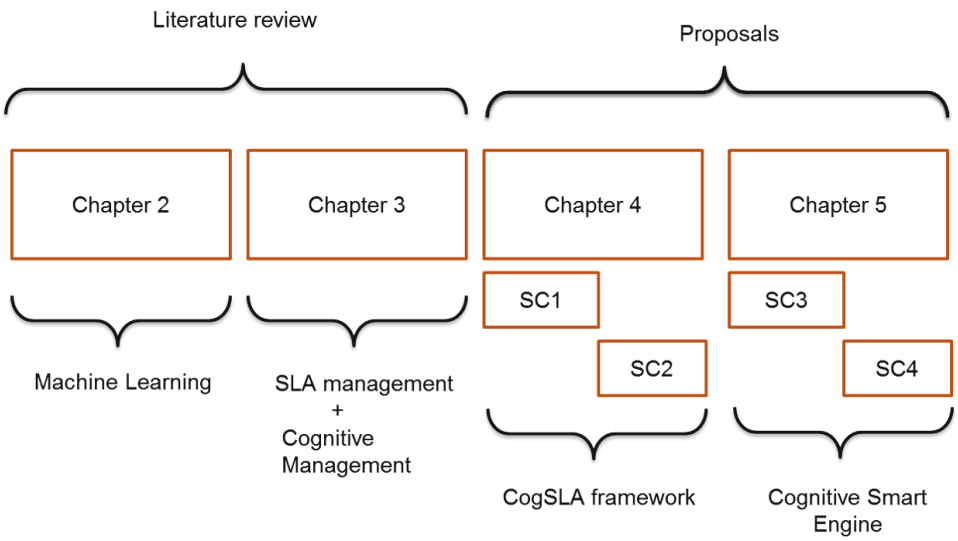


FIGURE 1.3 – Thesis structure

Chapitre 2

Machine Learning : Basics, Challenges, and Network Applications

How is it possible for a slow, tiny brain [...] to perceive, understand, predict and manipulate a world far larger and more complicated than itself?

Peter Norvig.

Contents

I Introduction to Machine Learning (ML) 10
- A Definition 10
- B Machine Learning Types 13
- C A Brief History of Machine Learning 14

II Supervised Machine Learning 16
- A How machines learn. 24
- B Evaluation 33

III Unsupervised Machine Learning 41
- A Clustering 41
- B Anomaly Detection 42
- C Dimetionality Reduction 42

IV Deep Learning 44
- A How Deep Learning is different? 45
- B Exploding/Vanishing gradients problem 45
- C Regularization 47
- D RNN 47

V Machine Learning Latest Challenges 49
- A Machine vs Human Learning 49
- B Scalable Machine Learning 50

VI Machine Learning for Network Management 53
A Machine Learning for Network Management a Brief History . 55
B FCAPS Management 57
C Cognitive Network Management Initiatives 60
VII Conclusion 66

I Introduction to Machine Learning (ML)

Artificial Intelligence (AI) has tremendously grown in popularity due to its recent successes in a host of different domains. Machine Learning (ML) is considered as a subset of this field. It explores the development of algorithms that can learn from data [6]. One of the first successful application of ML back in 1990s was the spam filter based on Bayesian logic. We dedicate this chapter to familiarize the reader with ML. Cutting through the recent hype, to understand how ML can provide cognitive capabilities for the network management. We will provide the reader an overview of ML history and recent successes. We will also emphasize its lasted State-Of-The-Art (SOTA) challenges and limitations and how to mitigate them.

A Definition

ML is a broad term that encompasses very different areas from regression to anomaly detection. In 1959, Arthur Samuel coined the term "Machine Learning" and defined it as follows :"Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed." [7]. Tom Mitchell 1997 defines ML as "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E." [8].

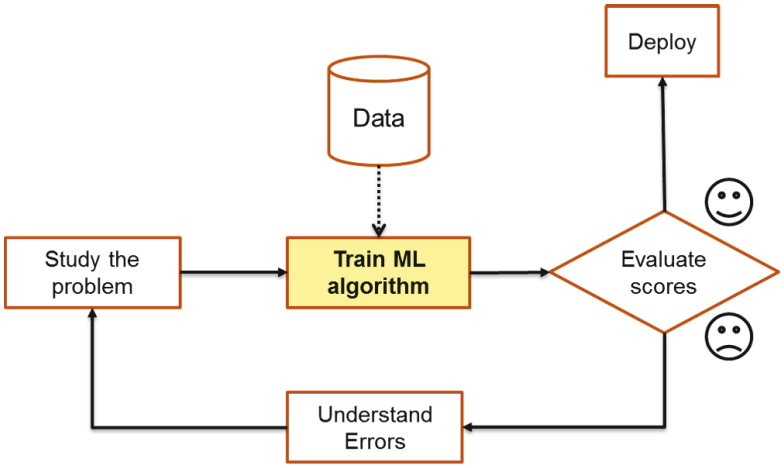


FIGURE 2.1 – Machine Learning approach

However, as we will see in this chapter ML spans to much larger of a definition. Incapable of defining all what ML touches on, we provide a non-exhaustive list of common ML algorithms concerns [9] (to iterate and check) : The inputted data structure has a strict requirement. It must have a tabular shape. Forming lines as samples and columns as features. The used data are considered as samples of real-world data. The inputted data are seen as being drawn from some unobservable distribution. Performing by computers huge calculations unfeasible by hand.

The most common attributes among all ML algorithms is their ability to process only tabular data, where the lines represent the samples and columns the features. These observations are thought as being drawn from a latent data distribution pertaining to a real world application.

Why Machine Learning ?

Assume we want to program a program a spam detector. In traditional programming, we would have to know and code *every* single condition and rule that makes an email a spam. For example, the use of certain words (e.g. ”free”, ”discount”, etc) should be hardcoded with other patterns such as the senders, email subject, etc. The program is most likely to be long and difficult to manage since it codes every pattern than we recognize as a spam. In contrast, a spam detector based on ML automatically learns by itself which pattern are relevant and which are not. Thus easier to manage and to write. Moreover, the update of such as system is equivalent to retraining it on a new dataset, whereas in traditionnal approach it means adding new rules.

ML vs programming. Figure 2.2 shows the difference between traditional programming and ML approach. Another perspective on the difference between ML and traditional programming is that traditional programming is based upon a rule-based approach, i.e. 'if - else - ' statements whereas ML adopts a probabilistic approach. Traditional Programming relies on hard written code for every rule, Long list of rules. The code is complex to maintain and need constant rule update. On the other hand, ML approach learns automatically from the data, the approach is dynamic update and requiere shorter program. Moreover, it is effective for very complex problems and dynamic environments. Finally, Machine Learning can also help us understand old problems by using another perspective based on the data (Figure **??**).

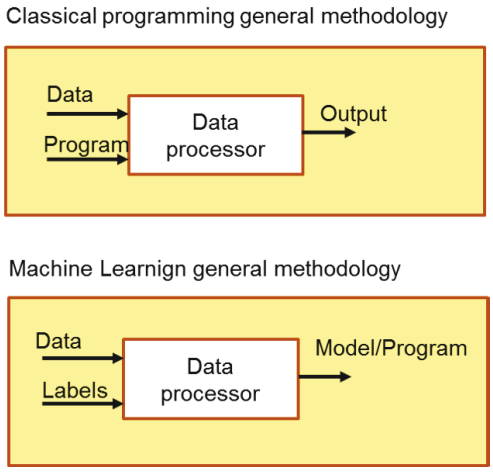


FIGURE 2.2 – Machine Learning vs programming

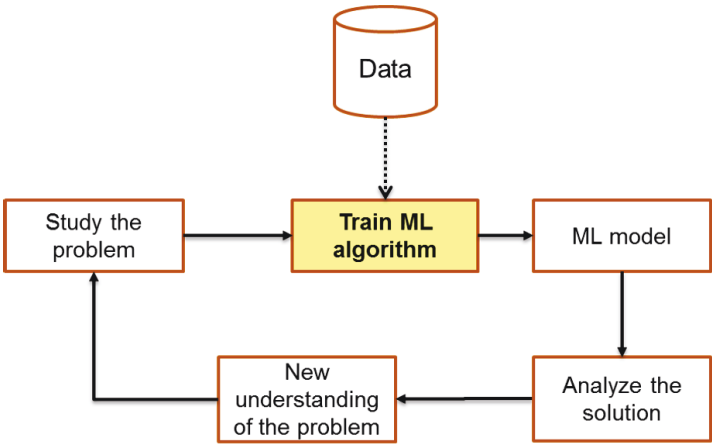


FIGURE 2.3 – Machine Learning can provide deeper understanding of old problems

B Machine Learning Types

ML can be classified broadly in three types : supervised, unsupervised and reinforcement learning. A fourth type is commonly accepted by some researcher as evolutionary learning [1]. In this thesis, we will focus on two most mature and commonly used ones, namely, supervised and unsupervised learning.

Figure 2.4 summarizes the most common ML types along with the most common tasks. Other supervised tasks are possible, though not represented in Figure 2.4, such as ranking and structural prediction for language translation.

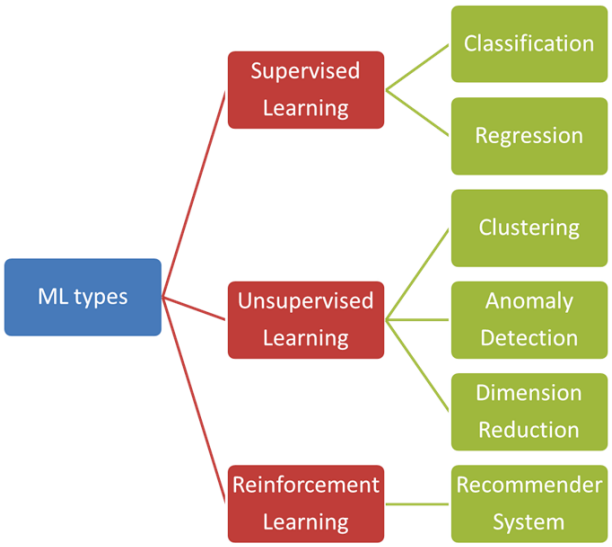


Figure 2.4 – Machine Learning types and different tasks type

Supervised learning. The key concept in supervised learning is labeling. Let's say we want to build an image cat detector system. First we collect as many images as possible and label manually each one as corresponding or not to a cat. We then train the machine by showing it the image and its corresponding label. We then create an objective function that computes the error (or distance) between what the machine outputted and the expected value, i.e. the label. The machine then reconfigures its internal 'knobs' (known as 'weights') to reduce the error accordingly. The typical tasks of supervised algorithms are classification and regression : predicting categories and numerical values respectively.

1. based on the evolutionary biology

One limitation of this approach is the long and tedious work needed for labeling that data. We will delve into more details in the next section.

Unsupervised learning. Unsupervised learning maim goal is to describe hidden structures from a given data input. Contrary to the supervised learning, unsupervised learning doesn't require labeled data. because it does not have a clear target. Fundamentally, unsupervised learning algorithms learn the data structure and characteristics. A common characteristic of unsupervised learning is the non-existence of accuracy metric. Unsupervised learning includes a broad range of algorithms that aim at reducing, transforming and describing the data such as K-means, Principal Component Analysis.

Other perspectives on ML algorithm can be : online and batch learning. All these types are not mutually exclusive, for example an algorithm can be supervised and learns online.

Before exploring the depth of the ML approaches, we present a quick overview of the ML history in the next section.

C A Brief History of Machine Learning

Before the age of computer, philosophers have wondered whether it is possible for machines (or automatons at the epoch) to mimic human thoughts. Rene Descartes and Gottfried Wilhelm Leibniz in the 17th century, believed that human perceptions can be replicated using complex machines []. In the early 20th century Alan Turing demonstrated that any mathematical logic can be translated into a computer program []. He later in 1950 addressed the idea of a learning machine in [10] which predates genetic algorithms.

In the 50s, after the invention of the transistor and the first programmable computers, these ideas could be tested which led to the inception of AI. Pioneering Machine Learning (ML) studies were conducted based on statistical methods (e.g. Least squares, Bayes theorem, Markov chains) and simple algorithms. The first generation of AI researchers thought that AI problems are simple and that they could be resolved in few decades (Herbert Simon). However, they quickly realized the great complexity of such a task. In 1951, Marvin Minsky and Dean Edmonds created the first neural network capable of learning called the SNARC [11](page 105) but quickly realized its limitation (more on this point

in the ANN section in **??**). In 1952, Arthur Samuel wrote the first program that learned from experience. A game of checkers that later beat its creator. The game learned to play against itself, as A. Samuel was very limited in term of resources he used a technique called *Alpha-beta pruning* [2][12].

A few years later, in 1957, an influential paper on ML was introduced by the mathematician Raymond Solomonoff [13]. He proposed a program that solves simple arithmetic problems by observing a sample of correct sequences. The machine learned to solve simple problems such as inferring the meaning of equality sign '=' from observing many sequences. R. Solomonoff used elementary "n-grams" technique that consists of calculating the frequency of appearance of a given n letters and selecting the most frequent one.

In the subsequent years, many algorithms were created such as Nearest Neighbors, Turing's Learning Machine and the introduction of backpropagation algorithm, i.e. the learning algorithm for neural networks. However the integration of ML solutions was hindered by the lack of data. Researchers didn't access huge amount of data to perform training on.

In the 90s, when enormous amount of data was becoming available, the ML approach shifted from knowledge-driven to a data-driven approach. In this period, Support Vector Machines and RNNs (and LSTMs in the late 90s) became popular. In this period of time, one limitation on ML was the training time necessary to train complex models. This led to using simpler models [14].

Another major limitation of ML algorithms was their ability to learn data in their raw format without feature engineering or domain expertise/knowledge. The solution was the representation learning approach which [15] consists of set of methods that allow the machine to automatically discover relevant features needed for detection. The most popular one is deep learning [16]. In 2010s, the advances in GPUs manufacturing and power, deep learning technique significantly improved ML tasks performance in many different domains. Deep Convolutional network revolutionized image processing [17], whereas recurrent networks advanced progress in sequential data such as language translation[18].

2. small description

The development of deep learning was reinforced by the development of many open-source ML frameworks. For instance, in 2015, Google released the first version of TensorFlow.

Today, Deep learning is resolving major problems such as image, speech recognition thought to be insurmountable by AI. Deep learning success has expanded to particle accelerator analysis, drug discovery and DNA analysis [16]. Yet its performance is unquestionable in many tasks, we notice that its application for network and SLA management has not been sufficiently examined. We examined more thoroughly its application on some constrained application of network management in section **??**. In the following section we will provide a more detailed view of ML supervised and unsupervised techniques. After that, we will discuss the current SOTA limitations.

II Supervised Machine Learning

In supervised learning, the learning algorithm takes as input a vector denoted x. This vector consists of multiple training instances (or examples) denoted $x^{(i)}$, with i as the $i^{(th)}$ example. Each example has different attributes (or features) values denoted x_j, with j as the $j^{(th)}$ feature. The feature values can be either discrete (e.g. classes) or continuous (e.g. real numbers).

Thus we express x_j as $x_j = (x_1^{(j)}, x_2^{(j)}, \cdots, x_i^{(j)}, \cdots, x_n^{(j)})$. Similarly we denote the j^{th} corresponding labeled element as $y^{(j)}$ (also termed *target* or *expected value*). An observed example is the tuple $(x^{(j)}, y^{(j)})$, meaning that when we observe $x^{(j)}$ we expect to get $y^{(j)}$. In table 2.1, the first observerd example is $((3, 50, 10), 200.000)$.

# of rooms (x_1)	Surface (m^2) (x_2)	house age (x_3)	$price$ ($y = label$)
3	50	10	200,000
5	200	5	500,000
6	500	25	1,100,000

TABLE 2.1 – Labeled data example for a regression task.

If the output y is a set of single discrete values, we call the task clas-

sification. We refer to each possible set of discrete value as a class. For example given an animal characteristics find which species it belongs to. The animal characteristics are the features and the species is the class.

If on the other hand, the output value is continuous, we call the task regression. For example, given the house characteristics find its price as presented in table 2.1. Based on the house features we would like to *predict* its price.

The training data correspond to the set of examples upon which the learning algorithm will learn and is denoted as D_{train}. The ML algorithm can be represented as a function f_θ (also termed hypothesis H_θ) that maps the inputs to the outputs based on internal 'knobs' (or model parameters θ). The model parameters θ can be thought of as the degree of contribution of a feature to the output. In the context of classification tasks f is commonly refered to as a classifier.

Common supervised algorithms are Linear Regression, Logistic Regression, Support Vector Machines (SVMs), Decision Trees and Artificial Neural Networks (ANNs). Note that the ANN can be used for regression, classification, unsupervised and reinforcement learning.

In order to illustrate all these points and definitions, we will use two examples, a very simple learning algorithm, namely linear regression and a slightly more complex algorithm, Artificial Neural Networks (ANN).

Linear regression is simply an algorithm that fits the data using a linear function of form :

$$f_\theta(x) = \theta_1.x + \theta_2 \tag{2.1}$$

Given a set of training data D_{train} depicted in figure 2.5, we would like to predict the y values of a given input x.

ML hyperparameter. In Machine Learning (ML) we distinguish between two configuration variables : the variables that can be inferred from the data, this is termed the model parameter and the configuration variables that cannot be estimated from the data distribution, termed the hyperparameters. The hyperparameters are considered as an internal part of the ML models. They are generally set manually by the ML designer or require an intuition, using heuristics techniques.

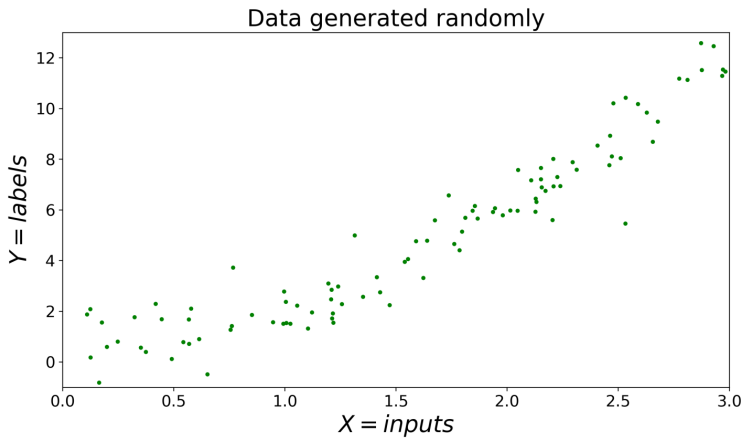


FIGURE 2.5 – Randomy generated data. the x-axis represent the input and y-axis represents the target values.

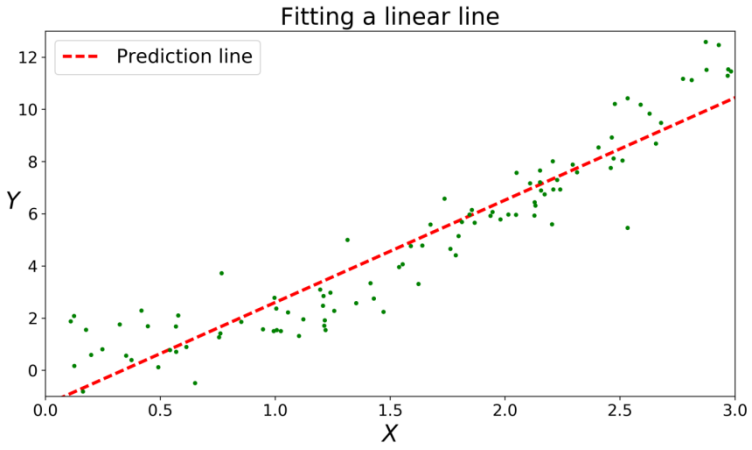


FIGURE 2.6 – Machine Learning types and different tasks type

The Case of ANN.

Artificial Neural Network (ANN) is a machine learning technique loosely inspired by the biological neural cells in the brain. A biological neuron figure **??** is composed of a cell body (nucleus In the figure) where most complex operations are performed. The branching extensions are called the dendrites. At the tip of the branches is a structure termed synapses. Biological neurons interact via electrical impulses called signals via the synapses. A neuron fires a signal when its total received signals exceed a certain threshold.

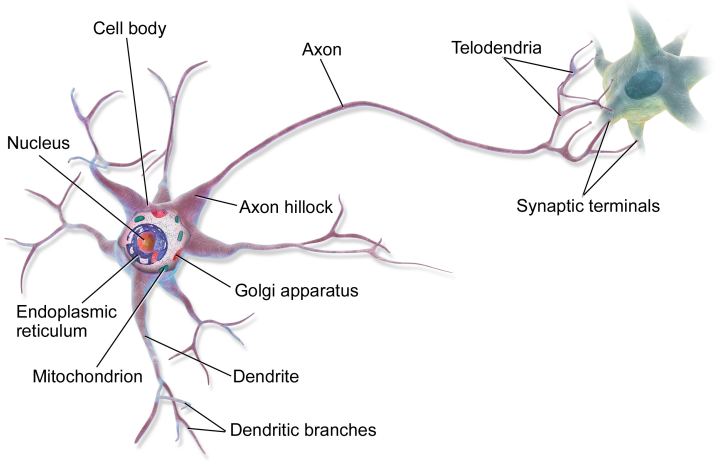
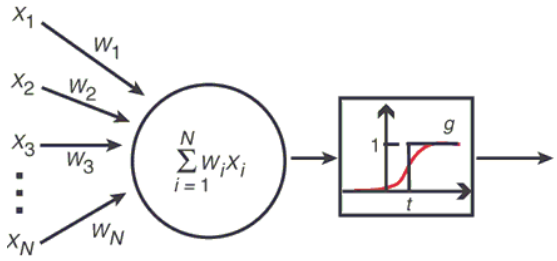


FIGURE 2.7 – a Biological neuron [2]

The artificial counterpart of the biological neuron is what is termed a perceptron firstly developed by Frank Rosenblatt on an IBM 704 computer [19] inspired from early works in neuroscience



(a) fig 1

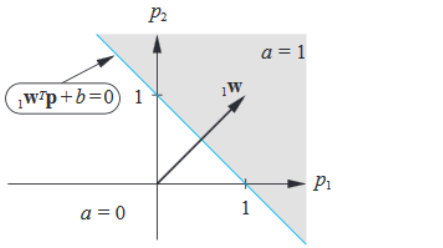


Figure 4.3 Decision Boundary for Two-Input Perceptron

(b) fig 2

FIGURE 2.8 – Artificial perceptron and its decision boundary

The perceptron takes inputs x (instead of binary values), perform some operations and outputs a signal y. As depicted in figure 2.8 each input is associated with a weight w that represents the importance of the input in the total calculation. The perceptron sums the weighted inputs plus a bias term $z = wx + b$ is less than or greater than some threshold value (similar to Biological neuron). Similarity between biological and artificial neuron stops here. Biological neurons are far more complex than perceptron, and we continue to learn more and more of the

biological functions of neurons.

$$output = \begin{cases} 0 & \text{for } \sum_i x_i.w_i \geq threshold \\ x & \text{for } \sum_i x_i.w_i > threshold \end{cases} \tag{2.2}$$

The decision boudaty drawn by a neuron is represented in figure 2.8-b.

For the perceptron the decision boundary is :

$Wp + b = 0$ The decision boundary of the perceptron is orthogonal to the weight vector. The perceptron can only draw lineraly separable decision boundaries.

Perceptrons are linear transformers, they can only split the data linearly. This limitation was pointed out by Marvin Minsky and Seymour Papert in [20].This caused the researches to drop and lose interest in the ‘connectionist‘ approach. More specifically, they pointed out that perceptron cannot learn XOR function (later termed the XOR problem). This problem is illustrated in figure 2.9 one cannot draw a straight line to separate + from -. As a result, the research community lost interest in ANN for several years.

A	B	$A\ XOR\ B$
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.2 – Exclusive OR : XOR Truth table

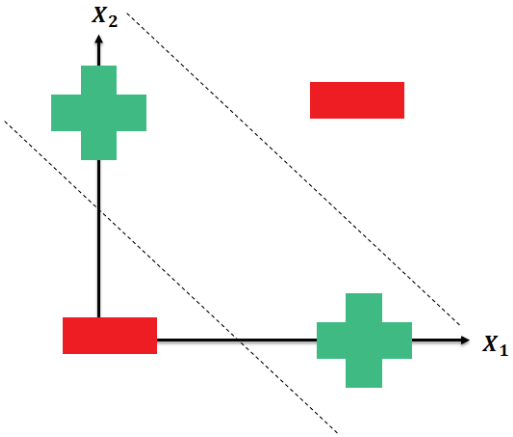


FIGURE 2.9 – Non-linear decision boundary XOR

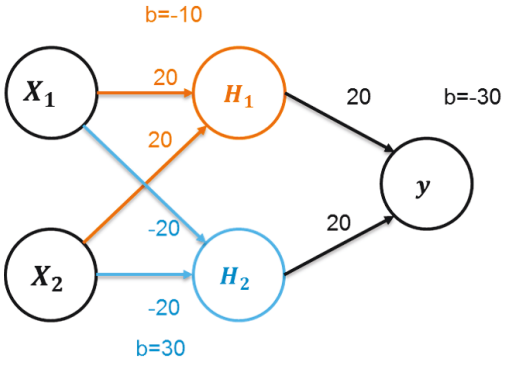


FIGURE 2.10 – MLP solving XOR problem

σ(20*0 + 20*0 – 10) ≈ 0	σ (-20*0 – 20*0 + 30) ≈ 1	σ (20*0 + 20*1 – 30) ≈ 0
σ(20*1 + 20*1 – 10) ≈ 1	σ (-20*1 – 20*1 + 30) ≈ 0	σ (20*1 + 20*0 – 30) ≈ 0
σ(20*0 + 20*1 – 10) ≈ 1	σ (-20*0 – 20*1 + 30) ≈ 1	σ (20*1 + 20*1 – 30) ≈ 1
σ(20*1 + 20*0 – 10) ≈ 1	σ (-20*1 – 20*0 + 30) ≈ 1	σ (20*1 + 20*1 – 30) ≈ 1

FIGURE 2.11 – MLP solving XOR problem

The XOR problem was solved using and OR and an not AND in the hidden layer and an AND gate in the output layer.

A	B	(A OR B) AND (A !AND B)
0	0	= 0
0	1	1
1	0	1
1	1	0

TABLE 2.3 – Exclusive OR : XOR Truth table

Later, this limitation was eliminated by stacking multiple Perceptrons together. This configuration is what we call Multi-Layered Perceptron (MLP). The layers between the input layer and the output layer is what we call the hidden layers. in the human brain there exits 10^{11} neurons of 20 types

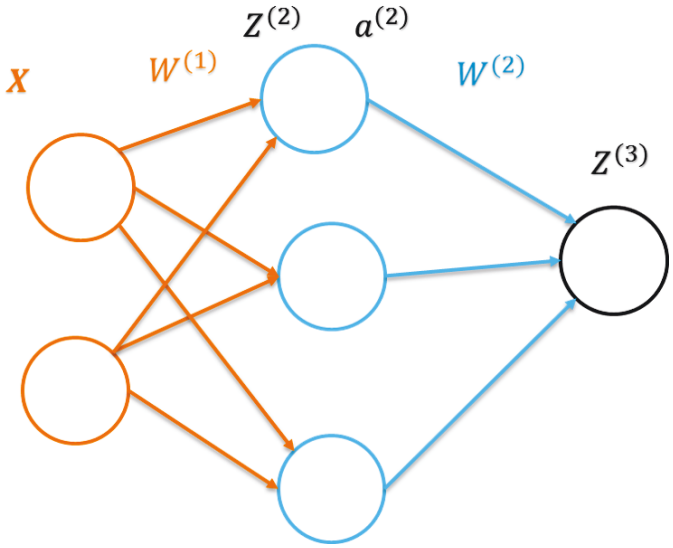


FIGURE 2.12 – simple MLP example

Example MLP.

We will use an MLP example as represented in Figure 2.12 throughout this section in order to illustrate the basic principles behind MLP.

The MLP is a 3-layered ANN, with two input units and one output. The hidden layer is composed of three units. The link between each neuron is refered to as a *synapse*.

More formally, the synapses parameters are termed weights W. $W^{(1)}$ and $W^{(2)}$ refer to the weights of the MLP's synapses of the first and se-

cond layer respectively.

$$X = \begin{bmatrix} X_1^{(1)} & X_2^{(1)} \\ X_1^{(2)} & X_2^{(2)} \\ \cdots & \cdots \\ X_1^{(n)} & X_2^{(n)} \end{bmatrix} \tag{2.3}$$

, where n is the number of examples.

$$W^{(1)} = \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} \end{bmatrix} \tag{2.4}$$

$$W^{(2)} = \begin{bmatrix} W_{11}^{(2)} \\ W_{21}^{(2)} \\ W_{31}^{(2)} \end{bmatrix} \tag{2.5}$$

Instead of the step function defined previously we used the sigmoid function as the activation function. We will explain in the learning phase the reason behind this choice.

The sigmoid function is a function that maps inputs from $[-\text{inf}, \text{inf}]$ to values from 0 to 1, $[0,1]$. it is defined as follows [3] :

$$f(x) = \frac{1}{1+e^{-x}} \tag{2.6}$$

Prediction.

The nominal behavior of the ANN is taking the inputs from the input layers, through hidden layers to the output. This is called the forward pass. In this section, we will follow/describe this process using matrix notation along with the resulting matrix shape. The matrix notation are very helpful in that it summarizes in one line the operations for each neuron. We used Z and A to refer to the weighted sum and the applied activation function over the result respectively (if confused see figure 2.12).

From the input layer to the hidden layer :

Multiplying the inputs by the hidden layer weights :

$$Z^{(2)} = X.W^{(1)} \tag{2.7}$$

3. see the representation of the sigmoid and its derivative in figure 2.16

, where X size is $(n*2)$ and $W^{(1)}$ is $(2*3)$, resulting in $Z^{(2)}$ of size $(n*3)$.

Then applying the activation function :

$$A^{(2)} = f(X.W^{(1)}) = Z^{(2)} \tag{2.8}$$

, $A^{(2)}$ is the output of the activation function and the input of the output and final layer.

Again, the same operation from the hidden layer to the output layer :

$$Z^{(3)} = A^{(2)}.W^{(2)} \tag{2.9}$$

, where $A^{(2)}$ size is $(n*3)$ and $W^{(3)}$ is $(3*1)$, resulting in $Z^{(3)}$ of size $(n*1)$. This means that for each input (i.e. each X row) we get one output.

Applying the activation function :

$$A^{(3)} = f(X.W^{(1)}) = f(Z^{(3)}) = \hat{y} \tag{2.10}$$

, $A^{(3)}$ is the outputed result. We will refer to it as $\hat{y}^{(i)}$ with i as i^{th} output corresponding to the i^{th} input $(X^{(i)})$. In matrix notation $\hat{y} = A^{(3)}$.

A How machines learn.

The fundamental objective of learning is to generalize. This means that the algorithm should obtain good performance not only with respect to the training examples but also over new samples never seen before.

Conceptually, we splitted the two concerns into two sections. (1) Obtaining good performance over the training set and (2) generalizing to new examples. For sake of completeness and simplicity, we dedicate this section "How machine learn" to the first point. We will then tackle the second point in section **??**. Notice that the (1) and (2) are fundamentally opposed. improving (1) hinders (2) and vice-verca. More details on this point later in this chapter.

Machines learn by finding a function f that maps the outputs y to the input x with the minimun error possible. In other words, learning is akin to optimizing a cost function. To this end, we define for each pair of output, label, i.e. $(y^{(i)}, \hat{y}^{(i)} = f_\theta(X_i))$ a *loss function*. The loss function measures the distance between the label (expected value or ground-truth)

and the outputted value. Typically, a loss function is defined as the squared difference between the two values as follows :

$$Loss_{\theta}^{(i)} = Loss((\hat{y}^{(i)}, y^{(i)})) = (f_{\theta}(X_i) - y^{(i)})^2 \tag{2.11}$$

From the loss function, we determine the *cost function* which is a more general term than the loss function. Learning from data means minimizing the cost function. Often, the cost function is computed by averaging over the losses of all the training examples. This is called the Mean Squared Error (MSE).

$$MSE(\theta) = \frac{1}{N}\sum_{i=1}^{n} Loss_{\theta}^{(i)} = J(\theta) \tag{2.12}$$

, the cost function is usually refered to as $J(\theta)$

Back to our linear regression example, in order to compute $J(\theta)$ we should start with an initial θ set (or coefficients). We can initialize these coefficients randomly, then proceed in tuning them to reduce $J(\theta)$ to its minimum.

Gradient Descent (GD).

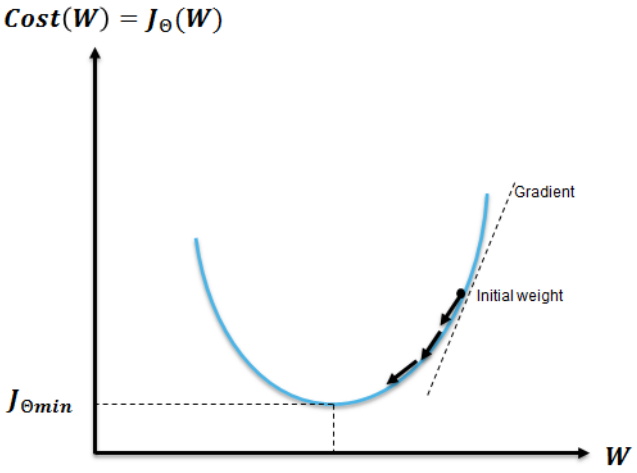


Figure 2.13 – Gradient Descent

We have described in equation 2.13 a method to compute the cost function. Now, we would like to try different θ parameters to find the optimal $J(\theta)$. Trying all the possible model parameters might seem sensible, however, the curse of dimensionality prevent us from doing so,

because of the large explosion of all the possible combinations. A more practical strategy is to compute the cost function derivative to find the direction that decreases the cost. This optimization method is called *Crandient Descent (GD)* A key parameter in this phase is the leanring rate or the step size (α in figure 2.14). if the step size is too small, the algorithm may take a long time to converge (figure 2.14 left). Conversly, if the step size is too large the algorithm might bounce from one end to another without converging (figure 2.14 right).

"It simply starts at an initial point and then repeatedly takes a step opposite to the gradient direction of the function at the current point. The gradient descent algorithm to minimize a function $f(x)$ is as follows :"

Algorithm 1: Gradient Descent algorithm

Data: Get the training data, D_{Train}
Result: Return optimal model parameters θ_j
initialization;
$\theta_j = 0$;
repeat
  read current;
  Update. $\theta_j := \theta_j - \eta.\nabla_\theta J(\theta_0, \theta_1)$ $(for\ j = 0\ and\ j = 1)$
until *Convergence*;

, with η asthe step size and $\nabla_w J(\theta)$ as the gradient

GD is an *iterative optimization* technique that starts at some initial points (i.e. θ), it changes w so that the cost function value decreases, guranteeing the lowest error. The gradient of the cost function comes in handy, because it allows us to determine the slope of the function, it is very practical when working in higher dimensions in order of hundred of thoudands. The GD has two hyperparameters ; namely the step size η and the number of iteration T.

η determines how the aggressive is the algorithms moves if it is set too low or too high see figure 2.14.

T is the number of iterations. each iteration requires going over all training examples - expensive when have lots of data ! Another problem is that it is slow.

GD is slow. There exists many optimization algorithm such as Stochastic Gradient Descent that instead of taking the gradient of the cost

function it uses the gradient of the loss function. SGD updates the weight based on each single example (x, y) instead of all examples. Empirically, SGD performs as well as GD in on epass over all the example as GD in 10 epochs[].

Other optimization method exists, we refer the reader to xxx [] for more details.

$$MSE(\theta) = \frac{1}{N} \sum_{i=1}^{n} Loss_{\theta}^{(i)} = J(\theta) \tag{2.13}$$

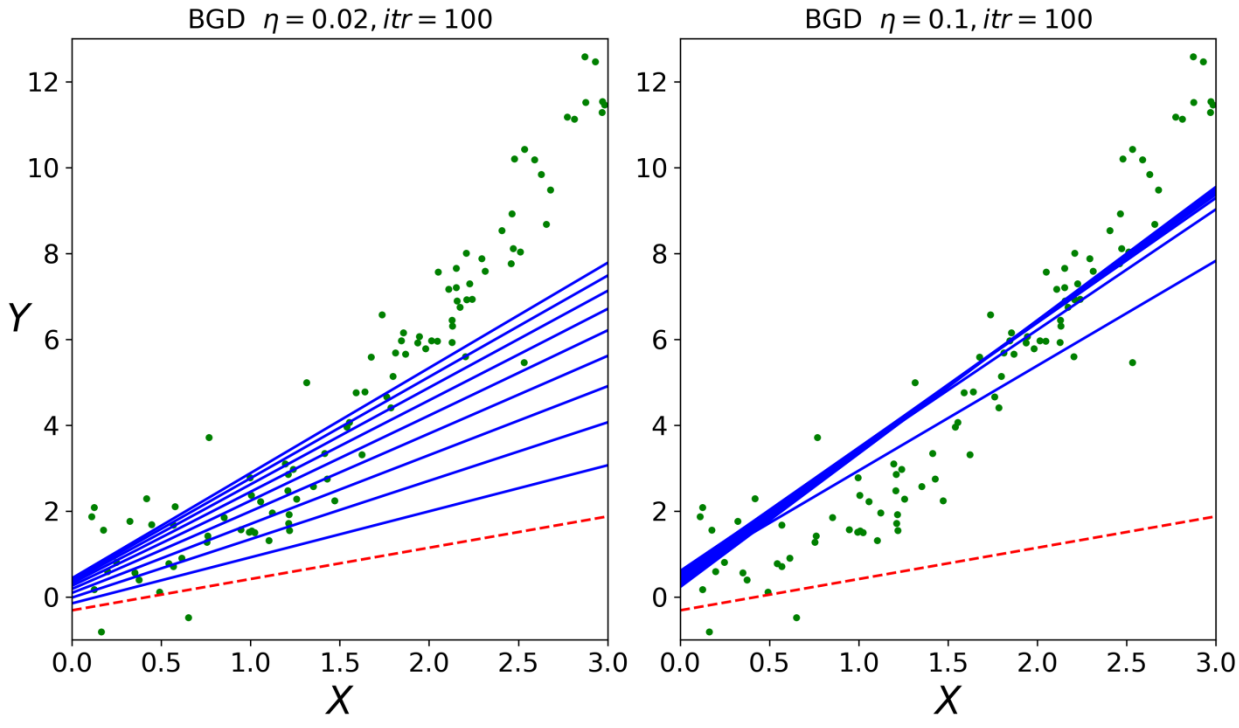


Figure 2.14 – Gradient Descent with a learning rate too small and too large

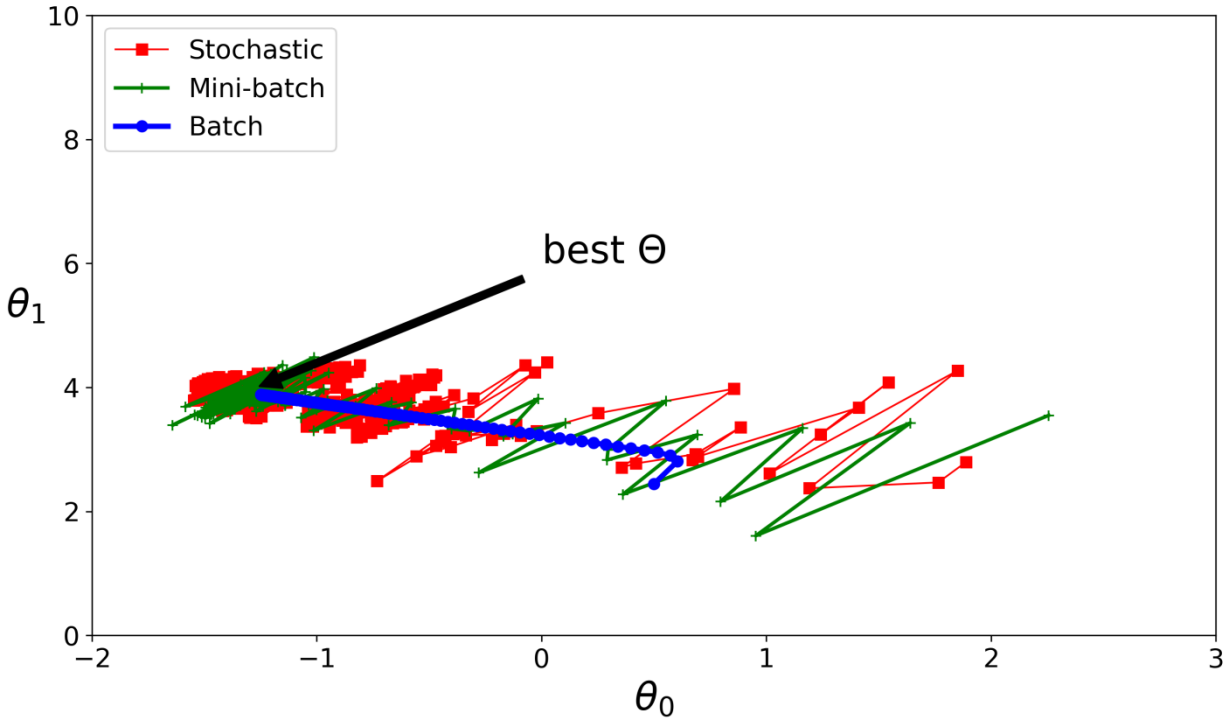


FIGURE 2.15 – Optimizer comparison

”In addition to local minima and the (rare) possibility of getting stuck at a saddle point, there are other issues that should be taken into consideration.

For example, if the step size t_k remains large, it may lead to an oscillatory behavior that does not converge.

Another issue is that, depending on the function and starting point, gradient descent could continue indefinitely because there is no minimum. Consider for example minimizing e^x : there is no finite x for which $\frac{d}{dx}e^x = 0$. Other functions could have such asymptotic minima but also a global minimum of lower value ; gradient descent, depending on its starting point, might forever chase the asymptote, unaware of the true answer elsewhere in the search space.”

The case of the ANN.

The perceptron training algorithm was inspired by Donald Hebb in [21]. Hebb proposed a theory that explains the learning process in the brain. The idea was later summarized as “Cells that fire together, wire together“, this means that in the learning process, the neurons that are stimulated together reinforce their mutual connection. ANNs are trai-

ned using a similar approach, more specifically, for each wrong prediction, the training algorithm reinforce the weights that would have contributed to the correct answer. More formally we write :

perceptron learning rule
w(new) = w(old) + ep
ep = t - a

In the literature, researchers have struggled for many years to train the MLP. In 1986, D. Rumelhart et al. [22] introduced backpropagation algorithm. Rumelhart et al. tackled the XOR problem upfront and demonstrated that with their procedure the network can learn to solve the XOR. Backpropagation caused the second resurgency of connectionism

For each training instance, the ANN computes the output this is called the forward pass. Then the algorithm computes the distance between the expected output and the output. Then, calculate the contribution of hidden layer to the overall error. By propagating the error from the output layer to the hidden layers and to each neuron in the hidden layer. Then it modifies the weight in the opposite direction of the gradient for each neuron. More specifically, it propagates the error gradient backwards, thus the name backpropagation. Rumelhart et al. remplaced the standard step function with the logistic activation function $s(z)$ in order for the backpropagation to work. The standard step functions contains only constant values, the derivative is then always null and cannot be used to compute the gradients as in Figure 2.16 in red. Whereas the Logit function has nonzero derivatives see figure 2.16 green

In order to simplify the backpropagation algorithm we will continue with our small MLP of one hidden layer :

Training of the ANN.

It means to optimize the network weights $\theta^1_{ij}, \theta^2_{ij}$) in order to minimize the cost function $J(\theta)$. In this paper, the Backpropagation algorithm is used to train the ANN. For each training vector $x^(i)$) and label vector $y^(i)$:

$$J(\theta) = \sum_{i=1}^{N} \frac{1}{2}.(\hat{y} - y)^2 \tag{2.14}$$

, the $\frac{1}{2}$ term is added to simplify calculations later on.

In order to apply Gradient Descent algorithm we should compute the cost function gradients with respect to $W^{(1)}$ and $W^{(2)}$:

$$\frac{dJ_\theta}{dW^{(1)}} = \begin{bmatrix} \frac{dJ_\theta}{dW_{11}^{(1)}} & \frac{dJ_\theta}{dW_{12}^{(1)}} & \frac{dJ_\theta}{W_{13}^{(1)}} \\ \\ \frac{dJ_\theta}{dW_{21}^{(1)}} & \frac{dJ_\theta}{dW_{22}^{(1)}} & \frac{dJ_\theta}{dW_{23}^{(1)}} \end{bmatrix} \tag{2.15}$$

$$\frac{dJ_\theta}{dW^{(2)}} = \begin{bmatrix} \frac{dJ_\theta}{dW_{11}^{(2)}} \\ \\ \frac{dJ_\theta}{dW_{21}^{(2)}} \\ \\ \frac{dJ_\theta}{dW_{31}^{(2)}} \end{bmatrix} \tag{2.16}$$

Let's start with $\frac{dJ_\theta}{dW^{(2)}}$,

$$\frac{dJ_\theta}{dW^{(2)}} = \frac{d\sum_{i=1}^{N} \frac{1}{2}.(\hat{y}-y)^2}{dW^{(2)}} \tag{2.17}$$

Recall rule 1. The sum of the differentiation :

$$\frac{d(U+V)}{dx} = \frac{dU}{dx} + \frac{dV}{dx} \tag{2.18}$$

For one training example we have :

$$\frac{dJ_\theta}{dW^{(2)}} = \frac{d\frac{1}{2}.(\hat{y}-y)^2}{dW^{(2)}} \tag{2.19}$$

Recall rule 2. The power rule :

$$\frac{d(U^n)}{dx} = n.U^{n-1} \tag{2.20}$$

Recall rule 3. The chain rule :

$$\frac{dy}{dx} = \frac{dy}{dz}.\frac{dz}{dy} \tag{2.21}$$

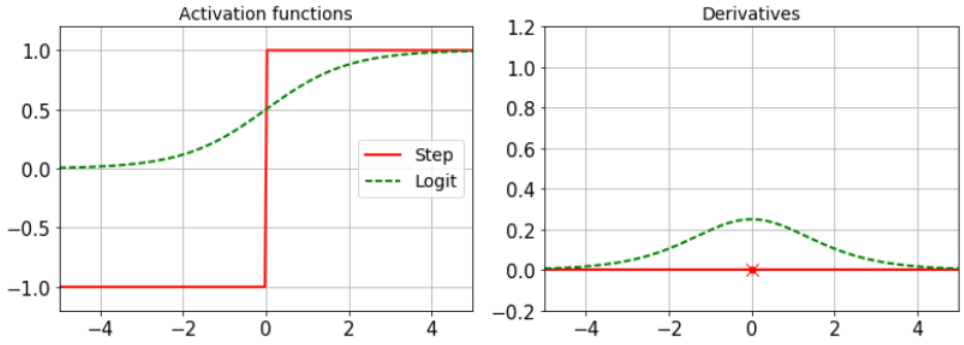


FIGURE 2.16 – Activation functions and their respective derivatives

Using rule 2 and rule 3 in equation 2.19, we get :

$$\begin{aligned}\frac{dJ_\theta}{dW^{(2)}} &= \frac{d\frac{1}{2}.(\hat{y}-y)^2}{d(y-\hat{y})}.\frac{d(y-\hat{y})}{dW^{(2)}} \\ &= (y-\hat{y}).\frac{d(y-\hat{y})}{dW^{(2)}} \\ y = cte \Rightarrow \frac{dy}{dW^{(2)}} &= 0 \\ &= (y-\hat{y}).\frac{-d\hat{y}}{dW^{(2)}}\end{aligned} \tag{2.22}$$

From equation 2.10, we have $f(Z^{(3)}) = \hat{y}$ and using rule 3 :

$$\frac{dJ_\theta}{dW^{(2)}} = -(y-\hat{y}).\frac{d\hat{y}}{dZ^{(3)}}.\frac{dZ^{(3)}}{dW^{(2)}} \tag{2.23}$$

we have $f(z) = \frac{1}{1+e^{-z}}$ and $f'(z) = \frac{e^{-z}}{(1+e^{-z})^2}$ see figure 2.16. we replace $\frac{d\hat{y}}{dZ^{(3)}}$ by $f'(Z^{(3)})$ since $f(Z^{(3)}) = \hat{y}$.

$$\frac{dJ_\theta}{dW^{(2)}} = -(y-\hat{y}).f'(Z^{(3)}).\frac{dZ^{(3)}}{dW^{(2)}} \tag{2.24}$$

Notice that in equation 2.24 the term $-(y-\hat{y}).f'(Z^{(3)})$ represents the gradient error propagated to the synapses in the hidden layers. The multiplication is interpreted as the weights with highest values get the highest blame for the error. This term is usually referd to as the gradient error $\delta^{(3)}$. The step function previously used as the activation function in the first ANNs does not work in this case since its derivative remains

null. It is for this reason that the researchers switched from threshold activation function to the sigmoid see figure 2.16.

For the term $\frac{dZ^{(3)}}{dW^{(2)}}$, we have from equation 2.9 $Z^{(3)} = A^{(2)}.W^{(2)}$. Thus we have :

$$\frac{dZ^{(3)}}{dW^{(2)}} = \frac{dA^{(2)}.W^{(2)}}{dW^{(2)}} \\ = A^{(2)} \tag{2.25}$$

We replace the terms in equation 2.24 while transposing the $A^{(2)}$ matrix to respect the dimensions :

$$\frac{dJ_\theta}{dW^{(2)}} = (A^{(2)})^T.\delta^{(3)} \tag{2.26}$$

The matrix transposed and multiplication takes account for all the training examples. We have $(3 * n_examples)$ for $(A^{(2)})^T$ and $(n_examples * 1)$ which equals to a $(3 * 1)$ matrix size with 3 as the number of hidden layers.

The equation 2.26 is propagating the error from the output layer to the hidden layer. Each synapse gets its share of the error. Next step, we will show how the error is propagated back from the hidden layer to the synapses in the input layer. To do so, we will compute $\frac{dJ_\theta}{dW^{(1)}}$.

$$\frac{dJ_\theta}{dW^{(1)}} = \frac{d\sum_{i=1}^{N} \frac{1}{2}.(\hat{y} - y)^2}{dW^{(1)}} \tag{2.27}$$

similarly to $\frac{dJ_\theta}{dW^{(2)}}$ in equation 2.24 we get :

$$\frac{dJ_\theta}{dW^{(1)}} = -(y - \hat{y}).f'(Z^{(3)}).\frac{dZ^{(3)}}{dW^{(1)}} \tag{2.28}$$

$$\frac{dJ_\theta}{dW^{(1)}} = \delta^{(3)}.\frac{dZ^{(3)}}{dW^{(1)}} \tag{2.29}$$

For the term $\frac{dZ^{(3)}}{dW^{(1)}}$ in equation 2.28, we now from equation 2.9 that $Z^{(3)} = A^{(2)}.W^{(2)}$. Thus, we write :

$$\frac{dJ_\theta}{dW^{(1)}} = \delta^{(3)}.\frac{dZ^{(3)}}{dA^{(2)}}.\frac{dA^{(2)}}{dW^{(1)}} \tag{2.30}$$

We replace $\frac{dZ^{(3)}}{dA^{(2)}} = \frac{dA^{(2)}.W^{(2)}}{dA^{(2)}} = W^{(2)}$

$$\frac{dJ_\theta}{dW^{(1)}} = \delta^{(3)}.(W^{(2)})^T.\frac{dA^{(2)}}{dW^{(1)}} \tag{2.31}$$

And we have for the term $\frac{dA^{(2)}}{dW^{(1)}}$, from equation 2.8, $A^{(2)} = f(Z^{(2)})$.

we write using rule 3 and insering $Z^{(2)}$:

$$\frac{dJ_\theta}{dW^{(1)}} = \delta^{(3)}.(W^{(2)})^T.\frac{dA^{(2)}}{dZ^{(2)}}.\frac{dZ^2}{W^{(1)}} \tag{2.32}$$

We have this term $\frac{dA^{(2)}}{dZ^{(2)}} = f'(Z^{(2)})$ and $\frac{dZ^2}{W^{(1)}} = \frac{dX.W^{(1)}}{W^{(1)}} = X$

Finally we write :

$$\frac{dJ_\theta}{dW^{(1)}} = (X)^T.\delta^{(3)}.(W^{(2)})^T.f'(Z^{(2)} \tag{2.33}$$

We refer to the term $\delta^{(3)}.(W^{(2)})^T.f'(Z^{(2)}$ as $\delta^{(2)}$ the gradient error propagated to the synapses in the input layer.

Using equation 2.33 and 2.26, we can compute the Gradient Descent as follows :

$$\begin{bmatrix} \theta_1 \\ \\ \theta_2 \\ \\ \end{bmatrix} = \begin{bmatrix} \theta_1 \\ \\ \theta_2 \\ \\ \end{bmatrix} - \eta \begin{bmatrix} \frac{dJ_\theta}{dW^{(1)}} \\ \\ \frac{dJ_\theta}{dW^{(2)}} \\ \\ \end{bmatrix} \tag{2.34}$$

, with η as the gradient step size and J_θ as a differentiable cost function.

B Evaluation

Machine Learning diagnosis : Bias and Variance

Failing to generalize can have two forms. First, not being capable of learning the complexity of the objective function, termed underfitting or bias, figure 2.20. Second, learning too much the data characteristics,figure 2.20. A common misconception about overfitting is that the ML algorithm learns the noise in the data. But serious overfitting can be caused without noisy data (e.g. maybe give an example, 40th order polynomial) [14].

In this section, we will take the example of a classifier that tries to detect sick patients from a population. we refer to the population size as N. We refer to the sick patients as $Positive$ or P and the Healthy patients as $Negative$ or TN. We set $N = 100$, and the ground-truth is $P = 5$, $N = 95$. When an algorithm classifies a patient into positive or negative, we denote P_{algo}, N_{algo} respectivelly.

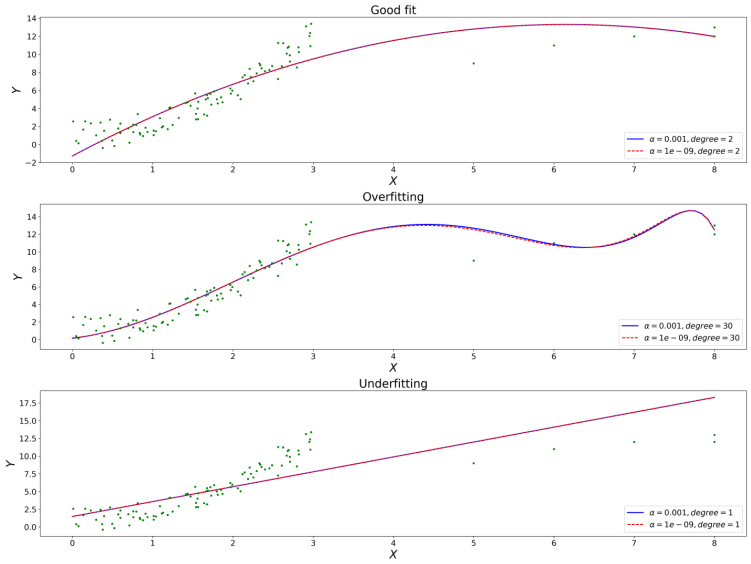


FIGURE 2.17 – bias vs variance

The evaluation of ML algorithm aims to assess how good or bad the evaluation generalizes to new results never seen before. To this end, a basic methodology in ML is to split the data into two sets, a training set on which to train the algorithm and a test set that represents new data on which to test the generalization capabilities.

There exist other techniques to detect and combat generalization problems, we will discuss them in the following sections. In the case of bias the algorithm scores poorly on the training data. In the second case, the algorithm gets a high score on the training set but scores poorly on the test or valisation data set.

Before delving into how to avoid bias and variance, we will define common evaluation metrics with examples for each one to demonstrate the importance of each one.

Machine Learning diagnosis : Confusion Matrix and others

The first evaluation metric used in ML is the *accuracy*. The *accuracy* is defined as the ratio of the correctly classified classes (i.e. sick and healthy patients)[23].

$$accuracy = \frac{P_{algo} + N_{algo}}{P + N} \tag{2.35}$$

One common error for evaluating a learning algorithm is to evaluate the algorithm solely on *accuracy*. *Accuracy* fails to capture the algorithms performance in skewed classes (i.e. when detecting rare events).

To illusrate this point and introduce the confusion matrix, we will follow two algorithms.

$algo_{bias}$: This is a biased algorithm that classifies all the examples as *negatives*. $algo_{tree}$: This is a decsion tree algorithm that : 2 correctly detected with cancer, 3 missed, 5 falsely detected with cancer 92%

N=100	$algo_{bias}$	$algo_{tree}$
Accuracy	95%	92%

Table 2.4 – Accuracy of the two algorithms

We notice in table 2.4 that the biased algorithm manages to get more accuracy than the engineered one. This is because accuracy does not provide the complete view of the classfication performance. The solution to tackle this issue is the confusion matrix. This method aim to evaluate the performance of the classfier in term of classification correctness and error of detection. the confusion matrix relies on some basic key indicators defined in table 2.5.

Basic metrics of confusion matrix	Definition
True Positive (TP)	Number of samples correctly classified as positive
True Negative (TN)	Number of samples correctly classified as negative
False Positive (FP)	Number of samples incorrectly classified as positive
False Negative (FN)	Number of samples incorrectly classified as negative

Table 2.5 – Basic metrics of the confusion matix

Using metrics introduced in table 2.5 we introduce new metrics : *precision* and *recall*. Precision answers the following question : For all the patients that we have detected as positive, how many of them are positive ? Recall answers the following question : For all the patients that we have detected as positive, how many did we miss ?

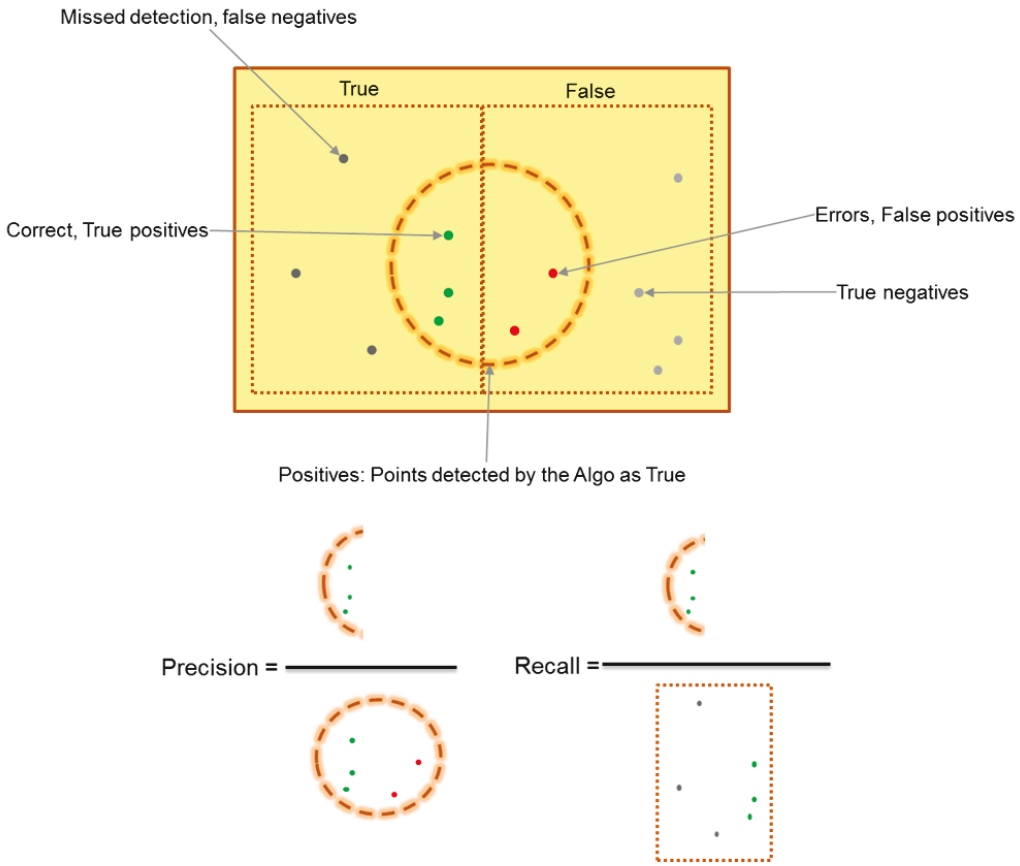


FIGURE 2.18 – Classification metrics. Precision and recall

$$Precison = \frac{TP_{algo}}{P_{algo}} \quad Recall = \frac{TP_{algo}}{P} \tag{2.36}$$

Precision and recall are much more precise than accuracy. However, we would like to have one performance metric in order to compare between many algorithms and to sense the overall performance. For this reason we present *F-score* which is a combination between precision and recall as follows :

$$F - score = 2.\frac{precision.recall}{precision.recall} \tag{2.37}$$

In light of these new metrics, we compare between the two algorithms in table **??**.

	Accuracy	Precision	Fecall	F-score
$algo_{tree}$	92	0.28	0.40	0.33
$algo_{bias}$	95	0	0	N/A

TABLE 2.6 – Comparison between the $algo_{bias}$ and $algo_{tree}$

We notice that algorithm $algo_{bias}$ has better precision than $algo_{tree}$, however, $algo_{bias}$ has a null precision. Based on these new terms we can draw the following matrix in figure 2.19

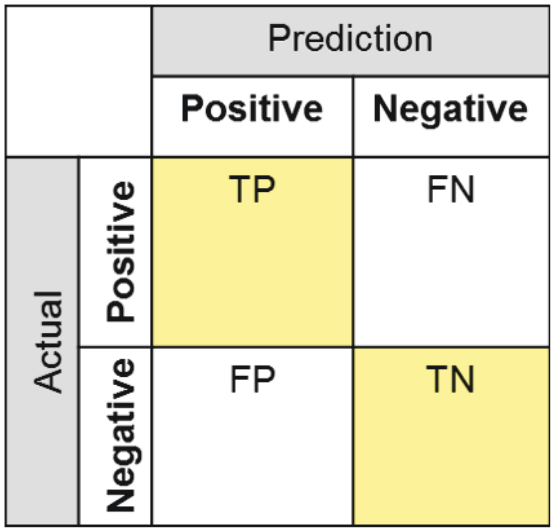


FIGURE 2.19 – Confusion matrix

The confusion matrix is an excellent way to diagnose the ML classifiers. Typically the confusion matrix should be performed over the training and test set. It is an efficient way to detect bias or variance problems. But once this diagnosis step completed, the next step is to resolve the problem. This is what we will do in the next section.

other technique exists such as cross validation ...

Solutions for Bias and Variance

As mentioned previously, the main obstacle on generalization are overfitting or underfitting the data (see figure 2.20). A crucial step before improving an ML algorithm is to detect which type of problem it exhibits.

The bias and variance have symptoms that can be detected when implementing the right methdology. in this section we will go through common solutions for these problems. As depicted in figure 2.20 we classified the solutions into two type : data-oriented and model-oriented solutions.

bias. The main sysmptom of a model that underfits the data is the classification performance over the training set. When the model fails to capture the underlying structure of the data.

Data-oriented diagnosis and solution

Bias can appear because the algorithm didn't find the input feature meaningful enough to discriminate the output. For example given a house size determine the house price. In this example, even the most complex algorithm can't discriminate the prices over just the size of the house. The algorithm can minimize the error, however the optimal error will still be large. This problem is very simple to detect in low-dimensions. However, in very large dimensions this might be tricky. Another varient of this problem is when the algorithm can't retrieve meaningful information from the input. Suppose you entered in the output the house price and a new feature *neighbourhood*, a string with four options [$'upper', 'medium', 'lower'$]. Retrieving sensible information from this feature is very hard, although possible with sophisticate algorithms such as deep networks.

The solution in this case, is represente the feature differenty, in a way that is statistically relevant. This can be done by creating dummies variables, creating three new features, namely, *upper*, *medium* and *lower*. When an input corresponds to a neighbourhood class, set all the features to 0 except the corresponding feature, it should be set to 1.

Also talk about combining multiple features together as inputs. for timeseries one can translate a timestamp to boolean values representing working days, holidays, working hours etc. The creativity is imporant.

Here put table a example

Another data-oriented error might stem from the number of training sample. The more features you have the more samples you need. This grows exponentially following $total\underline{possibilities} = 2^{n_features}$, if the inputs can take only binary values.

Model-oriented diagnosis and solution

For bias problem, a possible root cause can be that the model is too simple, thus cannot capture all the complexity of the data. Suppose using a linear regression for a non-linear problem. The XOR problem discussed previously stemmed from the fact that a single perceptron couldn't separate non linear data, the solution was to improve the model using the MLP.

In the case of Logistic regression, one could increase the degree of the polynomial etc.

overfitting. The main sysmptom of a model that overfits the data is the classification performance over the test or validation set.

Data-oriented diagnosis and solution

The algorithm overfits the data when it scores very well on the training set and poorly on the test or validation set. Possible cause of overfitting can be the drawing few relevent features with many non relevant features. This is a limitation of many ML algorithms, they cannot automatically determine which are the most useful piece of information [4]. Also overfiting can arise when feeding the algorithm with very correlate attributes such as patient high and patient footsize. This type of correlated feature should be avoided.

ANother cause can be that the data are not representative of the problem. In other words, the training data distribution is different from the typical event distribution. (here maybe add a figure or an example to explain)

Model-oriented diagnosis and solution

The common model-oriented issue for variance is that the model is too complex.

There exists many model-oriented techniques that can help combat overfitting. In this overview of ML we will explore three ones. Regularization and dropout.

Regularization aims at penalizing complex configurations ...

Dropout is used in the case of ANN, it consists of removing a number of ANN units...

stochastic gradient descent is preventing overfitting is you stop early. use validation set to now where to stop.

4. Again deep learning appears as an exception to this rule in the literature.

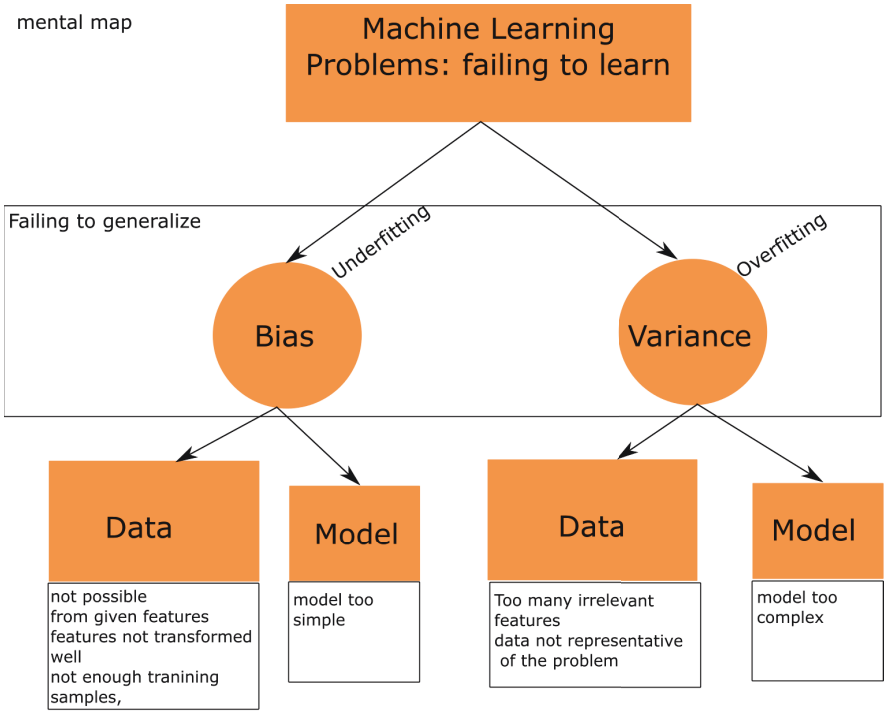


FIGURE 2.20 – bias vs variance

Discussion.

Notice that bias and variance have conflicting goals. Solving bias aims to maximize training score, solving variance problems aims to reduce the training score.

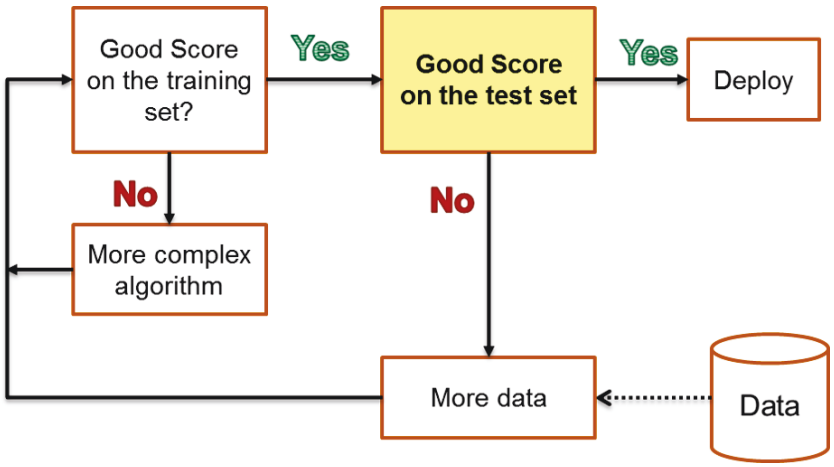


FIGURE 2.21 – Machine Learning Diagnostic process

III Unsupervised Machine Learning

In unsupervised learning the training data set D_{Train} is not labeled. The most common tasks in unsupervised learning is clustering, anomaly detection and dimensionality reduction. As such, the learning algorithm learns to identify and capture the data structure.

A Clustering

Clustering consists in regroup the training data into a small subset of similar charateristics 2.22. The number of clusters k is defined by the ML pratitioner. Common application of clustering is for regrouping the behavior of an oberved system into classes so that we can later reason about these classes. For example, clustering buying behavior in a supermarket, website visitors so that the advertisment can ...

There exist many algorithms for clustering, among the most common are k-means, t-SNE, HCA for Hierarchical Cluster Analysis.

Clustering can also be used in conjenction with supervised leartning, what we call semi-supervised learning. when we have only few labeled data clustering can come in handy to propagate the similar labeled data to all the unlabeled data in the same cluster.

Finally, clustering algorithms can also be used for visualisation purposes. for example in figure 2.22. In this example the original data have been transformed from 156 to 3 dimensions.

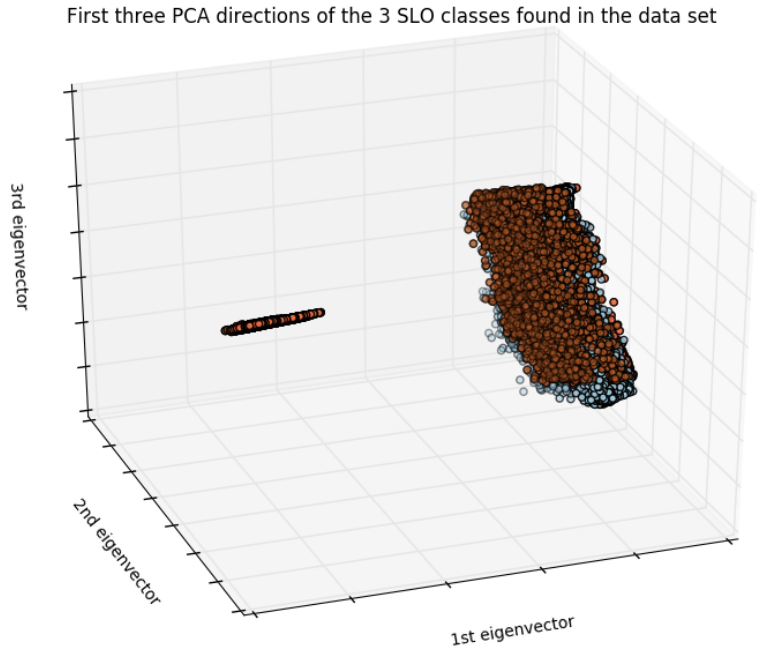


FIGURE 2.22 – Clustering example

B Anomaly Detection

The main idea of anomaly detection is to learn the system *base line*. Then, Any new data observed that is not bound in this base line is considered as an anomaly. Thus it is critical to careful when defining the base line. The observed 'normal' data should be representaive of the system behavior. The distance or deviation (usually standard deviation σ) from the base line (usually mean μ) can be used to express the probability that this new observation is an anomaly.

C Dimetionality Reduction

In statistics and machine learning dimensionality reduction is the process of reducing the number of random variables under consideration for the study problem [24]. The objective of the process is to reduce the set of principal variables. This process is considered as either feature selection technique or feature extraction method [25].

The most used algorithm is Principal Component Analysis PCA [26] based on eighen decomposition. PCA executes a linear mapping of the

data from the initial dimension to a lower-dimensional space, while maintaining the data variance the low-dimensional representation maximal. Dimentionality reduction can be critical to visualize and understand data originaly represented in high dimension space, by representing it into 3 or 2 dimension space.

Other algorithms based on neural networks can be used to capture data charcteristics such as autoencoders and Boltzmann Restricted Machines. Autoencoders for example (represented in Figure 2.23), are used to learn the encoding (i.e. the representation) of a set of inputs X for the goal of reducing its dimesion.

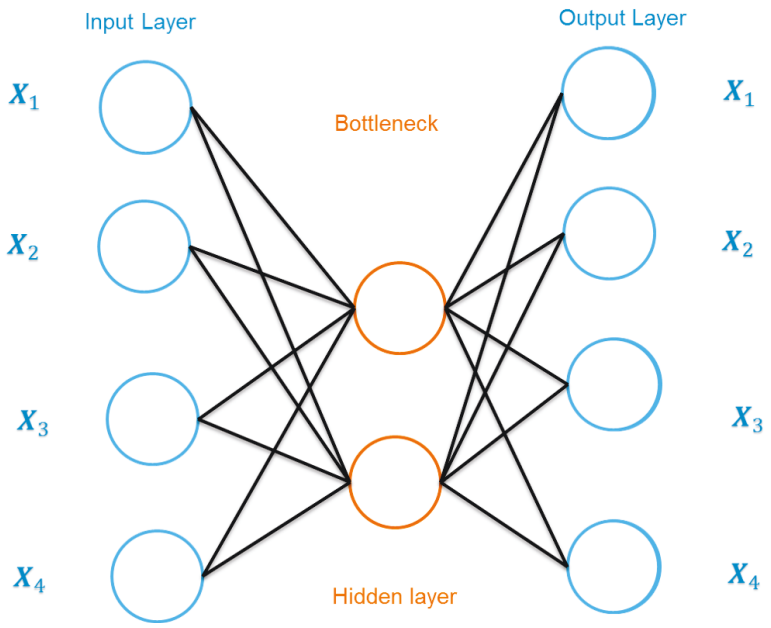


FIGURE 2.23 – Shemetic structure of an autoencoder with one fully connected hidden layer.

Similar to the MLP depicted in figure **??**, the autoencoder 2.23 has an input layer, an output layer and one or more hidden layers connecting them. The output layer having the same number of nodes as the input layer, and with the purpose of regenerate its own inputs X, instead of predicting the target variable Y.

An autoencoder always consists of two parts, the encoder and the decoder, which can be defined as transitions ϕ and ψ such that :

$$\phi : X \rightarrow F \tag{2.38}$$

$$\psi : F \to X \tag{2.39}$$

$$\phi, \psi \tag{2.40}$$

The dimensionality reduction reduces significantly the time and space capacity necessary to store the data. Moreover, it can remove the multicolinearity in the data set (i.e. different features describe the same phenomena, e.g. height and hand size) this improves the performance of the manchine learning algortirhms.

IV Deep Learning

For long in the ML researchers taught that training a deep neural network is hopeless since the 1990s [5] [28, 6]. But in 2006, Geoffrey E. Hinton et al. demonstrated that it is possible and proposed a Deep ANN called Deep Belief Networks (DBN) [29]. They demonstrated that DBN could achieve more than 98% precision on handwriting digits. Since then, the research community became more aware of the potential of Deep Learning (DL). The key drivers for such techniques is : (1) The development of programmable parallel processors, i.e. GPU [6] [30] and (2) the availability of huge amount of data.

The success of DL extended to new domains with state-of-art performance. From molecular biology, speech and image recognition to self-driving cars [16]. There is no consensus in the literature with respect to the definition of DL. Common definitions state that deep ANNs have more than 2 hidden layers others talk about more than 10 hidden layers. The most successful types of DL algorithms are Deep Neural Networks (DNN), Convolutional Neural Network (CNN) for image classification, Deep Belief Networks (DBN) and Stacked Auto-Encoder (SAE). These algorithms surpassed most existing ML algorithms in image recognition, Natural Language Processing and Voice recognition among others.

5. Y. Lecun used deep convolution networks with success at this time, although it was specific to image recognition and didn't generalize to other domains [27]

6. GPUs use hundreds of parallel processor cores executing tens of thousands of parallel threads

A How Deep Learning is different ?

The accuracy and performance of ML algorithm depends on the choice of their feature representation. It is for this reason that efforts are focused on finding the optimal preprocessing and transformation techniques to feed the ML algorithms. Usually, humans intervene at this stage to engineer manually the best representation of a given feature. This human intervention compensate for the weakness of ML algorithm into finding for theselves the best possible data representation. As such, DL is considered among the most promising techniques for making the ML algorithms less dependent on human ingeniuety. DL can court-circuit human intervation due to its ability to construct high-level abstraction of the inputs in the deeper layers [15].

The more neurons we add the number of the connexions grow exponentially, the amount of compuatation grow exponentially. The advantage of ANNs and deep ANN over other ML algorithms is the distributed representation, with multiple levels : having exponential gain for machine to represent well. very large networks can generalize pretty well ! But it needs a lot of labeled data. The DL is a strong algorithm to match complex data pattern. However, it suffers from some problems such as vanishing gradient and overfitting. We will discuss these concerns briefly below.

B Exploding/Vanishing gradients problem

As discussed previously, the backpropagation algorithm works from the output layer to the input layers. It propagates the error gradient on each neuron. The algorithm then computes the gradient of the cost function with respect to each parameter in the network.

The *vanishing gradient* problem occurs when the gradients became smaller and smaller for each iteration. This can block the algorithm from converging to a good solution. Often, the opposite problem occurs, called *exploding gradient*. In Figure 2.24, the sigmoid function is depicted. We can notice at the edge, when the inputs became larger and larger, negative or positive, the sigmoid function saturates at $y = 0$ or $y = 1$ with $\frac{du}{dt} \approx 0$. Hence, the backpropagation algorithm at these points has no gradient to propagate back to the lower layers.

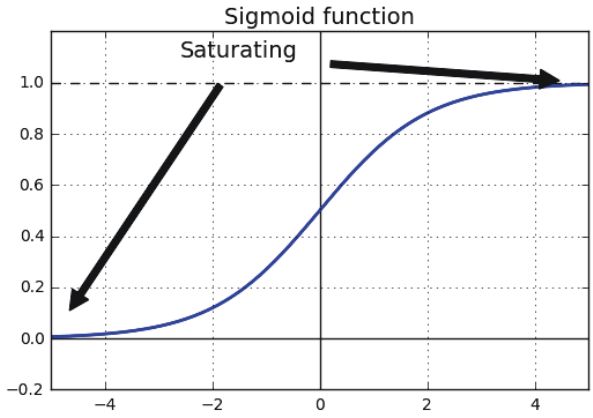


FIGURE 2.24 – Saturation

ReLU (equation 5.9 and Figure 2.25) : ReLU is consider as ”new” non-saturating activation function, its gradient is either 0 for $inputs < 0$ or 1 for $inputs \geq 0$. This means that when using many layers it just multiplies the gradients by 1. This reduces the likelihood of vanishing or exploding gradient problems. For this reason, ReLU outperfoms the other activation functions is Deep NN. [31] when it is trained in a single global training, i. e. not using the ”freezing training process”.

$$RELU(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{2.41}$$

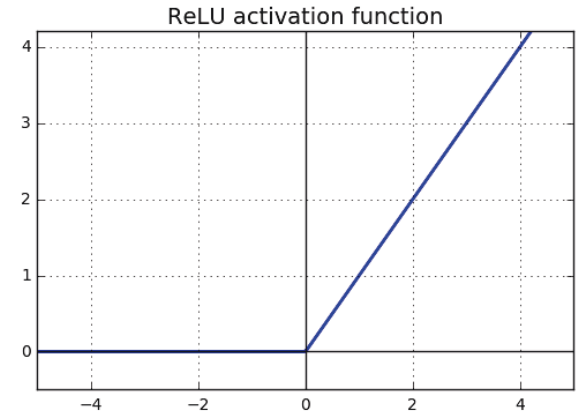


FIGURE 2.25 – RELU activation function

C Regularization

As discussed in section B, overfitting is a common problem in ML. However, in DL overfitting is the main issue, due to the large numbers of neurons (hundreds of thousands). With all these parameters the network can fit a multitude of complex data sets. This flexibility is what makes the DL prone to overfitting.

Fortunately, there exist techniques to prevent the DL algorithms from overfitting. In the following sections we will discuss the Early stopping and dropout methods because they were used in our proposal. Other techniques exists such as data augmentation, l_1, l_2 regularization and max-norm regularization [6].

Early Stopping : Early Stopping consists of saving the multiple copies of the network for each performance. Then select the copy of the network with the best performance on the training and test set.

Dropout : Dropout was introduced by G. Hinton in 2012 [32]. and explored more by N. Srivastava [33]. Dropout consists to give each neuron a probability p to be temporarily droped. This means that it can be deactivated during this training step but active in the next step. Dropout are considered a very powerful regularization technique with increase in performance going from 5 to 3 % [33].

In DL algorithms regularization is an important step to reduce overfitting. The key to succeed in this phase is to combine multiple regularization technique in an iterative manner.

D RNN

Recurrent Neural Networks (Figure 2.26) are a special type of ANNs, where the output of the node is redirected to the input. The next input is added or multiplied by the previous output, this means that the next steps depends on the previous step. However, RNNs suffer from long-term dependencies caused by fading error signals during the training phase, hence the invention of the LSTM [34]. LSTMs are designed to avoid the long-term dependency problem. Their default behavior is to remember information for long periods of time.

The main building block of LSTM is the LSTM cell as shown in figure 2.26 has the property of controlling the amount of information it

can remember. This property is materialized by three gates inside the LSTM cell :

— i_t : The input gate.

— f_t : The forget gate.

— o_t : The output gate.

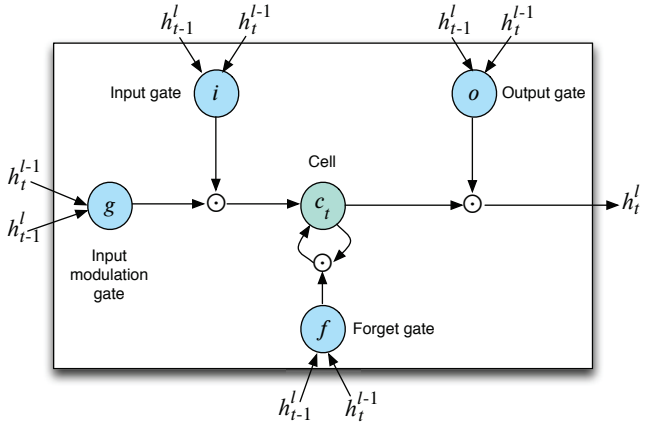


FIGURE 2.26 – LSTM Cell : The element wise multiplication is key for LSTM cell, helping the preservation of constant error when backpropagating the error. The forget gate is an identity function, when it is open the input is multiplied by 1.

These gates are responsible of the flow volume of data that passes through the LSTM cell. More formally, for time step t, and x_t as input, the flow that passes through the LSTM cell can be written as follows :

$$\begin{aligned}
&(1)\ g_t = \psi(W_g \cdot [h_{t-1}, x_t] + b_g) \\
&(2)\ i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\
&(3)\ f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\
&(4)\ o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\
&(5)\ C_t = f_t \odot C_{t-1} + i_t \odot g_t \\
&(6)\ h_t = o_t \odot \psi(C_t)
\end{aligned} \tag{2.42}$$

, where ψ is the tanh function, $\odot$ is the element wise multiplication, σ is the sigmoid function, W is the weight of the gate, [] represents the concatenation operation.

LSTMs improve significantly what we can accomplish with RNNs. The next step in the LSTM research will focus on attention. Researchers

suggest that RNNs can successfully describe images providing automatic captions. For example, Xu, et al. [35] explored how LSTM can be used to describe images. Other very promising RNN research are focused on the improvement of existing algotirhms such as Grid LSTM [36], Image generation [37] and stochastic RNNs [38].

V Machine Learning Latest Challenges

Many scientists consider that ML was born as engineers fail to reproduce human scale intelligence on machines[9, 39]. Researchers discovered that complex human tasks are simpler to program on machines than simple human tasks. With hindsight, we should remember that the ultimate goal for AI research is to reproduce a general-purpose intelligence. However, we are still far from achieving this goal. In this section, we will discuss current ML challenges and research direction for a more holistic and qualitative intelligence.

A Machine vs Human Learning

The first challenge we see in ML is the gap between the current ML techniques and human cognitive abilities. Humans learn with few labeled example and can generalize easily. Moreover, we can reuse one learned knowledge across domain, this is called transfer learning.

Labeled data

Recent success of deep leanring in image recognition is based on huge amount of labeled data. The labeled data are costly and usually requires manual labeling. it is difficult to get many labeled data for many real world problem. In fact, humans learn by observing the world categorizing objects learning their name then generalizing to other similar objects. Human babies learn to regonize object with only limit labeled data. Based on few examples, they can generalize well. In the ML research a promising field for solving this challenge is *semi-supervised learning*. It is the combination of unsupervised leanring and supervised learning. the unsupervised learning algorithm captures the data characteristics and then generalizes using few labeled data to similar data structure. A

successful application of this technique was the Deep Belief Network (DBN), where reseachers stacked multiple Restricted Boltzmann Machines (RBM) to capture the data structure then used a softmax layer to train the network on fewer labeled data [40].

The second gap between human and machine learning is the ability of humans to ignore irrelevant features and data. Humans are bombared by data. Every seconds, through different perceptual channels, however we managed to utilise only the necessary information. ML algorithms fall short in identifying relevant information, this causes (as we have seen before) overfitting. The most promising ML research in this direction is deep learning, however the training time is very expensive even on GPUs. Ultimatly, unsupervised learning will constitute the major progress or even the next breakthrough, since the majority of data on the internet is unlabeled. The research in new techniques of unsupervised learning with complex reasoning might yield the greatest benefit [16].

Transfer learning

The general-AI optimism was revived with the recent success of deep learning. However, machines are still too narrow, specializing on a single task, failing to generalizing across domain. This is one of the biggest challenges in ML. Neural networks can learn using a huge amount of labeled data to discriminate dogs' pictures but they need another huge amount of labeled data to classify cats. Each time the learning should start from scratch. Generally, ML do not have the ability to generalize inter-domains. In recent years, transfer learning has emerged as a new learning approach to address this problem [41]. Most existing transfer learning algorithms so far tackles the generalization across different domains. Some promising work have been done using higher level ANN learned features to retrain more rapidely on other tasks [42, 43].

B Scalable Machine Learning

Distributed Learning

Most ML algorithms learn in a centralized fashion. The best ANNs are trained on one computer using one or multiple GPUs. However, many

real world problems are distributed by nature. Thus the need for (1) new distributed ML algorithms that can run on different machine distributed across the network and (2) more sophisticated way to process huge amount of data efficiently. ML algorithm should manage distributed resources and utilize many computers working together to solve a given problem. Some early works have proposed a way to train multiple small ANNs across devices that rivals the performance of deep networks in [44]. Also design system such as HADOOP or Spark MLib can constitue a first basis for such a system.

Acting in the world & Training time

The next challenge and recent trend in deep learning is in applying it for reinforcment learning. Reinforcement learning consists of a learning agent that interacts with world and learns from experience. Once it undersands the world, it should be able to perform new things. How a learning agent can discover good representation about the world, how the agent can learn to control its world. like babies do. The second challenges is the existence of high range of possibilities.

Working in high dimensions

The curse of dimensionality is a termed coined by R. Bellman in 1961. Bellman noticed that for many ML algorithms that work well in low dimension became intractable in higher dimensions [45]. This issue materializes in ML by the fact that generalization becomes exponentially difficult to obtain when with increasing the number of features[14]. Let's say we have 20 features as integers between 0 and 9, and a training set of $1million$. The total size of all the possible combinations is $10^{20} = 10^{14}Million$. The training data corresponds to $\frac{10^6}{10^{20}} * 100$ % The problem gets worse if we add additional features that additional irrelevant features. The classifier gets confused and become equivalent to a random guessing algorithm. Moreover, adding more features that are correlated with each other (say height and arm length) decreases the overall performance of the ML algorithm. In our work as we will see in next section, the most difficult problem we face in high dimensions is that the human intuition fails to follow. Thus, impeding us to adopt

and propose relevant solutions and improve the algorithm performance. Fortunately, there is techniques to reduce dimensionality. And also for many problems the data are not spread uniformly across all the possible space range. For example for digit recognition, the relevant patterns are concentrated in a lower-dimension space.

in Learning, the curse of dimensionality materialized by the fact that moving downhill using gradient descent entails that in all the dimention space all the slopes should go donwhill which is exponentially less probable. The probability of the dimensions curve down becomes exponentially smaller Moreover it is impossible for hummans to visualize the landscape beyond 3 dimensions. For example optimizing an ANN over 10.000 or a million dimension is conceptually challenging

Machine Learning System Design

Research in Machine Learning applied to the real world scenario is empirically based and is an iterativaly based 2.27. The first step is to conceptualize an idea and the design of the Machine Learning system. Once the idea and brainstorming done, the implementation and development is performed. Then, the experimentation phase to assess the global performance and accuracy. Once this cycle completed the first iteration

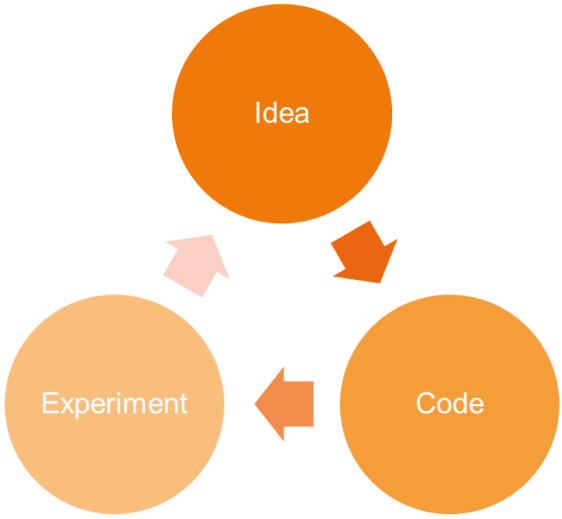


FIGURE 2.27 – The process of ML research

VI Machine Learning for Network Management

In the literature, there is a large consensus on the necessity to integrate ML and Artificial Inteligence (AI) onto the network management process. Surprisingly, this is not a new trend. The first implementation of AI for network management dates back to the late 1980s. Since then, the AI integration in network management went through different stage of evolution that we will describe in this chapter. Usually in the literature, works targeting ML for SLA management targets specific aspect of network management, for example network performance [46, 47, 48, 49], configuration management [50, 51, 52, 53, 54] or traffic classification[55, 56, 57]. In this chapter, we regrouped all these puzzle pieces together forming a holistic view on ML for network and SLA management.

In the last decade, the network users' expectations and network complexity grew immensely while operators seek to decrease their maintenance cost. ML emerges as a natural solution, especially with its recent successes. Additionally, we zoomed on what we consider a very promising ML algorithm for SLA management, namely deep learning [16]. We present the specificity of deep learnings for network management, it potential and its main challenges and research directions.

One of the biggest challenges when applying ML for network operation and control is that networks are inherently distributed systems, where each node (i.e., switch, router) has only a partial view and control over the complete system. Learning from nodes that can only view and act over a small portion of the system is very complex, particularly if the end goal is to exercise control beyond the local domain. The emerging trend towards logical centralization of control, brought by SDN, will ease the complexity of learning in an inherently distributed environment.

Moreover "softwarization" of the network and current network data plane elements, such as routers and switches, are equipped with improved computing and storage capabilities. This has enabled a new monitoring solution, commonly referred to as network telemetry [3]. Such techniques provide real-time packet and flow granularity information, as well as configuration and network state monitoring data, to a centra-

lized Network Analytics (NA) platform. In this context, telemetry and analytics technologies provide a richer view of the network compared to what was possible with conventional network management approaches.

The self-driving network is defined as one where (1) network measurement tightly integrated with the control of the network ; and (2) network control relies on learning from large-scale data analytics of the entire networked system, as opposed to individual protocols. Many recent initiatives have stated this high-level goal [58, 1, 59, 60].

Recent initiatives for using recent advance in Machine Learning for network managemend. : inspired by self-driving cars, Self-Driving Networks [58] by Juniper Networks. Other initiatives such as cognitive network management inspired from IBM's autonomic systems [1]. We notice that the general literature is aware of the necessity to enable the network to run it self [59]. Another noticable project COgnition-BAsed NETworkS (COBANETS) [60], inspired by the nervous system the based on advanced machine learning techniques, in particular unsupervised deep learning and probabilistic generative models, along with network optimization in combination with SDN technology. Knowledge Defined Networking (KDN) [61] is another initiative for cognitive-based management based on the pioneering work of D. Clark [62] that relies on Machine Learning (ML) and cognitive techniques to operate the network. In all of these projects, authors advocate the use of the centralized control offered by SDN, combined with a rich centralized view of the network provided by network analytics solutions. The KDN also focuses on the challenges posed by the NFV resource-allocation problem.

In this chapter, we will discuss the application of ML to for an automated network management. In the literature, multiple facets of the network management (i.e. FCAPS [63]) were tackled, we summarize the applied techniques in table B.

These constitutes the complementary pieces to a fully automated autonomic network. The principal contributions in the subsequent sections are :

1. We peresent a brief history of the application of AI and ML for network management.
2. We show how ML techniques have been used to realize cognitive management foreach of the FCAPS (Fault, Configuration, Accoun-

ting, Performance and Security) management domains.

3. We review the latest initiatives to realize self-managed networks, namely autonomic management, self-driving networks and COBANETS.
4. We examin the opportunities and challenges related to using ML for the management management.

For many years, the researchers have been working on multiple pieces of this puzzle, from predictive analysis of network performance [64, 65] to intrusion and anomaly detection algorithms [66, 67]. We notice that the state of the art contains different answers that lays the foundation for this ambitious goal. In this section we will start by a brief history of Machine Learning and Network management combined. Then we will review the literature of cognitive network management.

A Machine Learning for Network Management a Brief History

In the literature, we found that AI for network management is not a new thing[68, 69, 70, 71]. In the late 80's, when networks started growing in complexity [72], researchers proposed AI-based techniques to handle this growth. Up to our knowledge, the first project that tackled AI for network management is the IRMA-LAN[7] project, which aimed at assessing the impact of AI for network management in 1988[69]. The AI system managed a simulated LAN network using causal models and deep reasoning on the network configuration. Already, researchers pointed network management fields that can benefit from AI, namely, fault detection, fault diagnosis, fault recovery, fault prevention and fault prediction. In the late 80's, AI was seen as having practical value for managing the expected network growth for the next decade. Techniques that have been used are knowledge-based systems, heuristic search techniques and in some cases neural networks [73] K. D. Cebulka et al explored more in detail the potential of AI for network management. They classified the use of AI into three broad categories : Knowledge-based (expert) system techniques for provisioning management, heuristic search techniques(genetic algorithm) for network design and neural networks for performance and reliability management and

7. Intelligent Resource Management for Local Area Networks

fault detection (they pointed out that ANN can help anticipate problems in the network). The SLA management in this work was clearly exploded into the FCAPS. They pointed out the lack of general guidance for choosing ANN parameters Generally, we notice that early ML for network management focused on the use of Expert Systems (ESs) [68, 69, 70, 71] After that, a multitude of studies addressed network management using ML. We classified them into use cases that are considered as the building blocks for SLA Management. After this pioneering work, a multitude of research activities started to tackle network management issues using ML. Predominantly, the studies were on traffic/flow classification and security management. And most of the researches were based on k-means, decision trees and SVM.

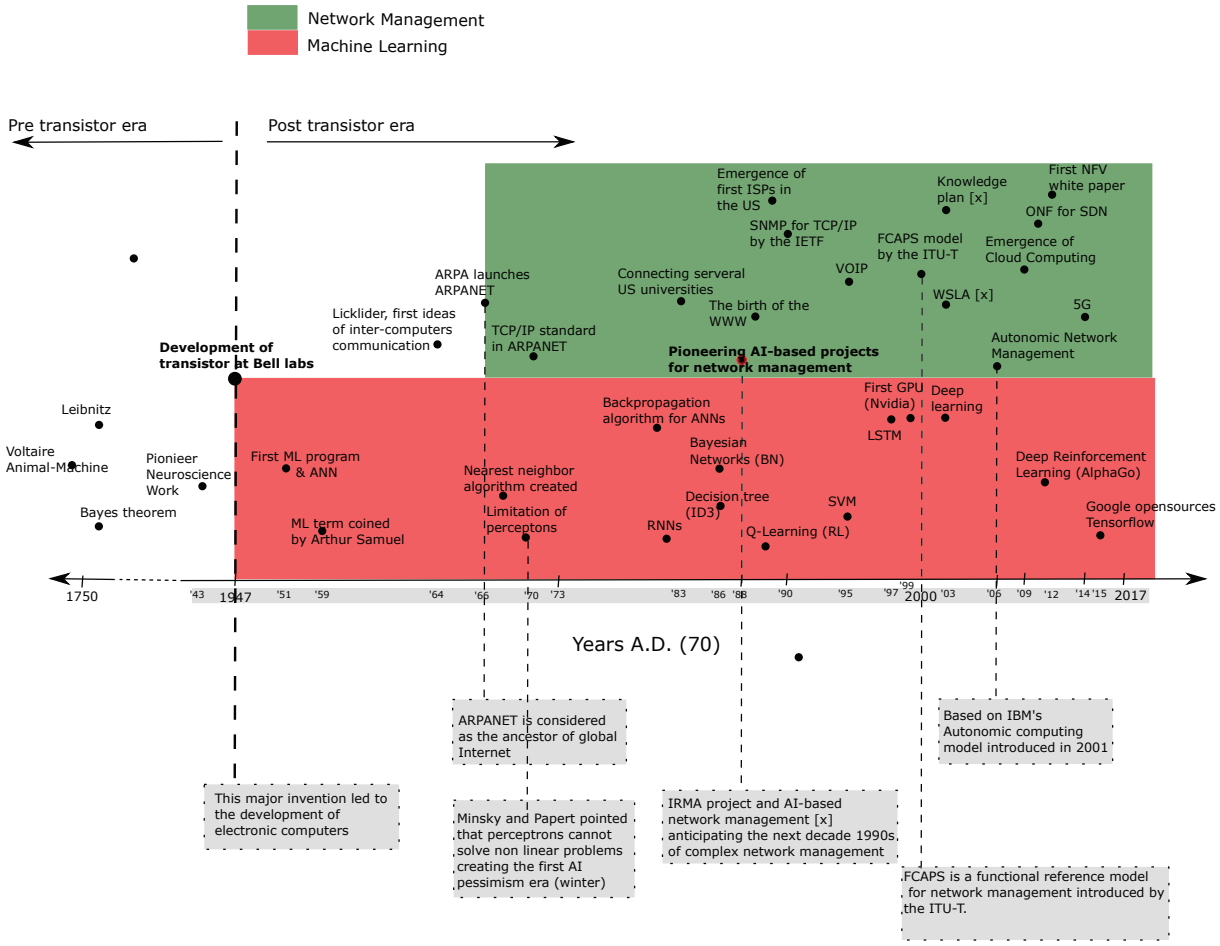


FIGURE 2.28 – The evolution of machine learning and Network Management

B FCAPS Management

Managing SLA targeting network performance involves reacting on short timescales to network changes. In order to provide acceptable performance for some services the management system should understand the relationship between the service level metrics and the system level metrics such as in [74]. SLA management in this context is particularly useful to determine the exact network state providing acceptable performance for some users and applications. The common objective for using ML in performance management is prediction of data traffic and throughput for proactive management [75, 76, 77]. In [76] authors used multiple algorithms for throughput prediction such as ANNs, SVMs and linear regression. Researchers tackling performance management also focuse on Quality-of-Experience (QoE). The authors in [78] estimated the evolution of QoE parameters for applications by monitoring network parameters. Similar concerns have been raised for Software-Defined Networking for Application Performance [79]. In NFV context, Chen Sun et al. [80] proposed and SLA-aware solution for NFV framework focused on the performance e.g. latency and throughput. Their solution comprises an accelerated data plane, SDPA (Software and hardware packet acceleration technologies), coupled with a performance-aware service chaining algorithm in the NFV orchestrator to fulfill both functionality and performance requirements with respect to SLAs. A common recurring theme in performance management is network traffic classification [81, 82]. Current ML method for traffic classification are based on supervized learning and unsupervized learning. K-means, ANNs, SVMs and other ML algorithms are used to cluster the network traffic application based on similarity between them. Moore et al. [83] use Bayesian algorithms with 248 features per-flow features for classification. They reached accuracy of 95%. In the literature, the majority of methods are based on supervised learning methods. Other works rely on unsupervised clustering instead, for classifying data flows without labels. For example, Hochst et al. [84] used autoencoders to cope with the varying delay and bandwidth mobile applications. They detected 7 different classes of mobile traffic flows. The traffic classification is imporant because the classified traffic can be then redirected for specific processing, e.g. browsing, download, livestream, etc. Recent work targe-

ting traffic classification in NFV discussed in [85]. The authors propose a NFV framework called vTC to dynamically instantiate and chain the constituting modules to realize adaptive learning approach in response to the dynamics in network flows.

In the literature, configuration management is usually approached using Reinforcement Learning (RL) approaches. Reinforcement Learning is a ML technique that is based on a Learning agent that acts on its environments and tries to optimize its decision based on a utility function. A pioneering work in 1994 tackling the problem of routing oprimization [86] using RL techniques. Manually changing the netwrok state is a cumbersome task, prone to error and implausible in large virtualized networks. Other researchs [50, 87] successfully used Q-learning a RL algorithm to autonomously reconfigured network systems for adaptative routing. Xu et al. [54] proposed a unified reinforcement learning approach to automate the configuration processes of VMs in the Cloud. RL have also been used for resource allocation management [51, 53] and for adaptive routing in SDN [52].

As the network became more complex, researchers turned their back from Expert Systems (ESs) towards statistical and ML methods. Ponting out that ESs can't handle new situations and changing data, can not learn from experience, and require extensive updates. From early 1990s, researchers argued that ANNs and Bayesian Networks (BN) are essential for network fault management []. ANNs are appropriate for pattern analysis and matching, their ability to approximate any continuous, nonlinear function and their ability to handle missing data, while BN can handle uncertainty and represent causality relationships. For this reason BN can be effective for automated fault diagnosis which is essential for Root Cause Analysis (RCA). Contrary to ANNs, BNs require expert knowledge and domain knowledge about the network topology and elements and faults and meta-knowledge, which is the description how to diagnose a network error[88]. We notice than ANNs for fault management in computer networks stemmed from previous researchs on fault detection in aerospace and chemical process [89]. Pioneering works using ANN for network fault management in early 1990s [90, 91, 92]. R. Patton et al. [90] used ANNs for alarm correlation. They used ANN to predict future system behavior. The ANN learned the normal system

behavior, when the system diverge from the ANN prediction a second ANN is triggered to analyse the residual signal and identify the anomaly category. Deng et al. [93] used BN to tackle fault diagnosis. SVM are used for fault diagnosis in distributed environment [94]. In the literature, we notice that BN are also used for Fault Diagnosis in SDN environments [95].

Security management approach based on ML techniques consists mainly of pattern detection of well-known observed threats. Thus, the main disadvantages of this approach is that it can not detect unseen patterns. To Answer this challenge, researchers turn to anomaly detection detection technique were algorithms focus on the nominal behavior of the system flagging as red any datapoint that deviates from the baseline. Recent years, there have been a significant advance in the application of ML to intrusion detection and anomaly detection. Many researchers have developed specific learning algorithms to anticipate and detect network attacks based on analysis of monitoring data. Examples are, DoS attacks detection using decision trees [96]. C. Sinclair et al. [66] were concerned by managing security, more specifically detecting network intrusion. They used decision trees and genetic algorithms to create "if - then -" rules for an intrusion detection system based on application name and port number. Deep learning technique also showed promising results for intrusion detection [67]. Authors justified the use Deep Learing algorithm because it can detect high level abstract features in encrypted traffic. A common obstacle in this aspect of network management is the difficulty for obtaining labeled data. Attacks are rare and dynamically changing, and often organization detaining these data are reluctent to sharable it. Existing network protocols and technologies were not design to provide labels with their monitoring data. Improving ML algorithms for network security necessitate improving the data quality fed to the ML algorithms. This is a relatively new concern in the literature and is considered as a major research question in the field of cognitive network management.

Management domain	Machine Learning algorithm
Fault	ANN, k-NN, k-Means, Decision Tree, SVM, Bayesian Networks, Rule-based reasoning
Configuration	RL, Q-learning, Deep RL
Accounting	-
Performance	ANN, SVM, Decision Trees
Security	DL, ANN, SVM, Decision Trees, Bayesian Networks

TABLE 2.7 – Machine Learning Algorithms applied to FCAPS management

C Cognitive Network Management Initiatives

State-of-the-Art Initiatives

The call for a self-managed network is not new. Researchers have proposed these approaches for decades. Perhaps the most noticeable are the Knowledge-Defined Networking [62] and Autonomic Management [1]. These early endeavors motivate the need for autonomy and self-management due to the increase in network complexity that will surpass human cognitive capacities. Moreover, the main cause of networking outages stem for human involvement[97]. Yet, today we notice that network management is still relying on scripts and manual tools for planification and orchestration.

However in the last years, we notice in the literature a revival of cognitive approach for Network Management in recent years [98, 60, 99, 58, 59]. All these initiatives revolve around the following motivations :

— recent advances in ML, for example the development of deep unsupervised learning networks to solve challenging classification problems.

— Improvement of Graphical Processing Units (GPUs), that are particularly useful for running very deep learning algorithms.

— the adoption and emergence of SDN and NFV paradigms,towards a more dynamic and flexible management of the network, making possible to execute network-wide optimization strategies.

In this section we will discussion the foundational work of Autonomic network management and present three of the recent initiatives, CogNet, self-driving networks and COBANETS.

Autonomic Network Management

In 2001, IBM introduced the concept of autonomic computing [100]. They proposed a Framework based on Self-X [8] functions inspired by previous work of Wooldridge and Jennings [101]. The reference model for achieving self-autonomy is the MAPE-K (Monitoring, Analyse, Plan, Execute, Knowledge) autonomic loop (see Figure 2.29).

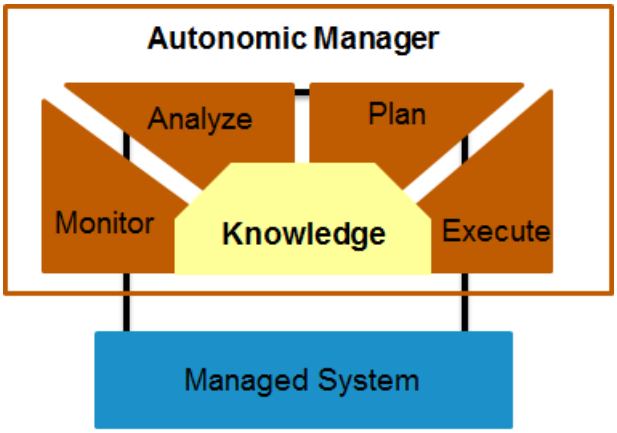


FIGURE 2.29 – IBM's MAPE-K (Monitor, Analyze, Plan, Execute, Knowledge) reference model for autonomic control loops

The IBM's model have inspired and laid the foundation in the networking fields, such as cognitive networks [1] and knowledge-driven networking [62]. fundamentally, these initiatives advocate for embedding intelligence and autonomy in network management.

The objective of autonomic networking is to automated the management decision. The decisions are based on a Policy-Based Management (PBM) module with a set of pre-defined self-X policies derived from human knowledge or from OSS/BSS policies. Here are for example the definition of self-configuration and self-optimization.

— **self-configuration :** "An autonomic computing system configures itself according to high-level goals, i.e. by specifying what is desired, not necessarily how to accomplish it [1]. This can mean being able to install and set itself up based on the needs of the platform and the user".

— **Self-optimization :** "An autonomic computing system optimizes its use of resources. It may decide to initiate a change to the system

8. self-configuration, self-optimization, self-healing and self-protecting

> Proactively (as opposed to reactive behavior) in an attempt to improve performance or quality of service [1]".

The role of ML in the autonomic network management is considered as a feature or set of techniques for constructing the Knowledge among other technics such as ontologies, semantic web, templates, policies, reasoning systems, etc.

We notice that at the time researchers pointed out the importance of evaluating the autonomous system not in performance but rather on its ability to meet a given SLA [102]. They stressed the challenge of how we can evaluate it with respect to a given SLA. Interestingly, IBMs reference model described Degrees of autonomicity from (support, core, autonomous to autonomic). In this last level of autonomicity the entity should take into account human goals described as SLA. Up to our knowledge, this is the first conceptual framework that introduces the concept of SLA with Cognitive behavior.

CogNet : CogNet is EU project in the context of 5G and softwarized networks. The goal of the CogNet project is to make a major contribution towards autonomic management of telecoms network infrastructure through using the available network data and applying Machine Learning algorithms to yield insights, recognise events and conditions and respond correctly to them. A fundamental aspect of the Project is to develop solutions that will provide a higher and more intelligent level of automated monitoring and management of networks and applications, improve operational efficiencies and facilitate the requirements of 5G. The project presents a high level architecture as a complement to the NFV reference architectural framework of European Telecommunications Standards Institute (ETSI).

The project defines six use cases in the 5G from the network management perspective, nameley, Just-in-Time services, Optimized Service in Dynamic Environments, User-Centric Services, SLA enforcement, Situational COntext, Collaborative resource management. The project defines the use of ML algorithms and processes for each of the use cases and challenges. In this PhD work, we contributed to the elaboration and implementation of the CogNet architecture and algorithms.

Self-Driving Networks

Inspired from the self-driving cars, the self-driving network initia-

tive [58] is an ambitious project that is set to regroup all previous ML , big data and telemetry work on network management to create a fully adaptive and predictive network. The initiative was launched by Juniper in early 2018, and aims clearly to ”eliminate the manual work required to keep networks running [103]”. The self-driving networks is based on five main technologies :

1. Telemetry : a system-wide real time monitoring technology
2. Multidimentional views : Heterogeneous data sources
3. Automation : language-based scripts (e.g. python, ruby) and configuration manager (e.g. ansible, puppet) orchestration and deployment
4. Declarative Intent : can be mapped to SLA or high-level policies
5. Machine learning for decision making and knowledge extraction

Among the technical challenges of the self-driving networks, for example, is to run a datacenter with 0-touch and no human intervention without any compromise in functionality. The vision of the self-driving netwroks on SLA management, is that it should be defined and automatically respected by the network using adaptation, reaction and anticipation principles.

At first glance, this vision of the future of networking viewed by juniper and shared by some researchers [59] has many common points with the other initiatives. If this vision is materialized, it will have disruptive consequences on the way networks are managed and on the evolution of human administrators and their skills. We notice that there is a consensus and an awareness to regroup ML for network management to a common vision with a map and a framework to achieve a fully automated network. The conclusion of ends with a recent call for papers [104] for a workshop at SIGCOMM conference.

Knowledge-Defined Networking

Based on the notorious paper of D. Clark et al. [62], The Knowledge-Defined Networking [99] is an effort to resurrect the former vision. The Knowledge-Defined Networking (KDN) is refered to as ”the paradigm resulting from combining SDN, telemetry, Network Analytics, and the Knowledge Plane” [99]. The KDN defines four planes in an SDN context. The data plane as a forwarding plane, the control plane that maintains

the network operational state, the management plane that handles provisioning and configuration of network equipments and the knowledge plane responsible for *its ability to integrate behavioral models and reasoning processes oriented to decision making into an SDN network.* .

KDN borrows ideas from the autonomic self-X functions, and thus relies on a theoratical framework based on control loop with the possibility of interaction with humans.

The KDN defines a set of specific use-cases : (1) Routing in an Overlay Network : traffic recommending policies, (2) Resource Management in an NFV scenario for optimal Virtual Machine (VM) placement, (3) Knowledge extraction from network logs and (4) Short and long-term network planning (based on the historical data stored in the analytics platform.).

COBANETS : COgnition-BAsed NETworkS

COBANETS ambition of realizing fully cognitive management [105]. COBANETS is a recent example that focuses on Deep Learning for Cognitive Network Management. The approach is inspired from the nervous system, it aims at creating a network-wide cognitive infrastructure composed with multiple cognitive nodes.

The main algorithms used in this solution are the unsupervised deep learning and probabilistic generative models that are suitable for massive unlabeled data. COBANETS relies massively on SDN to build a network-wide view and optimization. More specifically, the project aims at integrating of different generative models. Generative models represent how the data are being generated and can be used for imputing missing data and compression (dimensionality reduction). They give insights on how the data are generated/consist of ML models that generates all possible values for a given event. Generative models are used as an intermediate step to create rich and abstract network representations.

Current scientific and technical challenges for a fully cognitive Network

Despite the few presented research activities, the domain of deep learning for SLA and network management is still not formed yet. We believe that this is due to some technical and scientific challenges specific to network management. Firstly, system-wide cross layer data collection from different, heterogeneous, distributed sources and format.

Secondly, representing heterogenous data into a unified format that preserves the information. Thirdly, the large training time necessary for training a complex ML on Big Data. We have identified a research direction based on distributing ML computation in the network based on recent advances in large-scale deep learning models. In [44] researchers demonstrated that it is possible to devise better learning algorithms to train more accurate shallow feed-forward nets.

in dynamic networks. ML algorithms trains on a representative dataset, this means that the data represents the actual states of the network and the network errors or violations. However, in a dynamic, software-based network the network state is constantly changing affecting the data distribution. This in turn can mislead the training process and give rise to more and more errors in the classification and predictions. A typical solution in this case scenario is to monitor the ML performance and accuracy and to retrain it constantly.

training speed : As the netwotk state and condition is changing rapidly, the ML algorithm should keep up with the changing variables and network dynamics. This is typically done by implementing multiple ML algorithms and re-training them frequently. However, the one of the limitation of the stateofthe art networks is that they can take hours to days to complete training []. Fir this reason it is important to find an equilibrium between acceptable accuracy and rapid performance.

labeled data. In image and voice recognition, data labeling is a straightforward although a tedious task. These kind of tasks can be outsourced to human manual labeling, some services exists such as Amazon mechanical turk[106]. However, for network monitoring data the task is much more complex and requires often expensive high-skilled experts. Moreover, in dynamic networks many network states are unseen. These specific requirements stress the need for developing more sophisticated unsupervised ML algorithms. ML researchers also consider unsupervised learnind to be the next frontier in ML research with a potentially breakthrough [16].

Current ML models are often not designed to manipulate networking data, which is high-volume, distributed, and constantly changing. ML algorithms suffers from their inability to refine the input features used in a supervised learning to perform complex large-scale timeseries ana-

lysis. There is a necessity of identifying domain-specific deep models configuration for the telecommunication domain. Meanwhile, we notice that some existing ML algorithms fit well for some requirements such as RNN/LSTMs, Decision Trees and Bayesian Networks.

Interpretation, the ease to understand the model by experts. Among the most powerful ML algorithms are the ANN and more specifically the Deep ANN. These models are very powerful but they suffer from the lack of expressiveness and the ability for an ML practitioner to understand their output. These kinds of algorithms cannot be used for Root Cause Analysis. Bayesian Networks and Decision Trees on the other hand, are considered as white boxes and their decision can be understood for further analysis.

VII Conclusion

The increasing adoption of SDN/NFV in the networking community has transformed the infrastructure into more flexibility, agility and brought greater degrees of freedom. Driven by the potential of these evolutions and the recent success of ML, research initiatives have set as ambition to bring full autonomicity to network management.

We present in this chapter the major role that ML played in realizing cognitive network management in various management aspects and highlighted existing initiatives. We notice that even though there exists many challenges for ML for network management, the major technical building blocks are ready and should be regrouped under the MAPE-K control loop.

Chapitre 3

SLA management

The limits of my language are the limits of my world.

Ludwig Wittgenstein

Contents

I	**Context : Software Networks**		**68**
	A	Network Function Virtualization	68
	B	Software-Defined Networking	70
II	**Early SLA management**		**73**
III	**SLA in the Cloud**		**74**
IV	**SLA in Software Networks**		**77**
V	**Literature Gaps and Future Research Directions**		**80**

In this section we provide an overview of the SLA management in the literature. We decompose the SLA State-Of-The-Art into 3 subsections. First, (1) the Early SLA management. This comprises in the early IP networks, 3G, 4G Telecommunication networks, Service-Oriented Architecture, and the Grid. Secondly, (2) we present the literature work for the SLA in the Cloud and lastly (3) the SLA in software networks and 5G context. In table 3.2, we summarized the SLA evolution and requirements throughout this period.

The ITU-T defines SLA as a formal agreement between two or more entities that is reached after negotiation with the scope to assess service characteristics, responsibilities and priorities of every part [107]. SLA agreement generally comprises of parameters describing the service functional behavior and non-functional properties such as : the mi-

nimum acceptable QoS values (referred to as SLOs (Service Level Objectives) e.g. Maximum VNF instantiation time. The SLA incorporates the Business Level Policies i.e. monetary compensation, regulatory requirements, contractual conditions, billing conditions [107], the conditions for SLA renegotiation, and the procedures guarantees in case of SLA violations. In most instances, the SLA comes in machine-readable format (e.g. XML [108], OWL [109], JSON [110], YAML [111]) to facilitate the automation of the SLA negotiation process. Service Providers (SPs) are legally obliged to detect, and notify the Service Consumers (SCs) of any SLA violation, which can have considerable monetary impact.

I Context : Software Networks

This section reviews two major networking concepts : programmability and softwarization. The programmability is driven by the Software-Defined Networking (SDN), while softwarization is introduced by Network Functions Virtualization (NFV). These new technologies are disructing the networking indusrty and the way networks will be managed. SDN achieves programmability by decoupling the control plane from the data plane. On the other hand, NFV brings agility and and flexible placement of virtualized network functions across the network and the Cloud. SDN & NFV are considered to be complementary technologies for achieving full network programmability and flexibility [112, 113].

A Network Function Virtualization

Telecom and Network Operators traditionally rely on proprietary hardware equipments to deliver their services. To create a new network service they often rely on yet another variety of appliances and finding the right personal and space to setup these boxes. These type of operations are increasingly difficult to accomplished ; coumpounded by the rarety of skills necessary to integrate and operate these hardware. Moreover, the operators are locked-in to the devices' vendors for upgrades and maintenance. Whatsmore, hardware lifecycles are shortening with the advances in technology and innovation in services ; setting the operators back of new revenue streams [112].

Definition

Network Functions Virtualisation (NFV) aims to transform the way that network operators architect networks. NFV brings the evolution of IT virtualization technology to consolidate telecommunication appliances into the high industru standards. NFV addresses the implementation of network functions using software agnostic to the underlying hardware. In this work, we refer to this endeavor as "'Network Softwarization".

The NFV concept was introduced by the European Telecommunications Standards Institute (ETSI) Industry Specification Group (ISG). The ISG was started as a consortium of multiple vendors and network operators including Orange, AT&T, Deutsche Telekom and many others. The ISG work resulted in a white paper [112] introducing the concept of NFV. They described the core principle, the requirement and pose the conceptual framework for representing and managing the core elements called Virtual Network Function (VNF). Typically, a VNF can be a network address translation (NAT),

firewall, intrusion detection (IDS), domain name service (DNS), etc [112].

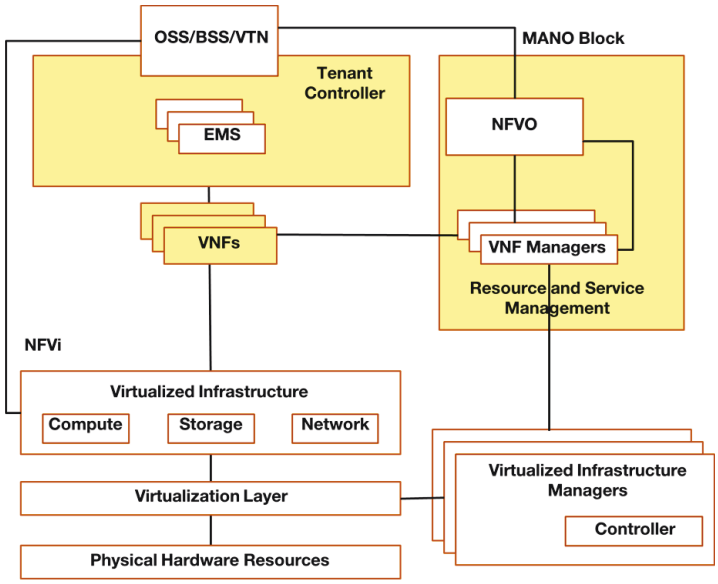


FIGURE 3.1 – NFV reference architecture

As presented in Figure 3.1 above, the NFV framework describes three main components :

— **Network Function Virtualization Infrastructure (NFVI) :** provides

the virtual resources to support the execution of VNFs. It includes hardware commodities such as x86, accelerator units and hypervisors.

— **Virtualized Network Function (VNF) :** A Software-based implementation of a network function deployable of the NFVI.

— **NFV MANagement and Orchestration (MANO) :** Cope with management tasks and on the orchestration and VNFs life-cycle management. The MANO also interacts with the Telco OSS/BSS solutions, which allows NFV integration with already existing network management system.

Recently many projects were combined under the umbrella of the Linux Foundation Networking (LFN). This entity aims at increasing the compliancy between the various networking projects hosted by the Linux Foundation.

An example targeting the NFVI is OPNFV. OPNFV [114] is a project started in 2012 with the goal of accelerating NFV adoption through system level integration, deployment and testing. Recently, they launched the OPNFV Verified Program (OVP) [115], a testing program for vendors and suppliers who are creating systems for the OPNFV reference platform.

Other projects target the orchestration layer. For instance, Open Network Automation Platform (ONAP) [1] aims to develop a policy-based orchestration of virtual network functions. It is also ETSI MANO compliant and includes further software subsystems, as well as integration for SDN controllers.

Other notable initiatives are, PDNA [116], which aims at developing an open source data analytics platform for network device and service telemetry in NFV. And DPDK [117], a software acceleration technology for improved packet handling.

B Software-Defined Networking

The second major pradigm shift in networking is network programmability achieved by SDN. The SDN decouples the network control from the forwarding layer. Figure 3.2. The Open Networking Foundation (ONF),

1. Open Source Mano project, https ://www.onap.org/

a leader in SDN standardization, describes the goal of SDN from the reference white paper [118] as follows :

Software-Defined Networking (SDN) is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today's applications. This architecture decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services

Consequently, this decoupling enables the control plane and the data plane to evolve independently from each other allowing high flexibility, programmability and fast service deployment [119]. Among the core problems SDN aims to tackle is "mananing the complexity of computer networks" [118, 120] by providing a centralized view and hides the low-level configuration details over a Network Operating System (NOS) which creates a network abstraction for the applications on top.

Figure 3.2 introduces the basic SDN components, with terminology similar to that from the original ONF white paper, "Software-Defined Networking : The New Norm for Networks" [118]. The initial architecture comprised of three main layers, namely, infrastructure, control and application layers (Main boxes), which are refered to as data, controller, and application planes. The infrastructure layer (data plane) comprises of network elements, which expose their capabilities to the control layer (SDN controller) via interfaces SouthBound from the controller. The SDN applications reside in the application layer, and exchage their network requirements toward the controller plane via northbound interfaces. In the middle, the SDN controller mapes the applications' requirements and dictates low-level control over the network elements, while providing relevant information up to the SDN applications.

The most common SounthBound API is the OpenFlow protocol. OpenFlow is considered as a standard communications interface between the control and forwarding layers of an SDN architecture. OpenFlow allows direct access to and manipulation of the forwarding plane of network devices, such as switches and routers. The ONF opted for OpenFlow to be a foundational element to the SDN solution.

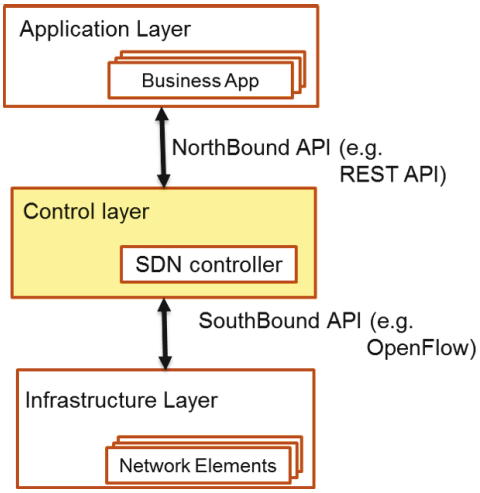


FIGURE 3.2 – SDN Architecture

SDN is considered as a complementary technology to NFV. They are expected to work together for achieving more flexibility. the SDN layer abstracts the network, creating a virtualized environment for NFV applications.

SDN has been widely appropriated in the industry, especially in datacenters e.g. [121, 122, 123] and stimulates the curiosity of the scientific community and paved the way to a series of innovations and open source projects (e.g. ODL [124], ONOS [125], FloodLight [126], RYU [127], etc.).

Finally, in table B we summarized the main differences between SDN and NFV.

Software Defined Networking (SDN)	Network Functions Virtualization (NFV)
Virtualization of the network infrastructure. Centralizes command and control of the network. Allows rapid provisioning of new networks without the complexity of physical upgrades.	Network functions that are fully virtualized, and free from dependencies on proprietary appliances and devices. Software-only in nature, and can be deployed on generic servers and SDN platforms.
Standards guided by the Open Networking Foundation (ONF). Initial standard is OpenFlow.	Standards guided by the ETSI NFV Working Group. Basic NFV architecture/use case has been specified by ETSI.
Located in the Cloud data center, and extends to secondary server farms in support of network operations.	Located in the service provider's SDN network.
Initial uses are orchestration, Cloud infrastructure & networking.	Initial uses are virtualized routers, firewalls, gateways, vEPC, etc.

Table 3.1 – Summary and comparison between NFV and SDN

II Early SLA management

Epoch : 1999 - 2008

SLA management in the literature used to be fragmented over FCAPS [2] management. Afterward, in late 90s with the creation of Information Technology (IT) departments, much effort was to regroup all these dimensions into one overarching domain : SLA management. SLA management captured more the attention of researchers due to the use of best-efforts networks to support new services with expected performance quality requirements such as multimedia and e-commerce.

SLA for telecommunication services gained in momentum in early 2008 due to the progress of Quality of Service (QoS) at the time and the emergence of dynamic monitoring and control systems of telecommunication resources [128]. As a result, SLA management concepts and system were introduced to deal with these issues. SLA attracted Telco providers as a mean to provide attractive offers while maintaining complex SLA promises. In the mobile context, 3GPP has defined four service classes based on delay requirements. This was the basis for SLA enforcement solutions. For example, in 2005 Y.Cheng et al. [129] have pre-

2. Fault, Configuration, Accountability, Performance and Security

sented SLA-Based Management approaches based on call level service differentiation.

SLA management was a great concern in Service Oriented Architecture (SOA) [130, 131, 132]. This was mainly due to the fact that web-service-based applications were designed to be used across companies ; commercial applications demanded service and quality assurance (e.g. availability, security) [131]. This resulted in several initiatives, among the most influential was IBM's WSLA [133] that was specially developed for SOA. There was many other attempts to represent the SLA using representation languages such XML, UML, internal SLA templates or using semantic web such as SLAng [9], and WS-Agreement [134] From Open Grid Forum (OGF), WS-Policy [135] from World Wide Web Consortium (W3C). The proposed SLA management solutions in this era focused mainly on static scheduling for SLA enforcement. WSLA formalization language had considerable success among the research community. It was the basis for the following SLA management in the Grid (2004) [133] and the Cloud [136].

III SLA in the Cloud

Epoch : 2009 - Today

In 2009, as more and more households gained access to the internet, Cloud computing gained enormous momentum in the computing field. Naturally, expectation concerns over QoS emerged between Cloud Service Providers (CSPs) and service consumers (SCs). Moreover, as the cloud was new to the public with many privacy concerns, CSPs needed to build trust to attract more SCs. SLA management offered the opportunity to formalize the expected service level delivered by the CSPs. Consequently, a plethora of initiatives, projects and studies emerged for addressing the new SLA requirements in the Cloud.

In the literature, active and advanced work has been conducted on SLA management for Cloud infrastructure. Emeakaroha et al. [137] classify the related work of SLA management into three categories, 1) resource monitoring [138], 2) SLA management including SLA violation prediction [139] and 3) mapping approaches from low level monitored metrics to SLA [74].

The first study tackling SLA in the CLoud was conducted by P. Patel et al. [136]. They proposed and adaptation of WSLA solution to the Cloud requirements. Authors noted that the main challenge in the Cloud is to maintain the service quality and reliability in a highly fluctuating demand/load environment. Aside from deliverying high performance, CSPs should be able to deliver their promises. Thus, the need to SLA management to define expectations/agreements end enforce them. Along these lines, most of the first SLA management in the Cloud considered the Cloud as an extension of SOA [136], hence applying the same SLA management concepts with some minor extensions. Moving forward, studies of SLA management in the Cloud detached from SOA and WSLA starting to see the short comings of the old approach in terms of static approach to negotiation and scheduling. Novel projects and initiatives started to develop novel dynamic conceptual frameworks more adapted to the Cloud elasticity [140]. Some examples are CSLA [141], Cloud4SOA(SLA*) [142] and SLALOM projects [143, 144] that we will describe more in detail. Common characteristics of these new SLA management approaches are their use of probabilistic models, dynamic models and biding strategies using game theory. Other phases of SLA lifecycle were also considered such as SLA specification ; SLA enforcement.

In [141], authors designed an SLA specification adapted for the cloud, named Cloud Service Level Agreement (CSLA), which introduces concepts of fuzziness and confidence to deal with the Cloud QoS uncertainty. Emeakaroha et al. [137] point out that in the cloud ecosystem, a key challenge is the adaptation of SLA monitoring strategies and timely detection of SLA violations. They present an architecture for detecting SLA violations in the cloud, named DeSVi. The motivation behind the development of DeSVi is the provisioning of services based on SLA. The DeSVi architecture leverages the LoM2HiS framework to identify SLA violations at run-time. DESVi runs at the actuator level in managing VMs configurations and deploying system components. A reverse process is suggested in [74] where the authors use a Neural Network to map Application level requirement to resource attributes and characteristics. The authors identify a specific challenge for SLA in the cloud, which is translating Consumer-PaaS SLA to PaaS-IaaS level SLA. They imple-

ment their solution using an ANN with one hidden layer of 13 nodes. The evaluation of their system demonstrates great accuracy while respecting their pre-defined thresholds with respect to each time frame in service oriented infrastructure. Hani et al. [145] raise the issue of predicting violations of SLA. They define SLA violation as deviations from the conditions agreed on in the SLA. They use a Support Vector Machine (SVM) adapted for regression, termed SVR, to forecast future values of time series. They identify two SLOs, namely, throughput and response time in cloud database. The final evaluation shows a minimum accuracy of more than 80% for 10 days look ahead. Authors of [146] state that the cloud service providers tend to maximize their profit by overbooking their resources with user applications. The authors note that an arbitrary overbooking ratio may degenerate into SLA violation and cost penalties for CSP especially with real-time application such as online video streaming. To optimize the resource utilization and reduce the risk of SLA violations the authors introduced iOverbook framework which uses an Artificial Neural Network to find correlation in the historical data and predict future resource usage. They used two Feed-Forward Neuronal Networks (FFNN) with one hidden layer of 23 and 22 neurons. The first ANN predicts the resource usage whereas the second predicts the performance. In their approach they use a trivial SLA definition which is an SLO on performance in IPC (Instruction per Seconds). Moreover, the system of SLA checking is not completely automated. The evaluation shows that iOverbook can allow optimization in Overbooking and allow a power saving up to 32%. Among the major SLA-centric european project is SLALOM [144]. The SLALOM (Service Level Agreement Legal and Open Model) EU H2020 research program aims to define practical approach of Cloud SLA management based on ISO standards. SLALOM provides two SLA reference models for cloud computing consisting of legal clauses and technical SLA specification. The SLALOM contributors performed a questionnaire [147] on the importance of the *"Proactive SLA Violation detection"*. They define proactiveness as any mechanism that protects the cloud application from failure or service degradation. The questionnaire was performed on Cloud Service Providers (CSP), Cloud Adopter. The questionnaire was based on the SLALOM Handout [148]. We find that these early poll results shapes the new awarness concer-

ning the new ways SLA should be managed. The results are depicted in Figure 3.3.

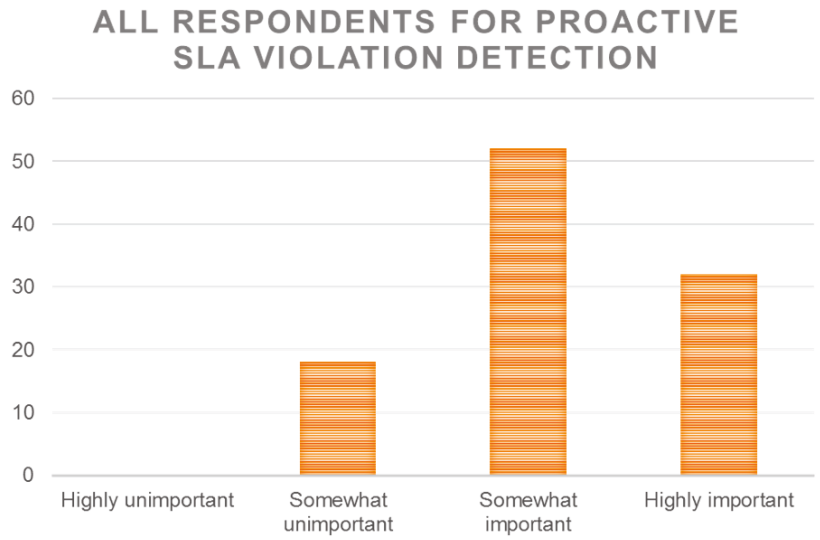


Figure 3.3 – Overall feedback on the importance of the "Proactive SLA violation detection" research area [3].

IV SLA in Software Networks

Epoch 2012 - Today

Research activity on SLA management for softwarized and programmable networks is still in its infancy. Up to our knowledge there is no effort to manage SLA in NFV-based environments per se. Research activities are more concerned by improving VNFs performance than SLA. However, we notice many studies working on different aspect of SLA from QoS, FCAPS [63] management to anomaly detection. In this section, we will give a quick overview of these "silo" studies and classifying them into : NFV monitoring, QoS Management, FCAPS Management and Anomaly Detection. We then elaborate on how we see them merging and gaining in maturing to form a proper SLA Management for SDN/NFV. In this regard, we consider SLA management as an umbrella term that encompasses multiple network management facets. Successfully managing SLA entails to succeed in network and service supervision, QoS management, FCAPS management as well as anomaly de-

tection. For this reason, we focus this PhD towards SLA management. NFV/SDN monitoring :

Proper NFV monitoring is the starting point for a complete SLA Management solution. G. Gardis et al. in the context of the FP7 T-NOVA European project (2013) [3] [149, 150] presents a VIM monitoring framework for NFV supervision. They focused on the process of collecting metrics from the NFVI and process them at the VIM level. They pointed out that current Monitoring tools can only partially fulfill NFV requirements. When the VNFO is managing hundreds of NFVIs, the VIM should integrate an intelligent agent that filters unnecessary monitoring data to avoid flooding the Orchestrator. And that the monitoring solutions should enable proactive and intelligent and self-learning approaches. The T-NOVA solution aims at monitoring, configuring and automate the provisioning of VNF-as-a-Service (VNFaaS). Their solution allows SC to associate SLA to a VNF. Their VIM monitoring solution is set to interact with OpenDaylight Statistic API and Openstack Ceilometer for low-level metrics, while monitoring the VNFs via a VNF-based agent using collectd-core module (Linux module for statistics).

QoS Management. The challenge in QoS management for NFV networks is that it should consider - beside the resource level- the QoS through the chaining of the VNFs. T. Kim [151] formalized this problem as multi-constraint path selection and proposed a genetic algorithm to ensure QoS while creating a service chain. R. Mijumbi et al. [152], authors adopted a novel approach for dynamic resource management in a NFV context. The authors proposed an extended recurrent neural network based on graphs called Graph Neural Network (GNN) that relies on VNF-FG (Forwarding Graph) to predict the evolution of resource requirements. They yield an average accuracy of 90%. The authors also based their experimentations on the open source Clearwater framework. O. Chaignon et al. [153] discussed the disruptive monitoring capabilities brought by SDN stateless control plane. They emphasized the scalability challenges brought by central management point of control. Solutions to this problem have been proposed in the literature in several forms and approaches. We cite here, local resource optimization, state-

3. T-NOVA will design and implement a management/orchestration platform for the automated provision, configuration, monitoring and optimization of Network Functions-as-a-Service (NFaaS).

less network-wide resource optimization based on greedy algorithms. V. Riccobene et al. [154] raised the concern of SLA compliancy with respect to efficient resource allocation in NFV. They presented an automated deployment framework for VNF deployment in two folds : (1) the characterization of VNF workload using and (2) the mapping between VNF performance and resources. For (1) they used Openstack Heat template as VNFD (VNF Descriptor) and for (2) used Decision Tree algorithm to select an Openstack VM flavor according to the network requirements and SLA. R. Mijumbi et al. [152], authors adopted a novel approach for dynamic resource management in a NFV context. The authors proposed an extended recurrent neural network based on graphs called Graph Neural Network (GNN) that relies on VNF-FG (Forwarding Graph) to predict the evolution of resource requirements. They yield an average accuracy of 90%.

FCAPS Management. Perhaps the most challenging FCAPS aspect in NFV is the Performance one. Migrating network functions that used to run on dedicated hardware to cloud servers ultimately impacts their performance. The research community and industry are very concerned by the VNFs performance [155]. In a recent study [80], Chen Sun et al. proposed and SLA-aware solution for NFV framework focused on the performance e.g. latency and throughput. Their solution comprises an accelerated data plane, SDPA (Software and hardware packet acceleration technologies), coupled with a performance-aware service chaining algorithm in the NFV orchestrator to fulfill both functionality and performance requirements with respect to SLAs. In the literature, solutions targeting security in programmable networks usually leverage the centralized role of SDN controller [156, 157]. A notable work has been done by M. Miyazawa et al [158] to detect faults in NFV networks. They proposed a distributed VNF to manage the NFV network, called vNMF. They pointed out the challenge for fault detection in NFV due the separation of roles between the VIM that manages the physical resources and the VNFM that manages the VNFs. Additionnaly they stressed the need for Machine learning algorithms to learn rapidly to adapt to this dynamic environment. They construct a VNF fault detection model based on the assumption that VNF failures are linked to the underlying structure, i.e. NFVI. We think that this assumption might not hold for

all the VNFs and all the cases.

Domain/Era	Main issue	Research direction(s)
IP networks	how best effort could insure QoS ?	Adoption of the SLA Concept, regrouping FCAPS under SLA. Manual
Telecommunication	How to differentiate ?	Diffserv networks
SOA	integration. was to ensure performance, and integrating the SLA contract to the SLA management system : bridging the gap between Sla spec and enforcement (performed by personel).	WSLA, XML, semantic web
Grid	handling complexity (geographically distributed domains) and heterogeneity (of services).	WSLA
Cloud	maintain trust and managing fluctuation (load, elasticity)	WSLA, CSLA, biding strategies, probabilistic models
NFV	main issue : Performance, reliability management. secondary issues : managing fluctuation	AI, ML, automation, Cognet, Abstractions (model, templates)
SDN	Reliability management (coherence in the control plane), and skill set	Abstractions (model, templates) , central view
5G	Very high performance expectation (Throughput, Latency) Heterogeneity management Better QoE	Cognitive & autonomic Management

TABLE 3.2 – summary of SLA evolution in the literature

V Literature Gaps and Future Research Directions

The main limitation in the presented related work is that SLA and SLO are not specifically targeted toward novel use case of NFV and SDN. Furthermore, in general the discussed studies defined SLOs in very simple term which is not realistic in operational settings. Although, many related works [159, 160] proposed ANN for forecasting in an SLA context, there is clearly a lack of a well-defined Framework for SDN and NFV, for complete automation of SLA management that combines all

the necessary blocks of cognitive management and proactive provisioning. Moreover, to the best of our knowledge, LSTM-based approaches have never been used for SLA Management for Programmable Networks (i.e., Cloud computing, NFV/SDN). A foreseeable opportunity of LSTM in Programmable Network context is that, contrary to FFNN, it can process data with different input size, allowing a versatility of usage for processing data with different sampling rate across different channels.

Chapitre 4

Proposal : Cognitive SLA Management Framework

The wise is one only. It is unwilling and willing to be called by the name of Zeus.

Heraclides Ponticus

Contents

I Introduction 84
II Problem Statement 87
A Service Level Agreement 89
B SLA and SDN/NFV 90
C Formal Description 92
D SLA Example 93
III Cognitive SLA Architecture 95
A Cognet Smart Engine : 98
B Policy Engine 101
C NFV Architectural Framework 102
D Proposed workflow 103
E Policy Engine 107
F Cognet Sequence Diagram 108
G Operational Application & Use Cases 109
IV Data Analysis 124
A Data Gathering 124
B Data preparation 126
C Dimensionality reduction 129
D Visualization 138
E SLA Assurance Services 141
V Conclusion 143

In this chapter presents the cognitive framework for SLA violation anticipaton in Programmable Networks. In this approach, we aim to combine multiple Machine Learning and data preprocessing methods for optimal violation prediction. We present the our two use cases and the end-to-end methodology to manage them.

I Introduction

SLA (Service Level Agreement) Enforcement will have a predominent position in the value proposition of 5G. Service providers can offer higher SLA gurantee as a way to different themselfves from competitors. Certain services may be required by law to use higher SLA such as emergency communication and military communications.

In the telecoms realm, administrators refer to garrier grade offerings as a highly reliable and available service (i.e. 99.999%). However, in the Cloud and modern IT communication the services have less gurantee and requirement than the carrier grade, their were designed to provide besteffort communications. The SLA in the context of softwarization and virtualization is paramount, for it allows to track service providers' promises in a highly fluctuating environments.

For example, real time voice communication traffic is sensitive to lantency and jitter, a momentary drop of these metrics heavily impacts the perceived Quality of Experience (QoE) by the users. Other service, such as web-based applications are less. SLA allows to resolve ressource contention between applications and services such as emergency services. SLA can also target security and authentication requiering specific encryption techniques.

In this chapter, the first focus is on the problem definition, SLA description and the generation of the data. We generate the dataset by implementing two use cases. A streaming use case and a multimedia NFV-based use case. We designed an end-to-end framework, namely *CogSLA* where we focused on problematics such as data collection, preprocessing and cognitive processing.

We discuss then the results of the preprocessing and the cognitive processing in two broad categories, prediction and classification.

In the last section, we focus on improving the process of ML selection and fine tuning their hyperparameters. We introduce then the concept of MetaLeanring as a technique to recursively use ML algotithms to extrapolate and guide the selection of the optimal algorithm with respect to precision and training time.

Analytical approaches	**Descriptive**	**Predictive**	**Prescriptive**
NFV mgmt	Clear restitution of the service current state	Anticipating SLA violation	Automatically managing NFV for optimal performance
Challenges	Subject to penalties	Subject to service degradation	Subject to misconfiguration
Opportunitues	Defining clear opportunities and challenges	Inputs for decison systems/ policies systems	Optimization of costs and QoE states

Table 4.1 – Three approaches of Analytical approaches. [Inspired by Dursun Dellan, Decision Support Systems, Elsevier 2012]

Although, the research community pushes towards a predictive and prescriptive analytics-based network management (see Table 4.1). Current management approaches and tools are mainly decriptive in their approaches, i.e. reactive, threshold-based. In this case, the most used strategy to avoid SLA violation is overprovisioning. Overprovisioning is allocating more resources to a service than needed. This strategy is effective for avoiding SLA penalties, but severly limits the Service Provider (SP) in its growth for acquiring new customers and markets. Moreover, this results in a under-utilized resources that can be allocated elsewhere where most needed.

What is more, Cloud environments are composed of a huge amount of managed elements that produces large amounts of monitoring data and alarms from disparate sources. A human operator can hardly process and correlate between all the alarms, the visible data and infere a root cause or a suitable management action. In this context, there is a need of a straight-to-the-point analysis of network and service behavior.

In this chapter, we will go through all the cognitive process implemented as the cognitive framework.

We will see the preprocessing steps

1. **the preprocessing steps :** Initial steps to clean and normalize data.
2. **Feature engineering (PCA and expert-based) :** From the system-level metrics to the service/application level metrics.
3. **Forecasting :** How ML algorithms can be designed to help to anticipate SLO violations.
4. **Classification :** How different ML algorithms can be trained to identify SLO violations when they occur.
5. **Actuation using REST API :** How to interpret ML outputs into an actionable network manahement task.
6. **Hyperparameter problem and selection :** How to tune the ML algorithms to have the best performance and accuracy.
7. **Meta-learning :** How ML can help on the previous point.
8. **How Visualization can help :** Data visualization to grasp and use the human intuition to understand the studied problems in higher dimensions.

Machine Learning Motivation

The design and itegration of a cognitive framework for managing NFV-based deployments is able to provide the following benefits to network management :

— Instantaneous service deployment : One of the biggest opportunities of Virtualization is that it allows to spawn and remove ressource remotely and rapidely

— Demand/offer Optimization : A machine learning based management can forecast and anticipate the demand to optimally allocate ressources.

— Reduce CAPEX : ML based system will reduce the need to purchase new hardware ressource. the only costs incurred is related to the allocation of virtualized infrastructure.

— Rapid time to market with acceptable risks : the NFVI can grow in and shrink size as soon as the system detects the change is ressource need resources.

The NFVI introduce a context of uncertainty and high resource load variation , rapidly changing network patterns, thus hampering the ability of a network administrator to manage them in real time. It is therefore necessary to introduce automated and intelligent systems with the ability to make anticipate and identify the most accurate action on the fly.

PhD Blueprint :

This work has gain in maturity by going through different stages. The first stage is the design of the global cognitive architecture were all the main components are defined to enable autonomic management principles for NFV/SDN-based networks. Afterwards, we zoom on multiple components of the architecture that tackle data analysis tasks such as dimentionality reduction and data filtering. Next, we zoom on the Cognitive Smart Engine were we combine multiple ML algorithms to anticipate SLA violations. Finally, we focus on mechanisms to select the most suitable ML algorithm and optimize its parameters.

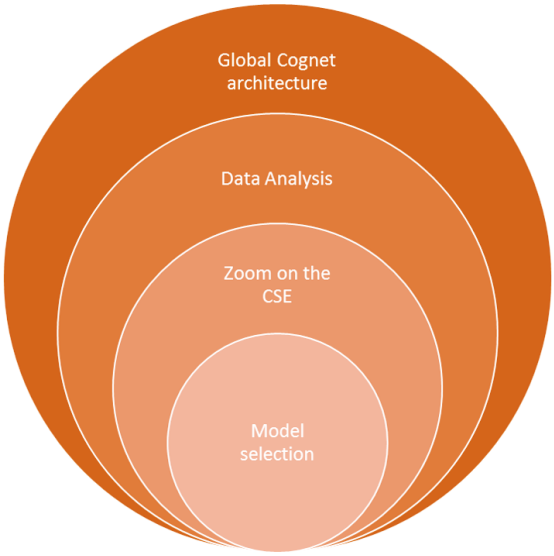


Figure 4.1 – PhD Blueprint.

II Problem Statement

A key challenge in the 5^{th} generation networks is to improve even more the users' Quality of Experience (QoE). Continually growing users' expectations in highly dynamic and heterogenous networks necessitate

the definition and commitment of clear and precise Service Level Agreement (SLA)s. SLA enforcement should be able to map high SLA directives into low-level configuration in the network while monitoring continiously the state of the SLA. We believe that network programmability and centralization brought by Software Defined Networking (SDN) are key concepts to tackle this point.

SLA management in programmable network is necessary to deliver on the expectation of the 5G high quality of experience. Whatsmore, the radical network transformation with the rise of IoT and massive connectivity. This exposes SLA even more to new vulnerabilites

The emergence of network softwarization driven by Network Function Virtualization (NFV), makes it more challenging to represent and enfore SLA. Although, the ETSI NFV specifies SLA descriptor as a Virtual Network Function (VNF) Descriptor (VNFD) for a single VNF[]. Requirements on SLA management are similar to SLA for the Cloud with a stress on performance and reliability for NFV. The main challenge brought up by softwarization of networks is that the placement and configuration of VNFs impacts the network performance itself. This adds an additional layer of complexity to classical SLA management in the cloud, were previous work on VMs migration didn't consider the overall performance. This also raises monitoring capabilities in NFV networks were the NFVI and the VNF are conceptually decoupled but physicaly highly interdependent.

These new requirements for SLA management in programmable networks calls for a more intelligent and data-driven approach. Data-driven approach by extracting meaningful information from heterogenous data source. Intelligent by forecasting and anticipation SLA violations.

Machine Learning (ML) comes as a natural alternative to classical solution because of :

- Recent success and revival of ML, specially deep learning (i.e. representation learning)
- Proliferation of multiple data sources and heterogenous, massive amout of data (i.e. IoT, 5G use cases)
- Analytical platforms and Big Data solutions
- Programmability and centralized network-wide view technology (SDN)

- Many distributed monitoring solution across layers
- Less constraints and more degrees of freedom with respect to management action brought by the virtualization, i.e. Dynamicity of execution with NFV

The recent success of ML techniques especially deep learning and the proliferation of new, heterogeneous data, analytical platforms, distributed monitoring solution across layers and big data solutions along with the rise of SDN/NFV technologies.

A Service Level Agreement

The network operators deliver networking services to their users. In return, the consumers pay for the product with the expectation of good quality and networking services performing to their specifications.

The networking product are highly flexible and customized to meet individual consumers need. Beside its functional part, the networking servic is expected to have technical properties such as performance, reliability, availability, etc. Those properties are refered to as the non-functional properties or the Service Level Objectives (SLO). The SLO targets Key Performance Indicators (KPI) or Key Quality Indicators (KQI) as shown in figure 4.2. The specification of a given service is written as multiple SLOs targeting different non-functional service properties. This specification is what we call Service Level Agreement (SLA). SLA describes other specification between the service consumer and service provider such as penalties for not respecting SLOs. Hence SLA is at the core of the relationship between the operators and the consumers.

The SLA management topic is of fundamental importance for operators that want to increase the consumers overall satisfactions. In our study, SLA is intresting because it touches different network management areas and is transverse to the FCAPS -Fault, Configuration, Accounting, Performance, Security. Working on SLA management will autoatically propagate to all these network management subfields.

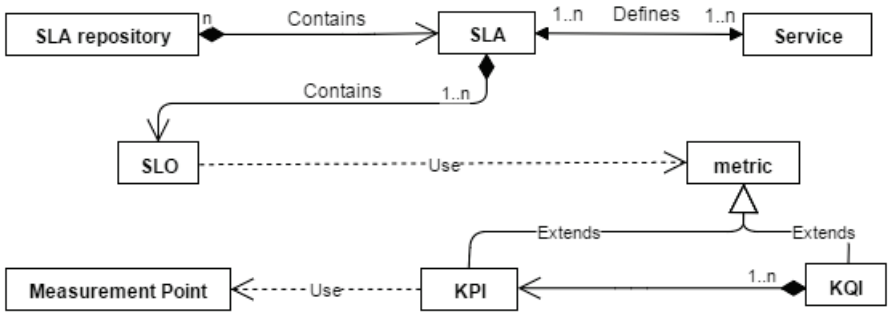


Figure 4.2 – Simplified UML diagram of SLA.

The SLA is described in machine readable format. In this study, we propose an SLA descriptor in JSON and YAML (Yet Another Markup Language) as follows : The main directories : metrics, SLOs and service descriptor. Only the necessary metrics to define SLOs are stored in the metrics directory. We show an example of SLO description in listing 4.1 as JSON.

B SLA and SDN/NFV

In the NFV context and accordying to the ETSI standards, the SLA definition are stated in Descriptor files in YAML.

The NFV-MANO Description files are an ongoing work at the ETSI-working group. The file descriptors are organized as service catalog and element catalog. The service catalog contains file descriptors that targets parameters at the service level whereas, The element catalog is concerned by the low level description of the Framework such as VMs and link descriptors. We see the SLA as playing a role at each level of these desctiption files as depicted in figure 4.5 in the MANO block. At each stage, different SLOs targets different specification in the NFV framework. All of these are important and should be managed as a global SLA, translating and mapping high level SLA into multiple sub SLOs of different aspects of the framework.

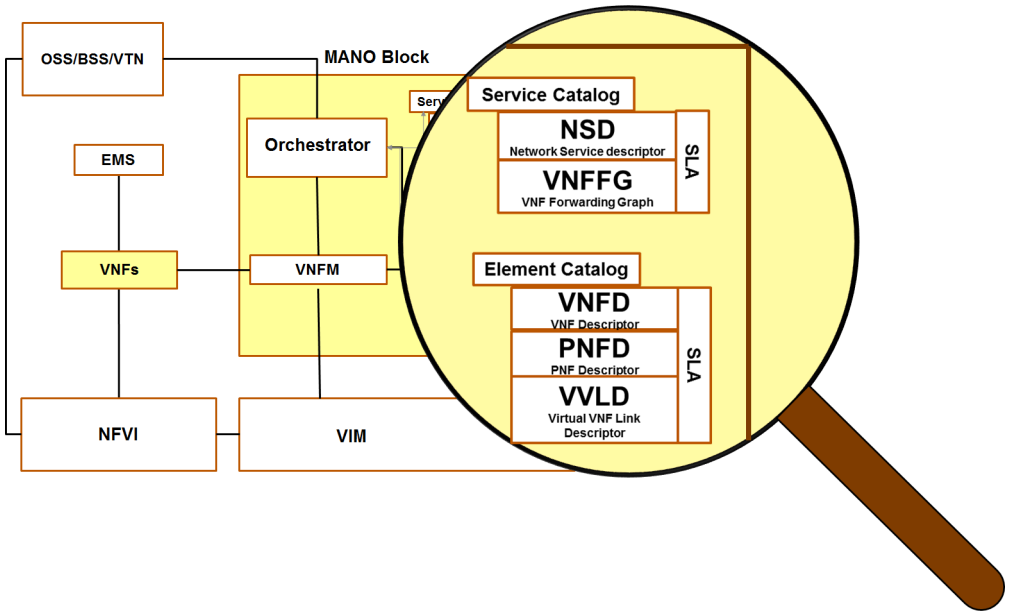


FIGURE 4.3 – SLA NFV description In ETSI framework

In the service catalog :

NSD - Network Service descriptor : This aims to describe the end-to-end service description including SLOs, covered VNF-FG, VNF as well as list of supported service monitoring parameters.

VNF Forwarding Graph descriptor - VNFFGD : This describes the VNFFG, VNFs and VNFDs and their needed for orchestration, reference to link information and description of physical/logical interfaces.

In the element catalog, the most important file descriptors fors SLA are the VNF/PNF Descriptor. They describe the functional behavior of elements and their non-functional behaviors, namely the SLOs :

VNF Descriptor - VNFD : The VNFD provide Links to scripts for initiation and termination, description of internal and external connectivity, dependencies between VNFCs. Moreover, it targets SLOs as VM specification, i.e. required storage and computation resources, initiation and termination scripts, high avaliability redundancy model, scale out/in limits.

PNF Descriptor - PNFD : The PNFD references to link information, exposed external interfaces, PNF addresses, PNF status, systems subscribed for notifications as well as SLOs on throughput and internal latency.

Discussion.

The SLA management in the NFV context requires special attention to the mapping between different levels, from the service to the application and element specification. In the ETSI specification the way to perform the mapping is not explicitly specified and seems to be designed for a manual allocation. This procedure is highly unlikely for that virtualized services should be highly flexibly and automatically adapt new elements to the global SLA. For this reason a hierarchical definition of SLA is necessary in this context. Machine Learning algorithms are particularly useful for these kind of tasks.

In the case of NFV the most important SLOs are those targeting performance and reliability. For this reason, we focused the SLOs on the response time, throughput and transactions.

C Formal Description

In the following, we proceed to the mathematical formalization of SLA and its SLOs.

Let M_i be the observable variables for $VNFC_i$:
$M_i = m_{i1}, m_{i2}, m_{i3}, ..., m_{in}$, where m_{ij} is the metric j of the VNFC i. Moreover, we have the VNF as be the machines $V\vec{N}F = VNFC_1, VNFC_2, VNFC_3, ..., VNFC$

The SLO space can take two values : $S_i = \{S^+, S^-\}$ for n SLOs,

$$\Gamma(V_{state}) = \bigvee_i \gamma(S_i),\ S_i =< V\vec{N}F, SLO_i > \tag{4.1}$$

,where $\exists \vec{M^*} \in M$, such as :

The equation 4.1 defines the SLA violation as the violation of at least one SLO.

$MAX_{(k,\vec{M^*})}[corr(\vec{M^*}, SLOV_i) - \alpha]$

k as M^* elements and $k \leq n$ and α is the correlation threshold.

Working on the relationship between the metrics $\vec{M}$ and the SLO state S. in order to compute the relationship between the observed metrics and the SLO state, we can write the conditional distribution $P(S^+ \mid \vec{M})$: The probability of observing an SLO violation given the observable set of metrics. The bayes Theorem below gives us an insight into the probability as follows :

$$P(S^+ \mid \vec{M}) = \frac{P(\vec{M} \mid S^+)\,P(S^+)}{P(\vec{M})}$$

$$P(S^+ \mid \vec{M}) = \frac{P(\vec{M} \mid S^+)\,P(S^+)}{P(\vec{M} \mid S^+)\,P(S^+) + P(\vec{M} \mid S^-)\,P(S^-)}$$

, where $P(\vec{M} \mid S^+)$ is is the probability of an SLO violation given a set of metrics M and $P(\vec{M})$ represents xxxx and $P(S^+)$ is the probability of the violation occurring.

D SLA Example

In this work, we refer to an SLO as a range of values (i.e. lower or upper thresholds) that guarantee a certain level of quality with respect to a specific service and to a specific set of variables (or aggregates, i.e. mean value or percentiles).

SLA Violation (SLAV) occurs when at least one SLO related to the SLA is broken or breached. The SLA violation prediction is when our system correctly identifies that an SLO or a set of SLOs will no longer be compliant.

To have a realistic SLOs definition, each SLO is defined as a combination of at least two metrics and thresholds. The metrics are defined as average over a certain period of time. For example, SLO_1 is the combination of response time with respect to the workload. As shown in figure 4.4, SLO_1 is defined as a multi-step function. For each workload interval, a specific threshold is set on the response time : If the workload is between 0 and 20% the minimum response time is 20 *ms*. If the workload is between 20% and 70% then the threshold is set at 50 *ms*.

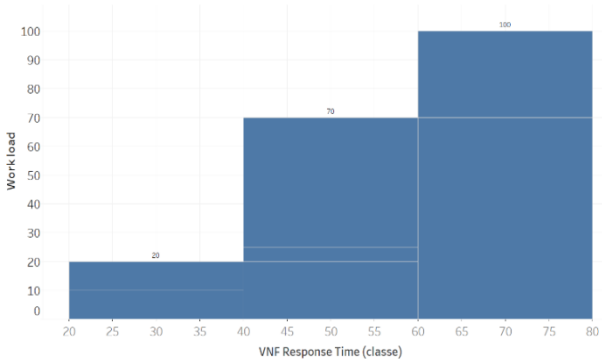


FIGURE 4.4 – SLO step function of response time.

Based on the elaborated SLOs definition, we used a machine readable SLA description using YAML. YAML (Yet Another Markup Language) is a data-oriented markup language in plain text. YAML is human friendly, easy to read and edit. We opted for YAML to easily represent SLOs for network services and use it as an input to our system in a dictionary format, i.e. key, value tuple. YAML has also the advantage of being interoperable with other serialization languages such as XML and JSON. Figure 2 illustrates an example of YAML description of SLOs contained in the SLO description file. Alongside that, we used other directories and files including, metrics, profiles and service descriptor.

The 3 SLOs targets respectively, the response time of the service, The availability of the proxy and the database transaction. The role of the ANN is to identify from these observations the relevant metrics to watch for each SLO and to identify effectively when an SLO breach is occurring.

As aforementioned, SLA agreement comprises of a set of measurable, low-level objectives, i.e. SLOs that are contracted with the service consumer. In this section we formalize the three SLOs that will be used for the Clearwater use case.

— SLO_1 *response time* : This service objective targets the response time of the SIP proxy VNFC (i.e. Bono). The overload tests generate a large amount of connections forcing the proxy to drop multiple connection requests. This in turn, reduces the mean time necessary to answer a connection request.

— SLO_2 *database transaction* : This SLO targets mainly Ralf and Homestead node. The SLO in this context is defined as the Cassandra

database performance. We observed the behavior of the database and labeled the data accordingly.

— SLO_3 *service availabiltiy* : This SLO monitors the behavior of the service as a whole. When the communication service is down we label the corresponding dummy variable with 1, 0 otherwise.

```
{
  "nameSLO": "$SLO_1$",
  "QoE": "Premium",
  "isAlive": true,
  "SLAid": 25,
  "dhcp": on,
  "variables": {
    "cpu": "0.6",
    "memory": "0.5",
    "disk": "0.2",
    "networkIN": "10021-3100"
  },
  "callMethod": [
    {
      "type": "AVG",
      "number": "0.5"
    },
    {
      "timeSTART": "1518431604",
      "timeEND": "1518451604"
    }
  ]
}
```

Listing 4.1 – SLO Description in JSON

III Cognitive SLA Architecture

In this section, we present in detail the SLA management framework, namely *CogSLA*. CogSLA is a framework that describes the process by

which operators comply with SLA agreements contracted with consumers and other parties using knowledge computed via ML techniques supported by a monitoring systems. CogSLA has been design to be system and architecture agnostic. Building on this frameworks we contribted to create a global cognitive management architecture (figure 4.5) in the Cognet project [98]. We define Cognitive SLA enforcement as to the process by which operators comply with SLA agreements contracted with consumers and other parties using knowledge computed via machine learning techniques supported by monitoring systems. CogSLA is extendable, as it is intended to cover a diverse group of services beside PNs, such as IoT, and unknown 5G services i.e. we seek extendable languages and templates that, once tuned, enable the establishment of the required agreement levels of different and heterogeneous services.

In figure 4.5, we show how the generic framework is supporting cognition. We first consider as a reference the ETSI NFV architecture which positions the SDN controller within the NFV framework.

Two main challenges were tackled with this framework :

1. The sheer amount of data pushed by the monitoring system.

2. The heterogeneity of data and data sources, i.e. timeseries, SLA descriptions

The Cognet architecture presented in figure 4.5 is designed for an NFV/SDN based environment for Autonomic 5G Network Management using Machine Learning. The architecture design is based on the ETSI NFV framework. The objective of this architecture is to develop a scalable, high performing real-time network management platform that accesses multiple data sources, enabling autonomic infrastructure managment.

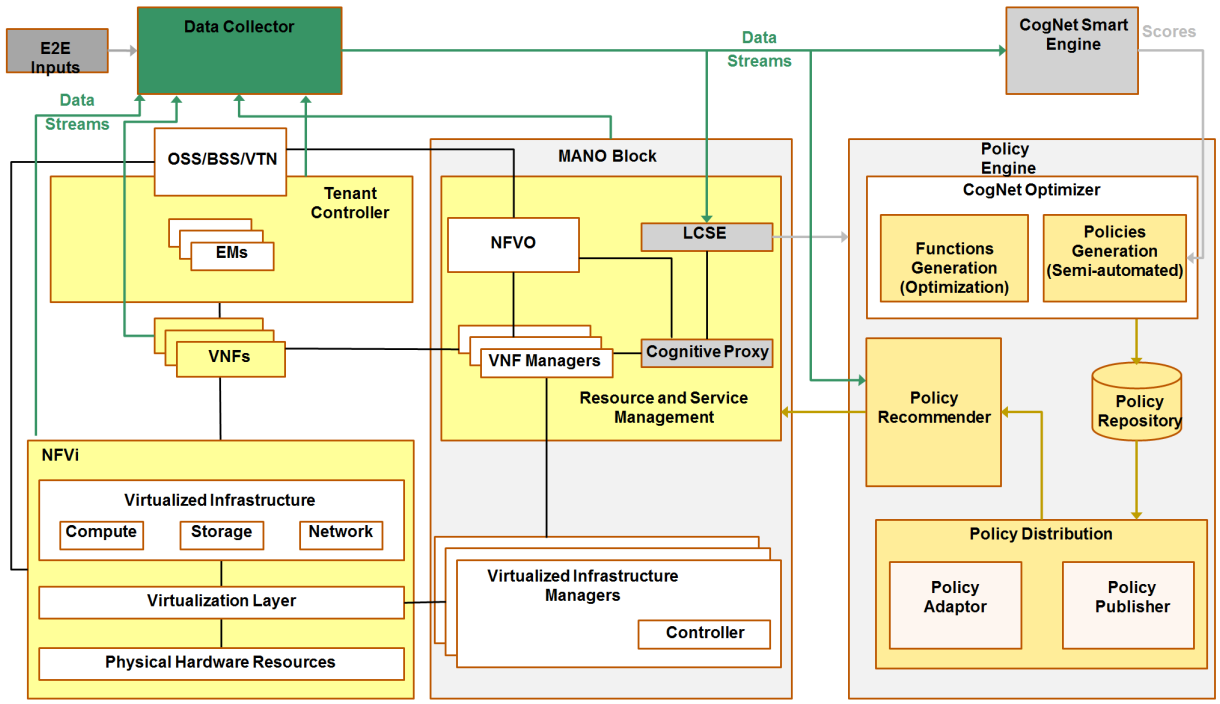


FIGURE 4.5 – Cognet architecture

We will procede in the description of all the three key components of the architecture from an SLA-centric perspective :

1. *CogNet Smart Engine (CSE) :* responsible for receiving the state and resource consumption records, pre-processing the records, selecting suitable algorithms, and then applying selected models to further process the received data. The CSE is enhanced by a Batch Engine (BE) that processes data in batches, and by a (Near) Real-time Engine (NRE) that processes data in lower latency manner. The CSE supports various machine learning modules that in turn help deliver different data analysis. These include SLA violation anticipation, SLA violation and simple action recommendation service for the policy engine.
2. *Policy Engine :* mainly responsible for mapping insights from the CSE into appropriate policy actions that can be directly understood by related components in the Management and Orchestration functions. In our implementation the policy engine actions are statically

defined.

3. *NFV Architectural Framework :* leverages the ETSI NFV architecture. One key architectural innovation of CogNet is the adoption of network intelligence to NFV MANO. Specifically, the MANO stack is enhanced by the CSE, a processing component that offers similar functionality as CSE but is only equipped with the (Near) Real-time Processing Engine (NRE). Such a component is designed to be embedded into MANO, located as close to data as possible to reduce access and processing time.

A Cognet Smart Engine :

The CSE, depicted in Figure 4.6, is responsible for receiving the state and resource consumption records, pre-processing the records, selecting suitable algorithms, and then applying selected models to further process the received data. The objective of the CSE is to support the various Machine Learning modules that in turn contribute to the delivery of various data service solutions, such as data gathering service, forecasting and prediction services, anomalies and fault recognition service and action recommendation service for the policy engine . These services in turn have associated policies in the Policy Repository of the Policy Engine. The input of the CSE will be a data stream on the relevant events whilst its output will be scored on the states of given components in the architecture. The CSE collects data from both resource provider-side and consumer-side. This is intended to increase the openness and transparency of services delivered by 5G networks, and subsequently provide better user experience. Moreover, the output of the CSE are prediction scores, which can be divided into :

1. Thresholds for specific policies, such as the maximum CPU utilisation before reaching performance degradations and

2. Metric data at timestamp t, such as predicting the : (i) %CPU (numerical), or (ii) presence of an anomaly or fault (categorical).

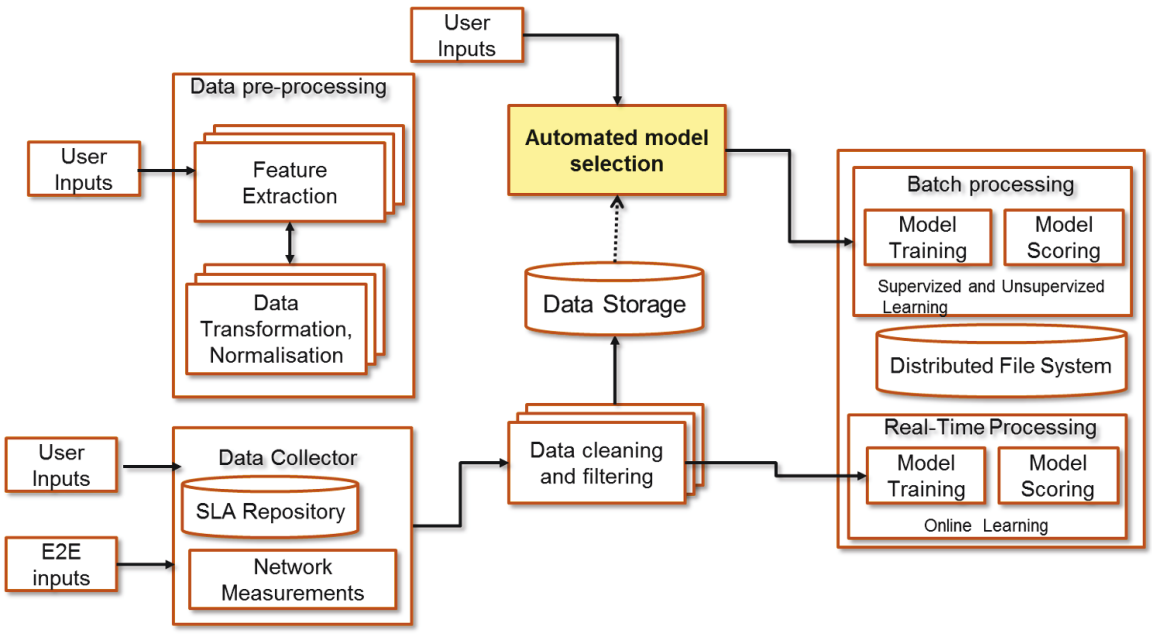


FIGURE 4.6 – Cognet Smart Engine

The CSE consists of following sub-components :

— *Data Collection & Adaptors* – it collects data from multiple resources, namely, from SLA repository (from user inputs Figure 4.6) and from system-level monitoring metrics (i.e. cpu, network, disk and memory). Then, maps collected data into those that can be processed directly by the following components.

— *Data Storage* – it stores historical data and SLA data as key-values pairs, and makes them available for multiple components constituting the CSE.

— *Data Cleaning & Filtering* – it cleans and combines the heterogeneous data (i.e. SLA and metrics), and then stores it into the Data Storage or forwards it to the CSE.

— *Data Pre-processing* – it can work in either automatic or manual mode to pre-process collected data stored in the Data Storage and make them ready for the Processing Engine. Feature extraction can be achieved by Deep Neural Networks, which will allow to generate highly abstract features automatically. Such functionality is essential to support the overall flexibility of the architecture and to keep it adjustable to constantly changing environment. It controls

the noise and reduces the processing time of analytic works for the CSE.

— *Algorithm Selection* – similar to the Data Pre-processing, this component is able to identify the best Artificial Neural Network models based on multiple customized metrics(more on that in section III). The model selected will be deployed into the CSE.

— *Batch Processing Engine* – The main componenet of the CSE. it retrieves consumption and state data from the data Storage, and applies these data to train a model or generate scores. In the former case, the Batch Processing Engine will evaluate the distortion of current model. If the model has become stale or no model is available, it will generate a new model from scratch to facilitate the work of the Real-time Processing Engine. In the latter case, this engine works independently to analyse collected records in a more accurate but higher latency manner. Note that the scoring in both the Batch Processing Engine and Real-time Processing Engine is not to simply apply one machine learning model but may involve a sequence of models associated with post-processing. For example, to detect network anomaly, we may need to score a number of records and then make a conclusion based on a linear combination of generated scores.

— *Distributed File System* – it stores models generated by the Batch Processing Engine that will be deployed on the (Near) Real-time Processing Engine. Note that this component is optional since the Batch Process Engine may forward generated models directly, such as through message queues/RESTful Web Services or the two processing engines may not shared data between each other without writing it to an external storage system if they are implemented and deployed in some cluster computing systems, such as Apache Spark.

The detailed discussion of the CSE is in section II. Note that the two last building blocks, the distributed File System and Real-time Processing Engine are out of the scope of this work and thus will not be covered.

B Policy Engine

This component is mainly responsible for mapping the output from the CSE into the policy actions that can be directly understood by the related components in the MANO stack, Tenant Controller and OSS/BSS. It consists of the following sub-components :

— *Policy Recommender* – The Recommender matches events that represent system state of interests in the context of business objectives and their operational realization with the policies in the repository, to determine which policy is relevant. The events are received from the CSE and can be in the form of (i) predictions such as resource utilization (e.g., expected CPU/RAM consumption) or anomalies detected (e.g., network intruder alert, or performance degradation detection), or (ii) thresholds optimized for specific monitoring values, such as considering the dynamic environment a CPU consumption exceeding 80% might cause degradations at a point in time t, however with the changes in the dynamic environment this amount might vary at a $t+1$. The Policy Recommender is a decision point that takes into account the state of individual network elements, but also helps to achieve business objectives by suggesting how to manage the overall resources required for network services. By integrating the predicted results from CSE, the Policy Recommender can make policy decision dynamically to respond to situational context, as well as changes to network environment due to ongoing operations. In addition to the above, optionally the recommender can feed the repository with adapted/new policies. The Recommender can be extended to adapt/recommend new policies based on the experience gathered from applying previously existing policies. Hence, it will be analyzing historical data of the effect of existing policies, by looking at performance indicators, such as delay, throughput, degradations caused, resource consumption patterns.

— *Policy Repository* – it stores policies related to all components of 5G networks. These policies normally are bindings of policy events with policy actions. They contain parameters and data structure that the Policy Recommender needs to evaluate for policy recom-

mendations. The policies should be consistent with the predefined SLA.

— *Optimizer* – The Optimizer consumes what is in the repository, and is responsible for dealing with the fine tuning of the parameters/-condition. The Policy Optimizer receives policy decision from the Policy Recommender, and then transforms the abstract actions specified in selected polices into concrete ones based on the state and configuration information from the MANO stack. This information can be static, such as source or destination addresses or dynamic such as current available network resources.

— *Policy Distribution* – The Policy Distribution invokes APIs offered by the components that are hosted in the MANO stack, Tenant Controller and OSS/BSS/VTN based on specified actions. It recommends actions according to the decision of Policy Recommender and current network conditions.

The detail implementation of the policy engine by ML algorithms is out of scope and is consideration for future work.

C NFV Architectural Framework

The NFV Architectural Framework of CogNet leverages the ETSI NFV architecture. One key architectural innovation of CogNet is the adoption of network intelligence to NFV MANO. Specifically, the MANO stack is enhanced by :

— *Light CSE (LCSE)* – it is a processing component that offers the similar functionality as CSE but is only equipped the (Near) Real-time Processing Engine. Such a component is designed to be embedded into the MANO that is located as close to data as possible so that the data can remain local. In such a way, access and processing latency can be minimized.

— *Proxy* – it forwards the concrete actions from the Policy Engine to related components constituting the MANO stack, but also converts the actions into a format that can be consumed by these components directly. The components connected to Proxy including NFVO, VMF and VIM are equipped with built-in policy enforcement mecha-

nisms, which are able to adjust or re-configure those network elements managed by them. The CogNet NFV architectural framework supports two types of connectivity services, both of which require control capabilities and the orchestration and management of different types of resources for building and accomplishing a proper delivery as NFV uses the network at two layers. The first is the SDN controller that deals with the network services provided at the service tenant layer, in particular with the operation and management of the network service's constituent VNFs by instructing the various VNFs that are deployed on the NFVi to take different actions on the traffic. Moreover, the second SDN controller supported by the CogNet architecture is the one in the infrastructure domain, which supports the setup of the required connectivity (including the WAN) for the communication of the deployed VNFs.

D Proposed workflow

Gathered data from the Data collector is pre-processed in Data Cleaning and Data Pre-Processing. This aims to clean them and extract relevant features that forms the basis of a low generalization error model. Afterwards, the Automated Model Selection component searches and then finds out the machine learning models that can offer the best performances based on processed feature sets. The workflow of the data pre-processing is depicted in Figure 4.7, details will be introduced as follows.

Pre-processing

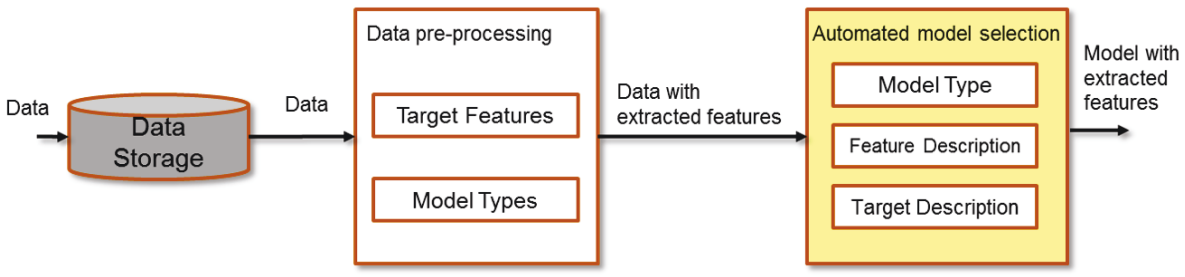


Figure 4.7 – Data Pre-processing

The Data Pre-processing flow is illustrated in Figure 4.7. Initially, collected records are cleaned by some light-weight approaches that aims to reduce the size of data for storage. The feature extraction may also be performed on the cleaned records based on the request from the Data Pre-processing if the records will be forwarded to the Real-time Processing Engine. Afterwards, processed records are stored in the Data storage or consumed directly by the (Near) Real-time Processing Engine. The Data Pre-processing will further process the stored data by normalisation and extraction. The former operation refers to adjust values measured on different scales to a notionally common scale, which is essential for certain machine learning algorithms, such as classifiers that calculate the distance between two points by the Euclidean, and can potentially facilitate convergence of given machine learning approaches, such as gradient descent. The latter one covers methods that transform raw data into informative features for machine learning algorithms. In the 5G era, hundreds or, in some cases, tens of thousands of input features are available in network management services, such as network traffic classifications and network-wide monitoring. Data extraction is intended to increase the accuracy of machine learning models by extracting salient features from the raw input data but also potentially remove noise and redundancy from monitoring records. It also simplifies and facilitates model selections since if relevant features can be extracted, even a simple model can offer remarkable results. Additional aims include lowering the dimension of data to facilitate training speed and visualization. Following the work on pre-processing, the Automated Model Selection component checks hardware resources available to processing engine(s), and then evaluates the performance of machine learning models based on the configuration on a specific job requested by a user . Instead of comparing all available models recursively, the work can be processed based on certain selection criterion, such as **??**. The model offering the desired performance will be deployed on selected processing engine(s).

Data Processing

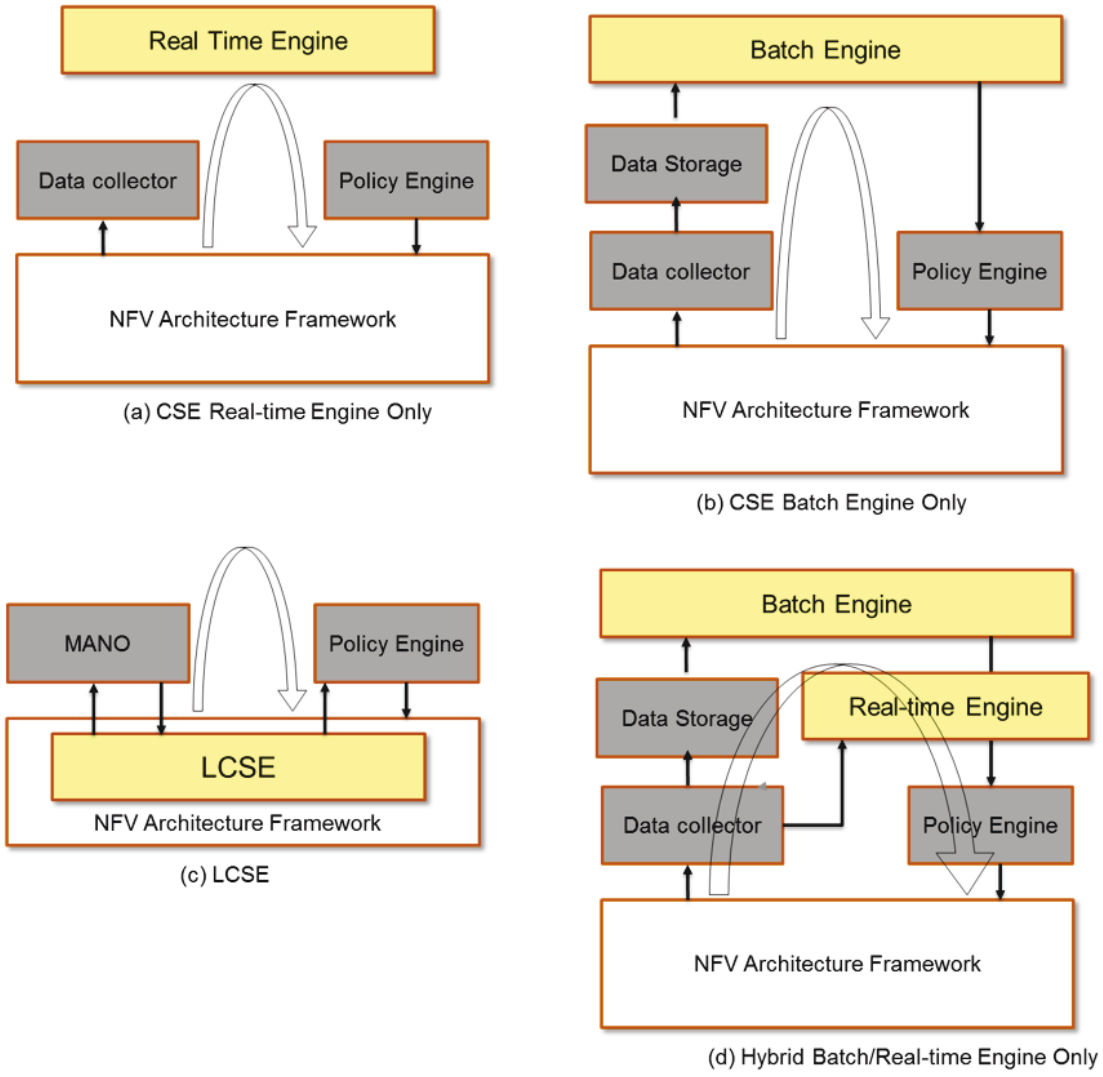


FIGURE 4.8 – Processing Engines.

As introduced in Section A and C, LCSE and CSE are two processing units, which will train machine learning models and then apply them to analyse gathered records . They will be evaluated and selected based on certain criteria. As depicted in Figure 4.8(c), if LCSE is assigned as the processing component in a given task, which is only equipped with a (Near) Real-time Processing Engine, then it collects monitoring information directly from a Data collector component hosted in the MANO stack since the MANO has the knowledge on all components constituting the NFV architecture. This forms the basis for low latency analysis.

The output of the LCSE will be consumed by the Policy Engine. If the CSE is assigned the processing task, both the Batch and (Near) Real-time Engine may be requested to cooperate with each other or work individually. Figure 4.8 (a)(b)(d) depict how these engines work in different modes. In the case, only the (Near) Real-time Processing Engine is activated, data are pre-processed and models are selected and deployed on this engine. The (Near) Real-time Processing Engine then operates based on its configuration, and its analysis results are forwarded to the Policy Engine. In the case, the Batch Processing Engine works independently, it consumes models from Automated Model Selection and data from Data Storage instead of Data collector directly. Its outputs will also support the operation of Policy Engine. In the case, both engines are selected, the Batch Processing Engine trains selected models and then forwards these models directly or indirectly to the (Near) Real-time Processing Engine. The models will be applied to generate analysis results by the (Near) Real-time Processing Engine. In practice, the historical telco network and environment data may contain sufficient knowledge for network management. However, the scale of these data, in the order of petabytes, can prohibit the learning of predictive models in a timely manner. The hybrid mode unifies both batch and real-time processing. It enables the CogNet architecture to process massive amounts of data in order to build predictive models based on previous network behaviour. It also brings the ability to process fresh network data for predicting customer or network behaviours based on the historical models within a short period of time.

E Policy Engine

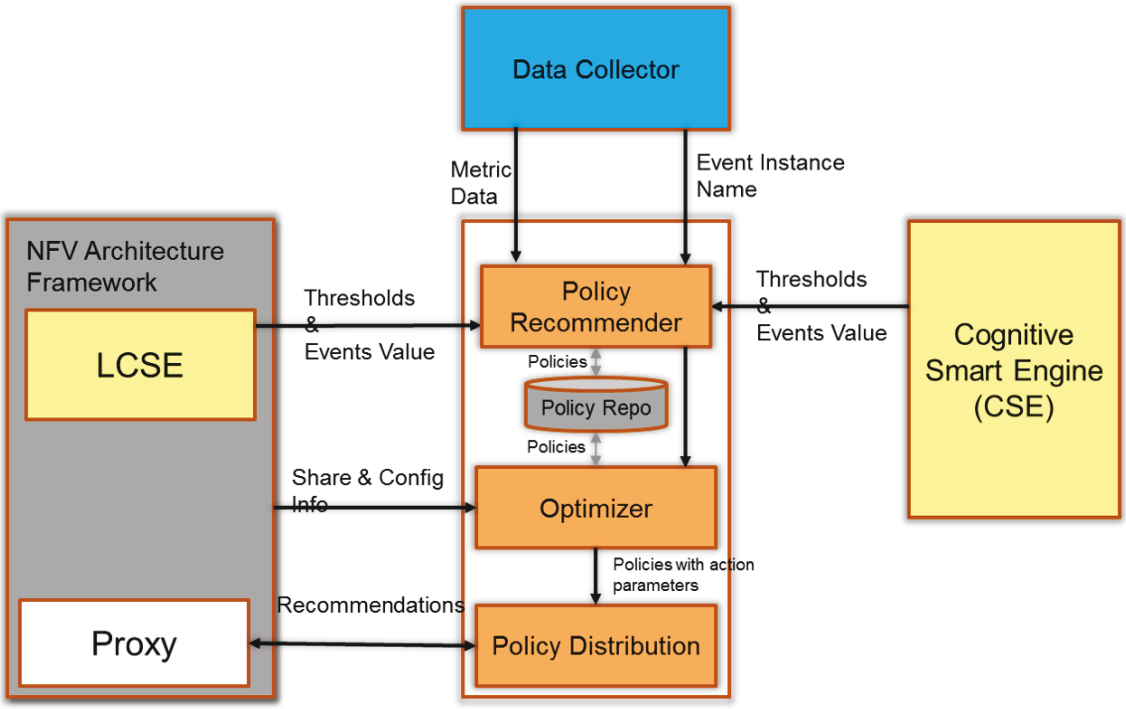


Figure 4.9 – Processing Engines.

The CSE or LCSE sends its outputs, which can be predicted values on specific events, such as the CPU usage of a given server, to the Policy Engine. As illustrated in Figure 4.9, upon the inputs, the Policy Recommender evaluates conditions of all policies given in the Policy Repository and then identifies the policies that will be triggered. Afterwards, the selected policies are sent to the Optimizer that maps the high-level action specifications of the polices into concrete ones based on the state information from the NFV architecture. The concrete actions will be forwarded to the Policy Distribution and then the Proxy to recommend further to the MANO/SDN controller how to adjust resource provisioning in order to avoid the violation of network management rules.

The policy engine includes the engine itself, which selects from the policy repository the most appropriate set of actions based on the event and on the system conditions. Additionally, the policy engine is able to distribute actions to the different components. The policy engine may be extended to have other responsibilities (conflict resolution, suppressing of policies, optimization of the policy parameters, etc.); however,

currently these (including policy generation/adaption) are an extension of the current scope of the work. The policy engine includes a policy repository where the policies are stored. The policy engine uses the stored policies to determine the appropriate actions. The policy recommender has currently the role to retrieve the violated policies and will aim to adapt the existing policies in the policy repository with new ones specific or customized for the specific deployment. This could be the addition of new policies or the adaptation of the parameters in the policies (e.g. threshold levels). The policy recommendation could be based on the direct monitoring of the events, on the history/status of the system and especially on the machine learning insight (which provides dynamic statistic results on specific events). The cognitive processing flow includes monitoring, CSE, policy engine, distribution of actions. When the scenario is not enabled by machine learning, the processing flow skips the CSE component. The experience accumulation processing flow includes monitoring, CSE, Policy Recommender, Policy Repository (with new or adapted policies).

F Cognet Sequence Diagram

In Figure 4.10 below, the NFV monitored system is pushing repeatedly metrics to the Data Collector through the Inbound API. On the meantime, the [L]CSE is continuously getting these metrics to find a target situation. Once it detects an event it is transferred to the Policy Engine which gets a list of potential policies to be applied from the Policy Recommender and check their conditions. In the case a policy condition is satisfied, the Policy Optimizer find the appropriate empty fields, if any, and delegates to the Policy Distributor to send to the specific endpoint (VIM or SDN controller) the action through the Outbound API.

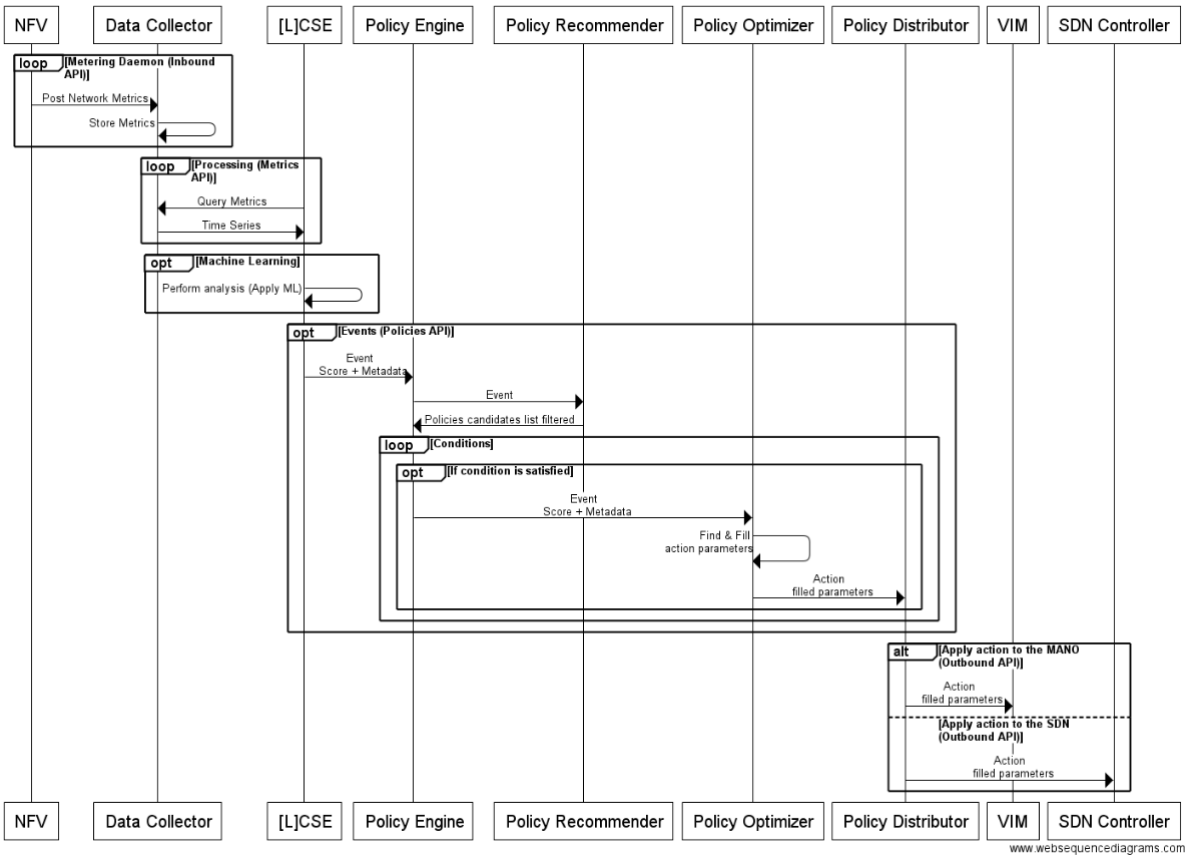


FIGURE 4.10 – Cognet global architecture sequence diagram.

G Operational Application & Use Cases

We introduce in section III the theoretical framework that allows an autonomic and a generic cognitive approach to managing NFV-based networks. Based on these global principles, we generate a simplified/-derived framework targeting specifically SLA management, presented in Figure 4.11.

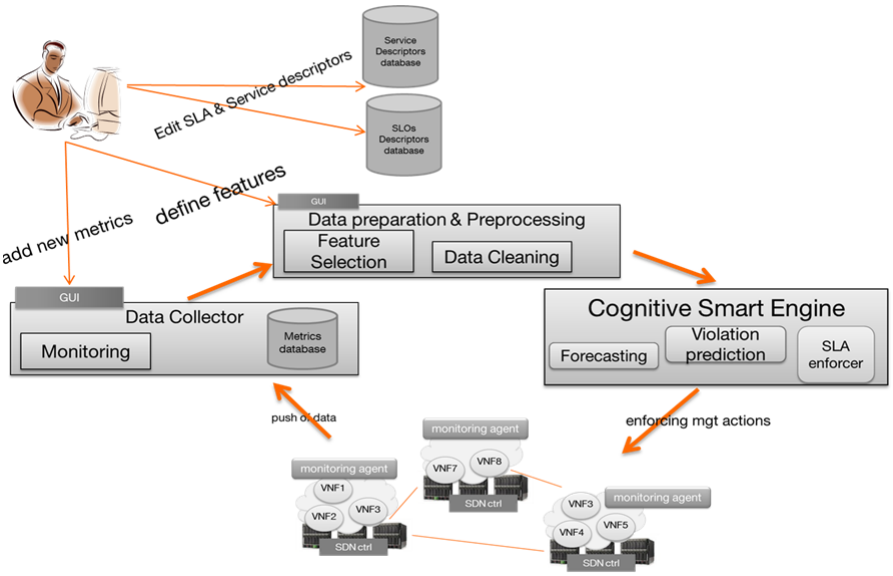


FIGURE 4.11 – Simplified Cognitive SLA Architecture.

We reused some of the main building blocks depicted in Figure [**?**]. In this section, we descibe the specifity of each block with respect to SLA management and give pratical and more detailed application of each block.

Data Collector : The data collector gets data from two sources, (1) the system-level supervision tools and (2) the SLA repository (described in section D). It collects raw unprocessed metrics (e.g. cpu, disk, network) from the monitored service of the running VMs, VNFs and virtual switches. The data collector stores the monitoring metrics in a time series database and SLA metrics in SQL database. The aim of this module is to provide ready-to-use data to the Data Preparation and Preprocessing module.

Data Preparation and Pre-processing : One of the most crucial steps in the CogSLA framework. It performs two types of transformation on the data. First, cleaning/filtering, reducing the data dimentions. Secondly, transforming the inputs into a comprehensible format for the ML algorithms. In this second phase we also incorporate feature engineering process.

Cognitive Smart Engine (CSE) : The CSE is responsible for the data processing and the application of ML algorithms to anticipate SLA violations. It has direct interfaces with the data preparation block. It in-

cludes three main modules, namely, CSE/Forecasting, CSE/Violation prediction, CSE/SLA enforcer.

The cse/Forecasting (see sequence diagram in Figure **??**) module takes as inputs the preprocessed features from the data preparation module. These features are numerical values at a given time t. The CSE/Forecasting module computes the next values of these features at time $t+n$, e.g. $n = 1$ for one-step-ahead forecasting. More details on this operation in section A. The CSE/Forecasting module sends the forecasted values (Fig. 4) to the CSE violation prediction module. The objective of this module is to determine based on the forecasted values whether an SLO might be impacted or not. The CSE violation prediction has also access to the SLO repositories. SLO repositories contain the low-level/metric-level definition of SLOs targeted by the administrator i.e. clear objectives to meet.

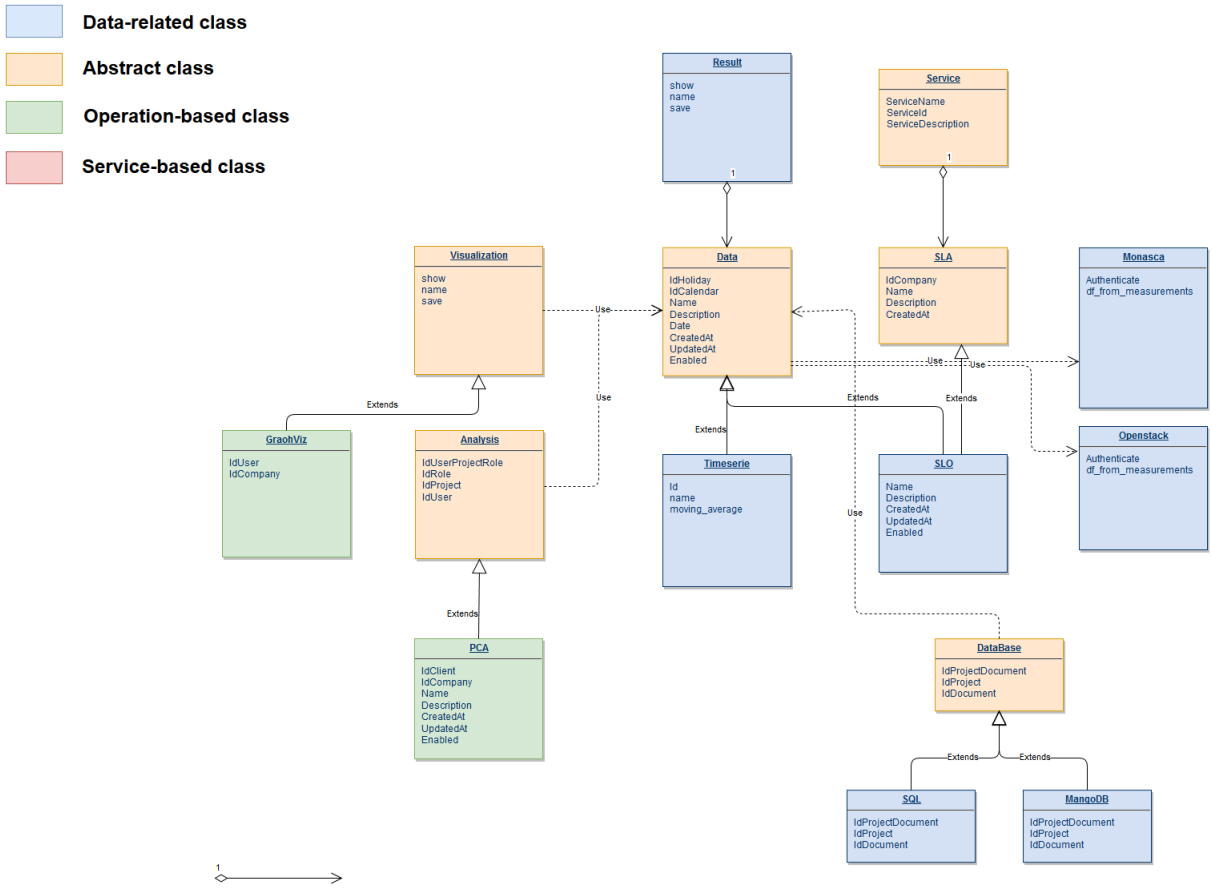


FIGURE 4.12 – UML and data model for CogSLA.

Use cases

Among the first challenges in this PhD was to acquire monitoring

dataset from NFV-based networks. At Orangte Labs, the legislation and concerns about users privacy was the main hurdle to acquire the data. Although some open sites and initiatives provide networking data such as DARPA [1] with both SLA violation and nominal behavior, we didn't find data that answers our requirements. We have the following requirement on the data :

1. Data should be representative of the typical SLA problems in softwarized networks, with both SLA violation and SLA compliant states. Moreover, the dataset should exhibit multiple SLO violations.

2. The ground truth should be known in advance, i.e. what is the root cause of the problem and the period of start and end.

3. Ability to test our own architecture and algorithms.

In order to overcome these obstacles, we setup our own infrastructure on the Cloud. We started with a first virtual platform for data streaming services ; then later on we installed an opensource VNF for multimedia services. The advantage of this approach is that it allows us to have more degree of freedom with respect to SLO violations and also enables us to test counteractive management actions and ML algorithms.

Use case I : Streaming service

The objective of the streaming use case test bed is to detect video degradation and ensure recovery. Streaming application relying on SDN controller, identify the metrics that we can monitor or obtain. For this use case we rely on Prometheus monitoring tool [161]. Prometheus is the next generation of monitoring system as well as a time series database, it is a pull-based monitoring system, i.e. it scraps(listens to) monitoring objects over time such as latency or mean throughput. An initial SWOT analysis of Prometheus was performed (see Table G) :

1. DARPAhttps ://www.ll.mit.edu/ideval/data/

Strenght	**Weaknesses**
Performance by design Reliability by design (Prometheus server is a standalone, not connected to zookeper for ex) Monitoring as a service (not as a machine) a flexible query language to leverage dimensionality multiple modes of graphing and dashboarding support	Only Pull-based : prometheus is about metrics collection not events collection recording only purely numeric time series (no complex structural data such as topologies) if you need 100% accuracy, such as for per-request billing, Prometheus is not a good choice. Suited for dockers and web-services !
Opportunities	**Threats**
pushing time series is supported via an intermediary gateway a multi-dimensional data model (time series identified by metric name and key/-value pairs)	time series collection happens via a pull model over HTTP

TABLE 4.2 – SWOT Matrix analysis of Prometheus

In order to evaluate the work to be done in the SLA enforcement for a given video steaming application, Orange will set up a testbed environment on top of opensource platforms including (OPNFV/ OpenPlatform for NFV ; Cloudify and Openstack /orchestrator ; OpenDaylight/ the SDN controller ; Clearwater/ the open platform for vIMS (virtual IP Multimedia Subsystem) as the VNF). The role of this testbed is to deploy, run and supervise fault and performance of services. By services we mean streaming applications and voice calls using a softphone application.

In our testbed we will operate the SLA enforcement use case to target video degradation in a unicast streaming application. We plan to first detect and then predict any streaming quality degradation that can affect a particular SLO. The minimum setup is composed of a group of interconnected VMs :

— a VM where OpenDaylight controller is run

— a VM where a server streams video in a predefined rate and encoding

— a VM where the client is receiving the streaming

— a stitching VM that connects the VM client to the VM server and

where an emulated network of different topologies and switches with different capacity links and random traffic generation that may cause sever quality degradation such as sever delay.

The data acquisition in our testbed will be based on real time monitoring in different data point using Prometheus monitoring tool. The advantage brought by Prometheus in this context is the data model based on time series database. For the SDN controller however, the monitoring is performed in two phases : offline using Cbench tool, a benchmarking tool for SDN controllers. The online monitoring will be based on inference from the VM that hosts the controller using Prometheus. [source : https ://github.com/dfarrell07/wcbench]. In the SLA enforcement use case, it is necessary to have an SLA repository where all SLAs are stored. An SLA is legally binding contract between a service consumer and a service provider that define all the term of services from QoS to billing and penalties as well. In our Testbed we are more interested in the QoS guarantee. These QoS threshold are refered to as SLO - Service Level Objectives. . Typically, an SLO for streaming video offering should take into account the following parameters :

— The encoding format and rate of the video stream

— The minimum video resolution, i.e. quality in term of pixel, e.g. minimum 480p with no latency

— Minimum FPS, Frame per seconds Translated into Low level SLO :

— Loss Rate (e.g. should be less than 5 percent)

— Latency (e.g. should be less than 5 seconds)

— Guarantee Bandwidth

0	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'guest', u'cpu': u'cpu0'}, u'value': u'0'}	node_cpu	COUNTER
1	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'idle', u'cpu': u'cpu0'}, u'value': u'8747.87'}	node_cpu	COUNTER
2	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'iowait', u'cpu': u'cpu0'}, u'value': u'8.14'}	node_cpu	COUNTER
3	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'irq', u'cpu': u'cpu0'}, u'value': u'3.8'}	node_cpu	COUNTER
4	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'nice', u'cpu': u'cpu0'}, u'value': u'0'}	node_cpu	COUNTER
5	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'softirq', u'cpu': u'cpu0'}, u'value': u'0'}	node_cpu	COUNTER
6	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'steal', u'cpu': u'cpu0'}, u'value': u'0'}	node_cpu	COUNTER
7	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'system', u'cpu': u'cpu0'}, u'value': u'19.63'}	node_cpu	COUNTER
8	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'user', u'cpu': u'cpu0'}, u'value': u'17.45'}	node_cpu	COUNTER
9	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'guest', u'cpu': u'cpu1'}, u'value': u'0'}	node_cpu	COUNTER
10	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'idle', u'cpu': u'cpu1'}, u'value': u'8769.3'}	node_cpu	COUNTER
11	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'iowait', u'cpu': u'cpu1'}, u'value': u'1.45'}	node_cpu	COUNTER
12	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'irq', u'cpu': u'cpu1'}, u'value': u'1.22'}	node_cpu	COUNTER
13	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'nice', u'cpu': u'cpu1'}, u'value': u'0'}	node_cpu	COUNTER
14	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'softirq', u'cpu': u'cpu1'}, u'value': u'0'}	node_cpu	COUNTER
15	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'steal', u'cpu': u'cpu1'}, u'value': u'0'}	node_cpu	COUNTER
16	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'system', u'cpu': u'cpu1'}, u'value': u'15.67'}	node_cpu	COUNTER
17	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'user', u'cpu': u'cpu1'}, u'value': u'8'}	node_cpu	COUNTER
0	The number of I/Os currently in progres	{u'labels': {u'device': u'sda'}, u'value': u'0'}	node_disk_	GAUGE
1	The number of I/Os currently in progres	{u'labels': {u'device': u'sr0'}, u'value': u'0'}	node_disk_	GAUGE

FIGURE 4.13 – Overview of Prometheus data set.

After preparing the data we receive 10 native metrics per VNF. New features are then created from this data. The total outcome reaches 20 features. In order to emphasize variation and reduce the dimensionality of the inputs, we used two different techniques. Firstly, High Correlation Filter to remove all the metrics that have more than 80% Pearson correlation. Then, Principle Component Analysis (PCA) technique that uses orthogonal transformation to reduce the features dimensionality while preserving their main characteristics. We used PCA to reduce the number of initial features to a smaller set of features, termed PC - Principal Component. We used this technique to preserve at least 80% variance of the initial dataset. This operation transformed features such as $\frac{CPU}{RAM}^2$ or log(CPU) to abstract component denoted $PC1 = F1, PC2 = F2, PC3 = F3$. The result of this phase is the reduction from 20 to 3 dimensions. The raw data is received at a frequency of 30 seconds. In order to predict next values at a lower frequency, we perform an autocorrelation test on the data and reduce their frequency using rolling mean up to 30 minutes frames. Lastly, we normalize (equation 1) and de-trend the data by computing percentages and subtracting the mean to capture exclusively the data fluctuations. This last operation is crucial to improve the learning of time series i.e. reducing the cost function. The fluctuation of the (1)

0	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'guest', u'cpu': u'cpu0'}, u'value': u'0'}	node_cpu	COUNTER
1	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'idle', u'cpu': u'cpu0'}, u'value': u'8747.87'}	node_cpu	COUNTER
2	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'iowait', u'cpu': u'cpu0'}, u'value': u'8.14'}	node_cpu	COUNTER
3	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'irq', u'cpu': u'cpu0'}, u'value': u'3.8'}	node_cpu	COUNTER
4	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'nice', u'cpu': u'cpu0'}, u'value': u'0'}	node_cpu	COUNTER
5	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'softirq', u'cpu': u'cpu0'}, u'value': u'0'}	node_cpu	COUNTER
6	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'steal', u'cpu': u'cpu0'}, u'value': u'0'}	node_cpu	COUNTER
7	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'system', u'cpu': u'cpu0'}, u'value': u'19.63'}	node_cpu	COUNTER
8	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'user', u'cpu': u'cpu0'}, u'value': u'17.45'}	node_cpu	COUNTER
9	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'guest', u'cpu': u'cpu1'}, u'value': u'0'}	node_cpu	COUNTER
10	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'idle', u'cpu': u'cpu1'}, u'value': u'8769.3'}	node_cpu	COUNTER
11	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'iowait', u'cpu': u'cpu1'}, u'value': u'1.45'}	node_cpu	COUNTER
12	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'irq', u'cpu': u'cpu1'}, u'value': u'1.22'}	node_cpu	COUNTER
13	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'nice', u'cpu': u'cpu1'}, u'value': u'0'}	node_cpu	COUNTER
14	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'softirq', u'cpu': u'cpu1'}, u'value': u'0'}	node_cpu	COUNTER
15	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'steal', u'cpu': u'cpu1'}, u'value': u'0'}	node_cpu	COUNTER
16	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'system', u'cpu': u'cpu1'}, u'value': u'15.67'}	node_cpu	COUNTER
17	Seconds the cpus spent in each mode.	{u'labels': {u'mode': u'user', u'cpu': u'cpu1'}, u'value': u'8'}	node_cpu	COUNTER
0	The number of I/Os currently in progres	{u'labels': {u'device': u'sda'}, u'value': u'0'}	node_disk_	GAUGE
1	The number of I/Os currently in progres	{u'labels': {u'device': u'sr0'}, u'value': u'0'}	node_disk_	GAUGE

Step	VM1_CPU	VM2_CPU	VM3_CPU	VM4_CPU	VM5_CPU
0	0.150986422881	0.269691590649	0.450029619263	0.365317227983	0.472509982433
1	0.573975087138	0.202128595953	0.489841958063	0.380519177937	0.0943374829268
2	0.184800437359	0.163116309533	0.589979974021	0.211981983277	0.314831563328
3	0.24962778947	0.224630752635	0.264295929727	0.471283053293	0.462342405757
4	0.67220747144	0.483885609923	0.697582511433	0.823885652838	0.382131581514
5	0.336330014499	0.699983764393	0.765509341404	0.267278691863	0.380897164051
6	0.253887821303	0.583676779931	0.736480517457	0.420109141108	0.064334442319
7	0.0499354422689	0.186851778565	0.225457832488	0.110617677523	0.257155873305
8	0.759619888493	0.175033118023	0.517359709423	0.250954447546	0.196362435252
9	0.546705609708	0.416332320168	0.787194591481	0.580826486554	0.53938924273
10	0.325936630413	0.191487815978	0.505739227941	0.215343447172	0.420126869253
11	0.216122658448	0.275807118582	0.708170073604	0.36203133203	0.139873405507
12	0.31351846441	0.429353071945	0.645777181387	0.445374609102	0.707159486423
13	0.537475814493	0.37482525608	0.402239255277	0.174703931762	0.198194191666
14	0.441082070936	0.328456802318	0.177195826644	0.859929113264	0.542789038096
15	0.295668533262	0.332989327513	0.587431502579	0.514987759719	0.0837626726753
16	0.542939787384	0.568897450691	0.00892867714171	0.286426714286	0.173605685547
17	0.0594280376933	0.422110419275	0.432786400486	0.323025950583	0.767515899662
18	0.175335751699	0.694380772338	0.533698647997	0.511019496851	0.258861447559
19	0.328735870993	0.0610590153703	0.531506984253	0.0903779339415	0.516707588684

FIGURE 4.14 – Overview of Prometheus data set.

In the literature, recent work have used streaming services to improve the network management. Hasan et al. [162, 79] have used streaming service based on VLC media player in a load balancing setting to detect SLO violations.

The streaming services are easy to install, modify the traffic and it is important because you can see the SLO violation directly as the video stops or changes in quality.

The figure **??** repsents the architecture of the streaming testbed. This testbed is composed of five VMs running on Ubuntu server with 16 *GB* of RAM and 4 vCPU for each VM. We instantiated a Monasca server

in the first VM and Openstack in the second one. The latter is playing the role of the authenticator using Openstack Keystone. The third and fourth VMs host a virtual network, composed of Virtual switches (Open vSwitch) and multiple VM guests. Guests correspond to three clients, three servers and five vSwitches, and one OpenDaylight controller. Figure 5 depicts how we design our virtual network. The goal of this architecture is to allow the creation of multiple paths between the sources (i.e. streaming servers) and the destinations (i.e. the clients) using RTP streaming protocol. In the fifth VM, we construct our CSE.

In each guest VM, we instantiate a monitoring agent. The agent retrieves the local information and sends it to the Monasca server. The Monasca server in turn, receives all the data generated in the form of time series from several data point and data source (e.g. VMs hosting VNFs, PMs, virtual switches) that are monitored by Monasca agents. The Monasca server then stores all the incoming data in InfluxDB, an open source database for storing and managing time series. The raw monitoring data received by Monasca are 180 system-level metrics per VM, each metric has 86.400 entries corresponding to 30 days monitoring with 30 seconds push frequency. Using the Monasca REST API, we can retrieve at real-time all the stored metrics. This should in turn be selected and filtered before feeding the CSE. Each streaming VNF runs locally in an infinite loop a high video quality (720p) of one hour and a half-broadcasted over Real-time Transport Protocol (RTP) video streaming protocol. Each client accesses and reads the video in its original format, which generates a network stream over the two end-points. Once the streaming service is up and running, the three VM clients are connected and receive the video streaming, we start by injecting 4 types of faults. (1) Node failure, (2) link failure, (3) node overload, and (4) link overload.

The aim of fault injection is to generate training example for the Artificial Neural Network (ANN) as in Figure 6 to learn on and then to generalize to other similar types of faults patterns.

The testbed generates several traces that are reshaped as stream of matrices with 20 rows (5 inputs per feature * 4 features) which corresponds to the size of the sliding window and the size of the input layer in the ANN (see Figure **??**). In the literature, we found no clear metho-

dology or consensus on the appropriate size of the input layer. However, we stress that this parameter is the most important one in the cognitive engine, since it can capture the hidden correlation in the stream.

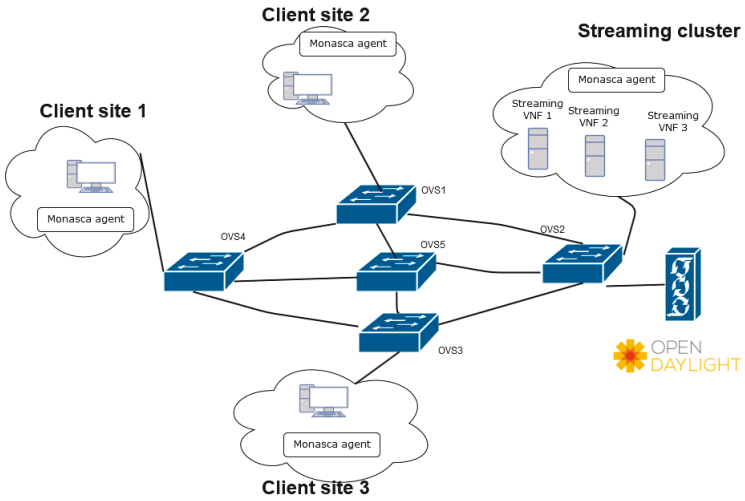


FIGURE 4.15 – Network diagram for the streaming use case.

TABLE 4.3 – Data summary of the use case I

The observation window	1 week
Number of entry lines per metric	20.000
The sampling frequency	30 seconds
Number of raw metric per VM	10 metrics
Number of feature per VM	5 metrics
Number of total features	5 metrics * 6 VNFC = 30 features
Number of SLO violations	1

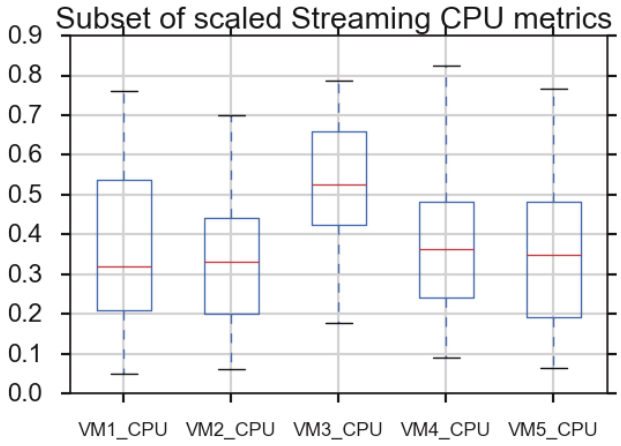


FIGURE 4.16 – Data distibution of the streaming use case.

Use case II : virtual IP Multimedia Subsystem (vIMS)

Clearwater is an open source VNF that uses SIP as a call control for voice and video communications [163]. Clearwater respects the basic IMS architectural principles and interfaces well-known in the telecommunication world. In the literature Clearwater has been use as the main testbed for anomaly detection in nfv [164], S. Makhsous et al. used it to study high-availability [165] for NFV deployments.

Clearwater has interesting properties with respect to our study. It is first designed to support horizontal scalability (i.e. adding more VMs) which can be considered as a management counteraction. Second, it supports test generation tools such as SIPp [166], stress-ng [167] to generate traffic and create different SLO violations to test our prediction accuracy. Third, it can be deployed on Openstack VIM (Virtual Infrastructure Manager), which allows us to fully integrate it to our existing framework. Clearwater is built around the interworking of 6 VNFCs (see Figure 4.17). The VNFCs are : Ellis, Bono, Sprout, Homer and Homestead, each having a specific function. Ellis is a provisioning portal providing sign-up, password management, and sip identities management. The IMS I-CSCF (Interrogating-Call Session Control Function) and S-CSCF (Serving-Call Session Control Function) functionality are implemented in Sprout. Bono is the Clearwater edge Proxy, it uses Sprout (SIP Router). It implements the P-CSCF functions (Proxy-Call/Session Control Functions). It is the entry point of SIP clients, which is in turn routed SIP requests to Sprout. Homer is a XDMS, a standard XML Document Management Server, it stores multimedia telephony service (MMTel). It runs Cassandra database. Homestead (HSS Mirror) relies on HTTP RESTful interfaces and Cassandra as data store. It is used by Sprout to retrieve authentication credentials and user profile. It also delivers some I-CSCF and S-CSCF functionalities. Ralf (CTF) is responsible for charging and billing. For more details on Clearwater VNFs, readers can refer to the project documentation [168]. Our Clearwater setup consists of 10 VMs (6 Clearwater VMs, 1 Monasca VM, 1 SDN VM, 1 VM where we run the framework, 1 DNS VM) and over of 2 *Gbits* of collected Monitoring traces. In addition, 30.000 different SIP profiles were created and stored using Homestead in the local Cassandra Database. These profiles are used to generate traffic and anomalies when they are launched si-

multaneously. The monitoring tool for our framework is Monasca [169]. Monasca is a MaaS, Monitoring-as-a-Service solution from HP, built as a highly scalable Openstack service.

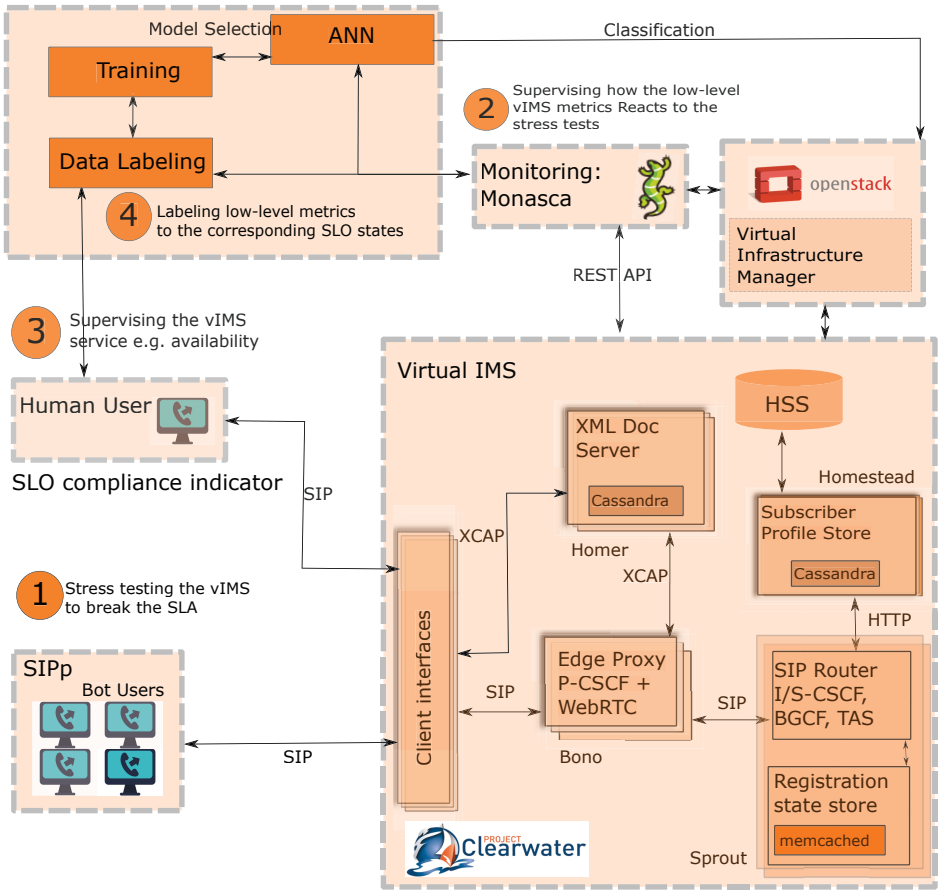


FIGURE 4.17 – Our Testbed. Clearwater virtual IMS functional architecture in the box lower right. Upper left the Cognitive Smart Engine (CSE). The experimental process is : (1) Stress testing for SLA violation generation. (2) System-level supervision. (3) Reporting SLO violations. (4) Data labeling, merging observations on the SLO state and the system-level metrics.

TABLE 4.4 – Data summary of the use case II

The observation window	2 months
Number of entry lines per metric	200.000
The sampling frequency	30 seconds
Number of raw metric per VNFC	30 metrics
Number of feature per VNFC	26 metrics
Number of total features	26 metrics * 6 VNFC = 156 features
Number of SLO violations	3

Figure 4.18 shows twelve whisker plots representing different low-level metrics of the core IMS nodes. The figure shows difference in the distribution of the same metric in different VNFCs (for example ralf-cpu and homer-cpu). This means that the stress test impacts different VNFCs with different degrees.

The dataset describes both normal SLA state and the violation state. The scale of the time series varies widely from one metric to another. Therefore we decided to rescale the metrics to [0-1] range (in figure 4.18 the range is from [0-10]), so that the classifier considers all the metrics with no scale bias. The raw data is processed into a machine compliant format (i.e. tensor matrix). In the cleaning phase, we reduce the entry lines from more than 400.000 lines to 200.000 lines. In this phase, the majority of the discarded lines are either redundant entries or non-exploitable errors or missing values. The dimension reduction in our case refers to reducing the number of entries while keeping the properties of the time series. This technique consists of eliminating few samples in the data that do not entail a change in the form of the curves. It is formally described as follows : Given a time seriesT_1, with n data-point, generate a new time series T_2 with p data points such as $p < n$ and T_1 approximates T_2 . In this phase we reduce the data size from 200.000 steps to 177.000 entries.

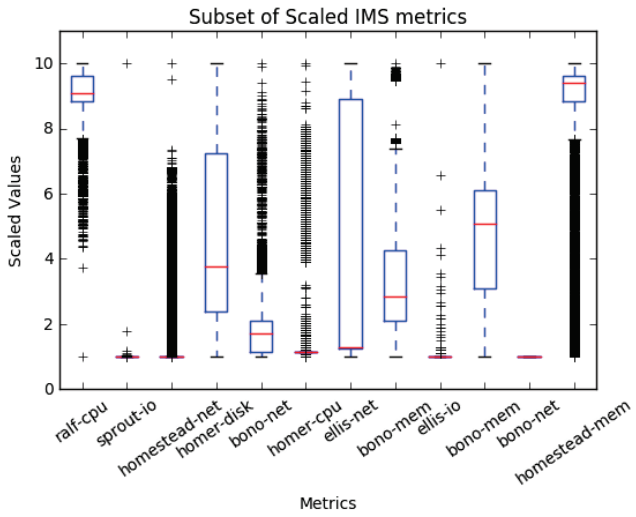


Figure 4.18 – A subset of the data set distribution. The small boxes represent the quartiles of the distribution. The red line in the middle represents the median, i.e. the point separating the data into half. The outliers are drawn as black crosses outside the box.

We used as inputs to the ANN a 3D Tensor, which is a mathematical representation of the inputs. The tensor has the following shape : $< \#of lines,\ timesteps,\ \#of features >$. The first dimension corresponds to the number of entries per time serie, it corresponds also to the number of lines. The second dimension is the timesteps used, we keep the default value of 1 corresponding to 1 timestep per entry. The third dimension corresponds to the number of features that is 156 metrics. Equation 5.3 represents the inputs in 2 dimensions, $x_{i,j}$ where i is the line number and j is the feature number :

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \dots & x_{1,156} \\ x_{2,1} & x_{2,2} & x_{2,3} & \dots & x_{2,156} \\ x_{3,1} & x_{3,2} & x_{3,3} & \dots & x_{3,156} \\ x_{4,1} & x_{4,2} & x_{4,3} & \dots & x_{4,156} \\ & & \dots & & \\ x_{m,1} & x_{m,2} & x_{m,3} & \dots & x_{m,156} \end{bmatrix} \tag{4.2}$$

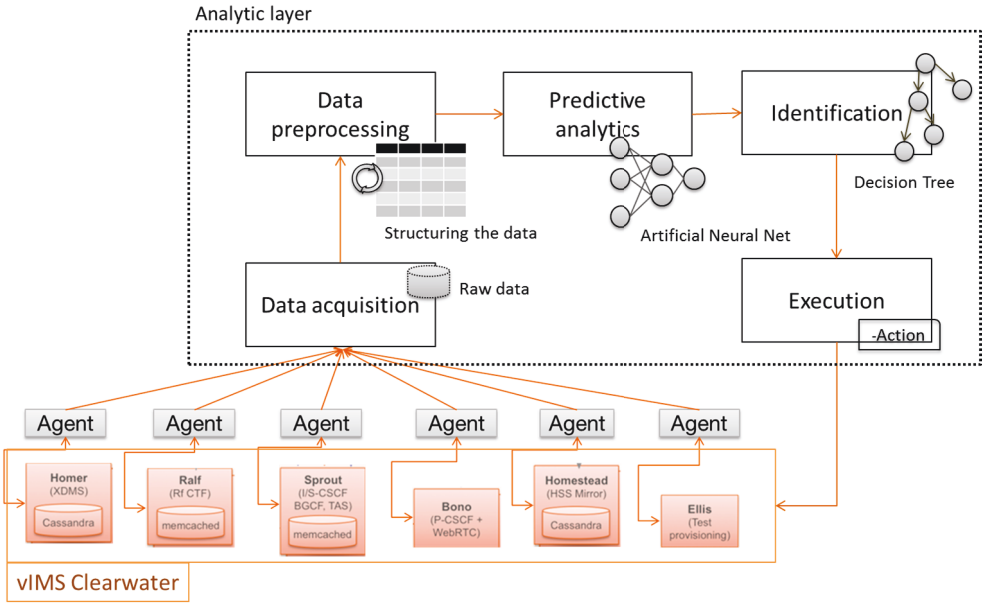


FIGURE 4.19 – Cognitive SLA Architecture.

Data Acquisition and Monitoring

Monasca Introduction

The select monitoring tool for both testbeds is Monasca [169]. Monasca is a MaaS, Monitoring-as-a-service solution from HP, built as a highly-scalable Openstack service. It uses Apache Kafka queue technology [170] to prioritize incoming flows and InfluxDB as a time series database. The advantages of Monasca over current monitoring solutions (e.g. Zabbix, ceilometer) are than it is highly scalable, integrated as an Openstack project and it allows us to define new sets of metric for our research problem.

In our example, the monitoring interval (data pushed by Monasca forwarder by Monasca agent) is the default 30 seconds.

Monasca by default collects 30 metrics per VM, four of which are of no use in our case (e.g. idle values or Null values). We keep 26 metrics per VM, which correspond to 156 metrics with all the 6 VMs (Bono, Ralf, Sprout, Homer, Homestead, Ellis).

We have been faced with two implementation choices of the CSE, the first choice is to integrate the CSE as kafka subscriber. The second implementation is to connect the CSE as an external service using REST API.

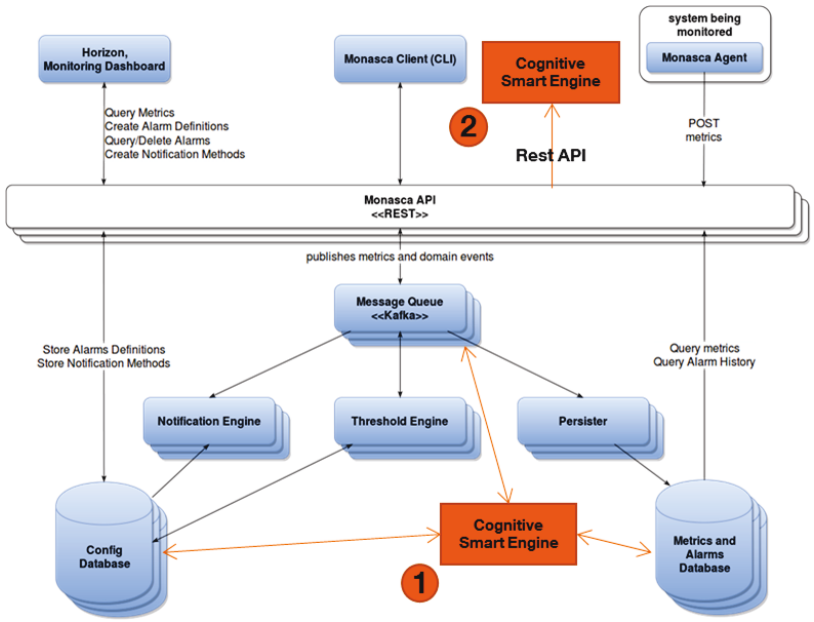


FIGURE 4.20 – CSE implementation choices.

Implementation choice 1 : seamless integration

The implemntation number 1 (Figure 4.20) allows a rapid, low-latency processing. This approach is suited for real time analytics. It allows a

seamless integration with the monitoriung system. However, this implementation is too system oriented thus can not easily generalize to other implementation. Furthermore, it necessitates an in-depth understanding of Monasca and Kafka technology.

Implementation choice 2 : service-based integration

The REST API implementation is the most compelling in our context because it is system agnostic, loosly integrated with Monitoring system.

This approach is more suited for a data science analysis. Offline analysis, is an important step to understand the data structure and distribution, the nature of the problem before starting to implement the CSE.

This approach will be the basis of the next chapter were we discuss the preprocessing phase.

IV Data Analysis

In the 5th generation networks, we expect tens of thousands of input features across layers that are available for the network management system. Data extraction is intended to increase the accuracy of machine learning models by extracting salient features from the raw input data but also potentially remove noise and redundancy from monitoring records. It also simplifies and facilitates model selection since, if relevant features can be extracted, even a simple model can offer remarkable results. Additional aims include lowering the dimension of data to facilitate training speed and visualization. All of these techniques and examples are discussed in this section.

A Data Gathering

Data gathering service extracts data from internal data sources such (network KPI monitors) and external sources that are applicable for a given machine learning service. The service can be configured based on which machine learning services that are actively used and caches the data it gathers in a database for later usage. Data gathering service can import raw data in batch mode or streaming mode depending on the service requirements and scheduled for periodical data imports.

Data extraction and collection

The extraction of the monitoring data is performed by installing monitoring agents on every VM. The agents collect metrics from different sources as follows :

— System metric such as network i/o and memory

— Host live checks. The agent can check periodically the state of the host to determine if it is alive using ICMP ping.

— Process checks, checking active service endpoints by sending HTTP requests to an API

A metric is identified by a name and dimensions. Each metric is as a metric are name, timestamp, and value.

The chartflow

The flow of the agent application goes like this : (you should change it please because It was copied from here monascahttps ://github.com/openstack/mon agent/blob/master/docs/Agent.md)

1. The collector runs based on a configurable interval and collects system metrics such as cpu or disk utilization as well as any metrics from additional configured plugins such as mySQL or Kafka.
2. The statsd daemon allows users to send statsd type messages to the agent at any time. These messages are flushed periodically to the forwarder.
3. The forwarder takes the metrics from the collector and statsd daemon and forwards them on to the Monasca-API.
4. Once sent to the Monasca-API, the metrics continue through the Monasca pipeline and end up in the Metrics Database.
5. The collector then waits for the configured interval and restarts the collection process.

In this phase, we collect SLA information as key, values pairs that describes specific SLOs. The data input was in the form of a YAML file as depicted in figure **??** below.

One important aspect of the data preparation phase is the ability to filter out the data that cannot be used by the machine learning algorithms. For example, white noise, or other data with no correlation. The technique used is the autocorrelation ; it allows us to study if the time

series data are (1) forecastable and (2) to identify for how many time steps we can see in the future. In the figure 1, the autocorrelation identified the correlation between the previous step and the next step, the results are that there is a correlation between x(t) and x(t-1) up to 80%, but between $x(t)$ and $x(t-2)$ the correlation drops to 60%

In our work, the time series are stored in three different formats :

1. CSV - Comma Separator Value : Simplest way to store data, it is effective for small data and data preparation phase. It can be easily analysed by a human operator.
2. SQL : It is used as the back end of our system, can store high dimensionality data. However this solution might be a hurdle when scaling out to millions of read/write operations per second.
3. InfluxDB : a hight availability throughput database specialized for time series storage, it is used by the monitoring system Monasca.

B Data preparation

Data preparation service processes raw data stored by the data gathering service based on the requirements of each machine learning service. Data preparation service provides the following functionality :

— Aggregation of data.

— Clean data by detecting noise.

— Split data for machine learning experiments.

The output of the data preparation service is processed datasets that can be used by the machine learning services directly or after dimensionality reduction. The output can be sent in batch mode or streaming mode based on the machine learning service requirements.

Data preprocessing : From naive to feature engineering

Data cleansing and transformation. Initially, we remove corrupted values, redundant entries and null or undefined metrics in order to reduce the size of the data storage. This dramatically improves the data quality and is a necessary step that takes multiple iterations and analysis before complete automation. Afterwards, the time series are transformed into a stationary state with a mean of 0 and a standard variation

of 1, this step is important because it allows ML algorithms to compute Euclidian distance and can theoretically facilitate the convergence of the training process such as backpropagation algorithm. The extraction of the features covers methods that transform the raw data into features that contains discriminative information for machine learning algorithms. We present in table 4.6 an example of informative features.

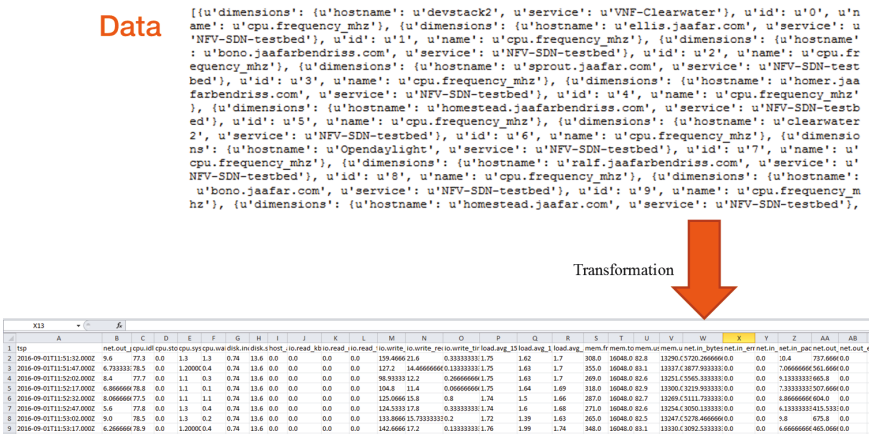


FIGURE 4.21 – Raw data in JSON into a Table

Feature Engineering

Feature engineering is one of the most important phase of the data services. In this work, we distinguish between three types of data engineering :

1. Human-based/ Manual (iterative, intuition)
2. PCA
3. Minimal process without feature engineering

Engineered KPI :

The engineered KPI selection is a method of selecting a subset of the total inputs based on intuition. The role of the human expert is key in this phase. The expert role is two folds : (1) selects the most meaningful metrics with respect to the use case, (2) create a - usually creative - combination of metrics. Generally, when the human experts are involved in the machine learning loop, the process is more iterative and selecting from the first shot the right metric is unlikely. However, in this study

we limit the iteration to the first iteration in order to compare the human intuition to other more advanced techniques. Moreover, for sake of simplicity we limit the number of features to four.

The selected metrics using the Engineered KPI are of four broad categories : performance, memory, network and disk. The metrics are at the level of the Virtual Machines and are defined as follows :
Performance-related feature :

$$X^{(1)} = \frac{(Cpu_{load})^2}{RAM_{load}}$$

Memory-related feature

$$X^{(2)} = RAM_{load}$$

Network-related feature

$$X^{(3)} := \frac{Packet\ rate}{RAM_{load}}$$

Disk-related feature

$$X^{(4)} = \frac{Network_{InputPackets}}{DiskAccess}$$

And we refer to X as a matrix that contains all the features represented as a vector, namely, $X^{(1)}$, $X^{(2)}$, $X^{(3)}$ and $X^{(4)}$.
Formally, for each last N values of X we want :

$$\hat{X}_t = f(X_{t-1}, X_{t-2}, X_{t-3}, \ldots, X_{t-N})$$

$\hat{X}_t$: is the forecasted matrix that contains the forecasted values of all the four features. It comes with an error ε :

$$\hat{X}_t = X_t + \varepsilon$$

X_{t-i} : is the previously observed matrix X at time $t-i$
Based on the forecasted matrix $\hat{X}_t$, we want to determine which SLOs can be violated. We write it as a probability for an SLO, s :

$$P(\ s\ is\ breached\ /\ \hat{X}_t) \geq \alpha$$

α : is the threshold above which we consider the SLOs s as the future violated SLO.

C Dimensionality reduction

The goal of machine learning algorithms, which are the driving force of the cognitive capabilities of CogNet, is to find functions that aid in performing tasks automatically. For instance, given a set of cloud infrastructure monitoring metrics, we might want to have a function that estimates the probability that a virtual machine will fail during the following 30 seconds; or given a set of TCP flow characteristics, such as average round-trip time and packet payload size, we could benefit from a function that tells us whether the flow contains video data or not. To find those functions, we need to provide machine learning algorithms with input variables - such as the cloud monitoring metrics or the flow features. As the number of variables are increases in a dataset, the information content of the data will increase. However, if there are too many variables, learning the patterns in the data might become harder exponentially due to the increased sparsity of the data. It is therefore often necessary to reduce the number of input variables to a sufficiently small number while retaining as much relevant information as possible. The process of decreasing the number of variables is known as dimensionality reduction. There exist many algorithms to perform this task. Perhaps the best known is Principal Components Analysis (PCA) [171], which finds a set of independent latent variables whose cardinality is smaller than the dimensionality of the data which retain as much variance of the original data as possible. A similar approach, although nonlinear and thus more powerful, is that of auto-encoders [172]. Despite being more powerful in its representational expressiveness, it is more difficult to use, and should be employed only if the results of simpler methods such as PCA are unsatisfactory. Other popular approaches are especially helpful for visualization, such as t-SNE [173]. These methods have one drawback : they transform the data into a new variable space, making the resulting models difficult to interpret in practice. This can be overcome resorting to feature selection techniques, which instead of transforming the original variables, choose a subset of them. There exist many feature selection methods designed for the supervised setting, that is, when we have a labelled data set (a set of input observations, each of which is accompanied by a target value that the machine learning algorithm should learn to guess). Notable examples are wrapper methods

[174], which greedily select variables that maximize the ability of the machine learning algorithm to guess the target values, or information theoretic approaches [175]. The output of the dimensionality reduction service can be used by all the machine learning services of the CogNet service portfolio.

PCA-based KPI :

The PCA-based KPI is a method of selecting the most significant metrics based on the abstraction of the inputed features into Principal Components (PCs). As inputs, the PCA algorithms receives 30 metrics per VM and outputs a set of PCs with their relative variance. We select the PCs with the highest cumulative variance.

Table 4.5 – PCA-based metrics results

	sdev	varprop	cumprop
	Standard Deviation	Proportion of Variance	Cumulative Proportion
PC1	1.725645	0.425407	0.425407
PC2	1.306952	0.244017	0.669425
PC3	1.049037	0.157211	0.826636
PC4	0.773208	0.085407	0.912043
PC5	0.606810	0.052603	0.964646
PC6	0.433620	0.026861	0.991507
PC7	0.243830	0.008493	1.000000

In the table 4.5, the PCA algorithm outputed 7 Principal Components. Each of which with a standard deviation and the proportion of the variance. In the table, the results are printed in the descending order with respect to the PC relevance. We select the first four PC. Note that their cumulative variance is more than 91%. This means that using these four PCs we will miss only 9% of the system behavior.

For our first implementation, we used the threshold on the cumulative variance to 80%, which means that we can limit the features into : PC1, PC2 and PC3. The objective of the machine learning is to lean the patterns within the time series in order to forecast their evolution through time. The Machine Learning approach is based on an a special type of Recurrent Artificial Neural Networks termed LSTM for Long Short Term Memory. The LSTM approaches were introduced in 1997 [18], their main advantages are to retain information over many time

intervals. They have been used successfully in image recognition, translation, language representation, driverless cars, image description [19]. The RNNs have distributed hidden states that allow them to store information about the past and update the information in a non-linear way. They leverage the concept of associative memory and can identify a previously seen pattern from a new distorted version. This is particularly useful to model stochastic dependencies in Time series [20]. We built our LSTMs using the Keras library based on Google's TensorFlow [176].

Feature-Engineered KPI :

Before using time series as inputs to the CSE module, we performed a series of techniques to clean and organize the data. Then, we used other techniques to select only the time series that can be exploitable by the CSE. Firstly, the data received through REST API from our monitoring service was in a raw JSON form (see Figure 1). The first operation is to filter the data and reorganize it as a data frame table as in Figure 1. The output of this phase is data structured as time series with multiple variables (columns). The variables represent all the metrics that are collect by the monitoring agent. They are the information that we have on the system. The manipulation of these variables can extend/augment the information we perceived on the system being monitored. The process to do so is called : feature engineering. From these variables we generate another set of variables (or features) using two techniques :

1. The combination of two or more variables into one, e.g. $(\frac{CPU}{RAM})^2$
2. Applying a function to only one variable to stress its behaviour e.g. stressing the evolution of a variable CPU^3

The feature engineering is an empirical and iterative process. The features that yields the best output in terms of cost function are kept. Then the data are normalized and rescaled to [0-1] interval.

The second step complementary to the feature engineering is the dimensionality reduction. The aim of this phase is to reduce the dimensionality of the data by keeping only the most relevant components that capture the system behaviour. In the literature multiple techniques are used such as high correlation filtering, Low variance filter, backward feature elimination, etc. For this work we settled on PCA - Principal

Components Analysis, a technique that uses an orthogonal transformation on the data to create new vectors uncorrelated that capture the most variance in the data set. We applied this technique to 24 variables and the results as shown in Figure 2 was 7 Principal Components (PC). These PCs are abstractions of the input variables.

In order to forecast metrics behaviour in the MMCC scenario, we based our approach on the following axioms at time t :

— Each metric exhibit a non-null correlation, i.e. a pattern exists in the TS

— Correlation between different metrics and other more abstract features exists in the network (e.g. CPU, network utilization)

— Network centric metrics can affect the service quality and/or the whole end-to-end service

Features	Rule
Feature 1	cpu.userperc^2 / mem.usableperc
Feature 2	mem.usableperc
Feature 3	net.outpacketssec / mem.usableperc
Feature 4	net.outpacketssec / disk.spusedperc

TABLE 4.6 – Feature Engineering Rules

Autocorrelation and lagged function

The autocorrelation represents the correlation of a Series with respect to itself delayed in time. The autocorrelation allows us to draw lagged function, which serves as an indicator to the predictability of the metric. For stochastic random noise (i.e. White noise) the autocorrelation is small and constant which means that it is not predictable. However, for cosine function, the autocorrelation is very high (near 1) and constant in multiple time steps, which means that knowing previous values one can predict theoretically to signal evolution to infinity. In our Example in Figure 4.22, the autocorrelation shows that for most metrics we can predict one step-a-head values with more than 80% and then drops to 60% for the two steps-a-head values, this is due to the accumulated effect of uncertainty. The insights revealed by the autocorrelation is answering the question whether the data is forecastable or not and to which extent. For our case, the answer is a clear yes for the step-a-head value.

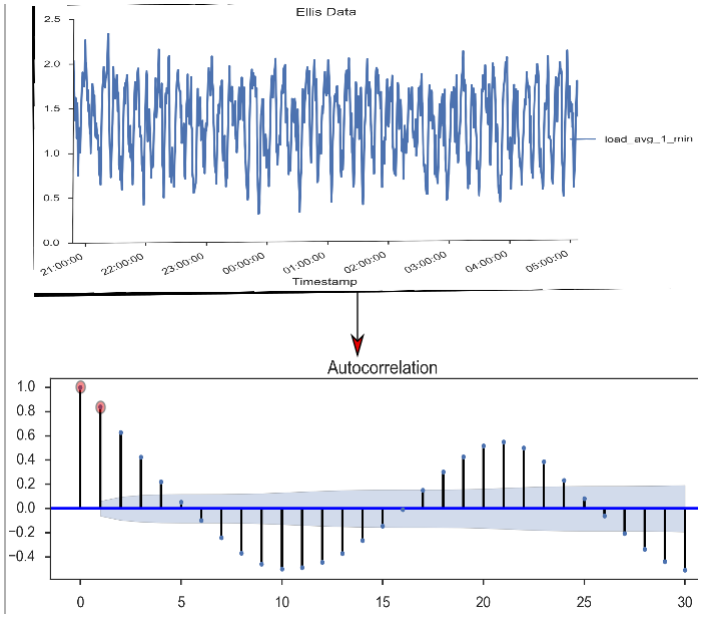


FIGURE 4.22 – Example of the autocorrelation function applied on the load of the SIP proxy.

Stationary data and decomposition

–A time series can be decomposed into 4 sub-time series, one that capture the general trend, the white noise, the absolute signal itself and the seasonality effect that is the repeating cycle as shown in Figure 4.22. Decomposing the metrics is essential to understanding the problem before the prediction phase. Each component of the time Series can be approached as a unique problem with different algorithms. In this phase, we discarded the general trend of the time series and focus on the remainder of the signals in order to capture the evolution of the metrics. Moreover, all the metrics that we collect have very different scales which make them difficult to compare and can ultimately yield a poor performance in the training and forecasting phase. One solution to this problem is to make all the metrics stationary, that is rescale them with common mean of 0 and standard variation of 1 (shown in **Figure 4.24**). Another solution to compare multiple different time series is to divide all the data by their first value to obtain a common starting position from one

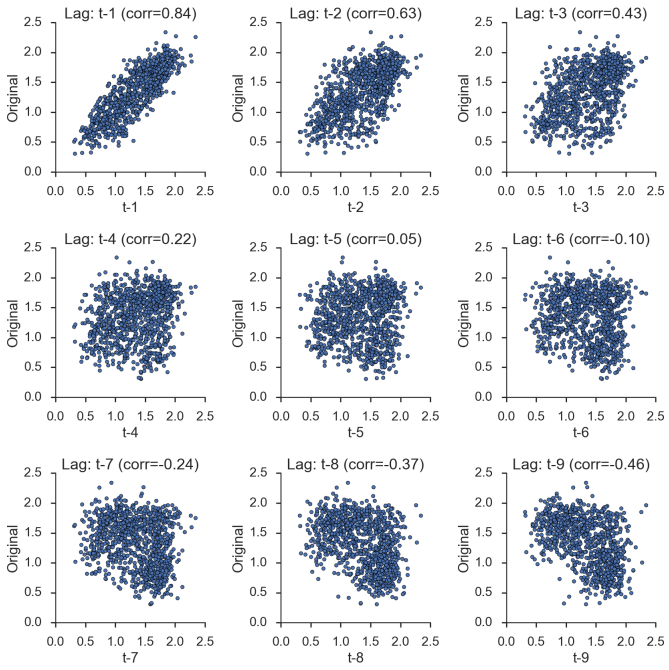


FIGURE 4.23 – Lagged autocorrelation.

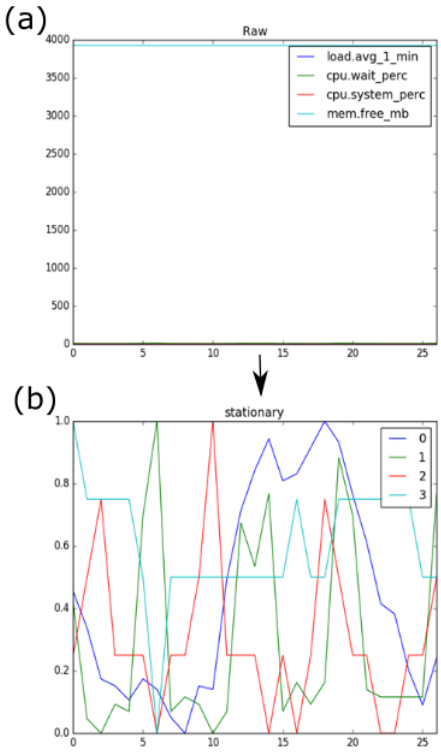


FIGURE 4.24 – Transforming raw timeseries into stationary ones

Correlation analysis

The correlation results over normal data set allow us to capture the relationship between

Also, the correlation matrix is specific to a given service, in Figure C the correlation matrix is tightly coupled to the IMS service. An intuitive example is the correlation between *Network.Packets.In* and *Network.Packets.Out* in the Proxy VNFC. Tracking this correlation for example can give us an insight or an early alarm wherever a decorrelation happened. For example, if in the proxy the *Network.in* becomes decorellated with the *Network.Out* that might be a first symptom of an anomaly and worth more investigation. This method have been used by A. Antonescu et al. [177] to define new SLOs i.e. pair of correlated metrics. One should keep in mind that correlation does not imply causation, the check the causation more controlled experiments should be performed.

Additional information is provided by the probability distribution of all these metrics. This can be leverage to (1) dimensionality reduction whereby tracking one metric of two highly correlated ones. Another approach (2) might be in tracking the decorrelation of previously known highly correlated metrics to trigger further investigation if the decorrelation occurs. For example in our case study, the VNFC proxy has net.input and net.out highly correlated, this rule doesn't hold for every service or VNFC.

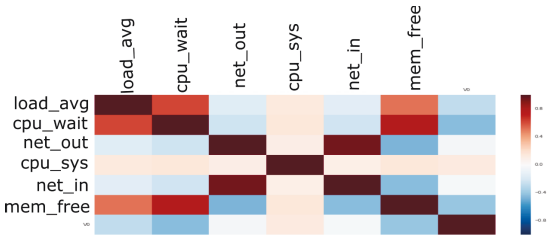


Figure 4.25 – Inta VM correlations

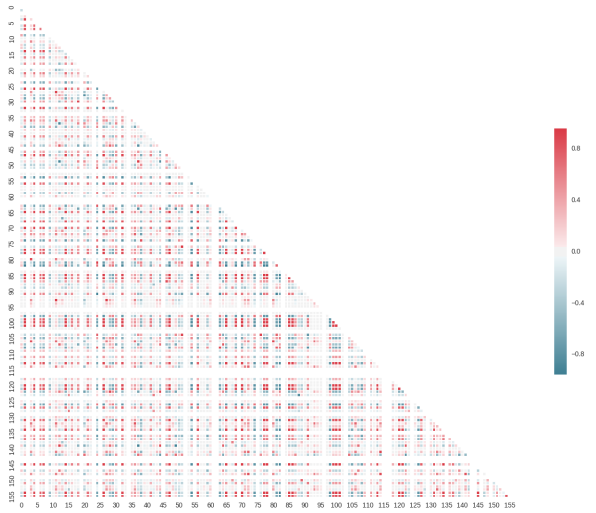


FIGURE 4.26 – Correlation bewteen all the 156 metrics of the testbed.

In this study we have used the Pearson correlation technique as a baseline for future analysis. The correlation captures the similarity degree of the evolution of a pair of metrics. We distinguish two types of correlations :

1. **Inter-VM correlation :** represents the correlation between features of all the VNF Components (VNFCs) that compose the network service (VNF). These calculations are service specific. An example is the correlation of *network_packets_in* of the proxy and the *cpu_perc* of the database.
2. **Intra-VM correlation :** deals with the correlation of the metrics within a a specific VNFC. Note that the correlation is computed based on an observation window without anomalies or SLO violations, to exhibit all the normal correlation of the variables.. An intuitive example is the correlation between *Network.Packets.In* and *Network.Packets.Out* in the Proxy VNFC. Tracking this correlation for example can give us an insight or an early alarm wherever a decorrelation happened. For example, if in the proxy the Network.in becomes uncorrelated with the Network.Out that might be a first symptom of an anomaly and worth more investigation. Note that correlation does not imply causation ; to verify the causal link more controlled experiments should be performed.

Correlation coefficient between two random variables X and Y is defined as

$$\rho(X,Y) = \frac{\mathbf{Cov}(X,Y)}{\sqrt{\mathbf{Var}(X)\mathbf{Var}(Y)}}.$$

In figure 4.26 we ploted pair-wise correlations for the entire test bed as a heat map figure. The correlation factor 1 means that the metrics are perfectly correlated, a near −1.0 means that the variables are negativaly correlated. The diagonal line in both figures 4.26 and C. whereas the factor 0.0 means that there are no correlation between variables.

— here we can add a discussion about some highly correlated variable and why is that : In the figures depicted above we observe many high correlations. Here we will discuss some of them and explain the meaning of this correlation. The VNF's proxy, the "network.in" and "network.out" in figure C are correlated with a factor of 0.86, this can be explained by the nature of the service. In the nominal IMS behavior the data plane is expected to have data flowing in and out in order to make and receive calls. The CPU and memory are also highly correlate with a factor 0.67, the more data is processed by the memory the more CPU resources are needed to efficiently finish the task. The "network.in" and "memory.free" in figure C are also inversly highly correlated, the more the VM receives traffics the less memory remains to treat local processes.

— It is important to note that correlation does not imply causation. When observing Two higly correlated variables we can not conclude definetly that one is causing the other, an example is presented in figure 4.27. In fact, the there can be up to 19 different causal relationships between these variables. However, one can infer causality from additional information such as the time during which the events happened or using experimental studies.

— how can correlation be leveraged for SLA violation detection? By tracking known correlations we could generate a first alarm when a decoleration happen. However, for this methodology to work, we should track the correlation for only the normal states. the tracked correlations depend highly on the use case. This approach is more suitable for anomaly detection. Antonescu et al. used this technique to report SLA violation [177].

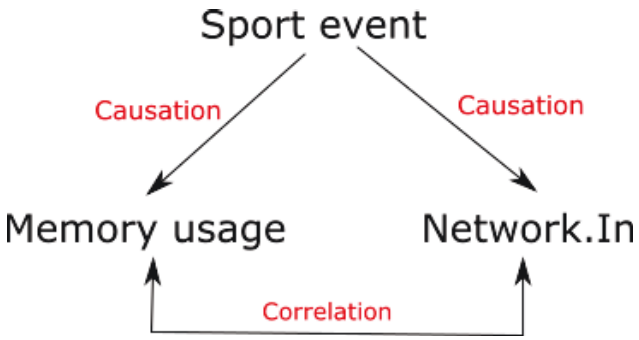


FIGURE 4.27 – Example of correlation versus causation.

D Visualization

In our data driven approach, the huge amount of data makes it difficult for human intuition to grasp or apprehend the degree to which

The operations performed on the data remains abstract and non intuitive. In this section, we ask the question of how can we remove the abstraction barrier by visualizing the evolution of the data.

The human mind can grasp up to 3 dimensions. 156 dimensions can not be represented to humans. So we opted for transforming the time series into a 2D plot representing in each column a feature and in each line a timestamp. After scaling the data per column to values ranging from 0 to 1, we correspond a range of visible color from red to blue as shown in figure 4.28.

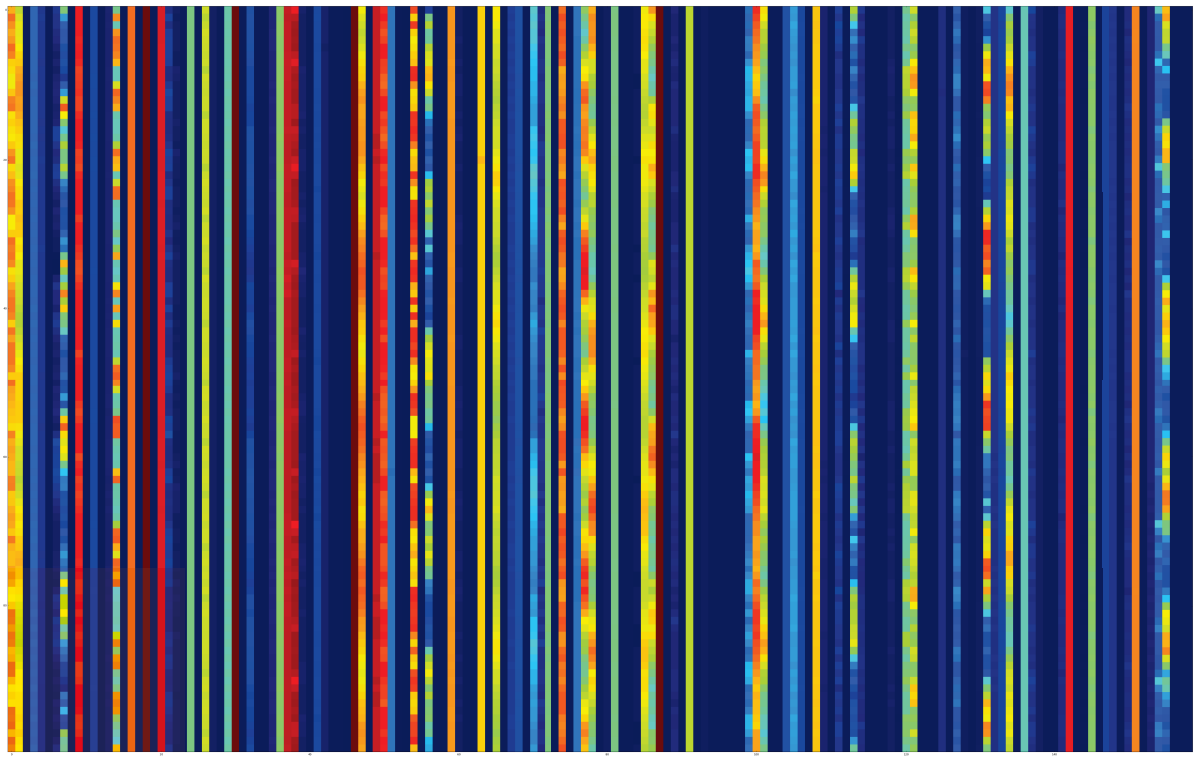


Figure 4.28 – time series visualization.

Similarly, we wanted to represent visually the SLA violation states and the nominal states. We based the visualization of the SLA based on a visualization algorithm called t-SNE, firstly introduced by Maaten et a. [173]. t-SNE is particularly useful in this context because it can measure their pairwise differences, t-SNE visualization can help also identify clusters in our data.

t-SNE is capable of retaining the local structure of the data while also revealing some important global structure.

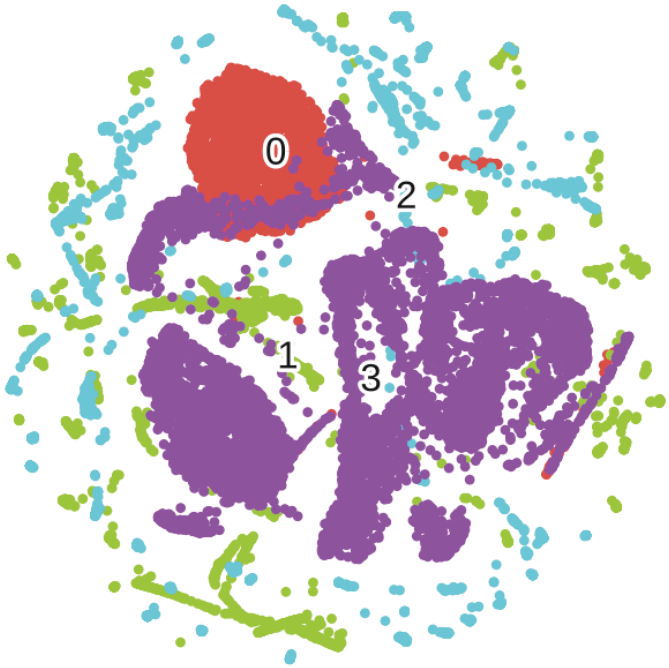


FIGURE 4.29 – T-SNE plot of SLA and SLA Violation.

Finally, SLA also can be visually represented. In this case, we use a radar map to note the degree of sensitivity of each SLO with respect to some KPIs (or PCs), an example is provided in the Figure below (Figure 4.30)

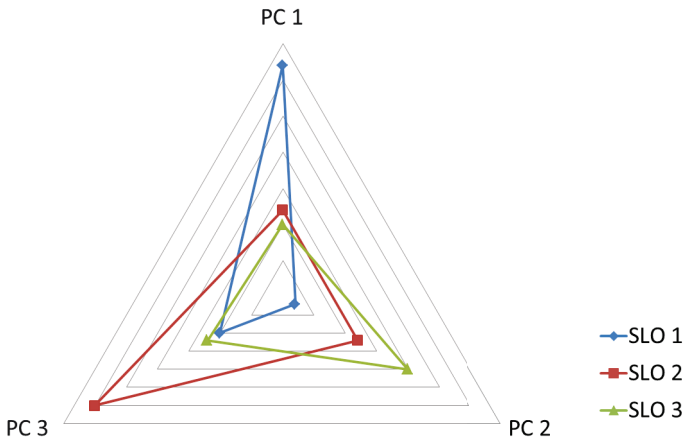


FIGURE 4.30 – SLO visualized as a radar map.

E SLA Assurance Services

Efficient Service Level Agreements (SLA) management and anticipation of Service Level Objectives (SLO) breaches become mandatory to guarantee the fulfilled services in the context of softwarized networks. SLA agreement [107] generally comprises parameters describing the service functional behavior and non-functional properties such as : the minimum acceptable QoS values (referred to as SLOs), e.g. Maximum VNF instantiation time. In this service, we consider a streaming service running on SDN and NFV infrastructure. This service is used for "SLA management" which is one of the key services in expected 5G networks. For this given service, we define the following three SLOs :

— SLO_1 **:** Response time ratio

— SLO_2 **:** Service Availability ratio

— SLO_3 **:** Downlink throughput ratio and downlink latency ratio.

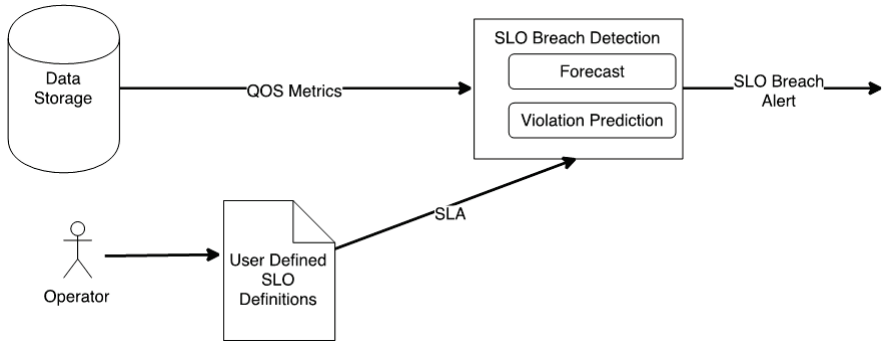


FIGURE 4.31 – Service Quality Monitor.

Cognitive SLA enforcement is the process by which operators maintain the compliance of services and their predefined SLA while using a computed knowledge with machine learning techniques [178]. Predefined SLA and service description, as well as monitored data are mandatory inputs to ensure cognitive processing. The service quality monitor has interfaces with CogNet data services. The CSE relies on two main modules :

— CSE/Forecasting

— CSE/Violation prediction

The CSE/Forecasting module takes as input the preprocessed and filtered data and computes the forecasted values i.e. the next-step ahead values of the given inputs [104]. The forecasted values are feeding the second CSE/Violation prediction module to identify (if any) the most probable affected/violated SLO based on the one-step-ahead prediction. To do so, CSE/Violation prediction module has direct access to repositories where Service, SLA and SLOs descriptors are predefined and stored.

From the outputted table in the data acquisition module, we construct a second table based on combination of features as shown in Table 4.6. The data preprocessing phase aims at improving the data quality in terms of structure, noise and consistency. Moreover, this phase acts as a requirement check-list between the data sources and the analytical framework.

V Conclusion

It is expected that the fifth generation mobile networks (5G) will support both human-to-human and machine-to-machine communications, connecting up to trillions of devices and reaching formidable levels of complexity and traffic volume. This brings a new set of challenges for managing the network due to the diversity and the sheer size of the network, mainly related to resource allocation, security and resilience, performance degradation, energy efficiency, big data and traffic management. It will be necessary for the network to largely manage itself and deal with organisation, configuration, security, and optimisation issues. The CogNet project proposes to tackle these challenges based on the identified use cases and scenarios presented in Deliverables 2.1 and 2.2 through an architecture of an autonomic self-managing network based on NFV/SDN, MAPE loop and Lambda architecture model and capable of achieving or balancing objectives such as high QoS, low energy usage and operational efficiency. The main novelty of the architecture is the CogNet Smart Engine introduced to enable Machine Learning, particularly (near) real- time learning, in order to dynamically adapt resources to the immediate requirements of the virtual network functions, while minimizing performance degradations to fulfil SLA requirements. Moreover, this deliverable introduced the associated information model which focuses on illustrating the essential information captured within the building blocks of the CogNet architecture. To support the use cases of CogNet, facilitate the identification of specific research questions and illustrate a significant impact in a real-life context, a set of scenarios were identified and presented in D2.1. The selection of the final CogNet scenarios depended on the available data and specific interest and ongoing work within the project. Each scenario was illustrated further by the sequence diagrams which show how the scenarios will be implemented based on the architectural blocks and their communication, together with their technical and business requirements. The technical requirements were identified and described in detail and the value proposition model was applied for each of the scenarios. Furthermore, to achieve the overall goal of CogNet, this deliverable presented the portfolio of CogNet services relevant to network management. The techniques de-

veloped within the other core work packages aim to deliver these services where ML techniques will be applied in some cases to address the identified 5G network challenges. It is intended that the deliverable will not only provide the groundwork and requirements for the integration and validation work package but also motivate the core contribution of other work packages in CogNet. Based on the final scenarios, architecture and portfolio of services, CogNet will propose candidate solutions to address the challenges identified in the project. These solutions will be then evaluated and validated on the basis of the given technical and business requirements. Finally, the presented CogNet architecture serves as a supporting piece for the common view on 5G architecture in [118] of the EU 5G PPP to guide the design of architecture in the 5G era. The CogNet architecture and information model aim to cover broad scenarios and ongoing implementation research activities around heterogeneous CogNet services portfolio. The architecture is being implemented and applied in scenarios of the project, such as the Noisy Neighbour and Media SLA. Their outputs will be investigated to validate the solution in the near future.

Chapitre 5

Proposal : Cognitive Smart Engine

The wise is one only. It is unwilling and willing to be called by the name of Zeus.

Heraclides Ponticus

Contents

I	**Introduction**		**146**
II	**CSE Algorithms**		**149**
	A	Anticipation and forecasting	150
	B	Classification	157
III	**Model selection**		**171**
	A	Problem Formulation and Choices	171
	B	Search Methods	173
	C	The hyperparameter Search Space	177
	D	Research Methodology	182
	E	Data	183
	F	Meta-Learning	188
	G	feature relevance	193
IV	**Conclusion**		**195**

In this chapter presents the cognitive framework for SLA violation anticipaton in Programmable Networks. In this approach, we aim to combine multiple Machine Learning and data preprocessing methods for optimal violation prediction. We present the our two use cases and the end-to-end methodology to manage them. we then recollect the results of all the different SLA management tasks. We present the methodologies that we adopt for getting the results. We then provide the interpretation of the results and stress our key findings.

We structure this chapter in two main sections. Section I discusses classification and prediction, section II presents the model selection module and summarizes all the results.

I Introduction

The previous chapter introduced the global architecture for SLA management in software-based networks and the data analysis processes SLA anticipation and SLA violation identification. The previous chapter presents all the components of the architecture. The most important building block of the architecture is the CSE. In this chapter we will zoom on the CSE, the different algorithms used and how we trained them for effective SLA management.

The CSE performs three main tasks (Figure 5.1), (1) Anticipation of SLO violation (2) Identification when an SLO breach occurs and (3) model selection, i.e. selecting the most suitable ML algorithm for a given task. The SLA enforcement module is implemented but statically using ”if - then” rules.

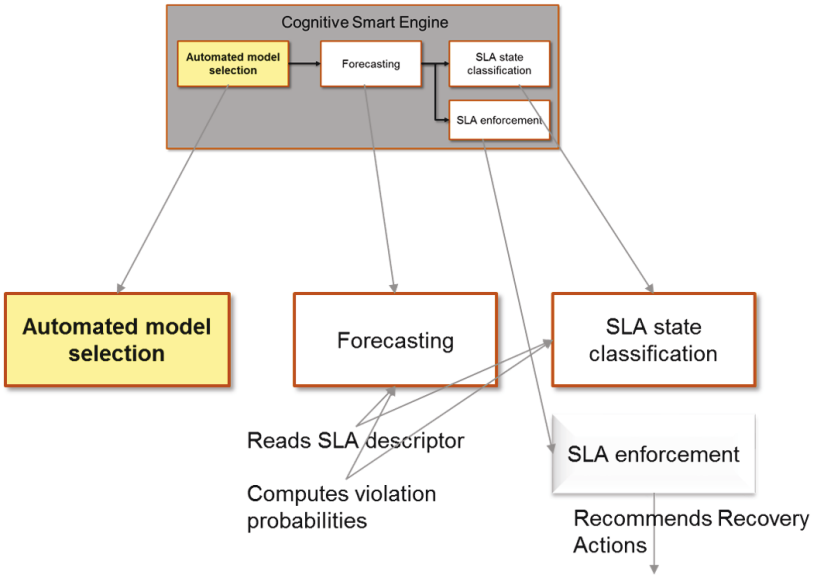


Figure 5.1 – Zoom on the Main building blocks of the CSE.

Early Methodology

In the early phase of our work, we collected data from the streaming use case. We trained our Machine Learning algorithms on the collected data. The model selection experimentations were done iterativaly and empiraclly until we obtain good results. In the testing phase, we split the original data into a training set (usualy 80% of the dataset) and testing set 20% of the dataset. We trained on the training set and note the results on our test set. Initially, we started with a single trial in our experimentations. Each algorithm was test one time. We train our algorithms on CPUs in the local machine.

Advanced Methodology

Through out this study, we have been using the first methodology to collect results and assess the performance and accuracy of our algorithlms. However this approach suffered from two drawbacks. First, the results and performance were not stable with many outliers, depending on the system usage. Second, even when the results showed that there is not overfitting, later results showed that some algorithms overfits the data. Hence, the need for a new methodology for our study.

In the advanced method, testing an algorithm is called an experiment. Each experiment has a number of trials between 3 and 20. A trial refers to a given algorithm tested on the same dataset. The output of the experiment is represented as a boxplot. a boxplot is a visual representation to summaries the output results. The boxplot as presented in figure 5.2 represents the distribution and range of the data. The data are ordered from the smallest (left or down) to the highest (up or right). The median (the data point that splits the data into two equal groups : lower half and upper half) is represented as the vertical line As shown in figure 5.2. The box represents the data by quartiles. A quartile is a quantile that divides the data into three parts. The first quartile (Q1) is defined as the median of the lower half. The second quartile (Q2) is the median. The third quartile (Q3) is the median of the upper half.

The outliers are defined as the data point that are beyond the lower and upper fence.

$Lower fence = Q1 - 1.5(IQR)$

$Upperfence = Q3 + 1.5(IQR)$

,with $IQR = Q3 - Q1$

The outliers may need additional analysis and do not represent the nominal behavior of the system. In this methodology we rely on the median as a summary and overview of the algorithm performance and accuracy. We also use the boxplot to compare the record high and record low of each experiment. Finally the length of the boxplot is also taken into account for the stability of the algorithms. algorithms with short boxplot are prefered for there result consistency

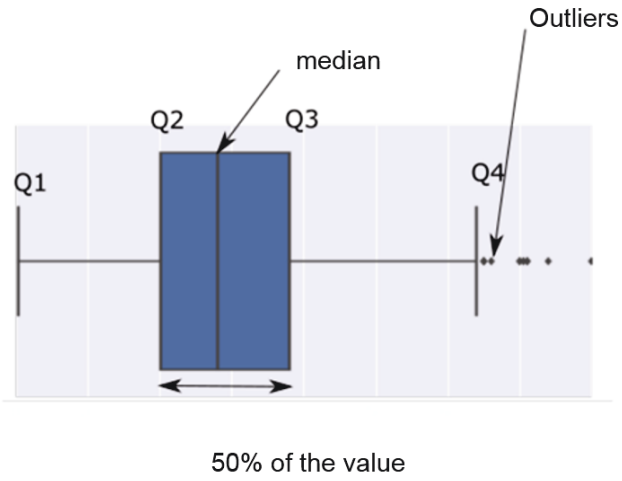


FIGURE 5.2 – Picture of a boxplot.

In the second phase of our work ; we collected data from the vIMS use case, an open source VNF. The data contain both normal and abnormal SLA. The abnormal SLA is divided into 3 types of SLOs. The Machine Learning algorithms trained on all the observed SLO violations. The model selection module is performed automatically using a random process. In the testing phase, we split the data into a training set, test set and validation set. with respectively 70, 20, 10 percent. The comparison between the validation and test set allow us to judge and compare more efficiently the performance of our algorithms. The training was performed on the cloud on both GPU and CPU machines. Because the testing in this phase is dynamic and automatic we collected a huge amount of data. The analysis in this phase was done using scripts and Machine Learning algorithms.

II CSE Algorithms

In this PhD work we have targeted real-time and live applications with highly heterogeneous SLA requirements in terms of QoS, restrictive network conditions, changeable bandwidth, latency, jitter and thresholds for error resilience. In terms of data streams, estimating the network capacity even for the near future is challenging. Inaccurate estimates can lead to degraded QoS. If network capacity is underestimated, the end point will receive the data in a worse condition basis, even though the current network condition allows a higher quality to be delivered. On the contrary, if it is overestimated the end point requests a bit rate greater than network capacity blocking the client processing with waits. The CSE focusing on the SLA management is illustrated by the sequence diagram presented and Figure 5.3 from the ML process perspective.

Firstly, the monitoring phase takes place where the Data Collector retrieves metrics from the NFV components of the architecture (MANO, NFVi, etc.). The CSE queries the metrics data from the Data Collector and further processes the received data. The CSE applies forecasting techniques in order to detect potential SLO violations on the forecasted values. If one is detected, the CSE SLA enforcer is notified and it will compute proactive management actions in order to avoid the SLO breach. In this regards, the CSE reports an event to the Policy Engine which checks the rules and actions associated with the SLA enforcement demo. In order to apply the actions, the Policy Engine requests the topology from the NFVO and it forwards the request of (de)allocation of resources to the VIM and to setup the NFV Infrastructure topology to the NFVO and SDN controller. The latter further initiates the setup of the NFV. Next, the dashboard receives the performance report.

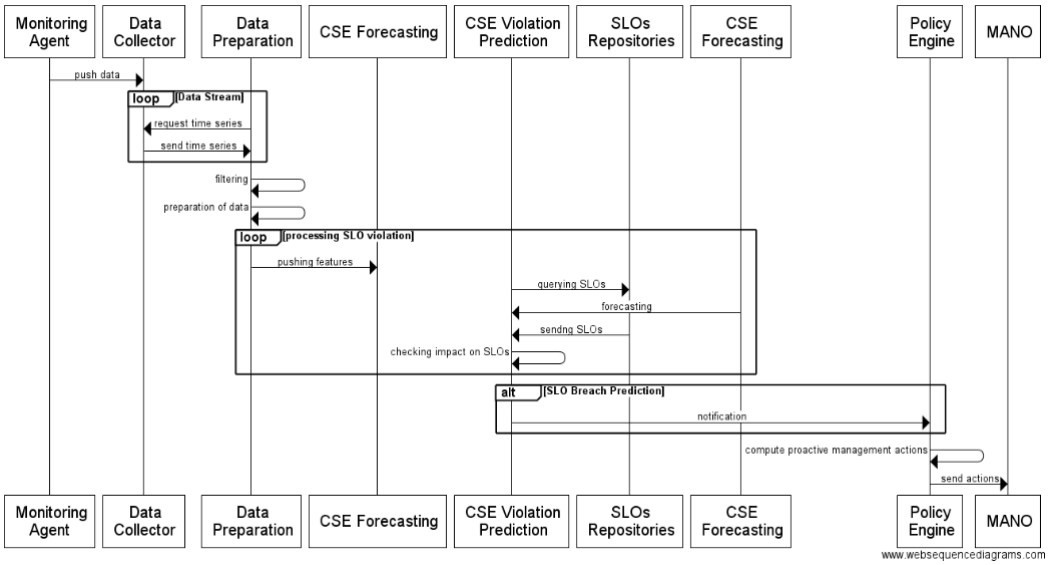


FIGURE 5.3 – Media SLA sequence diagram, focusing on the ML process.

The Cognitive Smart Engine (CSE) contains two stages. Anticipation and identification. The anticipation stage refers to prediction process that based on historical and contextual data to predict the evolution of a set of metrics, thus anticipating and preparing for the impact of the SLA. In the next section, we will explore the anticipation task and how to train ML algorithm for anticipating SLO violations.

A Anticipation and forecasting

One of the most challenging tasks in Machine Learning is predicting the evolution of a sets of metrics. In this section, we focused on this task by using two types of ANNs, nameley, the FeedForward Neural Network (FFNN) and the Long Short Term Memory (LSTM). We invistigate the use of shallow ANN and ANNs with multiple layers. Moreover, we will explore in details :

— The methodology and a step-by-step process to training an ANN and allowing for convergence.

— Illustrate how one can leverage FFNN and LSTM to predict SLO violations.

— Show the impact of tuning ANN parameters on the prediction accuracy and on the performance (i.e. training time).

— Discuss the difference between FFNN and LSTM for the anticipation.

— How to include ANNs trained for forecasting in an end-to-end framework.

In this work we define a *time series* as a sequence of observations ordered in time at regular intervals $s_t \in \Re$ e.g. meteorology weather variables, electric loads, a VNFs memory usage over time. A *model* as is a mathematical function that describes the process by which time series are generated. *Forecasting* is the process by which we apply a model to time series to compute a predicted output. The error between the predicted output and the real output is : $\hat{s_{t+1}} = g(s_t, s_{t-1}, ..., s_{t-n}) + \epsilon for t = 1, ..., T$. Where function g is the model, s_{t+1} is the predicted value and T is the number of observations and ϵ the error terms governed by a probability law. The prediction problem is to find the model g so that $\hat{s_{t+1}}$ is the closest to s_{t+1}. Moreover, the following assumptions at time t are considered :

— Recurrent patterns exist within each metric.

— There exists correlation between different metrics and other more abstract features in the network (e.g. CPU, network utilization).

— Network centric metrics can affect the VNFs service quality and/or the whole end-to-end service.

— The service quality metrics can be defined from one or many available metrics (e.g. maximum, percentile, average, etc.) or from service-centric KPI such as SIP connections, SIP client, etc.

preprocessing. The main technique forecasting preprocess is the autocorrelation. Autocorrelation is key for two reasons. First, it allows us to detect recurrent pattern in a given metric. Second, when provided an accuracy threshold, it computes exactly the maximal number of steps that we can use to predict the evolution of the signal. For example, in figure 4.22, the autocorrelation function shows that we can use the previous step to predict the next step with a precision exceeding 90%, and that we can use the second previous step with a precision of approxiamatly 80%. Unfortunately, in this example the maximal precision after the second step is 60%. Hence in this case we will limit the forecasting at the second step. The autocorrelation is important to apply before launching

the training, if the signal there is no correlation the learning algorithm may not converge or take an exponential convergence time.

Learning for forecasting. The next step is windowing. Windowing is a preprocessing step that is specific for time series. windowing is a technique that turns time series into machine leanring problems. In time series we have just a list of measurements by timestamp. if the window length is W and the observed data size is N then we have $N - W + 1$ potential windows. Windowing is important in the context of service monitoring. Often, the decision based on time series forecasting depends on the current input at time t plus some additional observations.

We can not use all the windows possible because of the high potential data points and the risk of overfitting due to the overlaping phenemenon. The window selection will impact the validity and the reusability of the results. Once we detect using the autocorrelation technique the recurrent pattern in the target metrics and the necessary steps to predict the evolution of the signal, we prepare the observed data into a training data. The idea during this phase is to 'label' each point in time t with its subsequent observed time $t + n$, with n as the 'look back' given by the autocorrelation function. This results in two series lagged by $n - step$, namely X, Y.The main difference between the prediction task and the classical supervized machine learning tasks is that time series adds the complexity of the sequence dependence among the variables. For this reason, We trained two ANNs types, an Feedforward Neural Network (FFNN) and a Recurrent Neural Network (RNN) of type Long Short Term Memory (LSTM). We variate manually the ANNs hyperparameters and training epochs. An epoch corresponds to the FFNN runs through all the training data set. We used Google's TensorFlow library for the ANN design and training. Another approach to the forecasting is that instead of labeling $x(t)$ with $x(t + 1)$, we label $x(t)$ with the next event, 0 for the nominal behavior and 1 if we observe that in the next step an anomaly or SLO violation.

Learning to predict. The objective is to infer the future behaviour of the service and of the underlying network at time $t + 1$ 1 (i.e. in the next step) of multiple metric. For example, we consider at time $t + 1$ a

forecasted Matrix Y defined as :

$$Y = \begin{cases} X^1(t+1) + \epsilon_1 \\ X^2(t+1) + \epsilon_2 \\ X^3(t+1) + \epsilon_3 \\ X^4(t+1) + \epsilon_4 \end{cases} \tag{5.1}$$

Prediction requires a supervised learning approach. The supervised learning means the labelisation of input data, each entry has a corresponding label. For timeseries, in order to learn the relation between the past and the future, we set as inputs and the target the same graph but shifted by one step. The ML algorithm then computes the optimal cost function between the past and the future. The resulting model, i.e. nodes coefficients represents how we can derive $X(t+1)$ from $X(t)$. Then we feed the four metric to a FFNN as shown in Figure 5.4. In this example, the four metrics can be derived from PCA analysis, Feature Engineering approach or using all the available metrics. The idea in this case is to predict the evolution of the system state using multiple low-level metrics.

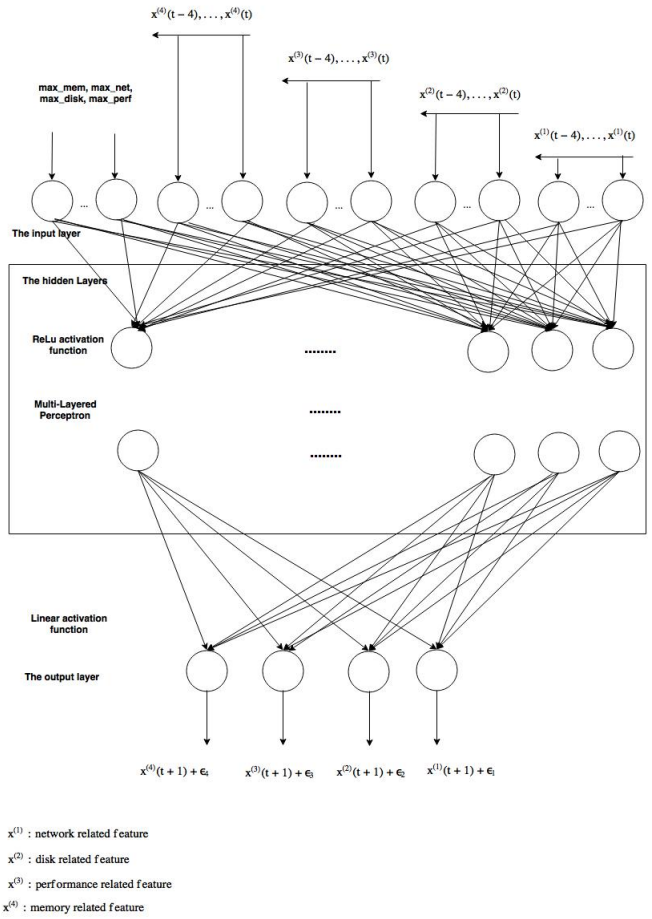


FIGURE 5.4 – FFNN architecture used for forecasting on 4 features following equation 5.1.

RNN/LSTM. Among the strengths of the RNN is that it manages dynamic input vectors. One vector for each time-step, each vector with many columns. For the SLA violation anticipation case, we used LSTM as "Many-to-one" (i.e. Input organized as sequence with one output) architecture. With multiple inputs given rise to one prediction. The inputed data for both the LSTM and the FFNN are represented according to three parameters : the mini-batch size - i.e. the chunks size of the inputs -, the number of metrics per time-step and the number of time-steps. The Figure 5.5 shows the difference in how RNNs manage their inputs. RNNs can accept timeseries with different sequenece legnths.

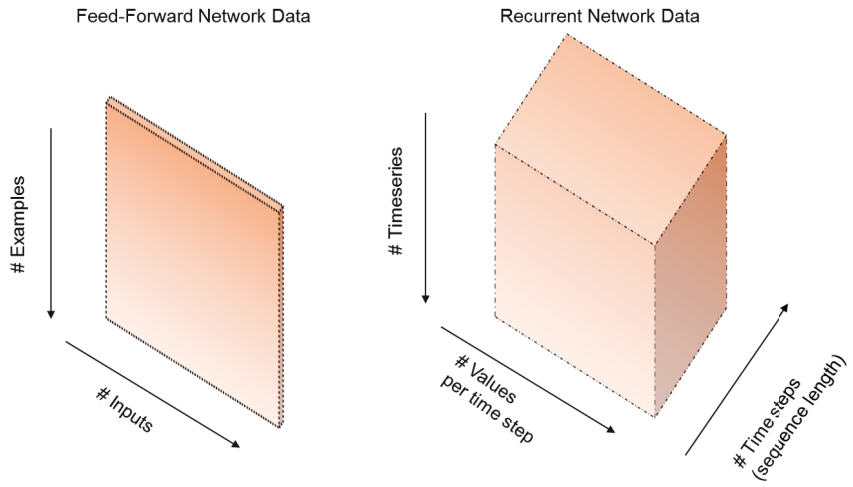


FIGURE 5.5 – Normal inputs vector vs recurrent neural network inputs.

We divide the data into testing and training data with a proportion of 70%, 30% respectively. The ANN training uses backpropagation algorithm to optimize the neural weights finding the coefficients that captures the best relationship between the past and the future. The learned model is stored as a 2D-matrix for the online phase. We initialized the standard LSTM parameters with 60 cycles (i.e. the data passed through all the nodes) and 10 epochs per cycle (i.e. one epoch is when the LSTM sees all the training data). The standard LSTM uses a batch size (i.e. input size) of one.

Results

We summarized our results in two tables. Table 5.1 represents our tests on the LSTM architecture in training 2 months, 20.000 data points, using Ubuntu $64bit$ VM, with $8GB$, $16GHzCPU$ for the video streaming scenario. Table 5.1 shows that the LSTM can take an extremely long time for training with only small improvement on the RMSE metric. One important parameter in our test is the batch size. In the literature, we found no clear methodology or consensus to determine the appropriate size of the input layer. However, we insist here that this parameter is the most important in the cognitive engine, since it can capture the hidden correlation in the stream.In Table 5.1, a batch size of one yields the best results over batches of 50 and 10 respectively. Overall, the ANN training

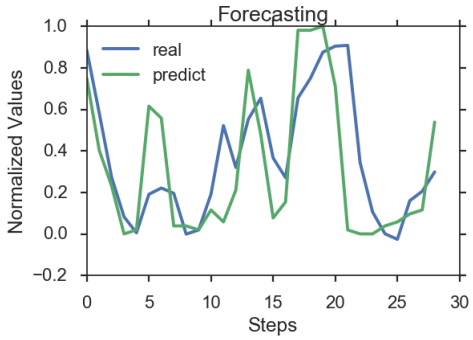


FIGURE 5.6 – Stacked Stateful LSTMs Trained for prediction

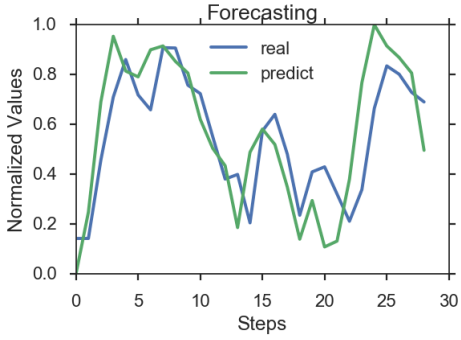


FIGURE 5.7 – Stacked FFNN for predicting

FIGURE 5.8 – Difference between FFNN and LSTM for signal prediction.

time is time consuming and is highly correlated with its architecture especially the number of cells in the hidden layer. Moreover, we noticed that after a certain RMSE value adding LSTM cells does not significantly improve the forecasting accuracy capabilities and can backfire into overfitting i.e. difficulty to generalize to unseen data.

TABLE 5.1 – Comparison of results yielded from different LSTM & FFNN architectures.

Batch size	Epochs	Layers	Training time	RMSE
1	5	1	2min12sec	3.95
1	60	4	22min54sec	3.82
1	500	4	3h19min12sec	3.58
10	5	1	2min26sec	10.62
10	60	4	2min12sec	11.51
10	500	4	16min43sec	17.38
50	5	1	37sec	12.01
50	60	4	1min03sec	11.50
50	500	4	4min10sec	11.48

The Figure A shows how forecasting values (green curve in Figure A) follow the real-time values (green curve in Figure A) for both LSTM and FFNN. Note that the general forecasting model follows the pattern and the evolution of the graph but not the exact values. This is the result of the pre-processing stage where inputs were prepared as the variation between values. This prevents the model from overfitting by waiting for the first value and computing its evolution. Moreover, we notice a clear

distinction between the LSTM and FFNN forecasting power. LSTM is much stronger but with a higher risk of overfitting.

Finally, the table 5.2 summarizes the RMSE comparison between the LSTM and FFNN over both use case and using different preprocessing methods. As noticed in Figure A, the best LSTM model outperforms the best FFNN.

method	Use case I			Use case II		
	FFNN	LSTM	DT	FFNN	LSTM	DT
PCA	5.69	3.95	-	6.60	4.42	-
FE	7.08	5.32	-	7.04	4.05	-
ALL	13.85	11.54	-	15.96	10.11	-

Table 5.2 – RMSE comparison

We have based our work on substantial related work done in the area of ANN for regression and forecasting, SLA enforcement in the cloud environment. In this paper, we defined and evaluated modules that allow the forecasting and prediction of SLOs breaches for a VOIP service running on an NFV and SDN infrastructure. Our next step is to close the control loop with a sophisticated policy engine (CSE SLA enforcer) that pushes proactive management actions according to different policies, reducing the number of SLO breaches as they occur.

B Classification

Classification is one of the most common task in Supervized Machine Learning. In this section, we present an experimental comparison between various neural network architectures on a SLA violation classi

cation task. This classification task tests an algorithm's ability to segment and recognise the constituent parts of a signal, and requires the use of contextual information. Context is of particular importance in SLA violation detection due to phenomena such as co-sub-service dependencies, where the application system depends on the services provided by other subsystems. In many cases it is difficult to identify a particular SLA violation frame without knowing what occur before and after it. The main result of this chapter is that network architectures capable of accessing more context give better performance in SLA violation classi

cation, and are therefore more suitable for identification and reporting. In the section before, we used LSTM and FFNN algorithms. In this section we re-used these algorithms for classification and we introduce a Decision Tree (DT) algorithm. We focus on this section solely on the second use case, namely the vIMS Clearwater

Because classification is a supervized ML technique, labeling is a critical phase. For this reason, we will detail it in the next section.

Data Labeling

As mentioned before, the labels in our case are the SLOs. Our three SLOs target respectively, the response time of the service, The availability of the SIP proxy and the database transaction. The role of the ML algorithm is to identify from inputed observations the relevant metrics to track for each SLO and to identify effectively when an SLO breach is occurring.

As introduced in previous sections, SLA comprises of a set of measurable, low-level objectives that are contracted with the service consumer. We define the three SLOs that will be used for the Clearwater use case as follows :

— SLO_1 *response time* : This service objective targets the response time of the SIP proxy VNFC (i.e. Bono). The overload tests generate a large amount of connections forcing the proxy to drop multiple connection requests. This in turn, reduces the mean time necessary to answer a connection request.

— SLO_2 *database transaction* : This SLO targets mainly Ralf and Homestead node. The SLO in this context is defined as the Cassandra database performance. We observed the behavior of the database and labeled the data accordingly.

— SLO_3 *service availabiltiy* : This SLO monitors the behavior of the service as a whole. When the communication service is down we label the corresponding dummy variable with 1, 0 otherwise.

We depict the representation of the SLO_1 as an example as a black line in Figures 5.9. The SLO violation state is represented by the non-null value in black color, while the compliant state is set to 0.

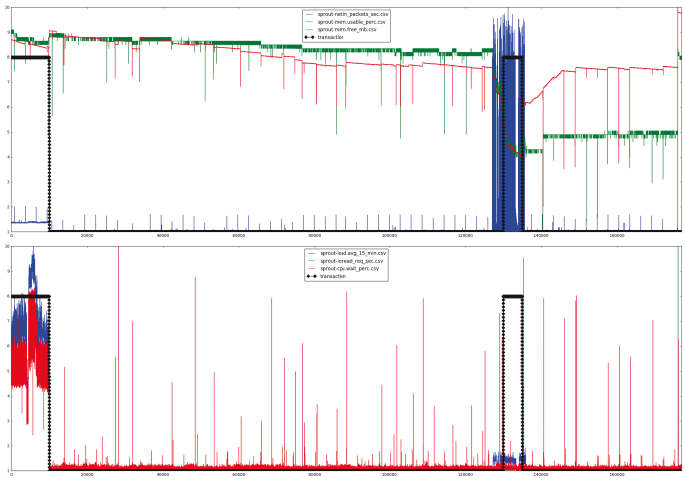


FIGURE 5.9 – SLO_1 targeting the Sprout VNFC.

SLA violation generation. We proceed in our performance degradation by incrementing linearly and abruptly the number of SIP connexions until the service fails. Each time, we targeted different service components of the vIMS service. Then, we noted the specific timestamps corresponding to each SLO violation. Finally, we connect the SLO violations with the observed monitored data as a global labeled matrix (see Figure 5.10), where each line correspond to the observed monitoring data and its SLOs labels. The SLO violation is set to 1, whereas the compliant state line is set to 0.

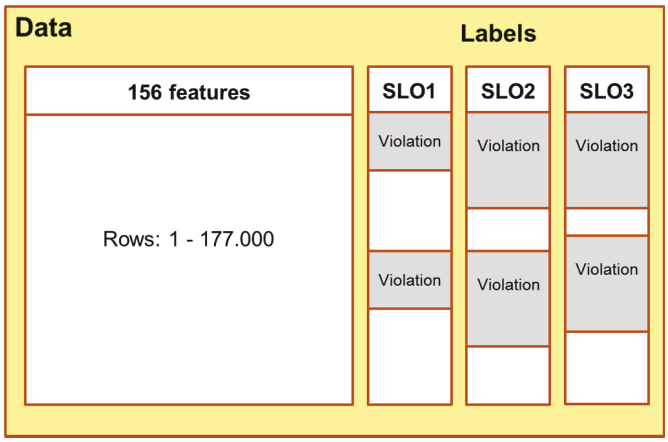


FIGURE 5.10 – Layout of the dataset used in Use case II.

We formalized our approach as a supervised machine learning technique, which is for each entry line composed of all the metrics we associate a binary value that labeled the line (i.e. the set of metrics va-

lues). We used the mathematical formalization of the SLA and SLO in Section C, that describe the SLA violation as at least one SLOV and the functions and that maps low-level metrics to the SLO as shown in Table 5.10. The definition of the functions can be defined empirically or found using machine learning techniques that map the observed SLOs to the variation of the metrics. Labelling each row with a binary variable will result is a vector Y (see Eq. 5.2) with value {0,1} that maps all the input matrix M of 156 features (see Eq. 5.3). This method allow the ANN to learn how to classify the data into two SLA violation states.

$$[ht]Y = \begin{bmatrix} Y_1 & Y_2 & Y_3 & ... & Y_m \end{bmatrix}^T \tag{5.2}$$

$$[ht]X = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & ... & x_{1,156} \\ x_{2,1} & x_{2,2} & x_{2,3} & ... & x_{2,156} \\ x_{3,1} & x_{3,2} & x_{3,3} & ... & x_{3,156} \\ x_{4,1} & x_{4,2} & x_{4,3} & ... & x_{4,156} \\ & & \cdots & & \\ x_{m,1} & x_{m,2} & x_{m,3} & ... & x_{m,156} \end{bmatrix} \tag{5.3}$$

It is worth noting that it is necessary to use one Neural Network per Violation, in other terms the weights that we find for this problem is specific to the SLO violation that is learned. Keeping the labels at their default position will result in an ANN that can identify the SLOV but not anticipate them. In order to learn to predict the SLOV the labels should be shifted back from their position, using this technique we can at the same time predict and classify SLO Violations.

We used the following
algorithms in our experiments :

— FFNN 1 (see Figure 5.11 and 5.13) : Classical FFNN with 10 hidden layer, sigmoid as the last activation layer and a learning rate α = 0.01, training epochs : 200

— FFNN 2 : Classical FFNN with 10 hidden layer, sigmoid as the last activation layer and a learning rate $\alpha = 0.01$, training epochs : 300

— LSTM 1 : , with one hidden LSTM layer, containing 140 one-cell memory blocks, training epochs : 300

— LSTM 2 : Unidirectional RNN with one hidden layer containing 275 sigmoid units, trained with target delays from 0 to 10 frames (RNN), training epochs : 500

— MLP + DT : Classical FFNN with 10 hidden layer, sigmoid as the last activation layer and a learning rate $\alpha = 0.01$, training epochs : 120, see Figure 5.12, for this algorithm we combined an FFNN with a DT algorithm, the FFNN finds the next step ahead values and transfer them to a DT that classifies them.

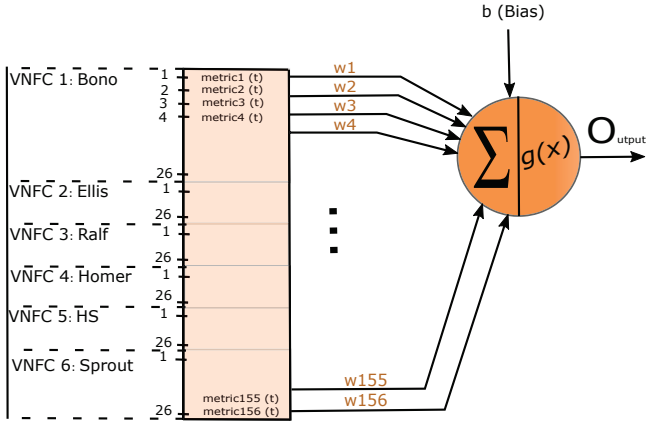


FIGURE 5.11 – The activation function $g(x)$ - if non-linear - applied to the output of the neuron allows the ANN to behave as a universal approximator by introducing non-linearity.

Note however that for the MLPs the network grew with the time-window size, and W ranged from 22,061 to 152,061. All networks contained an input layer of size 26 (one for each MFCC coecient), and an output layer of size 61 (one for each phoneme). The input layers were fully connected to the hidden layers and the hidden layers were fully connected to the output layers. For the recurrent networks, the hidden layers were also fully connected to themselves. The LSTM blocks had the following activation functions : logistic sigmoids in the range [-3; 1] for the input and output activation functions of the cell (g and h in Figure 4.2), and in the range [0; 1] for the gates. The non-LSTM networks had logistic sigmoid activations in the range [0; 1] in the hidden layers. All units were biased.

For all networks, the computational complexity was dominated by the O(W) feedforward and feedback operations. This means that the bi-

directional networks and the LSTM networks did not take signi

cantly more time per training epoch than the unidirectional or RNN or (equivalently sized) MLP networks.

For the output layers, we used the cross entropy error function and the softmax activation function, as discussed in Sections 3.1.2 and 3.1.3. The softmax function ensures that the network outputs are all between zero and one, and that they sum to one on every timestep. This means they can be interpreted as the posterior probabilities of the phonemes at a given frame, given all the inputs up to the current one (with unidirectional networks) or all the inputs in the whole sequence (with bidirectional networks).

We formulate this problem as a Multiclass Classification Decision Tree (MCDT). DT (Decision Tree) –see Fig. 5- is a representation of a problem in the form of a tree. Each leaf represents a dependent variable, i.e. SLOV. The non-leaf edges form subgroups with vertices that express conditions on independent variables, i.e. features.

We define K as the number of classes that is the number of SLOV plus one, represented in (5.4). Each SLOV is associated with a class as well as normal state (no SLOV) as specified in (5.5). For K classes

$$V = \{SLO_1, SLO_2, \ldots, SLO_K\} \tag{5.4}$$

Our objective is to learn to classify correctly the matrix Y into all the K classes.

$$f : Y \rightarrow \{SLO_1, SLO_2, \ldots, SLO_K\} \tag{5.5}$$

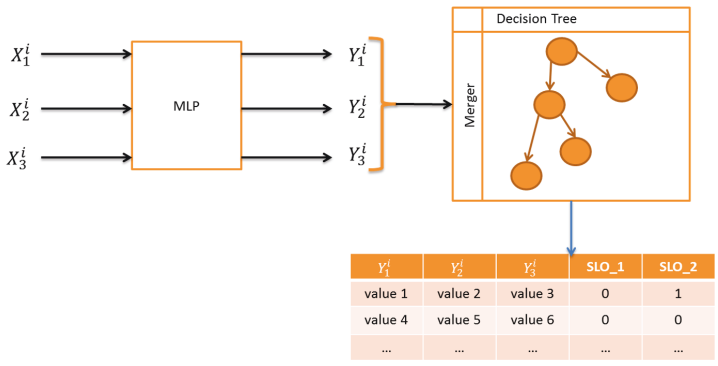


FIGURE 5.12 – Combined FFNN with Decision Tree (MLP-DT)

Network Training. For the LSTM architectures, we calculated the full error gradient using BPTT (Back Propagation Through Time) for each

utterance, and trained the weights using online steepest descent with momentum. We kept the same training parameters for all experiments : initial weights chosen from a at random distribution with range $[-1;1]$, a learning rate of 0.01 and a momentum of 0.9. As usual, weight updates were carried out at the end of each sequence, and the order of the training set was randomised at the start of each training epoch. Keeping the training algorithm and parameters constant allowed us to concentrate on the effect of varying the architecture. However it is possible that different training methods would be better suited to different networks.

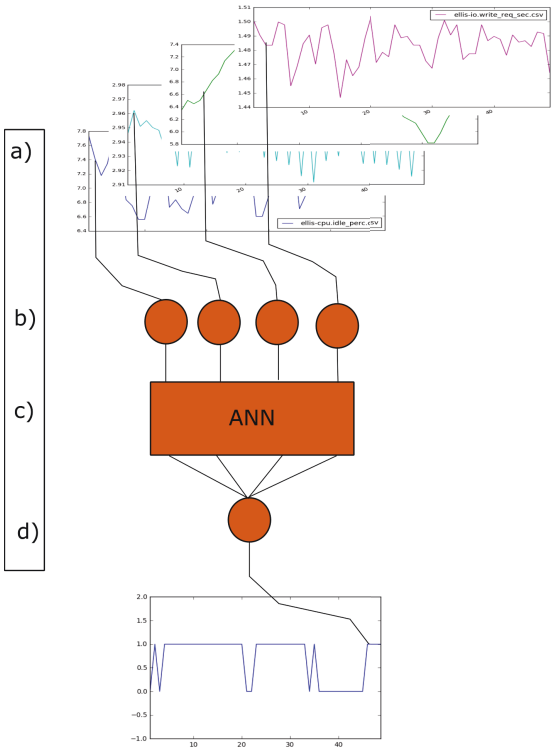


FIGURE 5.13 – A miniature representation of our experiments, where inputs are limited to 4 (we use 156) and output to one (we used 3). a) is the features represented as time series data. b) is the point of contact, the inputs fed to the ANN model. c) The ANN model which can be LSTM or FFNN. d) is the binary result that categories the inputs into 0 or 1 for non SLA violation and SLA violation respectivelly.

Results

We split the data set into three part, the training set at 70%, the test set at 20% and the validation set at 10%. We labeled the data based on the experiments that we run beforehand and the three SLO definition in section B. The results is a $N * 3$ matrix as the $D_{T\underline{Y}}$ in the equation 5.6.

We evaluate the ML algortihms over the test set. Figure 5.13 provides a simplified version of this process with only four inputs.

Table 5.3 – Classification results of the algorithms

Network	Accuracy	Recall	Precision
LSTM 1	0.95	0.93	0.95
LSTM 2	0.99	0.99	0.99
FFNN 1	0.87	0 .92	0.63
FFNN 2	0.77	0.80	0.69
MLP+DT	0.49	0.93	0.46

Table 5.3 and Figure 5.16 summarises the performance of the different network architectures. For the DT, FFNN and LSTM networks we give both the best results, and those achieved with least contextual information (i.e. with no target delay or time-window). The most obvious dierence between LSTM and the RNN and MLP networks was the number of epochs required for training, as shown in Figure 5.16. In particular, LSTM took more than 2 times as long to converge as the FFNN, despite having more or less equal computational complexity per timestep. There was a similar time increase between the unidirectional LSTM and RNN networks, and the MLPs were slower still (300 epochs for the best MLP result). A possible explanation for this is that the MLPs and RNNs require more

ne-tuning of their weights to access long range contextual information.

Confusion matrix. The confusion matrix depicted in Figure 5.18 represent an example of the best LSTM model (LSTM 2). The LSTM show particularly powerful for complex classification tasks, it even gets a 100% accuracy over SLO1 (depicted as SLO0). The confusion matrix clearly shows the details and missclassifcation errors the algorithm makes. The diagonal columns represent the correct classifcation and are hopefully the most dense area. This means that the algorithm successfully classifies most of the examples. Moreover, the Confusion matrix can provide insight into where one should direct more resources to improve classification accuracy. In this example, clearly the SLO1 classfication can be enhanced. This can be done via a new feature engineered metrics or by changing the algorithm configuration e.g. hidden layer, width, etc.

We used Decision Tree algorithm IR3, that takes as inputs the 4 ANN

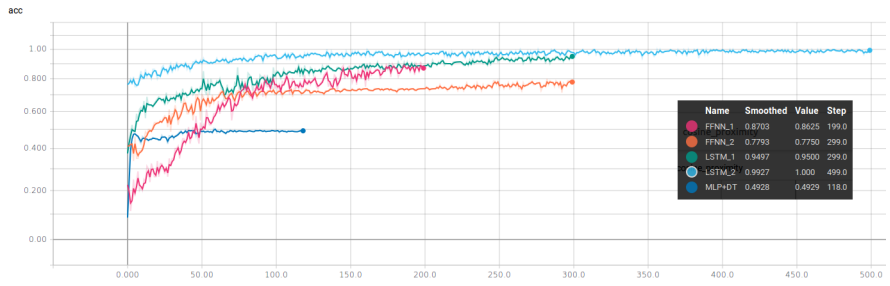


FIGURE 5.14 – Accuracy over all the training steps

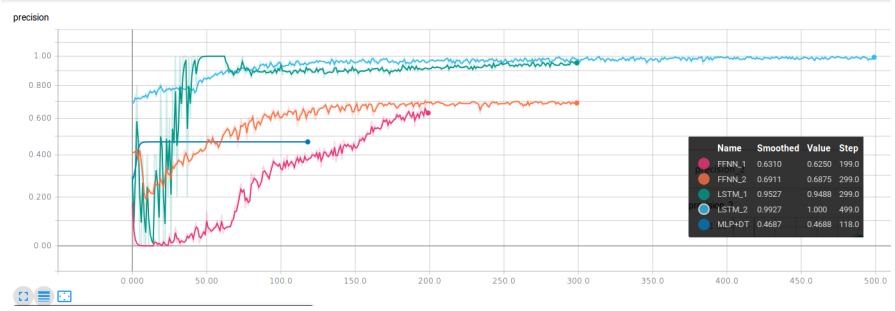


FIGURE 5.15 – Precision over all the training steps

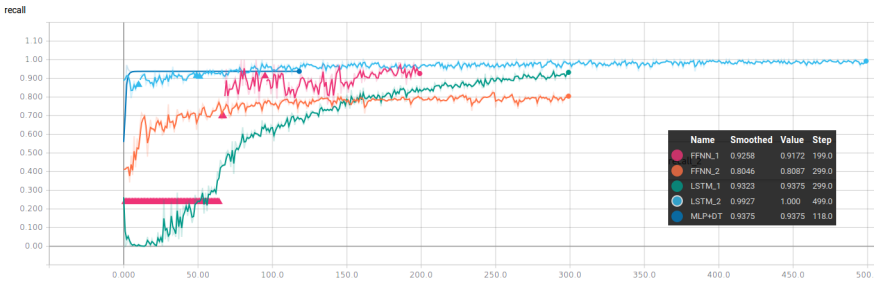


FIGURE 5.16 – Recall over all the training steps

FIGURE 5.17 – Precision, recall and accuracy over all the training steps

outputs (forecasts) and to classify them it into SLA classes (i.e. SLA violation, non-SLA violation). Table 5.4 shows the performance of our MCDT (MultiClass Decision Tree) implementation for the streaming video use case. The Table 5.4 shows the promising performance of the ID3 algorithm when applied to our prediction problem. The minimum accuracy (F-score) found is 0.843 that corresponds to binary classes (either SLOV or not). We progressively added SLOs to measure how the MCDT would handle different classes. The results show that the accuracy improves with additional SLO classes. This can be explained by the variation of the precision and recall metrics with respect to each new SLO class. Overall, the MCDT performs well on the testing set ; however

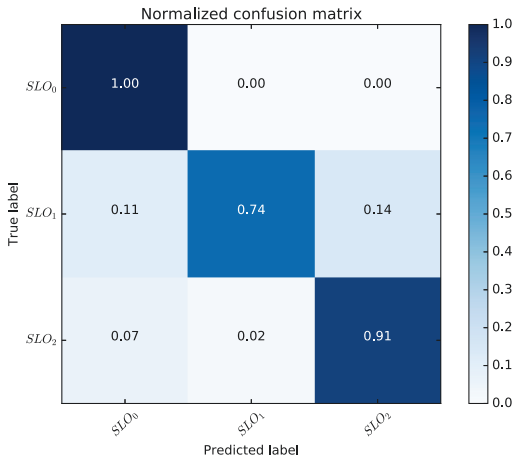


FIGURE 5.18 – Confusion matrix of the Best algorithm (LSTM2)

a more thorough testing is necessary over longer period of time and with more SLOs.

SLO Classes	# SLO lines	Precision	Recall	F-score
1	60	0.822	0.879	0.849
2	180	0.890	0.917	0.903
3	260	0.909	0.913	0.910

TABLE 5.4 – MCDT evaluation on the video streaming test set.

We initialized the LSTM parameters with 60 cycles (i.e. the data passed through all the nodes) and 10 epochs per cycle (i.e. one epoch is when the LSTM sees all the training data). Overall the training time of the ANN is considerable and it is highly correlated with the architecture especially, the number of nodes in the hidden layer. Moreover, we notice that after a certain RMSE value adding ANN node does not significantly improve the forecasting accuracy.

In Table 5.19, we classified our results according to the threshold used by the SLO violation module α. Recall that α is the threshold that determines whether a set of forecasted values are considered as violation SLOs or not. We present here for each α the three SLOs with their respective precision, recall and F-score evaluation metric. We then compute the mean for each metric to summarize the impact of α on all the SLOs. From Table 5.19 we draw two conclusions :

1. Our Framework has a high accuracy with respect to both precision

Prediction threshold	SLO_D	Precision	Recall	F-score
$\alpha = 0.7$	SLO 1	0.821	0.956	0.883
	SLO 2	0.919	0.951	0.934
	SLO 3	0.892	0.933	0.912
	Mean values	0.877	0.946	0.909
$\alpha = 0.8$	SLO 1	0.888	0.919	0.903
	SLO 2	0.941	0.923	0.931
	SLO 3	0.894	0.926	0.909
	Mean values	0.907	0.922	0.914
$\alpha = 0.9$	SLO 1	0.922	0.868	0.894
	SLO 2	0.957	0.889	0.921
	SLO 3	0.946	0.877	0.910
	Mean values	0.941	0.878	0.908

FIGURE 5.19 – Results of offline evaluation mode of the FFNN with three different SLO breach threshold (Based on the streaming use case)

and recall metrics. This is due to the large dataset used for the training with an important diversity in fault injection.

2. The probability threshold α is a key tuning parameter in our framework. The value associated with it determines if the operator gives more weight to the precision (i.e. maximize the confidence of predicted SLO breaches) or the recall metric (i.e. minimizing the risk of missing SLO breaches).

In other words, a small α means that the system will detect most SLO breaches but with a higher risk of false positive, whereas a higher α yields a higher precision with a larger false negative. Additionally, we classified our results in Table 5.19 according to the threshold used by the SLO violation module α to determine if a forecasted value is considered or not as impacting an SLO.

Precision/Recall tradeoff. With a high classification threshold (α = 0.9) the precision is up to 95%.

the main functional behavior of the SLA violation approach is the ability to forecast the evolution of a given network device/service/element.

Regularization : Becasue we have 100.000 lines overfitting becomes increasingly an issue. For this reason we deploy dropout technique and l_2 weight. In our results we find that dropout can reduce considerably overfitting allowing us to add more hidden layers to our model.

From a machine learning perspective, our evaluations show that LSTM is very robust when it comes to predicting a one step ahead in a sequence. In this work we considered only the data for streaming service and the underlying networks, one can consider a multi-tiered approach. This means the combination of data from different sources e.g. social media, TV, etc. to improve accuracy of prediction.

method	Use case I			Use case II		
	FFNN	LSTM	DT	FFNN	LSTM	DT
PCA	0.84	0.88	0.72	0.72	0.77	0.33
FE	0.75	0.85	0.62	0.45	0.73	0.21
ALL	0.90	0.90	0.86	0.63	0.99	0.46

Table 5.5 – Precision

method	Use case I			Use case II		
	FFNN	LSTM	DT	FFNN	LSTM	DT
PCA	0.84	0.82	0.82	0.66	0.56	0.45
FE	0.65	0.49	0.55	0.23	0.65	0.46
ALL	0.92	0.0	0.0	0.92	0.99	0.93

Table 5.6 – Recall

We trained multiple LSTMs architecture on the training set. Then we performed our evaluation on the testing set. The test set is considered as unseen/new data. The results are summarized in Table1. We summarized and discuss our results in section V-C.

GPU vs CPU performance. GPUs are essential for ANN training because these neural networks training computations can be efficiently parallelizable and computationaly independent. The FFNN model proposed in this study is not considered too deep compared to the state of the art Convolution Neural Networks, still the time necessary for training up to 500 hidden layer is long (i.e. up to 8 hours on standard server). The use of the GPU technology allowed us to perfrom such experiment is a much more acceptable time (i.e. from 8 hours to 1 hour for deep FFNNs). We notice however that GPUs are not very efficient when it

come to LSTM architectures. This can be explained by the sequential nature of these networks.

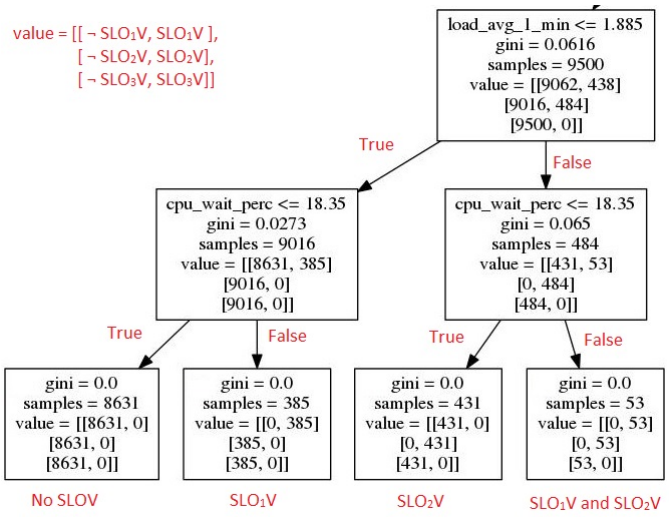


FIGURE 5.20 – Example of a subgraph in a Decision Tree over 10.000 sample.

We used ID3 (Iterative Dichotomiser) algorithm to build our DT. The ID3 algorithm separates reclusively the dataset into binary classes according to the feature with highest entropy until it converges. The advantage of DT compared to other classification technique is that they are considered as a white box (i.e. The internal mechanics is not too complex and can be understandable) and are less prone to the curse of dimensionality. The Figure 5.20 for example shows a subgraph that classifies 9500 samples into four leafs. The samples were classified by answering multiples predicates e.g. $load\underline{a}vg\underline{1}\underline{m}in < x$. The result is a two-dimension array, where each line represents an SLO and each row an SLO state : ¬ SLOV or SLOV, see Fig. 5 top left. The first leaf from the bottom left classified 8631 samples as ¬ SLOV. The second leaf classified 385 samples as SLO_1V. The third one classified 431 samples as SLO_2V. Finally, the last leaf represents 53 samples as SLO_1V and SLO_2V. This result can be reduced into only two classes : SLOV (i.e. every leaf with at least one SLOV) and non-SLOV (i.e. the remaining leaf). The *gini* variable in the Figure 5.20 is entropy. It decreases from the first level until reaching null value in all the leafs, which means that all samples are homogenous, i.e. classified.

Decision making : Execution In this example, we set the network management action as a corrective Openstack action that shuts down

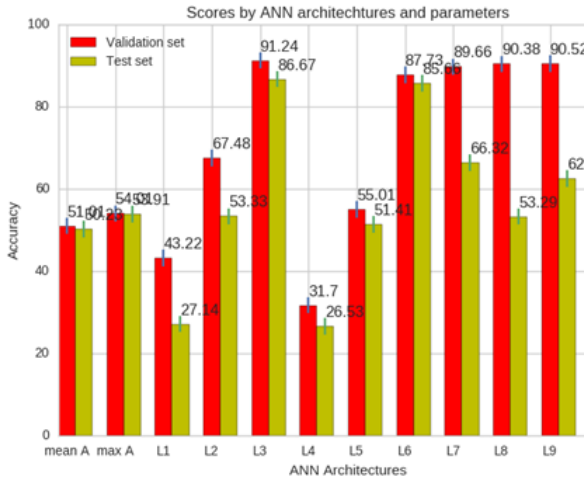


FIGURE 5.21 – Results comparing FFNN and LSTM based on the validation and test set.

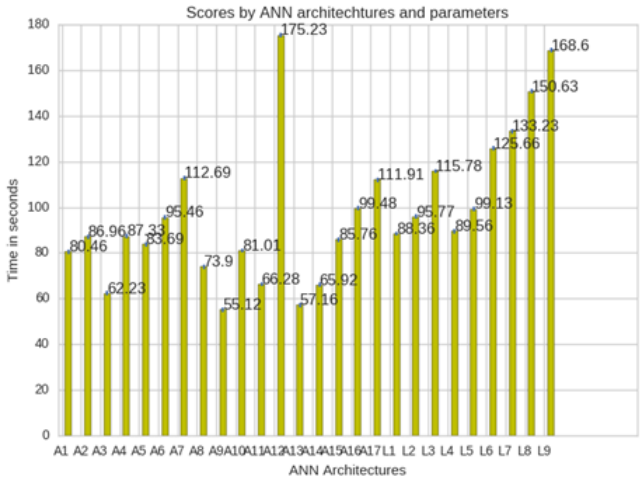


FIGURE 5.22 – The Training time of the FFNN for 10,000 samples.

FIGURE 5.23 – Overall performance of FFNNs vs LSTMs (The model code are in the index section B)

the VM responsible for the overload. The corrective action uses Openstack Nova API. Nova API service allows the programmability of the Cloud compute service. As an example, we demonstrate during the demo how our system reacts to drop on the service quality by predicting the evolution of network input metric *net.inpacketssec*.

Figure B summarizes our findings. The difference between the FFNN and the LSTM is that the LSTM requires 433 times less epochs to converge to the optimal solution (between 660-1300 for the FFNN and only 3 for the LSTM). A possible explanation of these results is that the FFNN requires more epochs to optimize the weights in order to incorporate the contextual nature of the Time series. Moreover, the LSTM took approximatively the same mean time to train than the mean FFNN (Figure 5.22), however yields far better results for only few hidden layers with a highest score of 86.67%, whereas the FFNN only manages to 53%. The poor performance of the FFNN is due to its structure, even though we used Deep FFNN, the ANN couldn't update the weight is such a way to capture the sequential nature of the features. A. Graves et al. [179] found similar results with the exception that their LSTM configuration

needed significantly less training time that the FFNN. We can conclude that a wisely selected LSTMs hyper-parameters and structure is the best approach to the SLOV problems.

Another remark is that The LSTM shows particularly prone to overfitting (Figure 16). Contrarily to the FFNN, the dropout in the LSTM cells and the difference between the test set and validation set is substantial. For after only 10 epochs, the LSTM results show a significant difference between the validation and test set (Figure 16, L7, L8, L9). A recent study on handwriting classification by V. Pham et al. [180], showed in details the importance of LSTM Dropouts to improve results and reduce overfitting.

For the FFNN, the ReLu function does not allow the ANN to converge for classification problems contrarily to the forecasting [181], sigmoid, linear and tanh showed all similar performance and results. This is explained by the mathematical characteristics of the ReLu function which do not squash the outputs to a limited range (see Figure activation functions). Also, using Deep FFNN only increases training time without improving significantly the overall accuracy. It appears that even deep neural net couldn't successfully captures the contextual information. The deep neural net is limited in term of the input dimension, this is a significant limitation, since that our problem requires no a-priory knowledge. A study in [14] raised this problematic for deep neural net for sequenced data and showed that the LSTM is a sensible alternative.

III Model selection

In this section we zoom on the model selection module of the CSE module.

A Problem Formulation and Choices

In supervised ML, models are built based on a given data set denoted further as D. More specifically, the Learning Algorithm LA aims at learning the relationship between the training set $D_{X\underline{T}}$, where $D_{X\underline{T}} \subset D$ and the labels also referred to as target $D_{Y\underline{T}}$. This operation is performed by minimizing the Loss function $Loss$. Each Learning Algorithm LA has a

set of all hyperparameters denoted by $\lambda = \{\lambda^{(0)}, \lambda^{(1)}, \lambda^{(2)} ... \lambda^{(n)}\}$. These parameters configure the Learning Algorithm LA in n different ways. We denote λ^* be the optimal hyperparameter space for the training algorithm that reduces the error Err on the validation set. We write :

$$\lambda^* \equiv \mathbf{argmin}_\lambda \, Err_{x,x\subset D}[(Loss_{x,y}; LA_\lambda(D_{X\underline{T}}, D_{Y\underline{T}}))] \tag{5.6}$$

There are no straightforward processes that guarantee to find λ^*. However, there exist search methods to approximate the optimal solution. Up to our knowledge, λ^* can only be approximated empirically, using trial and errors in an iterative way.

We consider the distribution of D as known empirically from our data set (see Figure 4.18). In order to find the λ set that verifies the equation 5.6 we approached the problem as a search problem, were each point in the search space is a combination of all possible ANN parameters. We can simplify the function in equation 5.6 by replacing the term Err_x with γ and we write :

$$\lambda^* \equiv \mathbf{argmin}_\lambda \gamma_x(\lambda) \tag{5.7}$$

In equation 5.7, γ is known in the literature as the *hyperparameter response func*
The optimization of the hyperparameter over λ is equivalent to minimizing the function $\gamma_x(\lambda)$. The space defined by λ and the function γ is highly dependent of the machine learning algorithm understudy and to the data set used. For example, in this paper we are focusing on the ANN which sets a different λ space than would an SVM or a regression model.

The traditional search method that consists in trying all the possible options is unfeasible since the high number of hyperparameters will generate an explosion of trial number. For example, with $n = 10$ parameters and if each parameter can have 5 different values, we would require to search $5^{10} \approx 10 million$ different configurations. If the algorithm spends 5 *seconds* per configuration, this would take more than a year and a half of computation to complete.

$$SLA = \bigvee_0 \gamma(S_i), \; S_i =< V\vec{N}F, SLO_i > \tag{5.8}$$

The equation 5.8 is an interpreted as an SLA violation is ocurying if at leat one SLO is violated.

Figure 5.24 depicts the overall methodology. The problem is a multivariate time series classification. The aim of this methodology is to find the most suited ANN classifier by generating and training multiple ones. We compare the ANN performance on the validation set and select the one with best performance.

The overall methodology we are proposing, is presented in Figure 5.24. The details of the techniques and the developed algorithms are presented in the next section.

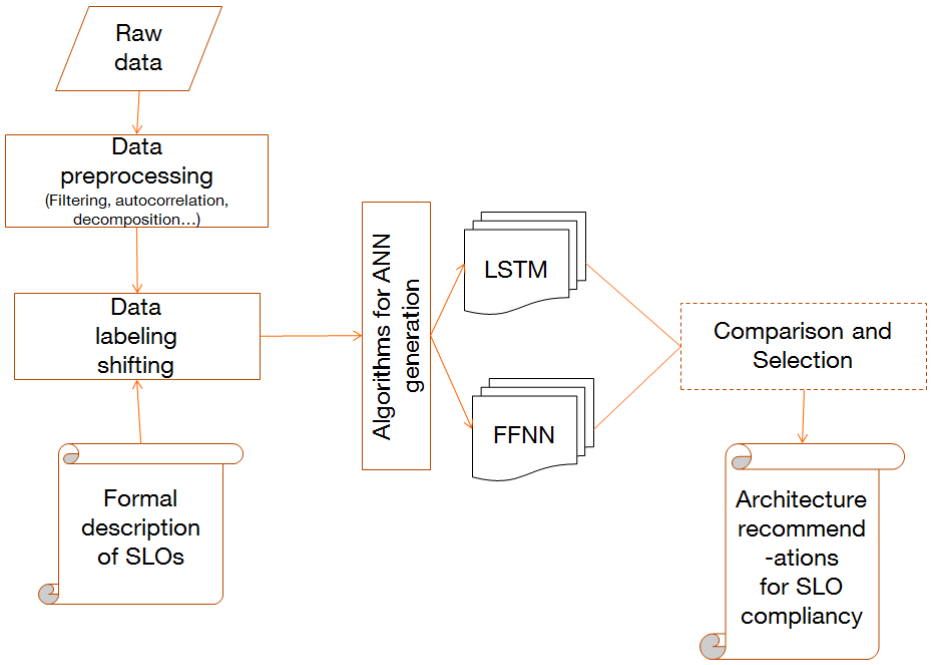


Figure 5.24 – Methodology.

B Search Methods

In ML we distinguish between two configuration variables : the variables that can be inferred from the data, this is termed the model parameter and the configuration variables that cannot be estimated from the data distribution, termed the hyperparameters. The hyperparameters are considered as an internal part of the ML models. They are generally set manually by the ML designer or require an intuition, using heuristics techniques. Since one cannot know the best hyperparameters a priori, many techniques have been developed to find the best parameters. Amongst all the techniques developed we focused in this study on

empirical approaches. In the literature, there exist four main search methods. Manual search, Grid search as proposed by Larochelle et al. [182], random search first used by Begstra et al. [183] and Bayesian search. Manual Search consists in relying on the intuition, the experience and some heuristics. It does not guarantee the optimal configuration space. It suffers from the fact that one should manually test models and thus limit the number of trials. Both grid and manual search are the most used search methods.

The second most used search strategy is the grid search. Grid search consists of two steps : Firstly, manually selecting a subset of hyperparameters values to reduce the size of the configuration space, and secondly, performing an exhaustive search over all the combination of these parameters. The Grid search combines in this way the operator intuition and the machine performance. The Grid search algorithm is typically directed by an accuracy metric using K-fold cross validation technique or accuracy over a validation set. The grid search was wildly adopted in the ML community due to the insight it provides for finding the optimal hyperparameters, it simplicity in operational settings as well as its performance in low-dimension spaces (i.e. when combining between two or three factor of variation (hyperparameter)). However, Grid Search suffers from the curse of dimensionality ; it does not scale well.

Random Search is a search strategy that draws randomly and uniformly from the hyperparameter configuration space. Bergstra et al. [183] showed that a simple random search performs much better in high-dimensional space than grid search. Moreover they demonstrated that the operator initial intuition in the grid search is not reliable since for each different problem and different data set different subspaces are important to different degrees.

Bernoulli Categorical Distribution.

The Bernoulli categorical distribution is a discrete probability distribution over multiple classes (in this case number of hidden layers) with specified probability for each class. In our case, the probability is biased towards ANNs with less than 10 layers. The categorical Bernoulli distribution allows us to bias the selection towards smaller ANNs. The idea is to generate diverse ANN structure but not too many Deep ANNs due to the high computing, memory and time cost.

Search methods	definition	+ +	- -
Manual Search	Manually testing multiple configuration based on intuition	No technical overhead. Simple and Quick when intuition is right	Tedious. Doesn't gurantee optimal λ
Grid Search	Exhaustive search through a manually specified subset of the hyperparameter space.	finds the optimal space when the intuition is right. Grid search is reliable in low-dimensions [183]. Better than manual search (used in compute cluster). Simple and can leverage parallelization.	computationally expensive. Impratical for large search spaces.
Random Search	sampling hyperparameter settings and testing them	scalability - simplicity - distribution - fault tolerence - can perform indep. experiments.	Do not guarantee to find optimal hyperparameters
Bayesian Search	Finding a statistical model (or function) that maps parameters to the target	obtain better results in fewer experiments than grid search	complex - assumes the existence of a smooth function

TABLE 5.7 – Search methods summary

ANN layer size	probability p
2	0.16
3	0.16
4	0.16
7	0.11
11	0.11
17	0.08
27	0.06
41	0.04
64	0.04
100	0.03

TABLE 5.8 – The Categorical Bernoulli distribution used in our experiments. To the left, the ANN layer size in the log domain (log_{10}). the p values sum to one and define the probability of appearance of the layer size variable. Note that the size of the variable was arbitrary set to 10 and can be changed.

The Figure 5.25 depicts the worst case scenario for a grid search algorithm. The space in red represents the exhaustive search space of the grid algorithm, while the jointures of the gray lines represent the local random points of random search. If the machine learning operator fails to estimate the range of the hyperparameter subspace, grid search will perform a long and exhaustive search for finally coming up with a non-optimal set of hyperparameters. Random search on the other hand doesn't require this intuition and is computationally more efficient than grid search and has a high probability of approaching the optimal hyperparameters combination even in higher dimensions where grid search cannot finish in a reasonable time.

However, if properly configured, grid search can outperform random search. In our study, we have set budget constraints in terms of computation and search time. For this reason, we opt for random search as an efficient method that respects our constraints.

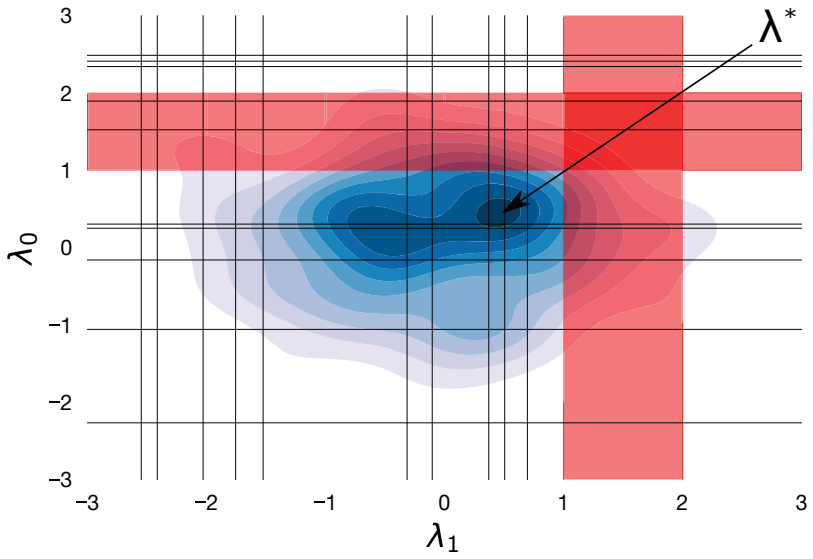


FIGURE 5.25 – Random search is computationally less expensive and scans a wider area in the configuration space shown as interconnection of gray lines. On the other hand, Random Search is very effective especially when the intuition of the operator fails to approximate the range of the search as shown in the intersection of red rectangles.

Bayesian search or Bayesian Optimization derives a statistical model M from the hyperparameters to their corresponding accuracy metric computed from the validation dataset. The assumption behind this technique is the existence of a smooth function between the hyperparameters and the objective. The idea is to try to guess a functional mapping between some few hyperparameters and the results. Ultimately, It tries to guess empirically the density function in equation 5.7 from multiple points. The Bayesian search shows relatively better results than Grid search and Random Search in fewer experiments. However Begstra et al. showed that Random search is sufficient for learning neural networks [184].

In Table 5.7, we summarized the four search method with their advantages and drawbacks.

C The hyperparameter Search Space

In this section we present a non-exhaustive list of the hyperparameters search space that we used in our empirical machine learning approach to the SLA management.

The hyperparameter space we describe in this section refers to two ANN architectures shown in Figure 5.27. In right hand part of the Figure 5.27, a FFNN that takes 156 features at a time and classifies them into 2^3 classesi is presented. Similarly, in left hand part of the Figure Figure 5.27 a the LSTM is presented which classifies the inputs while tracking the sequential pattern in the inputs. In the study presented in this paper, we will vary the hyperparameters of both ANN and observe how they behave in terms of accuracy and training time.

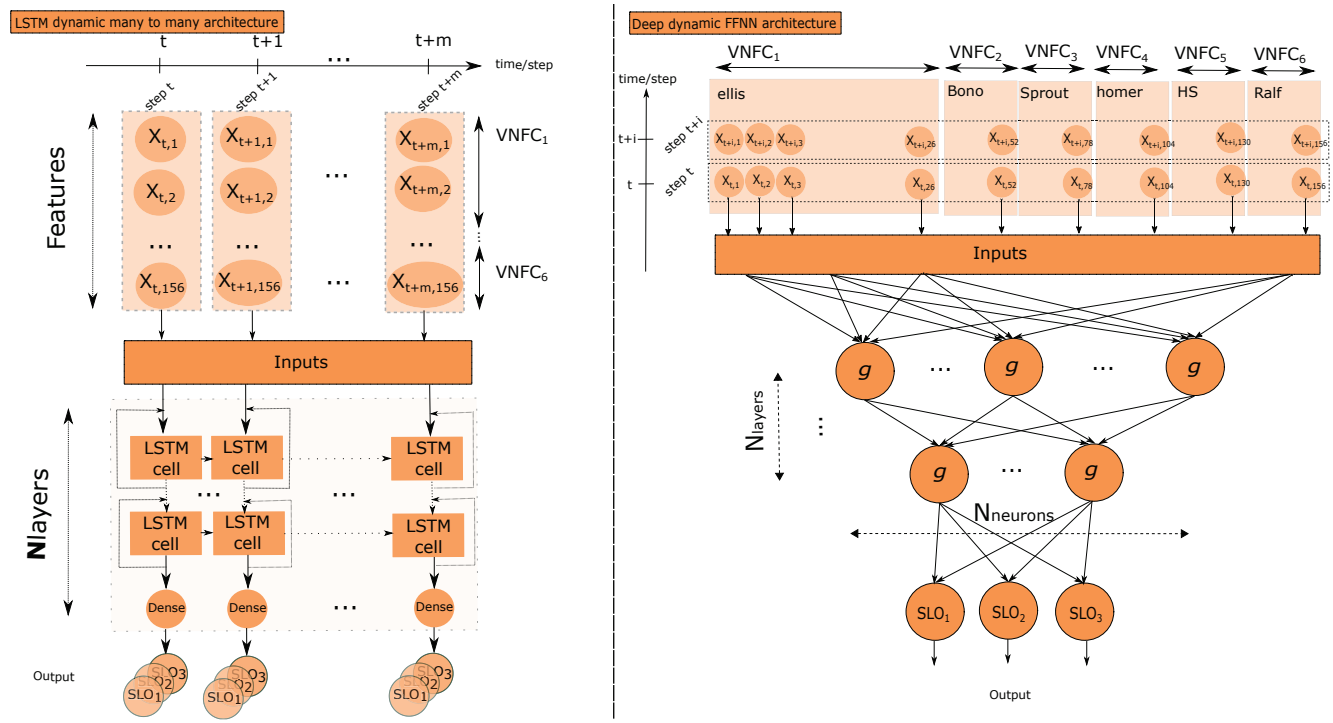


FIGURE 5.27 – Overview of the dynamic ANNs architectures. LSTM on the left and FFNN on the right.

Hyperparameters could be classified into two categories : the *structural* and *non-structural* parameters. The structural parameters deal with ANN architecture, such as the number of neurons per layer. The non-structural parameters deal with specific hyperparameter values related to the training algorithm such as the learning rate α, the momentum, etc.

The meaning of the hyperparameters used in this paper are presented below in the list 1. The Table 5.9 gives their distribution and their range of values.

1. Structural hyperparameters :

(a) *Number of hidden layers* : Selected following a Categorical Bernoulli Distribution [1]. For more details, see index in the annexe B.

(b) *Number of hidden nodes per layer* : Sampled randomly, log-uniformly [2] between 2 to 100.

(c) *Layer types* : For the LSTM we draw from LSTM layer and Dense Layer. For the FFNN, we draw uniformly from perceptron layer.

(d) *Activation function* :

The activation function or transfer function is a function that is applied to the output of each neuron as depicted in Figure 5.11 as $g(x)$. For example, the linear function is used when we need a real number as output whereas sigmoid squashes the output into a range of 0 to 1. Moreover, the activation functions bring the non-linear property to the ANN which can be used to break from the constraint of linearity and allow ANN to approximate nonlinear function. Intuitively, using the linear function in the output layer seems more adapted to regression problems and sigmoid to the classification ones. However, there is no theoretical basis for this claim that we are aware of, thus one aspect of the need for this empirical study.

ReLU (equation 5.9) : ReLU is consider as "new" non-saturating activation function, its gradient is either 0 for $inputs < 0$ or 1 for $inputs \geq 0$. This means that when using many layers it just multiplies the gradients by 1. This reduces the likelihood of vanishing or exploding gradient problems. For this reason, ReLU outperfoms the other activation functions is Deep NN. [31] when it is trained in a single global training, i. e. not using the "freezing training process".

$$RELU(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \tag{5.9}$$

1. Categorical Bernoulli is a discrete probability distribution over multiple classes (in this case number of hidden layers) with specified probability for each class. In our case, for implementaton purposes, the probability is biased towards ANNs with less than 20 layers.

2. log-uniformly between x and y means sampling from the log_{10} domain between $log_{10}(x)$ and $log_{10}(y)$.

SIGMOID : The sigmoid (equation 5.10) has an output that ranges between 0 and 1. In the literature, it is recommended to use for classification problems. The function is defined in equation 5.10 :

$$f(x) = \frac{1}{1 + e^{-x}} \tag{5.10}$$

Hard Sigmoid : The hard sigmoid function is similar to the sigmoid function to the exception that it approximate the value of e^{-x} for faster implementation speed. The resulting error can be then alleviated by adding more training steps.

SOFTMAX (equation 5.11) is similar to SIGMOID and TANH it can be used as the activation function of the last layer for classification, however its main advantage is that it sums to one.

$$SOFTMAX = \frac{e^{x_j}}{\sum_i e^{x_i}} \tag{5.11}$$

TANH : The tanh function (equation 5.12) is similar to the sigmoid function but it has an output that ranges between -1 and 1.

$$tanh(x) = \frac{2}{1 + e^{-x}} - 1 \tag{5.12}$$

LINEAR (equation 5.13) : The simplest activation function is the LINEAR where the output is a direct mapping to the inputs.

$$f(x) = x \tag{5.13}$$

We randomly assigned for each ANN layer an activation layer drawn uniformly from the list described above.

2. Non structural hyperparameters
   (a) *Dropout* :

   the Dropout is an ANN hyperparameter which is a recent regularization technique for deep architectures. Dropout aims at reducing the probability of overfitting [3] for ANNs with many hidden units. It was firstly proposed in [32]. It randomly removes

3. Overfitting is when the model learns too well the characteristics of the training data and fails to generalize to new data

few hidden neurons in the ANN during the training phase but keep them all in the testing phase. In our approach, the Dropout is selected randomly between 0 and 1 with a step of 0.1 which means that we select randomly from 0 to 100 % of the ANN units to 0 at each update while training to help prevent overfitting [33].

(b) *Weight Initialization* :

Weight initialization refers to the technique used to initialize the ANN weight values. They affect heavily the overall performance of the ANN. The Neural Network training is based on the optimization of a non-convex function. The gradient descent can converge to different local minimums depending on how the initial conditions (weight initialization among others) were set. The weight initialization of the ANN is as important as the ANN architecture. The basic pitfall in this section is to maintain a symmetrical initialization thus blocking the ANN from converging to the optimal weights. Many techniques can be used to break the weight symmetry using random processes. We explore these techniques in addition to some symmetric initialization. Below is the complete list of the explored weight initialization : Random Uniform, Random Normal, identity, zero, one, glorot normal, lecun uniform.

(c) *Optimizer type :*

add description from the Deep Leanring book

The optimization algorithms aim at minimizing the loss function of the ANN. The loss function can be expressed as an error function between the result Y and the inputs X as presented in equation 5.6. The optimizers that we experimented with are the following : Adaptive Moment Estimation (Adam), Stochastic Gradient Descent (SGD), Adamax, NAdam and adadelta. Their Learning rate α is chosen uniformly in a list starting from 0.001 to 0.1 with a step of 0.01. For the algorithms that support mini-batch, we used a random batch size varying from 10 to 1000.

(d) *Number of epochs :* After many offline trials based on conver-

hyperparameter	Min	Max	Distribution
classifier learn rate α	0.001	0.1	Log_2 Normal
N. layers	2	100	Generalized Bernoulli
N. hidden units	2	20	Generalized Bernoulli
Weight init.	9 choices	-	Uniform Random
Algorithm	6 choices	-	Uniform Random
Activ. function	6 choices	-	Distribution
Dropout	0%	90%	Uniform (step=0.1)
Layer Type	FFNN	LSTM	Uniform Random
Epochs	10	10,000	Log_{10} Normal
Batch size	20	1000	Log_{10} Normal
Sampling	10.000	177,000	Stepwise

TABLE 5.9 – Distribution over ANN hyperparameters and their range. Symbol Unif. means uniform, N means Gaussian-distributed, and log means uniformly distributed in the log-domain of base 10. Categorical Bernoulli For the choices see section C

gence time and performance, we came to bound the ANN training epochs to a range varying from 10 to 10,000.

D Research Methodology

We implemented the FFNN using Google's TensorFlow [185]. The neural network learned weights depends strictly on a specific SLO. The result is an FFNN architecture per SLO. The FFNN selected for an SLO is the best FFNN that converges during the training, that is the FFNN with the lowest cost function. For two FFNNs that converges on the same lowest cost, we select the one with the fastest training time. In order to do so, we introduce Algorithm 2 that randomly generates FFNNs within a predefined range of hyper-parameters (i.e., iterations, learning rate, activation functions) and a predefined range of global structure (i.e. number of hidden layers, number of neurons per layer). The initialization parameters are presented in Table 5.9.

In this paper, we limit our approach to resource-level metrics collected by Monasca. Certainly, the ANNs can be more effective using more high-level feature (i.e service-based metrics such as SIP connexions). However, our approach is easily generalizable to any new service. Table 5.10 exposes a subset of metrics used in this paper.

Algorithm 2: Random FFNN Generation

```
Input : Number of iterations and range of parameters;
Output : Selecting the FFNN with the lowest error;
// Initialization:
Initialize nbr; // number of neurons range
         nbl; // FFNN number of layers range
         epr; // epochs range [300-2000]
         alpha; // learning rate range [0.01-0.0001]
         actr; // activation functions range
for iteration in Iterations do
    Model ← generaterandomFFNNarch();
    Hyperparam ← generaterandomHyperp();
    Trainthemodel();
    Watcher[] ← trainingtime, minerrors;
Select min(Watcher);
Parse and Return the FFNN Model;
```

Algorithm 3: generaterandomFFNNarch

```
Input : Range of neurons per layer and range of hidden layers;
Output : FFNN architecture;
numlayers =
   random(1,max(rangeneuronsperlayer));
for layer in numlayers do
    layer[] = random(1, max(rangelayers));
return [layer, numlayers];
```

E Data

The Dataset used in our experiments were generated from the observations of our experimental environment. The monitoring data are collected by Monasca [4]. Table 5.10 shows a subset of the collected metrics. Monasca collects mainly low-level system metrics. The SIP load generator is SIPp [166]. We generate up to 30.000 bot SIP user to simulate service degradation and violate the SLA.

Results and Discussion

We run our experiments in local machines as well as in the Cloud with CPU and GPU configurations. The bulk of the tests were run at

4. MONitoring As a Service - http ://monasca.io/

Metric Name	Semantics
cpu.idle – perc	Percentage of time the CPU is idle when no I/O requests are in progress
cpu.wait – perc	Percentage of time the CPU is idle AND there is at least one I/O request in progress
cpu.stolen – perc	Percentage of stolen CPU time, i.e. the time spent in other OS contexts when running in a virtualized environment
disk.total – spacemb	The total amount of disk space aggregated across all the disks on a particular node.
io.read – kbytessec	Kbytes/sec read by an io device
io.read – req – sec	Number of read requests/sec to an io device
load.avg – 1 – min	The average system load over a 1 minute period
mem.swap – free – perc	Percentage of free swap memory that is free
net.inbytes – sec	Number of network bytes received per second
net.outbytes – sec	Number of network bytes sent per second
net.inpackets – sec	Number of network packets received per second
net.outpackets – sec	Number of network packets sent per second
net.in – errors – sec	Number of network errors on incoming network traffic per second

TABLE 5.10 – Subset of Monasca monitoring metrics.

night, we collected the results after many hours (average of 4 hours on GPU and 10 hours on CPU) as dataframes (i.e. similar to excel data sheet, see annexe **??**). Here also, a pre-processing phase was necessary before ploting and representing all the data.

We observe that ANN successfully managed to learn the underlying complex patterns of the dynamic environment expected in future networks. Through our methodology, to combat the obstacle of expert knowledge required for tuning such complex deep neural networks, we show that the best architecture was selected. We had some expectations in mind on what the results would look like. We expect that Deep layered ANNs will perform better than the classical ones. Eventhough this intuition was right, we get a broader and more complete perspective on what ANN can perform and how it is possible to utilize their hyperparameters for better results with reasonable resources and tight budget constrained.

Using the outputed dataframe, we notice that we can consider all the hyperparameters as features and accuracy over the validation score as

the target value. Reframing the problem in this way was helpful to determine the hyperparameters relevance for setting the most appropriate ANN. We have used entropy-based algorithm to determine the degree of purity of each hyperparameters. Finally, We found that Maybe for the Journal.

Since we run different random experiments on different platforms, we ended up with different dataframes. We regroup and represente all the intresting results as boxplots over all the 20 trials.

We gained much insight on the LSTM and FFNN learning dynamics. We find that random search coupled with Bernoulli distribution allowed us to explore entrenched area of the hyperparameter space that would take months to compute on standard grid search algorithm. We obtain insight into deep and wide architecture at a glance as we are constrained by the training time. We leverage Machine Learning again to extrapolate and estimate the *response function* [5] (as described in section A in equation 5.6)

Strong Hyperparameters

The collected results indicate that the most strongest hyperparameters are the width of the ANN as previously discussed (figure 5.36), the ANN initialization method (figure 5.33), and the optimization algorithm (figure 5.31).

In all these cases, the results show that the difference between a proper setting and an ill setting is up to 50% on the accuracy score. For example, in figure 5.31, the best optimizer, Adadelta, gives an accuracy of 94% for almost all the trials (except two considered as outliers), while the worst, RMSprop, yields an accuracy of 0 over all the trials. Other optimizers show unsettled results. we presume that for the other optimizers, their accuracy depend on other factors (i.e. hyperparameters).

Similarly, the results over the initialization method ((figure 5.33) show that *glorot uniform* is the best method. *orthogonal* and *unifrom* initialization give the highest probability of converging to an acceptable accuracy. While, *glorot normal*, *normal* and *zero* should be avoided.

We consider the activation function (in the output layer) as a strong hyperparameter. In our use case (figure 5.30) the best one is the linear

5. One caveat on the extrapolation is that the response function should be smooth and continuous

followed by the hard sigmoid and softmax.

Weak Hyperparameters

In the weak hyperparameters, we include the dropout (figure 5.28), the number of hidden layers (5.36), training epochs and the layer type (figure 5.32)

Although the layer type is technically a strong parameter, we classify it as a weak one due to its representation using the global metric as depicted in figure 5.32 left. The high LSTM accuracy averages out with its long training time [6]. Similarly, the number of epochs are strong indicators of the ANN accuracy (figure **??**) but the training time explode after only few hundreds iterations.

The ANNs with 50, 60 and 70 % dropout demonstrate in figure 5.28 slightly better results than ANN without dropout eventhough the training time of all the dropout are overall similar (figure 5.29)

Best ANNs

The figure 5.35 and 5.34 show the best ANNs accordying to the validation score and the $Global\underline{m}etric$. Intrestingly, the intersection between these two groups is null. This means that there is no easy compromise between accuracy and training time.

Another remark is that the best ANN "$\underline{7}1$" over validation score (figure 5.35) distributions are focused in one point, the 94% accuracy point. We suspect that given the initial settings, the best results was guranteed even after 20 different trials. Moreover, it seems that the ANNs couldn't go further that 94% due to the hyperparameters range restrictions (see Table 5.9). In other words, if we have allowed the ANNs to explore higher depth, width and epochs, we could have surpassed the barrier of 0.94. However, this claim should be backed with more evidence.

We summarized the structural hyperparameters of the 10 best ANNs in table B.3. The common thread in all these architecture is the need for if not Deep, Wide ANN structure.

LSTM Architecture seems to need fewer layers and training epochs. LSTM is much robust than FFNN but it outperforms LSTM in rapidity.

6. We found that the LSTM benefits less from the GPU configuration than the FFNN. The training

Model ID	Structure	type, epochs
ACC		
1	[15, 13, 15]	LSTM 57
42	[20, 3, 12, 7, 1, 15, 20]	FFNN 52
44	[18, 3, 1, 9, 17, 18, 17, 14, 11, 19]	FFNN 766
46	[16, 13, 17, 3, 4, 7]	FFNN 330
71	[8, 1, 17, 15, 15]	FFNN 64
GM		
66	[8, 6, 11, 9, 15, 20, 4]	FFNN, 674
84	[19, 7, 9, 17, 7, 7, 15, 17, 16, 19]	FFNN 335
49	[16, 20, 14, 20]	FFNN 763
16	[7, 12, 15, 8]	LSTM 930
15	[8, 20, 15, 19, 9, 14, 10, 16, 19, 10]	LSTM 334

TABLE 5.11 – Different FFNN and LSTM architectures and their name.

Poor-performance ANNs

We defined "the loosers" as ANN with accuracy less then 30% . In this category, we find mainly shallow ANNs, i.e. less than 3 hidden layers or less then 6 average neurons per layer. The worst ANNs are found also in the hyperparameters previously discussed in weak and strong hyperparameters. Surprisingly, in other experiments we found that a large proportion of the poor-performance ANNs are a combination of RELU and GLorot uniform or he uniform. This requires further investigations.

Overall, LSTM is more performant than FFNN but is more prone to overfitting when incrementing the number of epochs. One solution is to use early stopping. FFNN can yield comparable results but requires more hyperparameter tuning. The power of Random Search method lays in it simplicity. Although, it cannot guarantee the optimal performance, it scales very well and is a good alternative when human intuition fails.

In this section, we demonstrated that the ANNs constitute a great opportunity to manage dynamic and highly evolving networks. They are flexible and can adapt to any new contextual situation. We pointed out the pitfalls that could face future ANN-based systems and provided directives and a methodology to select the most suitable one in terms of

time decreases by about 2/3 for the LSTM while the FFNN's training time decreases up to 5 times more

performance and budget.

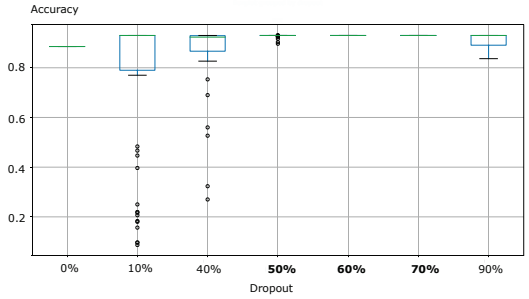


FIGURE 5.28 – Accuracy versus dropout. We can infer from this figure that dropout regularisation slightly improves the accuracy with no compromise in time (as represented by figure 5.29)

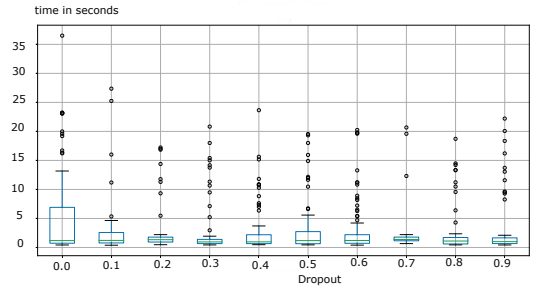


FIGURE 5.29 – Time versus dropout

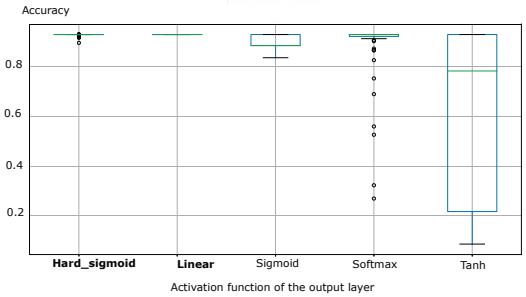


FIGURE 5.30 – Accuracy with respect to Activation functions in the output layer. This figure shows that the choice of the activation layer is amogst the most important ones.

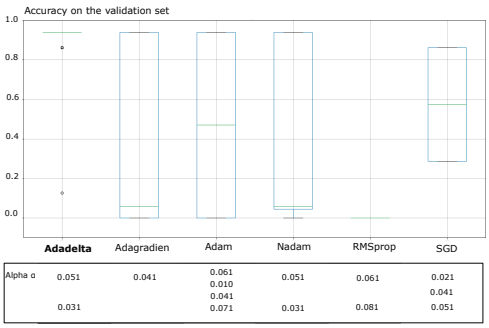


FIGURE 5.31 – Optimizer type with their respective learning rate α versus accuracy. The best optimizer is Adadelta, the worst is RMSprop.

F Meta-Learning

In networking context, many learning problems require an ability to learn rapidly from a limited size of data and to adjust to new information. These problems are particularly difficult using Artificial Neural Networks, because they rely on high volume of labeled data, and requires considerable time to train. In this section, we investigate the meta-learning approach to this problem.

Meta-learning is about the process of acquiring knwoledge about knowledge, i.e. meta-knowledge. Meta-knowledge can be defined as the type of knowledge that can be derived from using and observing a given lear-

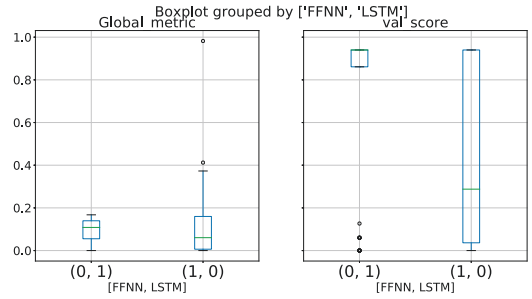


FIGURE 5.32 – comparison between LSTM and FFNN by accuracy over the validation set.

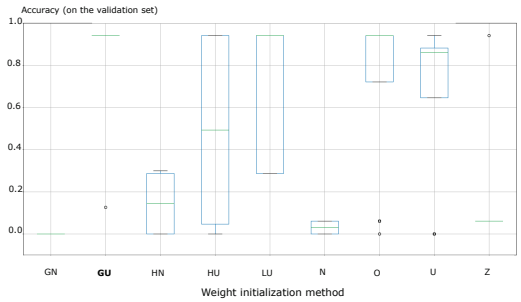


FIGURE 5.33 – ANN initialization method versus accuracy. GN : Glorot Normal, GU : Glorot Uniform, HN : He Normal, HU : He Uniform, LU : Lecun Uniform, N : Normal, O : Orthogonal, U : Uniform, Z : zero.

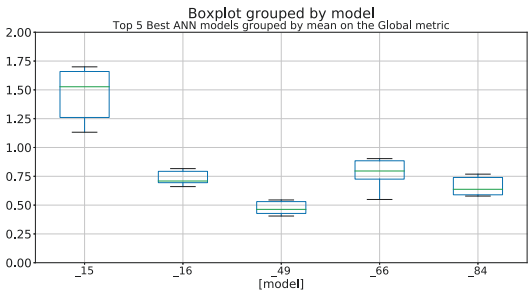


FIGURE 5.34 – Top 5 best ANNs based on the mean over Globalmetric

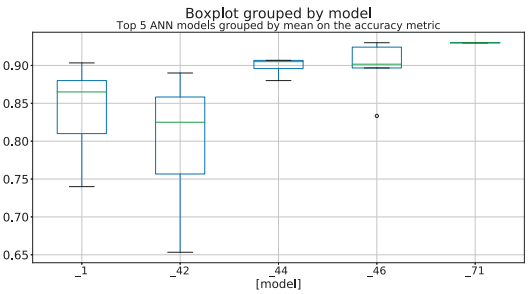


FIGURE 5.35 – Top 5 ANNs based on the mean over validation score accuracy. All the best ANNs appear to converge on 94% average accuracy.

ning algorithm [186]. In the field of meta-knowledge, the key question is how can we acquire and exploit knowledge about systems to understand them and improve their performance ? In this section, our aim is to examine how we can turn back the machine learning problems on itself.

The main idea of this section is to allow machine learning algorithm to profit from their repetitive use over similar dataset. The Meta-analysis approach is based on the observation of two parameters : the algorithm configurations (or hyperparameters) and its corresponding accuracy result. The Meta-analysis aim is to collect as many information as possible from the cimulative experience of training multiple and different Learning algorithms. In turn, meta-learning can be used to control the learning strategy and hyperparameters towards a more optimized algorithm.

As an example, the work of B. Zoph & Q. Le [187] exploits meta-learning by introducing two Recurrent Neural Networks, one that controls the other using a reinforcement process. Meta-learning can be used to perform a variety of sub tasks ... here give examples ...

Hence, learning to learn is as important as the learning itself. Expltoiting the meta-knowledge and extrapolating to new unseen configurations. In this section we will refer to these information as meta-features. A meta-feature can refer to the activation functions used, the learning rate, etc.

A second question that can be asked in the meta-learning analysis, is which algorithm is the most suited for a given task, or the ordering of a given algorithms ? In ou previous section, we showed that one way to answer these question is by gathering performance and multiple accuracy metrics of the learning algorithms and to compare them side by side.

Our contribution is to demonstrate how we can train few randomly selected ANNs to find the theoretical optimal ANN configurations. Secondly, we resused the information gain metric that is used in decision tree models to classify the importance and relevance of each hyperparameter.

We have noticed that after the experimentation phase, we can collect the outputed results as new inputs to a machine learning algorithm to guide the selection of the best possible combination of ANN configura-

tion.

The objective of this section is to lay the theoretical foundation of this approach, develop the methodology and interpret the early results.

Some consideration should be taken into account with respect to this approach

Overview

s

We run our experiments in local machines as well as in the Cloud with CPU and GPU configurations. The bulk of the tests were run at night, we collected the results after many hours (average of 4 hours on GPU and 10 hours on CPU) as dataframes (i.e. similar to excel data sheet, see annexe B). Here also, a pre-processing phase was necessary before ploting and representing all the data.

We observe that ANN successfully managed to learn the underlying complex patterns of the dynamic environment expected in future networks. Through our methodology, to combat the obstacle of expert knowledge required for tuning such complex deep neural networks, we show that the best architecture was selected. We had some expectations in mind on what the results would look like. We expect that Deep layered ANNs will perform better than the classical ones. Eventhough this intuition was right, we get a broader and more complete perspective on what ANN can perform and how it is possible to utilize their hyperparameters for better results with reasonable resources and tight budget constrained.

Using the outputed dataframe, we notice that we can consider all the hyperparameters as features and accuracy over the validation score as the target value. Reframing the problem in this way was helpful to determine the hyperparameters relevance for setting the most appropriate ANN. We have used entropy-based algorithm to determine the degree of purity of each hyperparameters. Finally, We found that Maybe for the Journal.

Since we run different random experiments on different platforms, we ended up with different dataframes. We regroup and represente all the intresting results as boxplots over all the 20 trials.

We gained much insight on the LSTM and FFNN learning dynamics.

We find that random search coupled with Bernoulli distribution allowed us to explore entrenched area of the hyperparameter space that would take months to compute on standard grid search algorithm. We obtain insight into deep and wide architecture at a glance as we are constrained by the training time. We leverage Machine Learning again to extrapolate and estimate the *response function*[7] (as described in section A in equation 5.6)

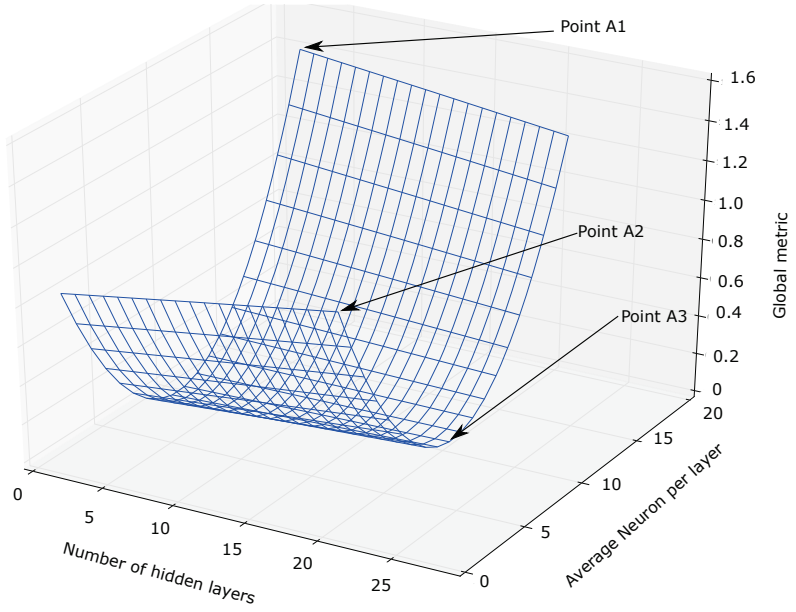


FIGURE 5.36 – Results for medium ANNs structure (under 30 hidden layers and 20 average neuron per layer). Point A1 is the global maximum. Point A2 is a local maximum. Point A3 is a local minimum. In the text below, we provide an interpretaion of these results.

The results obtained in Figure 5.36 were extrapolated from 200 experimentations as points scattered around the curve. The *Global_metric* is the z-axis. It is ploted with respect to the x-axis as the number of hidden layers and y-axis as the average neuron per layer (see annexe B). We will describe the figure in what we consider 3 critical points refered to as A1, A2, A3 respectively.

The point A1 : This point is the global maximum (in the observable interval of $(20, 20)$). The key observation is that the line crossing the global maximum is decreasing with the increase of the number of hidden layers. We explain this finding by the definition of *Global_metric* that

7. One caveat on the extrapolation is that the response function should be smooth and continuous

constrains ANNs with respect to the training time. In other words, the point A1 is the best compromise between accuracy and training time. Notice that in this setting, the average neuron per layer has more weight than the number of hidden layers. Through out our experiments, we notice that wide and shallow ANNs perform better that thin and deep. We suspect that this is due to the large number of features provided at the input layer(156).

The point A2 : This point is a local maximum. It shows that the depth of the ANN is important but not as much as the width of the ANN. we can observe this mismatch of importance by drawing a straing line from A2 parallel to the y-axis. the intersection with the curve gives a higher point, which means a higher accuracy.

The point A3 : This point represent a local minimum. It is in a blacklisted zone by the *Global*$\underline{m}$*etric*. Meaning that in this region of the hyperparameter space with many neurons, we obtain similar accuracy than regions with much lower neurons and shorter training time. The metric is telling us, do not waste your time here, if you have budget constraints use a smaller ANN, it will give you similar results.

Among all the hyperparameters presented in this paper, we found that only a few have strong correlation with the ANN overall performace. We can classify our findings into two classes, hyperparameters with strong influence on the result, we call them *strong hyperparameters* and hyperparameters with loose correlation to the performance, we call them *weak hyperparameters*. Note that the definition of these classes depends highly on the dataset and on the use case.

G feature relevance

The second question that we asked in the meta-learning study is how each meta-feature variabtions influence the accuracy output. In order to represent the problem in a simple way we present here a trivial example based on three features, sex, and years of study with a target value corresponding to the salary in table 5.12. Concretely we want to assess how the variation of a feature such as ”sex” can impact the salary for this given dataset.

TABLE 5.12 – Example of a data set

Bac+5	Sex	Salary
1	H	50
1	H	42
1	H	45
0	H	30
1	F	42
0	F	29
0	F	28
0	F	29

We used the entropy and the information gain to calculate how much information is gained from both features. The information gain of a variable X is obtained from an observation that a variable B=b is the Kullback-Leibler divergence $D_{KL}(p(x|b)||p(x|I))$ of the prior distribution $p(x|I)$ for x from the posterior distribution $p(x|b)$ for x given b.

In the context of decision tree, information gain is also refered to as mutual information and is equal to the total entropy for an attribute.

The information gain IG for an attribute b is defined using the entropy $H()$ as follows :

$$IG(T,b) = H(T) - \Sigma_{x \in vals(b)} \frac{|x \in T | x_b = v|}{|T|} . H(x \in T | x_b = v)$$

With $H(T) = -\log_2(X)$ and T is the set of the training instances $(x,y) = (x_1, x_2, ..., x$ *where* $x_b \in vals(b)$

Figure 5.37 show that the most relevant feature for the salary is the "Bac+5" with a value of 87.5%. The feature "Sex" is much less relevent with a value of 12.5%. Similarly, we used the same logic to rank the ANN hyperparameters impact on the accuracy. The results are shown in Figure 5.38. We notice that among the hyper-parameters the most impactful parameter is the Layer type, i.e. FFNN and LSTM. Empirically, we found that type of the ANN has the biggest importance up to 0.18 (Figure 5.38). Then the second most important parameter is the LEarning rate α up to 0.13. Then the third most relevant hyper-parameter are both the number of hidden layers and the width of the ANN i.e. average neurons per layer, with both 0.10. And among the least impactful hyperparameters we find the some optimization algorithms (RMSprop, Adadelta) and some activation functions (e.g. linear, softmax).

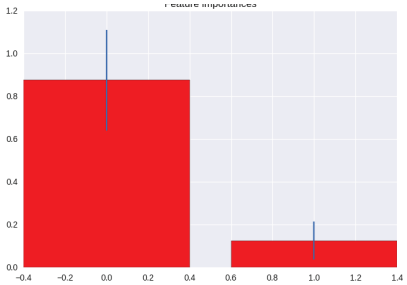


FIGURE 5.37 – Feature importance of the table 5.12.

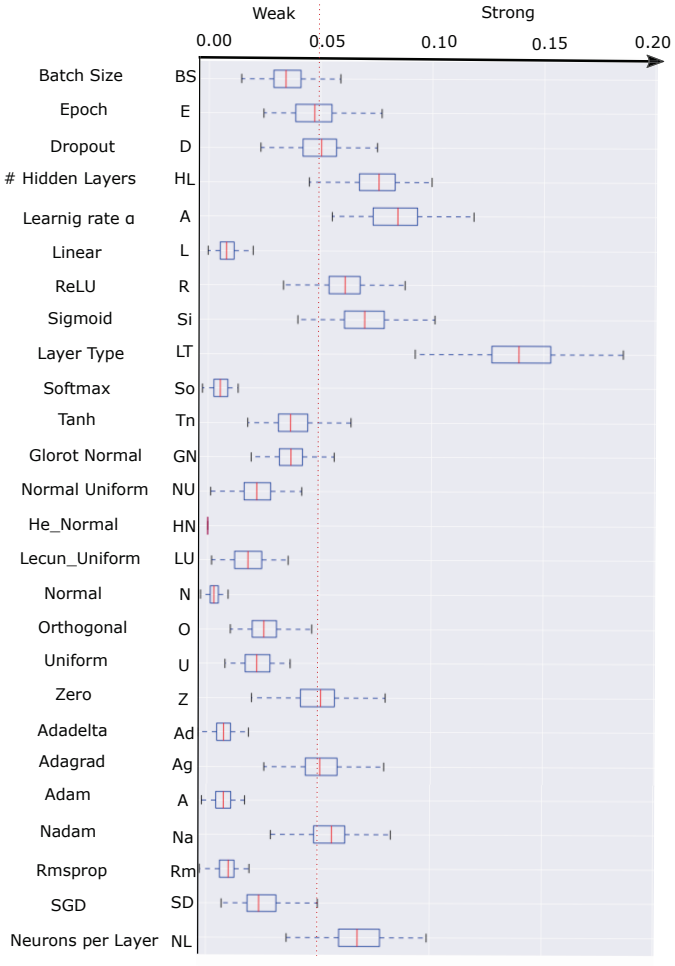


FIGURE 5.38 – Feature importance of the ANNs' hyperparameters

IV Conclusion

We have investigated an empirical approach to machine learning that consists of generating multiple random ANNs with different factor of

variations for the SLA use case in the context of Network Function virtualization. We provided an end-to-end data-driven methodology for network management. We used a real test case using a virtual IMS to instantiate the NFV framework. Generate data from experiment and stress the management opportunities and challenges brought by NFV and how a data-driven approach based on ANN can be leverage to tackle the management aspects. Our results support the following conclusions :

— FFNN and LSTM if properly configured can yield high accuracy (up to 94%) .

— ANNs are very sensitive to the hyperparameters. FFNN yields better results in Wide architectures, while LSTM requires Deep architecture.

— Overall, LSTM is more performant than FFNN but is more prone to overfitting when incrementing the number of epochs.

— The power of Random Search method lays in it simplicity. Although, it cannot guarantee the optimal performance, it scales very well and is a good alternative when human intuition fails.

To confirm our findings, we are currently running multiple experiments on the cloud with more resources and a wider range of hyperparameters. Moreover, we are considering adding additional ANN types from the ANN zoo to get a broader perspective on the power of ANNs for SLA manegement in a virtualized context.

We argue in this paper that the management of the programmable networks should necessarly incorporate intelligent and cognitive solutions. We demonstrate how Machine Leanring (ANN) can be leverage to this end.

Annexe A

Thesis Publications

I List of Contributions

A Accepted papers

— J. Bendriss, I. G. B. Yahia, and D. Zeglache, ”Sla enforcement in programmable networks,” in 9th International Conference on Autonomous Infrastructure, Management and Security (AIMS), 2015.

— J. Bendriss, I. G. B. Yahia, and D. Zeghlache, ”Forecasting and anticipating slo breaches in programmable networks,” in Innovations in Clouds, Internet and Networks (ICIN), 2017 20th Conference on, pp. 127-134, IEEE, 2017.

— J. Bendriss, I. G. B. Yahia, P. Chemouil, D. Zeghlache. AI for SLA Management in Programmable Networks. In Proc. of 13th International Conference on Design of Reliable Communication Networks (DRCN); 8-10 March 2017; Munich. Berlin : VDE; 2017. p. 1-8 - DRCN 2017

— B. Caglayan, I. G. B. Yahia, J. Bendriss, Cognitive Services Portfolio for 5G Network Management - 2nd Workshop on Network Management, Quality of Service and Security for 5G Networks @EuCNC17, June 12th 2017

— J. Bendriss, I. G. B. Yahia, and D. Zeghlache, ”A Deep learning based SLA management for NFV-based services,” in Innovations in Clouds, Internet and Networks (ICIN), 2018 21th Conference, IEEE, 2018.

— J.Bendriss et al. a journal, titled ”Cognitive SLA Management Using

Random ANNs", submitted to IEEE Transactions on Network and Service Management IEEE Transactions on Network and Service Management (IEEE TNSM) 2018.

— Imen Grida Ben Yahia, Jaafar Bendriss, Alassane Samba, Philippe Dooze : CogNitive 5G networks : Comprehensive operator use cases with machine learning for management operations. ICIN 2017 : 252-259

— J. Bendriss et al., Chapter Book accepted at EAI : Emerging Automation Techniques for Future Internet, book : Cognitive management for 5G : Going beyond Automation, Mars 2018

— Ongoing contribution on Machine Learning for Fault Management - IEEE JSAC Special Issue on Artificial Intelligence and Machine Learning for Networking and Communications to be submitted, July 2018

B Public Cognet Deliverables

Contributions to many CogNet deliverables to integrate my work in the context of 5G :

— D2.1 - Initial use cases, scenarios and requirements

— D2.2 - CogNet final requirements, scenarios and architecture

— D3.2 - Feature and Structure modeling, Structured Input/Output, Unsupervised Learning and Domain Adaptation

— D3.4 - Training, Validation and Deployment

— D4.1 - Initial Design - Raw Data Preprocessing, Prediction in NFV, Self-Managed NFV Ecosystem, Network Traffic Classification and Prediction

— D4.2 - Raw Data Preprocessing, Prediction in NFV, Self-Managed NFV Ecosystem, Network Traffic Classification and Prediction.

— D4.3 - Raw Data Preprocessing, Prediction in NFV, Self-Managed NFV Ecosystem, Network Traffic Classification and Prediction

— D4.4 - Standardized Analytics Module for NFV-MANO : A-MANO.

— D6.1 - Initial integration and validation plan.

— D6.2 - First release of the integrated platform and performance reports.
— D6.3 - Final release of the integrated platform and performance reports.

C Exhibition

— The Orange exhibition days in Orange Gardens in 2016
— 5G Global Event in Rome in November 2016
— Mobile World Congress in Mars 2018

Annexe B

Installation setup

Architecture name	Structure and hyper-Parameters
A1	Learning rate : 0.00261, Activation function : logistic Training iterations : 1760, Architecture : [156, 6, 1]
A2	Learning rate : 0.00681, Activation function : relu Training iterations : 660, Architecture : [156, 6, 9, 7, 8, 1]
A3	Learning rate : 0.00981, Activation function : tanh Training iterations : 820, Architecture : [156, 6, 4, 4, 1]
A4	Learning rate : 0.00201, Activation function : linear Training iterations : 640, Architecture : [156, 4, 1, 3, 5, 7, 2, 1, 8, 8, 1]
A5	Learning rate : 0.00301, Activation function : logistic Training iterations : 1800, Architecture : [156, 2, 2, 1, 1, 1, 3, 2, 1, 1]
A6	Learning rate : 0.00861, Activation function : relu Training iterations : 1320, Architecture : [156, 11, 1, 10, 9, 1, 8, 1, 8, 3, 4, 1]
A7	Learning rate : 0.00201, Activation function : tanh Training iterations : 640, Architecture : [156, 4, 2, 2, 1, 7, 7, 7, 6, 6, 1]
A8	Learning rate : 0.00981, Activation function : linear Training iterations : 820, Architecture : [156, 1, 3, 2, 1]
A9	Learning rate : 0.00681, Activation function : logistic Training iterations : 660, Architecture : [156, 1, 4, 9, 8, 1]
A10	Learning rate : 0.00461, Activation function : relu Training iterations : 400, Architecture : [156, 2, 1, 2, 2, 1, 3, 2, 3, 1, 2, 1, 1]
A11	Learning rate : 0.00301, Activation function : tanh Training iterations : 180, Architecture : [156, 1, 1, 3, 3, 2, 3, 2, 2, 1]
A12	Learning rate : 0.00081, Activation function : linear Training iterations : 780, Architecture : [156, 9, 8, 1, 3, 5, 2, 1]
A13	Learning rate : 0.00481, Activation function : logistic Training iterations : 1000, Architecture : [156, 1, 1, 1]
A14	Learning rate : 0.00881, Activation function : relu Training iterations : 1240, Architecture : [156, 2, 1, 2, 2, 1, 1]
A15	Learning rate : 0.00281, Activation function : tanh Training iterations : 920, Architecture : [156, 4, 1, 4, 2, 4, 5, 1, 1]
A16	Learning rate : 0.00981, Activation function : linear Training iterations : 820, Architecture : [156, 3, 6, 2, 1]
A17	Learning rate : 0.00861, Activation function : logistic Training iterations : 1320,

SLO_1	SLO_2	SLO_3	SLA
0	0	0	Compliant
0	0	1	Violated
0	1	0	Violated
1	0	0	Violated
0	1	1	Violated
1	1	0	Violated
1	0	1	Violated
1	1	1	Violated

Table B.1 – The SLA state is violated if at least one SLO is breached.

The ANN structures reports : for ANN with the following structure : [156(input) - 20 - 12 - 5 - 22 -60 - 6 - 3(output)], we report : (Number of hidden layers, Mean Neurons per layer rounded to the least integer) = (6, 20)

Feature importance : The Feature importance is defined as XXX

Maybe a Figure ?

We computed feature importance based on node purity and entropy. In the literature, there are many methods for implementing the algorithm, we used Decision Tree Model to compute the features importance.

Bernoulli Categorical Distribution.

The Bernoulli categorical distributionis a discrete probability distribution over multiple classes (in this case number of hidden layers) with specified probability for each class. In our case, the probability is biased towards ANNs with less than 10 layers.

The categorical Bernoulli distribution allows us to bias the selection towards smaller ANNs. The idea is to generate diverse ANN structure but not too many Deep ANNs due to the high computing, memory and time cost.

ANN layer size	probability p
2	0.16
3	0.16
4	0.16
7	0.11
11	0.11
17	0.08
27	0.06
41	0.04
64	0.04
100	0.03

TABLE B.2 – The Categorical Bernoulli distribution used in our experiments. To the left, the ANN layer size in the log domain (log_{10}). the p values sum to one and define the probability of appearance of the layer size variable. Note that the size of the variable was arbitrary set to 10 and can be changed.

Architecture name	Structure and hyperparameters
4	2 hidden layers, epoch=10
5	5 hidden layers, epoch = 10
6	10 hidden layers, epoch=10

TABLE B.3 – Different FFNN and LSTM architectures and their name.

Table B.4 – a Sample of Results generated for benchmarking different hyperparameters configurations. The best performance is marked in bold. The confidence interval is set at 95 % and epochs = 10

Model	Type	Architecture	Drop	Loss cal	Weight init	Optimization	activ last layer
m1	LSTM	8 - 6.25	1	mse	zero	adadelta	relu
m2	LSTM	7 - 4.25	0	mse	normal	sgd	sigmoid
m3	LSTM	6 - 4.5	0	mse	zero	nadam	relu
m4	LSTM	7 - 7.28	0	mse	lecununiform	nadam	relu
m5	LSTM	9 - 7.2	0	binary	heuniform	adamax	softmax
m6	**LSTM**	**10 - 6.7**	**0**	**mse**	**uniform**	**adamax**	**linear**
m7	LSTM	7 - 3.14	0	mse	lecununiform	sgd	hardsigmoid
m8	LSTM	2 - 1.5	0	binary	glorotnormal	adamax	linear
m9	LSTM	3 - 2.0	0	mse	orthogonal	adam	tanh
m10	FFNN	3 - 1.00	1	mse	henormal	adamax	relu
m11	FFNN	19 - 12.31	1	binary	lecununiform	sgd	softmax
m12	FFNN	20 - 7.04	0	mse	uniform	adam	hardsigmoid
m13	FFNN	13 - 7.61	0	mse	one	adadelta	softmax
m14	**FFNN**	**2- 12.5**	**1**	**mse**	**henormal**	**nadam**	**linear**
m15	FFNN	4 - 10.0	1	binary	orthogonal	sgd	linear
m16	FFNN	6 - 12.16	1	mse	zero	adadelta	sigmoid
m17	FFNN	20 - 10.1	0	binary	glorotuniform	adam	linear
m18	FFNN	15 - 12.93	1	mse	one	adadelta	hardsigmoid

Bibliography

[1] N. Agoulmine, *Autonomic network management principles : from concepts to applications*. Academic Press, 2010.

[2] Wikipedia, the free encyclopedia, Bruce Blaus, "Biological neuron," 2013. [Online ; accessed April 27, 2018].

[3] P. D. Aimilia Bantouna, Dimos Kyriazis, "Initial position paper d3.1," 2015.

[4] H. Freeman and A. Gelman, "Ieee technology initiatives and related comsoc standards activities [the president's page]," *IEEE Communications Magazine*, vol. 54, no. 7, pp. 4–6, 2016.

[5] "Gs nfv 003 - v1.2.1 - network functions virtualisation (nfv) ; terminology for main concepts in nfv," 2014.

[6] A. Géron, "Hands-on machine learning with scikit-learn and tensorflow : concepts, tools, and techniques to build intelligent systems," 2017.

[7] J. McCarthy and E. A. Feigenbaum, "In memoriam : Arthur samuel : Pioneer in machine learning," *AI Magazine*, vol. 11, no. 3, p. 10, 1990.

[8] T. M. Mitchell *et al.*, "Machine learning. wcb," 1997.

[9] F. Cady, *The Data Science Handbook*. John Wiley & Sons, 2017.

[10] A. M. Turing, "Computing machinery and intelligence," *Mind*, vol. 59, no. 236, pp. 433–460, 1950.

[11] D. Hillis, J. McCarthy, T. M. Mitchell, E. T. Mueller, D. Riecken, A. Sloman, and P. H. Winston, "In honor of marvin minsky's contributions on his 80th birthday," *AI Magazine*, vol. 28, no. 4, p. 103, 2007.

[12] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 44, pp. 206–226, Jan 2000.

[13] R. J. Solomonoff, "An inductive inference machine," in *IRE Convention Record, Section on Information Theory*, vol. 2, pp. 56–62, 1957.

[14] P. Domingos, "A few useful things to know about machine learning," *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.

[15] Y. Bengio, A. Courville, and P. Vincent, "Representation learning : A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[16] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

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

[18] T. Mikolov, A. Deoras, D. Povey, L. Burget, and J. Černockỳ, "Strategies for training large scale neural network language models," in *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*, pp. 196–201, IEEE, 2011.

[19] F. Rosenblatt, "Principles of neurodynamics. perceptrons and the theory of brain mechanisms," tech. rep., CORNELL AERONAUTICAL LAB INC BUFFALO NY, 1961.

[20] M. Minsky and S. Papert, "Perceptrons.," 1969.

[21] D. O. Hebb, *The organization of behavior : A neuropsychological approach*. John Wiley & Sons, 1949.

[22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[23] M. Sokolova, N. Japkowicz, and S. Szpakowicz, "Beyond accuracy, f-score and roc : a family of discriminant measures for performance evaluation," in *Australian conference on artificial intelligence*, vol. 4304, pp. 1015–1021, 2006.

[24] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[25] P. Pudil and J. Hovovicova, "Novel methods for subset selection with respect to problem knowledge," *IEEE Intelligent Systems and their Applications*, vol. 13, no. 2, pp. 66–74, 1998.

[26] S. M. Holland, "Principal components analysis (pca)," *Department of Geology, University of Georgia, Athens, GA*, pp. 30602–2501, 2008.

[27] Y. LeCun, Y. Bengio, *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[28] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010.

[29] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[30] J. Nickolls and W. J. Dally, "The gpu computing era," *IEEE micro*, vol. 30, no. 2, 2010.

[31] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, "Understanding deep neural networks with rectified linear units, 2016," *U RL : https ://arxiv. org/abs/1611.01491*.

[32] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv :1207.0580*, 2012.

[33] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout : A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[34] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[35] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell : Neural image caption generation with visual attention," in *International Conference on Machine Learning*, pp. 2048–2057, 2015.

[36] J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio, "A recurrent latent variable model for sequential data," in *Advances in neural information processing systems*, pp. 2980–2988, 2015.

[37] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra, "Draw : A recurrent neural network for image generation," *arXiv preprint arXiv :1502.04623*, 2015.

[38] J. Bayer and C. Osendorfer, "Learning stochastic recurrent networks," *arXiv preprint arXiv :1411.7610*, 2014.

[39] N. Bostrom, *Superintelligence : Paths, dangers, strategies*. OUP Oxford, 2014.

[40] G. E. Hinton, "Deep belief networks," *Scholarpedia*, vol. 4, no. 5, p. 5947, 2009.

[41] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[42] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pp. 1717–1724, IEEE, 2014.

[43] R. Collobert and J. Weston, "A unified architecture for natural language processing : Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, pp. 160–167, ACM, 2008.

[44] J. Ba and R. Caruana, "Do deep nets really need to be deep ?," in *Advances in neural information processing systems*, pp. 2654–2662, 2014.

[45] R. Bellman, "Curse of dimensionality," *Adaptive control processes : a guided tour. Princeton, NJ*, 1961.

[46] K. Burn-Thornton, J. Garibaldi, and A. Mahdi, "Pro-active network management using data mining," in *Global Telecommunications Conference, 1998. GLOBECOM 1998. The Bridge to Global Integration. IEEE*, vol. 2, pp. 1208–1211, IEEE, 1998.

[47] R. Beverly, K. Sollins, and A. Berger, "Svm learning of ip address structure for latency prediction," in *Proceedings of the 2006 SIGCOMM workshop on Mining network data*, pp. 299–304, ACM, 2006.

[48] T. W. Rondeau, B. Le, C. J. Rieser, and C. W. Bostian, "Cognitive radios with genetic algorithms : Intelligent control of software defined radios," in *Software defined radio forum technical conference*, pp. C3–C8, Citeseer, 2004.

[49] M. Mirza, J. Sommers, P. Barford, and X. Zhu, "A machine learning approach to tcp throughput prediction," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 35, pp. 97–108, ACM, 2007.

[50] M. Lee, D. Marconett, X. Ye, and S. B. Yoo, "Cognitive network management with reinforcement learning for wireless mesh networks," in *International Workshop on IP Operations and Management*, pp. 168–179, Springer, 2007.

[51] E. Barrett, E. Howley, and J. Duggan, "Applying reinforcement learning towards automating resource allocation and application scalability in the cloud," *Concurrency and Computation : Practice and Experience*, vol. 25, no. 12, pp. 1656–1674, 2013.

[52] S.-C. Lin, I. F. Akyildiz, P. Wang, and M. Luo, "Qos-aware adaptive routing in multi-layer hierarchical software defined networks : a reinforcement learning approach," in *Services Computing (SCC), 2016 IEEE International Conference on*, pp. 25–33, IEEE, 2016.

[53] X. Dutreilh, S. Kirgizov, O. Melekhova, J. Malenfant, N. Rivierre, and I. Truck, "Using reinforcement learning for autonomic resource allocation in clouds : towards a fully automated workflow," in *ICAS 2011, The Seventh International Conference on Autonomic and Autonomous Systems*, pp. 67–74, 2011.

[54] C.-Z. Xu, J. Rao, and X. Bu, "Url : A unified reinforcement learning approach for autonomic cloud management," *Journal of Parallel and Distributed Computing*, vol. 72, no. 2, pp. 95–105, 2012.

[55] N. Williams, S. Zander, and G. Armitage, "A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 5, pp. 5–16, 2006.

[56] S. Zander, T. Nguyen, and G. Armitage, "Automated traffic classification and application identification using machine learning," in *Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on*, pp. 250–257, IEEE, 2005.

[57] S. Zander, T. Nguyen, and G. Armitage, "Self-learning ip traffic classification based on statistical flow characteristics," in *International Workshop on Passive and Active Network Measurement*, pp. 325–328, Springer, 2005.

[58] "Designing self-driving networks workshop." `https://datatracker.ietf.org/meeting/98/materials/slides-98-nmrg-self-driving-networks`. Accessed : 2018-03-30.

[59] N. Feamster and J. Rexford, "Why (and how) networks should run themselves," *arXiv preprint arXiv :1710.11583*, 2017.

[60] M. Zorzi, A. Zanella, A. Testolin, M. D. F. De Grazia, and M. Zorzi, "Cobanets : A new paradigm for cognitive communications systems," in *Computing, Networking and Communications (ICNC), 2016 International Conference on*, pp. 1–7, IEEE, 2016.

[61] A. Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, E. Alarcón, M. Solé, V. Muntés-Mulero, D. Meyer, S. Barkai, M. J. Hibbett, *et al.*, "Knowledge-defined networking," *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 3, pp. 2–10, 2017.

[62] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, "A knowledge plane for the internet," in *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 3–10, ACM, 2003.

[63] A. Clemm, *Network management fundamentals*. Cisco Press, 2006.

[64] Z. Li, M. Zhang, Z. Zhu, Y. Chen, A. G. Greenberg, and Y.-M. Wang, "Webprophet : Automating performance prediction for web services.," in *NSDI*, vol. 10, pp. 143–158, 2010.

[65] M. Tariq, A. Zeitoun, V. Valancius, N. Feamster, and M. Ammar, "Answering what-if deployment and configuration questions with wise," in *ACM SIGCOMM Computer Communication Review*, vol. 38, pp. 99–110, ACM, 2008.

[66] C. Sinclair, L. Pierce, and S. Matzner, "An application of machine learning to network intrusion detection," in *Computer Security Applications Conference, 1999.(ACSAC'99) Proceedings. 15th Annual*, pp. 371–377, IEEE, 1999.

[67] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, pp. 21–26, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016.

[68] M. Ross, A. Covo, and C. Hart, "An ai-based network management system," in *Computers and Communications, 1988. Conference Proceedings., Seventh Annual International Phoenix Conference on*, pp. 458–461, IEEE, 1988.

[69] R. Meike, "Intelligent resource management for local area networks : Approach and evolution," 1988.

[70] L. Bernstein and C. M. Yuhas, "Expert systems in network management-the second revolution," *IEEE Journal on Selected Areas in Communications*, vol. 6, no. 5, pp. 784–787, 1988.

[71] T. Liao and D. Seret, "Toward the intelligent integrated network management," in *Global Telecommunications Conference, 1990, and Exhibition.'Communications : Connecting the Future', GLOBECOM'90., IEEE*, pp. 1498–1502, IEEE, 1990.

[72] D. L. Cohrs and B. P. Miller, *Specification and verification of network managers for large internets*, vol. 19. ACM, 1989.

[73] K. D. Cebulka, M. J. Muller, and C. A. Riley, "Applications of artificial intelligence for meeting network management challenges in the 1990s," in *Global Telecommunications Conference and Exhibition'Communications Technology for the 1990s and Beyond'(GLOBECOM), 1989. IEEE*, pp. 501–506, IEEE, 1989.

[74] G. Kousiouris, D. Kyriazis, S. Gogouvitis, A. Menychtas, K. Konstanteli, and T. Varvarigou, "Translation of application-level terms to resource-level attributes across the cloud stack layers," in *IEEE Symposium on Computers and Communications (ISCC)*, pp. 153–160, IEEE, 2011.

[75] N. Bui, M. Cesana, S. A. Hosseini, Q. Liao, I. Malanchini, and J. Widmer, "A survey of anticipatory mobile networking : Context-based classification, prediction methodologies, and optimization techniques," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 3, pp. 1790–1821, 2017.

[76] Y. Liu and J. Y. Lee, "An empirical study of throughput prediction in mobile data networks," in *Global Communications Conference (GLOBECOM), 2015 IEEE*, pp. 1–6, IEEE, 2015.

[77] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[78] S. Baraković and L. Skorin-Kapov, "Survey and challenges of qoe management issues in wireless networks," *Journal of Computer Networks and Communications*, vol. 2013, 2013.

[79] S. Zhao, *Application-Aware Network Design Using Software-Defined Networking for Application Performance Optimization for Big Data and Video Streaming*. PhD thesis, University of Missouri-Kansas City, 2017.

[80] C. Sun, J. Bi, Z. Zheng, and H. Hu, "Sla-nfv : an sla-aware high performance framework for network function virtualization," in *Proceedings of the 2016 ACM SIGCOMM Conference*, pp. 581–582, ACM, 2016.

[81] H. Singh, "Performance analysis of unsupervised machine learning techniques for network traffic classification," in *Advanced Computing & Communication Technologies (ACCT), 2015 Fifth International Conference on*, pp. 401–404, IEEE, 2015.

[82] A. McGregor, M. Hall, P. Lorier, and J. Brunskill, "Flow clustering using machine learning techniques," *Passive and Active Network Measurement*, pp. 205–214, 2004.

[83] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, pp. 50–60, ACM, 2005.

[84] J. Hochst, L. Baumgartner, M. Hollick, and B. Freisleben, "Unsupervised traffic flow classification using a neural autoencoder," in *2017 IEEE 42nd Conference on Local Computer Networks (LCN)*, pp. 523–526, IEEE, 2017.

[85] L. He, C. Xu, and Y. Luo, "vtc : Machine learning based traffic classification as a virtual network function," in *Proceedings of the 2016 ACM International Workshop on Security in Software Defined Networks & Network Function Virtualization*, pp. 53–56, ACM, 2016.

[86] J. A. Boyan and M. L. Littman, "Packet routing in dynamically changing networks : A reinforcement learning approach," in *Advances in neural information processing systems*, pp. 671–678, 1994.

[87] L. Peshkin and V. Savova, "Reinforcement learning for adaptive routing," in *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, vol. 2, pp. 1825–1830, IEEE, 2002.

[88] D. W. Gurer, I. Khan, R. Ogier, and R. Keffer, "An artificial intelligence approach to network fault management," *Sri international*, vol. 86, 1996.

[89] R. Patton, "Fault detection and diagnosis in aerospace systems using analytical redundancy," *Computing & Control Engineering Journal*, vol. 2, no. 3, pp. 127–136, 1991.

[90] R. Patton, J. Chen, and T. Siew, "Fault diagnosis in nonlinear dynamic systems via neural networks," in *IEE Conference Publication*, pp. 1346–1346, IET, 1994.

[91] D. Himmelblau, R. Barker, and W. Suewatanakul, "Fault classification with the aid of artificial neural networks," *IFAC Proceedings Volumes*, vol. 24, no. 6, pp. 541–545, 1991.

[92] T. Sorsa, H. N. Koivo, and H. Koivisto, "Neural networks in process fault diagnosis," *IEEE Transactions on systems, man, and cybernetics*, vol. 21, no. 4, pp. 815–825, 1991.

[93] R. H. Deng, A. A. Lazar, and W. Wang, "A probabilistic approach to fault diagnosis in linear lightwave networks," *IEEE Journal on selected areas in communications*, vol. 11, no. 9, pp. 1438–1448, 1993.

[94] J. Wu, J.-G. Zhou, P.-L. Yan, and M. Wu, "A study on network fault knowledge acquisition based on support vector machine," in *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*, vol. 6, pp. 3893–3898, IEEE, 2005.

[95] J. M. Sánchez, I. G. B. Yahia, and N. Crespi, "Self-modeling based diagnosis of software-defined networks," in *Network Softwarization (NetSoft), 2015 1st IEEE Conference on*, pp. 1–6, IEEE, 2015.

[96] P. Sangkatsanee, N. Wattanapongsakorn, and C. Charnsripinyo, "Practical real-time intrusion detection using machine learning approaches," *Computer Communications*, vol. 34, no. 18, pp. 2227–2235, 2011.

[97] Q. Mahmoud, *Cognitive networks : towards self-aware networks*. John Wiley & Sons, 2007.

[98] L. Xu, H. Assem, I. G. B. Yahia, T. S. Buda, A. Martin, D. Gallico, M. Biancani, A. Pastor, P. A. Aranda, M. Smirnov, *et al.*, "Cognet : A network management architecture featuring cognitive capabilities," in *Networks and Communications (EuCNC), 2016 European Conference on*, pp. 325–329, IEEE, 2016.

[99] A. Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, E. Alarcón, M. Solé, V. Muntés-Mulero, D. Meyer, S. Barkai, M. J. Hibbett, G. Estrada, K. Ma'ruf, F. Coras, V. Ermagan, H. Latapie, C. Cassar, J. Evans, F. Maino, J. Walrand, and A. Cabellos, "Knowledge-defined networking," *SIGCOMM Comput. Commun. Rev.*, vol. 47, pp. 2–10, Sept. 2017.

[100] S. R. White, J. E. Hanson, I. Whalley, D. M. Chess, and J. O. Kephart, "An architectural approach to autonomic computing," in *Autonomic Computing, 2004. Proceedings. International Conference on*, pp. 2–9, IEEE, 2004.

[101] M. Wooldridge and N. R. Jennings, "Intelligent agents : Theory and practice," *The knowledge engineering review*, vol. 10, no. 2, pp. 115–152, 1995.

[102] M. C. Huebscher and J. A. McCann, "A survey of autonomic computing - degrees, models, and applications," *ACM Computing Surveys (CSUR)*, vol. 40, no. 3, p. 7, 2008.

[103] "Juniper - self-driving networks," 2018.

[104] "Acm sigcomm 2018 afternoon workshop on self-driving networks (selfdn 2018) - acm sigcomm 2018," 2018.

[105] M. Zorzi, A. Zanella, A. Testolin, M. D. F. De Grazia, and M. Zorzi, "Cognition-based networks : A new perspective on network optimization using learning and distributed intelligence," *IEEE Access*, vol. 3, pp. 1512–1530, 2015.

[106] "Amazon mechanical turk," 2018.

[107] "E.860 : Framework of a service level agreement.," 2002.

[108] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible markup language (xml).," *World Wide Web Journal*, vol. 2, no. 4, pp. 27–66, 1997.

[109] S. Bechhofer, "Owl : Web ontology language," in *Encyclopedia of database systems*, pp. 2008–2009, Springer, 2009.

[110] D. Crockford, "The application/json media type for javascript object notation (json)," 2006.

[111] O. Ben-Kiki, C. Evans, and B. Ingerson, "Yaml ain't markup language (yaml) version 1.1," *yaml. org, Tech. Rep*, p. 23, 2005.

[112] C. Cui, H. Deng, D. Telekom, U. Michel, and H. Damker, "Network functions virtualisation,"

[113] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization : State-of-the-art and research challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, 2016.

[114] C. Price, S. Rivera, *et al.*, "Opnfv : An open platform to accelerate nfv," *White Paper. A Linux Foundation Collaborative Project*, 2012.

[115] "Opnfv - ovp," 2018.

[116] "Pdna - devnet." `https://developer.cisco.com/site/pnda/`. Accessed : 2018-03-30.

[117] "The data plane development kit." https://dpdk.org/. Accessed : 2018-03-30.

[118] O. N. Fundation, "Software-defined networking : The new norm for networks," *ONF White Paper*, vol. 2, pp. 2–6, 2012.

[119] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian, "Fabric : a retrospective on evolving sdn," in *Proceedings of the first workshop on Hot topics in software defined networks*, pp. 85–90, ACM, 2012.

[120] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for sdn ? implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.

[121] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, *et al.*, "B4 : Experience with a globally-deployed software defined wan," in *ACM SIGCOMM Computer Communication Review*, vol. 43, pp. 3–14, ACM, 2013.

[122] E. Escalona, J. I. A. Baranda, L. M. C. Murillo, O. G. de Dios, G. Cossu, F. M. Facca, and E. Salvadori, "Using sdn for cloud services provisioning : the xifi use-case," in *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pp. 1–7, IEEE, 2013.

[123] C. Chen, C. Liu, P. Liu, B. T. Loo, and L. Ding, "A scalable multi-datacenter layer-2 network architecture," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, p. 8, ACM, 2015.

[124] J. Medved, R. Varga, A. Tkacik, and K. Gray, "Opendaylight : Towards a model-driven sdn controller architecture," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*, pp. 1–6, IEEE, 2014.

[125] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, *et al.*, "Onos : towards an open, distributed sdn os," in *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 1–6, ACM, 2014.

[126] "Floodlight project.."

[127] "Ryu controllers.."

[128] J. Kosinski, P. Nawrocki, D. Radziszowski, K. Zielinski, S. Zielinski, G. Przybylski, and P. Wnek, "Sla monitoring and management framework for telecommunication services," in *Networking and Services, 2008. ICNS 2008. Fourth International Conference on*, pp. 170–175, IEEE, 2008.

[129] Y. Cheng, W. Zhuang, and A. Leon-Garcia, "Call level service differentiation for efficient sla management," in *Global Telecommunications Conference, 2005. GLOBECOM'05. IEEE*, vol. 2, pp. 6–pp, IEEE, 2005.

[130] C. Ward, M. J. Buco, R. N. Chang, L. Z. Luan, E. So, and C. Tang, "Fresco : a web services based framework for configuring extensible sla management systems," in *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, pp. 237–245, IEEE, 2005.

[131] A. Schmietendorf, R. Dumke, and D. Reitz, "Sla management-challenges in the context of web-service-based infrastructures," in *Web Services, 2004. Proceedings. IEEE International Conference on*, pp. 606–613, IEEE, 2004.

[132] A. Paschke and M. Bichler, "Sla representation, management and enforcement," in *e-Technology, e-Commerce and e-Service, 2005. EEE'05. Proceedings. The 2005 IEEE International Conference on*, pp. 158–163, IEEE, 2005.

[133] C. He, L. Gu, B. Du, and Z. H. S. Li, "A wsla-based monitoring system for grid service-gsmon," in *Services Computing, 2004.(SCC 2004). Proceedings. 2004 IEEE International Conference on*, pp. 596–599, IEEE, 2004.

[134] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, "Web services agreement specification (ws-agreement)," in *Open grid forum*, vol. 128, p. 216, 2007.

[135] S. Bajaj, D. Box, D. Chappell, F. Curbera, G. Daniels, P. Hallam-Baker, M. Hondo, C. Kaler, D. Langworthy, A. Malhotra, *et al.*,

"Web services policy framework (ws-policy)," *Specification, IBM, BEA, Microsoft, SAP AG, Sonic Software, VeriSign*, 2004.

[136] P. Patel, A. H. Ranabahu, and A. P. Sheth, "Service level agreement in cloud computing," 2009.

[137] V. C. Emeakaroha, M. A. Netto, R. N. Calheiros, I. Brandic, R. Buyya, and C. A. De Rose, "Towards autonomic detection of sla violations in cloud infrastructures," *Future Generation Computer Systems*, vol. 28, no. 7, pp. 1017–1029, 2012.

[138] A. Biswas, S. Majumdar, B. Nandy, and A. El-Haraki, "Predictive auto-scaling techniques for clouds subjected to requests with service level agreements," in *IEEE World Congress on Services (SERVICES)*, pp. 311–318, IEEE, 2015.

[139] R. Yanggratoke, J. Ahmed, J. Ardelius, C. Flinta, A. Johnsson, D. Gillblad, and R. Stadler, "Predicting service metrics for cluster-based services using real-time analytics," in *11th International Conference on Network and Service Management (CNSM)*, pp. 135–143, IEEE, 2015.

[140] M. Alhamad, T. Dillon, and E. Chang, "Conceptual sla framework for cloud computing," in *Digital Ecosystems and Technologies (DEST), 2010 4th IEEE International Conference on*, pp. 606–610, IEEE, 2010.

[141] Y. Kouki and T. Ledoux, "Csla : a language for improving cloud sla management," in *International Conference on Cloud Computing and Services Science, CLOSER 2012*, pp. 586–591, 2012.

[142] E. Kamateri, N. Loutas, D. Zeginis, J. Ahtes, F. D'Andria, S. Bocconi, P. Gouvas, G. Ledakis, F. Ravagli, O. Lobunets, *et al.*, "Cloud4soa : A semantic-interoperability paas solution for multi-cloud platform management and portability," in *European Conference on Service-Oriented and Cloud Computing*, pp. 64–78, Springer, 2013.

[143] A. Andrzejak, D. Kondo, and S. Yi, "Decision model for cloud computing under sla constraints," in *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, pp. 257–266, IEEE, 2010.

[144] E. Commission, "SLALOM european project," 2016.

[145] A. F. M. Hani, I. V. Paputungan, and M. F. Hassan, "Support vector regression for service level agreement violation prediction," in *Computer, Control, Informatics and Its Applications (IC3INA), 2013 International Conference on*, pp. 307–311, IEEE, 2013.

[146] F. Caglar and A. Gokhale, "ioverbook : intelligent resource-overbooking to support soft real-time applications in the cloud," in *IEEE 7th international conference on Cloud computing (CLOUD)*, pp. 538–545, IEEE, 2014.

[147] "Cloud providers adoption assessment d4.2," 2016.

[148] D. F. B. Beyer, D. Bicket, "Cloud providers adoption assessment d4.2," 2016.

[149] "T-NOVA project website | t-NOVA, FP7, ICT, network functions VIrtualisation."

[150] G. Gardikis, I. Koutras, G. Mavroudis, S. Costicoglou, G. Xilouris, C. Sakkas, and A. Kourtis, "An integrating framework for efficient nfv monitoring," in *NetSoft Conference and Workshops (NetSoft), 2016 IEEE*, pp. 1–5, IEEE, 2016.

[151] T. Kim, S. Kim, K. Lee, and S. Park, "A qos assured network service chaining algorithm in network function virtualization architecture," in *Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on*, pp. 1221–1224, IEEE, 2015.

[152] R. Mijumbi, S. Hasija, S. Davy, A. Davy, B. Jennings, and R. Boutaba, "Topology-aware prediction of virtual network function resource requirements," *IEEE Transactions on Network and Service Management*, 2017.

[153] P. Chaignon, K. Lazri, J. Francois, and O. Festor, "Understanding disruptive monitoring capabilities of programmable networks," in *Network Softwarization (NetSoft), 2017 IEEE Conference on*, pp. 1–6, IEEE, 2017.

[154] V. Riccobene, M. J. McGrath, M.-A. Kourtis, G. Xilouris, and H. Koumaras, "Automated generation of vnf deployment rules using infrastructure affinity characterization," in *IEEE NetSoft Conference and Workshops (NetSoft)*, pp. 226–233, IEEE, 2016.

[155] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization : Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.

[156] P. Rengaraju, V. R. Ramanan, and C.-H. Lung, "Detection and prevention of dos attacks in software-defined cloud networks," in *Dependable and Secure Computing, 2017 IEEE Conference on*, pp. 217–223, IEEE, 2017.

[157] M. E. Ahmed, H. Kim, and M. Park, "Mitigating dns query-based ddos attacks with machine learning on software-defined networking,"

[158] M. Miyazawa, M. Hayashi, and R. Stadler, "vnmf : Distributed fault detection using clustering approach for network function virtualization," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pp. 640–645, IEEE, 2015.

[159] G. Zhang, B. E. Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks : : The state of the art," *International journal of forecasting*, vol. 14, no. 1, pp. 35–62, 1998.

[160] L. Zhu and X. Liu, "Technical target setting in qfd for web service systems using an artificial neural network," *IEEE Transactions on Services Computing*, vol. 3, no. 4, pp. 338–352, 2010.

[161] "Prometheus - monitoring system and time series database." `https://prometheus.io/`. Accessed : 2018-03-30.

[162] P. Racz, B. Stiller, *et al.*, "Monitoring of sla compliances for hosted streaming services," in *Integrated Network Management, 2009. IM'09. IFIP/IEEE International Symposium on*, pp. 251–258, IEEE, 2009.

[163] "Welcome to clearwater," 2016.

[164] C. Sauvanaud, K. Lazri, M. Kaâniche, and K. Kanoun, "Towards black-box anomaly detection in virtual network functions," in *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop*, pp. 254–257, IEEE, 2016.

[165] S. H. Makhsous, A. Gulenko, O. Kao, and F. Liu, "High available deployment of cloud-based virtualized network functions," in *International Conference on High Performance Computing & Simulation (HPCS)*, pp. 468–475, IEEE, 2016.

[166] "Welcome to sipp," 2016.

[167] "Stress-ng," 2016.

[168] "Welcome to clearwater - project clearwater 1.0 documentation." `http://clearwater.readthedocs.io/en/stable/`. Accessed : 2018-03-30.

[169] "Monasca - monitoring at scale.,"

[170] J. Kreps, N. Narkhede, J. Rao, *et al.*, "Kafka : A distributed messaging system for log processing," in *Proceedings of the NetDB*, pp. 1–7, 2011.

[171] R. A. Johnson and D. Wichern, *Multivariate analysis*. Wiley Online Library, 2002.

[172] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[173] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[174] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[175] H. Peng, F. Long, and C. Ding, "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 27, no. 8, pp. 1226–1238, 2005.

[176] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow : A system for large-scale machine learning.," in *OSDI*, vol. 16, pp. 265–283, 2016.

[177] A.-F. Antonescu and T. Braun, "Improving management of distributed services using correlations and predictions in sla-driven cloud computing systems," in *IEEE Network Operations and Management Symposium (NOMS)*, pp. 1–8, IEEE, 2014.

[178] J. Bendriss, I. G. B. Yahia, and D. Zeglache, "Sla enforcement in programmable networks," in *9th International Conference on Autonomous Infrastructure, Management and Security (AIMS)*, 2015.

[179] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural Networks*, vol. 18, no. 5, pp. 602–610, 2005.

[180] V. Pham, T. Bluche, C. Kermorvant, and J. Louradour, "Dropout improves recurrent neural networks for handwriting recognition," in *14th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pp. 285–290, IEEE, 2014.

[181] J. Bendriss, I. G. B. Yahia, and D. Zeghlache, "Forecasting and anticipating slo breaches in programmable networks," in *Innovations in Clouds, Internet and Networks (ICIN), 2017 20th Conference on*, pp. 127–134, IEEE, 2017.

[182] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proceedings of the 24th international conference on Machine learning*, pp. 473–480, ACM, 2007.

[183] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[184] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, pp. 2546–2554, 2011.

[185] "Tensorflow - an open source software library for machine intelligence.," 2016.

[186] C. Giraud-Carrier, R. Vilalta, and P. Brazdil, "Introduction to the special issue on meta-learning," *Machine learning*, vol. 54, no. 3, pp. 187–193, 2004.

[187] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv :1611.01578*, 2016.