FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Agent-based modeling framework for complex adaptive organizations

Diogo Pinto

FOR JURY EVALUATION

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Eugénio da Costa Oliveira

Second Supervisor: Henrique Lopes Cardoso

June 24, 2016

Agent-based modeling framework for complex adaptive organizations

Diogo Pinto

Mestrado Integrado em Engenharia Informática e Computação

June 24, 2016

Abstract

Humans achieved their place in the world due to constant approximations, simplifications and beliefs designed by the human brain. These were the foundations that evolution found to enable each individual to pursue his or her life goals, and potentiate the ones around them. Today's world is not that simple, and such "shortcuts" are not enough: the systems that surround us are too complex, and our minds can't keep the pace.

Complex systems both amaze and scare us, with their deceivingly simple base rules and unforeseen consequences. They stand out because the results of the activities of the individuals that constitute them are bigger than the sum of their parts. This kind of systems are found in nature, as is the example of ants behavior when searching for food: their search method is collaborative and very efficient, even though each one only obeys a simple restricted set of rules.

Media production companies find the complexity that arises from their environments hard to analyze and comprehend. As such, much of the potential for improvement is still out of reach from existing tools. In this context, this thesis aims to provide a solution that helps to reason over the emerging behaviors and interactions from media production environments. By answering the questions "what is happening", "what will happen", and "what would happen if", the knowledge gathered simplifies the system of interactions in such a way that insight can be harnessed, and therefore, action can be taken.

To accomplish its goals, this thesis subdivides into development of a generic platform for data analysis in streaming settings, and research on simulation of complex systems through the usage of deep learning techniques.

The platform is composed by multiple extensible modules, and aims to be instantiable in environments where analysis of behaviors and interactions in complex systems is relevant. Thus, it is not only applicable to the example of media production environments, but also to, *e.g.*, the Internet of things and social networks.

A simulation approach helps to complements the framework, and seeks to answer the "what if" questions. Simulations are used to both understand the world, and design for it, and therefore holds great potential when properly integrated. In what regards deep learning, at its essence it is also based on a complex system of non-linear computing modules. Therefore, this thesis also seeks to take advantage of an approach based on complex systems to model and abstract over other complex systems.

Resumo

O ser humano alcançou o seu estatuto graças a constantes aproximações, simplificações e crenças desenhadas pelo seu cérebro. Estas foram as fundações que a evolução encontrou para capacitar cada indivíduo de levar a sua vida, bem como potenciar a dos que o rodeiam. O mundo atual não é assim tão simples, e esses "atalhos" não são suficientes: os sistemas que nos rodeiam são demasiado complexos, e as nossas mentes não conseguem acompanhar.

Os sistemas complexos tanto nos fascinam como amedrontam, com as suas regras base aparentemente simples, mas consequências imprevisíveis. Estes sobressaem-se porque os resultados das atividades dos indivíduos que os constituem são maiores do que a simples soma das partes. Este tipo de sistemas são até encontrados na natureza, como é o exemplo das formigas quando procuram comida: o seu método de pesquisa é colaborativo e muito eficiente, apesar de cada uma apenas obedece a um conjunto de regras simples e pequeno.

As empresas de produção de multimédia tem grandes dificuldades em analisar e compreender a complexidade que surge dos seus ambientes de desenvolvimento. Assim sendo, muito do potencial para melhorias está ainda longe do alcance de ferramentas existentes. Neste contexto, esta tese tem o objetivo de fornecer uma solução que ajude a raciocinar sobre os comportamentos e interações emergentes dos ambientes de produção. Respondendo às perguntas "o que está a acontecer", "o que vai acontecer" e "o que aconteceria se", o conhecimento adquirido simplifica o sistema de interações de um modo tal que se podem efetivamente tomar ações informadas.

Para atingir os seus objetivos, esta tese subdivide-se no desenvolvimento de uma plataforma genérica para análise de dados num ambiente de fluxo constante de dados, e investigação sobre a simulação de sistemas complexos através de técnicas associadas a deep learning.

A plataforma é composta por múltiplos módulos extensíveis, e tem como objetivo ser instanciável em ambientes onde a análise de comportamentos e interações em sistemas complexos é relevante. Assim, não é só aplicável aos ambientes de produção de multimédia, mas também, *e.g.*, à Internet of Things e redes sociais.

Um método de simulação visa complementar a framework, e tem como objetivo responder a questões do tipo "o que aconteceria se". Simulações são usadas tanto para compreender o mundo, como para desenhar para este, e, assim sendo, tem grande potencial quando integrada apropriadamente. No que se refere ao deep learning, na sua essência este é também baseado num sistema complexo de módulos de computação não-linear. Em consequência, esta tese procura também tirar partido de uma metodologia baseada em sistemas complexos para modelar e abstrair sobre outros sistemas complexos.

Acknowledgements

I would like to start by thanking professor Eugénio Oliveira and Henrique Cardoso. Even though I was not able to interact with them as much as I would like to, through our conversations they always encouraged and empowered my (somewhat ambitious) goals while keeping me grounded to what is effectively achievable. This experience will definitely follow me in my future professional and research endeavors.

Additionally, I want to thank all my colleagues and friends at MOG Technologies, namely the CEO Luís Miguel Sampaio, who always made sure no necessity was felt, Daniel Costa and Álvaro Ferreira who worked close with us in the integration tasks, potentiated vastly the experience, and Pedro Henriques and Rui Grandão, with whom difficulties and experiences were shared, and mutual aid strengthened all our projects.

I could never forget professor Rui Camacho and professor Vítor Santos Costa, who provided the possibilities for me to grow in the field that I love to work, setting the slipway both for this thesis as well as future projects.

I also want to thank globally to the Faculty of Engineering, for the opportunities provided during this five-year adventure.

Finally, I thank the persons who are the anchors and supporting foundations of my life. Starting with my parents, they always made sure I had the raw materials to achieve my full potential in a conscious manner, sacrificing themselves many times to ensure that. My brother and little sister, for their aid is constant and independent of pre-conditions. And, last but (definitely) not least, I thank my beloved girlfriend, for our complicity knows no bounds.

Diogo Pinto

"Intelligence is not the product of any singular mechanism but comes from the managed interaction of a diverse variety of resourceful agents. "

Marvin Minsky

"It is a profoundly erroneous truism (...) that we should cultivate the habit of thinking of what we are doing. The precise opposite is the case. Civilization advances by extending the number of important operations which we can perform without thinking about them. "

Alfred North Whitehead

Contents

List of Figures

List of Tables

Abbreviations

AADL	Architecture Analysis & Design Language
ABM	Agent-Based Model(ing)
AI	Artificial Intelligence
ANN	Artificial Neural Network
AOSE	Agent-Oriented Software Engineering
API	Application Programming Interface
CNN	Convolutional Neural Network
CPS	Cyber-Physical System
CPU	Central Processing Unit
DSL	Domain-Specific Language
EDA	Exploratory Data Analysis
FIPA	Foundation for Intelligent Physical Agents
fMRI	functional Magnetic Resonance Imaging
FP	Functional Programming
GPU	Graphics Processing Unit
IoT	Internet of Things
JADE	Java Agent Development Framework
JVM	Java Virtual Machine
LSTMN	Long Short-Term Memory Network
MAS	Multi-Agent System
ML	Machine Learning
MOA	Massive Online Analysis
NLP	Natural Language Processing
NN	Neural Network
ODAC	Online Divisive Agglomerative Clustering
OOP	Object-Oriented Programming
RAM	Random Access Memory
RBM	Restricted Boltzmann Machine
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SVD	Singular Value Decomposition
TCP	Transmission Control Protocol
UI	User Interface
UML	Unified Modeling Language
UX	User Experience
VFDT	Very-Fast Decision Tree
WEKA	Waikato Environment for Knowledge Analysis

Chapter 1

Introduction

The history of the universe is dictated by how complexity is able to emerge from entropy. From the formation of the first atoms of hydrogen and helium, to the emergence of DNA, the universe tries to build complexity layer after layer, in a slow and chaotic learning process. The human species, with the ability of speech and writing, further boost this learning process' velocity, and now the most complex systems we know emerge from our interactions, *e.g.* through traveling and the internet, creating what are also living dynamic complex systems.

This thesis explores complexity, and how it can be dealt with to extract value through current methodologies. Specifically, it analyses this problem through the lens of media production environments, entities which have determinant roles in everyone's lifes, as they significantly contribute in shaping our knowledge of the world.

1.1 Context

The content broadcasted to every home's television flows from video cameras to end users by traversing a complex and interleaved series of transferring and transforming processes inside a media production environment. Additionally, the data involved is aggregated in a diverse range of big media assets, that need to be efficiently handled.

Those processes are encapsulated over the abstract concept of *ingest steps*, *i.e.* the process of capturing, transferring, or importing different types of video, audio, or image media (*i.e.* media assets) into a given editing or storing facility. Such processes are already efficiently tackled by modern solutions, even though in a shallow manner: there is no analysis over the system of ingest steps.

That is the case because the setting where these processes happen is still very unstructured, facing many deficiencies in what regards the working methodology of content editors and producers. This leaves a wide range of optimizations (and subsequent profits) to be made, that are

not yet harnessed by existing systems. But there are no straightforward solutions to any of these problems, and they mainly involve a complex conjunction of several smaller solutions.

1.2 MOG Technologies

MOG Technologies is a worldwide supplier of solutions for improving workflow performance of broadcaster companies. The MOG's product line focuses on materializing the abstraction provided by the ingest step concept, and provide a full media management system.

MOG Technologies experience led them to devise a new project: Skywatch. This new system aims to, through machine learning and big data-related techniques, take advantage of the unexplored layers of complexity over ingest steps to enhance their already state of the art functionality. This enhancement is achieved through conferring a greater degree of autonomy and efficiency over the actions performed in the system, while altogether boosting the media production teams' own productivity.

1.3 Motivation and Goals

As it was aforementioned, there is still a lot of value to gain from analyzing the events in media productions environments. As the ingest steps essentially encapsulate those events, they essentially define most of the very own environment where they take place. By building on top of current systems that already manage and record these actions, this is the par excellence opportunity to introduce such optimizations.

As such, this thesis hypothesizes that, by taking advantage of the information gathered and generated in these ingest steps (*e.g.* sources and destinations, transformations, nature of the assets), it is possible to provide a versatile high-level way to reason over the behavior of the individual agents (*e.g.* people or content servers), as well as over the whole system of interactions. Concretely, this is achieved through the following dimensions:

1. To describe what is happening.
   e.g. time-series distribution of errors in ingest procedures; distribution of activity time throughout resources.

2. To predict what is going to happen.
   e.g. availability of asset repositories; spikes in activity in the servers.

3. To test what would happen if something was true.
   e.g. if a given resource becomes full, what is the impact in the rest of the resources.

If one abstracts away from the specifics of the context that this hypothesis targets, it is possible to understand that this environment naturally fits under the concept of a complex system, *i.e.* a system where from the interactions originated by the agents' simpler behaviors, complexer

phenomena emerges[1]. Therefore, it is also possible to abstract the prior hypothesis, widening its range of applicability to many other use cases. One such example is software development teams, where the model can be fed by, *e.g.*, a version control system. As the number of programmers, system designers and architects grows, there have to be efficient ways to ease communication and organize the human resources into smaller and efficient groups. Another example is social networks. The number of users and actions taken in these systems is very high, and underneath that dimension lies great value in understanding how simple user actions map to, *e.g.* politic decisions and movements.

Next, the benefits that can be accomplished with such solution over two other use cases are briefly discussed: enhancement of social web-based services (which involves in greater extent behavioral modeling), and analysis of cyber-physical systems.

1.3.1 Social Web-Based Services

There is a growing interest in enhancing the UX [2] of web-based services through experience personalization, specially in services where users interact among themselves. One heuristic for achieving this is to model the users' interaction behaviors, and then simulate scenarios, so that the system is able to adapt to them, if they ever occur in the real world. In addition to being a relevant problem for the development of machine learning in general, efforts are being made to accomplish this task even by big companies in the artificial intelligence field such as Google [SHM+16], who saw in the quest of Go game mastering a bridge to enhance their own services personalization.

1.3.2 Cyber-Physical Systems

The field of CPS[3], usually referred to as IoT[4] or Industrial Internet, merges physical and virtual environments, being particularly interesting for applying this thesis proposal. These systems are gaining ground and will certainly shape our lives as we know them. On the other hand, they are inherently complex: the possibilities for interaction between devices grow exponentially with the number of devices, therefore, this complexity has to be properly analyzed and maintained.

This thesis final hypothesis is, then, that it is possible not only to understand how higher-level system behaviors emerge from lower-level entity behaviors, but also how that mapping is performed. To accomplish this, this thesis has the goal of developing a scalable framework for modeling inter-related agents' behaviors in the context of complex systems, that also enables extraction of descriptive and inferential knowledge in real time about different complexity layers.

[1] This topic is further discussed in Section 2.2
[2] User Experience
[3] Cyber-Physical Systems
[4] Internet of Things

1.4 Contributions

The contributions of this thesis consist of not only solving problems faced everyday by media producers and editors at broadcasting companies, but also of providing a reusable and extensible open analytics platform for behavior analysis in complex systems to the scientific community (namely in the field of CPS). That system supports:

- a real-time simulation environment for complex systems, fed from a data stream;
- induction of resource's behaviors from the data stream;
- analysis over different layers of the complex system (individual, sub-graphs and global);
- provide predictions of future events;
- scale to reasonably-sized complex systems;
- enable features' extension through additional modules.

1.5 Document Structure

This document is structured as follows:

- **Chapter 1. Introduction:** Description of the context and motivation for the work, together with main goals and expected contributions.
- **Chapter 2. Literature Review:** Review of the work developed in the fields of Complex Systems, Cyber-Physical Systems, Agent-Based Modeling, Real-Time Data Mining, and Maintenance and Usability.
- **Chapter 3. Open Streaming Analysis Platform:** Using the knowledge gathered in the literature review, a solution is designed and developed.
- **Chapter 4. Simulation Module with Deep Learning:** Focusing specifically in the simulation module, it is studied how deep learning methodologies may aid in the simulation of complex systems.
- **Chapter 5. Conclusions and Future Work:** Final considerations and future directions.

Chapter 2

Literature Review

This thesis is built upon current efforts in understanding behaviors, and in scaling that analysis to complex systems' level. Some projects and research proposals are presented in this chapter, which directly or indirectly influence the decisions taken in the thesis development.

This Chapter starts with a brief clarification of concepts recurrently used throughout the document in Section 2.1. Being the main focus of this thesis, complex systems are reviewed in Section 2.2. Throughout this document, the Cyber-Physical Systems are used as an example of a complex system. As such, a proper overview of them is done in Section 2.3. Moving to strategies of how this thesis hypothesis may be approached, multi-agent-based computing usefulness both in the task of engineering the software tool and in modeling the entities in the simulation environment is discussed on Section 2.4. Going deeper in what regards the simulation environment, a review of Behavioral Cloning in Section 2.5 provides inspiration on how to tackle the problem. Moving forward, Section 2.6 introduces the Big Data era, and how latest developments boost CPS potential. This sets the tone for a careful analysis over real-time data mining in Section 2.7, as well as some key algorithms. The chapter ends with some considerations on maintenance and usability questions in Section 2.8 that guided the development of this project.

2.1 Concepts Clarification

Throughout this document a couple of concepts will be recurrent, and therefore clarified next:

- A **target function** is the true function we seek to model.
- An **hypothesis** is an educated guess about a function believed to be similar to the target function;
- A **model** is the subsequent testable manifestation of an hypothesis;
- A **learning algorithm** is a set of instructions devised to model the target function given a training dataset.

- A **classifier** is a special case of hypothesis, where the target for each data point is a set of categorical labels;
- **Hyper-parameters** are those parameters that constraint the possible model space for a given learning algorithm;
- **Model parameters** are, in contrast to hyper-parameters, the parameters learned by training the learning algorithm.

2.2 Complex Systems

Complex systems are systems made of many similar, interacting parts. Each part behaves according to a relatively simple set of rules, but from the whole system certain properties emerge that are not just a linear composition of its parts [Sor07].

One of the richer sources of complex systems is nature itself. This happens because natural selection finds in complexity a tool for communities to be able to adapt and persevere. Many animal species – such as ants, dolphins and bats – were already studied with the aim of understanding how from their simple behaviors complex societies emerge [KPS11]. Humans, being also a highly social specie, also give rise to complex properties from their (arguably) complex behavior [Sor08, SMG14].

Complex systems also greatly influence computing science. For example, the fundamental aspect of neural networks is that they aggregate many cells that map linear data transformations over non-linear functions, and share the result with their neighbors. This way, they achieve the ability to model highly complex functions[1]. Additionally, swarm intelligence studies originated ant colony optimization, which is inspired from natural ants' behavior to minimize objective functions within certain restrictions [DBS06].

Uncovering both the lower-level rules and how they map into the emergent properties of complex systems proves to be a challenge. Fundamentally, this thesis objective is to devise an empirical method for understanding them, through analysis of behaviors at their different complexity levels.

In the next Section, Cyber-Physical Systems are presented as a major impacting kind of complex system.

2.3 Cyber-Physical Systems

CPS are abstractly defined as the integration of virtual computation over physical systems [SGLW08]. They are expected to permeate our every-day lives in the near future, with a scope that ranges from transportation to health and energy. Due to recent improvements and democratization of data mining techniques, vast investments are being made for CPS development. As a matter of

[1] This topic is further extended in Section 2.7.2.

fact, according to a study by McKinsey Global Institute, Internet of Things (IoT)[2] has an economic potential of up to $11.1 trillion a year by 2025 [MCB+15].

Nonetheless, to harness their full potential, integrated systems must be coordinated, distributed, and connected, while assuring robustness and responsiveness [Lee08].

This unique combination of both high complexity and high pervasive power gives rise to many difficulties, and not only in what regards maximizing their utility. Cyber crime is one such problem, which already represents $100 billion of loss in the United States of America alone [The13]. Depending on the criticalness of the role performed, CPS may also be subject to timing predictability (*i.e.* the system must react within a specific time frame), and also to energy consumption restraints. Arguably more important are even the human, social, philosophical and legal issues that arise, which demand robust assurances from the CPS. Together with the dynamism and heterogeneity of the systems that underlie CPS (*e.g.* network standards), they prove to be a major challenge to reason about.

To ease the complexity of building such systems, one of the most promising approaches is to intertwine the development of the system with a simulation of it. Zhang [Zha14] proposed an architecture for modeling CPS through an object-oriented design, where he integrates AADL[3] [Hug13], ModelicaML [Sch09], and clock theory [Jif13]. AADL is used to model and analyze both software and hardware architectures of embedded systems, which comprises hardware and software components' description. Modelica is a language for virtually defining physical systems [Ass10], and ModelicaML enables definitions that use UML[4] class diagrams syntax to be transformed into Modelica language.

One of the aims of this thesis is to not only enable the definition of resources behaviors, like the system demonstrated by Zhang, but also to introduce in these systems effective ways to induce those behaviors from execution, therefore expanding the scope of these CPS simulations to other grounds.

2.4 Multi-Agent-Based Computing

Software systems vary widely in range of complexity. Numerous ways have been discussed and concretely implemented that aim to improve the ability of programmers and system designers to cope with such complexity, as is the example of the OOP[5] paradigm, from which a series of well-known design patterns have been derived [GHJV95], and UML [Omg98].

As levels of complexity grow, there is a trade-off between relaxation of optimal solution and computational power needed. To answer those questions, MAS[6] aims to divide complexity into

[2] the coarse term for CPS
[3] Architecture Analysis & Design Language
[4] Unified Modeling Language
[5] Object-Oriented Programming
[6] Multi-Agent System

manageable computational entities, with well-defined responsibilities [Bon02]. These computational entities interact between themselves according to specific protocols. This division of responsibilities leads also to an efficient distribution over multiple computing nodes.

This intuitive way to deal with complexity is extended even to the software design stage, in the form of AOSE[7]. For that, there exist several well-defined methodologies, like Tropos [BPG+04], Gaia [WJK00] and PORTO [CRO16]. These methodologies are successfully used in various examples, like the managing of an airline operations control centre by Oliveira and Castro [CO08]. Muller and Fischer, while reviewing the actual impact of MAS [MF14], present one other example of Google, which uses a MAS for automated bidding for their advertisement slots.

These fundamental principles affect this thesis in two ways:

- AOSE concepts are used to cope with the architectural complexity of the system.
- Each entity that is mapped into the system is modeled as an agent. That agent learns its behavior from the external entity, and reproduces it in the simulation environment. This results in a merge of ABM[8] and behavioral cloning[9], as a way to cope with behaviors in complex systems.

2.5 Behavioral Cloning

With the democratization of machine learning techniques, behavioral analysis became very relevant: such knowledge may lead to great and effective improvements in systems' UX[10]. This analysis can be accomplished mainly in two ways: by describing the behavior, or by really modeling (*i.e.* cloning) it, which is a task whose nature represents more of a challenge, but also holds the potential for greater rewards, as it is possible to both have access to a user's behavior on-demand, as well as to synthesize new behaviors.

Recommender systems essentially accomplish the task of describing users' behaviors: in seeking to extract patterns from how many users interact with a given system, the models built extract a set of distinct behaviors that tries the maximize the number of user actions justified by those behaviors. These systems can be divided into two groups: content based recommendation and collaborative recommendation [Par13]. The content based approach creates a profile for each user based on her characteristics and her past. The utility of a new item s for user u is estimated by looking at items s_i, assigned to the users that have similar profile to u. On the other hand, the collaborative approach predicts the utility of a new item s by looking at the utility of the same item s assigned to users with similar assigned items in the past.

Both methods have their strengths and weaknesses. The collaborative approach is considered to be more accurate than the content-based one, but suffers from the "cold start problem" due to the need for a large base of users in order to perform valid recommendations.

[7] Agent-Oriented Software Engineering
[8] Agent-Based Model(ing)
[9] Further discussed in Section 2.5.
[10] User Experience

A significant boost to research in recommender systems was due to NetFlix. This company offered a prize of $1,000,000 to the first person or team to beat their recommender system, called CineMatch, by 10%. The winning entry of the competition was a combination of several algorithms that had been developed independently. They used several SVD[11] models, including SVD++ (a slight modification to the original SVD algorithm), blended with RBM[12] models [Gow14].

Other recent efforts aim to model behaviors in a deeper level. One such effort was employed by Google, with the task of building AlphaGo, a system built to master the game Go using deep learning[13] [SHM+16]. Go is a game known not only to require high reasoning and logic skills from the players, but also demands creativity and imagination. This happens because of the enormous number of possible board states, roughly approximated to be 2.082×10^{170} [TFF+15]. One of the essential steps on the training stage was to mimic the behavior of professional players through reinforcement learning, and only after improving through playing the game against itself. With this approach, by copying how humans play, there is no need to parse the whole search tree for optimal moves. This confers a certain degree of "intuition" to the algorithm itself.

Probabilistic programming is one other category that holds the potential to further improve behavioral cloning performance. Lake *et al.* introduced a method to induce probabilistic programs when learning new concepts [LST15]. This process achieves learning behaviors more similar to what humans accomplish: generalizing from few examples, and learning to learn, *i.e.* increasing the learning performance of new concepts from past experience.

Video-games industry has already been modeling behaviors for a long time. In this environment, AI[14] reasoning has to be encoded in a way both familiar to designers and programmers.

Behavior tree is a framework for game AI, devised due to the lack of expressiveness and reusability of finite state machines in the job of describing AI reasoning [Ogr12, Isl05, Lim09, CMO14]. Instead of exposing the logic at the state level, behavior trees describe the flow of logic, where a given behavior is determined in an event-driven fashion. They are tree-like structures whose constitution varies, but generally there are three kinds of nodes: predicate nodes, sequence nodes and select nodes; where predicate nodes may have a mix of interactions and conditional checks with the world, sequence nodes are parents of other nodes that are executed in a specific order and that must return *True* for the parent to return *True*, and select nodes execute all its child nodes, returning *True* if one of them returns *True*. An example is showcased in Figure 2.1.

Due to the natural properties of tree representation, discussed here and expanded in Section 2.7.1, the adaptation of decision trees' induction algorithms to the structure of behavior trees presents an useful algorithm to use in this thesis context.

[11] Singular Value Decomposition
[12] Restricted Boltzmann Machine
[13] Topic further discussed in Section 2.7.2
[14] Artificial Intelligence

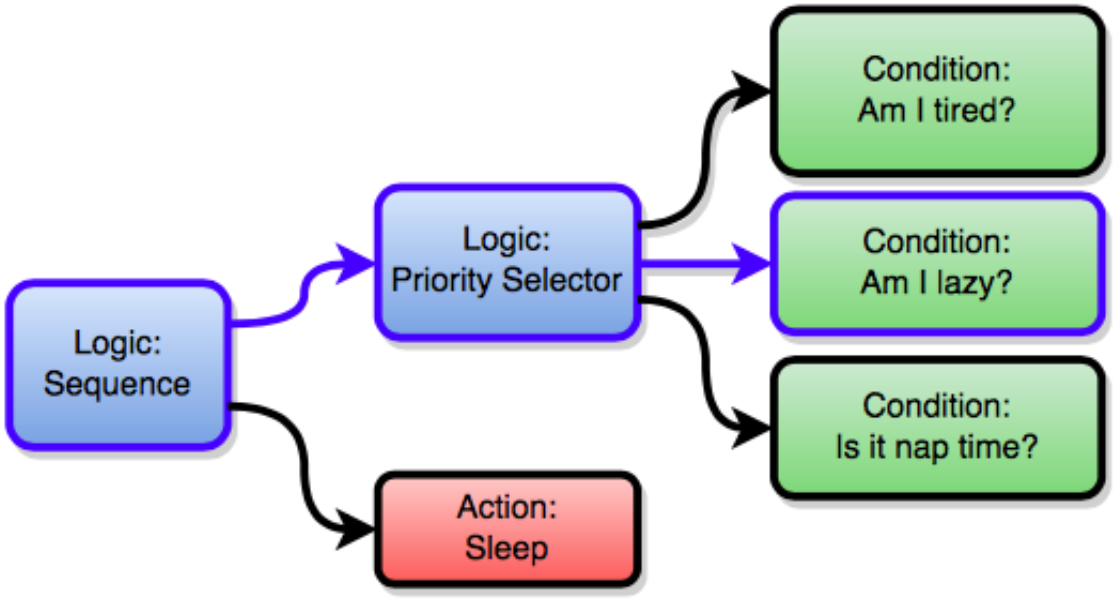


Figure 2.1: An example of an behavior tree, showcasing the Sequence and Priority Selector nodes: if any of Priority Selector node child conditions is met, that node returns and the following child of the Sequence node runs, otherwise do not. (Source: \hyphenation{https://gamedevdaily.io/managing-ai-in-gigantic-523dc84763cf})

2.6 Big Data

Fundamentally, the development of cyber-physical systems has been restricted by the cost of computer chips, the dissemination of the internet, and computing capacity to process large amounts of data in real-time. With the current democratization of the first two restrictions, big data-related techniques play a major role in what can be achieved with CPS. Big data is a broad concept, and is many times used in different contexts, but all its definitions have in common that it relates to the usage of vast amounts of data. It is usually described according to the following aspects (known as the seven Vs)[Zha14]:

- **Volume:** data is in the order of terabytes or larger.
- **Velocity:** capture, transfer, computation, store and access of data is time critical, and commonly performed in real-time.
- **Variety:** data may have different types and be in various degrees of structuredness.
- **Veracity:** data may contain uncertainty or impreciseness.
- **Validity:** there is a need for ensuring that both the measurements are correct, and that the process is sufficiently transparent.
- **Value:** big data brings down past limits on what can be accomplished with vasts amounts of data.

- **Volatility:** data may vary in degree of volatility, being many times susceptible to only one usage, at its arrival.

One very prominent branch of big data is real-time data mining. As it is also the branch where this thesis fits on, it is discussed in next Section.

2.7 Real-Time Data Mining

Traditional data mining techniques were developed and applied having static datasets in mind, *i.e.*, where all the data is available at the learning stage. That is the direct result of two assumptions: the data is generated from a stationary distribution, and we are able to gather enough data before the system is deployed into production. This led to the common methodology of retraining a classifier from time to time, as its performance deteriorates.

A widely-spread study performed by DOMO [DOM15] on several well-known providers of web-services tries to alert to the fact that data is growing in size and speed at exponential rates: in 2015, every minute, an approximate average of 350 thousand tweets are published on Twitter and 4 million posts on Facebook.

In this context of data streaming, traditional data mining strategies suffer from starting with the assumption that it is possible to gather the needed data for training before deploying: if we try to use vasts amounts of data, it is difficult to scale the methodologies and obtain result in a reasonable time-frame, but if we downscale the data size, the models we train may be nowhere near useful.

Altogether, data streams are characterized by:

- the data arriving sequentially, fast, dynamically and asynchronously;
- the streams are generally considered to be infinite;
- the data distributions change over time;
- the objects that arrive are unlabeled.

To answer the need for dealing with that different nature of data, real-time data mining emerged as a natural evolution of traditional data mining. Due to the nature of data streams, the developed techniques start off with the following assumptions:

- data must be accessed only once;
- data cannot be stored in (main) memory;
- decision models must be continuously updated.

This leads, essentially, to decision models that must be able to incorporate new information, detect changes and adapt accordingly, and to forget outdated information.

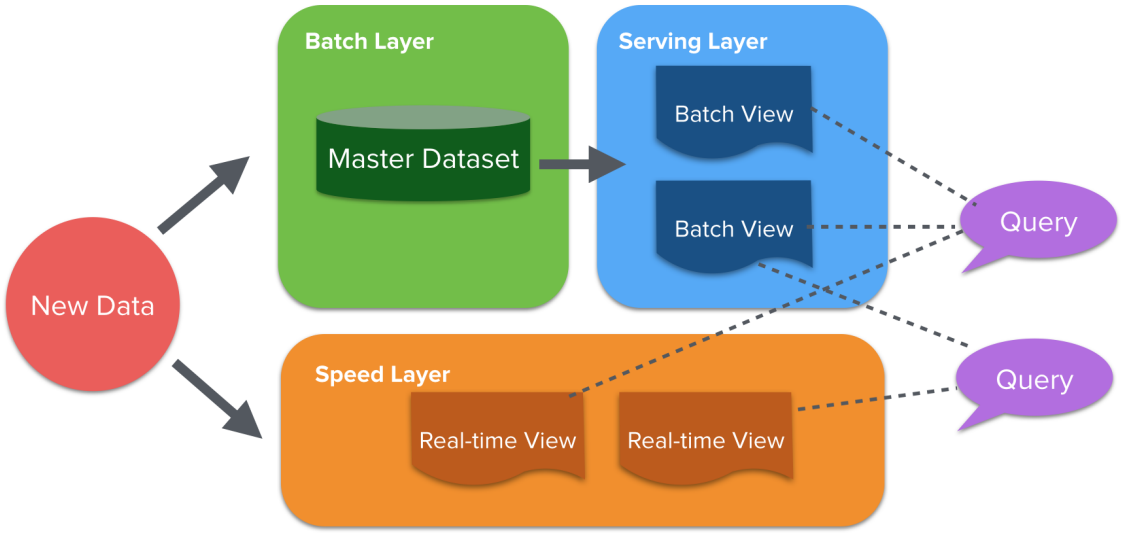


Figure 2.2: Generic schema for Lambda Architecture deployments (Adapted from: http://lambda-architecture.net/

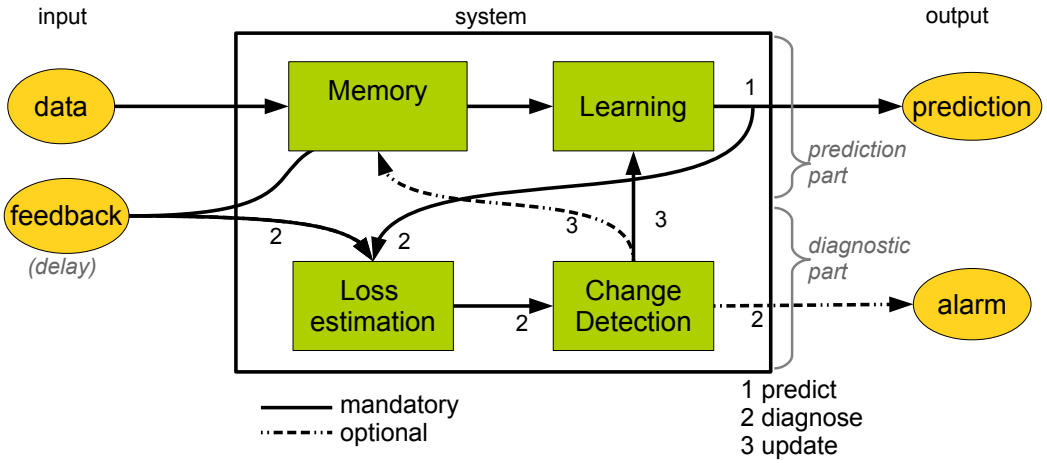


Figure 2.3: Generic schema for online adaptive learning algorithms [GŽB+14]

Currently, many deployments of streaming systems is performed according to the Lambda Architecture, whose schema is shown in Figure 2.2. Originally proposed by Nathan Marz [MW13], it focuses on managing the trade-off between batch systems, properly optimized but with higher latency, and streaming systems, with lower latency but also lower performance, combining their results in an adaptable service for its clients.

Gama *et al.* [GŽB+14], with the schema presented in Figure 2.3, demonstrate the architecture of a robust system, solely based in online learning. As a consequence of these systems' target of being deployed only once, they must be highly adaptable in their core. Therefore, not only the prediction part is relevant in the architecture, but a self-diagnosis part that provides feedback also represents a key feature for them to be reliable.

2.7.1 Decision Trees

Decision trees are widely used due to producing results that are very intuitive to interpret. Therefore, their use to model behaviors may provide a deeper insight into why certain decisions are performed.

They are one of the most common and traditional algorithms in AI, and the most common specifications are ID3 [Qui86] and C4.5 [Qui92]. Even though, these specifications are designed to work in a batch setting: all the data is available at model's creation time.

For the task at hand, the behavioral information arrives in a continuous stream of data. For that setting, Domingos and Hulten introduced VFDT[15] [DH00]. As they use the concept of Hoeffding bound in tree construction stage, the performance is guaranteed to converge asymptotically to the one of a batch algorithm fed with infinite data.

Nonetheless, this method still has the disadvantage of considering that the distribution from which the data is obtained is static; in order to address that, VFDT was adapted into CVFDT [HSD01], introducing the notion of concept drift into the algorithm.

2.7.2 Deep Learning

Neural networks are networks of simple processing units that collectively perform complex computations. They are often organized into layers, including an input layer that presents the data, hidden layers that transform the data into intermediate representations, and an output layer that produces a response.

Neural networks started as crude approximations of the mechanisms found in neuroscience studies. Even though, their value was only discovered when computing power enabled both the processing of high volumes of data, and also the construction of bigger neural networks. This gave rise to the concept of *deep learning*, which characterizes neural networks with at least one hidden layer.

Neural networks learning process is most often based on gradient descent over a given objective function. The computed gradients at the output layer are back-propagated through the network

[15] Very-Fast Decision Tree

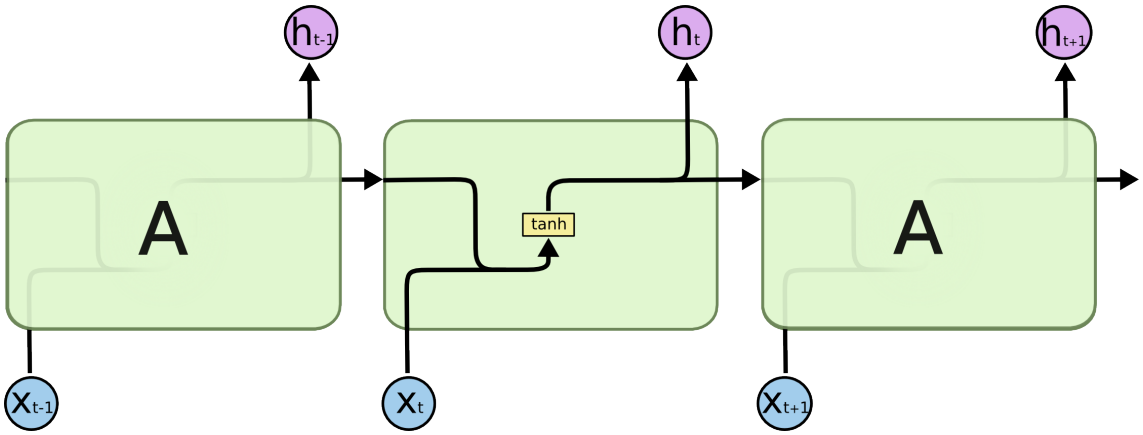


Figure 2.4: The repeating module in a RNN. (Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

layers, using the derivatives' chain rule, until the input layer. The weights used in the computations are then iteratively modified according to the gradients, in the direction that improves the objective function.

One of the earliest use cases that reborn this interest in neural networks was in computer vision: LeCun developed a CNN[16] named LeNet, to recognize handwritten characters in documents [LBBH98]. This neural network used multiple convolutions to abstract classification from position and scale in spatial representations. At its essence, these convolutions work like trainable filters, that activate when they detect some kind of pattern in the input data.

Due to their training being performed using stochastic updates (*i.e.* learns from examples), and are able to detect hidden and strongly non-linear dependencies, neural networks are also naturally suited for online learning. Nonetheless, the application of deep learning to data streams demanded also that they were able to produce decisions not only according to each sample of data that arrives,

[16]Convolutional Neural Network

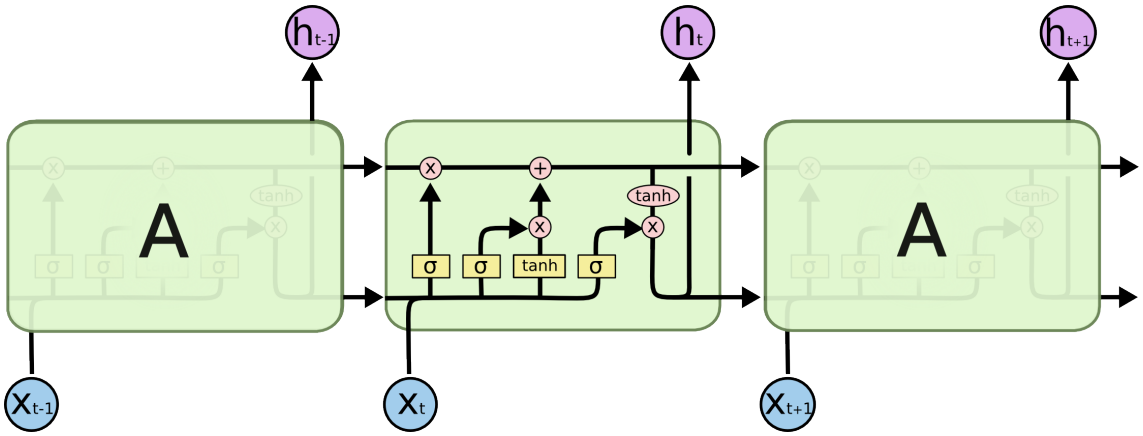


Figure 2.5: The repeating module in a LSTM. (Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

but using past information as well.

The answer for that was found in recurrence, with the proliferation of RNN[17] usage. Nonetheless, the "vanilla" architecture of these networks, showcased in Figure 2.4, was only competent in producing decisions when only the near past of data samples are relevant.

In that context, Hochreiter and Schmidhuber introduced LSTM[18] Networks [HHSS97], whose original architecture is showcased in Figure 2.5. They introduced into the cell's definition a long memory component, alongside four gates, which enabled the reliable usage of old knowledge from the sequence data into making decisions. These neural networks are proving to be very exceptionally powerful in sequence analysis [LBE15], and are even being used together with RL[19] techniques in order to not be fully dependable on data [SZR11].

For a broad review on the topic of deep learning, refer to the work of LeCun *et al.* [LBH15].

2.7.3 Graph Analysis

Behind a complex system is a network that defines the interactions between its components. By analyzing the underlying network, then, we can better understand the whole system.

Traditionally, most of the analysis performed on graphs were either at node and edge level, or at global level. As an example of this, Brin and Page incorporated in the Google search engine the node classification into hubs or authorities [BP98].

But nowadays, graphs commonly take dimensions ranging from 10^9 to 10^{23} nodes, representing social networks, chip designs or even the human brain. With this increasing relevance of complex systems, higher-level concepts have the potential to provide greater insight when used in the analysis.

Sub-graphs are being widely used as the building blocks of the larger complex systems they define. Milo *et al.* used sub-graphs distributions to characterize and discriminate graphs from different natures [MIK+04]. Based on that, Milo *et al.* introduced the concept of network motifs [MSOIK02], defining them as recurring (*i.e.* with high frequency) and significant (*i.e.* that are more frequent than expected) patterns of interconnections (*i.e.* the induced sub-graphs). Ribeiro and Silva further introduced G-tries as an efficient data structure for discovering network motifs [RS10].

These abstraction ideas prove to be very useful, and they are even found in the human brain's architecture, as a way to manage complexity. Through analysis of fMRI[20] experiments, Taylor *et al.* found that abstract thoughts are hierarchically dependent between themselves, where thoughts with higher-level abstract nature result from information aggregation, combination and refinement of lower-level thoughts [THBS15].

When analyzing a graph in its whole, one relevant analysis is community detection. Also referred to as clustering, it essentially bifurcates into inclusive clustering (*i.e.* a single node may

[17] Recurrent Neural Networks
[18] Long Short-Term Memory
[19] Reinforcement Learning
[20] functional Magnetic Resonance Imaging

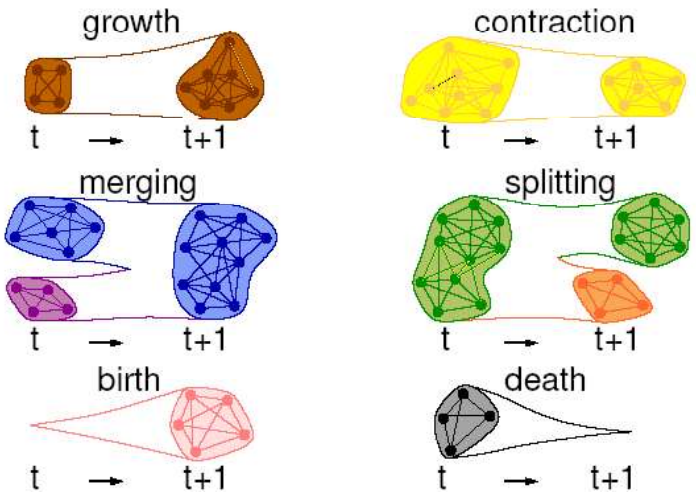


Figure 2.6: Events on graph evolution [PBV07]

belong to more than one cluster) and exclusive clustering (*i.e.* a single can only belong to one of the clusters).

Traditional exclusive clustering algorithms, like K-Means [Mac67], only work in batch setting, and therefore are not well-suited to the problem at hands. One of the first works is from Fisher [Fis87], who introduced CobWeb, a method for incremental clustering, which organizes the observations into a classification tree, enabling the prediction of the class for a new observation. Additionally, Aggarwal *et al.* introduced CluStream [AWC+03], which divided clustering process into two stages: an online one, where summary statistics were being updated according to the data stream, and another one, this time offline, which consists on answering the user queries. P. Rodrigues *et al.* introduced ODAC[21], which uses hierarchical clustering as the basis to perform progressive merges and partitions of clusters.

Analyzing graphs' clusters in the context of data streams presents an even more interesting aspect: the study of clusters' evolution. Palla *et al.* lists 6 different cluster events [PBV07], shown in Figure 2.6: growth, contraction, merging, splitting, birth, and death. Grene *et al.* provided a concrete implementation, based on those principles [GDC10].

2.8 Maintenance and Usability

Despite this thesis major focus on theoretical questions on how to cope with complexity, the resulting artifacts of this thesis are to be coupled into an existing tool (MOG's Skywatch), already deployed within a large volume of clients. Therefore, both maintenance and usability of the system should be key aspects from which to devise design and implementation decisions.

[21] Online Divisive Agglomerative Clustering

Sculley *et al.* warns about the tendency of easy and fast deployment of ML[22] systems, but difficult and expensive maintenance over time [SHG+15]. He argues that ML systems incur in an high technical debt by combining existing problems in software engineering with specific ones to ML, and this debt is not easily paid by usual methods, *e.g.* refactoring or improvement of unit tests. ML brings to the equation erosion of system's boundaries, data dependencies, and other problems with it, that must be carefully checked for anti-patterns.

Visualizations are a major vehicle for transmitting data and interacting with the users, and through careful planning of elements' visual arrangement, it is possible to control better how the end user interacts with the system [Shn96].

Dietvorst *et al.* [DSM15] alerts to other phenomena that threatens data-based systems: algorithm aversion. Empirically, he demonstrates that humans lose confidence in algorithms faster than in humans, even if the algorithms' error-rate is lower. For any reasonably sized and interesting problem, we can only asymptotically approximate a system's error to zero, but there are other ways to prevent consequences' escalation. Ideally, the system should persuade, by providing explanations and insight over the reason of its outputs, and also adapt, not only to new data, but also to users' feedback. Regarding the first aspect, a tree-like structure, discussed in Section 2.7.1, is expected to enable higher persuasion than, *e.g.*, a NN[23]. As of the latter aspect, the architecture by Gama *et al.* shown in Figure 2.3 addresses it, reinforcing the great need for adaptability in streaming environments.

Altogether, arguably the major evaluation metric for a given software project is how large the degree of the target users' usability is. This focus should permeate the design and implementation decisions throughout the development life-cycle of software.

2.9 Conclusion

This chapter reviewed the theory, algorithms and tools that support this thesis development.

While complex systems pose a big challenge, their value is very prominent, and by using the correct approximations, it is possible to tackle them in feasible time. On the other hand, behaviors are a delicate subject to model, but the adaptation of robust techniques that work in streaming environments may provide a useful insight to make concrete decisions. Additionally, the existing options for CPS augments the potential that inducing the resource's behaviors brings to CPS simulations is highlighted.

Both older but more robust knowledge, like agent-based computing, and very recent research, as is the example of online learning, are merged into a tool that provides new ways to analyze behaviors over complex systems.

[22] Machine Learning
[23] Neural Network

Chapter 3

Open Streaming Analysis Platform

Streaming-based analytics, reviewed in Section 2.7 systems are ascending into a major role in many organizations, as a way to rapidly adapt to new data, removing the need for traditional periodic batch analysis and model building. In this chapter, one such system is presented, targeting the needs of MOG in multimedia production environments. Additionally, the system is implemented as a platform for open source analytics on streaming settings: a platform that targets easy development and collaboration on analytics modules independently of the nature of the streaming setting they are to be deployed.

Assembling the knowledge gathered in the literature review into a solution proposal, this chapter starts with an overview of the problem in Section 3.1, followed by a discussion on the most relevant design decisions in Section 3.2, after which a description of the architectural approach is done in in Section 3.3. Finally, an overview of each module's implementation is discussed in Section 3.5, and an anomaly detection service implemented on top of the platform is shown in Section 3.6.

3.1 Problem Overview

The platform aim is to support the analysis of complex systems in the form demonstrated in Figure 3.1. The information about this system arrives in a (virtually) infinite, time-ordered, event stream, where each event represents an interaction between two entities, that may contain additional information besides the source and target entities, and time of occurrence. Throughout this document, such systems are referred to as interactions graphs.

In the context of media production environments, the interactions graph entities are, *e.g.*, Skywatch instances, Media Asset Managers or editors' workstations, while the interactions represent ingest ingest steps between those entities, managed by the Skywatch system.

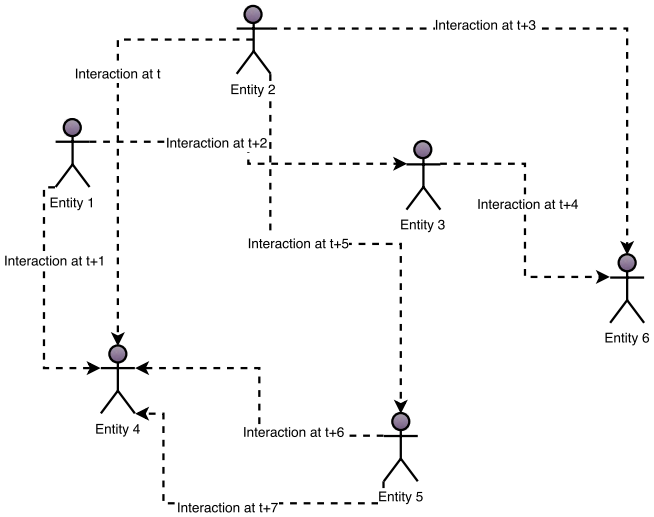


Figure 3.1: Architecture of the simulation framework

3.2 Design Decisions

The requirements of this solution mainly focus on two distinct topics:

1. Be deployable in an environment in accordance to MOG's machines specifications;
2. Be generic enough to be applicable to other scenarios, as an open-source analytics platform.

Touching the first item, the deployment environment is characterized by a single, efficient machine deployed on-site (*i.e.* at a client's location), and the system must be able to be upgraded and fixed remotely.

In what regards the second item, the main intents are to both contribute to the open-source community with an easy to use environment for analysis over data streams, as well as take advantage of the community-developed analytics procedures to potentiate the MOG's version of it.

There are already some solutions in the market targeting analytics on streaming environments. Although, they generally are to be deployed in-house, in a cluster of machines to be properly effective. Systems such as Apache Flink[1] and Apache Spark[2] have very low efficiency when installed in a single-node cluster.

On the other hand, Akka provides a collection of libraries that support the development of highly concurrent applications, with high performance both in multiple machine clusters and single-node ones. This platform is, then, developed on top of Akka toolkit, taking advantage of three of its modules:

- At the core of the Akka toolkit is Akka Actors, which provides strong guarantees when dealing with concurrency. These Actors exchange messages between themselves, and are able

[1] https://flink.apache.org/
[2] http://spark.apache.org/

to make local decisions and modify private state according to them [HBS73]. Essentially, they enable the distribution of both responsibilities and computation needs, enabling the development of systems according to agent-based software engineering practices, reviewed in Section 2.4;

- Akka Streams was developed to tackle the limitations that arise when Actors are deployed to deal with data streams, to enforce proper measures that guarantee stable streaming communication between Actors;
- Akka HTTP takes advantage of the other Akka libraries to provide support for HTTP communications, seamlessly integrating with the remainder Akka environment.

3.3 System's Architectural Approach

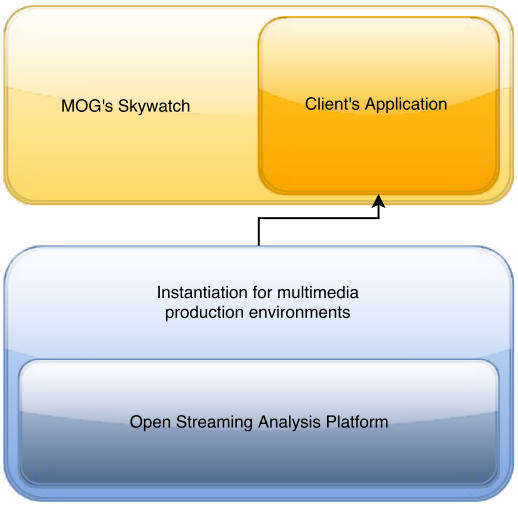


Figure 3.2: Architecture of the system for analysis and simulation of multimedia environments' systems

The system to be developed aggregates three main layers:

- the framework for simulation of behaviors in complex systems;
- the instantiation of the framework in the multimedia production environment;
- a client application, that connects to the previous layer through an API, and is integral part of the existing Skywatch solution.

The disposition of these layers is showcased in context in Figure 3.2. These components are further described In the following sub-sections.

3.3.1 Streaming Analytics Platform

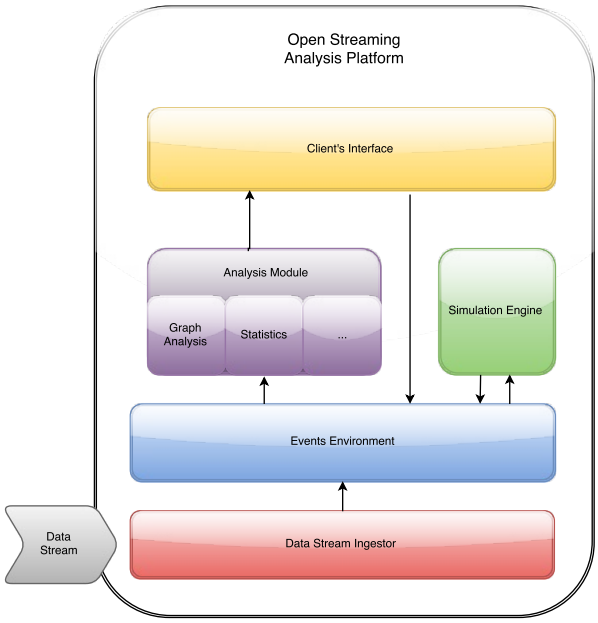


Figure 3.3: Architecture of the open streaming analytics platform

This underlying framework is the most complex component of the overall system. It divides itself into five different modules: data stream ingestor, events environment, simulation engine, analysis modules, and client's interface. The organization is shown in Figure 3.3.

The system is fed from the stream of information about events happening in the physical world. The **data stream ingestor** is responsible to parse this data according to a set of rules defined by the framework, and others defined by the framework's concrete instantiation. This information is then fed to the events environment.

The **events environment** aggregates a set of agents – whose representations are bound to the external entities – and manages their interactions. It uses the information provided by the data stream ingestor both to feed the simulation process of the agents, as well as to evaluate its own progress. Additionally, it provides support for changes in its running clock, for past or future points in time. At its core, the distributed framework discussed in Section 3.4.1 is responsible for the coordination of the actions performed.

The **simulation engine** coordinates the agents' behavioral cloning process. The agents' behavioral models evolve according to the data stream that arrives. These models essentially consist on a mapping from sensors criteria to actions, which can be nested within themselves. These actions may involve triggering sensors in other agents. It also supports requests to override agents' behavior with certain rules. In the context of this thesis, the objective is to test the utility of LSTM networks for this role, as will be described in Chapter 4.

The **analysis modules** continuously gathers information from simulation engine, providing descriptive and inferential knowledge about what happens. In it, there is an extensible environment of smaller modules, *e.g.* graph analysis and statistics.

Finally, the **client's interface** gathers processed data about the event's environment through the analysis module, and provides it to the outside world through an API.

3.3.2 Instantiation of the Framework for Multimedia Production Environments

This component binds the platform to the context of a given media production environment. Its responsibilities are:

- define a set of sensors and actions for agents;
- implement additional modules in the analysis module;
- bind the data stream processor to a given stream.

3.3.3 Client's Application

The client's application essentially retrieves information from the framework's interface, namely the one that originates in the analysis module. This information is then presented to the user, which in the context of media production environments is performed through a web UI[3]. It also sends requests to override agents' behavior and to simulate different clock times.

3.4 Technological Review

In this section it is discussed the utility of two tools, Scala and Python, and how they have aided the development of the solution.

Scala was used for the core of the solution, as seen throughout the examples that follows in this chapter. Python was useful for the task of prototyping and testing the simulation engine, which development is analyzed in Chapter 4. Next, the environments comprised by these two programming languages are reviewed, and how they aid this project is discussed.

3.4.1 Scala's Environment

Scala is a language that seamlessly combines functional and object-oriented programming [OSV08]. Together with keeping up-to-date with the research in programming languages, it achieves a very expressive, scalable and reliable environment on which to develop applications.

Due to its minimal syntax and powerful compiler with various implicit transformations, it is adapted to defining DSL, which enables the development of an intuitive framework. Its functional nature also eases the tasks of dealing with data flows concurrently, and enable the native support for parser generation.

[3] User Interface

One other advantage of working in Scala is that we do not have to compromise libraries availability, as it is able to run on top of the JVM[4]. Therefore, this project benefits from the following libraries:

- **Akka**, previously mentioned and described in Section 3.2;
- **WEKA**[5], a collection of machine learning algorithms for data mining tasks [HFH+09];
- **MOA**[6], an open source framework for data stream mining, with its roots in WEKA [BHKP10].

As an example of Scala's applicability to the problem at hands, in "Programming in Scala", by Odersky *et al.* [OSV08], it is explained the development of a simulation framework, based on the work of Bernardinello and Bianchi [BB12]. Here, the power of Scala's scheme for parallelization with Actors, and reactive-style of programming are enhanced, both being key aspects for analyzing complex CPS, as seen in Section 2.2.

Altogether, Scala encompasses an environment that has been growing both in size and adoption, as is the example of Spark's implementation migration to Scala [ZCF+10]. Due to these reasons, it is the tool of choice for implementing this project.

3.4.2 Python Machine Learning Stack

Python is a general-purpose scripting language that has been establishing itself as a language of choice in the data science and machine learning fields, together with R. Even though it is not as suited for large-scale development as other languages without special support, Python's environment is very diverse and useful for rapid development, encompassing libraries like NumPy[7], SciPy[8], Science-Kit Learn[9] and Pandas[10].

Due to its visibility and adoption, it is becoming also a relevant tool for deep learning development. Theano is one example of a platform for this field [BBB+10], being already widely used for these demanding tasks. More recently, TensorFlow was made available by Google [AAB+15], reinforcing the relevance of this field in the (near) future of machine learning. In this project, Tensorflow is used in developing the deep-learning simulation model.

3.5 Modules Breakdown

This section provides a more detailed look into each of the modules introduced previously.

Sections of the platform's code are provided both for clarification and exemplification, while the full documented source code is available at the GitHub project page [11].

[4] Java Virtual Machine
[5] Waikato Environment for Knowledge Analysis
[6] Massive Online Analysis
[7] http://www.numpy.org/
[8] https://www.scipy.org/
[9] http://scikit-learn.org/
[10] http://pandas.pydata.org/
[11] https://github.com/diogojapinto/complex-systems-simulation-framework

3.5.1 Data Stream Ingestor

The data stream ingestor module subdivides into two traits[12] that the developer should define as part of the framework's instantiation.

```
1 trait StreamSource {
2   def source: Source[String, NotUsed]
3   (...)
4 }
```

Listing 3.1: StreamSource trait

Listing 3.1 presents the first one, `StreamSource`, which requires from the developer the implementation of a stream source which outputs strings. The `source` attribute should be defined using the utilities already provided by the Akka Streams package. These are very complete, robust, and provide great customization options according to the nature of the source.

```
1
2 object SkywatchSource extends StreamSource {
3   (...)
4   val flowBatchDataSource =
5     flowBatchCollection
6       .find(BsonDocument.empty)
7
8   val flowCursorDataSource =
9     flowCappedCollection
10      .find(query, tailable = true)
11
12  override def source: Source[String, NotUsed] =
13    flowBatchDataSource
14      .concat(flowCursorDataSource)
15      .mergeMat(Source.empty)(Keep.right)
16 }
```

Listing 3.2: Skywatch Stream Source instantiation

In the case of the instantiation at MOG's environment, there is the need to firstly feed through the processing graph past data, and only after feed the data that arrives asynchronously. To accomplish that, `SkywatchSource` is defined according to Listing 3.2. Two individual sources are created: `flowBatchDataSource`, that feeds the past data from `Flow_Transformation` one item at a time, and `flowCursorDataSource`, which connects a cursor to the MongoDB capped collection, being notified each time a new item arrives. Finally, `source` – required by the `StreamSource` trait –

[12] Traits may be considered as common Java interfaces, with the addition that data and methods may be already implemented

is defined as the concatenation of these two instances on line . This method enables the remainder of the processing graph to be completely agnostic to the intricacies of the source definition.

```
1 abstract class AgentAction {
2   val sourceId: String // action's source
3   val targetId: String // action's target
4   val attributes: mutable.Map[String, DataType] // other attributes
5   (...)
6 }
```

Listing 3.3: AgentAction abstract class

```
1 trait Parser {
2   def parse(obj: String): AgentAction
3   (...)
4 }
```

Listing 3.4: Parser trait

The main data format that traverses the platform is `AgentAction`, whose definition is shown in Listing 3.3. It is intended to be inherited by concrete implementations from the developer. Given this definition, nodes of the interaction graph are simply modeled by the actions they perform throughout most of the platform, while the assembling of a concrete graph is left as a task for additional services (as the criteria for it might diverge according to the use-case).

The second component is the `Parser` trait, shown in Listing 3.4, where the developer defines a function, `parse`, that converts each incoming string into an `AgentAction` element.

```
1  class StreamIngestor(sourceEnv: StreamSource, parserEnv: Parser) {
2    (...)
3    val sourceGraph: Source[AgentAction, (NotUsed, NotUsed)] =
4          (...)
5          streamSource ~> parserFlow
6
7          SourceShape(parserFlow.out)
8      })
9    (...)
10 }
```

Listing 3.5: StreamIngestor class

Finally, these two components are combined in the `StreamIngestor` class, as seen in Listing 3.5. Essentially, the custom source's output is bound to the custom parser's input in line 5, and a new source is returned by exposing the custom parser's output in line 7.

3.5.2 Event's Environment

```
1  class SystemManager(val ingestor: StreamIngestor) extends ServicesProvider {
2    (...)
3    def runnableGraph =
4          (...)
5          val broadcaster = b.add(new Broadcast[AgentAction](analysisGraphComponents.
               size + 1, false))
6
7          src ~> broadcaster
8
9          for ((name, model) <- analysisGraphComponents) {
10           broadcaster ~> Sink.actorSubscriber(model).mapMaterializedValue{
11             actorRef => analysisDataModelActors.+=(name -> actorRef)}
12         }
13
14         broadcaster ~> Sink.empty
15
16         ClosedShape
17     })
18   (...)
19   def init(): Unit = {
20     serverThread.start
21     streamGraphThread.start
22   }
23 }
```

Listing 3.6: SystemManager class

The incoming `AgentAction` elements are managed by the `SystemManager` class, partially defined in Listing 3.6. This class takes the compound source from a `StreamIngestor` and connects it to the desired services. For that, a broadcaster object is used in line 5, whose output is then feed to each service module actor, in line 10.

The shape of the processing graph is finally closed by returning `ClosedShape` in line 16, and as such the graph is now ready to be run.

3.5.3 Analysis and Client's Interface

Both the analysis and the client's interface are encoded in the concept of Service. A Service is responsible for defining one or more pairs of `AnalysisDataModel` and `AnalysisApi`, as well as registering them in the `SystemManager`.

```
1  abstract class AnalysisDataModel extends ActorSubscriber {
2    (...)
3    def storeData(data: AgentAction): Unit
```

```
4
5   def processRequest(request: DataRequest): ProcessedData
6
7   def broadcastProcessedData(processedData: ProcessedData): Unit = {
8     subscribedActors.map(actorRef => actorRef ! processedData)
9   }
10  (...)
11  override def receive: Receive = {
12    case OnNext(data: AgentAction) =>
13      storeData(data)
14    case request: DataRequest =>
15      sender ! processRequest(request)
16    case _ =>
17      sender ! "Invalid request"
18  }
19 }
```

Listing 3.7: AnalysisDataModel abstract class

`AnalysisDataModel`, shown in Listing 3.7 is implemented as an Akka Actor, which subscribes certain `AgentAction` elements, processes and stores them (calling the user-defined `storeData`), and answers incoming requests both from `AnalysisApi` instances, or even other `AnalysisDataModel` (calling the user-defined `processRequest`). Some utility functions are also provided, like the `broadcastProcessedData`, which enables the developer to control when to feed-forward new data to streams requested by `AnalysisApi`.

```
1  abstract class AnalysisApi {
2
3    def getHandler(request: List[String]): String
4
5    def socketHandler(request: List[String]): Source[Message, Any]
6
7    def dashboardPath: String
8
9    def requestProcessedData(dataModelIdentifier: String, dataRequest: DataRequest):
         ProcessedData = {
10     val request = analysisDataModelActors(dataModelIdentifier) ? dataRequest
11     val result = Await.result(request, timeout.duration).asInstanceOf[ProcessedData
           ]
12
13     result
14   }
15
16   lazy val route =
17     pathPrefix(moduleName) {
18       path("get" / Segments) { requestSegments =>
19         get {
```

```
20            complete(getHandler(requestSegments))
21          }
22        } ~
23          path("socket" / Segments) { requestSegments =>
24            handleWebSocketMessages(
25              Flow[Message].merge(
26                socketHandler(requestSegments)
27              )
28            )
29          } ~
30          pathPrefix("dashboard") {
31            pathSingleSlash {
32              getFromFile(s"$dashboardPath/index.html")
33            } ~
34              getFromDirectory(s"$dashboardPath")
35          }
36      }
37  }
```

Listing 3.8: AnalysisApi abstract class

In order for the Services to be accessible to the outside, `AnalysisAPIs` should be defined. There are three access points: REST GET requests, WebSocket connections, and dashboards, and the developer defines handlers for the ones desired. Akka HTTP package enables great dynamism in the server routing mechanism, and integrates seamlessly with Akka Streams. One example of how these elements connect is seen when a new webSocket request arrives:

1. a bi-directional stream Akka Streams Flow is automatically opened for communication from and to the client;
2. a dedicated `ActorPublisher` from an `AnalysisDataModel` to stream data is requested according to the request parameters;
3. the `ActorPublisher` is merged into the Flow created at point 1., as seen in lines 25 and 26 of Listing 3.8.

An utility function, `requestProcessedData`, is also provided, to abstract the process of performing requests to `AnalysisDataModel`s.

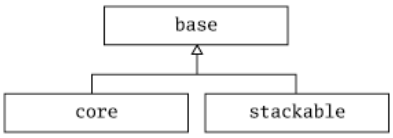


Figure 3.4: Diagram of the stackable traits pattern.

In order for the `SystemManager` to have access to services, while maximizing the compile-time guarantees, the stackable traits pattern was used. Essentially, it enables the definition of decorators

binded in compile time by both the core class and the stackable traits inheriting from a common superclass, as the diagram in Figure 3.4 shows.

```
object SkywatchMain extends App {

  val manager = new SystemManager(
                        new StreamIngestor(SkywatchSource, SkywatchParser)
                      ) with EchoService
                        with IngestAnalysisService
                        with ClusteringService
                        with AnomalyDetectionService

  manager.init()
}
```

Listing 3.9: Stackable trait pattern example in SkywatchMain class

```
abstract class ServicesProvider {
  protected val analysisGraphComponents = mutable.Buffer.empty[(String,
      AnalysisDataModelProps)]
  protected val analysisApis = mutable.Buffer.empty[AnalysisApi]
  implicit val analysisDataModelActors = mutable.Map.empty[String, ActorRef]

  def addAnalysisModule(name: String,
                        dataModel: AnalysisDataModelProps,
                        api: AnalysisApi): Unit = {

    analysisGraphComponents.+=:((name, dataModel))
    analysisApis.+=:(api)
  }
}
```

Listing 3.10: ServicesProvider abstract class

An example of this pattern is seen in Listing 3.9. As both the services and the `SystemManager` are subclasses of `ServicesProvider`, shown in Listing 3.10, the services use the method `addAnalysisModule` to register service modules, and the `SystemManager` is automatically able to access the buffers at start-up.

3.6 Example Service: Anomaly Detection

To close the chapter, in order to better understand the capabilities of the developed platform, an example of an anomaly detection service is shown in this section. For clarity, the focus will be on the transcription of the process into the platform, provided the needed mathematical libraries are already available.

For this task, there is both the need to extract the long-term trend, T, of the continuous data stream D, as well as the seasonality encoded in it, S.

In order to extract T, a rolling median is computed for each time-stamp, given the values for the immediate past across a given time duration. A rolling median is advantageous as it is less sensible to outliers than a rolling mean, while containing more information than the simple application of a linear regression.

In what regards seasonality, Fourier analysis enables the representation of the data stream in the frequency domain, where we are able to select the dominant frequencies according to each term's amplitude, and consider them as the seasonality patterns. Programmatically, Fast Fourier transform method enables just that [CCF+67], efficiently computing the Discrete Fourier transform, which is given by the following formula:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i 2\pi k \frac{n}{N}}, k = 0, \ldots, N-1$$

where N is the estimated number of periods which the sequence obeys to.

After obtaining T and S, the error of the model E is computed by combining the trend and seasonality, and subtracting that construct from the original data, according to the formula

$$E = D - (T + S)$$

```
1  object AnomaliesDetectionServiceModule {
2
3    val AnomaliesDetectionServiceIdentifier = "anomalies-detection"
4
5    case object AnomaliesDataRequest extends DataRequest
6
7    case object AnomaliesStreamRequest extends DataRequest
8
9    case class AnomaliesData(val data: mutable.Map[String, List[(DateTime, Float)]])
         extends ProcessedData {
10     override def toString: String = data.toJson
11   }
12
13   case class AnomaliesSocketSource(source: Source[Message, ActorRef]) extends
         ProcessedData
14 }
```

Listing 3.11: AnomaliesDetectionService companion class

Firstly, there is the need to define the messages and data types that will be used in the context of services. That is done in a companion object as shown in Listing 3.11, so that it is static and accessible by other services. Two types of request are defined: one for sporadic HTTP GET requests (`AnomaliesDataRequest`), and another for streaming WebSocket requests

(`AnomaliesStreamRequest`). Additionally, a data type is defined to store data already processed, in `AnomaliesData`, corresponding to a map of attribute keys to lists of time-indexed values.

```
trait AnomaliesDetectionServiceModule extends ServicesProvider {
  this: SystemManager =>

  implicit val moduleName = AnomaliesDetectionServiceIdentifier
  (...)
```

Listing 3.12: AnomaliesDetectionServiceModule trait class initial definition

As stated previously in this Section, services are defined as traits, in order to comply with the stackable trait pattern. Listing 3.12 presents that definition. One thing to notice is that the self-reference `this` is cast to `SystemManager` type, effectively implying that this trait is only bindable to `SystemManager`, while providing access to its constructs.

```
trait AnomaliesDetectionServiceModule extends ServicesProvider {
  (...)
  class AnomaliesDetectionDataModel(val stdTolerance: Float) extends
      AnalysisDataModel {
    (...)
    override def storeData(action: AgentAction): Unit = {
      for ((k, v) <- data.attributes) {
        // the following code is simplified for brevity purposes
        data = Some(data.append(v))
        trend = Some(rollingMedian(data))
        seasonality = Some(fft(data))
        error = Some(data - (trend + seasonality))
        errorStd = std(error)

        anomalies = error.filter{(time, err) =>
          abs(err) >= errorStd * stdTolerance
        }
      }

      broadcastProcessedData(anomalies)
    }

    override def processRequest(request: DataRequest): ProcessedData = request
        match {
      case AnomaliesDataRequest => anomalies
      case AnomaliesRequest =>
        val source = establishProcessedDataPublisher
        AnomaliesSocketSource(source)
    }
  }
  (...)
```

```
30 }
```

Listing 3.13: AnomaliesDetectionDataModel class

An `AnalysisDataModel` effectively works as a bridge between raw events data and knowledge-infused processed data. When new data arrives, anomalies are calculated as stated previously, as shown in Listing 3.13 from line 8 to 17. After that, updates are broadcasted to all subscribed WebSockets by calling `broadcastProcessedData` in line 19.

Lastly, requests are processed according `processRequest` function. The most up-to-date anomalies are forwarded in the case of a batch request, or a new Akka Streams source is built to forward future updates, in the case of a streaming request.

```
1  trait AnomaliesDetectionServiceModule extends ServicesProvider {
2    (...)
3    object AnomaliesDetectionApi extends AnalysisApi {
4
5      override def getHandler(request: List[String]): String = {
6        requestProcessedData(AnomaliesDetectionServiceIdentifier,
             AnomaliesDataRequest)
7          .toString
8      }
9
10     override def socketHandler(request: List[String]): Source[Message, Any] = {
11       val source = requestProcessedData(AnomaliesDetectionServiceIdentifier,
12                                         AnomaliesStreamRequest)
13         match {
14           case AnomaliesSocketSource(src) => src
15         }
16       source
17     }
18
19     override val dashboardPath = "resources/anomalies-dashboard"
20   }
21   (...)
22 }
```

Listing 3.14: AnomaliesDetectionApi object definition

After defining how data is processed and stored, an interface must be defined. In Listing 3.14 are defined two handlers for HTTP GET and WebSocket requests, `getHandler` and `socketHandler` respectively. Both take advantage of the method `requestProcessedData`, which performs all the logic of asking for data from the data model actors.

Additionally, a path for web-page files is defined, so that a GUI may be provided by the developer without the need for additional web-servers.

```
1 trait AnomaliesDetectionServiceModule extends ServicesProvider {
2   (...)
3   val dataModel = Props(new AnomaliesDetectionDataModel)
4   val api = AnomaliesDetectionApi
5
6   addAnalysisModule(moduleName, dataModel, api)
```

Listing 3.15: AnomaliesDetectionServiceModule trait class ending definition

Finally, the data model and API are binded and registered as an analysis module, as shown in Listing 3.15.

This service is extendable to simulate future points in time by combining the computed trend and the Fourier Transform into a forecast. This would showcase the expected values, translating into an expected error of zero – therefore, there is no anomaly detection in future points in time.

Chapter 4

Simulation Module with Deep Learning

This chapter addresses the development of the previously mentioned simulation module mentioned in Section 3.3.1. Due to the complexity of this subject, this module was developed decoupled from the remainder platform, to achieve fast prototyping and testing.

As discussed in Section 2.7.2, Deep Learning has been seeing countless applications, and multiple techniques have been developed to tackle problems of different natures. In this chapter, a methodology that takes advantage of these Deep Learning-related techniques is devised, in order to enable scalable on-demand simulation of complex systems.

This chapter starts with an overview of the methodology and objective of the simulation engine in Section 4.1, after which the testing dataset is reviewed in Section 4.2, and the the environment in which the development and deployment of the prototype is done is itemized in Section 4.3. Next, the architecture of the neural network model is presented in Section 4.4, the training procedure is reviewed in Section 4.5, and the model is evaluated in Section 4.6. Finally, the results obtained are discussed regarding the requirements and limitations of the model in Section 4.7.

4.1 Methodology Overview

Starting with the definition of interactions graph provided in Section 3.1, the modeling objective for the context of this thesis is formally defined as, given a set of nodes V, a sequence of directed edges E (each with source s and target t) up until time t, and the source node for the edge at time $t+1$, obtain a probability distribution over the possible target nodes in the form

$$P(t = v_j | Context, s = v_i)$$

. Intuitively, the aim is to predict the target of an interaction given a context.

This simplification enables diverse use cases for a system that complies to it. One example methodology is as follows:

1. Chose a desired node (or set of nodes), according to some criteria, to be the first interaction source;

2. Sample, according to the system's output distribution, a target for the interaction.

3. Use the target as the next source (or apply an alternative policy, *e.g.* the last n sources have probability $1/n$ of being selected);

4. Repeat the process, until a given criteria is met.

5. Analyze the evolution of the system.

To achieve such system, inspiration is taken from state of the art methodologies used in language modeling.

Language modeling seeks to formulate a probability distribution over sequences of words. Zaremba *et al.* feed to a deep learning architecture sequences of words in a large corpus, and for each input word, the models seeks to predict the word that follows in the text [ZSV14]. This essentially achieves the premise of language modeling, while obtaining great results.

One thing to notice is that the objective of language modeling is similar to the one this thesis proposed in the beginning of this section. Essentially, if we consider a graph in which each node is a word, and each edge has source in a word that precedes the target one, this problem is translated into the modeling objective of this thesis.

The two following sections review the two main aspects of language modeling with deep learning: words embeddings, and LSTMs, and seek to adapt them to the context of interactions graphs.

4.1.1 Nodes Embeddings

In some domains, where the data seems to be represented by discrete atomic symbols, the traditional way to encode it is through a sparse one-hot encoding representation (similar to using discrete unique identifiers). This is the case of traditional Natural Language Processing techniques: a large, sparse vector is used to represent a single word, where each position of the vector represents each possible word in the vocabulary. Nonetheless, such representations do not encode in themselves information regarding relationships that may exist between the individual symbols, and the systems that use them either have to infer that information, or do not use it at all. In contrast to NLP, in Computer Vision the information is naturally encoded as a tensor of pixel intensities (in the case of an image), and the information at each pixel is relatable to others by, *e.g.*, position and color intensities.

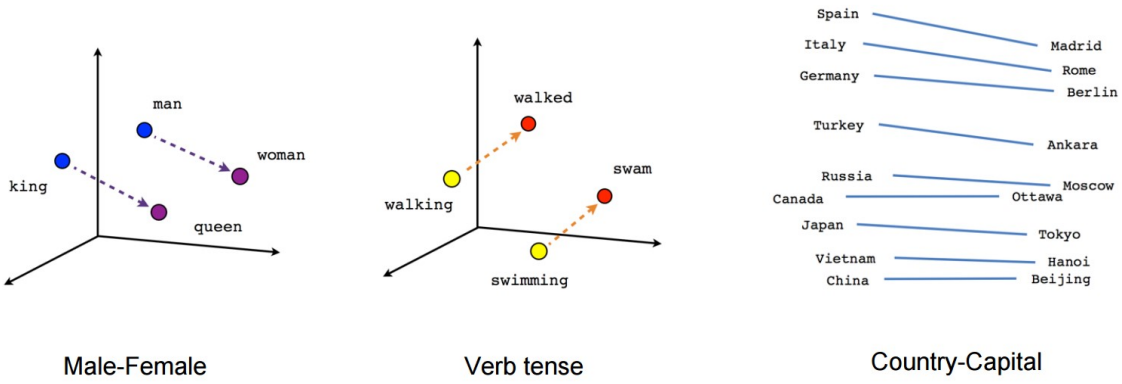


Figure 4.1: Visualization of word2vec embeddings projection. Each diagram showcases different semantic relationships in the words embeddings space: gender (male to female) on the left, verb tenses (gerund to past tense); and country to capital in the right. (Source: https://www.tensorflow.org/versions/master/tutorials/word2vec/index.html)

Seeking an unsupervised way to infer a dense representation for words, which encodes the relationships between them, Mikolov *et al.* developed word2vec [MCCD13], a system that learns embeddings for words (*i.e.* vector representations in an k-dimensional space), managing to keep representations of same-context words closer, as well as keeping distances in pairs of words that share a semantic relation, as shown in Figure 4.1.

Inspired by this methodology, this thesis aims to extend it to another context: the one of interaction graphs. To clarify, an useful analogy is that the words in word2vec represent the nodes of the interaction graph. Such translation of the method is useful in the context for two reasons:

- It enable the Deep Learning model to work on top of a higher abstraction layer, where (at least part) of the relationships between graph nodes are already encoded in the input;

- Such latent representations encode valuable information that may be used in experiments and analysis over the interaction graph, fully independent of the original system intents.

4.1.2 Long Short-Term Memory Networks

Reviewing Section 2.7.2, Recurrent Neural Networks are a collection of neural networks in which layers of neurons include, beyond a feed-forward and a back-propagation interface, an hidden mechanism that enables a re-flux of information. That re-flux has the consequence that, for a given task, an output is not only dependent on the input, but also on the internal state of the neural network. Effectively, that confers a kind of memory, property which is highly desirable to work on a broad kind of problems, where the input and/or output desired is a sequence with arbitrary dimension. The possible configurations of RNNs usage is shown in Figure 4.2.

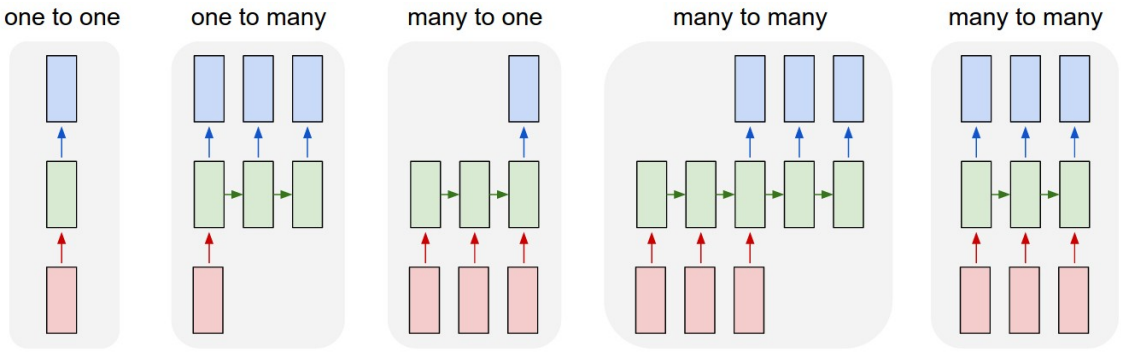


Figure 4.2: Possibilities of RNNs to work over sequences. The red boxes represent input, the green boxes represent hidden states, and the blue boxes represent the output. The variations all default to the last "many-to-many" example, only changing the when we decide to provide input or measure the output, over an iteration of step-size k. (Source: http://karpathy.github.io/2015/05/21/rnn-effectiveness/)

Despite the theoretical possibility for a RNN to influence output with information seen in a distant past, in practice the signals vanished rapidly across iterations. LSTM cells were developed, in which the hidden state already present in RNNs is derived at each iteration from a cell state, that intuitively works like a long memory. This cell state changes are then strictly controlled by a collection of gates.

Next follows an explanation of the computations happening inside an LSTM cell, at a given time t (at $t = 0$, both hidden state and cell state are initialized at 0, everything else holds true).

- At the arrival of x_t, the input at time t), it is concatenated with the previous hidden state, h_{t-1}, into $[h_{t-1}, x_t]$. This works as a compound input for the remainder operations.

  The forget gate, f_t, is responsible for selecting which activations in the cell state are dropped for the next iteration. It is computed according to the formula:

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

  where W_f and b_f are the weights and biases for f, respectively, and $\sigma()$ is the application of the logistic function

  $$S(t) = \frac{1}{1 + e^{-t}}$$

  This function is used because the obtained values are skewed to near 0 or 1, enabling the output to work like a filter by multiplying it with some value.

- Next focus is on deriving new information to store in the cell state C_t. For that, from $[h_{t-1}, x_t]$ are both derived an ignore filter, i_t, and the candidate information, $\tilde{C}_t$, according to the formulas

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

  $$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

where W_i and b_i are the weights and biases for i, W_C and b_C are the weights and biases for $\tilde{C}$, and $tanh()$ is the hyperbolic tangent function application (which pushes the values to be near -1 or 1).

- Now, the new cell state, C_t, is derived by firstly "forgetting" activations from the previous cell state according to the forget gate f_t, and then adding the candidate information $\tilde{C}_t$, with some activations ignored according to the ignore filter i_t. This is summarized by the following formula:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- Finally the output is generated. This output is derived from the cell state C_t, and filtered by an output filter, o_t. The output, h_t, is also feed to the next iteration as the hidden state. The formulas to compute them are:

$$o_t = \sigma(W_o \cdot [ht-1, x_t] + b_o)$$

$$h_t = o_t * tanh(C_t)$$

where W_o and b_o are the weights and biases for o.

4.2 Dataset

At the time this thesis was developed, a well-structured and properly diverse dataset in the context of multimedia production environments was not available. Therefore, there was the need to select one dataset that is sufficiently similar the the desired one. The conditions for that were:

1. being constituted by a series of interactions from well-defined source and target entities;
2. each interaction should be tagged with a time-stamp, enabling their ordering;
3. be diverse and vast both in the number of source and target entities, as well as in the number of interactions which involved those same entities.

The dataset selected that fulfills those constraints was one that contains the activity in Twitter during the surrounding days of the Higgs boson discovery. This dataset was gathered by Domenico *et al.* to study the evolution of graph processes regarding scientific rumors [DDLMM13], and is publicly available in the Stanford Network Analysis Project webpage [1].

Number of interactions (edges)	563 069
Number of users involved (nodes)	304 691

Table 4.1: Dataset dimensions

[1] https://snap.stanford.edu/data/higgs-twitter.html

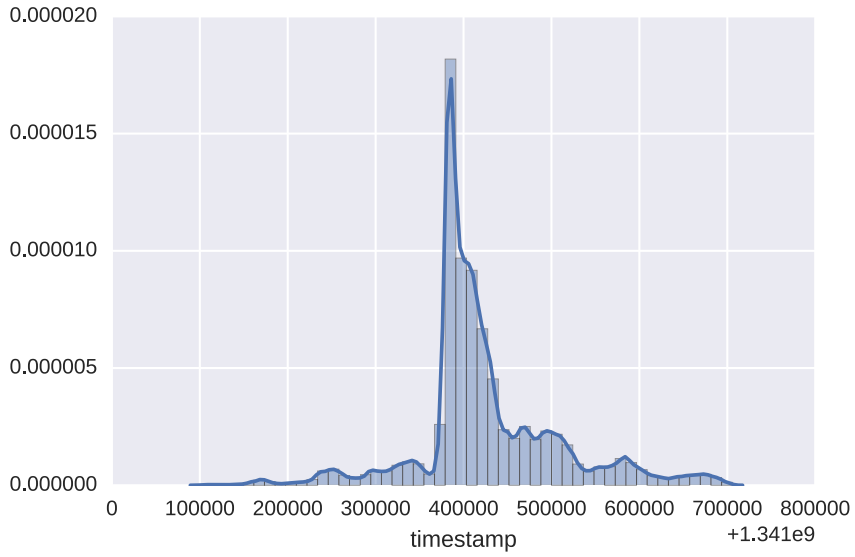


Figure 4.3: Distribution of interactions in the Higgs Twitter dataset through time. The spike in density of interactions correspond to the moment of discovery of the Higgs boson.

Table 4.1 provides an overview of the dimension of the dataset, and the distribution of the interactions through time is visualized in Figure 4.3 (additional analysis are provided in Appendix A).

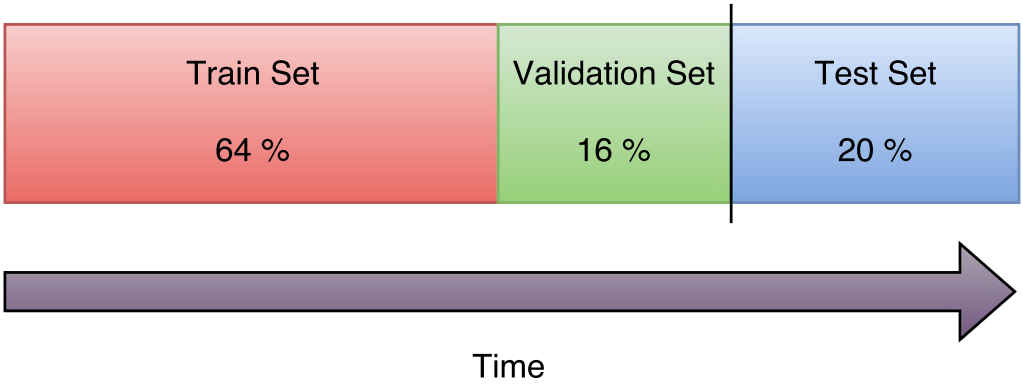


Figure 4.4: Diagram of dataset division into training, validation and test sets. The dataset is ordered in time and firstly divided into two portions (represented by the black vertical line): one with 80% of the data, and other with 20%. The 64% in the train set represent 80% of the first 80%, and the 16% of the validation set represent the last 16% of the first 80%.

The dataset was divided into three sections, each section containing a consecutive portion of the dataset ordered in the time dimension, as shown in Figure 4.4. The sections are as follows:

- the last 20% is the test set
- the first 80% are further divided:

- the first 80% of that portion is the training set
- the last 20% of that portion is the validation set

4.3 Environment

The software was tested in a notebook with an Intel®Core™i7-2670QM CPU at 2.20GHz, 8 GB of RAM, and no GPU. It is developed in Python 3.4, using TensorFlow [AAB+15] to build and run the neural network.

4.4 Neural Network Architecture

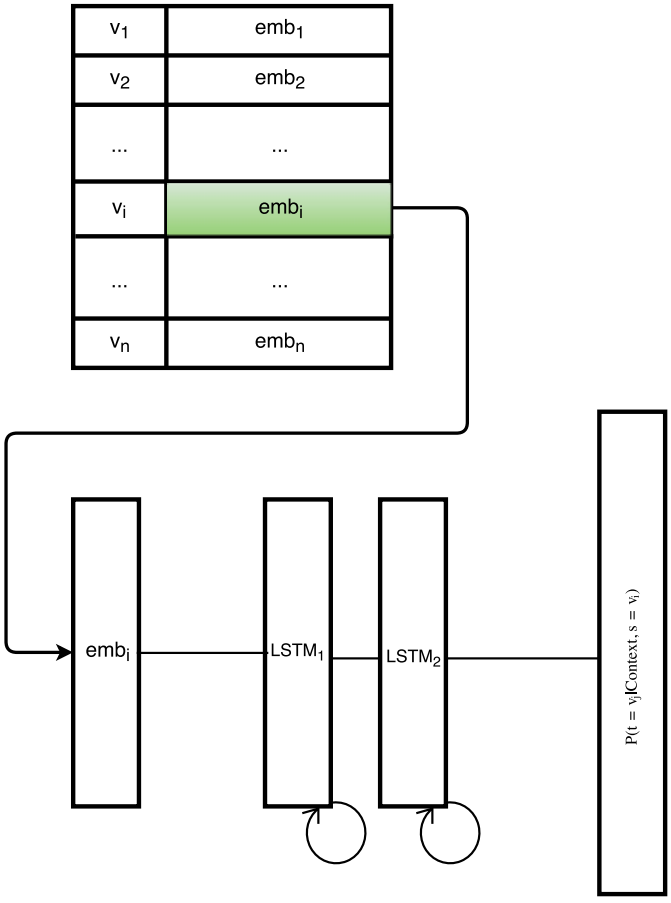


Figure 4.5: Schematics of the neural network model. The forward process happens as follows: (1) the embedding (emb_i) for the source node (v_i) is fetched from the lookup table; (2) that embedding is fed into $LSTM_1$, after which the following k embeddings are also fed, where k is the number of unrolled steps; (3) that sequence of k embeddings is forwarded through the unrolled $LSTM_1$ and $LSTM_2$; (3) for each k step, a distribution for the predicted target node (v_j) is computed.

The network's input is an embedding layer. Essentially, it maps for each unique interaction source node an unique embedding in a fixed-size space.

This embedding is feed-forward to a two-layers deep LSTM, each with 200 units, with the gradients truncated to 20 unrolled steps (this essentially prevents the network from back-propagating the gradients to the beginning of the sequence in each train step).

The last layer is a softmax linear classifier, which maps the 200-dimensional output of the previous LSTM layer to an n-dimensional vector, being n the number of unique nodes in the graph, which corresponds to the predicted probability distribution of the interaction targets.

Appendix B contains the computational graph of the operations performed over the data.

4.5 Training Procedure

The network was trained using mini-bat gradient descent, with a mini-batch size of 20, during 10 epochs (10 full iterations over the whole dataset). The model was evaluated in each epoch in the training set and in the validation set. After the tenth epoch, is was evaluated in the test set.

This process took 27 hours in the environment mentioned in Section 4.3. This training is then very slow in the environment described in Section 4.3, which justifies the limitation on the number of epochs.

4.6 Evaluation

The network is evaluated according to the perplexity measure per event, which corresponds to the exponentiation of the cross-entropy between two probability distributions. Intuitively, a perplexity of x means that the obtained model is as confused on the data as if it had to predict each label by choosing independently from an uniform distribution of size x.

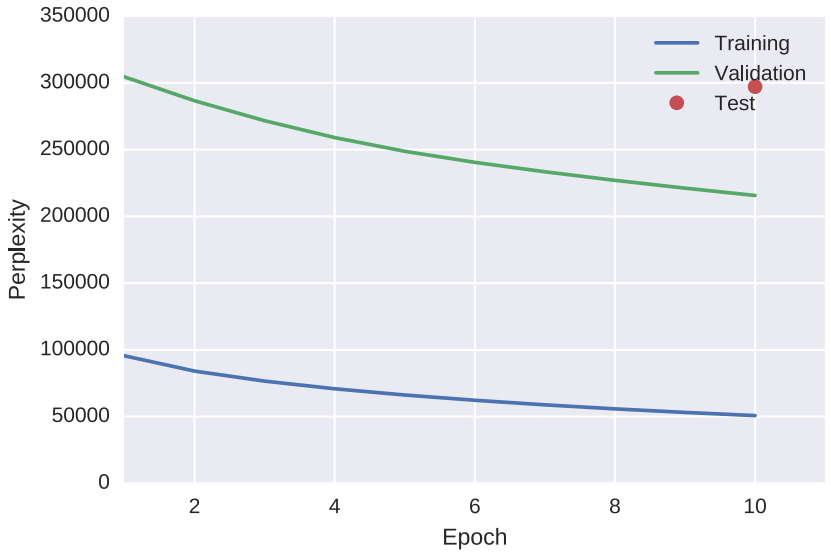


Figure 4.6: Evolution of perplexity across epochs. Each data point was obtained after processing a full epoch of training, besides the test perplexity, which was only measured after the all the training epochs.

Train Perplexity	**Validation Perplexity**	**Test Perplexity**
50717.969	215642.580	297165.892

Table 4.2: Perplexity on training, validation and test set after training in 10 epochs on the training set.

From the evolution of training, shown in Figure 4.6, and due to how the three sets of data were extracted, we are able to infer some knowledge about the model.

Firstly, as the validation set is temporally subsequent to the training set, and as the validation perplexity decreases as the training perplexity decreases, training the model on interactions up until time t, effectively provides it with some predictive power for interactions that happen on time $t+n, n > 0$.

Secondly, the fact that the test perplexity (shown in Table 4.2) is near the number of nodes in the interaction graph, and that the test set is temporally away from the training set (with the validation set in between), means that the model's predictive power is only significant in a restricted time window.

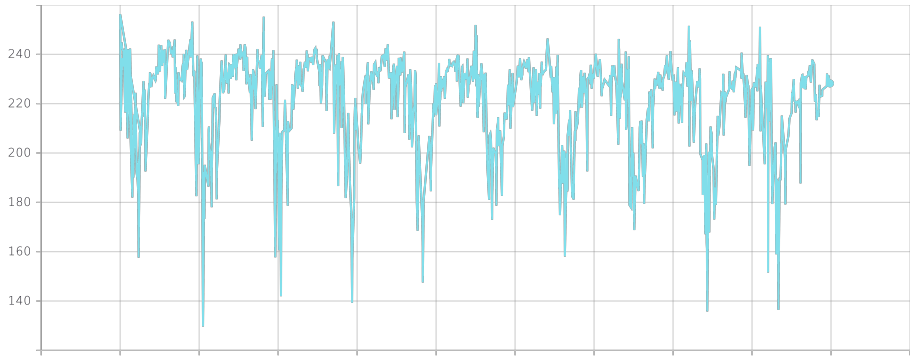


Figure 4.7: Perplexity during training across mini-batches. Each data-point in the line corresponds to the perplexity measured in a mini-batch of the data after training the model with that same mini-batch.

The variation of perplexity across subsequent mini-batches during the training process, shown in Figure 4.7, indicates that, despite a progressive global decrease in perplexity, the model struggles to capture the variation of the dataset itself.

One additional insight might be taken, due to the shape of the plot. There are 10 evident cycles, each starting and ending with a period of greater perplexity, and with a period of low perplexity in between. This central period corresponds to the period in the interactions distribution for which the density is greater. This translates into existing a greater predictability in the interactions made during this time, as the model is better able to capture them.

4.7 Discussion

The results obtained are positive, as the model is able to effectively learn from the data distribution, but they are not very significant. The difference between the training and testing perplexity hints that there are two main hypothesis for that to be happening:

- The dataset, besides being relatively small, portrays and centers around a single external major event (facts shown by the EDA attached in Appendix A, which by itself weakens the capacity of the NN to extract patterns;
- The architecture of the network is relatively small and not properly optimized (mainly due limitations of the environment where it was deployed), suffering from high bias.
- Alongside the number of epochs, the small number of unrolled steps processed in each mini-batch greatly limits the ability of the algorithm to capture long-term patterns in the data.

Suggestions on how to improve these results are detailed in Section 5.2.2.

Besides the advantages that Deep Learning related techniques bring, there are also a couple of disadvantages, evidenced by the experiment performed, which follows:

- The high computational power needs makes this technique prohibitive in certain scenarios;
- There are no well-grounded end-to-end architecture development procedures;
- Despite relying on the same fundamental building blocks, the deployment of a solution in different scenarios almost always needs individual configuration to be effective,

Altogether, it is expected that, by following the analysis done in this chapter, and iterating over the built model, it is possible to achieve substantial improvements, reaching a model that is deployable alongside the analytics platform described in Chapter 3.

Chapter 5

Conclusions and Future Work

This thesis presented and implemented a proposal for an open streaming analysis platform, targeting complex systems. This aim led to further research the usage of Deep Learning as an integrating part of the platform, in what regards the simulation component.

Inspired by the problems media production environments face, the system devised aims to support not only that reality, but also extend to other fields. With the advent of the Internet of Things and growing scale of web-based services, complex interactions have to be properly analyzed to extract concrete knowledge, before any real decision takes place.

With the increasing data dimensions and throughput, such systems that are able to properly handle streams of data are becoming the key to unlock many of the Big Data promises.

These are difficult problem to tackle, and there are no definitive solutions. By combining the methodologies discussed in this thesis it is expected to achieve a first iteration of a possible solution. It is expected that this system will improve over the very own insight it will provide, in an iterative manner.

5.1 Discussion

The development of the platform in the context of this thesis mostly focused on providing an interface such that the efforts of developers that use it is well spent, both in the development of data analysis processes and the user interface. Essentially, it relieves the burden of dealing with many problems that arise from the nature of data streams, like parallelism needs and buffer overflows.

Although the support for it was added, the compositionality of independent analysis processes was not fully explored. Additionally, the framework does not properly incentivize the development of generic processes, that can be shared and explored in different domains.

As pointed in Section 4.7, the results obtained by the simulation are positive, but not very significant. Nonetheless, possibilities arise by the novel way of thinking about graphs that are

defined through the time dimension, as well as by identifying nodes in graphs by interfaces that encode some meaning about how each relates to the other.

5.2 Future Work

The work developed in this thesis was a research attempt that serves as a starting point for additional contributions. Value may be added both by adding features, or by researching over alternative methods, namely for the simulation module.

Even though the simulation module is intended to be an integrating part of the platform as a whole, its nature and complexity dictates that it should be evolved as a separate entity. Future work is presented next separately for the platform and for the simulation module:

5.2.1 Open Stream Analysis Platform

- Include some `StreamSource` definitions for common use cases, *e.g.* connection to logging files or databases. These would further fasten the adoption of the platform, by covering from the start a large percentage of use cases. Additionally, the `StreamSource` implementation is still the only one that requires the developer to interact with Akka Streams library, and distancing developers from it would lower the learning curve;

- Provide Parsers for common data formats, *e.g.* XML and JSON. This would enforce some assumptions in how data is transformed, that would need to be fully transmitted to the developer;

- Enable the introduction of novel input/output concepts into the interactions in running-time. One example of where this is relevant is in the simulation module, which right now is only able to take into consideration the users that exist at start-up. One method to tackle this is to keep multiple models running, which start at different times, and attribute to each model a score proportional to its performance according to a given metric;

- Support compositional data and analysis visualizations, much like what is done with the analysis processes. This would complement the functionality and scope of the platform.

5.2.2 Simulation Module

- Improve the network architecture through cross-validation with multiple model instances. This would require an upgrade on the running environment in order to be feasible;

- Test on other datasets (namely larger ones). Twitter Higgs dataset comprises just a few days, and the data is deeply influenced by an external event, making it difficult for the model to generalize;

- Study the usage of deep learning architectures of other natures, like Bi-directional Recurrent Neural Networks [SP97] and Neural Programmer-Interpreters [RdF15]. These recent innovations in the field hold much potential, and open opportunities for further research;
- Support (in training and prediction) categorization of interactions. This would translate into additional features for each interaction, that need to be treated differently from the source and target fields, as they would not be considered in the embeddings;
- Design a system that enables model's architectural evolution through time. This may be tackled by maintaining a set of models, and from them select the one with the best fit in the most recent data.

Despite current limitations, the methodology used opened opportunities to tackle complex systems in new ways. Further research may be done by:

- Design other models to fill the role of the simulation engine. One option would be the VFDT mentioned in Section 2.7.1, which would enable an evaluation of the value provided by the inferred rules, in comparison with the lack of those rules in the deep learning approach;
- Devise experiments to check how classical graph analysis methods may be approximated or improved by taking advantage of the obtained nodes embeddings. These embeddings encode relationships between nodes, and one hypothesis this thesis introduces is that may be similar analysis for graphs to the semantics analysis done in words embeddings.

References

[AAB+15] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Man, Rajat Monga, Sherry Moore, Derek Murray, Jon Shlens, Benoit Steiner, Ilya Sutskever, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Oriol Vinyals, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow : Large-Scale Machine Learning on Heterogeneous Distributed Systems. 2015.

[Ass10] Modelica Association. Modelica {®} - A Unified Object-Oriented Language for Physical Systems Modeling Language Specification. *Interface*, 5(6):250, 2010.

[AWC+03] Charu C. Aggarwal, T. J. Watson, Resch Ctr, Jiawei Han, Jianyong Wang, and Philip S. Yu. A Framework for Clustering Evolving Data Streams. *Proceedings of the 29th international conference on Very large data bases*, pages 81–92, 2003.

[BB12] Luca Bernardinello and Francesco Adalberto Bianchi. A concurrent simulator for petri nets based on the paradigm of actors of Hewitt. *CEUR Workshop Proceedings*, 851:217–221, 2012.

[BBB+10] James Bergstra, Olivier Breuleux, Frederic Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math compiler in Python. *Proceedings of the Python for Scientific Computing Conference (SciPy)*, (Scipy):1–7, 2010.

[BHKP10] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. MOA: Massive Online Analysis. *The Journal of Machine Learning Research*, 11:1601–1604, 2010.

[Bon02] Eric Bonabeau. Agent-based modeling: methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, 99(suppl. 3):7280–7287, 2002.

[BP98] Sergey Brin and Lawrence Page. The Anatomy of a Search Engine, 1998.

[BPG+04] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[CCF+67] W.T. Cochran, J.W. Cooley, D.L. Favin, H.D. Helms, R.a. Kaenel, W.W. Lang, Jr. Maling, G.C., D.E. Nelson, C.M. Rader, and P.D. Welch. What is the fast Fourier transform? *Proceedings of the IEEE*, 55(10):1664–1674, 1967.

[CMO14] Michele Colledanchise, Alejandro Marzinotto, and Petter Ogren. Performance analysis of stochastic behavior trees. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 3265–3272, 2014.

[CO08] António Castro and Eugénio Oliveira. The rationale behind the development of an airline operations control centre using Gaia-based methodology. *International Journal of Agent-Oriented Software Engineering*, 2(3):350, 2008.

[CRO16] Antonio J M Castro, Ana Paula Rocha, and Eugenio Oliveira. Towards an Organization of Computers for Managing Airline Operations. In *CATA- 31st International Conference on Computers and Their Applications, Las Vegas, USA, April 4-6*, Las Vegas, USA, 2016.

[DBS06] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39, 2006.

[DDLMM13] M De Domenico, A Lima, P Mougel, and M Musolesi. The anatomy of a scientific rumor. *Scientific reports*, 3:2980, 2013.

[DH00] Pedro Domingos and Geoff Hulten. Mining High-Speed Data Streams. *Proceedings of The Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80, 2000.

[DOM15] DOMO. Data Never Sleeps 3.0, 2015.

[DSM15] Berkeley J Dietvorst, Joseph P Simmons, and Cade Massey. Algorithm aversion: People erroneously avoid algorithms after seeing them err. *Journal of Experimental Psychology: General*, 144(1):114–126, 2015.

[Fis87] Douglas H. Fisher. Knowledge Acquisition Via Incremental Conceptual Clustering. *Machine Learning*, 2(2):139–172, 1987.

[GDC10] Derek Greene, Dónal Doyle, and Pádraig Cunningham. Tracking the evolution of communities in dynamic social networks. In *Proceedings - 2010 International Conference on Advances in Social Network Analysis and Mining, ASONAM 2010*, pages 176–183, 2010.

[GHJV95] Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*, volume 206. 1995.

[Gow14] Stephen Gower. Netflix Prize and SVD. pages 1–10, 2014.

[GŽB+14] João Gama, Indre Žliobaite, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *{...} Computing Surveys ({...}*, 46(4):1–37, 2014.

[HBS73] Carl Hewitt, Peter Bishop, and Richard Steiger. A Universal Modular ACTOR Formalism for Artificial Intelligence. *Ijcai*, pages 235–245, 1973.

[HFH+09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The WEKA Data Mining Software: An Update. *ACM SIGKDD Explorations*, 11(1):10–18, 2009.

REFERENCES

[HHSS97] Sepp Hochreiter, Sepp Hochreiter, Jürgen Schmidhuber, and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[HSD01] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01*, 18:97–106, 2001.

[Hug13] Jérôme Hugues. AADLib, A Library of Reusable AADL Models. 2013(September), 2013.

[Isl05] Damian Isla. Handling Complexity in the Halo 2 AI. *Game Developers Conference*, page 12, 2005.

[Jif13] He Jifeng. A Clock-Based Framework for Construction of Hybrid Systems. pages 32–51, 2013.

[KPS11] Gerald Kerth, Nicolas Perony, and Frank Schweitzer. Bats are able to maintain long-term social relationships despite the high fission-fusion dynamics of their groups. *Proceedings. Biological sciences / The Royal Society*, 278(1719):2761–2767, 2011.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2323, 1998.

[LBE15] Zachary Lipton, John Berkowitz, and Charles Elkan. A Critical Review of Recurrent Neural Networks for Sequence Learning. *arXiv preprint*, pages 1–35, 2015.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[Lee08] Edward A. Lee. Cyber Physical Systems: Design Challenges. page 8, 2008.

[Lim09] Chong Lim. An AI Player for DEFCON: an Evolutionary Approach Using Behaviour Trees. *Final Year Individual Project Final Report*, page 109, 2009.

[LST15] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tnenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[Mac67] J B MacQueen. Kmeans Some Methods for classification and Analysis of Multivariate Observations. *5th Berkeley Symposium on Mathematical Statistics and Probability 1967*, 1(233):281–297, 1967.

[MCB+15] James Manyika, Michael Chui, Peter Bisson, Jonathan Woetzel, Richard Dobbs, Jacques Bughin, and Dan Aharon. The Internet of Things: Mapping the value beyond the hype. *McKinsey Global Institute*, (June):144, 2015.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. *Nips*, pages 1–9, 2013.

[MF14] Jorg P. Muller and Klaus Fischer. Application impact of multi-agent systems and technologies: A survey. *Agent-Oriented Software Engineering: Reflections on Architectures, Methodologies, Languages, and Frameworks*, 9783642544:27–53, 2014.

[MIK+04] Ron Milo, Shalev Itzkovitz, Nadav Kashtan, Reuven Levitt, Shai Shen-Orr, Inbal Ayzenshtat, Michal Sheffer, and Uri Alon. Superfamilies of evolved and designed networks. *Science (New York, NY)*, 303(5663):1538–1542, 2004.

[MSOIK02] R Milo, S Shen-Orr, S Itzkovitz, and N Kashtan. Network Motif: Simple Building Blocks of Complex Networks. *Science*, 824(2002):298., 2002.

[MW13] Nathan Marz and James Warren. Big Data - Principles and best practices of scalable realtime data systems. . . . . *Harvard Bus Rev*, 37:1 – 303, 2013.

[Ogr12] Petter Ogren. Increasing Modularity of UAV Control Systems using Computer Game Behavior Trees. *AIAA Guidance, Navigation, and Control Conference*, (August):1–8, 2012.

[Omg98] Omg. Unified Modeling Language (UML). *InformatikSpektrum*, 21(2):89–90, 1998.

[OSV08] Martin Odersky, Lex Spoon, and Bill Venners. *Programming in Scala*. 2 edition, 2008.

[Par13] Shameem Ahamed Puthiya Parambath. Matrix Factorization Methods for Recommender Systems. 2013.

[PBV07] Gergely Palla, Albert-László Barabási, and Tamás Vicsek. Quantifying social group evolution. *Nature*, 446(7136):664–667, 2007.

[Qui86] J. R. Quinlan. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, 1986.

[Qui92] J Ross Quinlan. *C4.5: Programs for Machine Learning*, volume 1. 1992.

[RdF15] Scott Reed and Nando de Freitas. Neural Programmer-Interpreters. pages 1–12, 2015.

[RS10] Pedro Ribeiro and Fernando Silva. G-tries: an efficient data structure for discovering network motifs. *Proceedings of the 2010 ACM Symposium on . . .*, pages 1559–1566, 2010.

[Sch09] Wladimir Schamai. Modelica Modeling Language (ModelicaML): A UML Profile for Modelica. *Last Accessed*, pages 1–49, 2009.

[SGLW08] Lui Sha, Sathish Gopalakrishnan, Xue Liu, and Qixin Wang. Cyber-Physical Systems: A New Frontier. *2008 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (sutc 2008)*, pages 1–9, 2008.

[SHG+15] D Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, and Dan Dennison. Hidden Technical Debt in Machine Learning Systems. *Nips*, pages 2494–2502, 2015.

[SHM+16] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

REFERENCES

[Shn96] Ben Shneiderman. The Eyes Have It:A Task by Data Type Taxonomy for Information Visualizations. *Proceedings 1996 IEEE Symposium on Visual Languages*, pages 336–343, 1996.

[SMG14] Didier Sornette, Thomas Maillart, and Giacomo Ghezzi. How much is the whole really more than the sum of its parts? 1+1 = 2.5: Superlinear productivity in collective group actions. *PLoS ONE*, 9(8):1–29, 2014.

[Sor07] D Sornette. Probability Distributions in Complex Systems. *Arxiv preprint arXiv07072194*, physics.da:27, 2007.

[Sor08] D. Sornette. Nurturing breakthroughs: Lessons from complexity theory. *Journal of Economic Interaction and Coordination*, 3(2):165–181, 2008.

[SP97] M. Schuster and K. K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[SZR11] Anton Schäfer, Hans-georg Zimmermann, and Martin Riedmiller. Reinforcement Learning with Recurrent Neural Networks. *Thesis*, 9(3):410–420, 2011.

[TFF+15] John Tromp, Gunnar Farnebäck, Gunnar Farneback, Gunnar Farnebäck, and Gunnar Farneback. Combinatorics of Go. *Computers and Games: 5th International Conference*, pages 84–99, 2015.

[THBS15] P. Taylor, J. N. Hobbs, J. Burroni, and H. T. Siegelmann. The global landscape of cognition: hierarchical aggregation as an organizational principle of human cortical networks and functions. *Scientific Reports*, 5(November):18112, 2015.

[The13] The Wall Street Journal. Annual U.S. Cybercrime Costs Estimated at $100 Billion, 2013.

[WJK00] Michael Wooldridge, Nicholas R. Jennings, and David Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[ZCF+10] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark : Cluster Computing with Working Sets. *HotCloud'10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, page 10, 2010.

[Zha14] Lichen Zhang. A Framework to Model Big Data Driven Complex Cyber Physical Control Systems. *Automation, International Conferenceon Science, Computer Guangzhou, Technology*, pages 12–13, 2014.

[ZSV14] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent Neural Network Regularization. *Icrl*, (2013):1–8, 2014.

REFERENCES

Appendix A

Exploratory Data Analysis and Simulation Model Generation

This appendix presents the source code and run of the simulation model, using a Jupyter notebook.

Exploratory data analysis

Import the needed libraries

- **numpy**: widely used scientific computing package
- **pandas**: library to deal with data frames
- **datetime**: provides tools to deal with dates and timestamps
- **seaborn**: visualization library on top of matplotlib

In [1]:
```
import numpy as np
import pandas as pd
import datetime
import seaborn as sns
import time
import os
from operator import itemgetter

% matplotlib inline
```

Load the dataset

In [2]:
```
higgs_df = pd.read_csv('higgs-activity_time.txt',
                       delim_whitespace=True,
                       header=None,
                       names=['user_a',
                              'user_b',
                              'timestamp',
                              'interaction'])
```

Create column with datetime derived from the timestamp:

In [3]:
```
higgs_df['datetime'] = pd.to_datetime(higgs_df['timestamp'], unit='s')
```

In [4]:
```
higgs_df.shape
```

Out[4]:
```
(563069, 5)
```

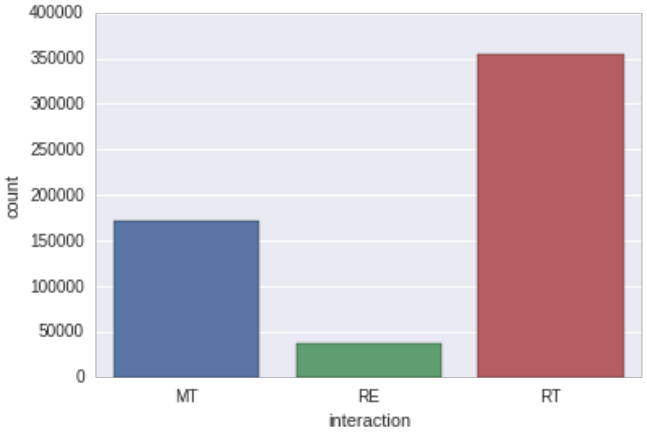
So we have a total of 563069 interactions. Lets see how this count is distributed by interaction type:

Note:

- **MT**: Mention
- **RE**: Reply
- **RT**: Retweet (forward)

```
In [5]: sns.countplot(data=higgs_df, x='interaction')
```

```
Out[5]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa6f5507a90>
```



Visualize the distribution of tweets through time

```
In [6]: timestamp_min = higgs_df['datetime'].min()
        timestamp_min
```

```
Out[6]: Timestamp('2012-07-01 00:02:52')
```

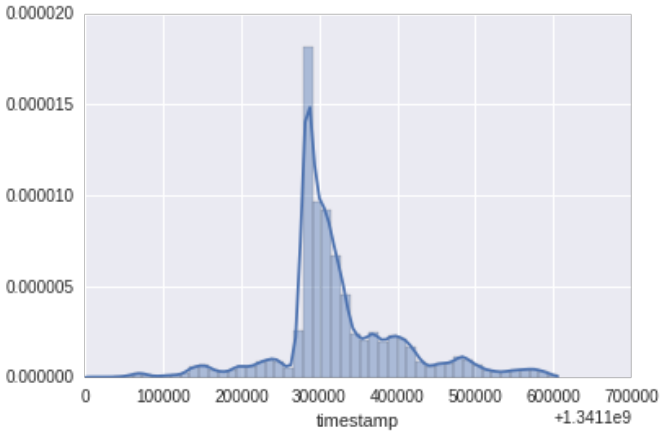
```
In [7]: timestamp_max = higgs_df['datetime'].max()
        timestamp_max
```

```
Out[7]: Timestamp('2012-07-07 23:59:53')
```

The data ranges from 2011-07-01 to 2012-07-07.

```
In [8]: sns.distplot(higgs_df['timestamp'])
```

```
Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa6f444a5c0>
```



The spike in tweets should be correlated with the time Higgs boson was discovered.

In [9]:
```
time_hist = np.histogram(higgs_df['timestamp'], bins=100)
timestamp_max_interactions = np.floor(
    time_hist[1][np.argmax(time_hist[0])])
timestamp_max_interactions
```

Out[9]:
```
1341379097.0
```

In [75]:
```
datetime_max_interactions = (
    higgs_df['datetime']
    [higgs_df['timestamp'] == timestamp_max_interactions]
    .unique())

print(datetime_max_interactions)
```

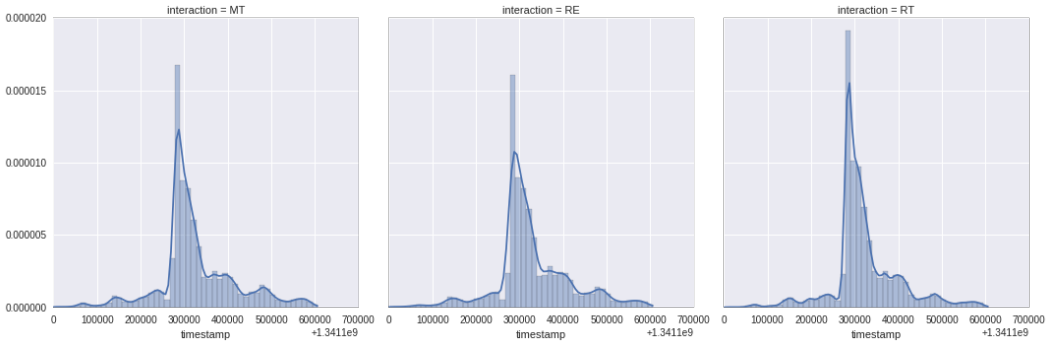
```
['2012-07-04T05:18:17.000000000']
```

The peak in interactions corresponds to 2012-07-04. Therefore, data was recorded in a range of plus and minus 3 days from the discovery.

It is also useful to plot

In [11]:
```
g = sns.FacetGrid(higgs_df, col='interaction', size=5)
g.map(sns.distplot, 'timestamp')
```

Out[11]:
```
<seaborn.axisgrid.FacetGrid at 0x7fa6f43e0cf8>
```



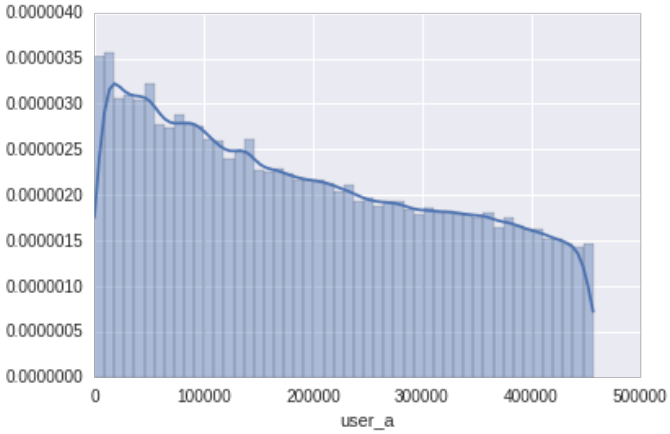
The distributions are quite similar across interaction types.

General graph analysis

The following plot corresponds to the distribution of interactions per source user, and the following one for the target ones.

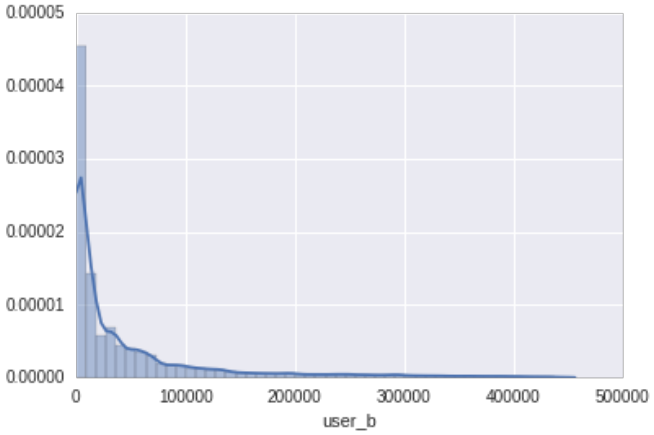
In [12]:
```
sns.distplot(higgs_df['user_a'])
```

Out[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa6f42140b8>
```



In [13]:
```
sns.distplot(higgs_df['user_b'])
```

Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa6f43dbf60>
```



Data preparation

shape: (steps, interaction)

In [14]:
```
higgs_reindexed = higgs_df.set_index('timestamp').sort_index()
```

In [15]:
```
higgs_reindexed.head()
```

Out[15]:

	user_a	user_b	interaction	datetime
timestamp				
1341100972	223789	213163	MT	2012-07-01 00:02:52
1341100972	223789	213163	RE	2012-07-01 00:02:52
1341101181	376989	50329	RT	2012-07-01 00:06:21
1341101183	26375	168366	MT	2012-07-01 00:06:23
1341101192	376989	13813	RT	2012-07-01 00:06:32

In [16]:
```
higgs_one_hot = higgs_reindexed.drop('datetime', axis=1)
higgs_one_hot = pd.get_dummies(higgs_one_hot, columns=['interaction'])

higgs_one_hot.head()
```

Out[16]:

	user_a	user_b	interaction_MT	interaction_RE	interaction_RT
timestamp					
1341100972	223789	213163	1.0	0.0	0.0
1341100972	223789	213163	0.0	1.0	0.0
1341101181	376989	50329	0.0	0.0	1.0
1341101183	26375	168366	1.0	0.0	0.0
1341101192	376989	13813	0.0	0.0	1.0

Build User Dictionaries

In [17]:
```
unique_users = pd.concat([higgs_one_hot['user_a'], higgs_one_hot['user_b
']]).unique()
```

In [18]:
```
num_users = len(unique_users)
```

In [19]:
```
idx2user = list(unique_users)
user2idx = {u: i for i, u in enumerate(idx2user)}
```

And convert the original ids to the new ones

In [20]:
```
higgs_converted = higgs_one_hot.copy()

higgs_converted['user_a'] = higgs_converted['user_a'].apply(lambda u : u
ser2idx[u])
higgs_converted['user_b'] = higgs_converted['user_b'].apply(lambda u : u
ser2idx[u])

higgs_converted.head()
```

Out[20]:

	user_a	user_b	interaction_MT	interaction_RE	interaction_RT
timestamp					
1341100972	0	50	1.0	0.0	0.0
1341100972	0	50	0.0	1.0	0.0
1341101181	1	279379	0.0	0.0	1.0
1341101183	2	279380	1.0	0.0	0.0
1341101192	1	54259	0.0	0.0	1.0

In [73]:
```
num_users
```

Out[73]: 304691

Obtain train, validation and test sets

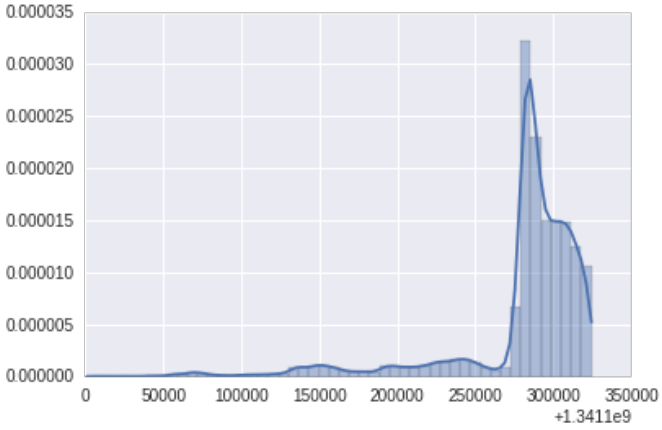
In [21]:
```
test_idx = int(len(higgs_converted) * 0.8)
higgs_test = higgs_converted.iloc[test_idx:]
```

In [22]:
```
valid_idx = int(test_idx * 0.8)
higgs_valid = higgs_converted.iloc[valid_idx:test_idx]
```

In [23]:
```
higgs_train = higgs_converted.iloc[:valid_idx]
```

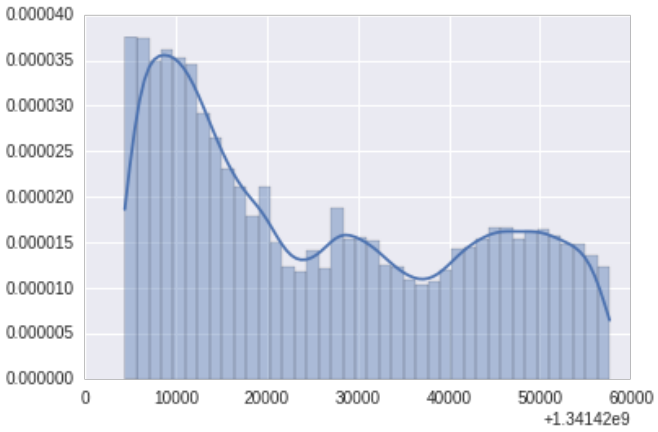
In [24]:
```
sns.distplot(higgs_train.index.values)
```

Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa6f3e600f0>



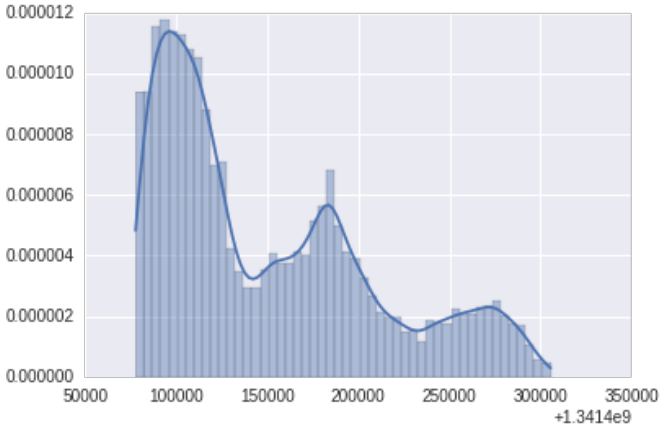
In [25]:
```
sns.distplot(higgs_valid.index.values)
```

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa6f3e67da0>



In [26]:
```
sns.distplot(higgs_test.index.values)
```

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x7fa6ef9a8f60>



Define features and labels

In [27]:
```
features = ['user_a']#, 'interaction_MT', 'interaction_RE', 'interaction_RT']
labels = ['user_b']
```

Long Short-Term Memory Network

In [28]:
```
import tensorflow as tf
```

Parameters

In [29]:
```
learning_rate = 0.001
num_epochs = 10
batch_size = 20
num_steps = 20
display_step = 10
```

Network parameters

In [30]:
```
is_training = True

num_input = 4
num_hidden = 200
num_embedding = num_hidden
num_layers = 2
keep_prob = 1.0
```

tf Graph input

In [31]:
```
x = tf.placeholder(tf.int32, [batch_size, num_steps])
```

In [32]:
```
y = tf.placeholder(tf.int32, [batch_size, num_steps])
```

In [33]:
```
global_step = tf.Variable(0, name='global_step', trainable=False)
```

Create LSTM cell

In [34]:
```
lstm_cell = tf.nn.rnn_cell.BasicLSTMCell(num_hidden, forget_bias=0.0)
```

```
WARNING:tensorflow:<tensorflow.python.ops.rnn_cell.BasicLSTMCell object a
t 0x7fa6ebc0ea58>: Using a concatenated state is slower and will soon be
deprecated.  Use state_is_tuple=True.
```

In [35]:
```
cell = tf.nn.rnn_cell.MultiRNNCell([lstm_cell] * num_layers)
```

In [36]:
```
initial_state = cell.zero_state(batch_size, tf.float32)
```

Create embeddings for users

In [37]:
```
with tf.device('/cpu:0'):
    embedding = tf.Variable(tf.random_uniform([num_users, num_embedding]
), name="embedding")
    embed = tf.nn.embedding_lookup(embedding, x)
```

Prepare the inputs tensor for LSTM

In [39]:
```
inputs = tf.transpose(embed, [1, 0, 2])
inputs = tf.split(0, num_steps, inputs)
inputs = [tf.squeeze(i) for i in inputs]
```

Define the forward pass

In [40]:
```
outputs, state = tf.nn.rnn(cell, inputs, initial_state=initial_state)
```

In [41]:
```
outputs = tf.reshape(tf.concat(1, outputs), [-1, num_hidden])
```

In [42]:
```
softmax_w = tf.Variable(tf.random_normal([num_hidden, num_users]), name=
'softmax_w')
softmax_b = tf.Variable(tf.zeros([num_users]), name='softmax_b')
```

In [43]:
```
logits = tf.nn.bias_add(tf.matmul(outputs, softmax_w), softmax_b)
```

Compute loss

In [44]:
```
loss = tf.nn.seq2seq.sequence_loss_by_example([logits],
                                              [tf.reshape(y, [-1])],
                                              [tf.ones([batch_size * num
_steps])])
```

In [45]:
```
cost = tf.div(tf.reduce_sum(loss), batch_size, name='cost')
```

In [46]:
```
tf.scalar_summary('cost', cost)
```

Out[46]:
```
<tf.Tensor 'ScalarSummary:0' shape=() dtype=string>
```

Compute accuracy

In [47]:
```
correct_pred = tf.equal(tf.cast(tf.argmax(logits, 1), tf.int32), tf.resh
ape(y, [-1]))
```

In [48]:
```
num_correct = tf.reduce_sum(tf.cast(correct_pred, tf.float32))
```

In [49]:
```
batch_accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

In [50]:
```
tf.scalar_summary('accuracy', batch_accuracy)
```

Out[50]:
```
<tf.Tensor 'ScalarSummary_1:0' shape=() dtype=string>
```

Perform optimization

In [51]:
```
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(co
st)
```

Create directory to save the model

In [60]:
```
ckpt_dir = 'tensorflow_ckpt'
```

In [61]:
```
if not os.path.exists(ckpt_dir):
    os.mkdir(ckpt_dir)
```

Merge all summaries

In [62]:
```
merged_summary_op = tf.merge_all_summaries()
```

Initialize the variables

In [63]:
```
saver = tf.train.Saver()
```

In [64]:
```
init = tf.initialize_all_variables()
```

Run the graph

In [69]:
```
def run_epoch(df, sess, op, epoch, summary_writter, verbose):

    num_batches = ((len(df.index) // batch_size)) // num_steps

    costs = 0.0
    iters = 0
    correct_count = 0
    start_time = time.time()
    for step in range(num_batches):

        batch_xs = (df
                    .loc[:, features]
                    .iloc[step * batch_size * num_steps : (step + 1) * b
atch_size * num_steps, :]
                    .as_matrix()
                    .reshape([batch_size, num_steps]))

        batch_ys = (df
                    .loc[:, labels]
                    .iloc[step * batch_size * num_steps : (step + 1) * b
atch_size * num_steps, :]
                    .as_matrix()
                    .reshape([batch_size, num_steps]))

        batch_loss, batch_correct, acc, _ = sess.run([cost, num_correct,
 batch_accuracy, op], feed_dict=
                                                     {x: batch_xs, y: batc
h_ys})

        summary_str = sess.run(merged_summary_op, feed_dict={x: batch_xs
, y: batch_ys})
        summary_writter.add_summary(summary_str, epoch * num_batches + s
tep)

        costs += batch_loss
        iters += num_steps
        correct_count += batch_correct

        if verbose and step % display_step == 0:
            print("{:.4f} perplexity: {:.3f} batch_acuracy: {:.3f} speed
: {:.0f} interactions/s"
                    .format(step * 1.0 / num_batches,
                            np.exp(costs / iters),
                            acc,
                            iters * batch_size / (time.time() - start_ti
me)))

    perplexity = np.exp(costs / iters)
    accuracy = correct_count / (iters * batch_size)
    return (perplexity, accuracy)
```

In [70]:
```
with tf.Session() as sess:
    sess.run(init)

    # check if there is already a saved model
    ckpt = tf.train.get_checkpoint_state(ckpt_dir)
    if ckpt and ckpt.model_checkpoint_path:
        print(ckpt.model_checkpoint_path)
        saver.restore(sess, ckpt.model_checkpoint_path) # restore all va
riables

    # get last global step
    start = global_step.eval()
    print("Start from {}".format(start))

    # Set logs into folder tensorflow_logs
    train_summary_writter = tf.train.SummaryWriter('tensorflow_logs_trai
n', graph=sess.graph)
    valid_summary_writter = tf.train.SummaryWriter('tensorflow_logs_vali
d', graph=sess.graph)
    test_summary_writter = tf.train.SummaryWriter('tensorflow_logs_test'
, graph=sess.graph)

    for epoch in range(start, num_epochs):
        train_perplexity, train_accuracy = run_epoch(higgs_train, sess,
optimizer, epoch, train_summary_writter, verbose=True)
        print("Epoch {:d} Train Perplexity: {:.3f} Accuracy: {:.3f}".for
mat(epoch+1, train_perplexity, train_accuracy))

        valid_perplexity, valid_accuracy = run_epoch(higgs_valid, sess,
tf.no_op(), epoch, valid_summary_writter, verbose=False)
        print("Epoch {:d} Validation Perplexity: {:.3f} Accuracy: {:.3f}
".format(epoch+1, valid_perplexity, valid_accuracy))

        # Save model at each epoch
        global_step.assign(epoch).eval()
        saver.save(sess, ckpt_dir + '/model.ckpt', global_step=global_st
ep)

    test_perplexity, test_accuracy = run_epoch(higgs_test, sess, tf.no_o
p(), 0, test_summary_writter, verbose=False)
    print("Test Perplexity: {:.3f} Accuracy: {:.3f}".format(test_perplex
ity, test_accuracy))
```

```
Start from 0
0.0000 perplexity: 431190.698 batch_acuracy: 0.000 speed: 60 interactions
/s
0.0111 perplexity: 166688.316 batch_acuracy: 0.130 speed: 57 interactions
/s
0.0222 perplexity: 215905.184 batch_acuracy: 0.002 speed: 53 interactions
/s
0.0333 perplexity: 180139.946 batch_acuracy: 0.097 speed: 52 interactions
/s
0.0444 perplexity: 174826.673 batch_acuracy: 0.020 speed: 51 interactions
/s
0.0556 perplexity: 173051.845 batch_acuracy: 0.025 speed: 50 interactions
/s
0.0667 perplexity: 164725.366 batch_acuracy: 0.058 speed: 50 interactions
/s
0.0778 perplexity: 150295.344 batch_acuracy: 0.055 speed: 50 interactions
/s
0.0889 perplexity: 151261.356 batch_acuracy: 0.023 speed: 49 interactions
/s
0.1000 perplexity: 143330.790 batch_acuracy: 0.123 speed: 49 interactions
/s
0.1111 perplexity: 137888.837 batch_acuracy: 0.062 speed: 49 interactions
/s
0.1222 perplexity: 134070.015 batch_acuracy: 0.132 speed: 49 interactions
/s
0.1333 perplexity: 132546.116 batch_acuracy: 0.075 speed: 49 interactions
/s
0.1444 perplexity: 134103.854 batch_acuracy: 0.055 speed: 49 interactions
/s
0.1556 perplexity: 137358.066 batch_acuracy: 0.045 speed: 48 interactions
/s
0.1667 perplexity: 122122.133 batch_acuracy: 0.558 speed: 48 interactions
/s
0.1778 perplexity: 104276.648 batch_acuracy: 0.195 speed: 48 interactions
/s
0.1889 perplexity: 92450.962 batch_acuracy: 0.207 speed: 48 interactions/
s
0.2000 perplexity: 87987.571 batch_acuracy: 0.062 speed: 48 interactions/
s
0.2111 perplexity: 86279.930 batch_acuracy: 0.123 speed: 48 interactions/
s
0.2222 perplexity: 85062.586 batch_acuracy: 0.060 speed: 48 interactions/
s
0.2333 perplexity: 80058.637 batch_acuracy: 0.078 speed: 48 interactions/
s
0.2444 perplexity: 75473.235 batch_acuracy: 0.087 speed: 48 interactions/
s
0.2556 perplexity: 71808.275 batch_acuracy: 0.083 speed: 48 interactions/
s
0.2667 perplexity: 63996.101 batch_acuracy: 0.153 speed: 48 interactions/
s
0.2778 perplexity: 61273.238 batch_acuracy: 0.160 speed: 48 interactions/
s
0.2889 perplexity: 59689.527 batch_acuracy: 0.185 speed: 48 interactions/
s
0.3000 perplexity: 58322.170 batch_acuracy: 0.140 speed: 48 interactions/
s
0.3111 perplexity: 58458.773 batch_acuracy: 0.095 speed: 48 interactions/
s
0.3222 perplexity: 59015.346 batch_acuracy: 0.127 speed: 47 interactions/
s
0.3333 perplexity: 59370.891 batch_acuracy: 0.127 speed: 47 interactions/
s
0.3444 perplexity: 59424.893 batch_acuracy: 0.112 speed: 47 interactions/
s
0.3556 perplexity: 58881.386 batch_acuracy: 0.162 speed: 47 interactions/
s
0.3667 perplexity: 57486.164 batch_acuracy: 0.115 speed: 47 interactions/
```

Appendix B

Simulation Model Computation Graph

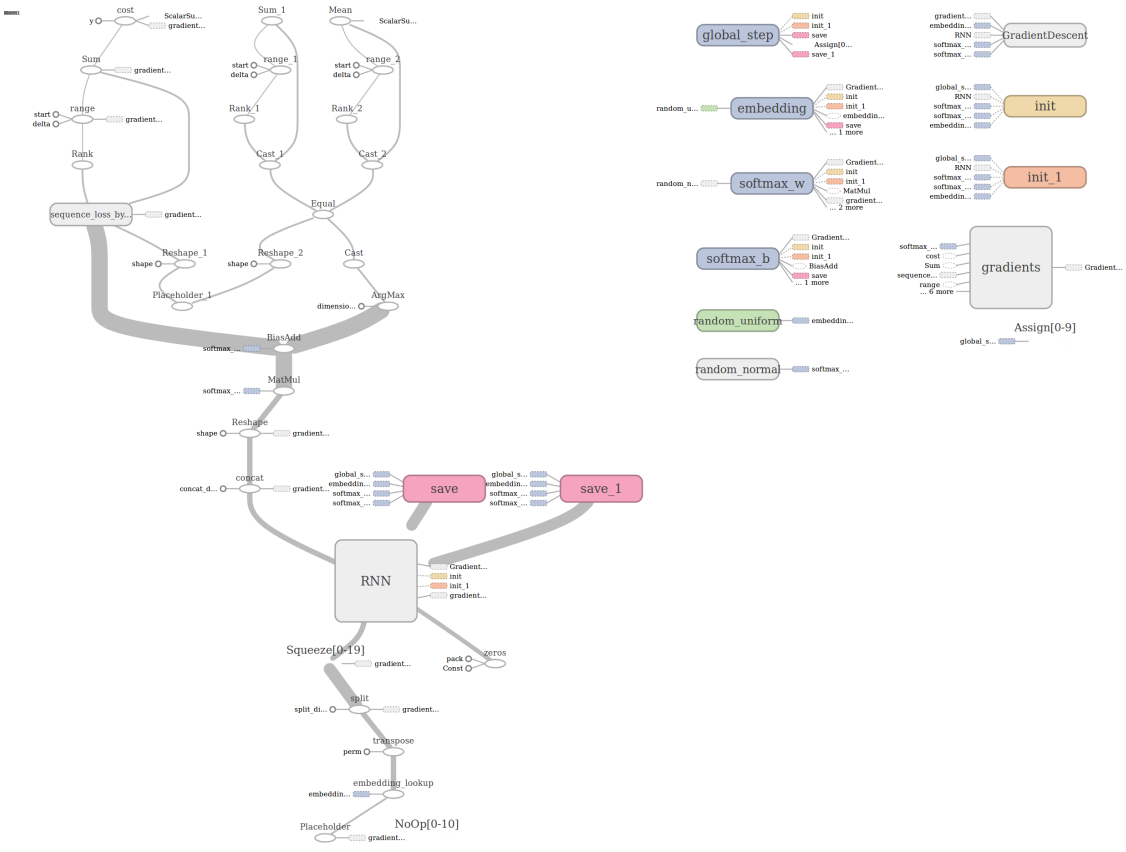


Figure B.1: Simulation Module computation graph (extracted using TensorBoard)