Dissertation presented to the Instituto Tecnológico de Aeronáutica, in partial fulfillment of the requirements for the degree of Master of Science in the Graduate Program of Electronic Engineering and Computation, Field of Informatics.

Gabriel Adriano de Melo

HYPERCUBE NEURAL NETWORKS FOR NATURAL LANGUAGE PROCESSING APPLIED TO STORY POINT ESTIMATION

Dissertation approved in its final version by signatories below:

[illegible]

Prof. Dr. Paulo Marcelo Tasinaffo

Advisor

[illegible]

Prof. Dr. Inaldo Capistrano Costa

Co-advisor

Prof. Dr. Pedro Teixeira Lacava

Dean for Graduate Education and Research

Campo Montenegro
São José dos Campos, SP - Brazil
2019

Cataloging-in Publication Data
Documentation and Information Division

Adriano de Melo, Gabriel
Hypercube neural networks for natural language processing applied to story point estimation / Gabriel Adriano de Melo.
São José dos Campos, 2019.
126f.

Dissertation of Master of Science – Course of Electronic Engineering and Computation. Area of Informatics – Instituto Tecnológico de Aeronáutica, 2019. Advisor: Prof. Dr. Paulo Marcelo Tasinaffo. Co-advisor: Prof. Dr. Inaldo Capistrano Costa.

1. Neural Networks. 2. Deep Learning. 3. Story Points. 4. Agile software development. 5. Natural Language Processing. 6. Scrum. I. Instituto Tecnológico de Aeronáutica. II. Title.

BIBLIOGRAPHIC REFERENCE

ADRIANO DE MELO, Gabriel. **Hypercube neural networks for natural language processing applied to story point estimation**. 2019. 126f. Dissertation of Master of Science – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSION OF RIGHTS

AUTHOR'S NAME: Gabriel Adriano de Melo
PUBLICATION TITLE: Hypercube neural networks for natural language processing applied to story point estimation.
PUBLICATION KIND/YEAR: Dissertation / 2019

Gabriel Adriano de Melo
Praça Marechal Eduardo Gomes, 50
12.228-900 – São José dos Campos–SP, Brazil

HYPERCUBE NEURAL NETWORKS FOR NATURAL LANGUAGE PROCESSING APPLIED TO STORY POINT ESTIMATION

Gabriel Adriano de Melo

Thesis Committee Composition:

ITA

To God, to my family, to my advisor and my friends.

Acknowledgments

I would like to express my profound gratitude to Professor Paulo Tasinaffo, my research advisor, and to Professor Inaldo Costa, my research co-advisor, for their helpful criticism, enthusiastic support and patient advice of this research.

I would also like to thank Dr. Morakot Choetkiertikul, for supplying me his dataset and a baseline model. My grateful thanks are also extended to 1st Lt Edney Da Silva Santos and 1st Lt Marcio Murilo Conte Monteiro for requesting the analysis and providing the case study data.

I would also like to extend my thanks to Professor Victor Curtis and to Professor Carlos Alonso for their help in offering me the department's laboratory and the computational resources required for training the artificial neural networks models.

To my parents and to my family I must express my very deep gratitude for providing me with unfailing motivational assistance, support and encouragement throughout those years of study and through the work of inquiring, researching and developing this thesis. Without them the elaboration of this research would not have been feasible. My dear friend Vitor Pimenta was also of great help as we developed the multi-frequency character model for binary analysis.

Finally, but by no means least, I take this opportunity to express gratitude to all the members, professors, technicians and colleagues of the Computer Engineering department for their support, assistance and contribution to my education. The Brazilian Aeronautics Institute of Technology (ITA), the Brazilian Aeronautics Center of Computation in São José dos Campos (CCA-SJ) and the Brazilian Air Force (FAB) were essential in funding this work.

"We know very little, and yet it is astonishing that we know so much,
and still more astonishing that so little knowledge can give us so much power."
— Bertrand Russell

Resumo

Pontos de estória (*story points*) são a unidade mais utilizadas na estimativa de esforço de uma estória de usuário (*user story*) nas metodologias ágeis de desenvolvimento de *software*. A utilização de redes neurais profundas na realização dessa estimativa é pouco presente na literatura mas poderia se tornar uma nova ferramenta de estimativa nas equipes ágeis. A entrada para o modelo de rede neural proposto é a própria descrição textual da estória de usuário e a sua saída é uma estimativa numérica. São utilizados modelos neurais recorrentes, na qual a sequência das palavras é levada em consideração a cada iteração da rede neural. Como dados de treinamento, foram utilizados projetos de código aberto que fazem uso de metodologias ágeis e descrevem suas funcionalidades em estórias de usuários com seus respectivos pontos de usuário associados, levantando-se milhares de casos de teste obtidos pela literatura. Uma nova arquitetura baseada em hipercubos de cliques dos neurônios foi proposta, caracterizando-se uma rede neural com conexões esparsas cuja eficiência computacional em espaço e potencialmente em tempo de treinamento foi superior à arquitetura de similar acurácia de classificação. A fim de melhorar a explicabilidade do modelo de linguagem e de regressão neural, utilizaram-se técnicas de visualização das suas ativações e de sua sensibilidade a variações nas entradas. Um estudo de caso também foi realizado no Centro de Computação da Aeronáutica de São José dos Campos (CCA-SJ) por meio da coleta de dados de projetos anteriores, preprocessamento e treinamento específico da rede, efetuando-se a sua posterior avaliação e comparação com modelos nulos que consideram apenas a distribuição dos pontos de estória.

Abstract

Story points are the most commonly used unit in estimating a user's story effort in agile software development methodologies. The use of deep neural networks to make this estimate is little present in the literature but could become a new estimation tool in agile teams. The input to the proposed neural network model is the textual description of the user story itself and its output is a numerical estimate. Recurrent neural models are employed, in which the word sequence is taken into account with each iteration of the neural network. As training data, we used open source projects that make use of agile methodologies and describe their functionalities in user stories with their respective associated user points, raising thousands of test cases obtained from the literature. A new architecture based on hypercube cliques of neurons was proposed, characterizing a neural network with sparse connections whose computational efficiency in space and potentially in training time was superior to the architecture of similar classification accuracy. In order to improve the explicability of the language and neural regression model, visualization techniques of its activations and its sensitivity to variations in inputs were used. A case study was also carried out at the Aeronautics Computing Center in São José dos Campos (CCA-SJ) by collecting data from previous projects, pre-processing and specific training of the network, making its subsequent evaluation and comparison with null models that consider only the distribution of story points.

List of Figures

List of Tables

List of Listings

List of Abbreviations and Acronyms

AI	Artificial Intelligence
Adam	Adaptive Moment Estimation
AGI	Artificial General Intelligence
ANN	Artificial Neural Network
ALPAC	Automatic Language Processing Advisory Committee
ARPA	Advanced Research Projects Agency
AWCFT	Algorithmic Warfare Cross-Functional Team
CCA-SJ	Aeronautics Computation Center in São José dos Campos (*Centro de Computação da Aeronáutica de São José dos Campos*)
CNN	Convolutional Neural Netork
DARPA	Defense Advanced Research Projects Agency
Deep-SE	Deep Story points Estimation
DL	Deep Learning
DNN	Deep Neural Network
FAB	Brazilian Air Force (*Força Aérea Brasileira*)
FCNN	Fully Connected Neural Network
GRU	Gated Recurrent Unit
ITA	Aeronautics Institute of Technology (*Instituto Tecnológico de Aeronáutica*)
JAIC	Joint Artificial Intelligence Center
LSTM	Long-Short Term Memory
MAE	Mean Absolute Error
MdAE	Median Absolute Error
ML	Machine Learning
MMRE	Mean Magnitude of Relative Error
NCE	Noise Constrastive Estimation
NLP	Natural Language Processing
NTLK	Natural Language Toolkit
ODE	Ordinary Differential Equation
PO	Product Owner

RG	Renormalization Group
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SE	Software Engineering
SGD	Stochastic Gradient Descent Optimization
SM	Scrum Master
SNN	Sparse Neural Networks
SVD	Singular Value Decomposition
t-SNE	T-distributed Stochastic Neighbor Embedding
TF-IDF	Term Frequency Inverse Document Frequency transformation

List of Symbols

$\boldsymbol{b}$	Bias vector b
$\boldsymbol{W}$	Weight Matrix W
$\mathsf{\mathbf{W}}$	Tensor W
$\mathbf{e}_j$	One-hot encoding for the j-th index
$\boldsymbol{W}^{(N)}$	N-th layer weight matrix W
$\boldsymbol{W}_{i,j}^{(N)}$	N-th layer weight between the i-th neuron at the layer $N-1$ and the j-th neuron at the layer N
$b_i^{(N)}$	N-th layer bias from the i-th neuron
$\boldsymbol{I}_n$	Unit matrix with n columns and n rows
$\boldsymbol{W}_{i,j}$	Element in i-th row and j-th column of matrix $\boldsymbol{W}$
$\boldsymbol{W}_{i,:}$	i-th tow of matrix $\boldsymbol{W}$
$\boldsymbol{W}_{:,i}$	i-th column of matrix $\boldsymbol{W}$
$\boldsymbol{W}^\top$	Transpose of matrix $\boldsymbol{W}$
$\det(\boldsymbol{W})$	Determinant of $\boldsymbol{W}$
$\boldsymbol{W} \odot \boldsymbol{A}$	Hadamard (Element-wise) product of $\boldsymbol{W}$ and $\boldsymbol{A}$
$\frac{dw}{dc}$	Derivative of w with respect to c
$\frac{\partial w}{\partial c}$	Partial derivative of w with respect to c
$\nabla_{\boldsymbol{c}} w$	Gradient of w with respect to $\boldsymbol{c}$
$\mathbb{L}$	Cost Function
σ	Activation Function
$\ln x$	Natural Logarithm (base e) of x
$\log x$	Logarithm in base 2 of x
$\mathbb{R}$	Real numbers set

$\Re(x)$	Real part from a complex number x
$\Im(x)$	Imaginary part from a complex number x
$\langle v_1, v_2, \ldots, v_n\rangle$	n-dimensional vector representation
$\|\|\boldsymbol{x}\|\|$	$\boldsymbol{x}$ vector L-2 norm
$\boldsymbol{v}_1 \cdot \boldsymbol{v}_2$	Internal product between vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$

Contents

1 Introduction

In this introductory chapter, a historical background of artificial intelligence research is presented in order to contextualize the natural language processing (NLP) problem motivation and definition in the Scrum environment, which are explained in sequence. It is also important to note the transferability of those definitions to other domains in NLP as a generalization of sentiment analysis.

Following the problem definition, the proposed solution is briefly described as a deep neural network model, accompanied by the objectives and contributions of this work. Formally, the work organization is lastly exposed, which succinctly enumerates and discloses the chapters of this thesis.

1.1 Research Context

Even though the idea of mechanical mens and the principles of logic may be dated back to antiquity with Aristotle, Greek mythology and other philosophers, the term of “Artificial Intelligence” (AI) was coined in 1956 by John McCarthy and Marvin Minsky at the Dartmouth Summer Research Project on Artificial Intelligence (DSRPAI). By this time, both Claude Shannon and Alan Turing had already published their work on a chess playing computer programs and the question whether can machines think, respectively. For the next decade, there was continuous improvements and the expectations grew larger as the American Government embodied by ARPA (Advanced Research Projects Agency) expanded its funding in AI. Fulfilling those expectations would require the development of new techniques and much more processing power than was available at the time, thus there was a stagnation on its popularity and a cut on the fundings known as the “AI Winter”.

At the 80’s, a new wave of research in AI was reborn as expert systems gained popularity and parallel computers were developed for Symbolic AI. The Japanese Fifth Generation Computer Project was also another great exponent for the ambitions pursued at the time. But again the expectations were unmet and the area was faced with another AI Winter.

In the age of Big Data and with more available processing power, AI research, specially the field of Machine Learning (ML), was able to thrive once again, becoming the cornerstone for innumerous significant applications, including product recommendations, language translation, speech recognition, search engines, image detection, autonomous vehicles and many others. Andrew Ng (2017) emphasizes the rapid progress that such applications may provide to start-ups and established companies driven by data availability and computational scale of contemporary advancement.

In this context, just like electricity have revolutionized the world a century ago, the technology for Artificial Intelligence is potentially transforming every sector in the economy with an estimate of over $10 trillion in GDP growth accumulated by 2030 (NG, 2018). Besides being already employed and crucial for the technology giants such as Facebook, Tencent, Google, Baidu and Microsoft to keep their competitive advantage, there is still much innovation and applicability outside of the software developing domain. Ng (2018) notes five principles for organizations that want to create an Artificial Intelligence Strategy, those are the execution of pilot projects in order to gain momentum and experience in the development and deployment of such systems, the formation of an in-house AI team, the broad AI training administered to all personnel, the design of an AI strategy aligned with the "Virtuous Circle of AI" positive-feedback loop and, finally, the development of internal and external communications in the field.

It is important to note that this "Virtuous Circle of AI" is characterized by its positive-feedback loop in the sense that a good AI application will result in more users which, in turn, will provide more data to the system, increasing its training database and potentially its quality. Such virtuous circle may be observed in Figure 1.1 depicting the Japanese strategy for developing AI countrywide as a multipurpose service (JAPAN, 2017).

In the international context, the American Government has recognized the tremendous potential that Artificial Intelligence possesses to benefit its people, which is demonstrated by the extensive value already noted in the White House AI Summit by enhancing their national security and by growing their economy (USA, 2018). Is was expressed that AI will effectively be employed a tool to empower the workers, compelling advances in the business, and enhancing the quality of life for its citizens. These benefits are declared in the American National AI Strategy, in which high-level authorities across the Federal Government recognize the critical nature of this issue and emphasize the dedication to leveraging AI techniques across agencies tasks (USA, 2018). Therefore The United States of America aims to continue its global leadership even though the recent advancement of China in AI threatens that objective in an AI Arms Race (MELO *et al.*, 2019).

The designation of Artificial Intelligence as an R&D priority by the White House reinforces the importance that this area has for the coming years, stating that "we are on the verge of new technological revolutions that could improve virtually every aspect of

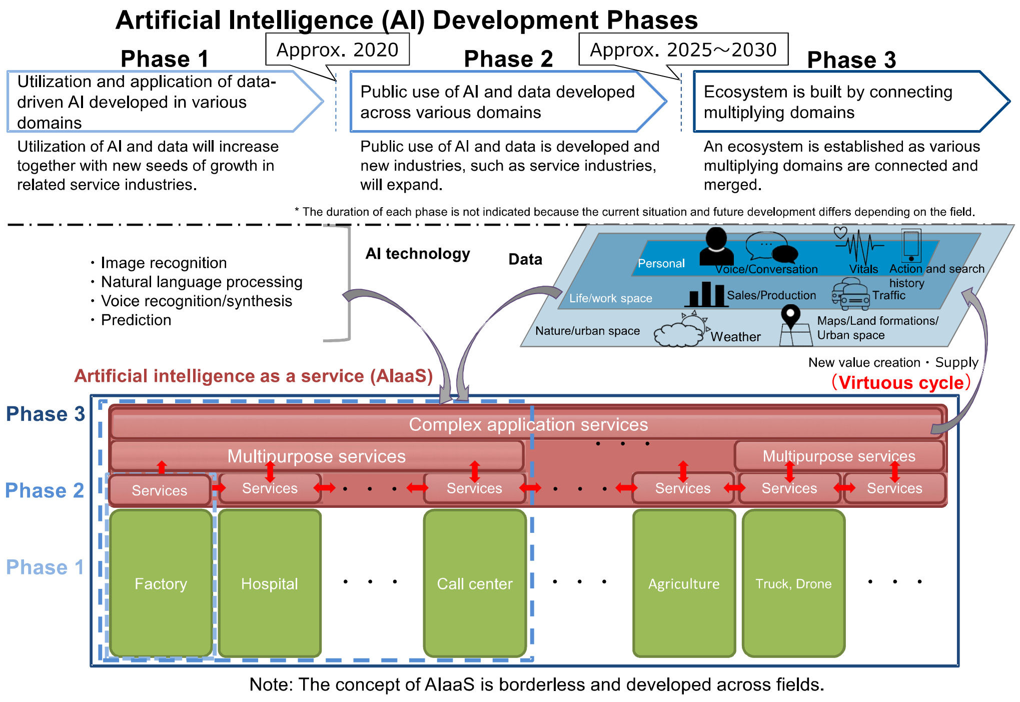

Artificial Intelligence (AI) Development Phases
Phase 1
Approx. 2020
Phase 2
Approx. 2025～2030
Phase 3
Utilization and application of data-driven AI developed in various domains
Public use of AI and data developed across various domains
Ecosystem is built by connecting multiplying domains
Utilization of AI and data will increase together with new seeds of growth in related service industries.
Public use of AI and data is developed and new industries, such as service industries, will expand.
An ecosystem is established as various multiplying domains are connected and merged.
* The duration of each phase is not indicated because the current situation and future development differs depending on the field.
・Image recognition
・Natural language processing
・Voice recognition/synthesis
・Prediction
AI technology
Data
Personal
Voice/Conversation
Vitals
Action and search history
Life/work space
Sales/Production
Traffic
Nature/urban space
Weather
Maps/Land formations/ Urban space
New value creation・Supply
（Virtuous cycle）
Artificial intelligence as a service (AIaaS)
Phase 3
Phase 2
Phase 1
Complex application services
Multipurpose services
Multipurpose services
Services
Factory
Hospital
Call center
Agriculture
Truck, Drone
Note: The concept of AIaaS is borderless and developed across fields.

FIGURE 1.1 – Japanese artificial intelligence technology strategy development phases. Source: (JAPAN, 2017).

our lives, create vast new wealth [. . .], open up bold, new frontiers in science, medicine, and communication" (USA, 2018, p.2). In order to fulfill those objectives, they acknowledge that it is necessary engagement with the private sector, academia and their agency partners by informing their policies, funding and supporting AI related projects.

Those main strategies may be summed up as the prioritized funding for AI fundamental research and development including autonomous systems, computing hardware infrastructure and machine learning; the removal of regulatory barriers to the implementation and deployment of technologies powered by artificial intelligence consequently enabling more innovation; mass education for the workforce by teaching the skills necessary to flourish in the new economy; leveraging a military advantage using AI in an asymmetrical and technological warfare implemented by the Department of Defense; the improvement of government services by applying AI in order to facilitate and to speed up the provision of public services; and, finally, the international leadership in AI negotiations by promoting G7 Innovation and Technology Ministerials by working with allied countries in order to influence them in recognizing the potential benefits of AI and promoting AI R&D (USA, 2018).

Although several countries have already published their National Artificial Intelligence Strategies, the Brazilian Government is yet to develop its own strategy. A research project is currently being conducted and some essential points based on others strategies and in the Brazilian particularities have been proposed (MELO *et al.*, 2019). The innovations brought by this technology may characterize the leapfrogging needed to accelerate the economy of developing countries, specially Brazil, by skipping obsolete infrastructure and by directly deploying the most advanced ones.

A variety of perspectives may be expressed about the issue of using AI for military applications. While the USA is indubitably interested in employing this new technology directly at the theater of war by using Lethal Autonomous Weapons Systems (LAWS) (DOD, 2019), Brazilian intends to develop dual applications that could be used for both civilians and armed forces. As such, systems that could automate a range of administrative and bureaucratic assignments are the main ambition for the Brazilian Armed Forces, freeing thousands of military personal from office work and increasing the efficiency of public services. Another non-combative usage is for gathering intelligence from terabytes of unstructured data, as done in Project Maven by the Department of Defense (DoD) Algorithmic Warfare Cross-Functional Team.

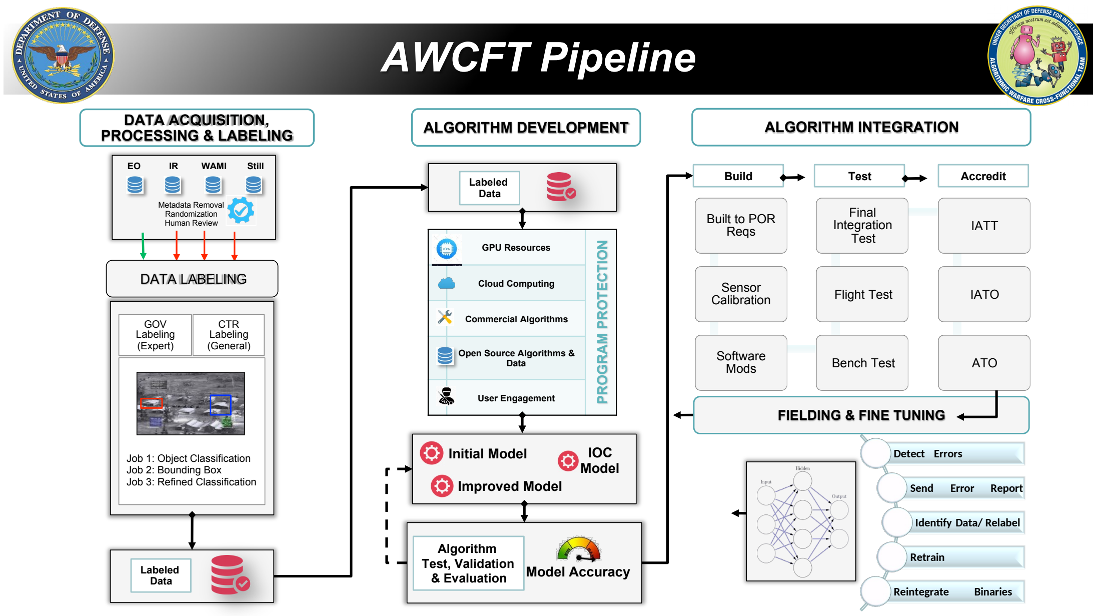


FIGURE 1.2 – Department of Defense (DoD) Algorithmic Warfare Cross-Functional Team (Project Maven) development pipeline. Source: (SHANAHAN, 2018).

Additionally to using natural language processing machine learning models to automate office work, convolutional neural networks are already employed by the Under Secretary of Defense for Intelligence (USDI) Warfighter Support in order to centralize existing algorithmic technology efforts and advance artificial intelligence efforts across the Defense Intelligence Enterprise (SHANAHAN, 2018). As examples, there are field AI for Processing, Exploitation, and Dissemination (PED) of Tactical Unmanned Aerial Vehicles (TUAV), medium altitude, and Wide-Area Motion Imagery Full Motion Video (FMV). The first algorithms were deployed in operation at the end of calendar year 2017. Its development pipeline is depicted in Figure 1.2, one that would seemingly be noted as a civilian application if it weren't for its contractors and label classes.

From this perspective, the Department of Defense Artificial Intelligence Strategy states the American ambition to use this contemporary technology in order to "support and protect their servicemembers" and civilians around the world, to create an efficient and streamlined organization and to "become a pioneer in scaling AI" across a global enterprise. They aspire to "anticipate the implications of those novel methodologies on the battlefield", rigorously defining the military problems to be anticipated in future conflict, and to stimulate a culture of "calculated risk-taking" and innovation. As such, the deliverance of AI-enabled capabilities capable of surmounting crucial challenges is key to those aspirations, potentialised from scaling its impact across their agencies through a prevalent authority for distributed innovation and development (DOD, 2019). And moreover, conjointly, they emphasizes the necessity to cultivate a leading AI workforce and to interplay with academic, commercial and international partners while maintaining the lead in military ethics and AI safety.

In a declaration that states the clear importance of having a AI leadership from an economic and military perspective, the American strategists weights the costs of a disregard for this technology, resulting in legacy systems that will be unable to defend their nation from uniques asymmetric adversaries (DOD, 2019). For that reason, there would be an erosion between their allies and an impairment of their influence around the world, resulting in a fall in the standard of living and diminishing prosperity as other competitors would take the lead. Nevertheless, despite the risk of growing inequalities, AI has the potential to increase global prosperity and to free human labor from tiresome tasks.

1.2 Motivation

The Aeronautics Computing Center in São José dos Campos (*Centro de Computação de Aeronáutica* – CCA-SJ) employs the Scrum agile methodology in order to manage and develop its in-house software projects. For this reason, it is remarkably important to make reasonable story points estimations in the interest of prioritizing and allocating resources for the software production.

According to Erdogan *et al.* (2018), one of the most common shortcomings in comparing the process and product quality is the lack of measurements. As such, a more effective and more predictive planning for the next sprints could be achieved by employing statistical analysis in the various metrics and data generated by the agile methodologies. As a novel tool in these processes, the machine learning model would support this objective by enabling the achievement of the quality target and key performance indicators while also maintaining the collaboration and interactions that exists in the agile framework.

Central to the Scrum framework are its core values derived from de Agile Manifesto (BECK *et al.*, 2001) which can be summed up as focus, respect, courage, commitment and openness. Based on these values, there are the following fundamental concepts for this methodology: iterative development, time-boxing, context, transparency, self-organization, empirical process control, collaboration, value-base, scrum master, product owner and experimentation.

It has been noted by Rubin (2012) that, since its creation in 1993 by Jeff Sutherland and his associates at the Easel Corporation, those organic principles, in conjunction with software process and productivity research, has guided the Scrum application. This methodology can be observed as an iterative process on Figure 1.3 in which others smaller dynamic procedures happen.

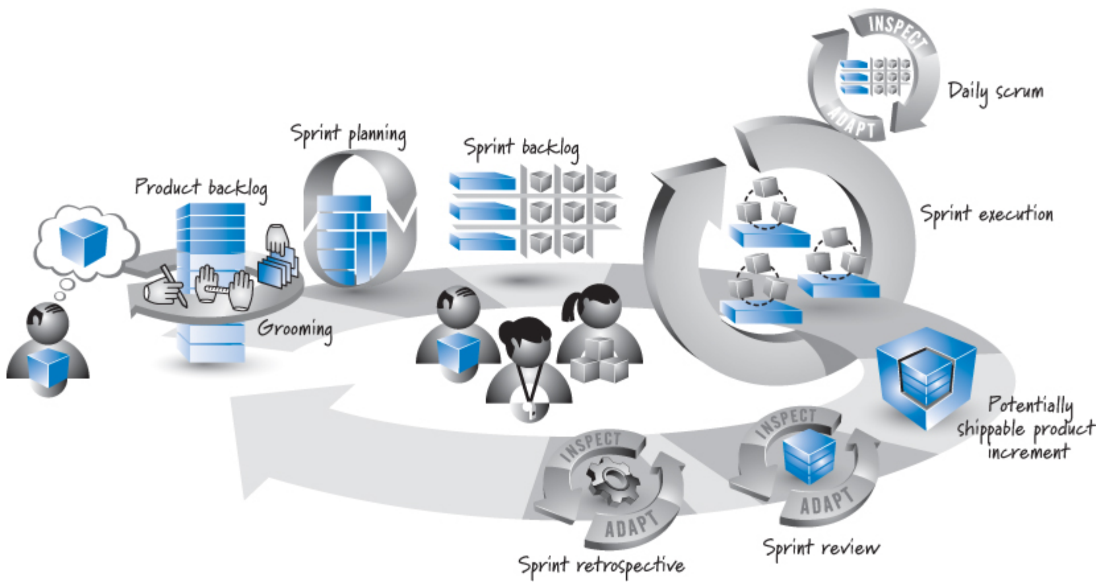


FIGURE 1.3 – The entirety of a Scrum sprint in conjunction with its smaller cycles depicting the framework. Source: (RUBIN, 2012).

In comparison to classical software development methodologies such as the waterfall model, as reported by Rubin (2012), the Scrum application was able to reduce tenfold the effort required to produce the specified computer program. It was also noted that the duration needed to deliver the application was decreased by a proportion of seven, thus being able to produce sevenfold more functions in the same time internal while employing a smaller team. From a customer's relation perspective, there is a closer approximation to the expectations characterized by a better alignment of their goals.

Therefore, it is important to note that Scrum administers and systematizes the teamwork in an organic way, not being a closed recipe in which a sequence of steps must be rigorously followed in order to achieve its goals of producing a excellent software within the resources constraints of time and money (RUBIN, 2012). From this point of view, the empowerment of its members allow a better cooperation in those projects, boosting the confidence and respectability the team has in itself and encouraging the focus and innovation essential for the agile development.

This way of organizing the team may be summed up as an integration between the development team, responsible for implementing and delivering results, the Product Owner (PO) who is the stakeholder in charge of expressing the requirements and priorities, and the Scrum Master (SM) accountable for applying the framework, protecting and helping the team. The genesis of this process comes from the Product Backlog in which the PO manages and determines the work that should be done in a priority queue. This collection of tasks must then be understood by the team and be assigned a value that correspond to its perceived difficulty. Namely, these are the user story, the requirements expressed in the end-user perspective, and the story points that convey the demanded effort's magnitude.

The collection of those user stories are assigned to a sprint which is a fixed time interval, typically between two to four weeks, whose work should create an incremental value tangible to the Product Owner. As show in Figure 1.3, the Sprint Planning marks the sprint's beginning when the three roles should gather together in order to define the goals for the sprint. A Kanban is employed in the sprint's execution, keeping track of the "to do, doing and done" possible states and allowing transparency of who is doing what to the team.

Additionally, there is a short daily meeting where the members ought to expose their challenges and accomplishments, sharing experiences to the team so that a solution may be found collaboratively. It is crucial to keep in mind the time constraints of this meeting not to waste precious developer's time. Finally, the value added in the sprint to the product should be reviewed and adapted. These are performed in the sprint review and retrospective activities providing a product and process analysis.

Motivated by the statistical work performed by Erdogan *et al.* (2018) in which the correlation between relative effort estimation made by the team (story points) and the actual effort measured in work-hours. The Figure 1.4 depicts this strongly positive Spearman correlation of approximately 0.79 for the range from zero to twenty-one in a Fibonacci scale. Therefore, based on the capacity that a human has to estimate the effort it was feasible that a machine learning system could have a performance better than random guess. Even further, such estimation system could also assist the planning poker activity in which the team gather together to make appropriate guesses until the consensus is reached by participating in the first round.

Another interesting point to mention from Figure 1.4 is the distribution of outliers that happens most frequently in the unitary story points estimation and the non-linear relation between the two variables at the lower and higher end of the graph. Such non-

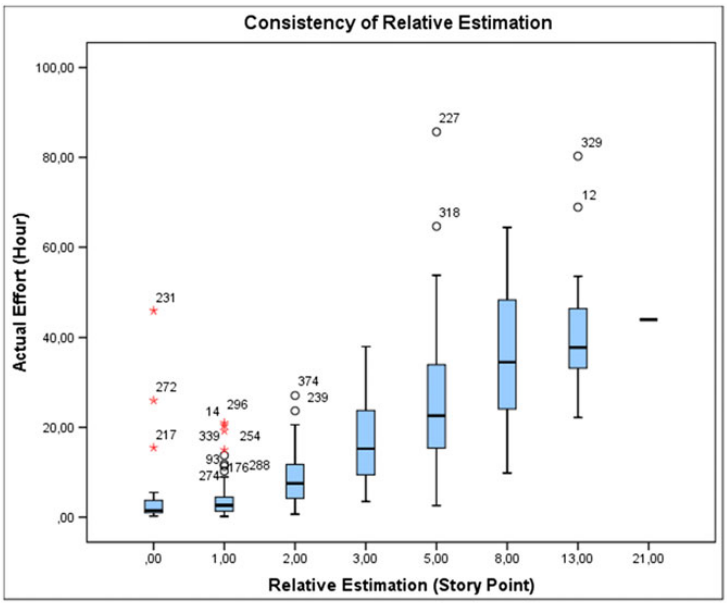


FIGURE 1.4 – Boxplot of actual effort in hours given its initial estimate in story points. Source: (ERDOGAN *et al.*, 2018).

linearity could be easily explored by an artificial neural network given enough data which would result in better estimations than a linear model.

1.3 Problem Statement

It is desirable to have a initial estimation that could be automatically produced while using software developing manager tools such as Jira, Redmine, Github or Gitlab issues tracking. This initial estimate would be used as a baseline for the planning poker guesses and other processes that would require effort estimation, leveraging the Product Owner's capabilities.

The problem this work aims undertaking is to estimate the story points from a user story's textual description of a given project that is inserted in the Scrum development framework. Given the specificities that different teams may have, it should be also possible to provide features that characterize the project and its team, this is being able to effectively calibrate the model to local data.

Given sufficient data for its training, more than ten thousand non-labeled user stories and more than one thousand labeled user stories with the actual story points the model should perform better than random guessing and average estimation. Thus, it is crucial to gather the most data as possible in order to reach these thresholds and increase the model's performance even further.

1.4 Proposed Solution

The proposed solution is to utilize artificial neural networks using deep learning techniques in order to develop a natural language processing model capable of estimating the story points from the user stories' textual descriptions. Recurrent neural architectures such as Long-Short Term Memory (LSTM) and Gated Recurrent Units (GRU) are employed, as well as a novel sparse architecture based on hypercubes.

The first and most essential step is to assemble the dataset and preprocess with the intention of creating a robust collection of user stories sampled from the true data distribution. Open source projects such as those found in the Apache issues tracking in Jira are leveraged as a means to compare the proposed architecture with the literature that had used the same dataset. A case study is conducted in the Aeronautics Center of Computation in São José dos Campos (CCA-SJ) whose data was collected from their Redmine issue tracking server.

Subsequently, a variety of hyperparameters that defines the architecture are evaluated in the test and validation datasets. Unique combinations formed by feedforward, sparse, GRU and LSTM networks are also assessed on those datasets. The selected model should then be evaluated only once in the test set not to compromise the unbiased estimation of its performance.

Finally, gradient based and black-box techniques are employed in an effort to create an explainable system. Thus, the end-user may take notice of the key features used by the neural network while making his decision. An explainable AI also inspires more confidence aggregating more value than a completely opaque model.

1.5 Objectives

The main objective is to develop a machine learning based computer program whose input is the textual description of a user story in the Scrum framework context and whose output is a story point estimate associated with it. This estimation should be more precise than random guess, average estimates and at least similar to other models presented in the literature.

As such, another objective concerning the case study is to assemble and organize the project management data from CCA-SJ, creating a better understanding and retrospective analysis of the processes employed. This work should also be an instrument to increase the data awareness and the importance of developing an Artificial Intelligence Strategy in order to apply this technology effectively and to increase overall productivity.

1.6 Contributions

The development of an application that could be integrated with project management tools such as Redmine or Jira to estimate story points given its user story in the central contribution of this paper.

A mathematical survey and development of a novel sparse architecture for neural networks is another important subsidy this work has to offer.

The appeal for developing an AI strategy and the endowment for an AI based culture is also a secondary contribution presented.

1.7 Outline

The present work is divided into: a bibliographical survey in Chapter 2; the mathematical foundations for deep learning in Chapter 3; the presentation and theoretical mathematical formulation of a natural language processing model in Chapter 4; the proposition of a new architecture that uses fewer parameters for neural networks in Chapter 5; a survey and characterization of the data, whether for the pre-training, training, validation and testing set in Chapter 6; the results obtained from the evaluation of the proposed model in the test set and its analysis in Chapter 7; and, finally, the final considerations of this paper in Chapter 8, along with suggestions for future research.

2 Literature Review

In addition to the applications of deep learning in software engineering, this chapter introduces an exploratory literature review focused on addressing the most recent and key researches developed in the artificial neural network and natural language processing fields. A brief presentation on Artificial Intelligence is also conducted in the first section to ensure an adequate background for understanding the later definitions.

2.1 Artificial Intelligence

There is not a unique definition for Artificial Intelligence (AI) that encompass all nuances concerning the term. Overall, there are four main conceptualizations that independently refers to the action exhibited by the system which may be **to act** or **to think** and the characteristics associated with these actions either **rationally** or **humanely** (RUSSELL; NORVIG, 2009).

Thereby, the first possible combination of those attribute is a machine capable of acting rationally. This concept motivates the clarification of an agent, an entity capable of interacting with its environment and, in this first case, maximizing its objectives given the data and capabilities it possesses. In contrast, the second combination would be a machine that acted like a human. This definition is directly connected with the Turing Test in which human examiners could not be able to differentiate the behavior of a human in comparison to a machine. In its weakest form, the interaction between test subjects and examiners is constrained by text.

The third possibility is a system that thinks rationally. This approach is based on the field of logics wherein a machine is able to derive truthful conclusions given a set of premises and the initial states concerning these premises. Lastly, the fourth most accepted definition is that of an artificial agent that thinks as a human does. This other agent could have a general capacity to solve a wide range of problems as humans do. These distinct approaches for artificial intelligence are tailored accordingly to different goals a researcher or a developer may have: to model the behavior or the thinking, and to be based on an ideal standard or on humans (RUSSELL; NORVIG, 2009). A generalized conception of a learning agent is show on Figure 2.1.

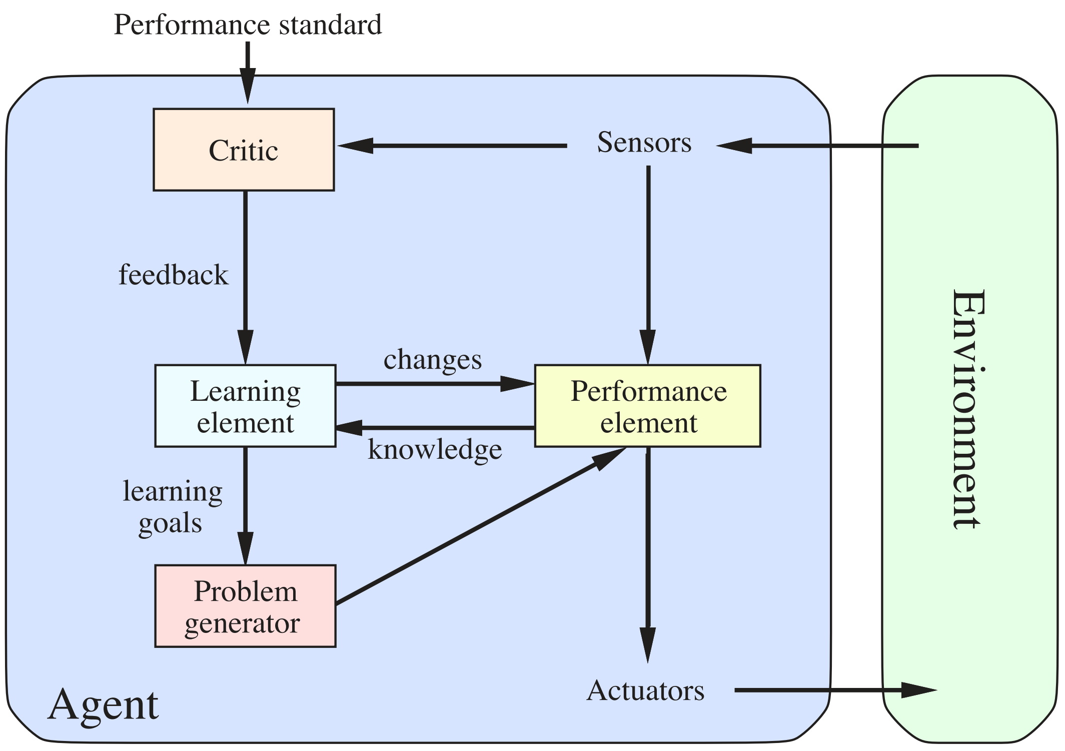


FIGURE 2.1 – The generalized design of a machine capable of learning. Adapted from: (RUSSELL; NORVIG, 2009)

A more informal approach to the definition of AI concerns the existential and utilitarian view on its employment. The artificiality characterizes a man-made, directly or indirectly (machine-made), product and its intelligence is a condition necessary for it to work. In this sense, a machine that couldn't work would be simply defined as unintelligent or, more informally, as dumb. This classification is, of course, dependent of the desired work specification, namely, the technical parameters of functionality. As such, the field of AI seeks the development of more capable agents, able to perform its task more efficiently.

Given this aspiration, its is central to AI the distinction between strong and weak AI. The first, also known as Artificial General Intelligence (AGI), is capable of solving problems that it was not initially programmed (or trained) to solve. In this context, a strong AI would have the capability of executing any intellectual task a human can perform. On the other hand, a weak AI can only solve the specific problems it was designed to determine. There are currently no AGI systems implemented, and its development is an open research question. As such, all AI created so far is characterized as weak, for example, a chess AI can only perform well in that game, not being able to do other tasks such as generating text.

An important subfield of Artificial Intelligence is that of Machine learning (ML), also know as statistical pattern recognition (BISHOP, 1995). In this subfield, the agent is capable of inferring structure from the data given during its training. In contrast to classical IA systems that requires an expert to explicitly program the rules, actions and states that such machine obeys, a ML approach is able to program itself just from observing examples of what is supposed to do. Its great challenge is to minimize the amount of data required and the computational resources employed, namely the problem of generalization.

Concerning the performance of such models, Wolpert (1996) has stated what is today know as **The No Free Lunch Theorem** of Machine Learning. In asserts that there are no a priori distinctions between models resulted from a statistical pattern recognition process for every possible problem. An application of this theorem is to extrapolate data given a finite set of prior points. There are infinitely many functions that could be a perfect fit for the prior points resulting in infinitely many possible and feasible extrapolations. If the error were to be computed in the entirety of such sets the result would be indistinguishable between different learning algorithms. In reality, the data distribution in the natural world favors some functions over others, enabling the effectiveness of more complex statistical pattern recognition models.

As the ambitions for developing an AGI system grows, Pearl (2019) brings seven instruments observed in human inference of causality. In this way, a machine learner should be able to understand the idea of causal relationship between connected events in order to exhibit the generalized aimed behavior. In the implementation order, those rules of cause and effect should be attainable by encoding the causal requirements in a model that permits transparency and testability, so that experts may understand and correct deviations; the so called "do-calculus" and the "control of confounding" or unobserved causes, estimating the consequences of changes in policies; the "algorithmitization of counterfactuals", that is to estimate the probability of an event happening had another parameter been different; mediation analysis in order to predict the variation in the consequences when perturbing its cause and the assessment of its direct and indirect effects; domain adaptability, sample selection bias in order to ensure robustness to changes in the environment and external validation of its performance; be able to recover the consistency of estimation from missing data; and finally, "causal discovery", in other words, to unveil the relationship that certain occurrences have on others (PEARL, 2019).

2.2 Deep Learning

The development of artificial neural networks has a long history dating back from 1943 when Warren McCulloch and Walter Pitts introduced the first model of such system. In 1949 Donald O. Hebb stated the prominent Hebbian Rule postulating that neurons firing together wire together, although this principle turned out to be unstable. By 1957, Frank

Rosenblatt and Charles Wightman created the Perceptron model, and soon publishing theirs learning algorithm capable of linear separation. At the time, great elaborations were made about this model even about differentiating military tanks from civilian buses which turned out to be simply because of the images' brightness as the tanks' pictures were taken on a cloudy day. Marking the first disillusion on neural networks, Seymour Papert and Marvin Minsky's 1969 work demonstrated the mathematical limitations that constrained the Perceptron Model. Following the reduced fund for AI research, by the 80', John Hopfield created his own network architecture to model associative memory, the Neocognitron was also presented by Fukushima, Miyake and Ito which would be similar to the coming Convolutional Neural Networks (CNN), and the backpropagation algorithm, also priorly developed by Paul Werbos, was published by Rumelhart, Hinton and Williams (KRIESEL, 2007). As a result, this research field is an particular application of machine learning that has gained popularity due to its increasing effectiveness when dealing with high amounts of data.

Zurada (1992) suggested that the development in this field was also fueled by the interdisciplinary nature of the area, attracting engineers, technologists, physicians and scientists from a wide range of disciplines. Mathematicians were inspired by the optimization problems and the modeling of complex systems while computer scientists were marveled by the parallel computing, supercomputing, networks and even more applications. Neuroscientists are also attracted by the biological networks models that may be simulated and artificially constructed together with psychologists that tries to understand the high-level processing that occurs in human brains. Also computer and electrical engineers use signal processing applied to artificial neural networks and develop neuromorphic computers.

Those shallow artificial neural networks architecture were the foundation to develop the deep networks existent at the present moment. The main differentiation is simply the greater number of hidden layers that the latter have. Naively increasing the number of layers in shallow networks may result in poor performance in case clever initialization and training techniques aren't employed. The creation of such techniques encourages the advances in deep learning.

Fast-forwarding to the most recent research, Zilly *et al.* (2016) proposed an architecture for a Recurrent Highway Neural Network (HRN). This approach was inspired on the Gershgorin's Circle Theorem which states the square matrix property of its eigenvalues positions, namely, its spectrum. Thereby, the Long-Short Term Memory (LSTM) architecture is extended in order to allow transitions made stepwise on deeper models. The final proposition is to build RHN layers consisting of multiples Highways feedforward layers in the transitions of the recurrent states.

Another recent and interesting architecture is the Ordinary Differential Equations

(ODE) networks. This approach was inspired in the residual networks (Res-nets) which directly summed the activation of posterior layers to anterior ones as this operation resembled an Euler numeric integration. As such, a large amount of res-nets layers may be substituted by one layer followed by a ODE-solver. The result was a faster model with fewer parameters that performed almost as good as a deeper res-net with a order of magnitude more parameters (CHEN *et al.*, 2018). Another interesting point to note is that the discrete amount of residual layers were replaced by a continuous interpolation of integrated states as if it were a real valued depth with latent time variable.

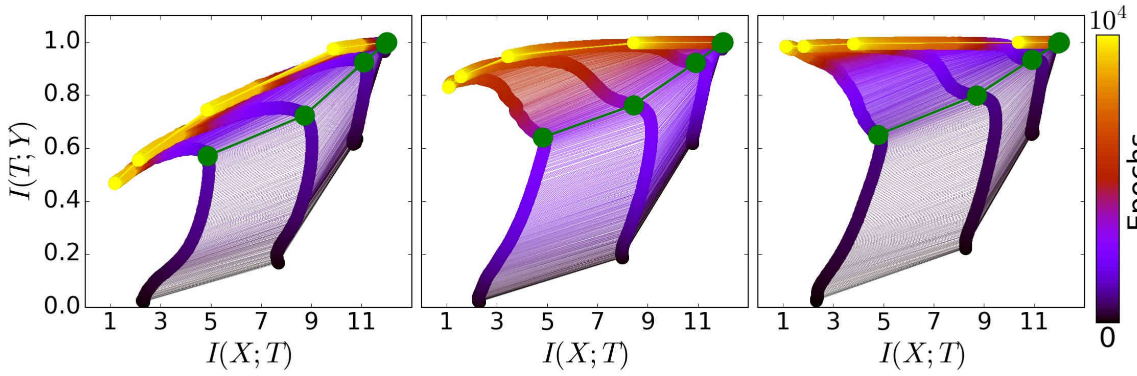


FIGURE 2.2 – A information plane depicting the training phases from bottom to up the drift phase until the green point that indicates the beginning of the diffusion phase, for three training samples with more data to the right. Source: (SHWARTZ-ZIV; TISHBY, 2017)

In contrast to proposing new architectures, Shwartz-Ziv e Tishby (2017) investigated the black-box nature that neural networks seems to exhibit due to its lack of immediate explainability. An information theory approach is presented in which the evolution of layers if analyzed in the information plane for various experiments, as shown in Figure 2.2. It was shown that there are two different phases of Stochastic Gradient Descent Optimization (SGD), namely the **drift**, characterized by gradients larger than the their standard deviations resulting in a more deterministic behavior, and **diffusion** in which there is a high stochasticity caused by the lower gradients. The main aspect of SGD stated was the "compression by diffusion" property that generates appropriates internal representations and that this learning algorithm optimizes the Information Bottleneck for each layer between precision and compression.

In this context, Arpit *et al.* (2017) observe what generalization capabilities and robustness a deep neural network may possess in face of its memorization behavior. Even though such model is able to learn from random noises, i.e., memorize, it is stated that its first optimization strategy concerns the patterns that are most basic and require the least complex representation. The usage of dropout for increase robustness in a noise dataset is also utilized without compromising its performance suggesting that the training data is important to access the level of memorization such model may exhibit as much as its architecture. This result of first fitting to the simpler pattern reinforces the results of Shwartz-Ziv e Tishby (2017) which justified this phenomenon in face of the drift behavior.

Yet trying to explain the effectiveness behind deep neural networks is Mehta e Schwab (2014) with its research that links in a one-to-one relationship a quantum physics complex systems to deep learning. This is the renormalization group (RG) physics theory, a iterative methodology that allows the extraction of various features at distinct scales from a physical system. It is suggested that the circumstances in which deep learning is effective could be explained by the employment of a generalizes renormalization group methodology. An interpretation is that the emergence of qualitative properties is a result of universal interaction at various scales, as those occurring in the natural world in a microscopic to macroscopic level, that is captured by the hierarchical displacement of neural networks architectures.

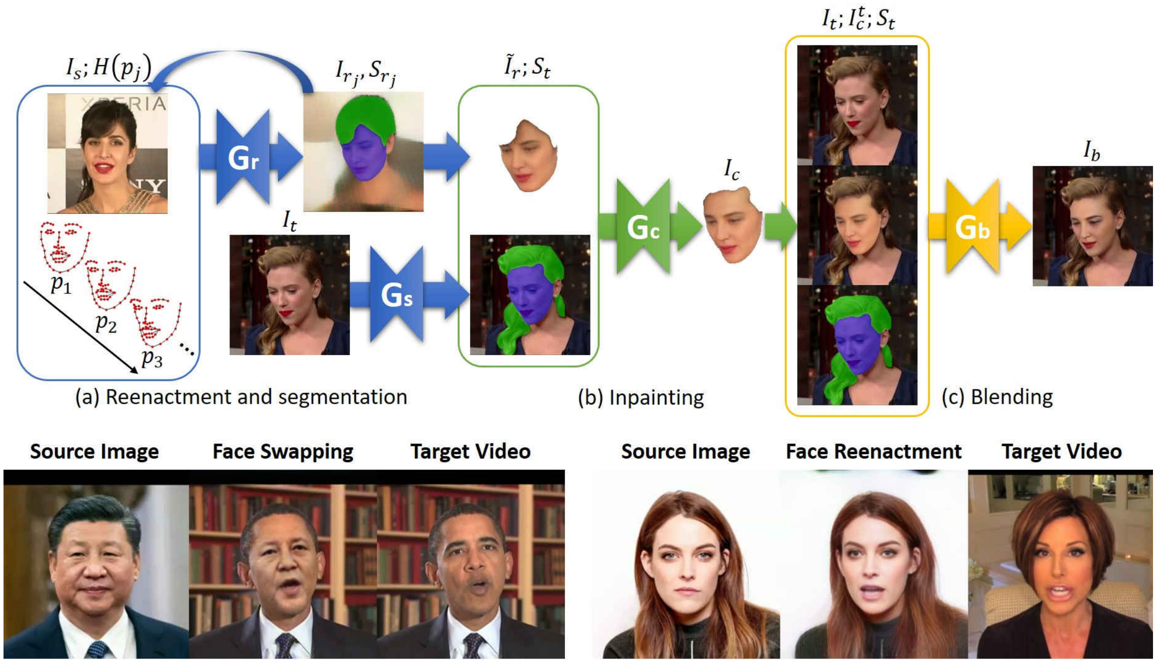


FIGURE 2.3 – Architecture overview of the Face Swapping Generative Adversarial Network and a example of its application. Source: (NIRKIN *et al.*, 2019)

Observing recent applications, Nirkin *et al.* (2019) presented a novel face transfer and swapping technique called Face Swapping Generative Adversarial Network (FSGAN), an improvement over the DeepFakes model. Being subject agnostic is the greatest advancement over the later technique, as it is applicable to untrained pairs of faces, resulting in a lower computational requirement and immediate execution. Not only limited to static images, it can also generate video output with temporal coherence by using an interpolation method of barycentric coordinates and Delauney Triangulation. The main network architecture is observed in Figure 2.3. It is noticeable the definition of two innovative loss function for this problem: the new face integration via Poisson blending loss and the progressive puppeteering provided by a stepwise consistency loss. Its approach is first constructed by the "recurrent reenactment generator" (G_r) for estimating the new face in parallel to its hair and face segmentation generator (G_s) in the target image. These encodings are then fed to the in-painting generator (G_c) responsible for predicting the final pose which is then combined by the blending generator (G_b) as shown in Figure 2.3.

Focusing in another problem that is the search for an optimal policy that an agent may utilized in order to maximize its reward in an environment, namely Reinforcement Learning (RL), there is the state-of-the-art work from Baker *et al.* (2019) in collaboration with OpenAI. In this research, a hide-and-seek goal was set in a multi-agent environment resulting in the emergence of six different strategies seen in Figure 2.4. The first and most basic strategy (a) was the coordination of movement having the seekers running towards the hiders which in turn tried to escape their visual range. The emergent tools started appearing at the second phase (b) when the hiders learned to block entrances and to construct forts around themselves giving a huge advantage over the seekers. This strategy race continues as the seeks utilizes ramps (c) but soon after are neutralized by blocking the ramps (d), followed by an exploitation of the physics engine (e) defended by blocking all objects available (f). It is important to note that all the cited behavior was learned by self-playing, not being coded by a human.

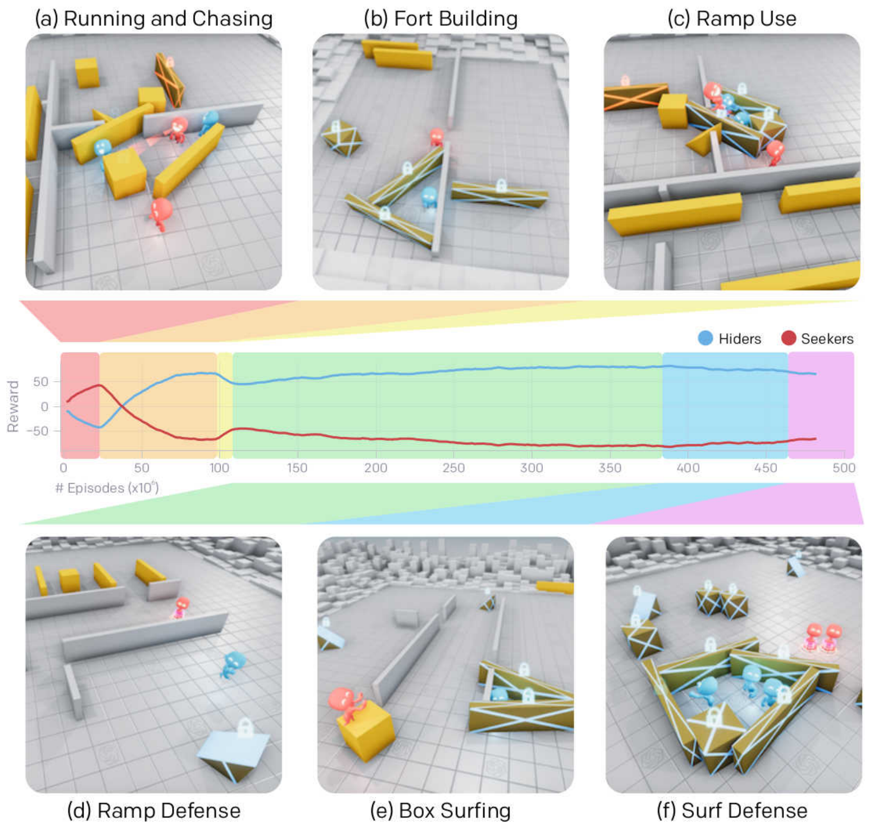


FIGURE 2.4 – Emergent strategies generated by a deep reinforcement learning model. Source: (BAKER *et al.*, 2019)

As the final application presented in this section, Payne (2019) has developed a fascinating implementation of a music generator supported also by OpenAI based on a transformer artificial neural network called Musenet. This foundation of GPT-2 is discussed in more details in the next section, but it is able to estimate the next sequence of musical notes given a prior context. In this way, the model is able to produce and blend distinct music styles, just as the DeepFakes did for faces swapping. Novel rhythms and melodies are also generated based no famous composers and on different instruments. The model captures long term structure using 24 attention heads in a 72 layered network trained on a MIDI dataset collected from the web.

2.3 Natural Language Processing

As stated by Manning e Schütze (1999), natural language processing (NLP) is the characterization of linguistic observations, identifying patterns that occurs in its usage as summed by rules or statistical inference. Two main approaches to language are prominent: the rational and the empirical methodologies. While the first claims that most knowledge from a language is determined in advance ("innate language faculty") the second argues that is the cognitive capacity that enables de construction of language ("tabula rasa"). In this context, linguists aims not only to determine the origin of natural language knowledge but to also understand how information is conveyed through language and how those structures are connected to the real world.

While not leveraging the computational capabilities or artificial neural networks, the NLP field was limited to rule based and frequency analysis approaches. Its initial ambitions stated at the Automatic Language Processing Advisory Committee (ALPAC) in the 60's was to build a translation machine, but it was ultimately abandoned in favor of basic research in the area. As described by Mitkov (2003) those subfield includes phonology, morphology, lexicography, syntax, semantics and discourse. In the distributed sense of a ANN, all those fields are interconnect indistinguishably through the network, being trained end-to-end.

There are many problems in NLP that may vary from learning language representation to understanding visual inputs and expressing that knowledge by generating text, such as in image captioning. It is important to capture the meanings from different languages not only to make effective machine translation possible but also to have a compact word embedding, enabling document understanding, summarization, sentiment analysis, named entity recognition, sensitive information detection, question answering, lexical analysis, parsing, knowledge graph, procedural generation of stories, social computing and optimization of the interface between speech and text. These research questions allow the expansion from spatial relations and humans interactions to a new level of technological advancements, breaking the language barrier in a multilingual world by using methods that easily support new languages. As the field progresses, office work that may have required hours of typing and tedious translations from structured data to natural language will be efficiently automated, freeing man-hours to more overall efficiency and productivity.

For this work, the main focus is a variation of sentiment analysis characterized not by evaluating if a sentence is negative or positive but what level of human effort is expressed in its meaning. Liu (2012) expresses this field, also known as opinion mining, as an evaluation of attitudes toward entities, namely, appraisals of linguistic attributes. The emergence of this area was mainly due to the vast data available from social networks and product reviews given in on-line shopping websites as both platforms desired to create a better model of its costumers. Although there are different levels on analysis, those being document, sentence and entity or aspect, the main scale of this research is focused on the mid-document sentence level, namely, short documents or long sentences. There are also other aspects studied such as regular to comparative sentences, explicit or implicit opinions, rational or emotional evaluations and idiosyncratic attributes to the writer.

Moving to more recent approach to NLP, Deng e Liu (2018) considers deep learning as the most fruitful methodology to tackle these problems given enough data and processing power, the so called third wave of natural language processing. It has provided a more solid background which resurrects the empiric ideals of learning from scratch like a human child would do. Five main points for advancements in NLP were proposed: firstly meta-learning, commonly known as learning-to-learn, where previous knowledge is utilized with a reduced new data in order to succeed in a novel task (approaching the AGI); leveraging multitask learning as well as multimodal in sparse cost systems from simultaneous tasks; developing foundations to integrate symbolic computation to ANNs; advancing the studies in episodic memory models as seen on humans; and finally, better paradigms for generative and unsupervised learning that allows a more effective utilization of available information.

Radford *et al.* (2018) work has surprised an amount of NLP researchers for its capacity in generating cohesive and consistent stories while also being able to perform relatively well on a wide range of NLP tasks such as reading comprehension, translation, summarization and question answering. It was demonstrated that these task can be learned with no explicit supervision given a sufficient amount of data which, in the paper, was about 40GB of raw text. The final architecture had over one and a half billion parameters and was called GPT-2 and it is based on Transformers Neural Networks (VASWANI *et al.*, 2017). The following excerpt is a textual generation from this model, given a small seed text about the Muslim rule in Spain. It is also remarkable that the code and parameters for this model were not fully released by the time this dissertation was written, as the perceived dangers and its implications of such powerful text generator had led to a slow and gradual release of the work.

> The Reconquista (literally, The return to the country) of 1092 and 1095 saw the establishment in the Iberian peninsula of a united kingdom of Castile and Aragon in which a Muslim dynasty would exist. This was the last time in Europe where Muslim rule would be considered ”native” and the last occasion where Christian-inflamed and organized conflicts would make themselves known in Europe. The reign of Muslim Moors in Spain is often re- ferred to, especially among non-Muslims, as the Conquest of Granada and Spain is sometimes referred to as Spain in Granada. The last years of Muslim rule were followed by the Spanish Reconquista where the Muslims were driven or compelled out of the Iberian Peninsula with the victory of Ferdinand IX. in the Battle of Poitiers in March 1189 and the conquest of Jerusalem in 1187 by Saladin.
>
> (RADFORD *et al.*, 2018, p. 15)

Another important approach from (GRAVE *et al.*, 2018) who utilizes word embeddings to leverage and identify 157 different languages. The data was based on Wikipedia articles as they had many languages available even though some had much more data than others such as English. It is remarkable that those word embeddings are employed in state-of-the-art models as initial layers in order to reduce the input dimensionality and even though they were initially trained in an unsupervised way it may be adjusted in the supervised step of learning. These 157 word embeddings are also used as a mean to identify the language from a text and to make analogies between them.

2.4 Story points estimation

Li *et al.* (2018) reviewed the recent applications of deep learning in the Software Engineering (SE) field. It was stated that those neural network architectures were integrated over forty SE tasks including requirements, design, development, testing, maintenance and management activities from numerous industrial practitioners besides academic researchers. Most of those assessed papers were found to utilize standard DL models, with initially little variation from the already existing but in novel applications. Data collection has been found to be a great challenge in some cases that requires specific frameworks such as agile ones.

In the context of software engineering, Choetkiertikul *et al.* (2017) and Choetkiertikul *et al.* (2018) have developed deep learning models for predicting the delay of programming implementations and for estimating the delivery capability in agile frameworks.

Satapathy e Rath (2017) used story points in order to make an empirical survey of machine learning techniques in effort estimation tasks in an agile framework context.

The results from story points evaluated by the team was used as a input to the model which could predict the normalized effort (in man-hours or man-months) to complete such task. Although shallow neural networks were also compared to other methods, the authors did not employ DL architectures, limiting the methodological scope to classical ML approaches.

On the other hand, Panda *et al.* (2015) employed ANNs to also propose a solution to the same problem of estimating effort (in man-hours) from actual story points. Its work was also limited by the availability of data with both attributes, constraining the developed network's size. The less usual architectures of Group Method of Data Handling (GMDH) a self-organizing model, Probabilistic Neural Networks (PNN) and four layered General Regression Neural Networks (GRNN) were employed and compared.

Porru *et al.* (2016) was the first author to publish a model that estimates story points from user stories given its textual description. Nevertheless, neither deep learning nor shallow neural networks were employed in this work which in turn utilized the Term Frequency Inverse Document Frequency transformation (TF-IDF). The dataset was mainly based on open source projects from different organizations whose software management happened on the Jira platform. Therefore, given its performance of a Mean Magnitude of Relative Error (MMRE) that varied from 0.16 to 0.61, it was demonstrated the feasibility of using a machine learner to assist humans making effort estimations.

The most remarkable and recent work found in this specific problem was from Choetkiertikul *et al.* (2019). The Figure 2.5 shows the architecture developed by those researchers for this task of using natural language as input in order to estimate story points. Their results were significantly better than random guessing and average estimation, having also more precision than Porru *et al.* (2016) in most cases. Thereby it sets a milestone in which this dissertation tries to improve upon by using another deep neural network architecture and more daring techniques while constrained by the reduced dataset size.

The first layer of the referred model is an Word Embedding that is trained in an unsupervised manner using user stories that did not have an user point assigned to them in the dataset. The second layer is composed from Long-Short Term Memory (LSTM) cells whose time distributed output is pooled using simply the mean sample of its values. A Recurrent Highway Network is then employed, characterizing more layers to a deeper network and refining the estimation that is finally output by a regression as noticed in Figure 2.5. In the supervised learning step, the model is trained end-to-end using root mean squared backprogation gradient descent (RMSProp) with mini-batches, implying in a stochastic nature of its training also reinforced by the random initializations in the weights.

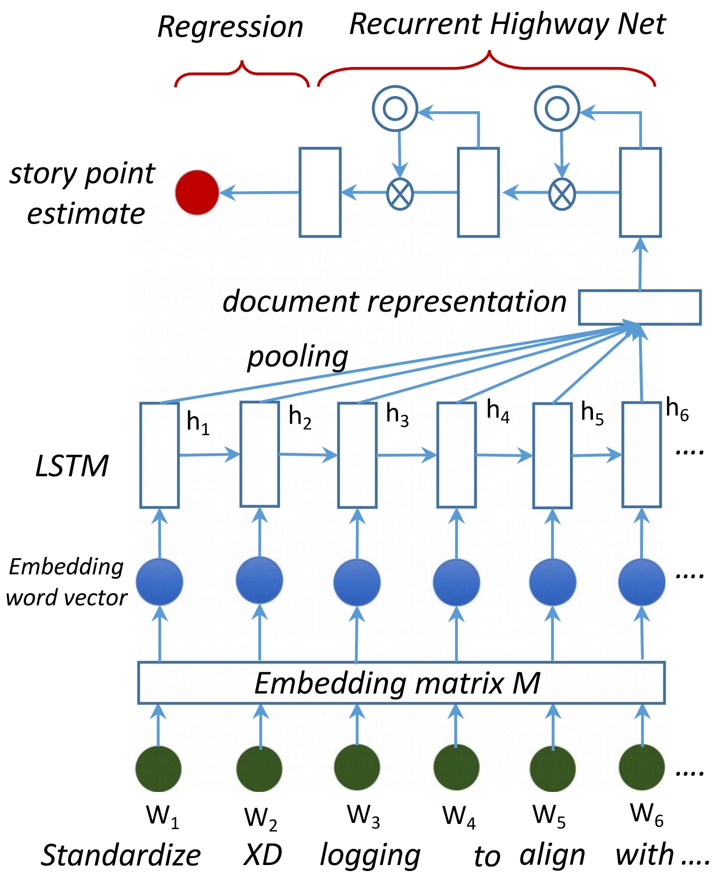


FIGURE 2.5 – The Deep-SE architecture proposed by Choetkiertikul *et al.* (2019) for estimating story points given its textual description. Source: (CHOETKIERTIKUL *et al.*, 2019)

2.5 AI safety

Another important topic in AI research concern the guarantees for its safe utilization, namely the potential impacts that artificial intelligence may have on society as a whole. From a technical perspective, this impact is studied as the possibles accidents and misbehaviors that may occur from poorly designed AI models (AMODEI *et al.*, 2016). The main causes for concern are the definition of a wrong objective function, reward hacking, lack of scalable supervision and hazardous exploration while training. Several possible solutions have been proposed to mitigate and access those risks, including model lookahead, trip wires, supervised reward learning, distant supervision, simulated exploration, human oversight, impact regularizers and other tools.

Leike *et al.* (2017) proposed a GridWorld to tackle those problems. Several scenarios were constructed in such a world with two reward functions: one which the agent tried to maximize and another not known to it but that evaluated its real performance. The problems of safe interruptibility, avoiding side effects, absent supervisor, reward gaming, self-modification, distributional shift, robustness to adversaries and safe exploration were analyzed have each one of those its unique environment.

In an application to natural language processing (NLP), there are AI safety techniques that aim to filter racial, gender, cultural biases and other prejudices from the language models trained in web based content. As there exists many websites which hosts offensive texts that may indiscriminately be used for trained, the machine learner eventually picks some of this inappropriate information, causing a risk for its future operations. In the context of word embeddings, a renormalization trick may be employed by subtracting the undesired components from the vectors and scaling it to maintain the same norm, although the model's accuracy may be degraded.

3 Deep Learning

The development of Artificial Intelligence was mainly in the area of Machine Learning (ML), specifically using Deep Neural Networks (DNN). In contrast to expert systems that use inference from a knowledge base by applying first-order logic rules, neural networks are characterized as connectionist artificial intelligence, building an internal representation of input data. Specially, deep learning is characterized by computing such representations in terms of the previous ones, composing a function of greater complexity that can obtain better results (GOODFELLOW *et al.*, 2016). This chapter is a expansion of the undergraduation work written in Portuguese developed simultaneously to this research (MELO, 2019).

In this context, neural networks can be interpreted not only as universal approximators of functions, but also as transformations applied in the input data space capable of progressively separating representations and finding latent variables (MELO *et al.*, 2019, no prelo). In other words each layer of a neural networks may be interpreted as successive applications of functions in which there must be agreement of data representation between each layer as show in Equation 3.1.

$$\text{activation} = \sigma\left(\text{bias} + \sum_{i=1}^{N} w_i x_i\right) \tag{3.1}$$

The basic processing unit of artificial neural networks is the artificial neuron, which canonically performs a linear combination of its inputs weighted by their synaptic weights followed by an activation function (MELO, 2019). Its mathematical formulation can be written as a matrix multiplication according to Equation 3.2.

$$\text{activation} = \sigma\left(\begin{bmatrix} w_0 & w_1 & w_2 & w_3 & w_4 & w_5 & \dots & w_n \end{bmatrix}\begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_n \end{bmatrix}\right) \tag{3.2}$$

Although inspired by biology, the artificial neural network models and its learning techniques, in reality, is an optimization of parameters, are distant from natural processes and the neuronal organization of biological networks, and their similarities are widely debated (MELO *et al.*, 2019, no prelo). Matrix algebra and multivariate calculus are the main tools for the construction, analysis and optimization of neural networks, and the biological analogy is convenient for its interpretation (NIELSEN, 2015). Thus, its mathematical formulation for multiple neurons, characterizing a layer of the neural network, can be observed in Equation 3.3.

$$A = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ \vdots \\ a_d \end{bmatrix} = \sigma\left(\begin{bmatrix} w_{10} & w_{11} & w_{12} & \dots & w_{1n} \\ w_{20} & w_{21} & w_{22} & \dots & w_{2n} \\ w_{30} & w_{31} & w_{32} & \dots & w_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{d0} & w_{d1} & w_{d2} & \dots & w_{dn} \end{bmatrix}\begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_n \end{bmatrix}\right) \tag{3.3}$$

Thus, the design of a neural network represents major engineering challenges, the estimation of control hyperparameters and the architecture and topology of the network itself. Like aeronautical engineering in the early twentieth century with vehicles heavier than air, its parameters were estimated by empirical equations, due to the complexity and intractability of analytical and even numerical solutions, the development of deep learning techniques also finds in its infancy (MELO *et al.*, 2019, no prelo), with great advances, although based on empirical methodology (LECUN *et al.*, 2015). A numerical example of your application is represented below in Equation 3.4.

$$A^{(1)} = W^{(1)}X = \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \\ a_2^{(1)} \end{bmatrix} = ReLU \left(\begin{bmatrix} w_{10}^{(1)} & w_{11}^{(1)} & w_{12}^{(1)} \\ w_{20}^{(1)} & w_{21}^{(1)} & w_{22}^{(1)} \\ w_{30}^{(1)} & w_{31}^{(1)} & w_{32}^{(1)} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \right) \tag{3.4}$$

3.1 Shallow Neural Networks

The basic computing unit of a neural network is the neuron, terms that were inspired by biology. A linear combination of the neuron inputs occurs whose output signal is modulated by a nonlinear activation function. For a neuron, the input signals are represented by a vector of the anterior layer output signals, as observed in Figure 3.1. The mathematical representation is indicated by Equation 3.2, where a indicates the resulting activation of the neuron, σ is the activation function, x_i is the i-th input, w_i is the weight associated with the i-th input, and b is the bias of the neuron bias (MELO, 2019). Already in terms of layers, there is a matrix multiplication of the activations of the previous layers, as exposed by Equation 3.5.

$$A^{(2)} = W^{(2)}A^{(1)} \begin{bmatrix} a_1^{(2)} \end{bmatrix} = ReLU \left(\begin{bmatrix} w_{10}^{(2)} & w_{11}^{(2)} & w_{12}^{(2)} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \right) \tag{3.5}$$

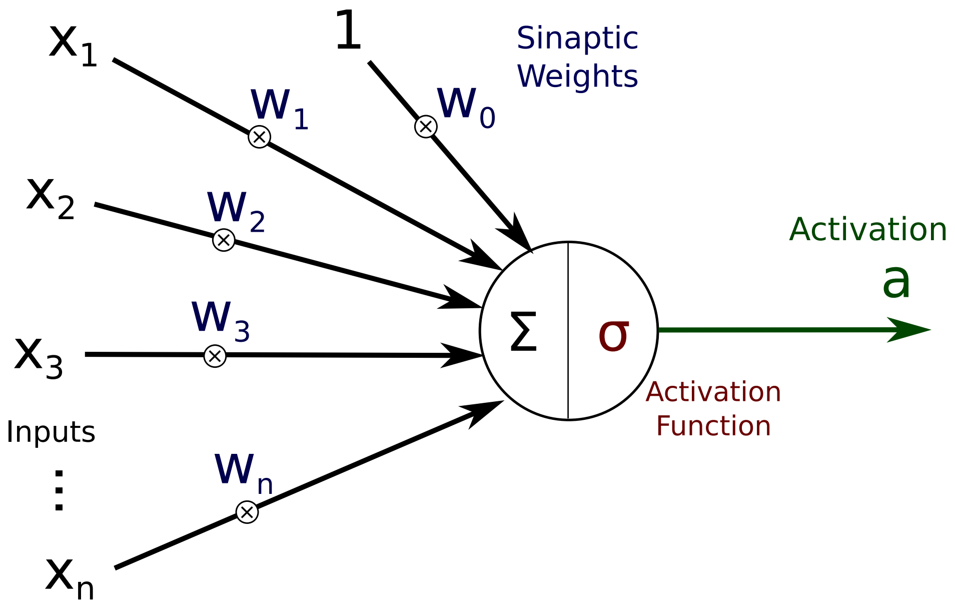


FIGURE 3.1 – Artificial neuron modeled as a linear combination. Adapted: (MELO, 2019)

The complete representation is given by the matrix that represents the weights of each layer. Thus, the computation of a $L-1$ index layer with 3 neurons to another of a L index index with 4 neurons can be written by the Equation 3.8. In this mathematical formulation, $A^{(N)}$ represents the activations of the Nth layer neuron, $w_{ij}^{(N)}$ represents the synaptic weight between the neurons i and j da layer N, $b_i^{(N)}$ represents the polarization threshold of the n-th layer N neuron and σ is the activation function that must be applied to each element of the matrix (MELO *et al.*, 2019, no prelo).

It is important to note that the computational power of the neural network occurs through the nonlinearity of the activation function since the linear combination of linear functions would also be a linear function, which would restrict the neural network to linear regressions only. An abstraction of this phenomenon is to conceive nonlinearity as an IF-ELSE rule of imperative programming, something that is clear in the ReLU Rectified Linear Unit activation functions (MELO, 2019), which have a signal check, that is, defined by Equation 3.35. Another commonly used activation function is sigmoid, defined by 3.31, which maps values between 0 and 1. Finally, there is a graphical representation of an artificial neural network in Figure 3.2.

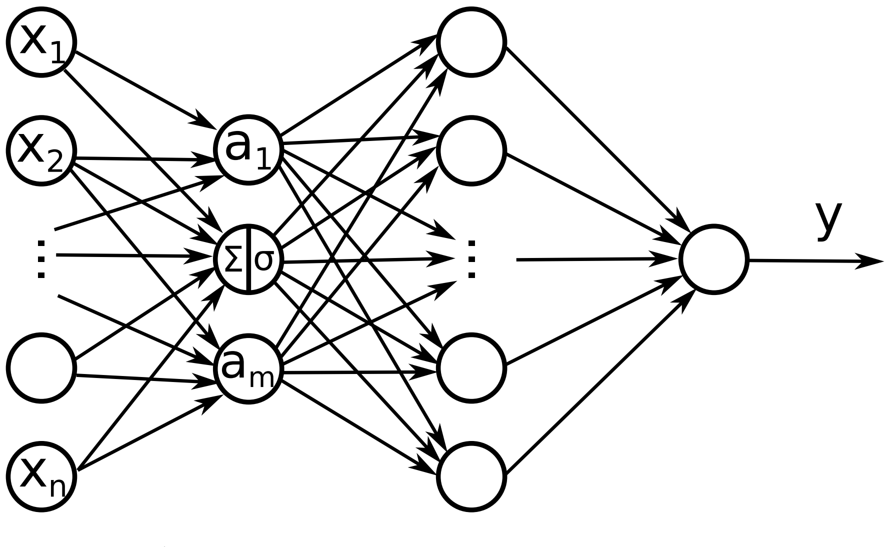


FIGURE 3.2 – Artificial neural network representation and its basic unit.

The objective function defines the cost, how far the result is and whose purpose of optimization is that it should be minimized. This characterizes what is called supervised learning, ie when the agent is trained with input values already mapped to known output values. Thus the cost function (also called loss, cost, objective) has as input values the weights and thresholds that define the network and has as fixed parameters the training cases. The equation 3.6 defines a cost representation as the mean square error, that is, the mean between the squares of the differences between the expected output value and the predicted output value $a^{(L)}$ in the set of N training cases (MELO, 2019). Cost thus becomes a function of the neural model W and B parameters, which represent the neural weights and biases. Again, the result of a layer can be exposed in the Equation 3.7.

$$C = (a^{(L)} - y)^2 \tag{3.6}$$

$$a^{(L)} = \sigma(z^{(L)}) \tag{3.7}$$

Thus, learning simply consists of decreasing this cost function. The Gradient Descent method is first-order in contrast to Levenberg-Marquardt, Quasi-Newton, and other algorithms that, while converging on fewer iterations by calculating the second derivative, take more time and spend the square of space, which makes intractable optimization for deep networks (MELO *et al.*, 2019, no prelo). In the case of only one layer, its behavior can be expressed by Equation 3.8.

$$z^{(L)} = W^{(L)} A^{(L-1)} \tag{3.8}$$

Mini-batch allows for better treatability, as well as giving a stochastic nature to training, which contributes to the optimization not being trapped in minimal locations. There are heuristics that improve your performance, with adaptive learning rates and calculating the amount of movement of the function. In addition to SGD Stochastic Gradient Descent there are other heuristics such as RMSProp, SGD-Nesterov Momentum, AdaDelta, AdaGrad and Adam. Equation 3.9 indicates the mathematical formulation of the Descending Gradient, in which the weights are updated proportionally to the cost gradient (MELO, 2019).

$$\Delta W = -\alpha \nabla_W C \tag{3.9}$$

Thus, for training it is necessary to find which partial derivative of the cost function with respect to the weight to be maintained. This question is informally expressed by Equation 3.10, where Λ is the unknown variable.

$$\frac{\partial C}{\partial w_{ij}^{(L)}} = \Lambda \tag{3.10}$$

System optimization, that is, learning, takes place through a peculiar kind of gradient descent: backpropagation. It is a method that calculates the partial derivatives of the cost function with respect to the synaptic weights. The advantage of backpropagation is the reduction in the amount of computation and space required, since the change in weight of a layer is calculated only as a function of the previous layer, which accumulates the recursive calculations as indicated in Equation. 3.11.

$$\frac{\partial C}{\partial Z^{(K-1)}} = \frac{\partial C}{\partial Z^{(K)}} \frac{\partial Z^{(K)}}{\partial A^{(K-1)}} \frac{\partial A^{(K-1)}}{\partial Z^{(K-1)}} \tag{3.11}$$

3.2 Convolutional Neural Networks (CNN)

The convolutional neural networks is a particular feedforward neural network in which the synaptic weights are interpreted as kernels for co-convolutions operations. In practice, it is widely used for image related task by learning filters.

This operation is effectively a weigh sharing technique and also a sparse architecture as the number of parameters is order of magnitude lower compared to a fully connected neural network. Besides bi-dimensional kernels that make filters for images, there are one-dimensional kernels used in another architecture called "Transformer" employed in natural language processing, as shown in the literature review chapter for GPT-2.

The backpropagation occurs similarly as the gradients linearly combined and the weights are updated accordingly to its average and combination between activations. These operations are more efficiently performed by a GPU, which may also leverage its texture's hardware.

3.3 Recurrent Neural Networks (RNN)

The Recurrent Neural Network (RNN) model provides an architecture that encapsulates temporal information. This is suitable for hydrological time series prediction as it is based on the previous values of the series depending on the number of memory components. There are feedback loops in the neurons of an RNN, which can generically send signals in any direction to and from all layers (LECUN *et al.*, 2015). Thus, the network output depends not only on the external inputs it receives, but also on the state of the network in the previous period of time, observed in Equation 3.12.

$$y = f(X) = \sigma(w_1 x_1 + \cdots + w_n x_n + b + \omega y(t-1)) \tag{3.12}$$

The Back-Propagation Through Time algorithm considers partial derivatives not only related to input at the same instant of time as output, but also to all past derivatives, as represented by the Equation 3.13 (HOCHREITER; SCHMIDHUBER, 1997) and also in Figure 3.3.

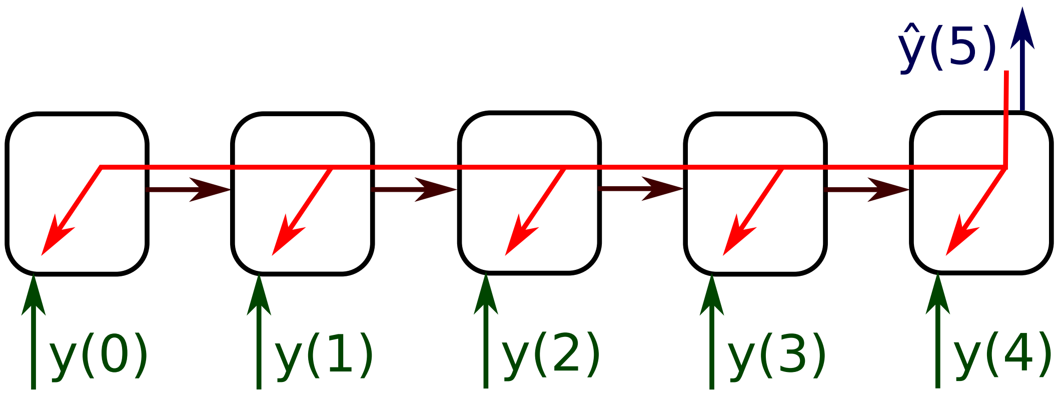


FIGURE 3.3 – Design of a neuron in a recurrent network

$$\frac{\partial \ni_{\ni} (t-q)}{\partial \ni_{u} (t)} = \sum_{l_1=1}^{n} \cdots \sum_{l_{q-1=1}^{n}} \prod_{m=1}^{q} f'_{l_m}(net_{l_m}(t-m))w_{l_m l_{m-1}} \tag{3.13}$$

The Figure 3.4 illustrates how the recurring model can be unfolded in the temporal dimension. Thus, the recurrent network can be characterized as a deep network, but with equal weights between the temporal layers. (SCHMIDHUBER, 2015).

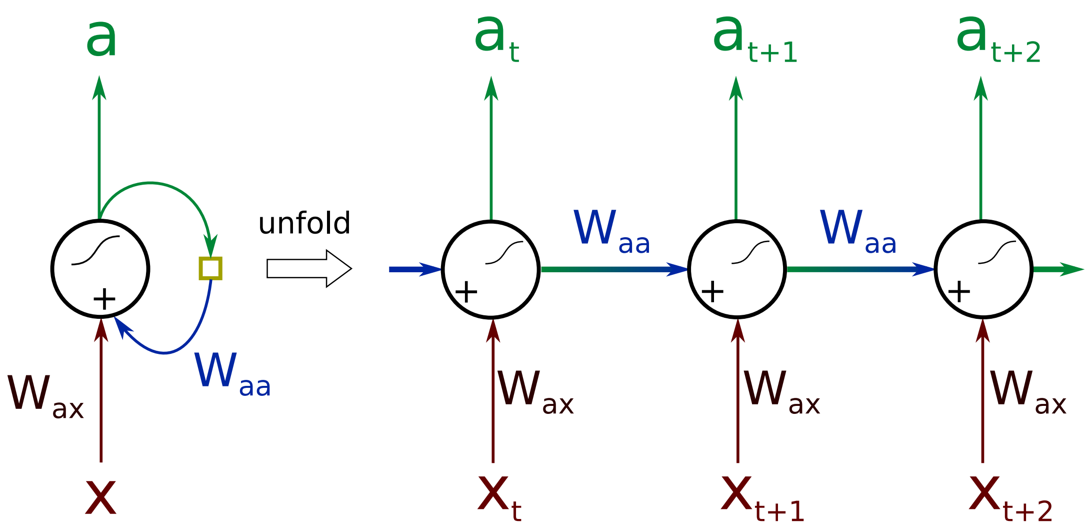


FIGURE 3.4 – Design of a neuron in a recurrent network. Adapted: (MELO *et al.*, 2019, no prelo)

Thus, as specific architectures of the recurring model, we can cite LSTM and GRU as their most notorious representatives. These special recurrent network models were designed to try to model a longer-term dependency, since the training and operation of the simplest model could not properly adjust the initial weights as the gradient backpropagation signal decreased with each layer. .

3.3.1 Long-Short Term Memory (LSTM)

LSTMs were initially proposed by Hochreiter e Schmidhuber (1997) as a first solution to their long-term dependencies. The essence of LSTM operation is in an additional state which is the cell state, which is virtually unchanged in the unit iteration process. There are three different types of gates: input gate, output gate and forget gate. Its representation can be seen in Figure 3.5.

The forget gate is responsible for erasing the cell state, for this it uses a sigmoid activation function (its image is between 0 and 1) whose output directly multiplies the cell state, which will be updated to a value less than or equal to previous value. Already the input port is responsible for summing the current value of the input in the cell state, making use of a sigmoid activation function and a hyperbolic tangent.

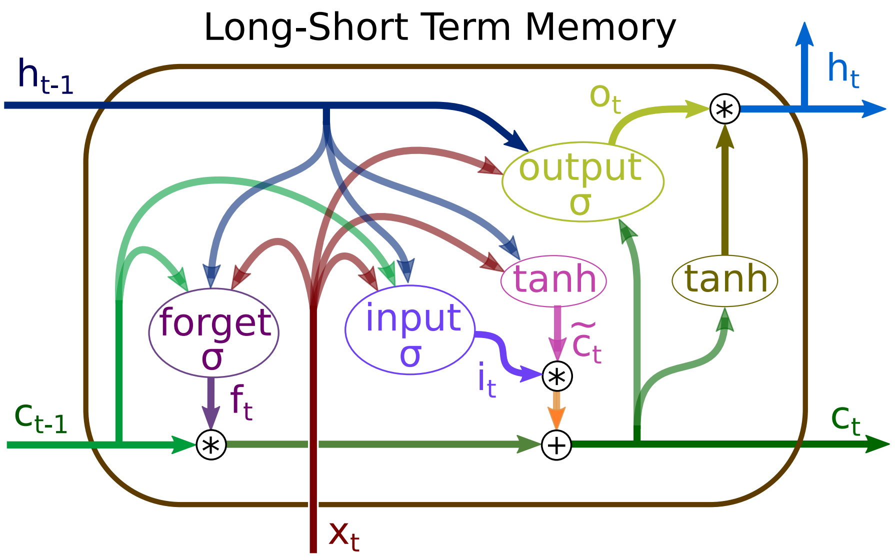


FIGURE 3.5 – Long-Short Term Memory (LSTM) cell inner architecture. Adapted: (MELO *et al.*, 2019, no prelo)

In the last step of LSTM characterization, there is the output port, which represents the influence of cell state and input values on the unit output value. In relation to the cell state, a sigmoid activation function is used, while in relation to the input, there is a hyperbolic tangent.

In mathematical terms, the following equations model the behavior of an LSTM cell, which can be interpreted as a set of neurons with a very characteristic topology, defined by the set of states represented by the equation 3.14 (HOCHREITER; SCHMIDHUBER, 1997).

$$\begin{aligned}
h_t^j &= o_t^j \tanh(c_t^j) \\
o_t^j &= \sigma(W_o x_t + U_o h_{t-1} + V_o c_t) \\
c_t^j &= f_t^j c_{t-1}^j + i_t^j \tilde{c}_t^j \\
\tilde{c}_t^j &= \tanh(W_c x_t + U_c ht - 1) \\
f_t^j &= \sigma(W_f x_t + U_f h_{t-1} + V_f c_{t-1}) \\
i_t^j &= \sigma(W_i x_t + U_i h_{t-1} + V_i c_{t-1})
\end{aligned} \tag{3.14}$$

3.3.2 Gated Recurrent Unit (GRU)

Similarly, the Gated Recurrent Unit (GRU) has been proposed as an improvement of LSTM computational efficiency through simplifications, although with no performance losses (CHUNG *et al.*, 2014). The GRU unit has only two ports and does not display cell status. It is the reset gate and update gate that characterize this model, as shown in Figure 3.6.

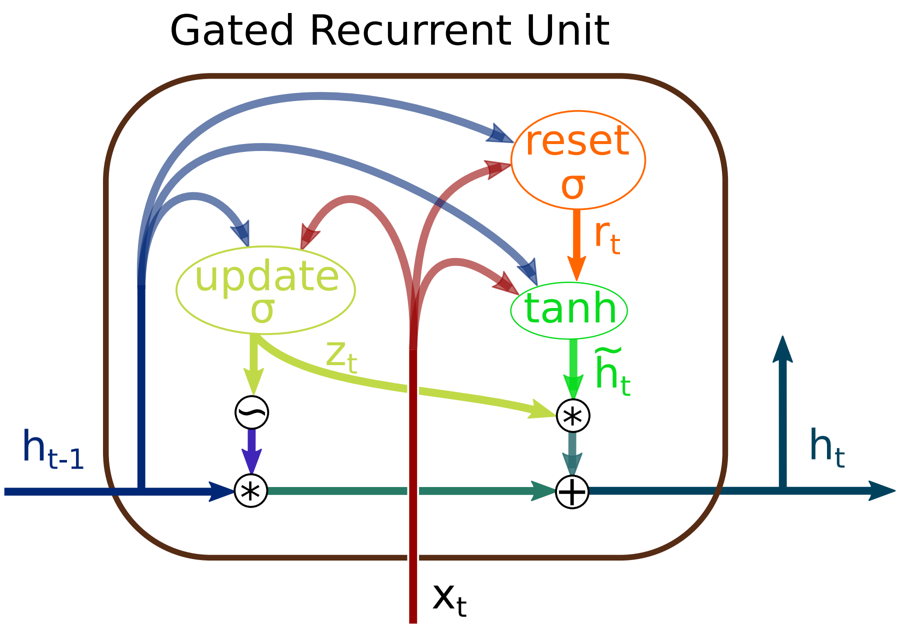


FIGURE 3.6 – Gated Recurrent Unit (GRU) inner architecture. Adapted: (MELO *et al.*, 2019, no prelo)

The upgrade port plays the role simultaneously of what would be the forgetting and entry ports in the LSTM. The intuition is that forgetting a previous state should also include a new state, and that is in just one step. The restart door, on the other hand, also resembles the oblivion door, but it has a hyperbolic tangent activation function. This way, there is no need for an output port, since there is no cell state, just its own output.

The definitions of their states are expressed in Equation 3.15 (CHUNG *et al.*, 2014).

$$\begin{aligned} h_t^j &= (1 - z_t^j)h_{t-1}^j + z_t^j \tilde{h}_t^j \\ z_t^j &= \sigma(W_z x_t + U_z h_{t-1}) \\ \tilde{h}_t^j &= \tanh(W x_t + U(r_t \odot ht - 1) \\ r_t^j &= \sigma(W_r x_t + U_r h_{t-1}) \end{aligned} \tag{3.15}$$

3.4 Backpropagation

The training of an artificial neural network is characterized by the minimization of an objective function. For commonly used architectures, synaptic weights are the parameters to be optimized in the training process. Thus, using iterative methods, one must determine whether a given weight should be positively or negatively disturbed in order to reduce the model error.

Performing such variations for each of the synaptic weights, reevaluating network performance at each step would be impractical for a large number of parameters and training set. In this context, the backpropagation method solves this problem by calculating such updates for all weights in only one evaluation of the network.

Thus, given the calculation of the activation of the neurons in the direct propagation step in order to calculate the output of the network, the output error is first calculated and then propagated in the reverse direction of this error signal. . The synaptic weights are updated to minimize the error signal of the next iteration, taking into account the activation of the neuron connected in the anterior layer and the error signal of the neuron in the next layer.

A mathematical demonstration of the formulas involved in backpropagation is given below (MELO, 2019).

First, we calculate the partial derivative of the cost function with respect to the activation of the last layer neuron, as indicated by Equation 3.16.

$$\frac{\partial C}{\partial a^{(L)}} = 2(a^{(L)} - y) \tag{3.16}$$

Then, using the chain rule, the partial derivative of the cost function with respect to the synaptic weight is calculated in view of the previous derivative. Thus, the equation 3.17 represents such a formulation.

$$\frac{\partial C}{\partial w_{ij}^{(L)}} = \frac{\partial C}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial w_{ij}^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \frac{da^{(L)}}{dz^{(L)}} \frac{\partial z^{(L)}}{\partial w_{ij}^{(L)}} \tag{3.17}$$

Again, performing one more step of the chain rule and one more substitution of the previous results, we present the Equation 3.18.

$$\frac{\partial C}{\partial w_{ij}^{(L)}} = \frac{\partial C}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial w_{ij}^{(L)}} = \frac{\partial C}{\partial z^{(L)}} a_j^{(L-1)} \tag{3.18}$$

Finally, we have the expected result for the synaptic weight gradient of the last layer. As a result of replacing the previous equations we get to Equation 3.19.

$$\frac{\partial C}{\partial w_{ij}^{(L)}} = 2(a^{(L)} - y)\sigma'(z^{(L)})a_j^{(L-1)} \tag{3.19}$$

As a prelude to the generalization of this result to be applied to earlier layers of the artificial neural network, the partial derivative of the cost with respect to the linear combination of the inputs of the last layer, as expressed by the Equation 3.20, is calculated.

$$\frac{\partial C}{\partial z^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \frac{da^{(L)}}{dz^{(L)}} \tag{3.20}$$

Thus, in order to obtain a formulation that can be recursively applied, the following question is asked: how to calculate the partial derivative of the cost function with respect to the linear combinations of the previous layer inputs. In mathematical terms, such notation is realized in the Equation3.21.

$$\frac{\partial C}{\partial z^{(L-1)}} = \Lambda \tag{3.21}$$

In view of Equation 3.22, we derive it in order to apply its result in the next step.

$$a^{(L-1)} = \sigma(z^{(L-1)}) \tag{3.22}$$

Because the activation function is real, having only one input and one-dimensional output, instead of partial derivatives we have the total derivative, according to the Equation3.23

$$\frac{da^{(L)}}{dz^{(L)}} = \sigma'(z^{(L)}) \tag{3.23}$$

Again, using the definition of the linear combination of the back layer inputs, given in the Equation 3.24, we have a function of the activations of the previous layers.

$$z^{(L)} = W^{(L)}A^{(L-1)} \tag{3.24}$$

Thus, the partial derivative of the linear combinations of the posterior layer with respect to the activations of the anterior layers is simply the matrix of synaptic weights, expressed by the Equation 3.25.

$$\frac{\partial z^{(L)}}{\partial A^{(L-1)}} = W^{(L)} \tag{3.25}$$

At this juncture, one of the last terms to replace in the expansion of the chain rule is the partial derivative of the linear combination of inputs by a synaptic weight of index ij, given by the Equation 3.26.

$$\frac{\partial z^{(L)}}{\partial w_{ij}^{(L)}} = a_j^{(L-1)} \tag{3.26}$$

This result can also be expressed in matrix form, characterized in Equation 3.27. It is important to remember that the activation function derivative is applied to each element of the matrix.

$$\frac{dA^{(L-1)}}{dZ^{(L-1)}} = \sigma'(Z^{(L-1)}) \tag{3.27}$$

Therefore, by applying the string rule expressed in Equation 3.28, it is sufficient to perform the substitutions of the terms previously found.

$$\frac{\partial C}{\partial z^{(L-1)}} = \frac{\partial C}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial A^{(L-1)}} \frac{dA^{(L-1)}}{dZ^{(L-1)}} \tag{3.28}$$

Finally, a general result is demonstrated and it can be applied recursively: given the error signal of a later layer it is possible to calculate the error signal of its previous layer. The mathematical expression that summarizes this result is given in Equation 3.29. Having the results of all error signals and all activations, to calculate the error gradient in relation to the synaptic weights simply make your product.

$$\frac{\partial C}{\partial z^{(L-1)}} = \left(W^{(L)}\right)^T \frac{\partial C}{\partial z^{(L)}} \odot \sigma'(Z^{(L-1)}) \tag{3.29}$$

3.5 Activation Functions

Nonlinear activation functions enable each layer to perform nonlinear transformations on the vector space represented by the activations of the previous layer, distorting the space and creating a representation that separates classes better. Thus, it is fundamental to choose the nonlinearities that can best represent the data relations.

3.5.1 Heaviside Step Function

The step activation function was one of the first nonlinear activation functions to be used. However, since its derivative is zero throughout its domain, unless it has a discontinuity origin, it cannot be used in gradient descent training. Point 0 can be set to 1 or 0.5. Equation 3.30 represents a definition used in which point 0 presents uniform convergence for the sigmoid function in the case where parameter a tends to infinity. The Figure 3.7 graphically represents this function.

$$U(z) = \begin{cases} 0, & z < 0 \\ \frac{1}{2}, & z = 0 \\ 1, & z > 0 \end{cases} \tag{3.30}$$

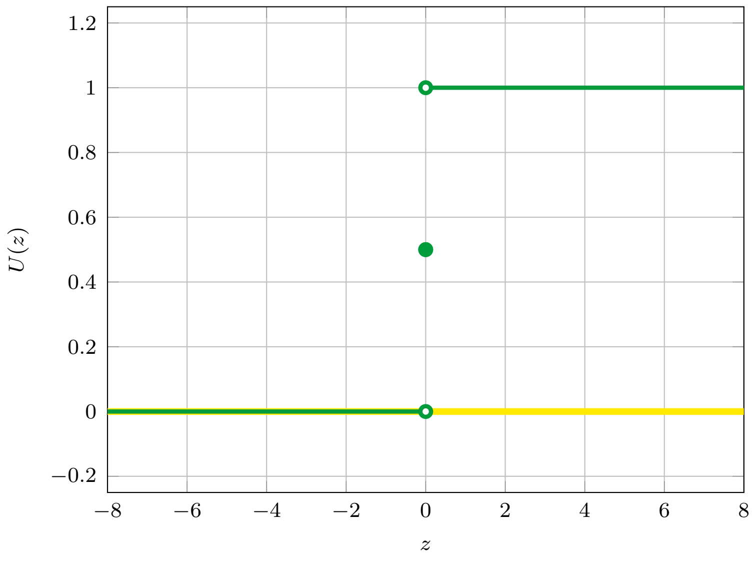


FIGURE 3.7 – Heaviside step function and its derivative.

3.5.2 Sigmoid Function

The sigmoid function is a continuous function that resembles the step function in that it is limited between 0 and 1, tending to these values for a sufficiently negative or positive input respectively. This function that has no discontinuities in its derivatives. Its mathematical formulation is represented by the equation 3.31, which is graphically observed in Figure 3.8.

$$\sigma(z) = \frac{1}{1 + e^{-az}} \tag{3.31}$$

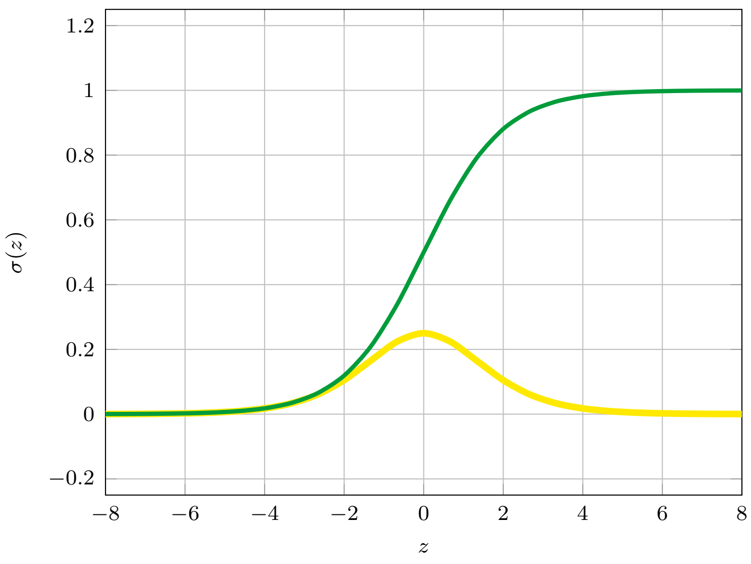


FIGURE 3.8 – Sigmoid activation function and its derivative.

3.5.3 Hyperbolic Tangent

The hyperbolic tangent function is similar to the sigmoid but with values between -1 and 1, as shown in Figure 3.9. Thus it can be interpreted as double the sigmoid function minus 1, a property observed in Equation 3.32.

$$\tanh(z) = \frac{1 - e^{-2z}}{1 + e^{-2z}} \tag{3.32}$$

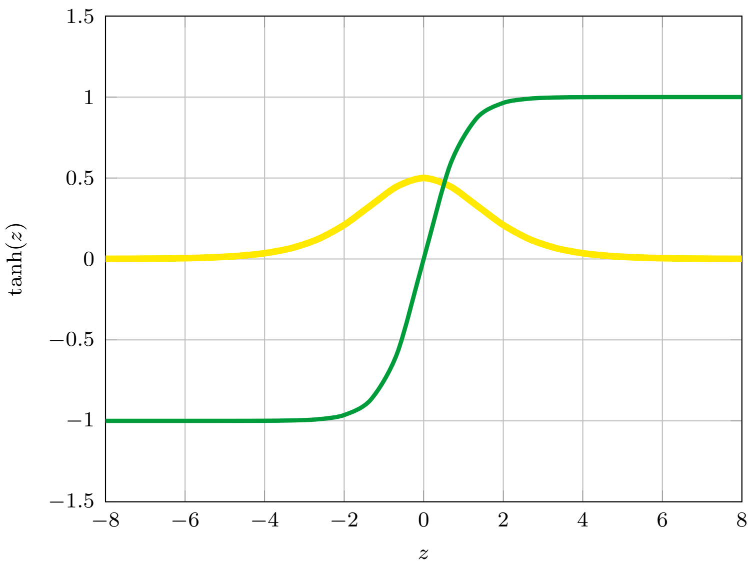


FIGURE 3.9 – Hyperbolic tangent activation function and its derivative.

3.5.4 Linear Activation

In contrast to nonlinear activation functions, it is necessary to have more parsimony in their use since the linear combination of linear functions is also a linear function, which generates no capacity gain with more than one linear layer. Therefore, such activation can be understood as a neutral element to the composition of functions, as if there were simply no activation. Its mathematical formulation is given by the identity function exposed in Equation 3.33 and whose polarization graph can be observed in Figure 3.10.

$$f(z) = z \tag{3.33}$$

The derivative of this linear function is simply the unitary constant, expressed in Equation 3.34. This value ensures that the gradients will be backpropagated without any attenuation or gain, allowing the early layers to be trained faster. Therefore, a class of ANN known as residual networks employs skip-layers connections (a connection that jump other layers) with linear activations to achieve state-of-the-art accuracy in many areas.

$$f'(z) = 1 \tag{3.34}$$

Although its usage as an activation for hidden layers increases the computing power required and does not offer a gain in the model's capacity, this function may be readily employed at the output layer in regression models.

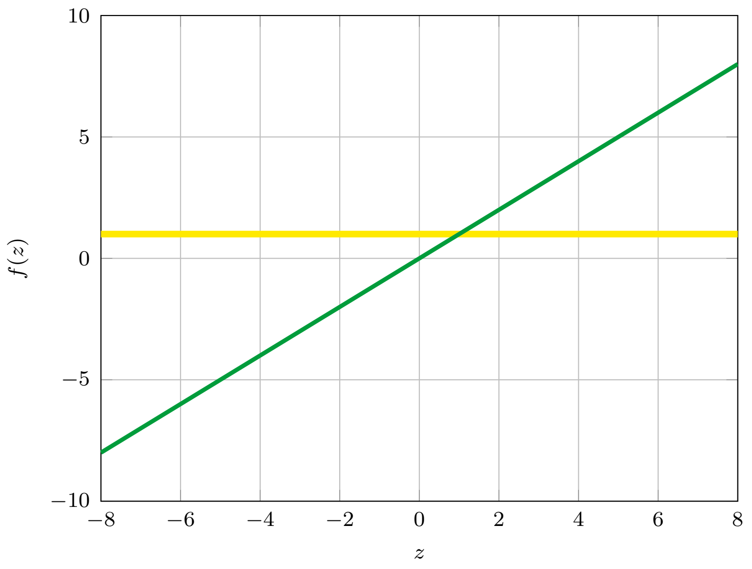


FIGURE 3.10 – Linear activation function and its derivative.

3.5.5 Rectified Linear Unit (ReLU)

This is an activation function that deals with the problem of the small gradients that accumulate in the sigmoid function in deep networks. Since its derivative is 1 for positive input values, the error signal that will be multiplied by this derivative will not be attenuated with the sigmoid function. The Equation 3.35 and the Figure 3.11 mathematically and graphically represents this function.

$$ReLU(z) = \begin{cases} 0, & z < 0 \\ z, & z \geq 0 \end{cases} \tag{3.35}$$

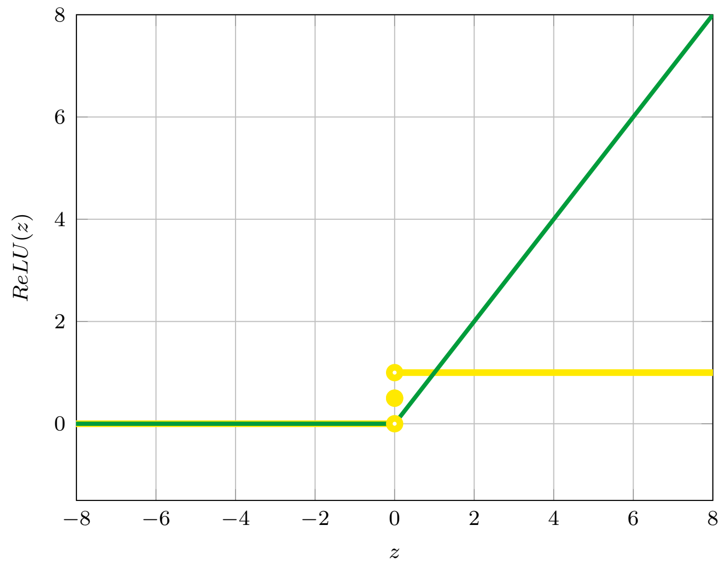


FIGURE 3.11 – Rectified linear unit activation function and its derivative.

3.5.6 Leaky ReLU (LReLU)

In contrast to the ReLU, the Leaky ReLU (LReLU) function is conceived as a ReLU which instead of being zero for negative numbers, which leaves the neuron completely inactive, gives a slight inclination, which still allows some learning. The Equation 3.36 and the Figure 3.12 mathematically and graphically represents this function.

$$LReLU(z) = \begin{cases} az, & z < 0 \\ z, & z \geq 0 \end{cases} \tag{3.36}$$

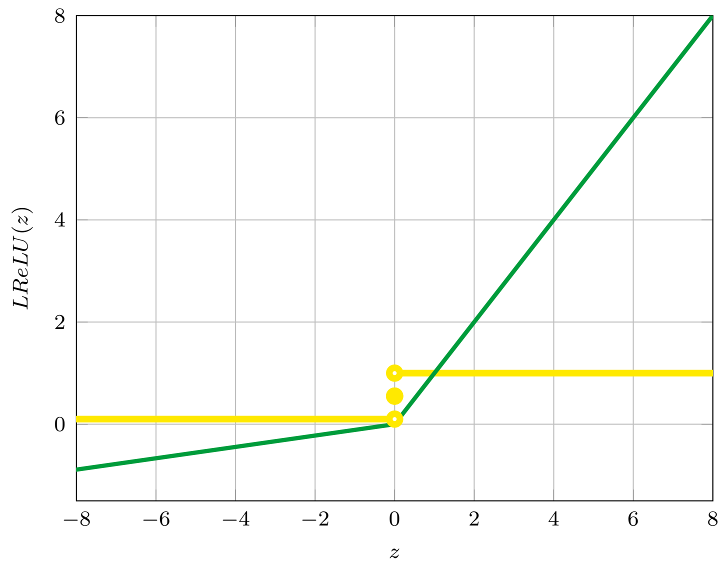


FIGURE 3.12 – Leaky ReLU activation function and its derivative.

3.5.7 Exponential Linear Unit (ELU)

Like ReLU's leaky modification, the ELU activation function is similar to ReLU, but is smooth (no discontinuity in the first derivative) and tends $a - 1$ for very negative input values. Thus, the Equation 3.37 and Figure 3.13 represent this activation function mathematically and graphically.

$$ELU(z) = \begin{cases} e^z - 1, & z < 0 \\ z, & z \geq 0 \end{cases} \tag{3.37}$$

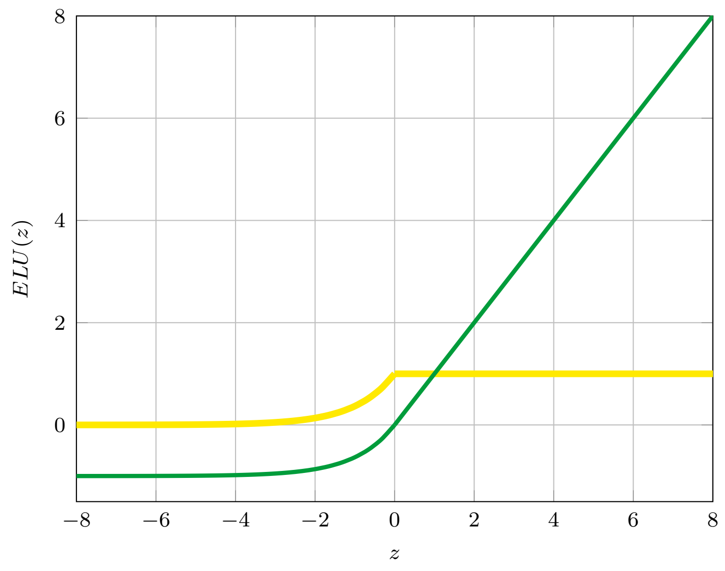


FIGURE 3.13 – Exponential linear unit activation function and its derivative

3.5.8 Softmax Function

Unlike other activations presented, softmax is a function that maps the output space to probabilities, as indicated by Equation 3.38. This normalization occurs by taking the penultimate outputs as logarithmic scale values. Thus such values are treated as the exponents of the softmax function, which still normalizes its sum to be equal to 1.

$$\sigma(z_1, z_2, \ldots, z_N)_i = \frac{e^{z_i}}{\sum_{k=1}^{N} e^{z_k}} \tag{3.38}$$

3.6 Loss Functions

Besides the already explained mean squared error, there are many different loss functions, but the most used are the squared hinge, mean absolute error, cross entropy, kullback-leibler and cosine proximity (EPELBAUM, 2017).

The Equation 3.39 indicates the mathematical formulation that the cross entropy has given C output classes, y the empirical distribution and p the predicted probability.

$$\mathbb{L} = -\sum_{i=1}^{C} y_i \log(p_i) \tag{3.39}$$

3.7 Regularization

Often in which the amount of parameters to be trained is orders of magnitude above the available data, or when there is enough data redundancy and few standards to be learned, the model continues to decrease the training error while the validation error. increases. In this case, the artificial neural network is said to be overfitting.

One of the clearest perceptions of this phenomenon happens in the interpolation of functions of only one variable. Thus, while a parable can fit perfectly to three points, or even a fifth-order polynomial to six points, the most reliable relationship between such data is simply linear. In this context, it is observed that functions with more parameters to be adjusted (in the case of polynomials are their coefficients) suffer more severely from the presence of disturbances, that is, noise present in the data or even the stochastic nature of the process than those.

3.7.1 L1 Norm

The L1 norm is a penalty of parameters that inhibits excessive increase in synaptic weights. Thus, we add to the cost function to be minimized the sum of the norm of all these synaptic weights weighted by the penalty rate calculated in Equation 3.40, where C represents the total cost. , E the approximation error, γ_{L1} the penalty parameter and w_i an index on all N weights.

$$C = E + \gamma_{L1} \sum_{i=0}^{N} \|w_i\| \tag{3.40}$$

3.7.2 L2 Norm

The L2 norm, like the L1 norm, is also a penalty for parameters that reach even higher values. Because it is quadratic, its derivative approaches zero smoothly near its zero input, ie parameters with values close to zero will hardly be penalized. Its mathematical formulation is indicated in Equation 3.41, where C represents the total cost, E the approximation error, γ_{L2} the penalty rate and w_i an index on all N pesos.

$$C = E + \gamma_{L2} \sum_{i=0}^{N} w_i^2 \tag{3.41}$$

In short, its difference with the L2 standard is the change from a sum of absolute values to a sum of quadratic values. It is important to note that the L1 standard favors the more frequent appearance of null weights in contrast to L2, which facilitates the construction of sparse nets. This is justified by the presence of a beak in the standard function $f(x) = \|x\|$ in contrast to the smoothness of the quadratic function $f(x) = x^2$ around the null point.

3.7.3 Dropout

Dropout is a technique that has recently become popular due to its effectiveness. It consists essentially of the random deactivation of neurons during the training phase. Later, in the execution of the model all neurons operate normally, performing a linear adjustment of their activations in order not to change their average in relation to the training phase.

3.8 Constraints

It is common to have constrains in neural networks so that the number of trainable parameters is diminished and a prior expert knowledge is inserted into the network. The most fundamental example of this phenomena is the Convolutional Neural Network (CNN) in which a fully connected becomes a filter that executes a convolution operation because its weights were constrained to be spatial equivariant.

3.9 Optimization Heuristics

Besides simply updating the synaptic weights with a gradient proportional perturbation by the α learning rate, there are other methods. One of the most popular heuristics has been the Adam Optimizer, a first-order gradient-based optimization algorithm for stochastic objective functions, by calculating individual adaptive learning rates for different parameters from first and second-moment estimations as shown in Table 3.1. gradients (CHOLLET *et al.*, 2015). The training parameters were those recommended by the original article and implemented by default by the Keras library learning rate 0.001 and decay of the null learning rate. Hyperparameters represent, respectively, the decay rates of first and second order moments, and the minimum value to avoid division by zero.

As an optimization strategy, the adaptive moment estimation (Adam) (KINGMA; BA, 2015) has been selected due to its efficiency and robustness provided by the estimates of the average first moment and second moments of the gradients. In other words, this estimation combines the momentum of the gradient, defined as its exponentially weighted sum (exponential moving average) observed in the equation 3.42, and also the normalization factor that is the momentum of the square of the gradient (element-wise) observed in the Equation 3.43. Its hyper-parameters are the learning rate α, whose initial value was set as 0.01; β_1, the smoothing factor for the first moment, with default value 0.9; β_2, the smoothing factor for the second moment, with default value 0.999; and ϵ, a small constant that guarantees numeric stability in the update equation's denominator, seen in Equation 3.44, avoiding division by zero.

In the Equations 3.42, 3.43, 3.44, W denotes the weight vector that represents all

TABLE 3.1 – Synaptic weight update algorithms. Adapted: (MELO, 2019)

Algorithms	Hyper-parameters
Descending Gradient	Learning Rate
Nesterov Moment	Learning Rate, Moment, Decay
Adam	Learning Rate, Moment, Decay, Mitigation
RMS-Prop	Learning Rate, Mitigation

trainable parameters in the neural network, V is the exponentially weighted sum from the gradient and S is from the square (element-wise) of the gradient. These equations are evaluated at every mini-batch iteration in the training phase, as the gradients estimates are evaluated.

$$V_{updated} = \beta_1 V + (1 - \beta_1)\nabla W \tag{3.42}$$

$$S_{updated} = \beta_2 S + (1 - \beta_2)(\nabla W)^2 \tag{3.43}$$

$$W_{updated} = W - \alpha \frac{V_{updated}}{\sqrt{S_{updated}} + \epsilon} \tag{3.44}$$

3.10 Bias and Variance Trade-off

One of the fundamental aspects during the training of any machine learning model is the observation of how well the model is performing in the training and validation sets by executing a comparative analysis between those data (NG, 2017). From this investigation, we can outline the next steps to be taken to improve the model, whether by collecting more data, introducing knowledge directly by architecture, adjusting the regularization factors.

It is crucial to note that for a final evaluation of the model in which the result is not biased, another data set should be used whose elements are not present in the training or validation sets. In addition, it is necessary that such a test set be sampled from the same distribution as the actual data, ie the actual operation in which the model will be employed. Thus, the average of the metric that has been optimized tends to the expected value of this same metric during the actual operation of the model, as observed in Figure 3.14. The yellow line indicates the error for the validation set and the green line is the training error.

In simply terms, the bias of a model refer to how well it performs on the training set whereas its variance comes from its error metrics in the validation set. While the usage of those terms is not precise to its statistical definition, it gives a sense whether the machine learner has oversimplified the training points, trying to fit a curve much less complex than necessary to correctly classify the points, as if it couldn't have a good estimation for the mean of the data, resulting in a high bias towards a wrong mean.

From the variance perspective of the machine learner, having learned correctly to estimate the mean of the data distribution, the model will make predictions centered on this mean point. Therefore, the sum of the square of the differences between the actual and the predicted values will be mathematically equals to the variance of the residuals.

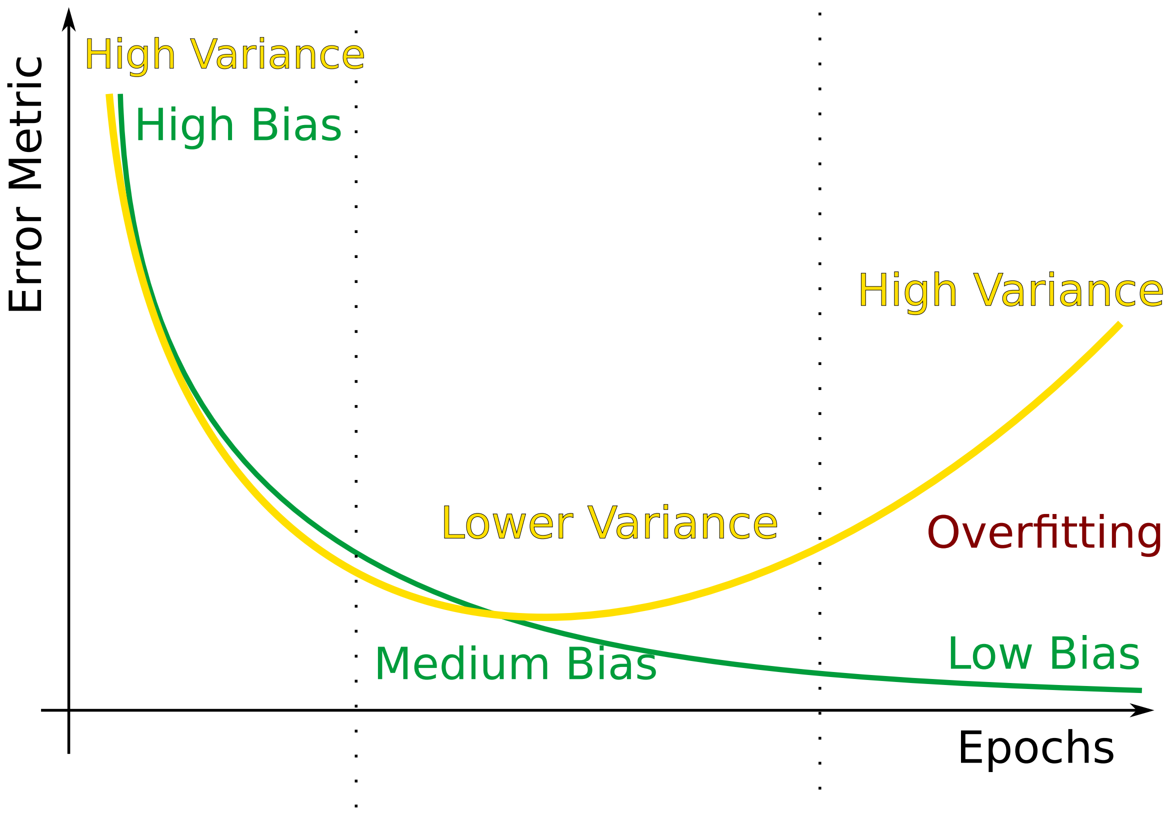


FIGURE 3.14 – Bias variance trade-off that may result in overfitting for large epochs. Adapted: (MELO, 2019)

As shown in Figure 3.14, starting from a random initialization in epoch zero, as the model learns the most basic properties from the data, both the variance and bias decreases until it hits a threshold were the variance starts to increase. This phenomenon is known as overfitting and results in a worse performance to new, unseen data.

4 Natural Language Processing

Natural language neural models have become the state-of-the-art technique for NLP (LIU, 2012) (MITKOV, 2003).

Based on Deng e Liu (2018) work, the methodology proposed in this paper consists of the following key stages:

- Determination and selection of input variables.
- Identification of the structure of neural networks for the natural language model.
- Determination of neural network parameters.

Each of the system-system research steps defines a methodology applicable to any natural language model it performs, which may be interpreted as an encoder-decoder architecture which observed in Figure 4.1.

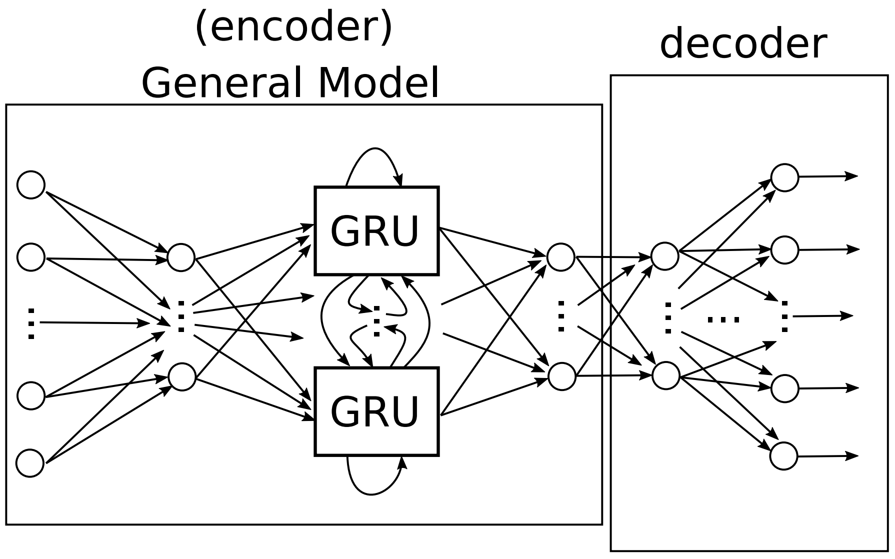


FIGURE 4.1 – Embedding model able to encode word features. Source: (MELO; TASINAFFO, 2019)

4.1 Discrete Representations

The most common representation of meaning by linguistics is the denotational semantics, that is the usage of a symbol, namely the signifier, in order to represent a signified, e.g. an idea or an object. The motivation for this concept may be even found in the dictionary definition of the word meaning, defined as the representation of ideas using word, signs and phrases or even as the expression of artistic of scientific works (MANNING; SCHüTZE, 1999).

This rational approach to language may be directly programmed into synonym sets or even hypernyms, which states the is-a relationships of words. This classical solution has the WordNet as a viable example, it is a dictionary of synonyms, a thesaurus, that contains the whole formal language explicitly assigned to semantics and syntax sets. The Natural Language Processing Toolkit (NTLK) provides those tools in an open-source environment including WordNet (LOPER; BIRD, 2002) in the Python programming language.

In this regard, word are treated as tokens, discrete symbols. In analogy to a dictionary, each of them may receive its unique number for identification, namely a localist representation. Although simply stored in the form of an integer, this number can also be interpreted as an index in a M-dimensional vector where all dimensions are null except for the one addressed by the index. The Figure 4.2 exemplifies such representation, having $M = 28976$ words and a identification taken directly from the dictionary's order. The third word is "Abandon" whose one hot encoding is a 28976 dimensional vector with its third position set to one while the rest is null.

Those discrete symbols produces the property of orthogonality for those vectors. In simpler terms, there is no prior notion of similarity between different words. This outcome is better than just assuming the index given by the alphabetic order as a feasible representation and using regression techniques for NLP problems because the regression loss would penalize more the error between the words "human" and "person" whose dictionary distance is 2971 than between "human" and "huge" that as a distance of 4. Using the one-hot representation assumes the same distance between every word pair and converts the regression problem into a classification one.

Using such techniques allows directly solving some problems concerning keyword searching, spell checking and synonyms finding. The problem rises when those mechanical tasks evolves to document parsing as ambiguities lurks in sentences and the number of edge cases starts to rise exponentially. Translations requires even more effort as not only syntactic structures may change but also the context dependency becomes more apparent. Other difficult tasks include question answering, co-reference and semantic analysis.

Besides this range of task hardly tractable by those tools, there are some limitations

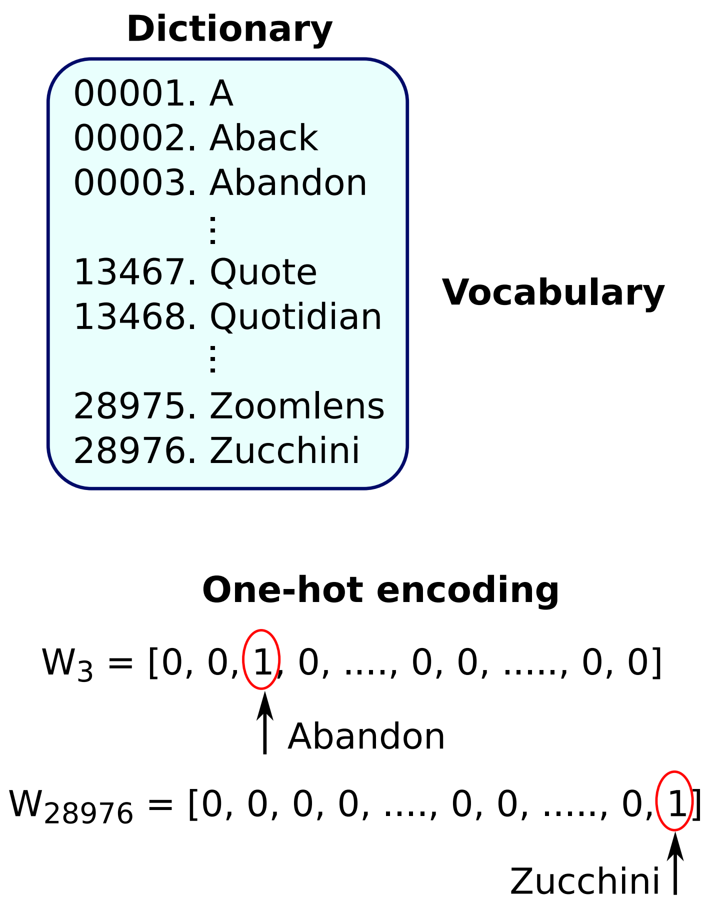


FIGURE 4.2 – One-hot encoding example given its vocabulary.

including the subjectivity experienced by the experts and programmers of those tools, the informal language and its new creations and meanings, low precision when computing word similarities (MANNING; SCHüTZE, 1999). Besides, they are human labor intensive to adapt and to program. The solution to part of those problems is presented in the next section as using the vectors themselves to encode similarity.

4.2 Word Embeddings

As a means to solve some limitations of the classical rule based methods in NLP, word embeddings try to capture the lexemes' meaning through their context. This methodology is also known as distributional semantics and it is the foundation for more elaborate techniques. Word embeddings are also known as word vectors or even word representations and their primary intuition is that each dimension should carry an attribute (or a linear combination of them) that should characterize the meaning such as gender, size, color, taste, goodness, hardness, liveness and others. Differently from one-hot encodings, each dimension in those distributed word representations is a real value, not just zeros and one (MIKOLOV *et al.*, 2013).

The main concept behind its training is that the context words, that is the nearby words in the sentence, should construct the meaning of the center word. Thus, the dense vector should have a high similarity relative to its context vectors as seen on Figure 4.3.

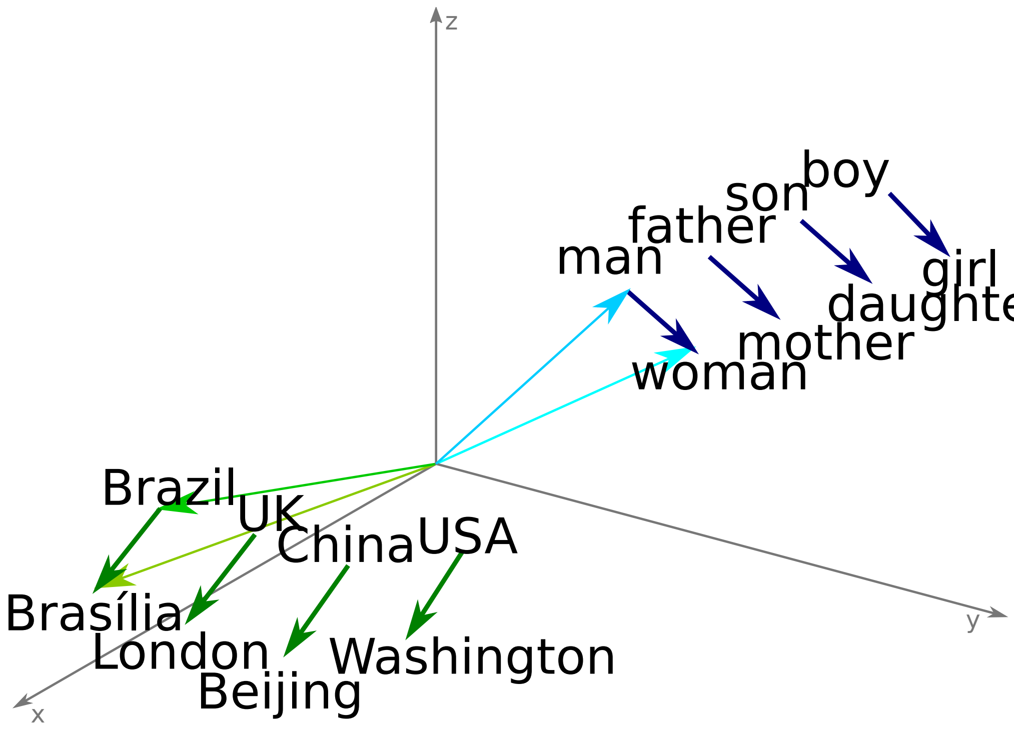


FIGURE 4.3 – Word vector visualization with similar meanings and analogies.

4.3 Unsupervised Learning in NLP

There are many approaches to the unsupervised learning in natural language processing from we which the most commonly used are briefly explained bellow.

4.3.1 Singular Value Decomposition

In this method the do-occurrence matrix is decomposed as the multiplication of three other matrices which are the solution for the Singular Value Decomposition (SVD).

The most prominent publicly available word embedding that employs this method is the Global Word Vectors (Glove) which uses a least squares objective function.

4.3.2 Neural Networks Initial Layer

A naive approach to full a softmax was originally employed in the early 90's work of Hilton.

Those classical approaches, as exemplified before, uses words as tokens in order to model its sequence, given its context as a set of cardinality N called N-gram.

There are two main approaches for learning unsupervised word vectors in artificial neural networks: skip-gram and continuous bag of words.

In the model introduced by Mikolov *et al.* (2013), the skip-gram method is employed in order to learn a compact vector for every word in the vocabulary. While the skip-gram tries to predict the center word given its context, the continuous bag of words does the reverse by predicting the context given a center word.

For instance, the Equation 4.1 defines the loss function for the skip-gram technique in a sequence of words $w_1, w_2, w_3, \cdots, w_R$.

$$L(\boldsymbol{\theta}) = \frac{1}{R}\sum_{r=1}^{R}\sum_{-t\leq i\leq t, i\neq 0} log(p(\boldsymbol{w}_{r+i,\boldsymbol{\theta}}|\boldsymbol{w}_{r,\boldsymbol{\theta}})) \tag{4.1}$$

Therefore, given the context of size $2t$, also called as the window that defines the N-grams, the loss functions is calculated as product of all those cumulative conditional probabilities. As a simplification, it is common to take the log of this product and transform it to a sum of log probabilities. As such, the optimizable parameter θ is iteratively tunned for each training case given the center word w_r and its context w_{r+i}.

For the continuous bag of words based model in which instead of predicting the context given the center, the objective is to predict the center given the context. Its loss function equation is somewhat similar to the skip-gram model.

4.3.3 Negative Sampling

In contrast, a more efficient methodology is to take pairs of words (bigrams) and train the neural network model to answer whether they belong or not to the same context.

The most known and used word embedding of this kind is the Word2vec.

4.3.4 Noise Contrastive Estimation

Noise Contrastive Estimation (NCE) is a especial case of negative sampling. The following structure was used, where P^h is the probability of a given pair of words being sampled from the same context $D = 1|w$ given the word embedding θ, which may be rewritten as the of founding this word $P^h_\theta(w)$ instead of the random distribution $kP_n(w)$. This whole task is considered by a logistic regression in which the sigmoid activation function σ is used, as depicted in Equation 4.2

$$P^h(D = 1|w, \theta) = \frac{P^h_\theta(w)}{P^h_\theta(w) + kP_n(w)} = \sigma(\Delta s_\theta(w, h)) \tag{4.2}$$

This approach learns faster from true data distribution than if a full softmax function were to be employed in a vocabulary of V words.

As the classic word embedding learning method was based on a linear combination followed by a softmax over all words from the vocabulary for calculating the probability of a given word being a center or context, the computational cost for finding the gradients was linear with respect to the size of the vocabulary. This amount is generally about tens of thousands of words up to millions of tokens.

The faster technique proposed by Gutmann e Hyvärinen (2010), also known as negative sampling, defines a probability metric that resembles the Monte Carlo Method for pairs of words. As such, its mathematical definition is expressed in Equation 4.3.

$$log(P(w_a|w_b)) = log\left(\sigma\left(\langle w_a\rangle\right)\right) + \sum_{i=1}^{r} log\left(\sigma\left(-\langle w_b\rangle\right)\right)\Bigg|_{w_i \sim P_n(w)} \tag{4.3}$$

It is important to note that a noise distribution P is required in order to make samples not only from the true distribution (or even a skewed one that favors less frequent words) but also from a "wrong" distribution of word pairs. Therefore, the negative cases will be calculated as a penalty for the neural network that is crucial for its learning (MNIH; KAVUKCUOGLU, 2013).

The idea for looking inside the words was also inspired by another approach Bojanowski *et al.* (2016). For the tasks presented in NLP, this represents a gain over larger vocabularies as the amount of characters is much lower than words. This approach was originally proposed for binary analysis project in the graduation Course in collaboration with Vitor Pimenta dos Reis Arruda.

In simply terms, all its need to be done is to have a Word Embedding that will be trained for calculating the internal product of two vectors, feeding the result to a logistic regression that will tell whether those two words belongs to the same context. The great difference between other negative sampling methods is in how the Monte-Carlo based choice of those words is done.

4.4 Character level encoding

Another contribution of this dissertation is a character level word encoding, observed in the Figure 4.4. It was originally proposed as a solution to a problem presented in a graduation course for binary analysis from compiled programs. Even though there already exists character level NLP models and encodings, the usage of multi-frequency LSTM was not known to this task. In other works, several layers of LSTM were used but they operated at the same frequency.

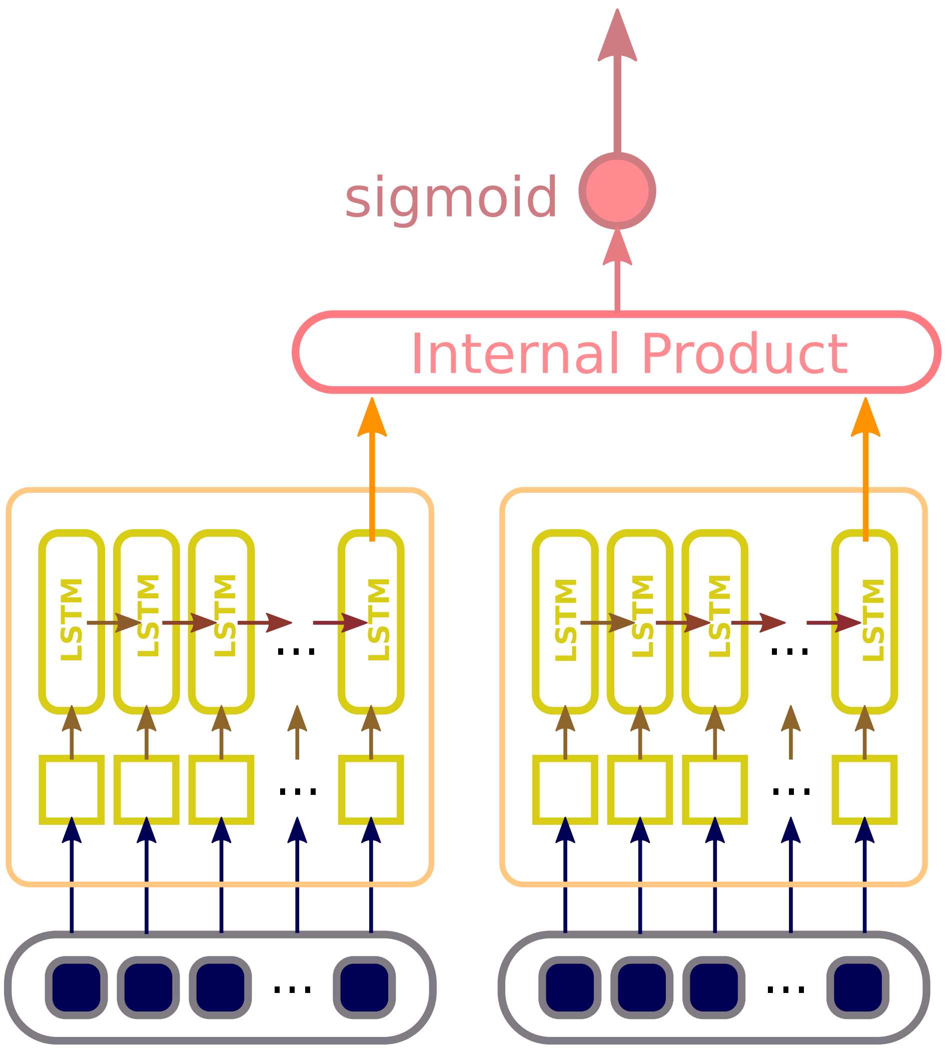


FIGURE 4.4 – Negative sampling for skip-gram learning in the character based encoding.

Utilizing the character-level word embedding, a recurrent neural network will be effectively divided as a dual-frequency sampler, as there will be memory cells operating at distinct time steps as indicated in Figure 4.5.

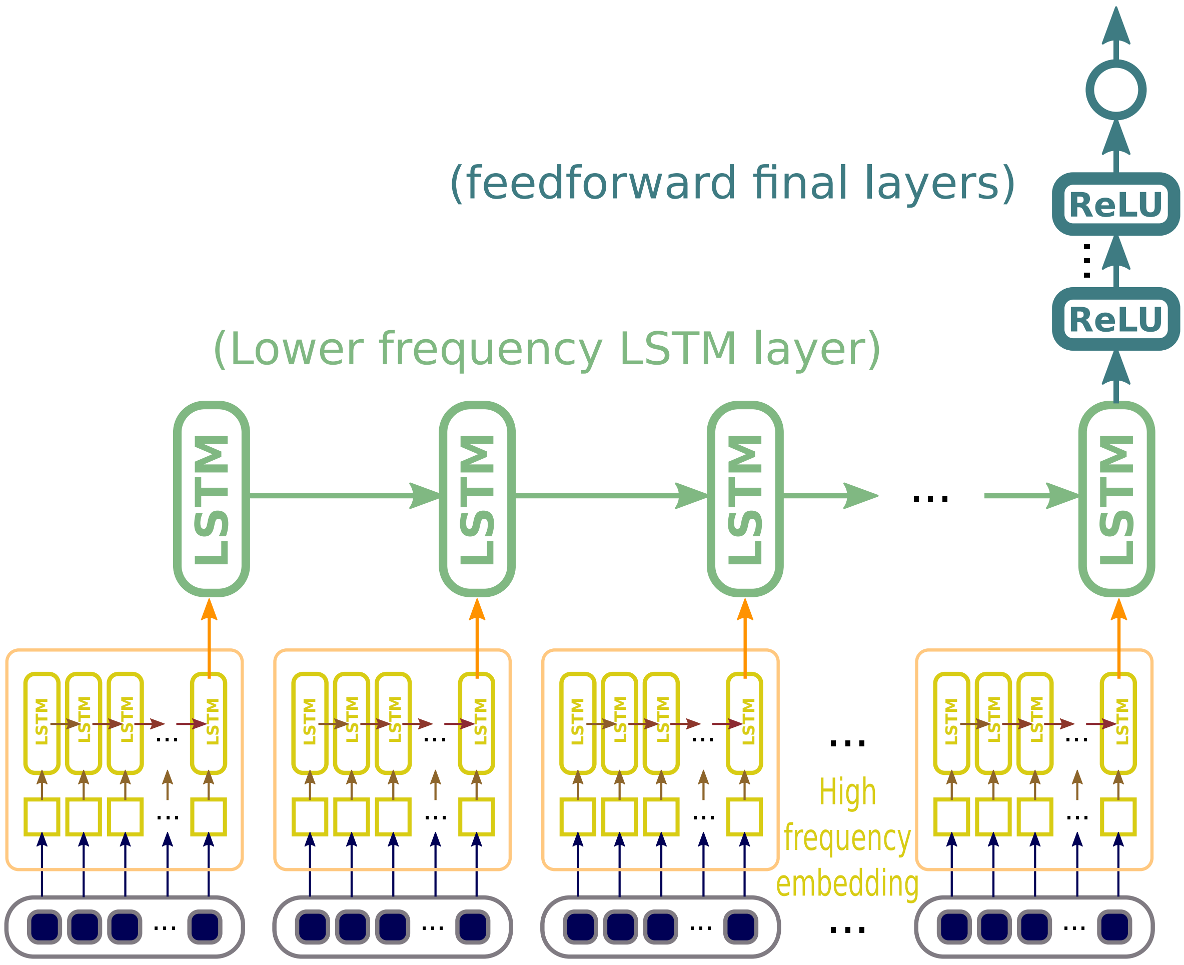


FIGURE 4.5 – Hierarchical LSTM placement for different sampling frequencies: character and word based.

5 Novel ANN Architecture

In this dissertation, a novel artificial neural architecture is proposed: The Hypercube of Cliques. The main aim was to keep the same level of learning capacity while diminishing the number free trainable parameters and potentially the amount of computation required to train and run the model. Therefore, a sparse structure was chosen as the foundation to develop this new architecture. This chapter presents the ideas behind the so called hypercube of cliques and similar works done as the Radix-Net, a tree-inspired approach.

5.1 Sparse Feedforward Models

A sparse neural network is characterized mainly for the absence of synapses between every pair of neurons in two consecutive layers. For instance, considering a 16-neuron layer as displayed in Figure 5.1, the first set consisting of three neurons connects only to other three in the next layer despite the existence of 12 other neurons. This naive approach was the first investigated in this dissertation that divided de N neurons in $\frac{N}{log(N)}$ cliques with size $log(N)$, resulting in only $Nlog(N)$ weights per layer. As evident in Figure 5.1, the cliques are disconnected of each other, so that the neural network behave as several networks executed in parallel, drastically decreasing its capacity.

There are two main approaches for sparse neural networks: the first is based on pruning the weights of an already fully connected neural network while the second focus on developing structures before the training, potentially decreasing the time during that stage. In a literature review, Elizondo e Fiesler (1997) cited the potential advantages of such architectures as an improved generalization capabilities, faster training and execution. There were methods for "weight dilution", which consisted of randomly cutting synapses give a distribution, simulation of neuronal biological networks, variational connectivity, geometrical relationships, local or shared weights (such as CNNs) and ontogenic methods such as genetic algorithms based hyper-parameters.

Other works present in the literature, such as Bourely *et al.* (2017), utilized random and even bipartite graph topologies in order to achieve the desired sparsity in their architectures while arguing that dense layers have unnecessary redundancies and that the

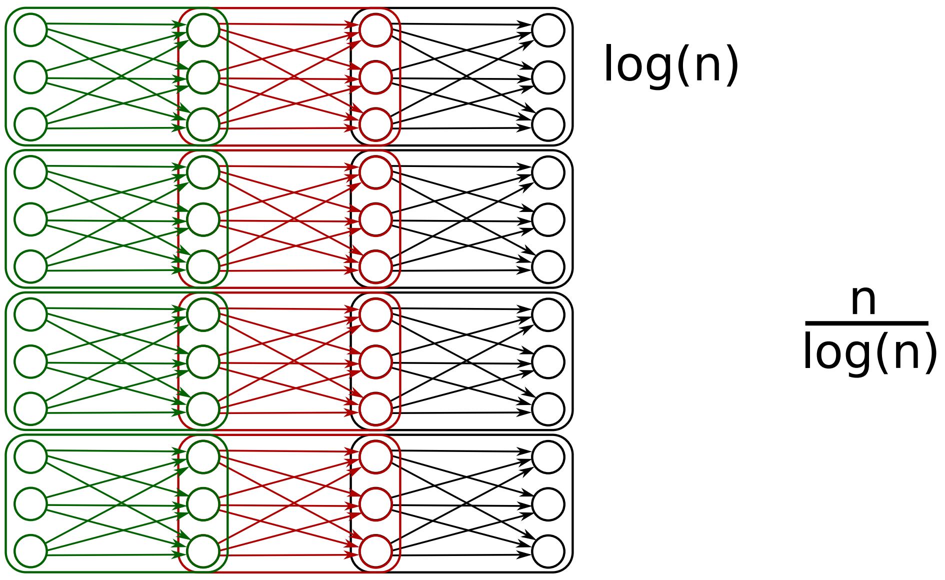


FIGURE 5.1 – Neural clique model proposed to increase structural sparsity.

performance of those sparse networks depended much in its connectivity.

The SqueezeNet (IANDOLA *et al.*, 2016) is a five hundred times smaller model than the AlexNet Convolutional Neuron Network (CNN) that performs similarly by using model compression techniques such as filter dimensionality reductions, downsampling and decreasing the number of input channels. As it was based on an already trained CNN, its performance suggests the ability to create more efficient connections while maintaining the error metrics.

The two approaches presented before may also be integrated by learning not only the weights but also the connections at the training stage (HAN *et al.*, 2015). This approach trains a fully connected model while also learning the importance of each synapses which, in a later stage, are pruned using a three-step method. This technique, while still not useful for decreasing the training time, allows for more efficient embed devices execution.

Similarly, Lin *et al.* (2017) proposed a framework that prunes the neural networks at runtime in a dynamic manner. They calculate the probabilities for a reward action of removing or adding weights by using a Markov Chain decision model, as in the Q-learning technique. Therefore, for each training step, a set of greedy actions are evaluated and executed, dynamically updating the structure. Although this tailors the pruned architecture for to have a similar performance (in means of its error metrics) and faster execution time, the training phase actually gets slower, as more computation is required for those procedures.

While testing the performance of those two approaches, Alford *et al.* (2018) observed that pruning may also act as a regularization technique, improving even the error metrics in cases of overfitted models or near this threshold. It was also noted that an excessive increase in sparsity may result in less stable training initializations, affecting the model's convergence, drastically changing its accuracy for small variations in the weight matrix.

Another approach, the HashedNets (CHEN *et al.*, 2015) constructs the synaptic connections by using a simple hash function in order to cluster different randomly selected neurons in a hash bucket. In this way, a weight sharing technique is employed among those neurons, as shown in Figure 5.2, where a stored weight matrix is expanded to a virtual dense matrix. The gradients over those shared weights are computed as the sum of each individual component, just as in CNNs. Furthermore, by carefully choosing a hash function and picking appropriate number of neurons, one may emulate a CNN in a densely connected layer. One claimed benefit with this hashing method is that there is no additional memory overhead. Nevertheless, the number of calculations performed is still the same as a densely connected network. A small reduction in the training time may be attributed to the lower memory bandwidth requirement if the training dataset were to be small enough to fit in the cache memory.

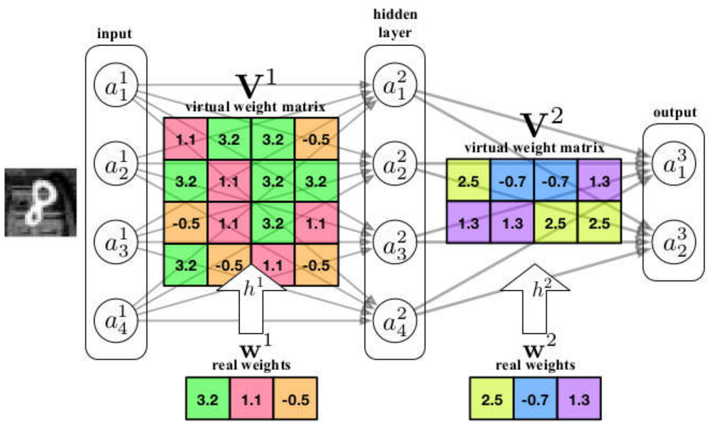


FIGURE 5.2 – The HashedNet functioning as a densely connected layer that share most of its weights across distinct neurons. Source: (CHEN *et al.*, 2015).

By using a hierarchical structure that employs $\mathcal{H} - matrices$ (FAN *et al.*, 2018) and $\mathcal{H}^2 - matrices$ (FAN *et al.*, 2019), these authors developed a N and $Nlog(N)$ sparse architecture respectively, and somewhat resembles the tree and tree-inspired approaches that this dissertation offers. Nevertheless, the multiscale decomposition is done by rewriting the hierarchical matrix into three different networks applied sequentially. Those are the restriction, the kernel and the interpolation linear networks constructed from an SVD (Singular Value Decomposition). The kernel may be substituted for a non-linear ANN that is translation invariant for several frequencies, resulting in a more robust model.

5.2 Tree inspired approach

Based on a simple binary tree, the connections of 2^N neurons may be arranged in N layers, each one having half the number of neurons on the previous. This structure is shown in Figure 5.3 at the dark green neurons and weights, which starts with 8 in the bottom layer and finished with a single one in the topmost layer.

It is important to note that the stride doubles every layer, in other words, the difference between neurons that connect to the same destination increases. By repeating this pattern over the next layers just by shifting this mask the neural network is constructed. This is represented by the other colors such as red and yellow for the representation in the second hidden layer. Four other combinations emerges from the last layer as the later activations are cyclically operated even though their inputs are the same for the case were the number of neurons is a power of 2. Other combinations of bases may be also tried, such as having a fan-in of 3 and increasing the strides three-fold for each layer.

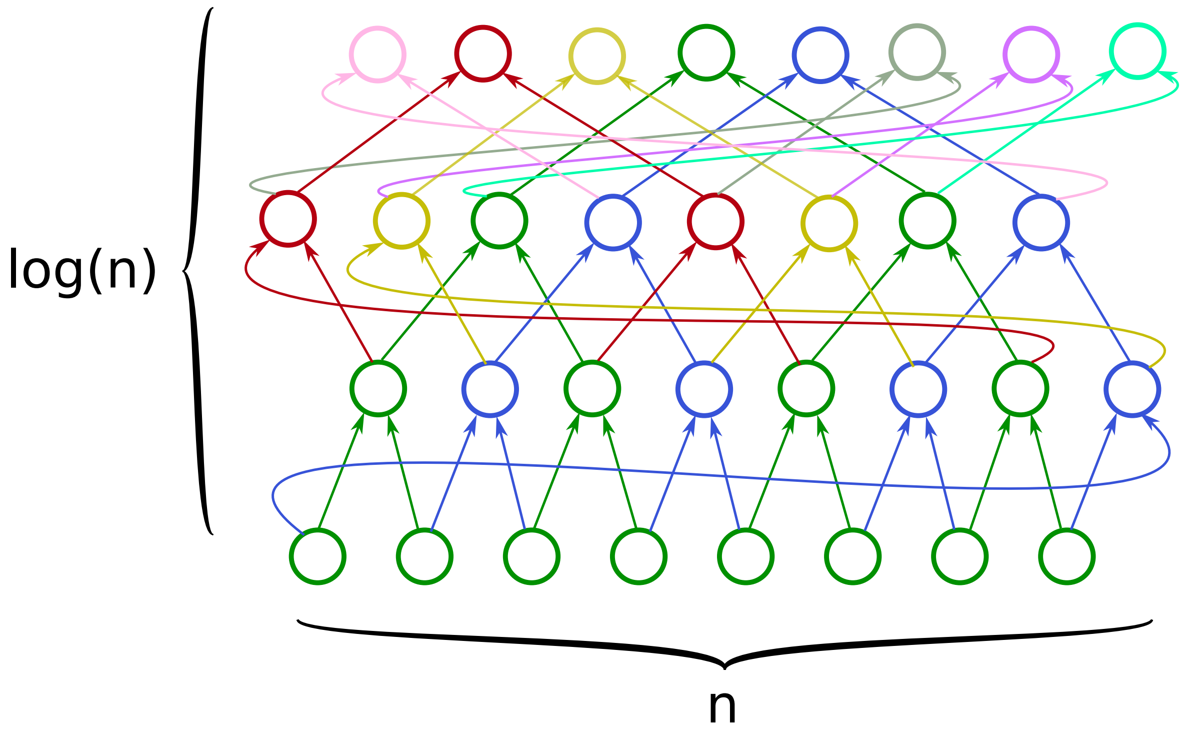


FIGURE 5.3 – Tree-based approach for sparse neural network having fixed fan-in.

Although this direct tree mapping was created in this dissertation unknowingly to the work of X-Nets by Robinett e Kepner (2018) and RadiX-Nets by Robinett e Kepner (2019) which has the same architecture as presented on Figure 5.3. Their construction was based on Kronecker products while this dissertation's used simply a geometric progression of strides modulo the number of layers, as shown on the layer mask created in the Listing 5.1. While this mask is useful for studying its graph properties, the computation on the forward and backward pass should be done as a the mask's construction with $2N$

operations, not as a matrix multiplication. For a more immediate application, the Python 3 programming language together with its Numpy matrix library were employed in those listings.

Listing 5.1 – Direct tree mapping to neural networks.

```python
import numpy as np

def tree_sparse_weights(input_dim, output_dim, base):
    mask = np.zeros((input_dim, output_dim))
    for i in range(input_dim):
        mask[i][i*base%output_dim] = 1
        mask[i][(i+i)*base%output_dim] = 1
    return mask
```

The main limitation for this direct tree mapping is the relatively high correlation between closer neurons, which limits the model performance significantly. This behavior may be observed not only on the first layers which there is a spatial correlation with the input but also on the latter layer where many neurons may share the same input combination, as shown on Figure 5.3.

A solution was to increase the fan-in of every neuron by a factor of $log(N)$ as if the whole tree was compressed into a single layer. Therefore, while strides were doubling in different layers, with this new increase in parameters the strides become a single operator over the as shown in the first layer of Figure 5.4. The colors were preserved in order to facilitate a closer inspection of the strides: the green represents a stride of 1, the blue is 2 and the red is 4.

Therefore, there are $2Nlog(N)$ weights represented in a single layer, as expressed on its mask in the Listing 5.2 which returns its adjacency matrix. As part of the implementation, self-connections was allowed contributing to more N weights which resulted in a total of $(2log(N) + 1)N$ weights considering N a power of 2.

Listing 5.2 – Condensed tree mapping to neural networks.

```
def condensed_tree_sparse_weights(input_dim, output_dim, base):
    mask = np.zeros((input_dim, output_dim))
    connections = [base * 2**i for i in range(math.ceil(
                                        math.log2(output_dim)))]
    for i in range(input_dim):
        mask[i][i%output_dim] = 1
        for j in connections:
            mask[i][(i+j)%output_dim] = 1
            mask[i][(i-j)%output_dim] = 1
    return mask
```

This tree inspired approach is very similar to the Radix-Net while the doubling stride technique was also employed at WaveNet for speech detection in several frequencies.

Even though the layer condensation was supposed to build more parameters, it was noted that an order of $Nlog(N)$ parameter for each layer was still significantly behind the best models performance that used densely connected layers, especially in wide ANN, with more than one thousand neurons per layer.

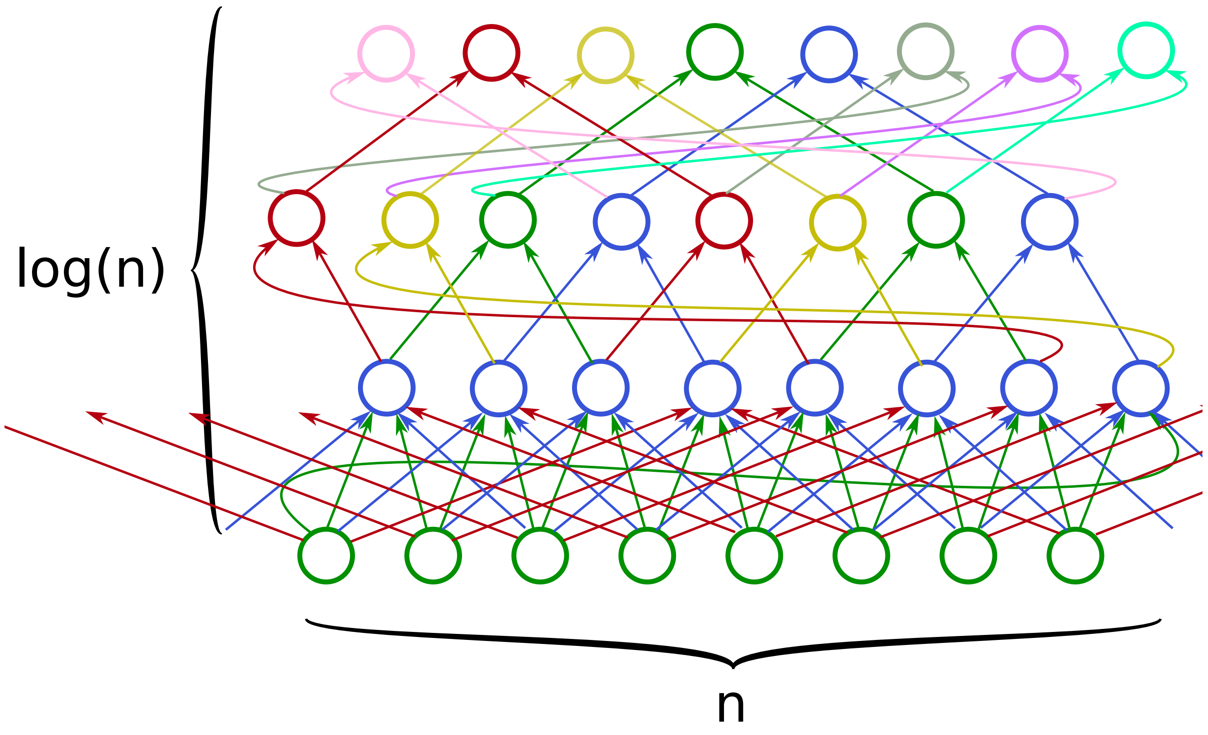


FIGURE 5.4 – Adaptable fan-in for more connectivity in the last model.

Therefore, another model that had somewhat more parameters per amount on neurons was needed for those tasks. Some graph topologies were investigated such as Hamming

Graphs, especially hypercubes, which were found to have a proportional quantity of parameters and similar properties.

5.3 Hypercube motivated architecture

5.3.1 Naive Hypercube

In its simplest form a Hypercube Graph is a Hamming Graph in which by representing every vertex as a binary encoded number, there will be an edge connecting all other vertices that differs by only one bit, in other words, that has a Hamming Distance of one. Therefore, a fast implementation of this operation is depicted in the Listing 5.3 where a bitwise exclusive-or is performed. Having an amount of parameters directly proportional to $Nlog(N)$, as observed for $N = 16$ in Figure 5.5, this direct mapping to a neuron network still suffers a severe performance drop for a wide (more than one thousand neurons) layer.

Listing 5.3 – Naive Hypercube.

```
def mask_hipercubo(input_dim, output_dim, dobro=0):
    mask = np.zeros((input_dim, output_dim))
    for i in range(input_dim): # input_dim > output_dim
        mask[i, i] = 1
        for j in range(int(math.log2(output_dim))):
            mask[i, (i^(1<<j)) % output_dim] = 1
    return mask
```

Another approach similar to this naive hypercube was shown by using crossbar switches, and guaranteeing five properties such as “pre-determined connectivity, uniform and high path diversity, full connectivity, and an efficient hardware implementation” (ISAKOV; KINSY, 2018).

5.3.2 Hypercube of cliques

As the largest value proposition for this dissertation, the idea of a hypercube of cliques came as a solution for having an amount of parameters proportional to $Nlog^2(N)$, as noted by the two loops in the Listing 5.4 that process $log(N)$ weights per step. Therefore, a clique (densely connected group) of $log(N)$ neurons is placed at each vertex, having $log^2(N)$ connections between themselves. The old edges give place to a group of $log^2(N)$

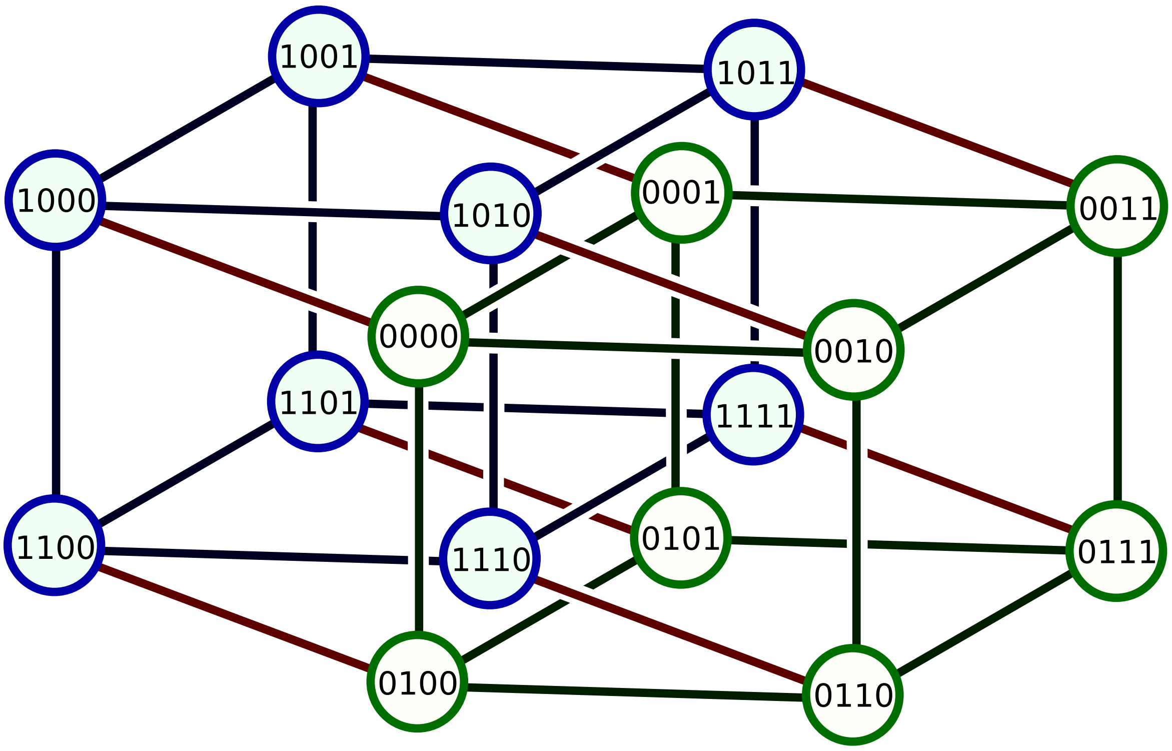


FIGURE 5.5 – Naive hypercube architecture for 16 neurons forming a tesseract.

edges resulting in the numbers presented in Equation 5.1.

$$\begin{aligned} E_o &= N_o log(N_o) \\ N &= N_o log(N_o) \\ E &= E_o log^2(N_o) + log^2(N_o) \end{aligned} \tag{5.1}$$

By expressing the old number of edges E_o and neurons N_o in terms of the new N, there is the approximate result demonstrated in Equation 5.2.

$$\begin{aligned} e^N &= e^{N_o} N_o \\ E = (E_o + 1)log^2(N_o) = (N + 1)log^2(N_o) &\approx Nlog^2(N) \end{aligned} \tag{5.2}$$

The simple tesseract has now transformed into a more complex, yet not fully connected, graph shown in Figure 5.6. As $N_o = 16$, there are now $N = 64$ in this new architecture. Their edges have been made wider in order to represent them not as a single connection but as a bus of edges. While giving more flexibility for the reader, the topmost-left group of neurons had three of their buses expanded to the elementary connections.

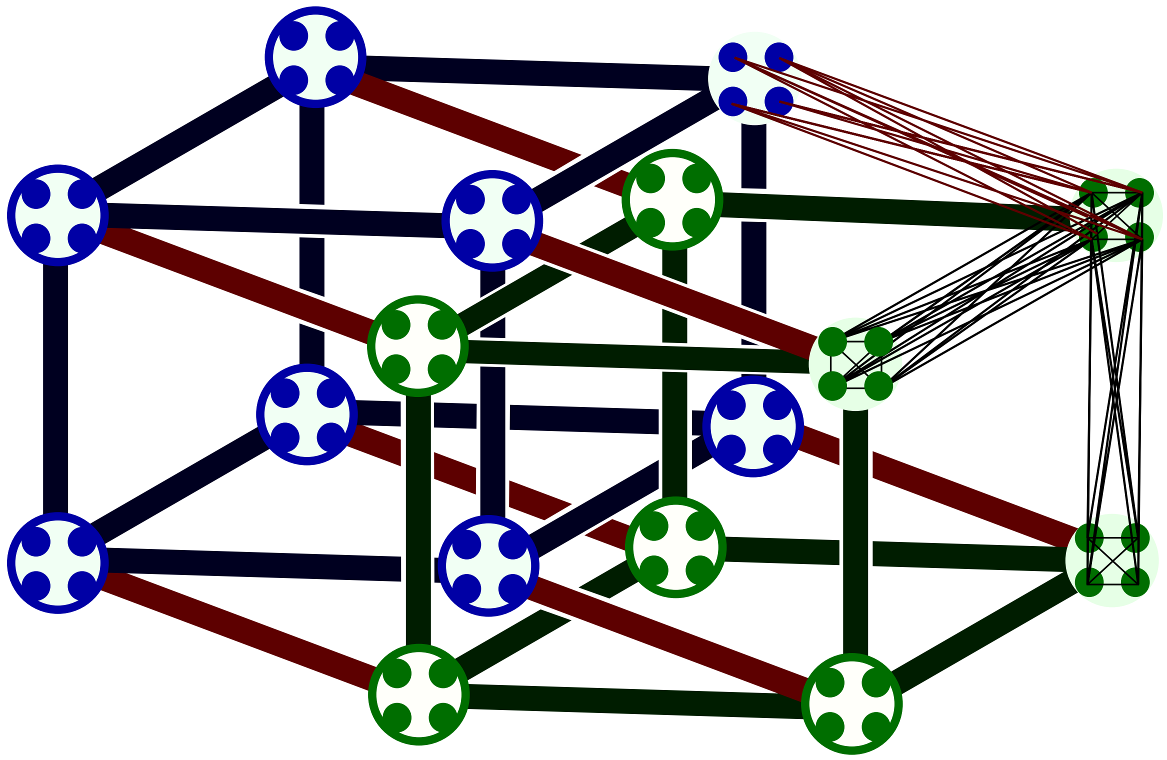


FIGURE 5.6 – Hypercube of clique architecture.

Listing 5.4 – Hypercube of cliques.

```
def mask_hypercube_cliques(input_dim, output_dim, dobro=0):
    mask = np.zeros((input_dim, output_dim))
    l2e = int(round(math.log2(input_dim)))
    qe = input_dim//l2e
    for i in range(qe): # input_dim > output_dim
        mask[(i*l2e):((i+1)*l2e), (i*l2e):((i+1)*l2e)] = 1
        for j in range(qe):
            mask[i*l2e:(i+1)*l2e,
                    l2e*(i^(1<<j)):l2e*((i^(1<<j))+1)] = 1
    return mask
```

While the *l2e* size of each cluster may be also defined as $log(N)$ as in Listing 5.4 having a bigger size, the Equation 5.2 presents another smaller solution yet larger than $log(\frac{N}{log(N)})$. While this bigger counterpart allows for exact mapping when $N = 2^{2^a}$, the smaller one has a exact solution when $N = a2^a$ where a is a positive integer. Both may be implemented using block matrices representations.

To put this proposition in perspective by comparing the different approaches researched and developed so far, the Figure 5.7 depicts, in a linear scale, the number of trainable weights each architecture presents for a given amount of neurons on a single square layer interface. There are small "bumps" in the sparse graphs caused mainly by the addition of a new bit such as from 511, represented as 111111111 in binary, to 512, which is 1000000000.

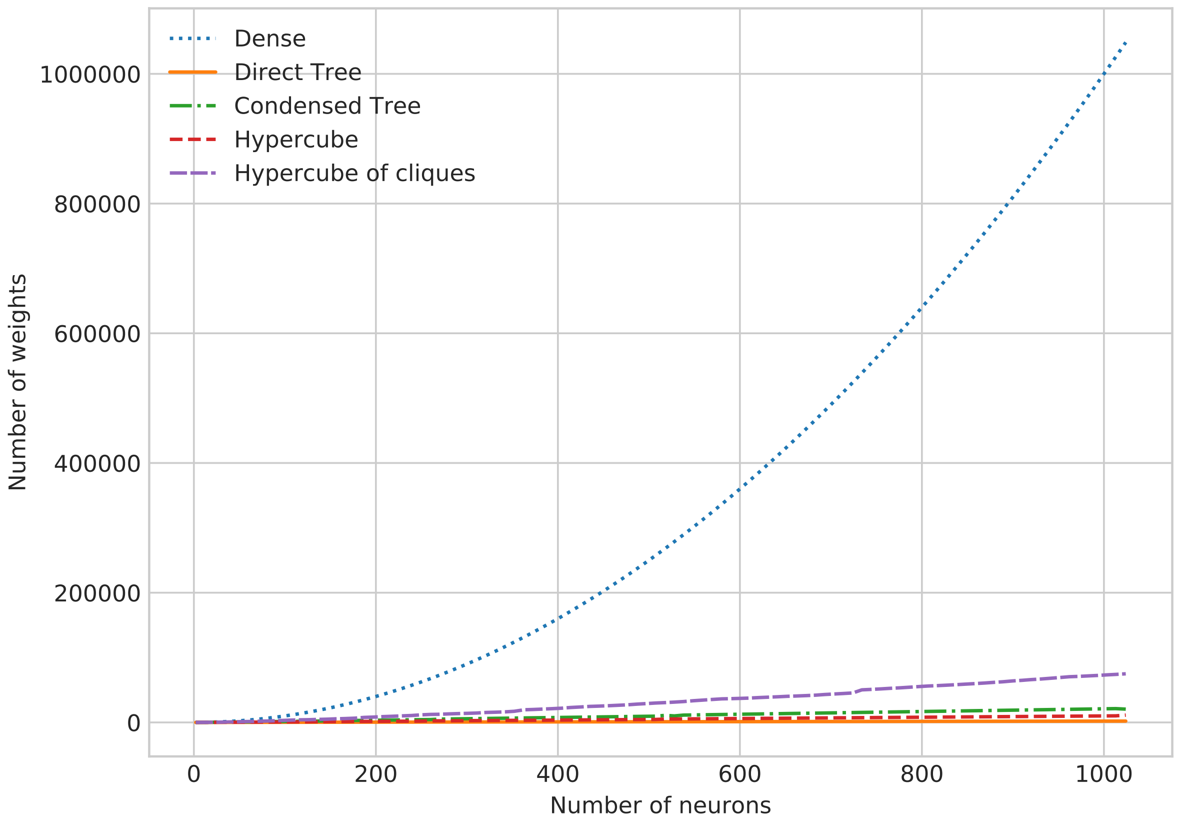


FIGURE 5.7 – Comparison of the free trainable weights amount between the developed sparse models.

A more detailed view on those sparse matrix as well as a new product definition can be found in Appendix A. Essentially, a sparse matrix may be represented in a compact manner occupying only a space proportional to its non-zero possible elements without using a linked list approach, but another essentially small dense matrix with $log(N)$ or $log^2(N)$ columns and N rows.

Listing 5.5 – Hypercube of cliques scattered.

```
def mask_hypercube2(input_dim, output_dim):
    mask = np.zeros((input_dim, output_dim))
    for i in range(input_dim): # input_dim > output_dim
        mask[i][i] = 1
        for j in range(int(math.log2(output_dim))):
            for k in range(j):
                mask[i][(i ^((1<<j))|(1<<k))] = 1
    return mask
```

While this sparse architecture has a small loss in capacity for the same number of neurons, as observed in Figure 5.8 for the N_2 amount, a modest increase on this hyper-parameter may recover this metric. For many practical applications, depending on the dataset and on the task being performed, this decrease in capacity could have a beneficial impact on the generalization accuracy provided the original hyper-parameters were close to the overfitting region. It is important to note that while the Figure 5.7 was constructed using real data by effectively running the presented listings, the Figures 5.8 and 5.9 represents a conceptual analysis on the model complexity based on empirical evidence.

A more formal definition for the model capacity is expressed through the Vapnik–Chervonenkis dimension (VC) measure. For a machine learning model this metric may be expressed as the maximum amount of arbitrary points that a model may classify correctly. Other definitions of capacity could also be explored, such as the Rademacher complexity, although it would require a more detailed analysis and a comprehensive study.

Nevertheless, the benefits of this proposition is better observed on Figure 5.9 in which not the width of the layer but its total amount of parameters is expressed. This relation is made possible by the result already shown in Figure 5.7 where the number of trainable weights is related to the quantity of neurons by a factor of n^2 for the densely connected architecture in comparison to its $nlog^2n$ sparse counterpart.

This significant decrease in the parameters' count is made evident by the negative curvature of those functions despite its strictly monotonic increasing characteristic. Another interesting observation to be made is the similar behavior that those functions have for small networks, as the sparse distribution inevitably becomes a densely connected one when $n = 2$.

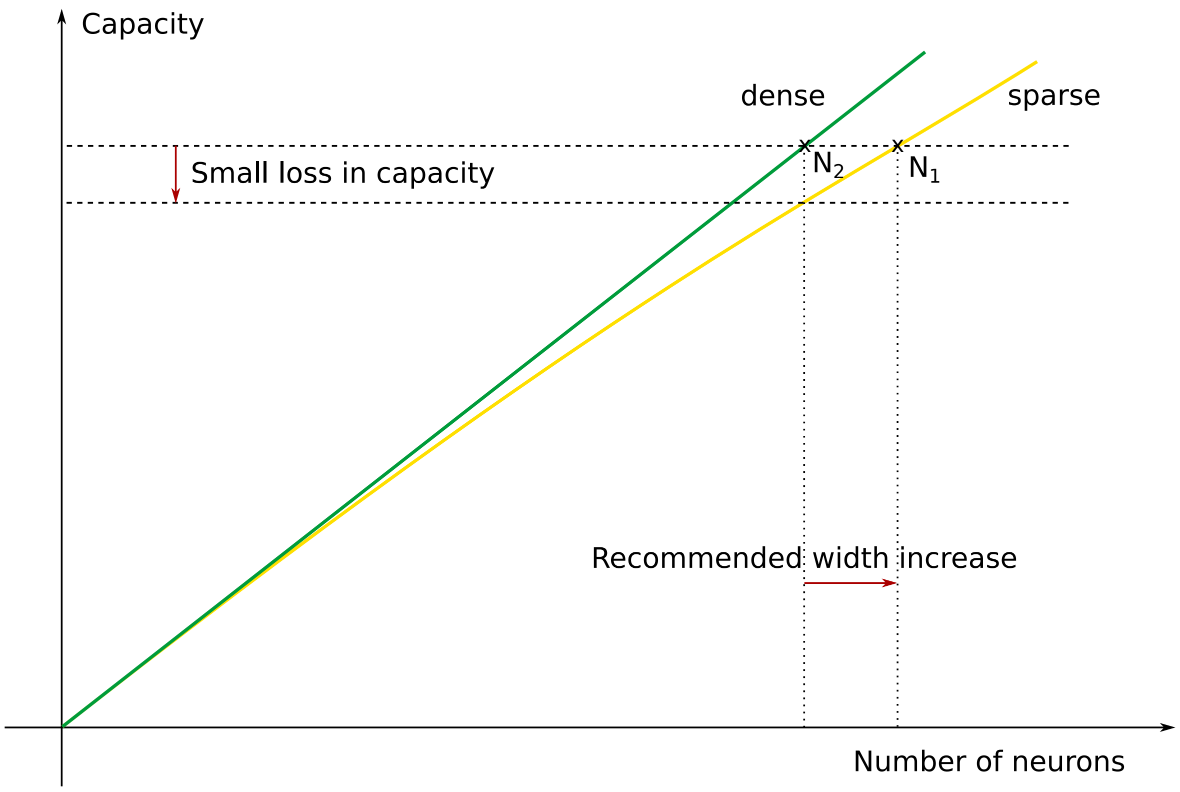


FIGURE 5.8 – Comparison of the neuron's quantity between a sparse model capacity and its dense counterpart.

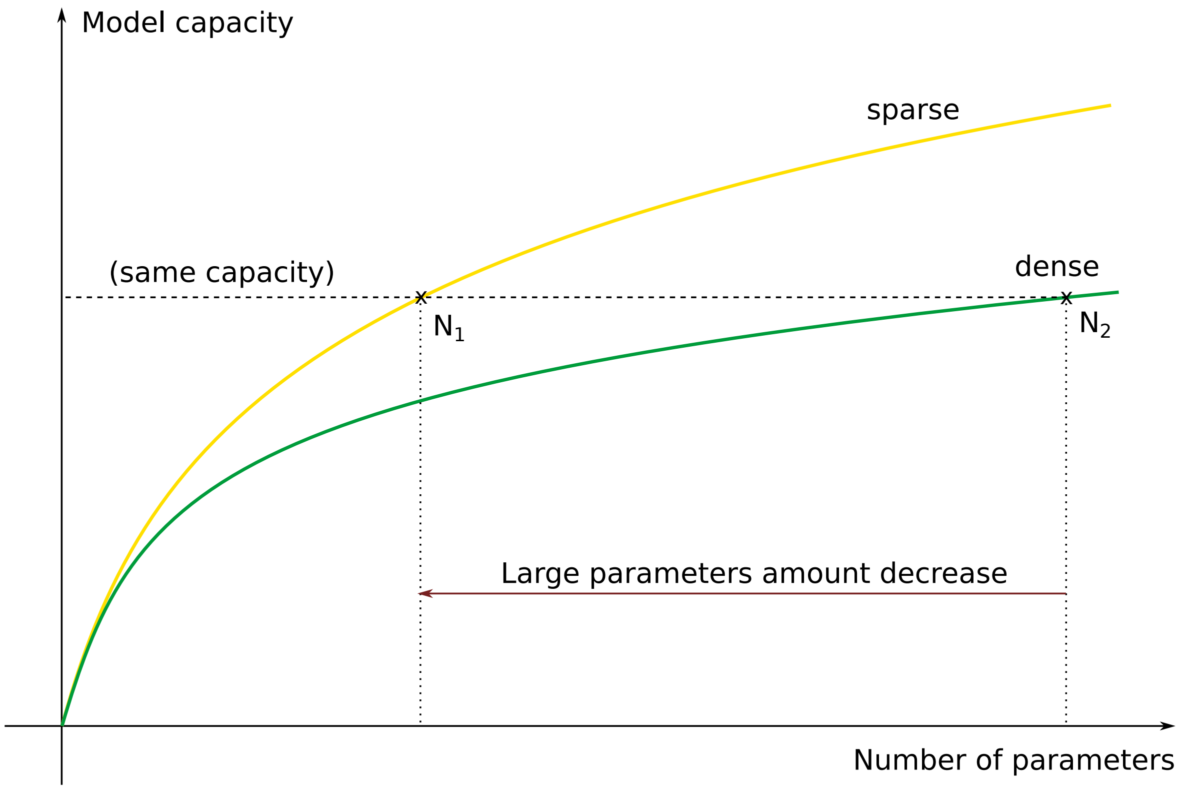


FIGURE 5.9 – Comparison between a sparse model capacity and its dense counterpart in terms of its amount of trainable parameters.

6 Dataset Analysis

In this chapter a preliminary analysis is presented on the data that were used to train the machine learning models. For the sake of comparison between the performance of past models, publicly available data from a wide range of open source projects developed by different organizations were processed as inputs for the neural networks models proposed and also to architectures reproduced from the literature. The Jira Software, a web framework that tracks issues and is utilized in agile project management, was the main source of this data.

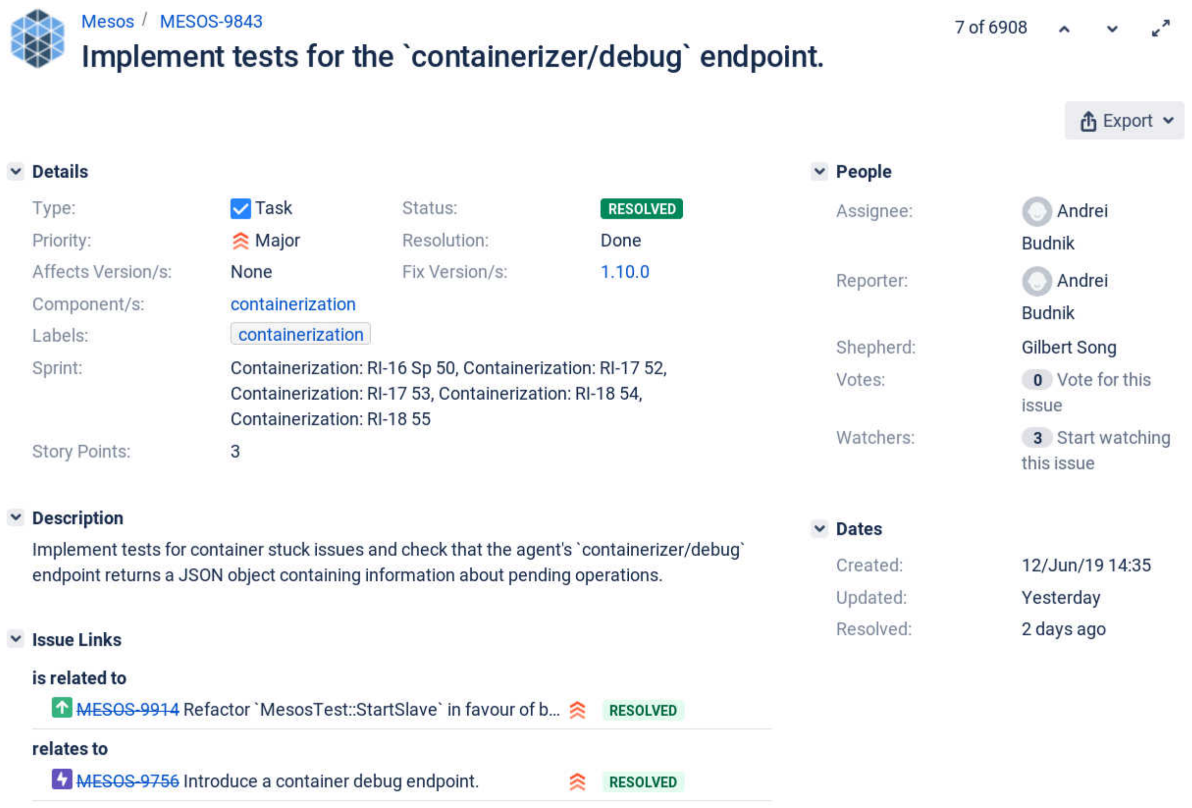
A case study was also performed in the CCA-SJ, where they employed the Redmine as their issue tracker platform Even though the data is another language, namely, Portuguese, the only difference on its treatment was the tokenizer, which had to be specifically tailored for the language. Unfortunately, due to the sensitive nature of the information present in this dataset, it couldn't be made available. Nevertheless, this case study provides an evaluation of the proposed model to new, unseen, datasets that are present in closed organizations.

6.1 Open Source Projects

As nowadays there are many open source projects available in conjunction with its development planning data. This gives a reasonable amount of user stories or even just issues to analyze and study and also enables the use of more complex machine learning models such as deep neuron networks.

The following organizations were had its Jira management website employed Apache, Appcelerator, Atlassian, DuraSpace, LSST Corporation, Moodle, MuleSoft, Spring, TalendForge.

The Figure 6.1 shows a user story in its original Jira project managing environment, namely for the Mesos project in the Apache repository. It is observed that the issue MESOS-9843 is actually an user story, even thought it is not written in its canonical format, that requires the programmer undertaking this task to implement test for the container. It was given the value of three story points and had a simple title and description text. Other data such as the priority, which as marked as Major, labels, status, sprint and resolution were not gathered even though it could be useful as most of the older issues in other projects hadn't those fields.



Mesos / MESOS-9843

Implement tests for the `containerizer/debug` endpoint.

7 of 6908

Export

Details

Type:	Task	Status:	RESOLVED
Priority:	Major	Resolution:	Done
Affects Version/s:	None	Fix Version/s:	1.10.0
Component/s:	containerization		
Labels:	containerization		
Sprint:	Containerization: RI-16 Sp 50, Containerization: RI-17 52, Containerization: RI-17 53, Containerization: RI-18 54, Containerization: RI-18 55		
Story Points:	3		

Description

Implement tests for container stuck issues and check that the agent's `containerizer/debug` endpoint returns a JSON object containing information about pending operations.

Issue Links

is related to

~~MESOS-9914~~ Refactor `MesosTest::StartSlave` in favour of b... RESOLVED

relates to

~~MESOS-9756~~ Introduce a container debug endpoint. RESOLVED

People

Assignee:	Andrei Budnik
Reporter:	Andrei Budnik
Shepherd:	Gilbert Song
Votes:	0 Vote for this issue
Watchers:	3 Start watching this issue

Dates

Created:	12/Jun/19 14:35
Updated:	Yesterday
Resolved:	2 days ago

FIGURE 6.1 – Apache Mesos Jira website example with an user story.

This resulted in the following projects being raised Appcelerator Studio, Aptana Studio, Bamboo, Clover, Data Management, DuraCloud, Jira Software, Mesos, Moodle, Mule, Mule Studio, Spring XD, Talend Data Quality, Talend ESB, Titanium SDK/CLI, Usergrid.

The table 6.1 depicts the main metrics from those datasets. The CSV (Comma Separated Values) files were obtained from Choetkiertikul *et al.* (2019) so that a fair comparison using the same data could be made between the models. The unique identification of the issue, as well as its title, description and story points are present in each line of the data files, separated by commas.

As a surprise during the evaluation of this data in the validation subset, the literature model was performing much better than the proposed one. After a thoroughly review of the results, activations and error distribution, it was found that the not only the literature model's predictions but also the data itself was altered to a maximum value for the 10% percentile, presented in Table 6.2.

Therefore, for each project and user story, it parametrized value was defined as maxi-

TABLE 6.1 – Statistical analysis of the main metrics present in the dataset versioned as presented by Choetkiertikul *et al.* (2019).

Project	Repository	Mean	Median	Autocorr.	Count	Std.	Min.	Max.
Mesos	Apache	3.09	3.00	0.18	1680	2.42	1	40
Usergrid	Apache	2.85	3.00	0.12	482	1.40	1	8
Appcelerator Studio	Appcelerator	5.64	5.00	0.26	2919	3.33	1	40
Aptana Studio	Appcelerator	8.02	8.00	0.23	829	5.95	1	40
Titanium SDK/CLI	Appcelerator	6.32	5.00	0.21	2251	5.10	1	34
DuraCloud	DuraSpace	2.13	1.00	0.24	666	2.03	1	16
Bamboo	Atlassian	2.42	2.00	0.15	521	2.14	1	20
Clover	Atlassian	4.59	2.00	0.39	384	6.55	1	40
Jira Software	Atlassian	4.43	3.00	0.35	352	3.51	1	20
Moodle	Moodle	15.54	8.00	0.16	1166	21.65	1	100
Data Management	LSST Corporation	9.57	4.00	0.40	4667	16.60	1	100
Mule	MuleSoft	5.08	5.00	0.17	889	3.50	1	21
Mule Studio	MuleSoft	6.40	5.00	0.08	732	5.39	1	34
Spring XD	Spring	3.70	3.00	0.25	3526	3.23	1	40
Talend Data Quality	TalendForge	5.92	5.00	0.16	1381	5.19	1	40
Talend ESB	TalendForge	2.16	2.00	0.24	868	1.50	1	13

TABLE 6.2 – Statistical analysis of the main metrics present in the clipped dataset.

Project	Maximum Cliped Story point	Count Cliped	Mean	Std.
Mesos	5.00	122	2.75	1.43
Usergrid	5.00	12	2.78	1.19
Appcelerator Studio	8.00	173	5.23	2.20
Aptana Studio	13.00	56	7.30	4.02
Titanium SDK/CLI	13.00	142	5.81	3.63
DuraCloud	4.00	63	1.84	1.05
Bamboo	5.00	17	2.22	1.28
Clover	13.00	17	3.87	3.71
Jira Software	8.00	22	4.00	2.34
Moodle	40.00	57	12.73	12.48
Data Management	21.00	463	6.51	6.43
Mule	8.00	57	4.69	2.62
Mule Studio	13.00	32	5.86	3.63
Spring XD	8.00	107	3.48	2.29
Talend Data Quality	13.00	61	5.51	3.95
Talend ESB	3.00	85	1.89	0.85

mum value between the real value and the clipping constant, as formalized by Equation 6.1.

$$Yr_i = max(Y_i, MCS) \tag{6.1}$$

As a means to easily identify the story points' distribution across all of those presented projects, the Figure 6.2 brings and unnormalized box-plot colored differently for each project. It is noticeable that while Moodle and Data Management have the most elongated distribution with a tail up to one hundred story points, the largest amount of the other projects are located under 20 story points. On the other hand of this specter, DuraCloud, Bamboo and Talend ESB story points are cramped under the value of five.

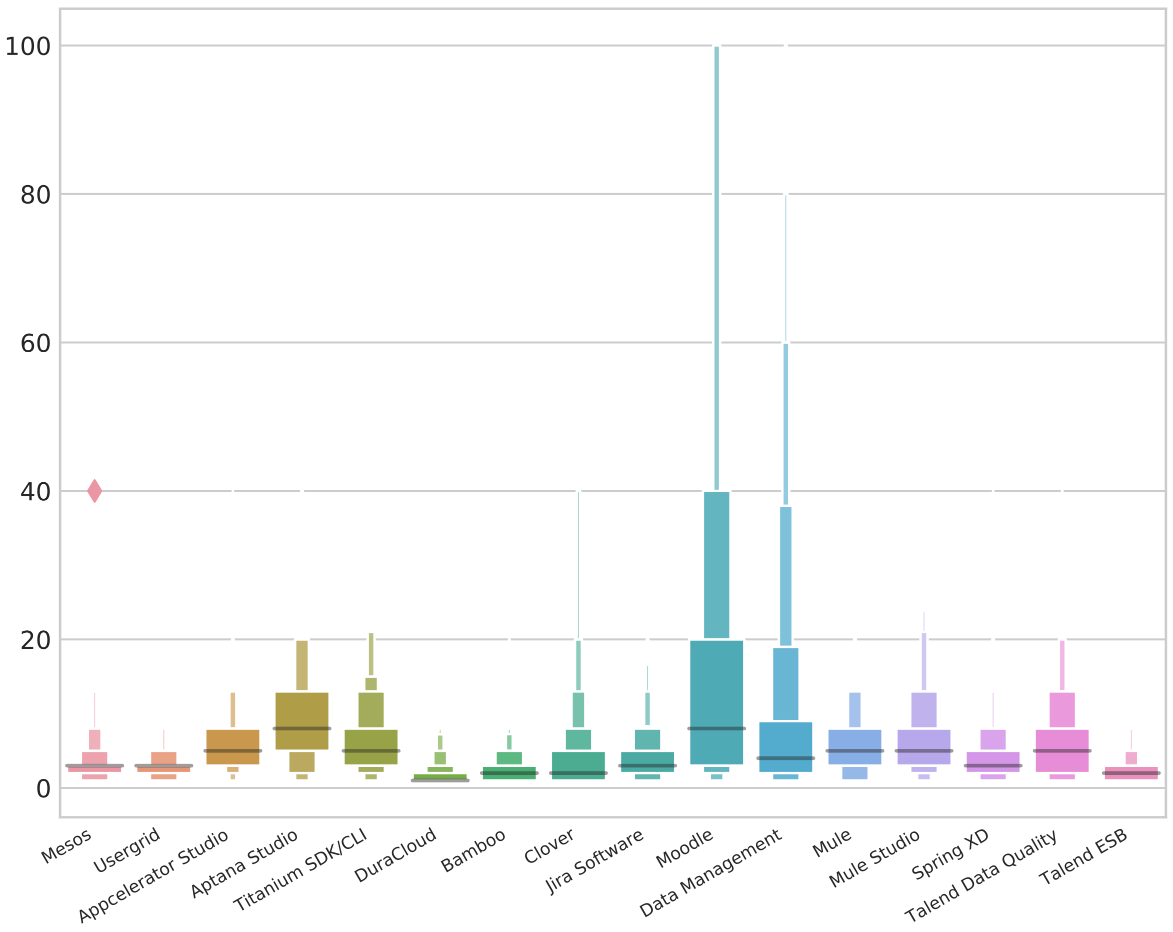


FIGURE 6.2 – Box-plot for the story points distribution in each project.

6.2 Case Study Data

The dataset was initially collected from SQL queries into a Comma Separated Values (CSV) file in which each row of the table represents a data point and each column is delimited by commas. The file consisted of four columns, the first being the number that uniquely identifies the issue, the second is its title, the third is its textual description, and the fifth is the amount of hours actually spent on the task. As ordered by the literature, the user stories are sequentially displaced according to its creation time, that is its sequential ID.

The initial processing of this file is tokenization of strings using the *SpaCy* library in order to properly identify compound words, so a string is transformed into a list of elementary strings, which are the words , eliminating the spaces but preserving the punctuation, as if it were words.

Then all string lists are traversed by counting the number of tokens, storing the result in a dictionary. Finally, IDs are mapped to tokens sequentially from number two in descending order of their count. Thus, words with more occurrence will have smaller IDs, which facilitates later filtering of words with low occurrence. After this step, we have a mapping of words (strings) into numbers (IDs) defined by a dictionary, and the word sequences are converted to a number sequence.

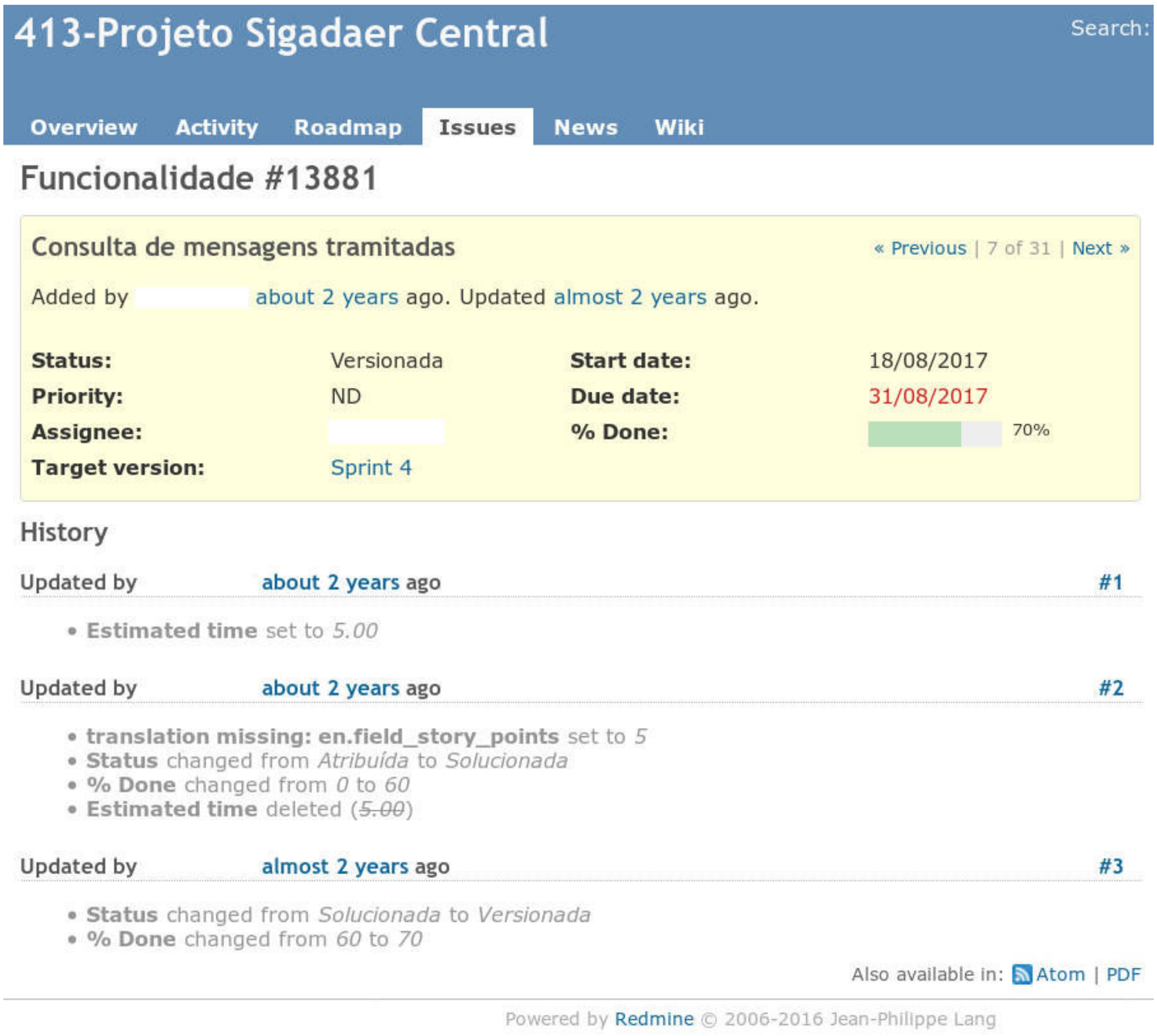
For pre-training, the maximum vocabulary size is defined, ie how many different words can exist in order to limit the size of the input layer. This way, words with an ID above this number will map to an ID equal to zero, which represents all words that are outside the vocabulary.

It was also important to properly escape newline characters and other special punctuation, favoring the UTF-8 encoding. If the wrong encoding were to be employed, the tokenizer could wrongly break accentuated words apart. This would take up more space in the dictionary and potentially produce senseless word embeddings for the broken phonemes.

As depicted in Figure 6.3, the Redmine project management environment is similar to Jira's, presenting the same basic data extracted from the open source projects although its API is different.

For a more detailed analysis and precise visualization of the data, Figure 6.4 shows the histogram of the CCA-SJ story points distribution. A gaussian kernel density estimation is depicted in the dashed yellow line whose fit was made in the log-scale of the real distribution.

It is important to observe that most of the story points are concentrated in integers numbers as expected for the Fibonacci sequence, with the number one being the most common estimation. As such, the spikes in 1, 2, 3, 5, 8, 13 and 21 are noticeable. Nevertheless, there were a significant amount of user stories with fractional value less than one and even intermediate values such as 2.5. This further supports the fact that some of these issues doesn't truly represent and agile framework, but in certain cases, simply tasks measured in hours.



413-Projeto Sigadaer Central

Search:

Overview | Activity | Roadmap | Issues | News | Wiki

Funcionalidade #13881

Consulta de mensagens tramitadas « Previous | 7 of 31 | Next »

Added by [redacted] about 2 years ago. Updated almost 2 years ago.

Status:	Versionada	**Start date:**	18/08/2017
Priority:	ND	**Due date:**	31/08/2017
Assignee:	[redacted]	**% Done:**	70%
Target version:	Sprint 4		

History

Updated by about 2 years ago #1

- **Estimated time** set to *5.00*

Updated by about 2 years ago #2

- **translation missing: en.field_story_points** set to *5*
- **Status** changed from *Atribuída* to *Solucionada*
- **% Done** changed from *0* to *60*
- **Estimated time** deleted (~~*5.00*~~)

Updated by almost 2 years ago #3

- **Status** changed from *Solucionada* to *Versionada*
- **% Done** changed from *60* to *70*

Also available in: Atom | PDF

Powered by Redmine © 2006-2016 Jean-Philippe Lang

FIGURE 6.3 – CCA-SJ Sigadaer Redmine example with an user story.

Another characteristic of this dataset is that it follows a rapidly decaying density function, that resembles a Paretto distribution displaced one unit to the left and clipped to a lower value. However, the its favor over integer values makes its maximum likelihood fit difficult for those common distributions.

Even with this mixture of management techniques, the machine learner is expect to discern between those different data distributions, being able to perform better than random estimators or null models. It is also important to note that the data given to the models were also clipped to 12.5 story points, following the same procedure employed by the literature as those models are unable to estimate high values for their occurrence is very scarce.

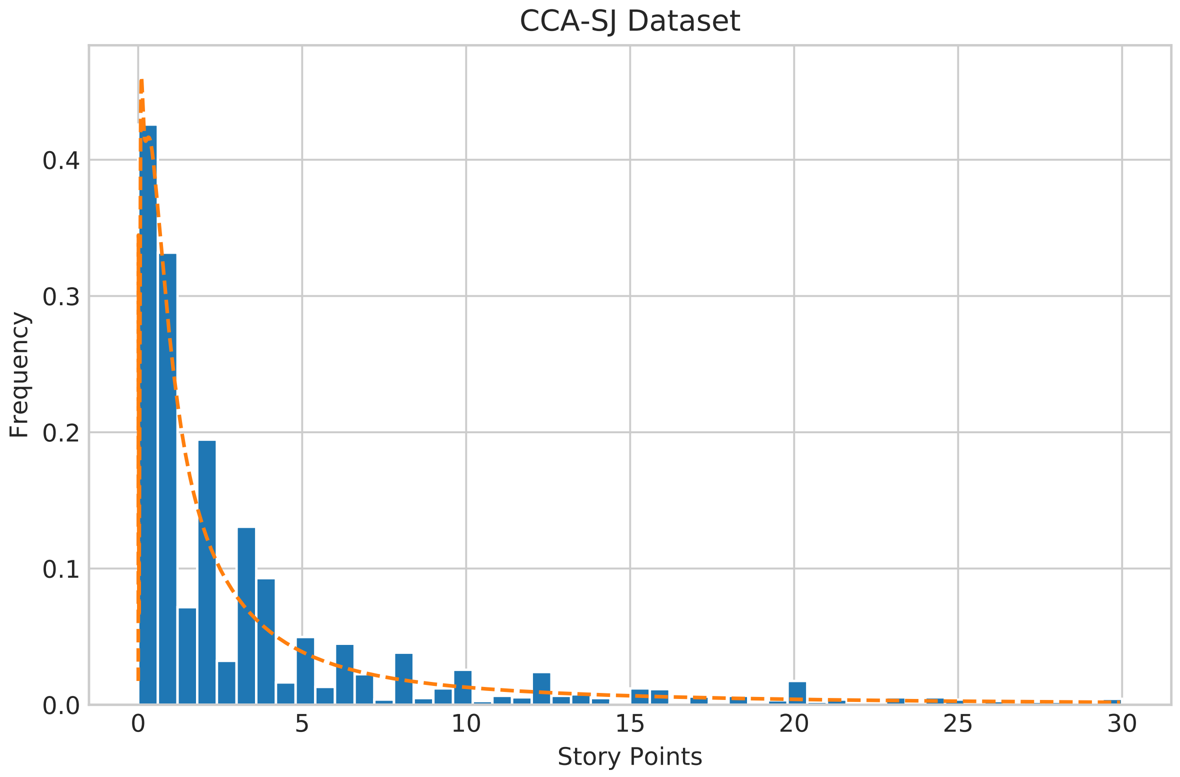


FIGURE 6.4 – CCA-SJ story points distribution.

7 Implementation and Results

In this chapter, implementations details are discussed and the results are analyzed. In the first place, a baseline from the literature was obtained in order to establish a baseline for comparison of the proposed model error metrics. Not only the original results are reported but also the *in loco* reproduction from end-to-end training to evaluation as preprocessed in the Listing 7.1. Other baselines such as a model that randomly sampled the estimated story point from its training distribution or always predicted the mean or the median from the data were also employed.

For the proposed model, a preliminary masking technique was employed in order to implement it in the existing Frameworks without needing to program low level hardware API such as in CUDA or SIMD assembly instructions. Nevertheless, it is imperative as a future work to code in those languages in order to achieve the desired and predicted reduction of training and execution time. The randomly weights' initialization should be also studied especially while using ReLU activation functions as there were many times in which the network "died", that is, the activations of the hidden neurons were zero and the training was unable to continue due to the zero derivatives.

Listing 7.1 – User stories description and title readings from csv file and tokenization.

```
def read_csv_tokenize(file_path, tokenizer):
    data_table = pd.read_csv(file_path)
    titles = data_table.values[:, 1].astype('str')
    seps = np.full(len(titles), '.')
    desc = data_table.values[:, 2].astype('str')
    values = data_table.values[:, 3].astype('float32')
    concat = [a + b + c for a, b, c in zip(titles, seps, desc)]
    tokenized = [[w.text.strip().lower() for w in tokenizer(
        sentence) if not w.text.isspace()] for sentence in concat]
    return tokenized, values
```

7.1 Python Implementation

Guided by a careful benchmark of the available frameworks (SHI *et al.*, 2016) the Theano was choosen (LAMBLIN *et al.*, 2016) having Keras (CHOLLET *et al.*, 2015) as its front-end high-level programming interface. While some of code snippets will be shown as listings, most of the scripts won't be directly reproduced in this dissertation for the sake of its size as all the code developed in this dissertation is fully available at the Github Reposity *gabrui/estimar-pontos-estoria*. As discussed in the Data Chapter, the preprocessing steps for tokenization and dictionary creation is shown in the Listing 7.2 in which the least frequent tokens are discarded in order to produce a lighter embedding and given that training words embeddings with lower than five occurrences is very difficult.

Listing 7.2 – Dictionary generation and vocabulary truncation for the most frequent tokens.

```python
def generate_dict(tokens_list, max_vocab=2000):
    counts = defaultdict(int)
    for tokens in tokens_list:
        for token in tokens:
            counts[token] += 1
    ordered = sorted(counts.items(), key=lambda x: x[1],
                                                   reverse=True)
    IDs = {'EOF_VALUE': 0, 'UNK_VALUE': 1}
    for i in range(min(max_vocab, len(ordered))):
        IDs[ordered[i][0]] = i + 2
    return IDs
```

As described by the previous chapters, a sequential language model was built using a word embedding that was unsupervised trained on user stories or reported issues that had no effort metric associated with them. Every word was expressed in its lowercase format and a dictionary was constructed whose order was given by the frequency. The size of this dictionary was limited to 2000 tokens that included not only words but also punctuations, numbers and two special symbols that denoted an empty or an unknown token. The conversion of those tokens represented as list of strings to a list of integer is shown on the Listing 7.3. The dimension for this embedding was set to 64, a compromise given the modest amount of data available. Following this word vectorization, a 64 LSTM layer was also applied which processed the whole sequence (that was limed to 250 words during training) resulting in a 64 dimensional vector that was then fed to a feedforward sparse layers. The training objective was the mean average error (MAE).

A custom class that extended the base Layer Keras class was programmed. The methods to its initialization, calculation and output shapes were also implemented. Because there is no implementation for the compact sparse matrix multiplication operator, the results were computed by using the canonical matrix multiplication between the masked weights and the activations of the latter layer. Those masked weights were computed as the elementwise multiplication between the dense weight matrix and the adjacency matrix.

Listing 7.3 – Dictionary substitution and train/validation/test split sequentially.

```
def convert_train(token_list, dic, split_ratio, max_len=256):
    data = np.zeros((len(token_list), max_len), dtype='int16')
    for i, tks in enumerate(token_list):
        compri = min(max_len, len(tks))
        data[i, :compri] = [(dic.get(tks[i]) or 1) for i in
                                                  range(compri)]
    train, val, test = quant_split(len(token_list), split_ratio)
    return data[:train], data[train:-test], data[-test:]
```

Therefore, this simple implementation could not compare the gain in training and execution time for using sparse matrix nor the memory usage as a full matrix multiplication was being performed even though most of the elements were just zeros. Nevertheless, if instead of using the existing tensor libraries *for-loops* as shown in the Listings 5.4 or 5.5 were employed for both models the performance difference would likely be noticeable. As a future work, the implementation of those sparse product operations is intended, enabling developers, machine learning practitioners and researchers to easily efficiently this architecture.

7.2 Deep Story Estimation Baseline Comparison

The code available at the Github repository *SEAnalytics/datasets* was employed to reproduce the work from Choetkiertikul *et al.* (2019). It was necessary to beware of newline characters $\backslash n$ as the tokenizer in the preprocess stage will join all the text with this newline character and the call an external Pearl script to split the whole string in a sequences of words: the Moses Tokenizer. As this external resource reads from the stdin and expects each input to be in a single line. Therefore, its is crucial for the correct execution from this code to sanitize all data, replacing the newline character with another whitespace symbol. For the CCA-SJ dataset, a blank space was used to make this replacement.

The results presented in the Table 7.1 are those initially obtained from the literature, exactly as it was reported in the original paper. A more careful analysis reveals that the Mean Classifier (Mean-C) performs much more poorly than initially expected by the standard deviation (Std.) in Table 6.1 as in the case of the Usergrid project whose standard deviation is 1.40 story points but the Mean Classifier reports a Mean Average Error of 1.48. This fact raises some concerns whether the data originally employed in the article is the same available in its code repository.

As expected for the Mean Absolute Error (MAE) metric, the Median Classifier (Median-C) has a lower error than the Mean, as it is a value that mathematically has the lowest MAE from a distribution. Except for the Mule and Mule Studio projects, the Deep Story Estimation model (Deep-SE) is, as stated in its paper, statistically significant better than those baselines.

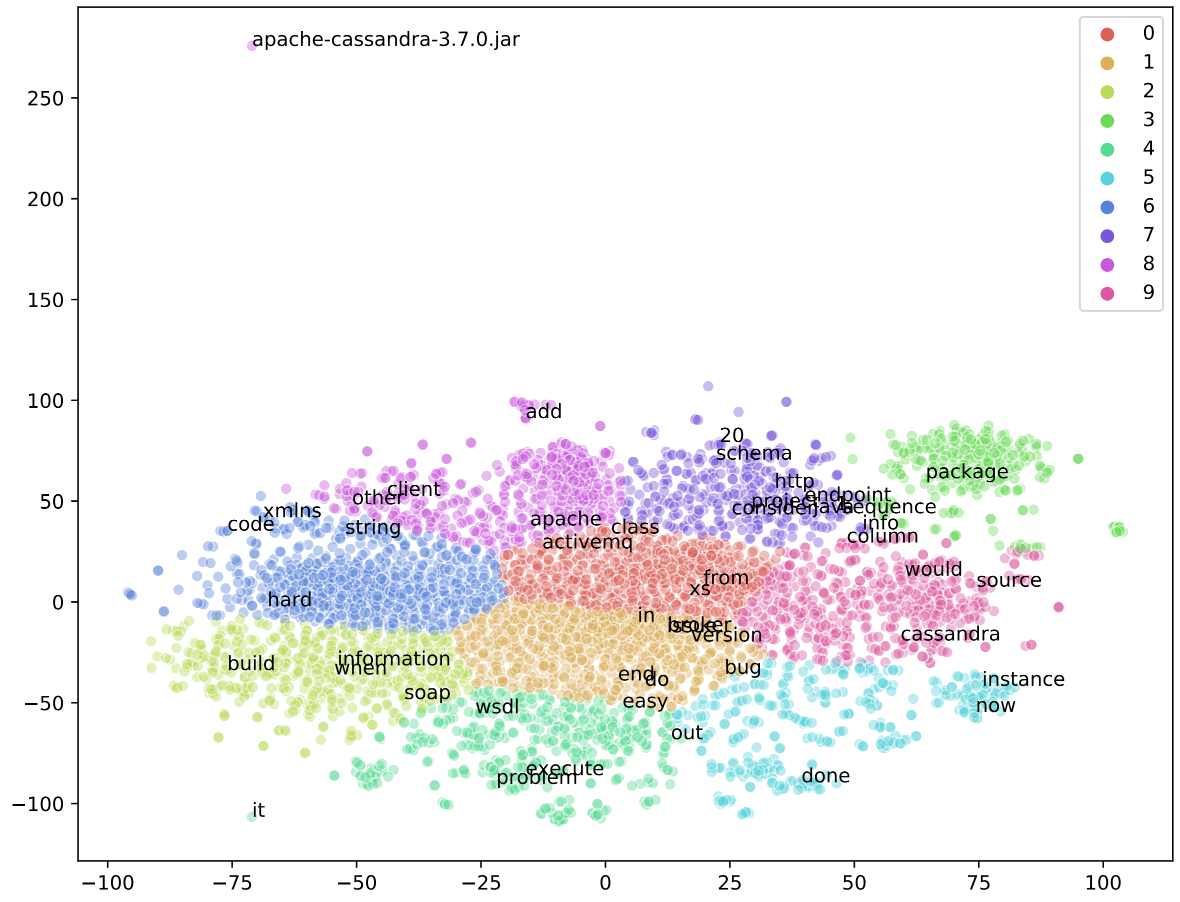


FIGURE 7.1 – t-SNE plot for the originally reported Deep-SE word embedding.

The word embeddings for each open source repository was available in the Deep-SE reproduction package and it was utilized as a starting point for its reproduction as the freshly trained word vectors had a somewhat worse accuracy. A technique for visualizing high dimensional data is the t-SNE, as it tries to preserver clusters as shown on Figure 7.1. Though there are some similar words together as in *code*, *xmlns* and *string*, many words doesn't seen to have a very consistent position in comparison to later works.

Listing 7.4 – Baseline generation for mean, median and random estimators.

```
def baseline_zero(data_file_path, max_value=999999,
                  test_fraction = 0.2):
    story_points = pd.read_csv(data_file_path)['storypoint'].
                      values.clip(0, max_value).astype('int32')
    num_test = int(round(len(story_points) * test_fraction))
    count_train = np.bincount(story_points[:-num_test],
                              minlength=story_points.max())
    count_train = count_train/np.sum(count_train)
    count_test = np.bincount(story_points[-num_test:],
                             minlength=story_points.max())
    count_test = count_test/np.sum(count_test)
    count = np.dot(count_train.reshape((-1,1)),
                   count_test.reshape((1, -1)))
    random_mae = 0
    for i in range(1, count.shape[0]):
        random_mae += i * (count.trace(i) + count.trace(-i))
    median_mae = np.mean(np.abs(np.median(story_points
                 [:-num_test]) - story_points[-num_test:]))
    median_mdae = np.median(np.abs(np.median(story_points
                 [:-num_test]) - story_points[-num_test:]))
    median_sa = (1 - median_mae/random_mae) * 100
    mean_mae = np.mean(np.abs(np.mean(story_points
                 [:-num_test]) - story_points[-num_test:]))
    mean_mdae = np.median(np.abs(np.mean(story_points
                 [:-num_test]) - story_points[-num_test:]))
    mean_sa = (1 - mean_mae/random_mae) * 100
    return [mean_mae, mean_mdae, mean_sa, median_mae,
            median_mdae, median_sa]
```

TABLE 7.1 – Results directly extracted from the Choetkiertikul *et al.* (2019) article.

Project Database	Classifier	Mean AE	Median AE	Gain over Random
Mesos	Deep-SE	1.02	0.73	59.84
Mesos	Mean-C	1.64	1.78	35.61
Mesos	Median-C	1.73	2	32.01
Usergrid	Deep-SE	1.03	0.8	52.66
Usergrid	Mean-C	1.48	1.23	32.13
Usergrid	Median-C	1.6	1	26.29
Appcelerator Studio	Deep-SE	1.36	0.58	60.26
Appcelerator Studio	Mean-C	2.08	1.52	39.02
Appcelerator Studio	Median-C	1.84	1	46.17
Aptana Studio	Deep-SE	2.71	2.52	42.58
Aptana Studio	Mean-C	3.15	3.46	33.3
Aptana Studio	Median-C	3.71	4	21.54
Titanium SDK/CLI	Deep-SE	1.97	1.34	55.92
Titanium SDK/CLI	Mean-C	3.05	1.97	31.59
Titanium SDK/CLI	Median-C	2.47	2	44.65
DuraCloud	Deep-SE	0.68	0.53	69.92
DuraCloud	Mean-C	1.3	1.14	42.88
DuraCloud	Median-C	0.73	1	68.08
Bamboo	Deep-SE	0.74	0.61	71.24
Bamboo	Mean-C	1.75	1.31	32.11
Bamboo	Median-C	1.32	1	48.72
Clover	Deep-SE	2.11	0.8	50.45
Clover	Mean-C	3.49	3.06	17.84
Clover	Median-C	2.84	2	33.33
Jira Software	Deep-SE	1.38	1.09	59.52
Jira Software	Mean-C	2.48	2.15	27.06
Jira Software	Median-C	2.93	2	13.88
Moodle	Deep-SE	5.97	4.93	50.29
Moodle	Mean-C	10.9	12.11	9.16
Moodle	Median-C	7.18	6	40.16
Data Management	Deep-SE	3.77	2.22	47.87
Data Management	Mean-C	5.29	4.55	26.85
Data Management	Median-C	4.82	3	33.38
Mule	Deep-SE	2.18	1.96	40.09
Mule	Mean-C	2.59	2.22	28.82
Mule	Median-C	2.69	2	26.07
Mule Studio	Deep-SE	3.23	1.99	17.17
Mule Studio	Median-C	3.3	2	15.42
Spring XD	Deep-SE	1.63	1.31	46.82
Spring XD	Median-C	2.07	2	32.55
Talend Data Quality	Deep-SE	2.97	2.92	48.28
Talend Data Quality	Median-C	3.87	4	32.43
Talend ESB	Deep-SE	0.64	0.59	69.67
Talend ESB	Mean-C	1.14	0.91	45.86
Talend ESB	Median-C	1.16	1	44.44

7.2.1 Results Reproduction

As a surprise, the reproduced results for the Mean and Median classifiers, reported in the Tables 7.2 and 7.3 respectively, were significantly better than those reported on the Deep-SE paper. With those new results the statistical significance calculated previously could be contested as the distance between the deep learning model and a simpler one is reduced.

TABLE 7.2 – Reproduced results for the Mean Baseline Classifier using the Deep-SE dataset.

Project	Mean MAE	Mean MdAE	Mean SA
Mesos	1.11	0.78	27.78
Usergrid	1.09	0.76	19.48
Appcelerator Studio	1.43	0.28	37.14
Aptana Studio	3.01	2.45	30.80
Titanium SDK/CLI	2.46	2.03	32.25
DuraCloud	0.72	0.87	29.65
Bamboo	0.93	1.00	25.90
Clover	2.80	2.93	21.27
Jira Software	1.67	1.13	31.43
Moodle	10.32	11.14	14.84
Data Management	4.81	4.55	25.12
Mule	2.21	2.77	24.56
Mule Studio	3.24	2.30	22.03
Spring XD	1.89	1.53	22.82
Talend Data Quality	3.93	4.08	11.75
Talend ESB	0.71	0.91	21.66

Except for the Mesos, Aptana Studio and DuraCloud projects, the Median Classifier was superior the the Mean one. This small discrepancy between the statistical prevision of a better Median-C is due to the fact that the training and testing distributions may differ, especially because these sets were constructed by separating the user story creation time.

The reproduced results for the Deep Story Estimation model in the literature's dataset is presented in Table 7.4. Although there are many similar metrics in comparison to the reported in Table 7.1, there are also many discrepancies that overall favors the published paper.

For Mesos, Usergrid, Titanium SDK/CLI and Appcelerator Studio there was a difference only about 0.02 story points but for Aptana Studio the paper was favored by a margin of 0.6, a value that represents about one quarter of the reported error.

For the Bamboo, Talend Data Quality, DuraCloud and Mule Studio projects the re-

TABLE 7.3 – Reproduced results for the Median Baseline Classifier using the Deep-SE dataset.

Project	Median MAE	Median MdAE	Median SA
Mesos	1.12	1.00	27.61
Usergrid	1.03	1.00	23.88
Appcelerator Studio	1.28	0.00	43.98
Aptana Studio	3.08	3.00	29.13
Titanium SDK/CLI	1.95	2.00	46.24
DuraCloud	0.74	1.00	27.72
Bamboo	0.76	1.00	39.44
Clover	2.31	1.00	35.02
Jira Software	1.36	1.00	44.29
Moodle	6.46	5.00	46.71
Data Management	4.28	3.00	33.44
Mule	2.20	3.00	25.17
Mule Studio	3.14	3.00	24.34
Spring XD	1.68	2.00	31.24
Talend Data Quality	3.13	3.00	29.74
Talend ESB	0.70	1.00	23.06

production performed slighter better than the original results, and while this difference was just about 0.06 story points, it represented almost 8% of the Bamboo mean.

The largest mismatch was in the Moodle project in which not only the deep learning model was originally reported to have performed about 2 story points better than the reproduction but also the baseline was distorted. Either some modified dataset was utilized by the authors or some mistake was committed in the original paper while preprocessing the story points. From the perspective of this dissertation, not only the results were double checked and the replication packaged code was utilized, but also a simple inspection on Table 6.2 reveals standard deviations lower than some mean MAE. For instance, the Talend ESB project with its values clipped has a standard deviation of 0.85, while the reported MAE for the mean and median classifiers are about 1.15 in contrast with the 0.71 average absolute error found in the reproduction.

Even though the word embeddings were also distributed pretrained in the Deep-SE repository, the weights for the end-to-end trained neural networks were not. This raises some concerns on the actual reproducibility and verifiability over the originally reported error metrics. Nevertheless, it is known that the random initialization of the weights that an ANN is subject heavily influences its performance as, in the cases of low dimensionality or fewer data points, its optimization heuristic may get stuck in a local minimum. As an example of this phenomena, Figure 7.2 show the evolution for the training and validation error metrics in the Usergrid dataset while depicting a plateau at its final epochs.

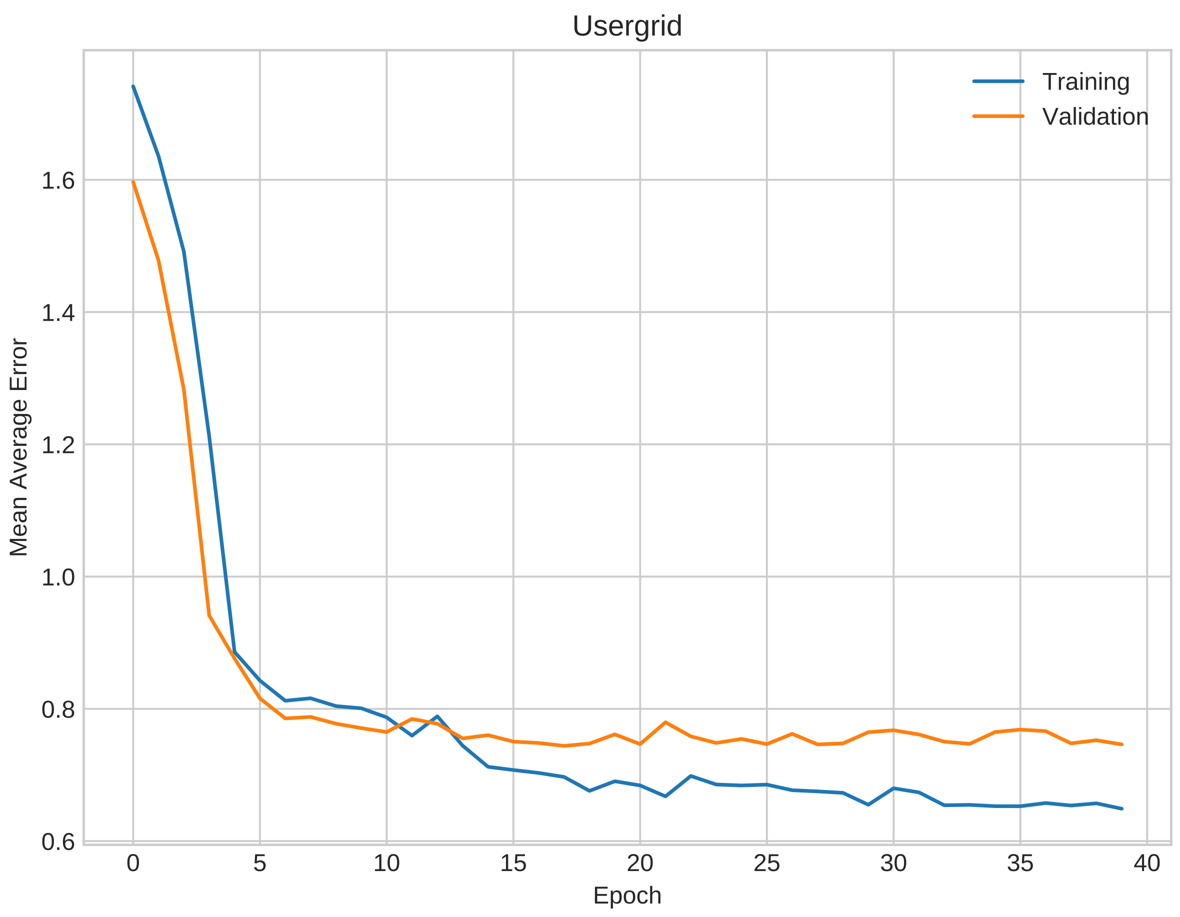


FIGURE 7.2 – Training and validation error progression for the proposed model in the Usergrid dataset.

For the sake of exemplification and visualization of at leas one of these results, Figure 7.3 depicts a regression plot for the DataManagement project from the Atlassian repository. Even though the problem was defined as a regression task, most of the true story points are concentrated in numbers that belongs to the Fibonacci sequence which could be more easily segmented. Nevertheless, the regression gives more flexibility during evaluation and a clearer error derivative during training as a mistake from one to thirteen would be more penalized than if it were just a class cross-entropy error as a difference from the class one to the class two would be the same as the class thirteen.

As a better visualization for the prediction and real values relation, the Figure 7.4 depicts a confusion matrix for the rounded ceiling values.

A more detailed investigation reveals a residual distribution with most of its values within a unitary distance from zero and a longer tail at the negative error, favoring some underestimation as shown in Figure 7.5.

As the sames datasets were used for both the reproduction as well for the novel model, its results are presented on Table 7.5 and its also comparable to the reproduction.

TABLE 7.4 – Reproduced results for the Deep-SE neural model in its own dataset.

Project	Mean Average Error	Median Average Error	Gain over Random
Mesos	1.04	0.74	32.72
Usergrid	1.04	0.98	23.29
Appcelerator Studio	1.40	0.90	38.78
Aptana Studio	3.33	3.32	23.31
Titanium SDK/CLI	2.09	1.71	42.37
DuraCloud	0.61	0.47	40.23
Bamboo	0.82	0.70	34.78
Clover	2.31	1.20	35.10
Jira Software	1.40	1.15	42.53
Moodle	7.96	7.40	34.34
Data Management	3.77	2.12	41.33
Mule	2.43	2.25	17.35
Mule Studio	3.18	2.59	23.48
Spring XD	1.63	1.24	33.27
Talend Data Quality	2.81	2.85	36.90
Talend ESB	0.64	0.59	29.13

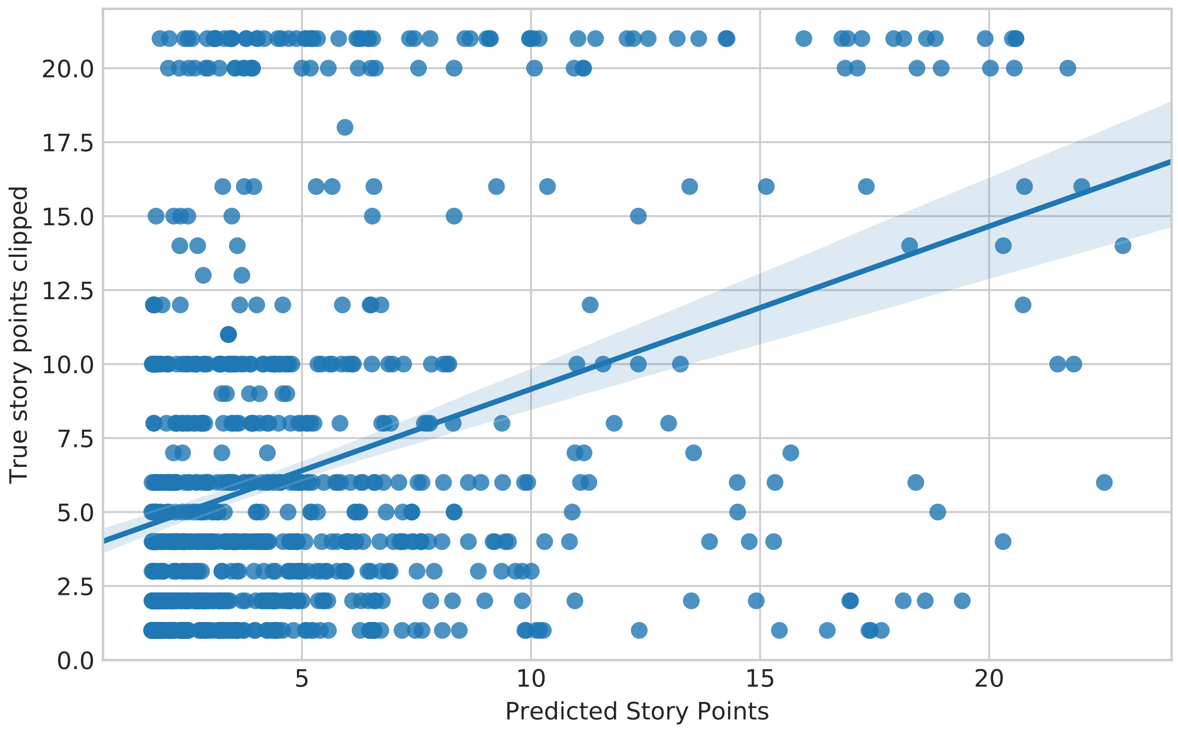


FIGURE 7.3 – Regression plot that compares the neural predicted story points to their real clipped value for Data Management repository.

For the project Mule, Moodle, Bamboo, Titanium SDK/CLI, Aptana Studio, Appcelerator Studio the proposed model performed slightly better, whereas to Mesos, Usergrid, Clover, Spring XD, MuleStudio it performed similarly compared to the reproduced model.

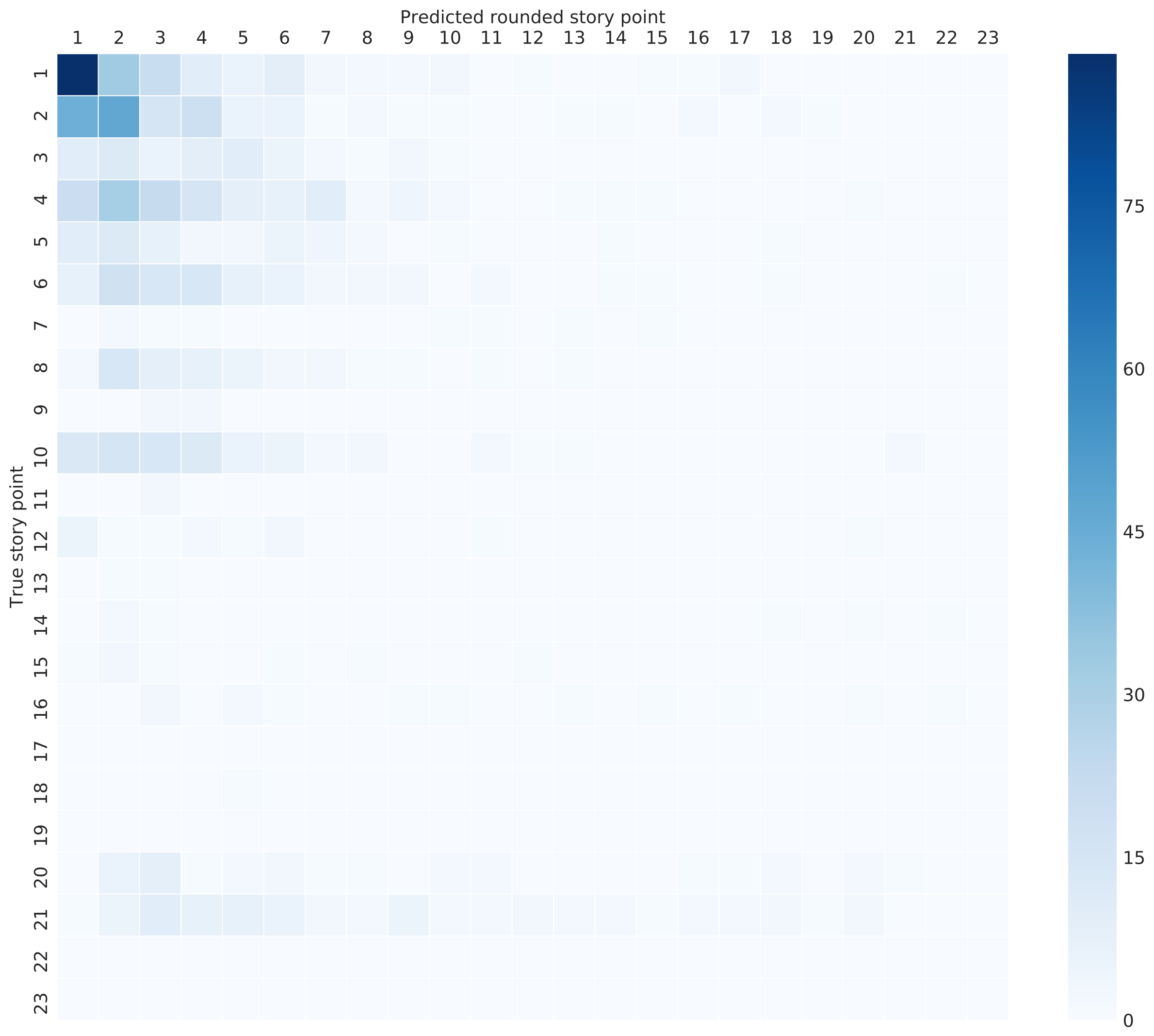


FIGURE 7.4 – Confusion matrix that compares the neural predicted story points to their real clipped value for Data Management repository.

The worst case comparatively was in the DuraCloud, in which the reproduced model had an mean average error of only 0.61 against the 0.74 from the proposed.

This problem was due to the fact that the instances trained on the DuraCloud dataset simply “died”. The activations coming from the inner layers were completely zeroed. This phenomenon occurred during the firsts batches in the first epoch as the backpropagated error decreased the positive weights and the ReLU function saturated in a negative value.

A bi-dimensional t-Distributed Stochastic Neighbor Embedding (t-SNE) plot is shown in Figure 7.6 the 64-dimensional embedding trained on the Apache repository. The points were colored according to ten clusters obtained from the K-Means algorithm applied over the bi-dimensional data for a clearer visualization.

The topmost right cluster numbered as 6 represents XML obtained from the POM (Project Object Model) as Apache Mesos is programmed in Java and many developers

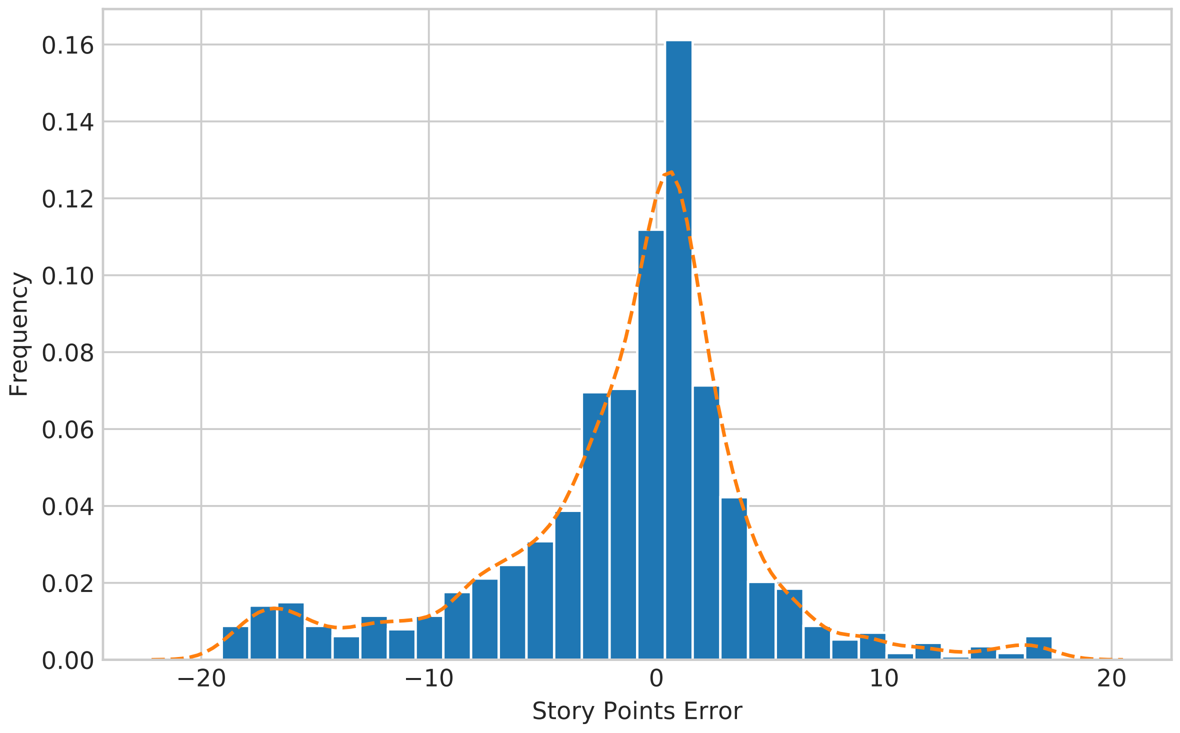


FIGURE 7.5 – Distribution for the prediction errors in the Data Management repository.

TABLE 7.5 – Proposed model results compared in the same dataset.

Project	Mean Average Error	Median Average Error	Gain over Random
Mesos	1.05	0.84	32.07
Usergrid	1.04	1.01	23.08
Appcelerator Studio	1.33	0.14	41.87
Aptana Studio	3.12	3.10	28.09
Titanium SDK/CLI	1.96	2.03	45.87
DuraCloud	0.74	1.00	27.64
Bamboo	0.77	0.96	38.58
Clover	2.35	1.24	33.82
Jira Software	1.42	0.83	41.84
Moodle	7.79	5.41	35.69
Data Management	4.11	2.29	36.12
Mule	2.20	2.99	25.18
Mule Studio	3.22	2.85	22.46
Spring XD	1.67	1.39	31.79
Talend Data Quality	3.18	3.02	28.70
Talend ESB	0.75	0.60	16.57

paste sections of its code in the user stories' description. Other semantic groups such as the zero represents database like definitions, such as *cassandra* (the NoSQL management system) and *column*. Overall, words with similar meaning in fact got euclidean close vectors. More details on this dimensionality reduction technique useful for visualizing

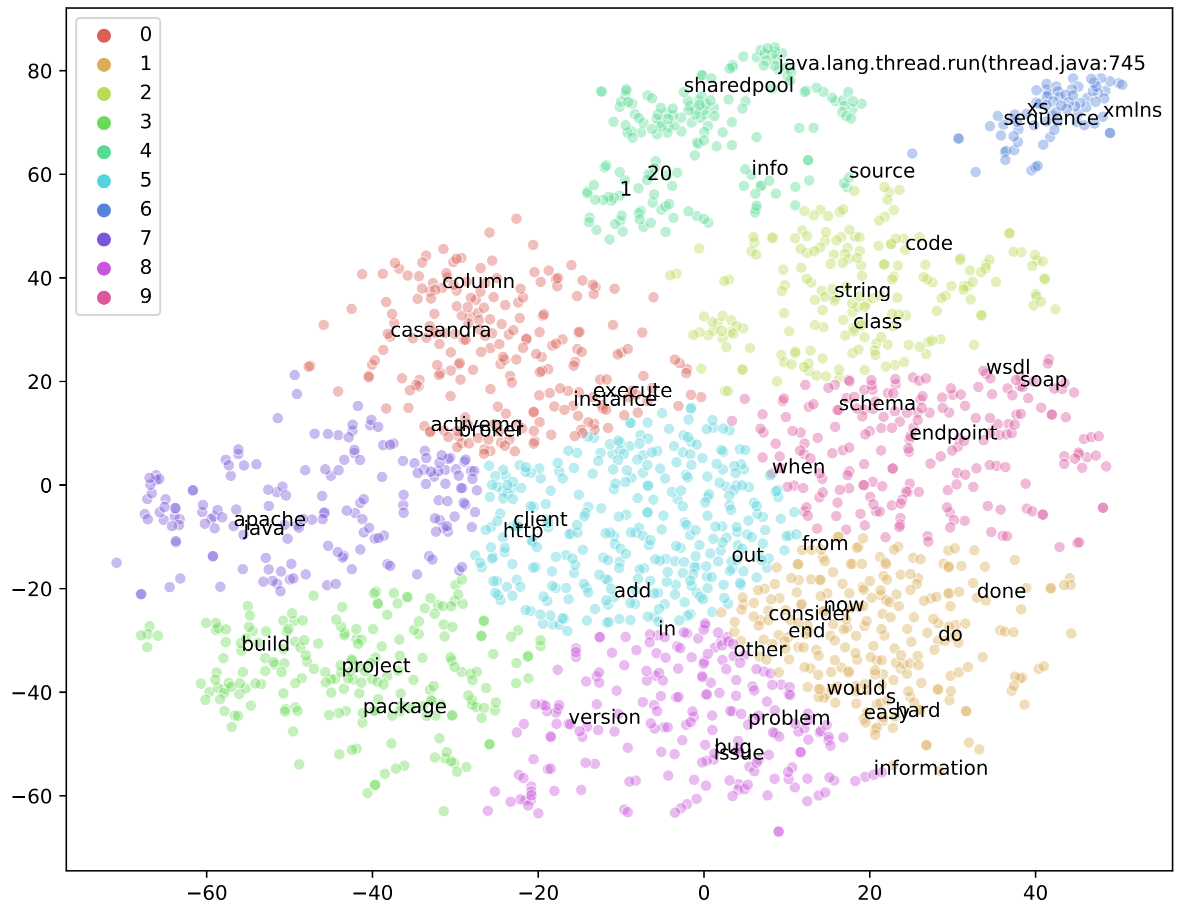


FIGURE 7.6 – Bi-dimensional t-Distributed Stochastic Neighbor Embedding plot for the 64-dimensional embedding trained on the Apache repository.

high-dimensional data is present in Annex A.

7.3 Case Study Results

Similarly to the Apache's repository embeddings, ten clusters have been highlighted using the K-means method on Figure 7.7. It was even more evident the grouping of related words such as the organizations *IFI*, *GAV*, *DIRAP*, *COMAR*, *CCASJ* that were closely related in the 8th cluster. At the topmost right, there is also words that denote code such as *false*, *null*, *error* and English lexemes separated from the Portuguese ones.

At the center, near the null vector, colored as the 9th cluster, there were words considered neutral to the description of a user story such as *quando* (when), *onde* (where), *como* (how), *mesmo* (even). It was remarkable that such vectors that did not convey any specific meaning had a lower absolute value.

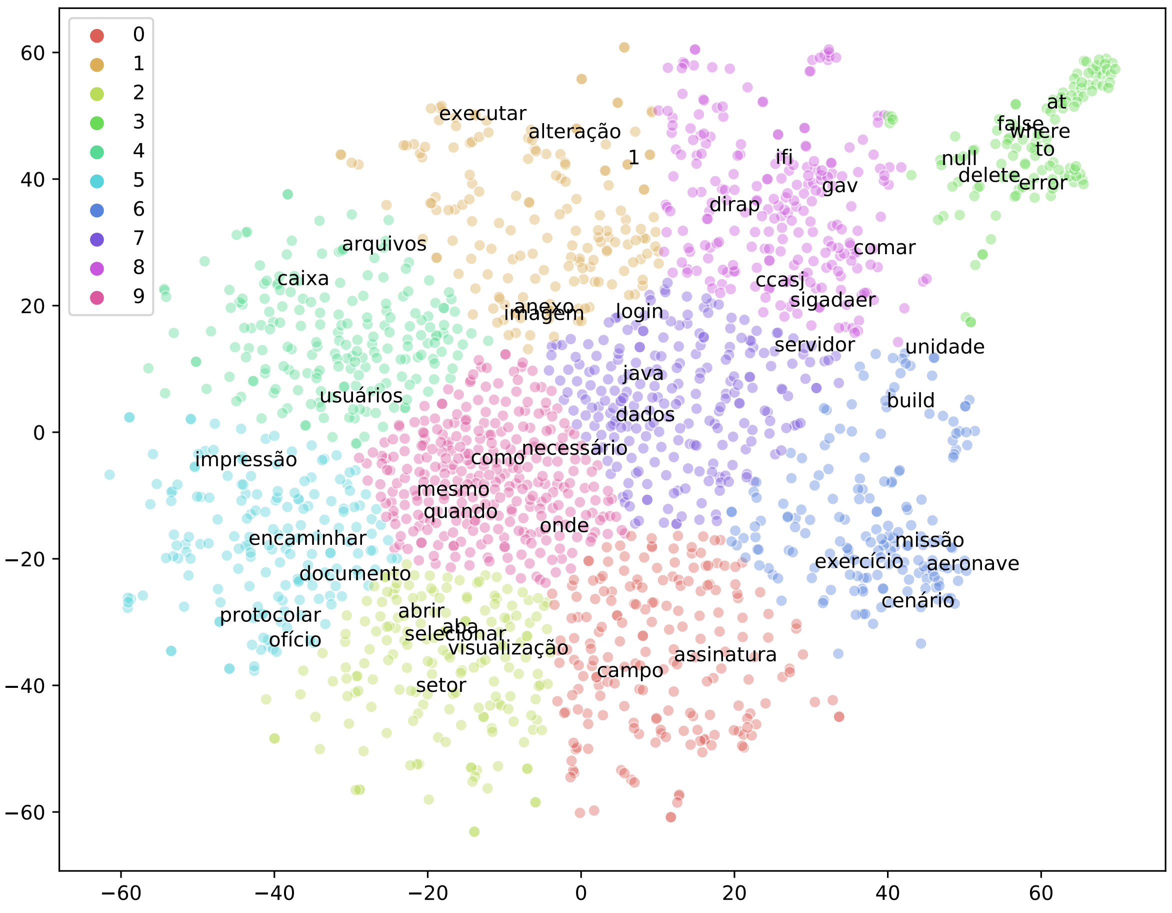


FIGURE 7.7 – Bi-dimensional t-Distributed Stochastic Neighbor Embedding plot for the 64-dimensional embedding trained on the CCA-SJ test dataset.

The word embedding matrix stores each word representation in a line. The number of this line correspond to the word's index in the dictionary. Therefore it is extremely important to preserve the matrix alignment as well as the dictionary enumeration. For instance, during the t-SNE plot elaboration, a bug had occurred in which the dictionary built from one repository was accidentally used in another's embedding, resulting in a random plot for the words' positions. As the problem was resolved, the plot became much more clearer for having similar semantic words grouped together.

Finally, the models reproduced and developed in this chapter are evaluated on the CCA-SJ Dataset, with the results described in Table 7.6. The baseline estimators were significantly surpassed by its deep learning counterparts by a margin approximately of 10%. While the random, the mean and the median classifiers were able to achieve respectively 4.35, 3.70 and 3.60 mean average error in story points, the deep learners were below the 3.35 threshold.

In comparison to the literature Deep Story Estimation, the novel proposed model had a slighter better accuracy. While in terms of the mean average error this difference seems close for just 0.11 story points, the 0.17 value for the median average error is relatively greater. Nevertheless, given the results in the open source dataset, those models are expected to have a somewhat similar accuracy for other projects.

TABLE 7.6 – Evaluation for the CCA-SJ Dataset.

Method	Mean Average Error	Median Average Error	Gain over Random
Random	4.35	4.42	0.00
Mean	3.70	3.07	15.59
Median	3.60	2.00	18.00
Deep-SE	3.34	1.92	23.84
Proposed	3.23	1.75	25.83

It is also important to note the distribution of the residuals shown in Figure 7.8 and its confusion matrix in Figure 7.9 characterizes better the results also found for the literature model.

Those models were unable to predict accurately the high effort story points even when clipped to a 90% threshold. This would require more data as this 10% is scarce for the training and the model tries to learn statistical correlations and does not employ any kind of causal relationship.

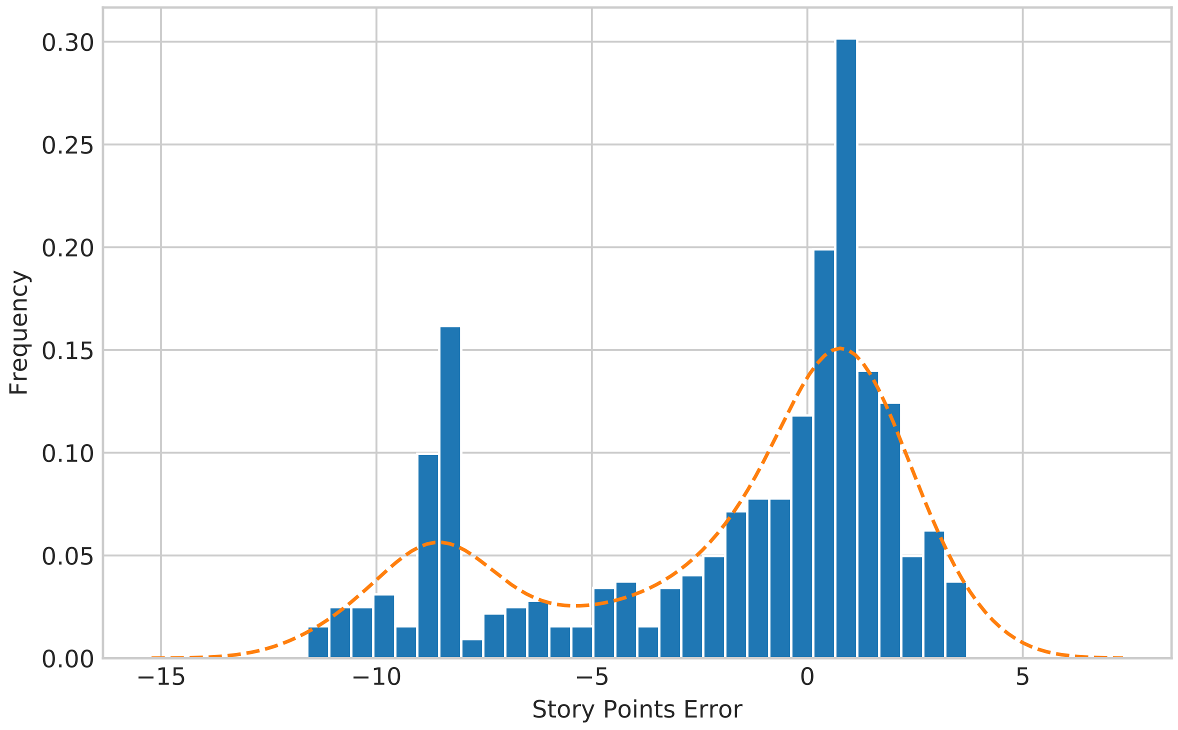


FIGURE 7.8 – Distribution for the prediction errors in the CCA-SJ test dataset.

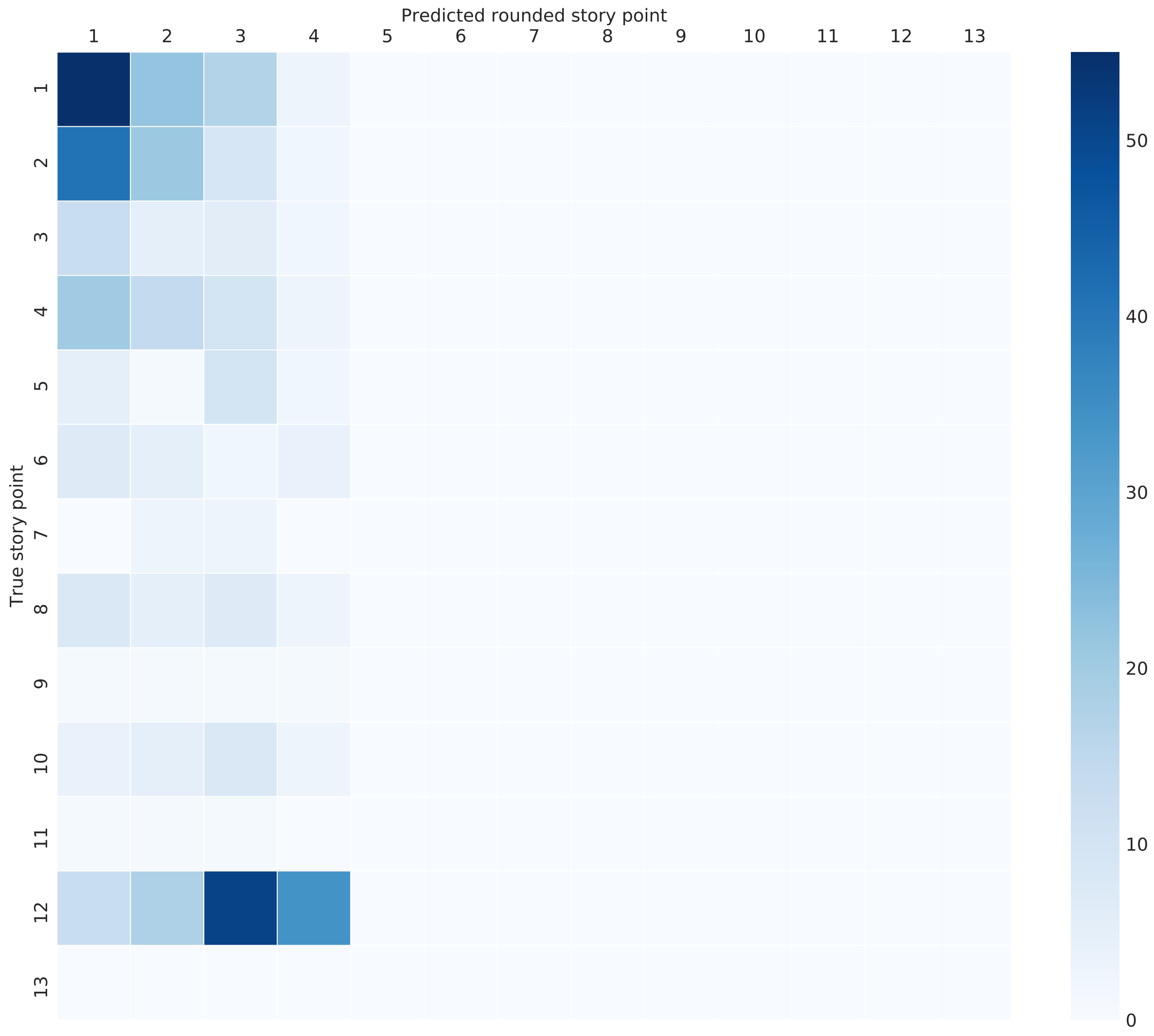


FIGURE 7.9 – Confusion matrix for the proposed model evaluated in the CCA-SJ test dataset.

8 Conclusions

8.1 Work Overview

Deep neural networks architectures were developed in order to estimate story points given the textual input from user stories, a specific application for text regression, that is to extract a quantitative measure from text. Thus, this was a natural language processing task that resembles sentiment analysis as it tries to predict a value associated with its text.

Issue reports and user stories from Open Source projects were utilized for training and comparing results from the literature. Using the Portuguese language, a case study was performed in CCA-SJ, aggregating data from past projects in which the SCRUM methodology wasn't fully deployed. Nevertheless it was possible to extract the effort estimation from the tasks' descriptions.

It was noted that the estimatives are project dependent, and thus, require calibration and retraining in order to use on other projects. This observation is consistent with the fact that distinct teams may have various scales and values for the story points even for similar user stories.

The model represents an improvement over other works by reducing the number of parameters required, namely from Choetkiertikul *et al.* (2019), presenting an statistically significant result in most of the datasets employed that it has a similar accuracy. In the later, it was discovered a data clip that made the results better than reality, as it excluded the high value story points. The inability to differentiate between those models' performances in some projects was mainly due to datasets that had less user stories than the average, namely, fewer than one thousand data points.

8.2 Improvements Outlines

For the scientific community:

- a bibliographic review of the state-of-the-art ANN techniques;
- the sparse architecture proposition for hypercubes;
- improvements in performance over other related works;

The social values that were generated are the following:

- an increased awareness for the AI Culture and its implications for economical development;
- a comprehensive introduction to artificial intelligence and deep learning;
- potential commercial applications for effort analysis from natural language;

For the Brazilian Air Force (FAB):

- a novel tool for CCA-SJ in order to assist agile teams in estimating story points;
- the data aggregation and analysis from past projects;
- personnel qualification in NLP and Deep Learning, with ramifications that may aid the development of automated administrative tools;

8.3 Future Works

The frontier for Artificial Intelligence is expanding in an accelerated pace, with many innovations being published while this dissertation was being developed. Many of those new techniques have rapidly been incorporated in this work, but some open questions remains:

- Utilize the complete GPT-2 model as a the initial layers for encoding: only half of the GTP-2 parameters were released by the time this work was finished. Having more encoding capacity pretrained on a 30GB corpus may increase the model understanding of the natural language resulting in better accuracy.
- Enhance the explainability and interpretability the deep neural network may have as a natural language interface, that is to have a text output explaining to the users the most probable reasons behind the prediction.
- Propose a robust initialization for sparse networks that use the ReLU activation: during the training, there were many runs in which the neurons died, that is, predicted only a constant value.

- Build a dataset focused on experienced SCRUM teams: As CCA-SJ had just implemented this methodology, its usage was still at its infancy when the data was gathered. I the following years, more stable and consistent estimations will be produced by the agile teams, improving the machine learner capability.
- Deploy the model for commercial usage: create a plugin for Jira and/or Redmine and advertise it, empowering an user base that will contribute with data, improving the model.
- Pave the way for causal models: in order to have consistent estimates for user stories with high story points a model not only for the language but also for the “world” is needed, understanding the causality between actions and events. In order words, a more generalist machine learning model is needed.

Bibliography

ALFORD, S.; ROBINETT, R. A.; MILECHIN, L.; KEPNER, J. **Pruned and Structurally Sparse Neural Networks**. Ithaca, NY: The Computing Research Repository (CoRR), 2018. Available at: http://arxiv.org/abs/1810.00299. Accessed on: 14 Mar. 2019.

AMODEI, D.; OLAH, C.; STEINHARDT, J.; CHRISTIANO, P.; SCHULMAN, J.; MANÉ, D. **Concrete Problems in AI Safety**. Ithaca, NY: Cornell University, arXiv, The Computing Research Repository (CoRR), 2016. Available at: https://arxiv.org/pdf/1606.06565.pdf. Accessed on: 10 Jul. 2019.

ARPIT, D.; JASTRZEBSKI, S.; BALLAS, N.; KRUEGER, D.; BENGIO, E.; KANWAL, M. S.; MAHARAJ, T.; FISCHER, A.; COURVILLE, A.; BENGIO, Y.; LACOSTE-JULIEN, S. A closer look at memorization in deep networks. *In*: **Proceedings of the 34th International Conference on Machine Learning - Volume 70**. JMLR.org, 2017. (ICML'17), p. 233–242. Available at: http://dl.acm.org/citation.cfm?id=3305381.3305406. Accessed on: 08 May 2019.

BAKER, B.; KANITSCHEIDER, I.; MARKOV, T.; WU, Y.; POWELL, G.; MCGREW, B.; MORDATCH, I. **Emergent Tool Use From Multi-Agent Autocurricula**. Ithaca, NY: Cornell University, arXiv, The Computing Research Repository (CoRR), 2019. Available at: http://arxiv.org/abs/1909.07528. Accessed on: 01 Oct. 2019.

BECK, K.; BEEDLE, M.; BENNEKUM, A. van; COCKBURN, A.; CUNNINGHAM, W.; FOWLER, M.; GRENNING, J.; HIGHSMITH, J.; HUNT, A.; JEFFRIES, R.; KERN, J.; MARICK, B.; MARTIN, R. C.; MELLOR, S.; SCHWABER, K.; SUTHERLAND, J.; THOMAS, D. **Manifesto for Agile Software Development**. 2001. Available at: http://www.agilemanifesto.org/. Accessed on: 25 Feb. 2019.

BISHOP, C. M. **Neural Networks for Pattern Recognition**. Oxford, England: Oxford University Press, 1995.

BOJANOWSKI, P.; GRAVE, E.; JOULIN, A.; MIKOLOV, T. **Enriching Word Vectors with Subword Information**. Ithaca, NY: Cornell University, arXiv, The Computing Research Repository (CoRR), 2016. Available at: https://arxiv.org/pdf/1607.04606.pdf. Accessed on: 2 Jul. 2019.

BOURELY, A.; BOUERI, J. P.; CHOROMONSKI, K. **Sparse Neural Networks Topologies**. Ithaca, NY: Cornell University, arXiv, The Computing Research Repository (CoRR), 2017. Available at: http://arxiv.org/abs/1706.05683. Accessed on: 09 Mar. 2019.

CHEN, T. Q.; RUBANOVA, Y.; BETTENCOURT, J.; DUVENAUD, D. K. Neural ordinary differential equations. Curran Associates, p. 6571–6583, 2018. Available at: http://papers.nips.cc/paper/7892-neural-ordinary-differential-equations.pdf. Accessed on: 11 Jul. 2018.

CHEN, W.; WILSON, J. T.; TYREE, S.; WEINBERGER, K. Q.; CHEN, Y. Compressing neural networks with the hashing trick. *In*: **Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37**. JMLR.org, 2015. (ICML'15), p. 2285–2294. Available at: http://dl.acm.org/citation.cfm?id=3045118.3045361. Accessed on: 22 Mar. 2019.

CHOETKIERTIKUL, M.; DAM, H. K.; TRAN, T.; GHOSE, A. Predicting the delay of issues with due dates in software projects. **Empirical Software Engineering**, v. 22, n. 3, p. 1223–1263, Jun 2017. Available at: https://doi.org/10.1007/s10664-016-9496-7. Accessed on: 11 Aug. 2019.

CHOETKIERTIKUL, M.; DAM, H. K.; TRAN, T.; GHOSE, A.; GRUNDY, J. Predicting delivery capability in iterative software development. **IEEE Transactions on Software Engineering**, v. 44, n. 6, p. 551–573, June 2018.

CHOETKIERTIKUL, M.; DAM, H. K.; TRAN, T.; PHAM, T.; GHOSE, A.; MENZIES, T. A deep learning model for estimating story points. **IEEE Transactions on Software Engineering**, v. 45, n. 7, p. 637–656, July 2019.

CHOLLET, F.; RAHMAN, F.; LEE, T.; MARMIESSE, G. de; ZABLUDA, O.; PUMPERLA, M.; SANTANA, E.; MCCOLGAN, T.; SNELGROVE, X.; BRANCHAUD-CHARRON, F.; OLIVER, M.; VIJAY, P. **Keras**. 2015. Available at: https://keras.io.

CHUNG, J.; GÜLÇEHRE, Ç.; CHO, K.; BENGIO, Y. **Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling**. Ithaca, NY: Cornell University, arXiv, The Computing Research Repository (CoRR), 2014. Presented at the Deep Learning workshop at NIPS2014. Available at: https://arxiv.org/abs/1412.3555. Accessed on: 05 Jan. 2018.

DENG, L.; LIU, Y. **Deep Learning in Natural Language Processing**. Singapore, Singapore: Springer Nature, 2018.

DOD. **SUMMARY OF THE 2018 DEPARTMENT OF DEFENSE ARTIFICIAL INTELLIGENCE STRATEGY: Harnessing AI to Advance Our Security and Prosperity**. DEPARTMENT OF DEFENSE, Defense Pentagon, Washington DC, USA, February 2019. Available at: https://media.defense.gov/2019-/Feb/12/2002088963/-1/-1/1/SUMMARY-OF-DOD-AI-STRATEGY.PDF. Accessed on: 14 Apr. 2019.

ELIZONDO, D. A.; FIESLER, E. A survey of partially connected neural networks. **International journal of neural systems**, v. 8 (5–6), p. 535–58, 1997.

EPELBAUM, T. **Deep learning: Technical introduction**. Ivry-sur-Seine, France: [*s.n.*], 2017. ArXiv:1709.01412.

ERDOGAN, O.; PEKKAYA, M. E.; GOK, H. More effective sprint retrospective with statistical analysis. **Journal of Software: Evolution and Process**, v. 30, n. 5, p. e1933, 2018. E1933 JSME-17-0071.R1. Available at: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.1933. Accessed on: 14 May 2019.

FAN, Y.; FELIU-FABÀ, J.; LIN, L.; YING, L.; ZEPEDA-NÚÑEZ, L. A multiscale neural network based on hierarchical nested bases. **Research in the Mathematical Sciences**, v. 6, n. 2, p. 21, Mar 2019. Available at: https://doi.org/10.1007/s40687-019-0183-3. Accessed on: 28 Mar. 2019.

FAN, Y.; LIN, L.; YING, L.; ZEPEDA-NUNEZ, L. A multiscale neural network based on hierarchical matrices. **arXiv e-prints**, p. arXiv:1807.01883, Jul 2018.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. Cambridge, Massachusetts: MIT Press, 2016. Available at: http://www.deeplearningbook.org. Accessed on: 22 Feb. 2018.

GRAVE, E.; BOJANOWSKI, P.; GUPTA, P.; JOULIN, A.; MIKOLOV, T. Learning word vectors for 157 languages. *In*: **Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018)**. Miyazaki, Japan: European Languages Resources Association (ELRA), 2018. Available at: https://www.aclweb.org/anthology/L18-1550. Accessed on: 05 May 2019.

GUTMANN, M.; HYVäRINEN, A. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. *In*: TEH, Y. W.; TITTERINGTON, M. (Ed.). **Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics**. Chia Laguna Resort, Sardinia, Italy: PMLR, 2010. (Proceedings of Machine Learning Research, v. 9), p. 297–304. Available at: http://proceedings.mlr.press/v9/gutmann10a.html. Accessed on: 06 May 2019.

HAN, S.; POOL, J.; TRAN, J.; DALLY, W. J. Learning both weights and connections for efficient neural networks. *In*: **Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1**. Cambridge, MA, USA: MIT Press, 2015. (NIPS'15), p. 1135–1143. Available at: http://dl.acm.org/citation.cfm?id=2969239.2969366. Accessed on: 15 Mar. 2019.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. **Neural Comput.**, MIT Press, Cambridge, MA, USA, v. 9, n. 8, p. 1735–1780, nov. 1997. Available at: http://dx.doi.org/10.1162/neco.1997.9.8.1735. Accessed on: 10 Jul. 2018.

IANDOLA, F. N.; MOSKEWICZ, M. W.; ASHRAF, K.; HAN, S.; DALLY, W. J.; KEUTZER, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. **The Computing Research Repository (CoRR)**, abs/1602.07360, 2016. Available at: http://arxiv.org/abs/1602.07360. Accessed on: 13 Mar. 2019.

ISAKOV, M.; KINSY, M. A. Closnets: a priori sparse topologies for faster DNN training. **The Computing Research Repository (CoRR)**, abs/1802.03885, 2018. Available at: http://arxiv.org/abs/1802.03885. Accessed on: 11 Mar. 2019.

JAPAN. **Artificial Intelligence Technology Strategy**. Strategic Council for AI Technology, 2017. Available at: https://www.nedo.go.jp/content/100865202.pdf. Accessed on: 15 Apr. 2019.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *In*: **Proceedings of the 3rd International Conference on Learning Representations (ICLR)**. [*S.l.*: *s.n.*], 2015.

KRIESEL, D. **A Brief Introduction to Neural Networks**. New York, NY: Cornell University, 2007. Available at: http://www.dkriesel.com/en/science/neural_networks. Accessed on: 12 Jan. 2019.

LAMBLIN, P.; AL-RFOU, R.; ALAIN, G.; ALMAHAIRI, A.; ANGERMUELLER, C.; BAHDANAU, D.; BALLAS, N.; BASTIEN, F.; BAYER, J.; BELIKOV, A.; BELOPOLSKY, A.; BENGIO, Y.; BERGERON, A.; BERGSTRA, J.; BISSON, V.; Bleecher Snyder, J.; BOUCHARD, N.; BOULANGER-LEWANDOWSKI, N.; BOUTHILLIER, X.; BRÉBISSON, A. de; BREULEUX, O.; CARRIER, P.-L.; CHO, K.; CHOROWSKI, J.; CHRISTIANO, P.; COOIJMANS, T.; CÔTÉ, M.-A.; CÔTÉ, M.; COURVILLE, A.; DAUPHIN, Y. N.; DELALLEAU, O.; DEMOUTH, J.; DESJARDINS, G.; DIELEMAN, S.; DINH, L.; DUCOFFE, M.; DUMOULIN, V.; Ebrahimi Kahou, S.; ERHAN, D.; FAN, Z.; FIRAT, O.; GERMAIN, M.; GLOROT, X.; GOODFELLOW, I.; GRAHAM, M.; GULCEHRE, C.; HAMEL, P.; HARLOUCHET, I.; HENG, J.-P.; HIDASI, B.; HONARI, S.; JAIN, A.; JEAN, S.; JIA, K.; KOROBOV, M.; KULKARNI, V.; LAMB, A.; LARSEN, E.; LAURENT, C.; LEE, S.; LEFRANCOIS, S.; LEMIEUX, S.; LÉONARD, N.; LIN, Z.; LIVEZEY, J. A.; LORENZ, C.; LOWIN, J.; MA, Q.; MANZAGOL, P.-A.; MASTROPIETRO, O.; MCGIBBON, R. T.; MEMISEVIC, R.; MERRIËNBOER, B. van; MICHALSKI, V.; MIRZA, M.; ORLANDI, A.; PAL, C.; PASCANU, R.; PEZESHKI, M.; RAFFEL, C.; RENSHAW, D.; ROCKLIN, M.; ROMERO, A.; ROTH, M.; SADOWSKI, P.; SALVATIER, J.; SAVARD, F.; SCHLÜTER, J.; SCHULMAN, J.; SCHWARTZ, G.; SERBAN, I. V.; SERDYUK, D.; SHABANIAN, S.; SIMON, E.; SPIECKERMANN, S.; SUBRAMANYAM, S. R.; SYGNOWSKI, J.; TANGUAY, J.; TULDER, G. van; TURIAN, J.; URBAN, S.; VINCENT, P.; VISIN, F.; VRIES, H. de; WARDE-FARLEY, D.; WEBB, D. J.; WILLSON, M.; XU, K.; XUE, L.; YAO, L.; ZHANG, S.; ZHANG, Y. Theano: A Python framework for fast computation of mathematical expressions. **arXiv e-prints**, abs/1605.02688, maio 2016. Available at: http://arxiv.org/abs/1605.02688. Accessed on: 14 Feb. 2018.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature**, v. 521, p. 436, May 2015. Available at: https://doi.org/10.1038/nature14539. Accessed on: 07 Jun. 2018.

LEIKE, J.; MARTIC, M.; KRAKOVNA, V.; ORTEGA, P. A.; EVERITT, T.; LEFRANCQ, A.; ORSEAU, L.; LEGG, S. **AI Safety Gridworlds**. Ithaca, NY: Cornell University, arXiv, The Computing Research Repository (CoRR), 2017. Available at: http://arxiv.org/abs/1711.09883. Accessed on: 10 Jul. 2019.

LI, X.; JIANG, H.; REN, Z.; LI, G.; ZHANG, J. Deep Learning in Software Engineering. **arXiv e-prints**, p. arXiv:1805.04825, May 2018.

LIN, J.; RAO, Y.; LU, J.; ZHOU, J. Runtime neural pruning. *In*: GUYON, I.; LUXBURG, U. V.; BENGIO, S.; WALLACH, H.; FERGUS, R.; VISHWANATHAN, S.; GARNETT, R. (Ed.). **Advances in Neural Information Processing Systems 30**. Curran Associates, 2017. p. 2181–2191. Available at: http://papers.nips.cc/paper/6813-runtime-neural-pruning.pdf. Accessed on: 14 Mar. 2019.

LIU, B. **Sentiment Analysis and Opinion Mining**. San Rafael, California: Morgan & Claypool Publisher, 2012.

LOPER, E.; BIRD, S. Nltk: The natural language toolkit. *In*: **Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1**. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002. (ETMTNLP '02), p. 63–70. Available at: https://doi.org/10.3115/1118108.1118117. Accessed on: 10 Sept. 2019.

MAATEN, L. van der; HINTON, G. Visualizing data using t-SNE. **Journal of Machine Learning Research**, v. 9, p. 2579–2605, 2008. Available at: http://www.jmlr.org/papers/v9/vandermaaten08a.html. Accessed on: 17 Aug. 2019.

MANNING, C. D.; SCHüTZE, H. **Foundations of Statistical Natural Language Processing**. Cambridge, MA, USA: MIT Press, 1999. ISBN 0-262-13360-1.

MEHTA, P.; SCHWAB, D. J. **An exact mapping between the Variational Renormalization Group and Deep Learning**. Ithaca, NY: Cornell University, arXiv, 2014. Available at: http://arxiv.org/abs/1410.3831. Accessed on: 15 Mar. 2019.

MELO, G. A. de. **Utilização de aprendizado profundo para estimar esforço de desenvolvimento de software**. 103 p. Monografia (monography) — Instituto Tecnológico de Aeronáutica (ITA), Trabalho de Conclusão de Curso (Graduação em Engenharia de Computação), São José dos Campos, 2019.

MELO, G. A. de; NETO, L. B. da C.; OLIVEIRA, G. S. N. de; CAMARGO, W. P. de; TOSO, G. D.; NASCIMENTO, M. A. do. A corrida armamentista pela inteligência artificial. *In*: **Anais do 16º Congresso Acadêmico sobre Defesa Nacional (CADN)**. Rio de Janeiro, RJ: Escola Naval, 2019. Available at: https://www.defesa.gov.br/arquivos/ensino_e_pesquisa/defesa_academia/cadn/artigos-/xvi_cadn/a_corrida_armamentista_pela_inteligencia_artificial.pdf. Accessed on: 30 Aug. 2019.

MELO, G. A. de; SUGIMOTO, D. N.; TASINAFFO, P. M.; SANTOS, A. H. M.; CUNHA, A. M. da; DIAS, L. A. V. A new approach to river flow forecasting: LSTM and GRU multivariate models. **IEEE Latin America Transactions**, 2019, no prelo.

MELO, G. A. de; TASINAFFO, P. M. Gated recurrent unit hierarchical architecture forfundamental stock analysis and forecast. *In*: **Workshop of Artificial Intelligence Applied to Finance (WAIAF 2019)**. São José dos Campos, SP: Instituto Tecnológico de Aeronáutica, 2019. Available at: http://www.comp.ita.br/labsca/waiaf/papers/GabrielMelo_paper_13.pdf. Accessed on: 30 Jul. 2019.

MIKOLOV, T.; CHEN, K.; CORRADO, G.; DEAN, J. Efficient estimation of word representations in vector space. *In*: BENGIO, Y.; LECUN, Y. (Ed.). **1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings**. [*s.n.*], 2013. Available at: http://arxiv.org/abs/1301.3781. Accessed on: 05 May 2019.

MITKOV, R. **The Oxford Handbook of Computational Linguistics (Oxford Handbooks in Linguistics S.)**. New York, NY, USA: Oxford University Press, 2003. ISBN 0198238827.

MNIH, A.; KAVUKCUOGLU, K. Learning word embeddings efficiently with noise-contrastive estimation. *In*: **Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2**. USA: Curran Associates Inc., 2013. (NIPS'13), p. 2265–2273. Available at: http://dl.acm.org/citation.cfm?id=2999792.2999865. Accessed on: 06 May 2019.

NG, A. Y.-T. **Machine Learning Yearning**. San Francisco, California: DeepLearning AI, 2017. Available at: https://www.deeplearning.ai/machine-learning-yearning/. Accessed on: 06 Feb. 2019.

NG, A. Y.-T. **AI Transformation Playbook: How to lead your company into the AI era**. San Francisco, California: Landing AI, 2018. Available at: https://www.deeplearning.ai/machine-learning-yearning/. Accessed on: 06 Feb. 2019.

NIELSEN, M. A. **Neural Networks and Deep Learning**. San Francisco, California: Determination Press, 2015. Available at: http://www.neuralnetworksanddeeplearning.com. Accessed on: 21 Feb. 2018.

NIRKIN, Y.; KELLER, Y.; HASSNER, T. FSGAN: Subject agnostic face swapping and reenactment. *In*: **International Conference on Computer Vision (ICCV)**. Seul, South Korea: [*s.n.*], 2019.

PANDA, A.; SATAPATHY, S. M.; RATH, S. K. Empirical validation of neural network models for agile software effort estimation based on story points. **Procedia Computer Science**, v. 57, p. 772 – 781, 2015. 3rd International Conference on Recent Trends in Computing 2015 (ICRTC-2015). Available at: http://www.sciencedirect.com/science/article/pii/S1877050915020037. Accessed on: 16 May 2019.

PAYNE, C. **MuseNet**. Apr. 2019. Available at: http://openai.com/blog/musenet. Accessed on: 06 May 2019.

PEARL, J. The seven tools of causal inference, with reflections on machine learning. **Communications of the ACM**, v. 62, n. 3, p. 54 – 60, 2019.

PORRU, S.; MURGIA, A.; DEMEYER, S.; MARCHESI, M.; TONELLI, R. Estimating story points from issue reports. *In*: **Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering**. New York, NY, USA: ACM, 2016. (PROMISE 2016), p. 2:1–2:10. ISBN 978-1-4503-4772-3. Available at: http://doi.acm.org/10.1145/2972958.2972959. Accessed on: 17 May 2019.

RADFORD, A.; WU, J.; CHILD, R.; LUAN, D.; AMODEI, D.; SUTSKEVER, I. Language models are unsupervised multitask learners. 2018. Available at: https://openai.com/blog/better-language-models/. Accessed on: 20 Aug 2019.

ROBINETT, R. A.; KEPNER, J. Neural network topologies for sparse training. **The Computing Research Repository (CoRR)**, abs/1809.05242, 2018. Available at: http://arxiv.org/abs/1809.05242. Accessed on: 18 Mar. 2019.

ROBINETT, R. A.; KEPNER, J. Radix-net: Structured sparse matrices for deep neural networks. **The Computing Research Repository (CoRR)**, abs/1905.00416, 2019. Available at: http://arxiv.org/abs/1905.00416. Accessed on: 11 Mar. 2019.

RUBIN, K. S. **Essential Scrum: a practical guide to the most popular agile process.** Upper Saddle River, New Jersey: Addison-Wesley, 2012. ISBN 9780137043293.

RUSSELL, S.; NORVIG, P. **Artificial Intelligence: A Modern Approach**. 3rd. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009. ISBN 0136042597, 9780136042594.

SATAPATHY, S. M.; RATH, S. K. Empirical assessment of machine learning models for agile software development effort estimation using story points. **Innovations in Systems and Software Engineering**, v. 13, n. 2, p. 191–200, Sep 2017. Available at: https://doi.org/10.1007/s11334-017-0288-z. Accessed on: 11 Aug. 2019.

SCHMIDHUBER, J. Deep learning in neural networks: An overview. **Neural Networks**, v. 61, p. 85 – 117, 2015. Available at: http://www.sciencedirect.com/science/article/pii/S0893608014002135. Accessed on: 10 Jul. 2018.

SHANAHAN, L. J. **Disruption in UAS: The Algorithmic Warfare Cross-Functional Team (Project Maven)**. Department of Defense (DoD), United States, 2018. Available at: http://airpower.airforce.gov.au/APDC/media/Events-Media-/RAAF%20AP%20CONF%202018/1130-1200-Shanahan-Disruption-in-UAS-The-AWCFT.pdf. Accessed on: 15 Apr. 2019.

SHI, S.; WANG, Q.; XU, P.; CHU, X. Benchmarking state-of-the-art deep learning software tools. *In*: **2016 7th International Conference on Cloud Computing and Big Data (CCBD)**. [*S.l.*: *s.n.*], 2016. p. 99–104.

SHWARTZ-ZIV, R.; TISHBY, N. **Opening the Black Box of Deep Neural Networks via Information**. Ithaca, NY: Cornell University, arXiv, The Computing Research Repository (CoRR), 2017. Available at: http://arxiv.org/abs/1703.00810. Accessed on: 12 Mar. 2018.

USA. **SUMMARY OF THE 2018 WHITE HOUSE SUMMIT ON ARTIFICIAL INTELLIGENCE FOR AMERICAN INDUSTRY**. The White House, Washington DC, USA, May 2018. Available at: https://www.whitehouse.gov-/wp-content/uploads/2018/05/Summary-Report-of-White-House-AI-Summit.pdf. Accessed on: 16 Apr. 2019.

VASWANI, A.; SHAZEER, N.; PARMAR, N.; USZKOREIT, J.; JONES, L.; GOMEZ, A. N.; KAISER, L.; POLOSUKHIN, I. Attention is all you need. *In*: **Proceedings of the 31st International Conference on Neural Information Processing Systems**. USA: Curran Associates Inc., 2017. (NIPS'17), p. 6000–6010. ISBN 978-1-5108-6096-4. Available at: http://dl.acm.org/citation.cfm?id=3295222.3295349.

WOLPERT, D. H. The lack of a priori distinctions between learning algorithms. **Neural Computation**, v. 8, n. 7, p. 1341–1390, Oct 1996.

ZILLY, J. G.; SRIVASTAVA, R. K.; KOUTNÍK, J.; SCHMIDHUBER, J. **Recurrent Highway Networks**. Ithaca, NY: Cornell University, arXiv, The Computing Research Repository (CoRR), 2016. Available at: http://arxiv.org/abs/1607.03474. Accessed on: 04 Mar. 2019.

ZURADA, J. **Introduction to Artificial Neural Systems**. St. Paul, MN, USA: West Publishing Co., 1992. ISBN 0-314-93391-3.

Appendix A - Linear Algebra foundations for ANN

A.1 The Sparse Synaptic Weights Matrix

The Sparse Synaptic Weights Matrix is represented in a compact manner without the N^2 amount of zeros that would occur in a canonical representation. As shown in Equation A.1

$$W = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{15} \\ w_{21} & w_{22} & w_{24} & w_{26} \\ w_{31} & w_{33} & w_{34} & w_{37} \\ w_{42} & w_{43} & w_{44} & w_{48} \\ w_{51} & w_{55} & w_{56} & w_{57} \\ w_{62} & w_{65} & w_{66} & w_{68} \\ w_{73} & w_{75} & w_{77} & w_{78} \\ w_{84} & w_{86} & w_{87} & w_{88} \end{bmatrix}_{Hypercube} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & 0 & w_{15} & 0 & 0 & 0 \\ w_{21} & w_{22} & 0 & w_{24} & 0 & w_{26} & 0 & 0 \\ w_{31} & 0 & w_{33} & w_{34} & 0 & 0 & w_{37} & 0 \\ 0 & w_{42} & w_{43} & w_{44} & 0 & 0 & 0 & w_{48} \\ w_{51} & 0 & 0 & 0 & w_{55} & w_{56} & w_{57} & 0 \\ 0 & w_{62} & 0 & 0 & w_{65} & w_{66} & 0 & w_{68} \\ 0 & 0 & w_{73} & 0 & w_{75} & 0 & w_{77} & w_{78} \\ 0 & 0 & 0 & w_{84} & 0 & w_{86} & w_{87} & w_{88} \end{bmatrix} \tag{A.1}$$

Therefore, its adjacency matrices, depicted on Figure A.1 may also be considered as the expanded counterparts for their dense representation.

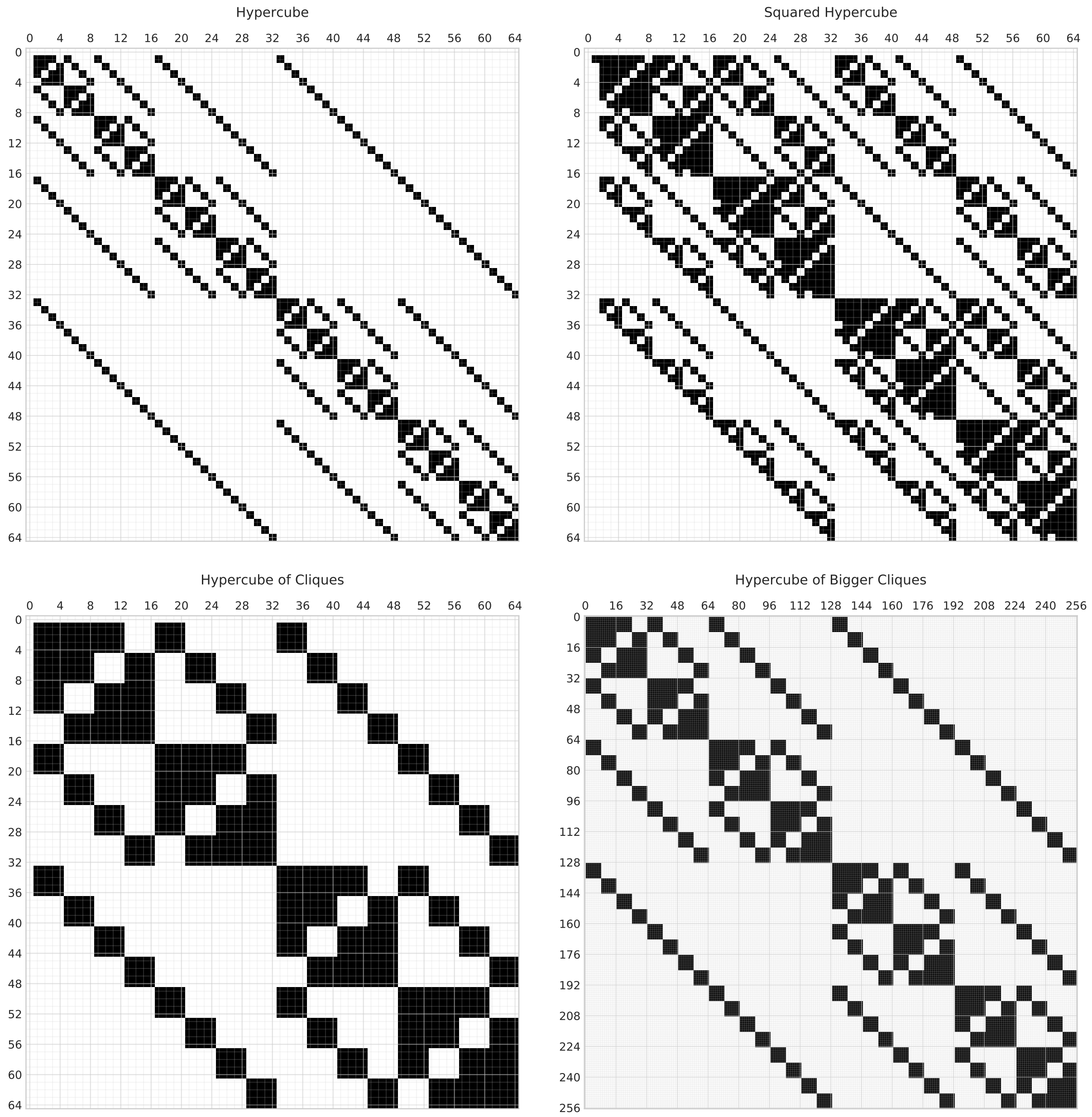


FIGURE A.1 – Adjacency matrices comparing four different architectures

Annex A - t-SNE Visualization

A.1 Dimensional Decomposition

The T-distributed Stochastic Neighbor Embedding (t-SNE) reduces a high-dimensional data to a lower dimension by trying to preserve the clusters. There are attraction and repulsion forces that moves the points in the projections in each step. Firstly, it determines how similar the pairs of data points are, using a t-student distribution on a euclidean metric. Therefore it is a N^2 order algorithm, meaning that close points have high similarity values. The sum of those similarities metrics are then normalized.

This results in a better capability to project clustered data, preserving the notion of similarity between elements in a cluster as observed in Figure A.1. The 10 digits clusters are clearly separated if not by some element or other that is randomly distributed. In comparison to other methods, as shown in Figure A.2 Sammon mapping in top left, Isomap in top right and LLE in the lower center, it is superior.

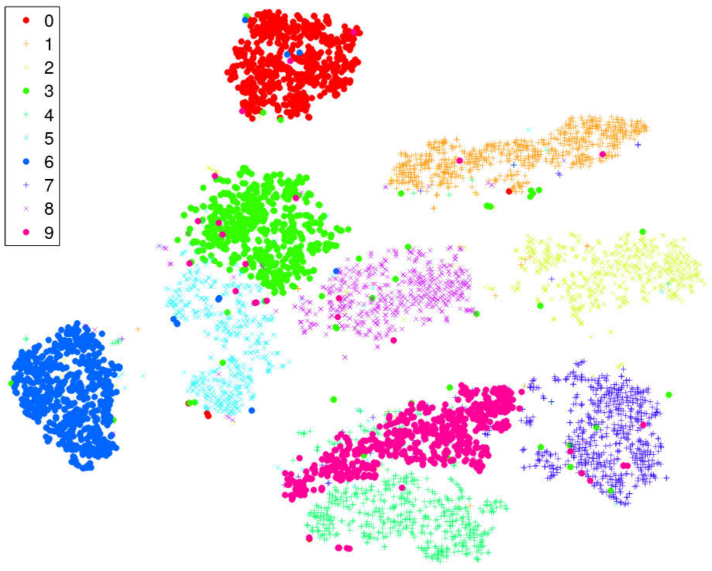


FIGURE A.1 – t-SNE projection for the MNIST handwritten digits database for 6000 points. Source: (MAATEN; HINTON, 2008)

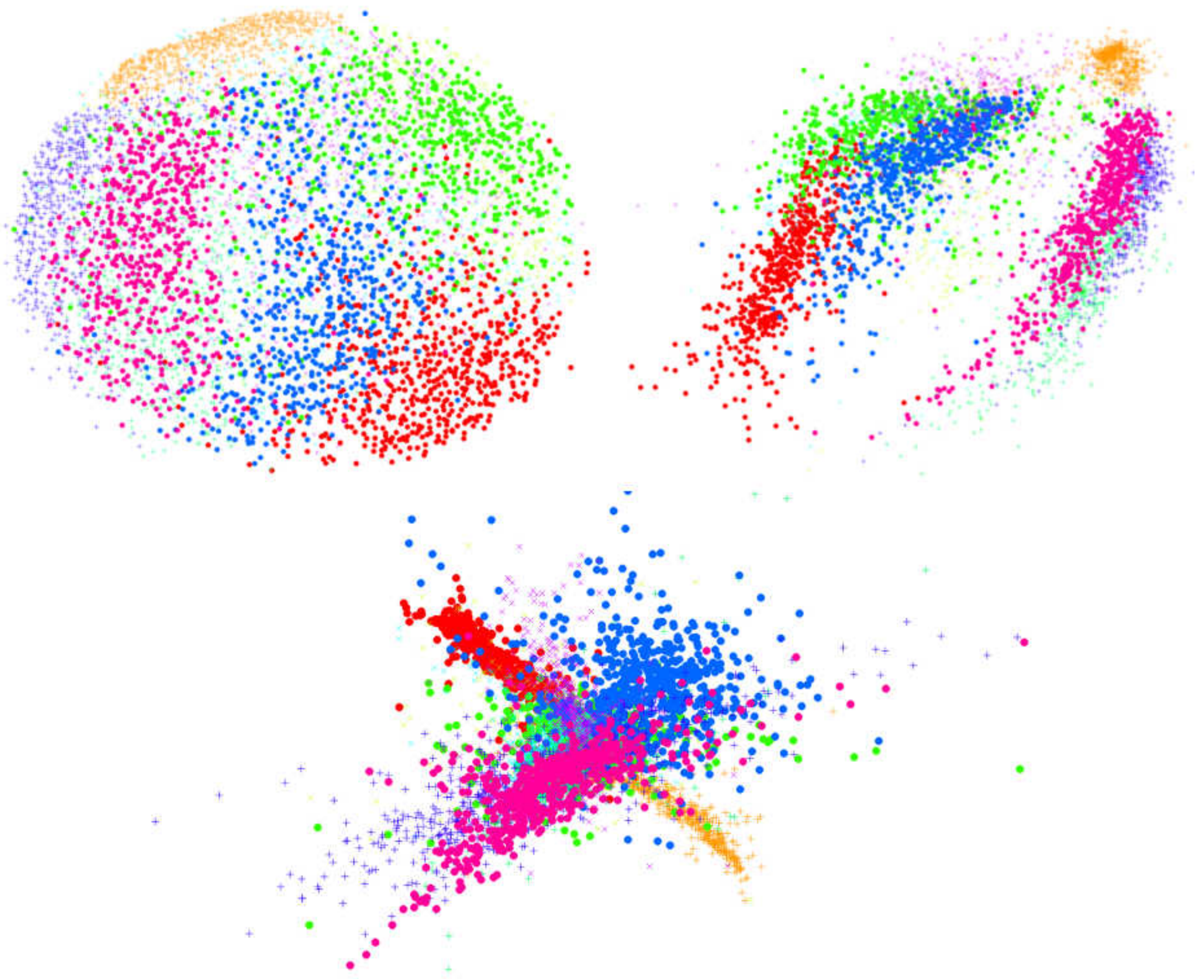


FIGURE A.2 – Sammon, Isomap and LLE dimensionality reductions for the same MNIST data points. Source: (MAATEN; HINTON, 2008)

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO DM	2. DATA 16 de dezembro de 2019	3. DOCUMENTO Nº DCTA/ITA/DM-093/2019	4. Nº DE PÁGINAS 126

5. TÍTULO E SUBTÍTULO:
Hypercube neural networks for natural language processing applied to story point estimation

6. AUTOR(ES):
Gabriel Adriano de Melo

7. INSTITUIÇÃO(ÕES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ÕES):
Instituto Tecnológico de Aeronáutica – ITA

8. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR:
Neural Networks; Deep Learning; Story Points; Agile software development; Natural Language Processing; Scrum.

9. PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:
Redes neurais; Aprendizagem (inteligência artificial); Desenvolvimento de software; Linguagem natural (computadores); Computação.

10. APRESENTAÇÃO: ( ) **Nacional** (**X**) **Internacional**
ITA, São José dos Campos. Curso de Mestrado. Programa de Pós-Graduação em Engenharia Eletrônica e Computação. Área de Informática. Orientador: Prof. Dr. Paulo Marcelo Tasinaffo; coorientador: Prof. Dr. Inaldo Capistrano Costa. Defesa em 16/12/2019. Publicado em 2019.

11. RESUMO:
Story points are the most commonly used unit in estimating a user's story effort in agile software development methodologies. The use of deep neural networks to make this estimate is little present in the literature but could become a new estimation tool in agile teams. The input to the proposed neural network model is the textual description of the user story itself and its output is a numerical estimate. Recurrent neural models are employed, in which the word sequence is taken into account with each iteration of the neural network. As training data, we used open source projects that make use of agile methodologies and describe their functionalities in user stories with their respective associated user points, raising thousands of test cases obtained from the literature. A new architecture based on hypercube cliques of neurons was proposed, characterizing a neural network with sparse connections whose computational efficiency in space and potentially in training time was superior to the architecture of similar classification accuracy. In order to improve the explicability of the language and neural regression model, visualization techniques of its activations and its sensitivity to variations in inputs were used. A case study was also carried out at the Aeronautics Computing Center in São José dos Campos (CCA-SJ) by collecting data from previous projects, pre-processing and specific training of the network, making its subsequent evaluation and comparison with null models that consider only the distribution of story points.