UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE MASTER

Development of a Deep Learning Based Sentiment Analysis and Evaluation Service

Ignacio Corcuera Platas

2018

TRABAJO DE FIN DE MASTER

Título:	Desarrollo de un Sistema de Análisis de Sentimiento basado en Deep Learning y un Servicio de Evaluación
Título (inglés):	Development of a Deep Learning Based Sentiment Analysis and Evaluation Service
Autor:	Ignacio Corcuera Platas
Tutor:	J. Fernando Sánchez
Ponente:	Carlos A. Iglesias
Departamento:	Departamento de Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	—
Vocal:	—
Secretario:	—
Suplente:	—

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO DE FIN DE MASTER

Development of a Deep Learning Based Sentiment Analysis and Evaluation Service

febrero 2018

Resumen

Este trabajo de fin de master recoge el resultado de un proyecto cuyos objetivos son: desarrollar el clasificador de sentimientos y emociones, publicarlos como un servicio web e implementar un servicio de evaluación.

Por un lado, los clasificadores se han desarrollado utilizando técnicas de Deep Learning, específicamente Redes neuronales. El proceso de construcción de estos clasificadores ha implicado probar varias técnicas y observar sus resultados. Los datasets utilizados para el entrenamiento de los clasificadores son Stanford Sentiment Treebank y el dataset ISEAR.

Por otro lado, tenemos la parte de los servicios. El proyecto ha utilizado Senpy como framework principal para desarrollar y desplegar los servicios. Senpy proporciona una capa semántica para la anotación de datos enlazados. Además, el objetio es extender la funcionalidad de Senpy para ofrecer la evaluación como un servicio de análisis. Este servicio permitirá a los usuarios no solo probar el algoritmo sino también evaluarlo con diferentes conjuntos de datos y poder hacer comparaciones entre algoritmos.

Finalmente, exponemos las conclusiones extraídas de este proyecto, la tecnología aprendidas en el transcurso de él.

Palabras clave: Análsis de Sentimentos, Deep Learning, análisis de emociones, servicios, NLP, ANN, RNN, GRU, LSTM

Abstract

This master thesis collects the result of a project whose objectives are: developing sentiment and emotion classifier, publish them as a web service and implement an evaluation service.

On the one hand, classifiers have been developed using Deep Learning Techniques, specifically Neural Networks. The process of building these classifiers has involved testing several techniques and observe theirs results. The datasets used for training the classifiers are the Stanford Sentiment Treebank and the ISEAR emotion dataset.

On the other hand, we have the services part. The project has used Senpy as the main framework for developing and deploying the services. Senpy provides a semantic layer for linked data annotation. Moreover, it will add the evaluation functionality to Senpy in order to offer it as an analysis service. The idea is to implement an evaluation service inside the Senpy structure, allowing users not only to test the algorithm but also to evaluate it with different datasets and be able to make comparisons between algorithms.

Finally, we gather the extracted conclusions from this project, the technologies we have learned during the development.

Keywords: Deep Learning, Sentiment Analysis, Emotion Analysis, service, NLP, ANN, RNN

Contents

Resumen **VII**

Abstract **IX**

Contents **XI**

List of Figures **XV**

1 Introduction **1**

1.1 Context 2
1.2 Project goals 4
1.3 Structure of this document 5

2 Introduction to Neural Networks **7**

2.1 Sentiment Analysis 8
2.2 Emotion Analysis 9
2.3 Word Embeddings 12
2.4 Artificial Neural Networks 15
2.5 Activation functions 17
2.5.1 Sigmoid function 18
2.5.2 Tanh function 18
2.5.3 ReLU function 19
2.5.4 Softmax function 20
2.6 Learning Rule 20
2.6.1 Cost functions 22
2.6.2 Optimizers 23
2.7 Architecture 24
2.7.1 Feed-forward Neural Network 25
2.7.1.1 Convolutional Neural Networks 26
2.7.2 Recurrent Neural Networks 27
2.7.2.1 Long Short Term Memory 28
2.7.2.2 Gated Recurrent Unit 32

2.8 Attention Mechanishm in Neural Networks 34

3 Enabling Technolgies **35**

3.1 Machine Learning Technologies 36

3.1.1 TensorFlow 36

3.1.2 Keras 36

3.1.3 Scikit-learn 37

3.1.4 GSITK 38

3.1.5 Pandas 38

3.1.6 Gensim 39

3.2 Senpy 41

3.3 Linked Data 43

3.3.1 Marl Ontology 44

3.3.2 Onyx Ontology 45

4 Architecture and Methodology **47**

4.1 Methodology 48

4.2 Architecture 49

5 Deep learning for Sentiment and Emotion Analysis **51**

5.1 Sentiment Analysis Using Deep Learning Techniques 52

5.1.1 First steps 52

5.1.2 Attention Layer 58

5.1.3 Combining and exploring models 59

5.2 Emotion Analysis Using Deep Learning Techniques 62

5.3 Conclusion 63

6 Machine Learning as Web Services **65**

6.1 Introduction 66

6.2 Developtment of the plugins 66

7 Developing Evaluation Services **69**

7.1 Architecture 70

7.1.1 API and Playground 72

8 Conclusions and future work **75**

8.1 Achieved Goals 76

8.2 Conclusion 77

A New Plugin Paradigm **79**

B Evaluation Ontology **81**

Bibliography **83**

List of Figures

2.1 Plutchik's Emotion Wheel 10
2.2 Three-dimensional PAD representation 11
2.3 Word Embedding Representation Example 13
2.4 Process from a string to sequence 13
2.5 Artificial Neural Network Diagram 16
2.6 Neuron Model 17
2.7 Softmax function 18
2.8 Hyperbolic tangent function 19
2.9 Rectified linear function 19
2.10 Structure of a recurrent and a feed-forward neuron 24
2.11 Single-Layer Perceptron 25
2.12 Multi-layer Perceptron 26
2.13 Convolutional Neural Network in Natural Language Processing 27
2.14 RNN context over the time 28
2.15 Sequential data 28
2.16 Long Short Term Memory Unit 29
2.17 Long Short Term Memory Unit: Forget Gate 30
2.18 Long Short Term Memory Unit: Memory Gate 30
2.19 Long Short Term Memory Unit: Output Gate 31
2.20 Long Short Term Memory Unit: Gradient Problem 32
2.21 Long Short Term Memory Unit: Gradient Problem 33

3.1 Senpy framework. Each layer represents a functional block in a service . . . 42
3.2 Senpy architecture 43
3.3 Class and Properties Diagram for the Marl Ontology 45
3.4 Class and Properties Diagram for the Onyx Ontology 46

4.1 Project phases 48
4.2 General Architecture 49

5.1 Base Model 54

5.2 Evolution of the accuracy during the training process 57
5.3 Evolution of the loss during the training process 57
5.4 Evolution of the validation loss during the traning process 57
5.5 Attention Mechanishm inside the model 58
5.6 Bidirectional Recurrent Neural Network Architecture 59
5.7 C-LSTM Neural Network 60

6.1 General architecture - Plugin development 66
6.2 Wrappers involved in the development of the plugins 67

7.1 General Architecture - Evaluation Process 71
7.2 Evaluation Wrapper inside the plugin context 71
7.3 Evaluation Playground - Results from the evaluation, table format 74
7.4 Evaluation Playground - Results from the evaluation 74

A.1 Black Box basic diagram 79
A.2 Life cycle from developing an algorithm to implement the BlackBox 80

B.1 Evaluation Ontology draft 82

CHAPTER 1

Introduction

This chapter is going to introduce the context of the project, including a brief overview of all the different parts that will be discussed in the project. It will also break down a series of objectives to be carried out during the realization of the project. Moreover, it will introduce the structure of the document with an overview of each chapter.

1.1 Context

In this technological era, the creation and storage of data is a common practice in day to day. This contribution to the massive accumulation of data can be found in various industries, companies maintain large amounts of transactional data, gathering information about their customers, suppliers, operations, etc., in the same way it happens with the public sector. This huge amount of data makes it impossible for a person to draw conclusions, and even more to make predictions, just by analyzing the data manually. In this context is where the concept of machine learning arises as the use of powerful algorithms to transform raw data into information that can be used to make predictions or give advice based on it.

Recently, Deep Learning techniques are giving better results than Convectional Machine Learning. Deep Learning is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms. One of the approaches within the Deep Learning field are the Artificial Neural Networks.[10], which is supervised approach where a model is build based on examples. This type of algorithms can be defined as "non-linear approaches to the way the brain works"[35]. Therefore, these networks base their functioning on the brain but they don't emulate it. They present the following characteristics (similar to the brain):

- Knowledge is acquired experimentally.
- Interconnection gains vary constantly

Artificial Neural Networks have turned into a very popular and useful tool for solving many problems such as classification, clustering, regression, pattern recognition, etc. This project will use the Artificial Neural Networks in Natural Language Processing tasks (NLP), specifically in Sentiment Analysis.

Sentiment analysis, also known as opining mining, is a type of data mining that measures the inclination of people's opinions through NLP, computational linguistics and text analysis, which are used to extract and analyze subjective information from the Web - mostly social media and similar sources[26]. The analyzed data quantifies the general public's sentiments or reactions toward certain products, people or ideas and reveal the contextual polarity of the information. Several approaches of the Neural Networks has proved to achieve good results in Sentiment Analysis taks, enhancing the conventional approaches. For example, Recursive Neural Networks [41], Bidirectional Recurrent Networks[24] and Convolutional Neural Networks for Sentence Classification [16].

On the other hand, evaluated the models is always necessary to ensure that it will do a good job of predicting the target on new and future data. Because future instances have

unknown target values, you need to check the accuracy metric of the Machine Learning model on data for which you already know the target answer, and use this assessment as a proxy for predictive accuracy on future data. Moreover, the cooperation between models it is an important task because it gives an idea about which model behaves better in a specific domain.

Furthermore, it is becoming more common to find classification algorithms (e.g: sentiment and emotion analysis, topic extraction, etc) deployed as a external service rather than having them locally. For example, Meaning Cloud [1] provides a powerful api that performs several natural language processing tasks: topic extraction, text classification, sentiment analysis, etc.

Following this tendency of machine learning as a web service, this project will use Senpy[37], which provides a semantic layer for linked data annotation, as the framework to publish the classifiers developed into a service. Moreover, it will add the evaluation functionality to Senpy in order to offer it as an analysis service. The idea is to implement an evaluation service inside the Senpy structure, allowing users not only to test the algorithm but also to evaluate it with different datasets and be able to make comparisons between algorithms. The idea is to implement this evaluation under some standard metrics such as recall, precision, f-score and accuracy [32].

[1] Meaning Cloud website: https://www.meaningcloud.com/developer/apis

1.2 Project goals

The aim of the project is to provide a comparative between deep learning and shallow learning in the Sentiment and Emotion Analysis domain. This general idea has been divided into three goals:

- Explore Artificial Neural Network techniques in order to develop a sentiment and an emotion classifier.
- Create analysis web services with the developed classifiers using Senpy as the framework to publish these services.
- Adapt Senpy so that it allows to carry out the evaluation process and include this as a new service.

1.3 Structure of this document

The remaining of this document is structured as follows:

Chapter 2 presents the state of art in Sentiment and Emotion Analysis, Word Embedding and Neural Networks. It provides an in-depth summary about some basic types of networks and all the parameters that are involved in their development.

Chapter 3 provides an overview of all the technologies that will be used on this project. Moreover, it will describe some resources that will be important.

Chapter 4 shows the overall architecture of the project.

Chapter 5 describes all the approaches carried out with the Neural Networks and a explanation of the advanced techniques that have been used for developing the classifiers.

Chapter 6 covers the development of web services to publish deep learning sentiment classifiers.

Chapter 7 describes the implementation of evaluation services insides Senpy framework and an overview of evaluation API.

Chapter 8 summarizes the conclusions of each one of the phases.

CHAPTER 2

Introduction to Neural Networks

Machine learning explores the construction and study of algorithms that can learn from and make predicts on data. Such algorithms operate by building a model from example inputs in order to make data-driven predictions or decisions, rather than following strictly static program instructions.

***Deep learning**[18] allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction Although there are many implementations, one of the most common is the neural network.*

Neural networks have proved to be a powerful tool in speech recognition as it can be seen in this study[38]. In this chapter will give an introduction to the neural networks, and it will explain all the networks developed for all the use cases.

This section shows an overview about Sentiment Analysis, Emotion Analysis and Word Embedding as well as an explanation of the capabilities of the neural networks and all their components. Moreover, it explains the architecture and the networks used on this project.

2.1 Sentiment Analysis

Sentiment analysis, also known as opining mining, is a type of data mining that measures the inclination of people's opinions through natural language processing (NLP), computational linguistics and text analysis, which are used to extract and analyze subjective information from the Web - mostly social media and similar sources. The analyzed data quantifies the general public's sentiments or reactions toward certain products, people or ideas and reveal the contextual polarity of the information.

The task of automatically classifying a written text into a positive or negative feeling, opinion or subjectivity [26], is sometimes so complicated that it is even difficult to agree to different human scorers on the classification to assign to a given text. The personal interpretation of an individual is different from other ones, and it is also affected by cultural factors and the experiences of each person. Furthermore, the popularity of social networks and the use of new expressions, such as emojis, abbreviations and colloquial expressions, make this task can be addressed in many ways.

The conventional machine learning approaches used by the researchers to extract sentiment from text automatically are mostly two: lexicon-based and text classification approach[43]. In the first case documents sentiment is calculated from the words or phrases [44]. In the second case classifiers are built from annotated instances of text or sentences also described as a statistical or machine learning approach[27][15]. Supervised machine learning methods based classifiers have gained high accuracy in detection of text polarity [3], but performance of machine learning is domain dependent. On the other hand lexicon-based approach works well in cross-domain and can be enhanced easily with source of additional knowledge for sentiment classification [42]. Lexicon-based approach also performs well in handling contextual valence shifter[43].

On the other hand, deep learning is a promising alternative to traditional methods. It has shown excellent performance in NLP tasks, including Sentiment Analysis[7]. Neural networks, as a deep learning techniques, have become a very popular topic of research in Sentiment Analysis. There are several approaches such as Recursive Neural Networks [41], Bidirectional Recurrent Networks[24],Convolutional Neural Networks (CNN) built on top of word2vec[16]and the combination of CNN and LSTM [46] showing an improvement of the results compared to conventional approaches.

2.2 Emotion Analysis

The Emotion Recognition is an important component of "affective computing" and has been implemented in many kinds of media. For example, speech and image are the most common ways to recognize emotion. Picard [29] proposed the concept of affective computing and deduced theories of emotion, recognition, and generation. Another approach use speech and text for complete its emotion recognition [5].

Most of the Emotion Analysis are focused on speech recognition. However, Emotion Detection and Recognition from text is a recent field of research that is closely related to Sentiment Analysis. Emotion Analysis aims to detect and recognize types of feelings through the expression of texts, such as the ones describes in the models above. This article [48] propose a emotion recognition based on rules and different estimator for predicting the labels of a corpus. Moreover, Deep Learning approaches using neural networks have developed in this field [51].

Psychologists have argued that some emotions are more basic than others [8]. However, they disagree in which emotions should be classified as basic. The problem lies not only in deciding which emotions are basic, but in the diversity of models and the little unification that exists between them. As a result, there is a plethora of rivaling emotion representation models with varying degrees of popularity, from categorical models such as Ekman's to Scherer's process model [2]. Basically, the most important representation models are: discrete and dimensional.

Emotional discrete approaches are focused on model emotions based on distinct emotion classes or labels. The categorical model assumes that there are discrete emotion categories. The Ekman's basic emotion model [8] conclude that the six basic emotions are: Anger, Disgust, Fear, Happiness, Sadness and Surprise. However, there are a more complete approach [30], which increases the set to eight categories, adding Trust and Anticipation. Figure2.1 represents the Plutchik's model, where this eight categories are organized into four bipolar sets 2.1: joy vs sadness, anger vs fear, trust vs disgust, and surprise vs anticipation.

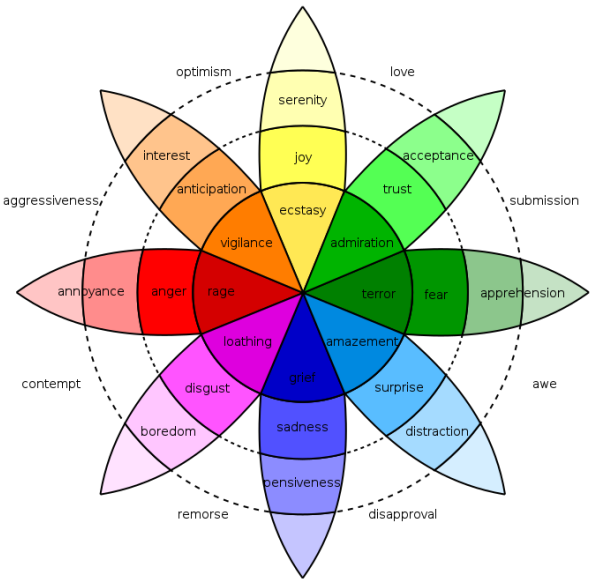

Figure 2.1: Plutchik's Emotion Wheel

Emotional dimensions approaches represent affects in a dimensional space, in which each emotion occupies a location in this space. Wilhelm Max Wundt [14] proposed that emotions can be described by three dimensions: pleasurable versus unpleasurable, "arousing or subduing", and "strain or relaxation". One of the most representative model of these approaches is [31], in which the emotions are distributed across a two dimensional circular space: valence and arousal. The valence dimension indicates the pleasure vs unpleasurable; the arousal dimension indicates the arousing or subduing. However, the Mehrabian's model is a three dimensional model[21], which is more close to the Wilhelm Max Wundt approach. The three-dimensional PAD (Pleasure-Arousal-Dominance), also referred as VAD (Valence-Arousal-Dominance), representation add the dominance to the previous approach which indicates whether the subject feels in control of the situation or not.

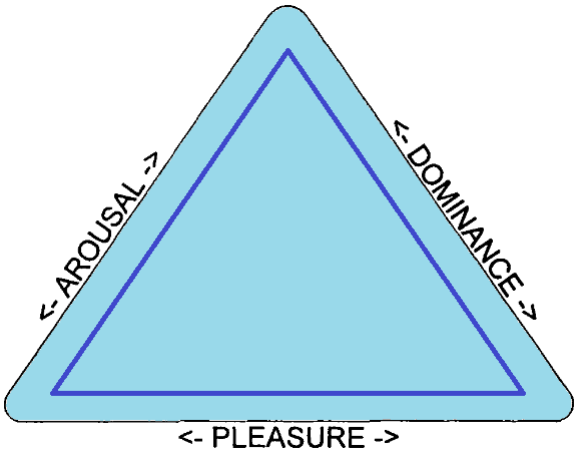

Figure 2.2: Three-dimensional PAD representation

2.3 Word Embeddings

A problem with modeling text is that it is messy, many Machine Learning algorithms and almost all Deep Learning Architectures are incapable of processing strings or plain text in their raw form, in the case of Neural Networks that happens too. Therefore, these techniques prefer well defined fixed-length features from which they can learn. Traditionally, the text is converted into a feature vector structure. A popular and simple method is Bag of Words, which represents the information as a zero vector of as many components as words exist in the vocabulary. The words that appear in a text or document will have a non-zero value in its correspondent index of the vector. This method has limitations such as overfitting, size in memory, etc.

As an alternative to this technique, emerge the vector space models such as Word Embeddings. Word embeddings are a class of techniques where individual words are represented as real-valued vectors in a predefined vector space. It can be said that Word Embedding is a parametized function that maps words in some language high-dimensional vectors.

$$W : words \rightarrow R^n \tag{2.1}$$

W is initialized to have random vectors for each word. The vector values are learned in a way that resembles a neural network, and hence the technique is often lumped into the field of deep learning. The distributed representation is learned based on the usage of words, the idea is to find the proper positions to embed a word into the semantic space, in this section it will be called as embedding matrix. This embedding matrix has fixed size for representing the vectors.

Given the way that the embedding vector are created, words with similar context will have close vectors. One simple way to understand this is to look at the image2.4. This case is exposing the male-female different vector, where $king - man + woman = queen$.

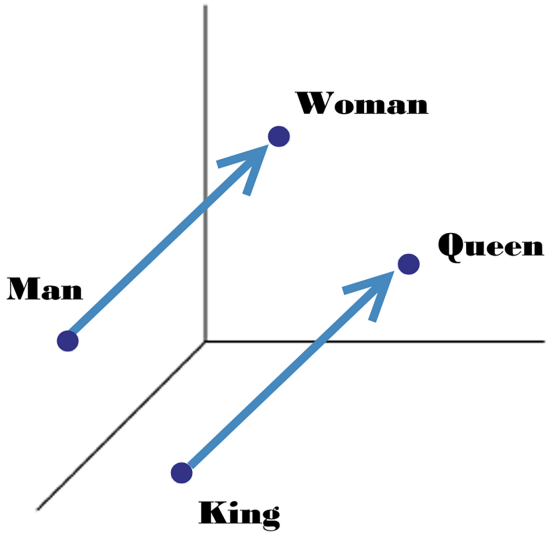

Figure 2.3: Word Embedding Representation Example

The process from having the string to get the sequence of vector could be summarized as follow. First, the string is cleaned from noise (deleting special chars, URLs ,etc.). Them, the string is going to be transformed as a index vector, where each value correspond to the index (id of the row) of the word in the embedding matrix. Once the index-vectors are obtained, the vector corresponding to each word is extracted and all of them are grouped together forming a sequence. This final representation will feed the neural network.

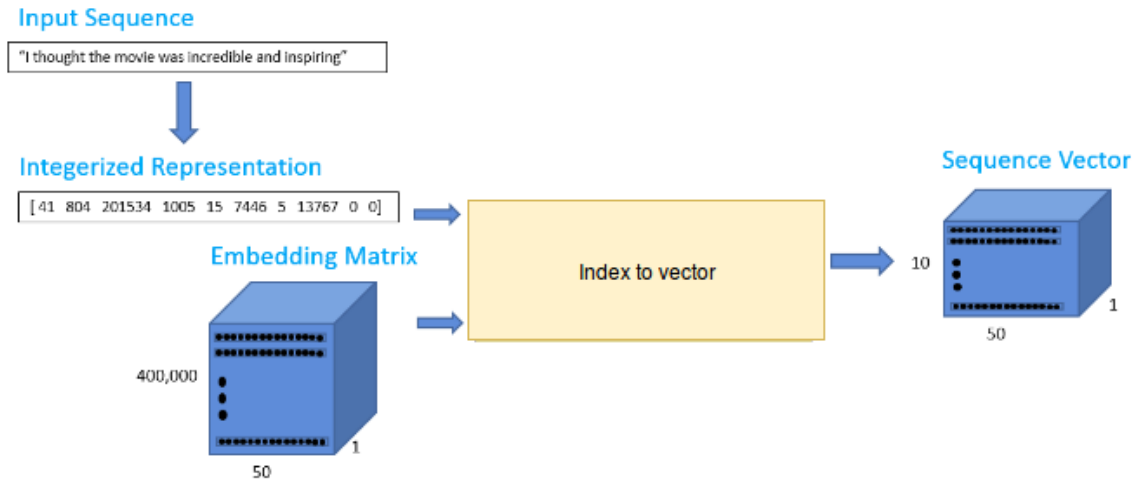

Figure 2.4: Process from a string to sequence

There are several ways of performing this process, but in this project have been used two:

- Embedding Layer, is a word embedding that is learned jointly with a neural network model on a specific natural language processing task, such as document classification. It requires that document text be cleaned and prepared.
- Word2Vec, is a statistical method for efficiently learning a standalone word embedding from a text corpus. The idea is to use the pre-trained models described in 3.1.6 for obtaining the sequence vector. In this case, an embedding layer will be used too but its weights corresponds to the ones from the Word2Vec models.

2.4 Artificial Neural Networks

A neural network is a mathematics tool that models, at a very low level, the functionality of the neurons in the brain. However, unlike a biological brain where any neuron is connected to any other neuron by a physical distance, these **artificial neural networks** (**ANNs**) have discrete layers, connections, and directions of data propagation.

ANNs offer the following properties and capabilities[10]:

1. **Non-linearity**, a neuron is basically a non-linear device; therefore, a neural network will be equally non-linear.

2. **Transformation input-output**, the learning process of a neural network involves the modification of the connections between the different artificial neurons that make up the network. Therefore, the use of a neural network will imply a transformation of the input parameters for getting a specific exit.

3. **Adaptability**, this kind of algorithms have a great capacity for adaptation by adjusting its parameters. Therefore, a trained neural network trained in a specific environment could be conditioned to operate in another similar environment.

4. **Indicative response**, in the context of patter recognition, a neural network can be designed to provide information not only of the pattern selected, but also the degree of confidence in the decision made. This information can be used later to reject old patterns.

5. **Contextual information**, the knowledge state comes from the activation of the neural network. Basically, each neuron is affected by the global activity of the other neurons that compose the network.

6. **Fault tolerance**, due to the massive interconnection, a neural network physically implemented is tolerant to failures and despite some neurons or connections are potentially damaged, the total performance is not seriously affected.

The diagram 2.5 shows a simple overview of the **ANNs** architecture. Each circle represents a neuron, that are grouped in layers. The layers can be divided in three main categories:

- Input layers produce information from some source. The input could be of any type: text, image, audio,etc. These layers generates the information that feeds other nodes. Figure 2.5 shows, they have no input-connectors with the **ANN**; only outputs.

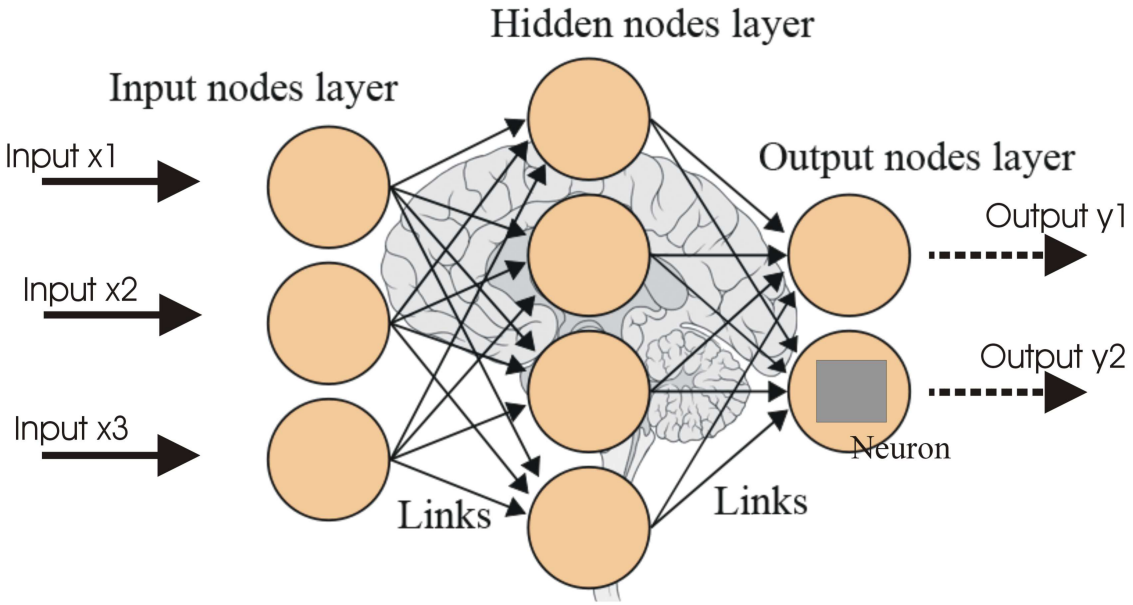

Figure 2.5: Artificial Neural Network Diagram

- Hidden layers contain intermediate calculations of the network. Its functionality is to transform the inputs received by the previous layer (input or hidden) into something that the following layer can use for the next step.
- Output layers combines and modify the data once the network has performed all the transformations. They are in charge of produce the output scale that is desired.

Neural networks can have any number of layers, and any number of nodes per layer. Most applications use the three layer structure. The hidden and output layers are categorized as 'active' due to they modify the data.

As it can be seen in figure 2.6, a neuron is formed by the next components:

- Set of synaptic connections: each one characterized with a weight
- Adder: that will add the components of the multiplied input signals by the weights of the synaptic connections.
- Activation function: responsible for limiting the output of the neuron between the desired values (non-linear transformation).
- Fixed entry: used to introduce a bias (b_k) in charge of increasing or decrease the input value of the activation function.

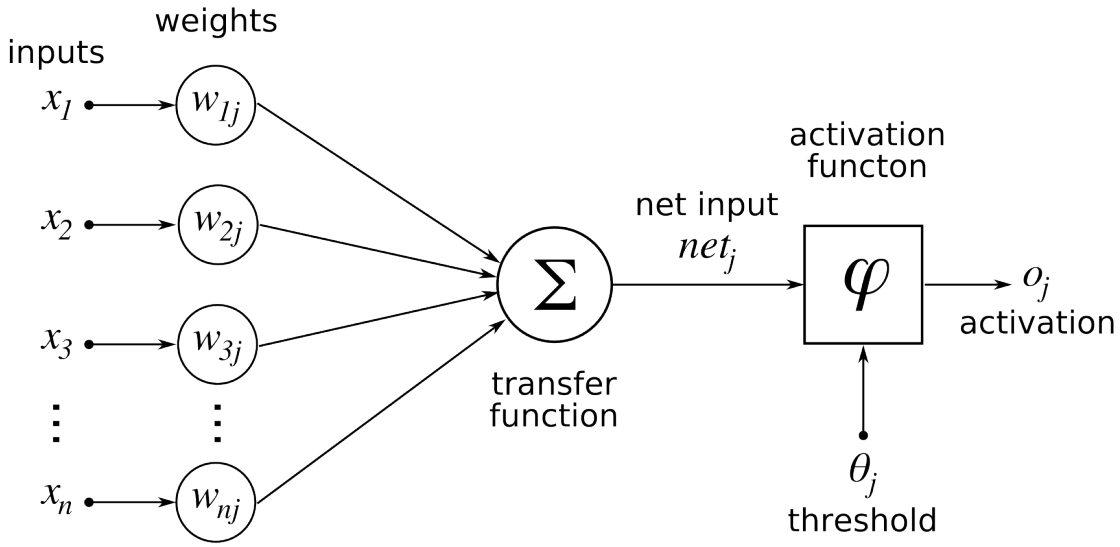

Figure 2.6: Neuron Model

Basically, the main operation performed by the network of neurons is to multiply the values of a neuron by the weights of its outgoing connections. Each neuron in the next layer receives numbers from several incoming connections, and the first thing it does is add them all together. Them, it applies the activation function at the sum. Mathematically, the model of a neuron is shown by the next formulas.

$$u_k = \sum_{j=1}^{m} w_{kj} * x_j \tag{2.2}$$

$$y_k = \varphi(u_k + w_0 \tag{2.3}$$

2.5 Activation functions

In computational networks, the activation function of a node defines the output of that node given an input or set of inputs. Once the neuron add the weights and all the values together, it gets a value v_k that could be anything ranging from -∞ to ∞. The neuron doesn't know the bounds of this value or where it has to route it. The activation function takes the decision of whether or not to pass the signal and transform that signal to something with bounds. This subsection will explain the different activation functions that will be related to the development of the deep learning analysis.

There are two main types of functions:

- **Linear Activation Function**, it is a simple linear function of the form $f(x) = x$, where the input passes through it without any modification.

- **Non-Linear Activation Functions**, these functions are used to separate the data that is not linearly separable and are the most used activation functions. TANH, RELU, SOFTMAX and SIGMOID will be explained below.

2.5.1 Sigmoid function

It consists in a special case of the Logistic Activation Function. Basically, it transform the input in an output with bounds between 0 and 1 so it can be interpreted as a probability, it converts large negative number to 0 and large positive number to 1. The mathematical and graphical representation are the following:

$$\sigma(x) = \frac{1}{1 + exp^{-x}} \tag{2.4}$$

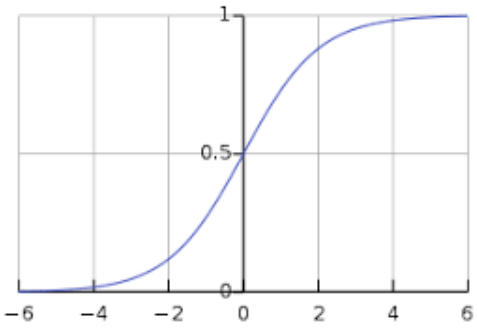

Figure 2.7: Softmax function

Sigmoid are one of the most widely used activations functions. However, they have VANISHING GRADIENTS, which could affect at the learning process of the neural network. At figure 2.7 can be appreciated the representation of a sigmoid function. The order axis (Y) tends to respond very less to changes in the abscissas axis (X). This makes the neuron to enter in a saturated regime, making the network to refuse learning further or slowing the process . This problem will appear in one of the use cases (USE:CASE EMOJI). Despite of this problem, there are ways to work around this problem and sigmoid is useful in classification taks.

2.5.2 Tanh function

It is also know as the hyperbolic tangent activation function. It has two main differences with the sigmoid function, as it can be seen in the image. The first one, it is the bounds. The **tanh** transform the outputs with bounds between -1 and 1. The results is that the representation is now centered at zero. The negative inputs considered as strongly negative, zero input values mapped near zero, and the positive inputs regarded as positive.

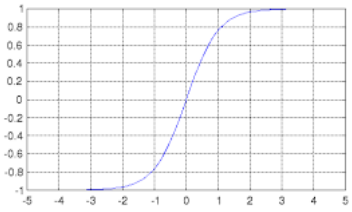

Figure 2.8: Hyperbolic tangent function

This function can be written as two sigmoid functions put together. The VANISHING GRADIENT is also present in this activation function.

$$\tanh(x) = \frac{2}{1 + exp^{-2x}} - 1 \tag{2.5}$$

2.5.3 ReLU function

It is also know as rectified linear unit activation function. This functions gives the input as output if it is positive and 0 otherwise. Figure 2.9 represents a Relu function.

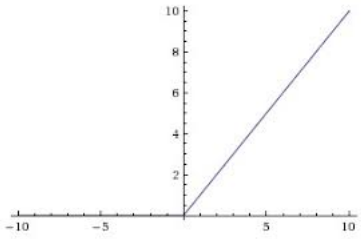

Figure 2.9: Rectified linear function

This activation makes the network converge much faster. It does not saturate which means it is resistant to the vanishing gradient problem at least in the positive region. Due to the horizontal line in **ReLu**(for negative X), the gradient can go towards 0. For activation in that region of **ReLu**, gradient will be 0 because of which the weights will not get adjusted during descent. That means, those neurons which go into that state will stop responding to variations in error/ input (simply because gradient is 0, nothing changes).

For solving this problem, there is a variant of the ReLU called LEAKY RELU. The are more implementations of the ReLU function, this paper describer a evaluation of a convolutional network using the different variants of the ReLU function as activation [49].

2.5.4 Softmax function

Sigmoid, tanh, relus are good activation functions for the neural networks. However, when you want to deal with classification problems, they seem to have some inconveniences. The softmax functions transforms the input into an output with bounds between 0 and 1, just like a sigmoid function. Moreover, it divides each output such that the total sum must be equal to 1. This output is equivalent to a categorical probability distribution.

Mathematically, the softmax function is shown below.

$$\sigma(x)_j = \frac{exp(x_j)}{\sum_{k=1}^{k}} \forall j = 1, ..., K \tag{2.6}$$

2.6 Learning Rule

Learning rule or learning process is a method or a mathematical logic. It improves artificial neural network's performance and applies this rule over the network. Thus, learning rules updates the weights and bias levels of a network when a network simulates in a specific data environment. Applying learning rule is an iterative process. It helps a neural network to learn from the existing conditions and improve its performance.

The different learning rules in the neural network are:

- **Hebbian learning rule**, it identifies, how to modify the weights of nodes of a network. It was the first learning rule, and it is a variant mechanism in time, local and highly interactive that increase the efficiency of the synapse as a function of the correlation between pre and post synaptic activities. The main idea behind this learning rule is that the connections between two neurons might be strengthened if the neurons fire at the same time and might weaken they fire at different times.

  $$\triangle\, w_{ji}(t) = \alpha x_i(t) * y_j(t) \tag{2.7}$$

  Where, the first term is the increment of the weight of connection, α is the positive and constant learning rate, and x,y are the corresponding input and output. All are evaluated at the same time step t.

- **Correlation Learning Rule**, based on a similar principle as the Hebbian learning rule. It assumes that weights between responding neurons should be more positive and weights between neurons with the opposite reaction should be more negative.

  $$\triangle\, w_{ij} = \eta x_i d_j \tag{2.8}$$

- **Error correction rule**, the idea is to minimize a cost function (criterion) based on the error signal, so that the response of each output neuron in the network should be as close as possible to the objective answer. The error signal commented will be the difference between the calculated output value with the expected value. Inside this rule are:

  - Perceptron learning rule, it is an error correcting the supervised learning algorithm of single layer feedforward networks with linear activation function. According to this rule, the network starts its learning by assigning a random value to each weight.It calculates the output value on the basis of a set of records for which we can know the expected output value. This is the learning sample that indicates the entire definition. As a result, it is called a learning sample.

    $$\sum_i \sum_j (E_{ij} - O_{ij})^2 \tag{2.9}$$

    Perform the first summation on the individuals of the learning set, and perform the second summation on the output units. E_{ij} and O_{ij} are the expected and obtained values of the j_{th} unit for the i_{th} individual. The network then adjust the weights of the different units.

  - Delta learning rule, it is called Least Mean Square (LMS), the base of this rule is gradient-descent approach, which continues forever (allowing to quantify the global error in each moment). Delta rule updates the synaptic weights so as to minimize the net input to the output unit and the target value.

    $$\triangle w_i = \alpha x_i e_j \tag{2.10}$$

    The idea is similar to the Hebbian learning but instead of the output, e_j is the difference between the desired output and the actual output.

- **Competitive learning rule**, this rule is a form of unsupervised learning in artificial neural networks, in which nodes compete for the right to respond to a subset of the input data. It is a rule based on the idea that only one neuron from a given iteration in a given layer will fire at a time. Weights are adjusted such that only one neuron in a layer, for instance the output layer, fires. Competitive learning is useful for classification of input patterns into a discrete set of output classes. The "winner" of each iteration, element i* , is the element whose total weighted input is the largest.

- **Outstar learning rule**, this rule is applied over the neurons arranged in a layer. It is specially designed to produce a desired output d of the layer of p neurons.

In conclusion to the learning rules in Neural Network, it can be said that most promising feature of the Artificial Neural Network is its ability to learn. Hence, there are several algorithms involved in the process of training neural network.

The backpropagation algorithm, also called back, in combination with a supervised error-correction learning rule, is one of the most popular and robust tools in the training of artificial neural networks. Back propagation passes error signals backwards through the network during training to update the weights of the network.

It was originally introduces in the 1970s, but its importance was not fully appreciated until this papr [34]. Backpropagation is a method used in artificial neural netwroks to calculate the error contribution of each neuron after a batch of data is processed. The idea is used to find a local minimum of the error function. The network is initialized with randomly chosen weights. The gradient of the error function is computed and used to correct the initial weights.

The application of the supervised error-correction learning rule is performed by a cost function and a optimizer, which is focused in minimize the cost function. The next two subsections are going to introduce the cost functions and optimizers.

2.6.1 Cost functions

A cost function is a measure of "how good" a neural network did with respect to it's given training sample and the expected output. It also may depend on variables such as weights and biases. It returns a single value, this value is the rate of how good is the network as a whole. The form of the cost function in neural networks is:

$$C(W, B, S^r, E^r) \tag{2.11}$$

Where W is the neural network weights, B is the biases, S^r is the input of a single training sample, and E^r is the desired output of that training sample. Another notation that will be used in this subsection is a_j^L, which is the activation value of the j^th neuron in the L^th layer.The main types of cost functions related with classification are:

- **Quadratic Cost**, the use of a quadratic loss function is common, for example when using least squares techniques. It is often more mathematically tractable than other loss functions because of the properties of variances, as well as being symmetric: an error above the target causes the same loss as the same magnitude of error below the target.

$$C(W, B, S^r, E^r) = C * \sum_j (a_j^L - E_j^r)^2 \tag{2.12}$$

- **Cross Entropy Cost**, the cross entropy between two probability distributions p and q over the same underlying set of events measures the average number of bits needed to identify an event drawn from the set, if a coding scheme is used that is optimized for an "unnatural" probability distribution q, rather than the "true" distribution p.

$$C(W, B, S^r, E^r) = -\sum_j [E_j^r \ln a_j^L + (1 - E_j^r) \ln(1 - a_j^L)] \tag{2.13}$$

- **Exponential Cost**, This requires choosing some parameter that you think will give you the behavior you want. Typically you'll just need to play with this until things work good.

$$C(W, B, S^r, E^r) = \tau e^{\frac{1}{\tau} \sum_j (a_j^L - E_j^r)^2} \tag{2.14}$$

2.6.2 Optimizers

Optimization algorithms helps us to minimize the cost function during the training process of a neural network.

- **Gradient Descent**, it is the most popular Optimization algorithms used in optimizing a Neural Network. Now, gradient descent is majorly used to do Weights updates in a Neural Network Model. It search the minimum of the cost function in a sequential way. The search direction of the minimum will be indicated by the negative form of the result vector of applying the gradient to the cost function.

$$\theta = \theta - \eta \Delta \triangle J(\theta) \tag{2.15}$$

  There are some variants of the Gradient Descent Optimizers:

  - Stochastics Gradient Descent (SGD), performs a parameter update for each training sample.

$$\theta = \theta - \eta \Delta \triangle J(\theta; x(i); y(i) where x(i) and y(i) are the training samples \tag{2.16}$$

  - Mini Batch Gradient Descent, it takes the best of the original technique and its variation described above.

- **Adagrad**, it allows the learning rate η to adapt based on the parameters. G is the historical gradient information. For each parameter we store sum of squares of its all historical gradients – this sum is later used to adapt a learning rate. In contrast to SGD, AdaGrad learning rate is different for each of the parameters. It is greater for parameters where the historical gradients were small (so the sum is small) and the rate is small whenever historical gradients were relatively big.

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{G_{t-1}}} \Delta \triangle J(\theta_{t-1}) \tag{2.17}$$

- **AdaDelta**, it is an extension of AdaGrad created for improve its weakness of decaying learning rate. Instead of accumulating all previous squared gradients in the historical, it limits the window of accumulated past gradients into a recent time window.

- **Adam**, it stands for Adaptive Moment Estimation. Update rule for Adam is determined based on estimation of first (mean) and second raw moment of historical gradients.

- **RMSProp**, it divides the learning rate of a weight through the mean of magnitudes of gradients recent that has had such weight.

To summarize, RMSProp, AdaDelta and Adam are very similar algorithm and since Adam was found to slightly outperform RMSProp, Adam is generally chosen as the best overall choice. However,

2.7 Architecture

There are two base neural networks used on this project: Feed-forward Neural Network and Recurrent Neural Network. The basic difference between a feed forward and a recurrent network consist in theirs neurons2.10. The feed forward neuron has only connections from his input to his output.. The recurrent neuron instead has also a connection from his output again to his input and therefore it has in this example three weights. This third extra connection is called feed-back connection and with that the activation can flow round in a loop. When many feed forward and recurrent neurons are connected, they form a recurrent neural network[20]

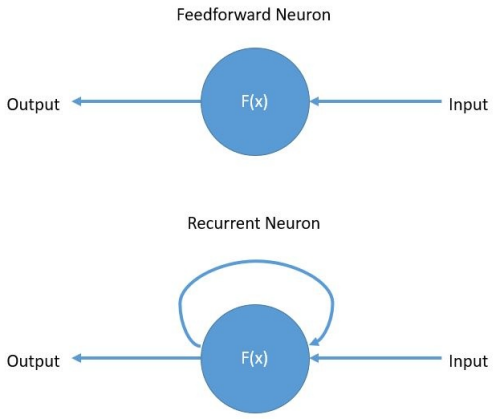

Figure 2.10: Structure of a recurrent and a feed-forward neuron

2.7.1 Feed-forward Neural Network

Feed-forward neural Network consist of a artificial neural network wherein connections between the units do not form a cycle. This kind of networks was the first and simple type of artificial neural network devised. They receive this name because the information flows forward the network until it reaches the output.

These networks are grouped into two types:

- **Single-layer perceptron**, this kind of neural network consists of a single layer of outputs nodes; as it can be seen in the diagram 2.11 the different input neurons feed directly the output neurons. it is the simplest type of feed-forward neural network, with no hidden layers.

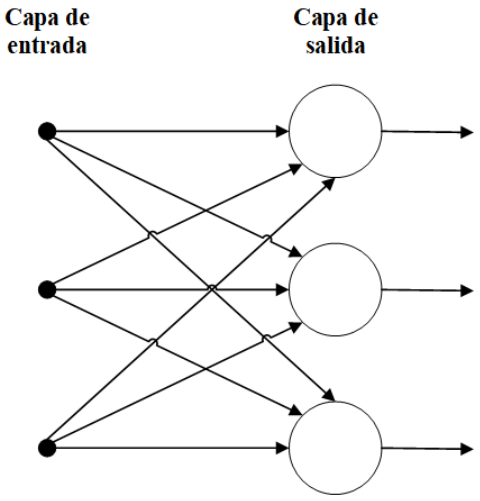

Figure 2.11: Single-Layer Perceptron

- **Multi-layer perceptron**, this kind of neural networks are similar to the single-layer perceptron but they at least one hidden layer between the input and the output layer2.12. The incorporation of the hidden layer allows the network to obtain higher order statistics than its inputs, this fact allows the network to acquire a global perspective despite its local connectivity. If each one of the nodes that compose a layer are connected to the nodes of the following layer, moreover, this happen in all the layer of the neural network, the neural network is fully connected. In case this doesn't happen, the network will be partially connected.

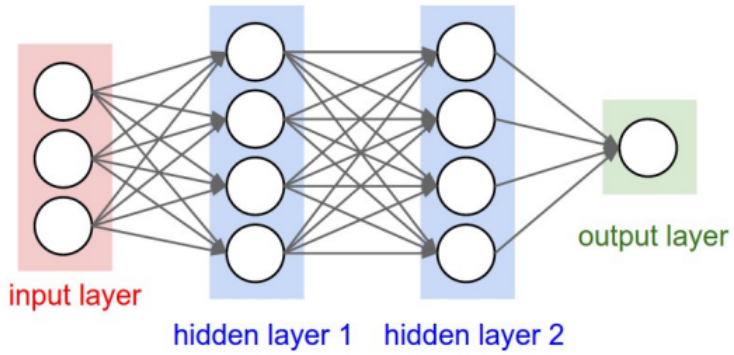

Figure 2.12: Multi-layer Perceptron

2.7.1.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) consist in a deep, feed-forward neural network that has successfully been applied to analyzing visual imagery. They use a variation of multi-layer perceptron designed to require minimal preprocessing.CNNs were responsible for major breakthroughs in Image Classification and are the core of most Computer Vision systems today, from Facebook's automated photo tagging to self-driving cars. However, they are beginning to be applied into Natural Language Processing problems and gotten some interesting results[16]. The idea is to explain of a CNN works in a image and then, extrapolates this information to text.

According to Wikipedia, a convolution is a mathematical operation on two functions in order to produce a third one, which is typically viewed as a modified version of one of the original functions, giving the integral of the pointwise multiplication as a function of the amount that one of the original function is translated.

Imagine an image as a matrix where each element represents a pixel, where a "0" is black and "1" is white. The convolution will be the result of applying a slide windows called kernel, filter or feature detector (typically, they are square matrix of $n * n$ dimensions), multiply its values element-wise with the original matrix and adding them as a new element of the result matrix. This operation is performed over all the whole matrix by sliding the filter.

The convolution layer is the main building block of a convolutional neural network. CNNs are formed by these layers accompanied with a non-linear activation function like ReLU or tanh. The key point is to apply the convolution over the input in order to produce the output. Each layer applies a series of different filters, which are going to be learned automatically during the training process.

There are two key aspects related to CNNs: Location Invariance and Compositionality. The aspect is provided by the pooling layer, which gives the model invariance to translation, rotation and scaling. The second one is provided by the filters of each convolutional layer,

each filter composes a local patch of lower-level features into higher-lever representation.

Pooling layers subsample their input. The most common way to do pooling it to apply a max operation to the result of each filter. You don't necessarily need to pool over the complete matrix, you could also pool over a window. Another hyperparameter for your convolutions is the stride size, defining by how much you want to shift your filter at each step.

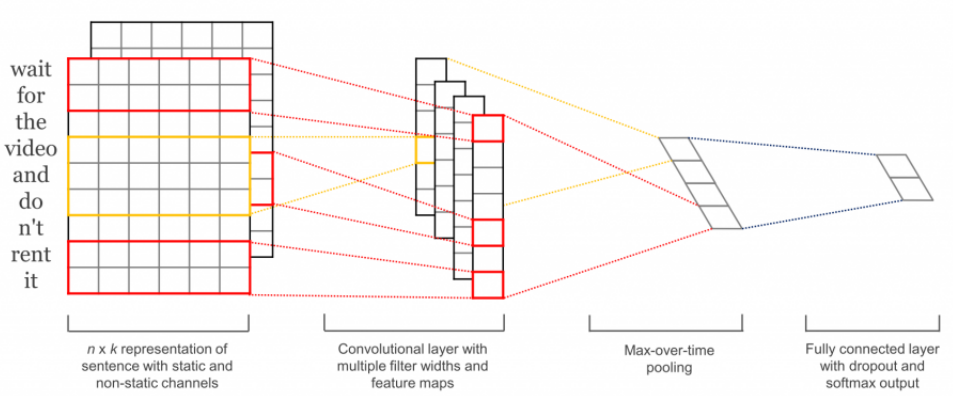

Figure 2.13: Convolutional Neural Network in Natural Language Processing

Instead of image pixels, the input to most NLP tasks are sentences or documents represented as a matrix. Each row of the matrix corresponds to one token, typically a word, but it could be a character. That is, each row is vector that represents a word. This part will be discussing on the section [Insertar referencia]

2.7.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) consist of a artificial neural network wherein connections between units form a directed cycle. The main idea behind this kind of networks resides in their particular networks, which have an internal memory to maintain information about the previous input. This feature has led them to become popular for analysis based on language, whether text, audio or video. That is because the network it is not taken the sentence isolated and analyzing it, it is picking up the context of each word from the words before it.

Moreover, Recurrent Neural Networks can handle sequential data of arbitrary length. What this exactly means is explained in figure 2.15. On the left the default feed forward network is shown which can just compute one fixed size input to one fixed size output. With the recurrent approach also one to many, many to one and many to many inputs to outputs are possible. One example for one to many networks is that you label a image with

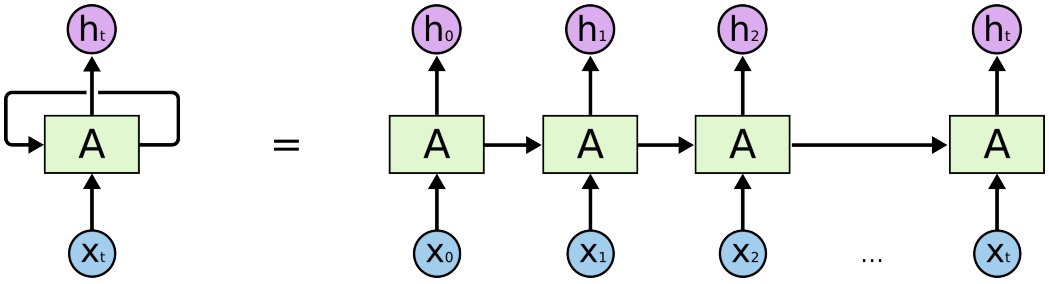

Figure 2.14: RNN context over the time

a sentence. The many to one approach could handle a sequence of images (for example a video) and produce one sentence for it and finally the many to many approach can be used for language translations.

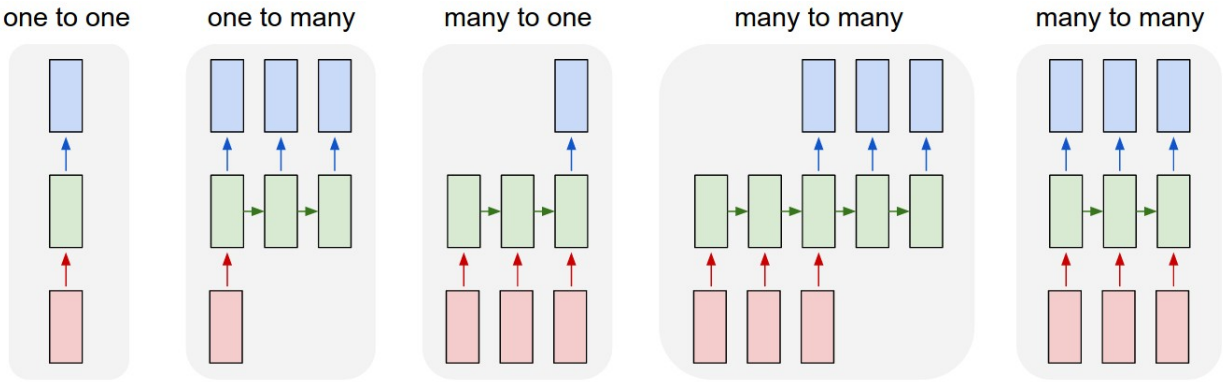

Figure 2.15: Sequential data

2.7.2.1 Long Short Term Memory

Long Short Term Memory networks, also knowns as LSTMs are a simple case of RNN which are capable of learning long-term dependencies. They were presented in 1997 [12] in order to deal with vanishing gradient and exploding gradient, two problems that have the RNN networks and make them unusable. LSTMs have become extremely popular in the past few years due to they are well-suited to classify, process and predict time series given time lags of unknown size and duration between important events.

The main idea behind this networks is the following: they implement a forgetting and adding mechanism. When a new input arrives to the network, the network first forgets any long-term information it decides it no longer needs. Then it learns which parts of the net input are worth using and saves them into its long-term memory. In the training process, the model learn which parts of its long-term memory are immediately useful. The next diagram 2.16 shows a LSTM building block.

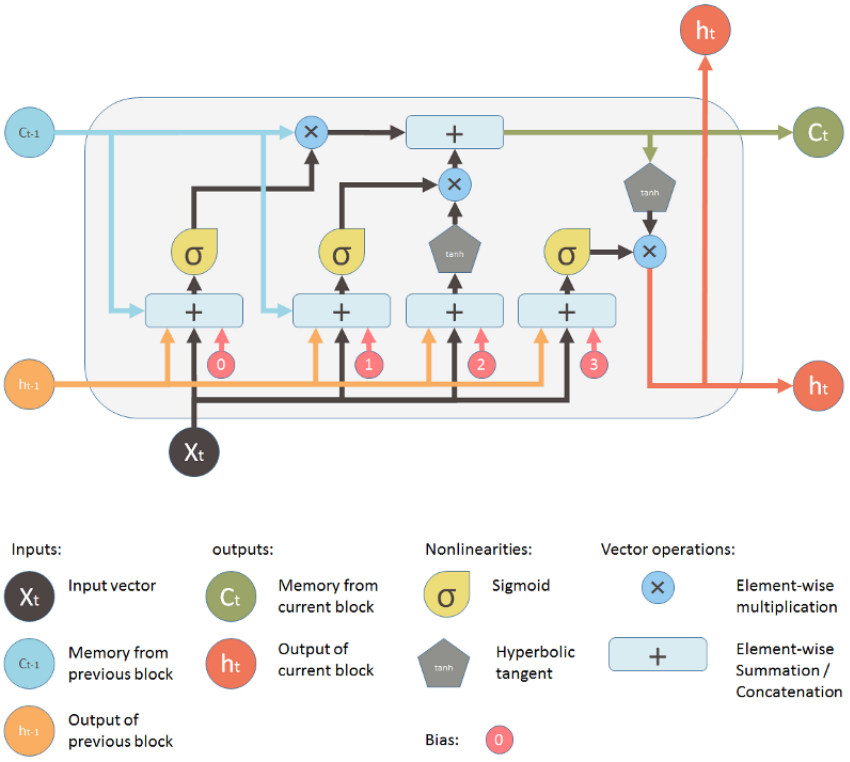

Figure 2.16: Long Short Term Memory Unit

According to the diagram, the network takes three inputs. X_t is the input of the sequence in that step, h_{t-1} is the output and C_{t-1} is the memory from the previous LSTM unit. Consequently, the outputs of the network are h_t, which is the currently output, and C_t the memory of the unit. Hence, the LSTM block generates its output accordingly to the new input, previous output and previous memory, then, they feed the next unit with its outputs and its memory.

The core idea behind LSTMs is the cell state, the horizontal line running through the top of the diagram. It links the previous memory with the output memory. Previously, it has been said that the LSTM implements a mechanism capable of forgetting and adding information. This mechanism is called cell gate, which let the information through accordingly to some criteria. They are composed of a sigmoid layer, which outputs "0" if the information does not worth enough and "1" to let it pass. An LSTM has three of these gates, to protect and control the cell gate.

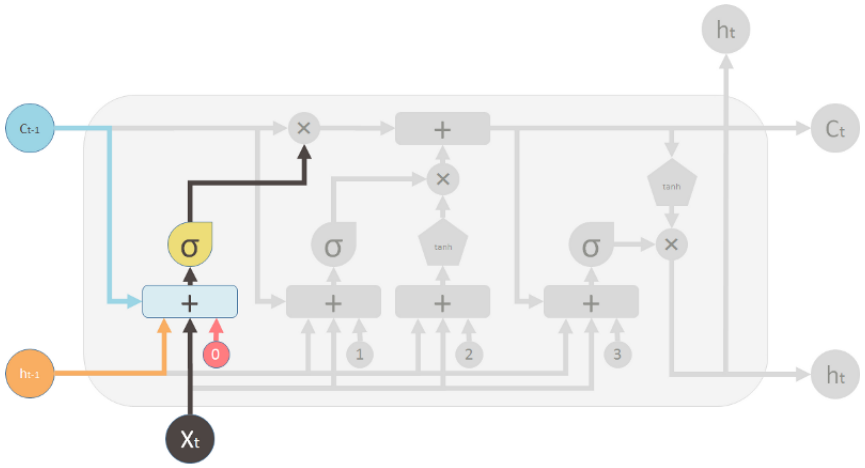

Figure 2.17: Long Short Term Memory Unit: Forget Gate

The first one is called the forget gate. This gate has as inputs, the output h_{t-1} and memory from the previous LSTM block, and X_t which is the input. The input of the sigmoid layer consists on the multiplication of a weight matrix with all the inputs and add a bias vector to the product. Then, it applies the sigmoid function as activation and decide to keep the input or discard it.

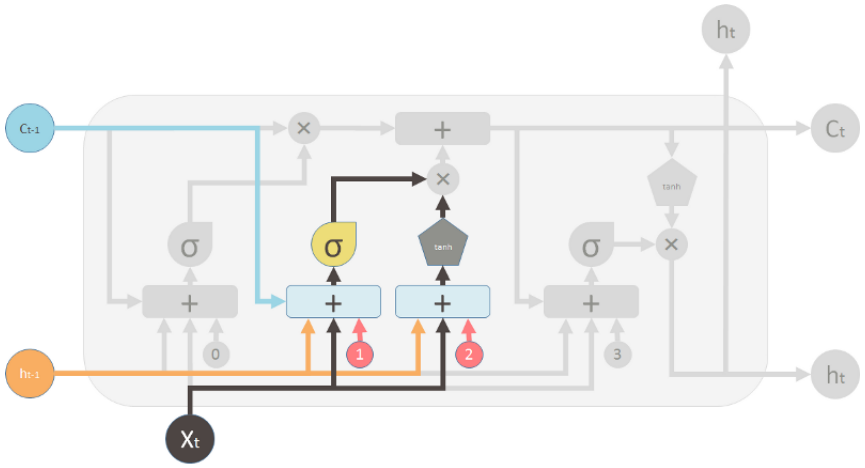

Figure 2.18: Long Short Term Memory Unit: Memory Gate

The second one is called the memory gate. This gate takes the same inputs as the forget gate. Its functionality is to decide what new information is going to affect the old memory. It is divided in two layers, the first one has the same structure as the forget gate and the second one is a tanh layer which creates a vector of candidates to be added to the old memory to form the new one.

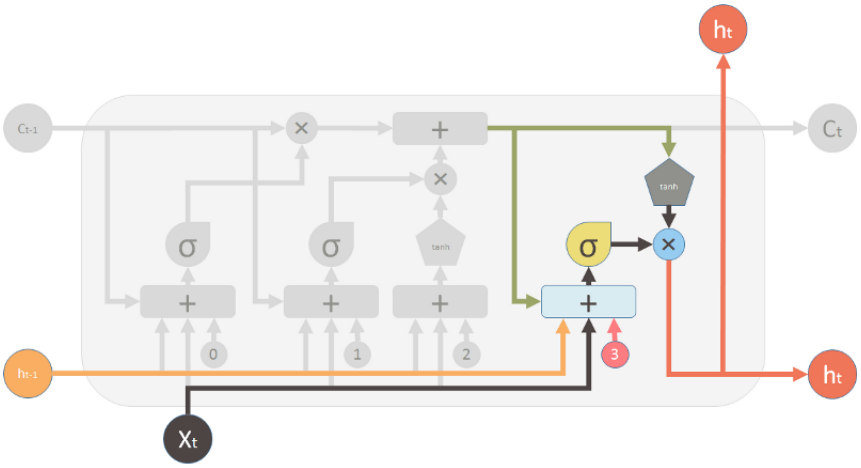

Figure 2.19: Long Short Term Memory Unit: Output Gate

The third one is called the output gate. This gate is connected to the new memory, which has been generated in the previous steps, and it controls the output value and the percent of memory that should output to the next LSTM block.

As it has been mentioned before, LSTM were specifically created to deal with the gradient problems. There are two factors that affect the magnitude of gradients; the weights and the activation functions (or more precisely, their derivatives) that the gradient passes through. If either of these factors is smaller than 1, then the gradients may vanish in time; if larger than 1, then exploding might happen. In the recurrency of the LSTM the activation function is the identity function with a derivative of 1.0. So, the backpropagated gradient neither vanishes or explodes when passing through, but remains constant. The effective weight of the recurrency is equal to the forget gate activation. So, if the forget gate is on (activation close to 1.0), then the gradient does not vanish. Since the forget gate activation is never ¿1.0, the gradient can't explode either. So that's why LSTM is so good at learning long range dependencies.

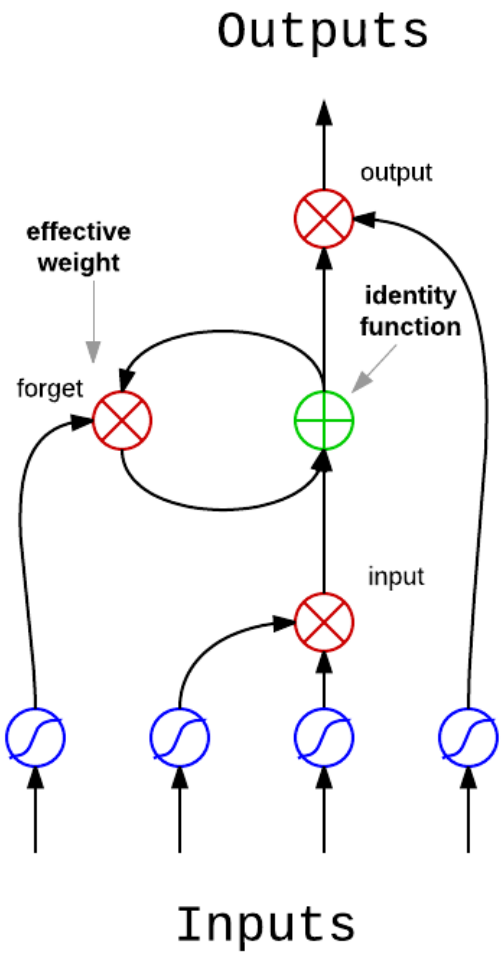

Figure 2.20: Long Short Term Memory Unit: Gradient Problem

2.7.2.2 Gated Recurrent Unit

Gated recurrent units (GRUs) are a gating mechanism in recurrent neural network [6]. It is a simpler variant of LSTMs that share many of the same properties. However, this networks doesn't have an output gate. Both GRU and LSTM networks can capture both long and short term dependencies in sequences, but GRU networks involve less parameters and so are faster to train.

It combines the forget and input gates into a single "update gate." It also merges the cell state and hidden state, and makes some other changes. The resulting model is simpler than standard LSTM models, and has been growing increasingly popular.

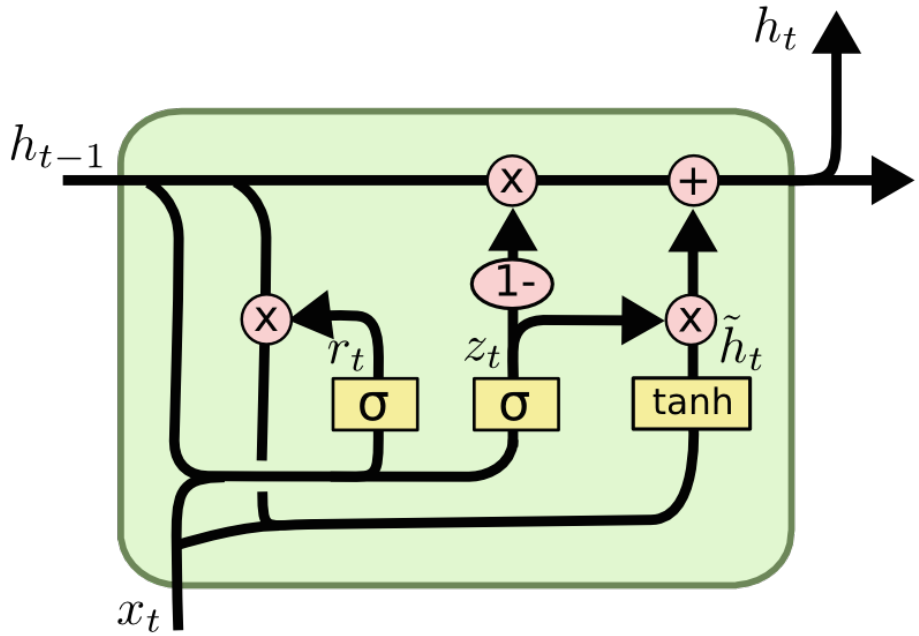

Figure 2.21: Long Short Term Memory Unit: Gradient Problem

The diagram 2.16 shows that the GRU block doesn't have the memory from the previous block and its inputs are only X_t and the output from the previous block H_{t-1}. The procedure of taking a linear sum between the exiting memory and the newly memory is very similar to the LSTM unit. However, the GRU does not have any mechanism to control the degree to which its state is exposed, but exposed the entire state each time.

Conceptually, a GRU network has a reset and forget "gate" that helps ensure its memory doesn't get taken over by tracking short term dependencies. The network learns how to use its gates to protect its memory so that it's able to make longer term predictions.

The differences between LSTM are few but it can be seen below:

- GRU has only two gates, meanwhile LSTM has three gates.
- GRU don't posses internal memory, as it has been presented in the diagram. They don't have the output gate which is presented in LSTMs.
- When computing the output is not necessary to compute a second nonlinearity.

2.8 Attention Mechanishm in Neural Networks

One of the most curious aspects of human understanding is the presence of attention. This particular feature allows us not to see an image or a text as something static or isolated, but instead allows the nuances within each of them to come to the forefront and endow them with a more complete meaning. A recent trend in Deep Learning are Attention Mechanism.

Attention Mechanism in Neural Network are based on this human capability. This concept has allowed the neural networks to learn alignments between different modalities. Some of this applications are: between image objects and agent action in the dynamic control problem [22], between speech frames and text in the speech recognition task [4], or between visual feature of a picture and its text description in the image caption generation task [50].

In the context of sentiment analysis, attention models are gaining a revealing role because not all the words that make up a phrase have the same meaning. Some words are more important than others depending on the message in a sentence. Attention models have proved that they improve the results in the sentiment analysis, such as the Attention Based-LSTM [47], the Cognition Based Attention Model [19] or the Structural Attention Neural Networks[17]. Let H be a matrix consisting of the output vectors $h1, h2, h3, ..., h_N$ that the MLP produced in the image 5.1, where N is the max vector size.

$$M = tanh(H) \tag{2.18}$$

$$\alpha = softmax(w^T \Delta M) \tag{2.19}$$

$$r = H \Delta \alpha^T \tag{2.20}$$

Where $H \in R^{dxN}$, $\alpha \in R^N$, $r \in R^d$, d is the dimension of the word vectors and w is a trained parameter vector and w^T its transpose. The weighted representation of a sentence is r and α is the attention weights vector. The final representation is given by:

$$h* = tanh(r) \tag{2.21}$$

CHAPTER 3

Enabling Technolgies

This chapter offers a brief review of the main technologies that have made possible this project, as well as some of the related published works.

This project exploits two types of technologies: deep learning and web services.On the Deep Learning side, Artificial Neural Networks are the techniques which is going to be used in this project for performing the analysis tasks.

On the other side, there are the different components that involves the development of the analysis services using the classifiers and the evaluation services. The main technology of the web services is Senpy, from which is going to be given a brief explanation. Futhermore, due to Senpy works in a Linked Data context, this section provides an overview of the JSON-LD format and the ontologies used in this project

3.1 Machine Learning Technologies

This section will give an overview of the main technologies used in the development of a sentiment and emotion classifier. In any case, the fields, for which these technologies have been used, are:

- Tensorflow and Keras have been used for developing the model of the Neural Network and the classifier.
- Scikit-learn has been used for building the pipeline classifier.
- GSITK and Pandas have been used for the pre-process and load the data. Moreover, GSITK has been used for performing the local evaluation of the models and the evaluation service.
- Gensim has been used for loading the Word2Vec vectors.

3.1.1 TensorFlow

TensorFlow is an open source software library for numerical computation using data flow graphs. It was develop by Google (specially Google Brain) and it was released on the last semester of 2015.

A computation expressed using **TensorFlow** can be executed with little or no change on a wide variety of heterogeneous systems, ranging from mobile devices such as phones and tablets up to large-scale distributed systems of hundreds of machines and thousands of computational devices such as GPU cards. The system is flexible and can be used to express a wide variety of algorithms.

This software visualizer of the graphs and some parameters during the process of training. It is call TensorBoard, which shows in "real time" information related to the graph and its components.

Nowadays it is probably one of the most used libraries for neural network, due to the fact that it has been developed to solved this kind of problems.

3.1.2 Keras

Keras is a high-level API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent and simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error. This makes **Keras** easy to learn and easy to use. This ease of use does not come at the cost of reduced flexibility: because Keras integrates with lower-level

deep learning languages (in particular TensorFlow), it enables you to implement anything you could have built in the base language. In particular, as tf.keras, the **Keras API** integrates seamlessly with your TensorFlow workflows.

Keras's purpose is to allow fast experimentation. The properties of Keras are:

- User Interface, the idea is that users can create their models through an easy and simple interface.
- Modularity: all the components needed for creating a neural network model are divided in modules, which provides a simple way of creating and adding them to a new model.

The main idea is to use the **Keras API** with **Tensorflow** as backend, in order to build the different neural network for performing the deeplearning analysis.

3.1.3 Scikit-learn

Scikit-learn[28] is an open source, BSD-licensed, Python library providing simple and efficient tools for data mining and data analysis. It is built on NumPy, SciPy, and matplotlib. Scikit-learn implements a range of machine learning, preprocessing, cross-validation and visualization algorithms. Scikit-learn exposes a wide variety of machine learning algorithms, both supervised and unsupervised, using a consistent, task-oriented interface, thus enabling easy comparison of methods for a given application. Since it relies on the scientific Python ecosystem, it can easily be integrated into applications outside the traditional range of statistical data analysis.

The main component of the scikit-learn package are the **estimators**. It is the general name of the classifier, regressors, transformers and clustering algorithms that are included in the scikit-learn library. A distinguishing feature of the scikit-learn API is its ability to compose new estimators from several base estimators. Composition mechanisms can be used to combine typical machine learning workflows into a single object which is itself an estimator, and can be employed wherever usual estimators can be used. In particular, scikit-learn's model selection routines can be applied to composite estimators, allowing global optimization of all parameters in a complex workflow. Composition of estimators can be done in two ways: either sequentially through **Pipeline objects**.

Pipeline objects chain multiple estimators into a single one. This is useful since a machine learning workflow typically involves a fixed sequence of processing steps (e.g., feature extraction, dimensionality reduction, learning and making predictions), many of which perform some kind of learning. A sequence of N estimators such steps can be combined into a **Pipeline** if the first $N - 1$ steps are transformers; the last can be either a predictor, a

transformer or both. The estimator and pipeline will be important parts in the development of this project.

3.1.4 GSITK

GSITK is a library developed by the Intelligent systems group (GSI) on top of scikit-learn that eases the development process on NLP machine learning driven projects.**GSITK** manages datasets, features, classifiers and evaluation techniques, so that writing an evaluation pipeline results fast and simple.

GSITK is divided in four main components:

- Datasets and DatasetManager. This module is based on datasets, they allow to load the information quickly . GSITK provides a series of default datasets that can be used, furthermore it allows you to import your own datasets by defining two files: a python and a yaml. The DatasetManager handled the datasets, it has persistence allowing you to load a dataset quickly once it has already been loaded a first time. All of the datasets are processed as a pandas dataframe. The way of creating and adding a dataset will be shown below.

- Feautures. This module can be used to extract features in a format supported by machine learning algorithms from datasets. Mostly of the components are build following the sklearn api, this allows them to be used as a part of a pipeline. Moreover, it provides some sentiment and deeplearning techniques for extracting features.

- Preprocess. This module provides tokenize and preprocess methods for cleaning the data before applying a machine learning technique.

- Evaluation. This module allows to create evaluations, this consists on evaluate one or more pipelines with one or more datasets. It will return the accuracy,f-score,prediction and recall of each pipeline-dataset evaluation. This allows the user to compare the different systems.

3.1.5 Pandas

Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. The main features about this library are:

- DataFrame object for data manipulation with integrated indexing.

- Tools for reading and writing data between in-memory data structures and different file formats.

- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of data sets.
- Label-based slicing, fancy indexing, and subsetting of large data sets.
- Data structure column insertion and deletion.
- Group by engine allowing split-apply-combine operations on data sets.
- Data set merging and joining.
- Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure.
- Time series-functionality: Date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging.

3.1.6 Gensim

Gensim[33] is an open source software for topic modelling with large corpora. Gensim provides a library capable of training word2vec models [9] and load pre-trained models. The main idea, it is to use Gensim to load pre-trained models for using them as a Embedding Layer in the ANNs. This Embedding Layer will be discussed in the chapter below.

The pre-trained models used in this project are:

- GloVe [1] is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space. The word embedding vectors are pre-trained on an unlabeled corpus whose size is about 840 billion.
- Google's pre-trained model[2], which includes word vectors for a vocabulary of 3 million words and phrases that they trained on roughly 100 billion words from a Google News dataset. The vector length is 300 features.
- FastText is an open-source, free, lightweight library that allows users to learn text representations and text classifiers. It works on standard, generic hardware. Models can later be reduced in size to even fit on mobile devices. They have published pre-trained word vectors for 294 languages, trained on Wikipedia using fastText [3]. These

[1]Pre-trained word vectors of Glove can be obtained from http://nlp.stanford.edu/projects/glove/

[2]Pre-trained word vectors of Google can be obtained from https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21

[3]Pre-trained word vectors obtained by FastText can be obtained from https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md

vectors in dimension 300 were obtained using the skip-gram model described in [1] with default parameters.

3.2 Senpy

Senpy[37] is an open source software developed by the GSI of the ETSIT-UPM. It is an implementation of a linked data model used for emotions and sentiments analysis. It is based on Marl, Onyx and NIF[13] semantic vocabularies and supports different formats as JSON-LD, Turtle, XML-RDF or simple text input. **Senpy** is a framework that turns your text analysis algorithm, which can be of any kind: sentiment, emotion, gender, etc. into a full blown semantic service. Senpy takes care of:

- Interfacing with the user: parameter validation, error handling.
- Formatting: JSON-LD, Turtle/n-triples input and output, or simple text input.
- Linked Data: senpy results are semantically annotated, using series of well established vocabularies, and same default URIs.
- User interface: a web UI where users can explore your service and test different settings.
- A clien to interact with the service. Currently only available in Python.

Senpy proposes a modular and dynamic architecture allowing the developer to implements its algorithm as a service, yet offering a common interface. The idea is that a developer a developer does not have to worry about the implementation of the plugin but the algorithm itself. Furthermore, it fosters the creation of common tools such as service validators, evaluation suites and testing tools.

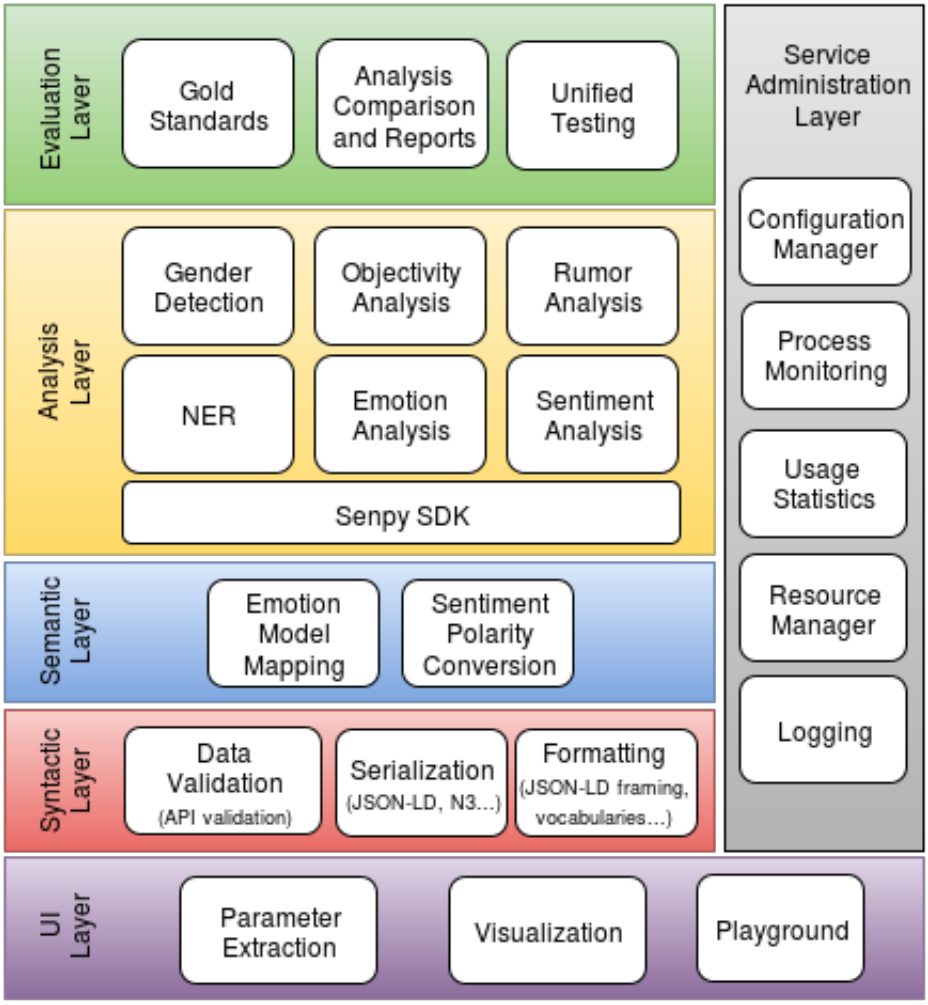

Figure 3.1: Senpy framework. Each layer represents a functional block in a service

The framework covers all the aspects of developing, publishing and using a sentiment analysis service. These aspects are grouped into layers: the Analysis Layer includes the core NLP process and the libraries to connect it to the rest of the layers; the Semantic Layer deals with conceptual models and their integration; the Syntactic Layer handles issues such as formatting, serialization and input/output validation; the User Interface (UI) is the way in which users interact with services; the Evaluation Layer allows users to benchmark different algorithms; and the Service Administration Layer offers tools and information to control running services.

The architecture of Senpy consists of two main modules: Senpy core, which is the building block of the service and Senpy plugins, which contain the code for each analysis algorithm. Figure 3.2 shows a simplified version of the processes involved in an analysis with the Senpy framework.

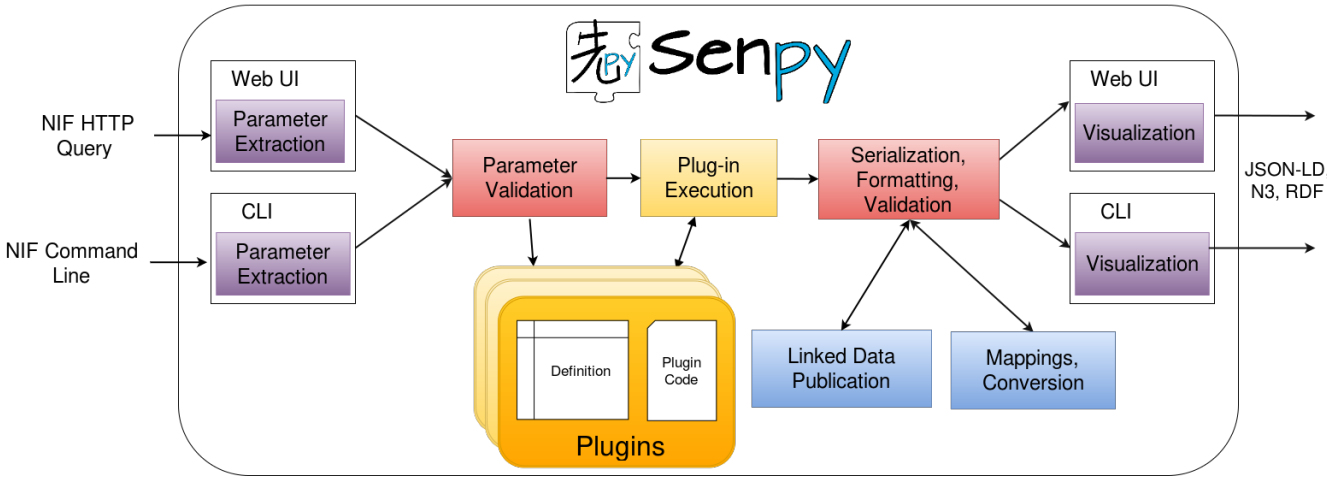

Figure 3.2: Senpy architecture

The plugins are defined by two files: one with the code and another one with all the information about it, which acts as a definition file. This definition file is written in YAML or JSON format and it provides the name, module to load(the code file), version, author, parameters and variables needed for the correct functioning,etc. It is ".senpy" file.

On the other hand, the code file has all the implementation of the algorithm. Moreover, the developer has to define the analyse function which process the input, called the algorithm for the analysis part and make the output following the Marl Ontology, in case it is a sentiment analysis, or the Onyx Ontology, in case it is an emotion analysis. Furthermore, there is an activation function which is in charge of loading all the resource needed for the correct functioning, if the algorithm needs to be trained this is where it happens.

3.3 Linked Data

Linked Data empowers people that publish and use information on the Web. It is a way to create a network of standards-based, machine-readable data across Web sites. It allows an application to start at one piece of Linked Data, and follow embedded links to other pieces of Linked Data that are hosted on different sites across the Web.

JSON-LD is a lightweight Linked Data format. It is easy for humans to read and write. It is based on the already successful JSON format and provides a way to help JSON data inter operate at Web-scale. JSON-LD is an ideal data format for programming environments, REST Web services, and unstructured databases such as CouchDB and MongoDB.

On the Semantic Web, vocabularies define the concepts and relationships (also referred to as "terms") used to describe and represent an area of concern. Vocabularies are used to classify the terms that can be used in a particular application, characterize possible

relationships, and define possible constraints on using those terms. In practice, vocabularies can be very complex (with several thousands of terms) or very simple (describing one or two concepts only).

There is no clear division between what is referred to as "vocabularies" and "ontologies". The trend is to use the word "ontology" for more complex, and possibly quite formal collection of terms, whereas "vocabulary" is used when such strict formalism is not necessarily used or only in a very loose sense. Vocabularies are the basic building blocks for inference techniques on the Semantic Web.

There are two ontologies used by Senpy: Marl [4] and Onyx[5]. Senpy used them for retrieve and publish the information in JSON-LD format.

3.3.1 Marl Ontology

Marl is a vocabulary that annotates and describes subjective opinions expressed on the web or in particular Information Systems. Marl is based in the W3C's Provenance [6], which provides information related to the entities, activities and people involved in producing some kind of data. Marl is focused on Semantic Analysis, which transform plain data into semantic sentiment information.

The goals of the Marl ontology to achieve as a data schema are:

- Enable to publish raw data about opinions and the sentiments expressed in them.
- Deliver schema that will allow to compare opinions coming from different systems (polarity, topics, features)
- Interconnect opinions by linking them to contextual information expressed with concepts from other popular ontologies or specialized domain ontologies.

The main classes of the Marl Ontology are: Aggregated Opinion, Opinion, Polarity and Sentiment Analysis. AggregatedOpinion is the set of Opinions specified in the "extractedFrom" source. The Sentiment Analysis represents the algorithm or the source that produces a marl:Opinion.

The SentimentAnaylsis instance contains information about the source from which the information was taken, the algorithm or source used to process it, and the polarity range that it returns. The SentimetnAnalysis generates an opinion (marl:opinion) or a set of opinions (marl:AggregatedOpinion) derived from many sentiments expressed in one text.

[4]Marl Ontology: http://www.gsi.dit.upm.es/ontologies/marl/
[5]Onyx ontology: http://www.gsi.dit.upm.es/ontologies/onyx/
[6]PROV Ontology: https://www.w3.org/TR/prov-o/

Basically, each opinion has a Polarity (marl:polarity), which indicates if the opinion is positive/negative or neutral; and a Polarity Value, which is the numerical value for the opinion.

The Marl class diagram presented in figure 3.3 shows connections between classes and properties used for descrbing opinions.

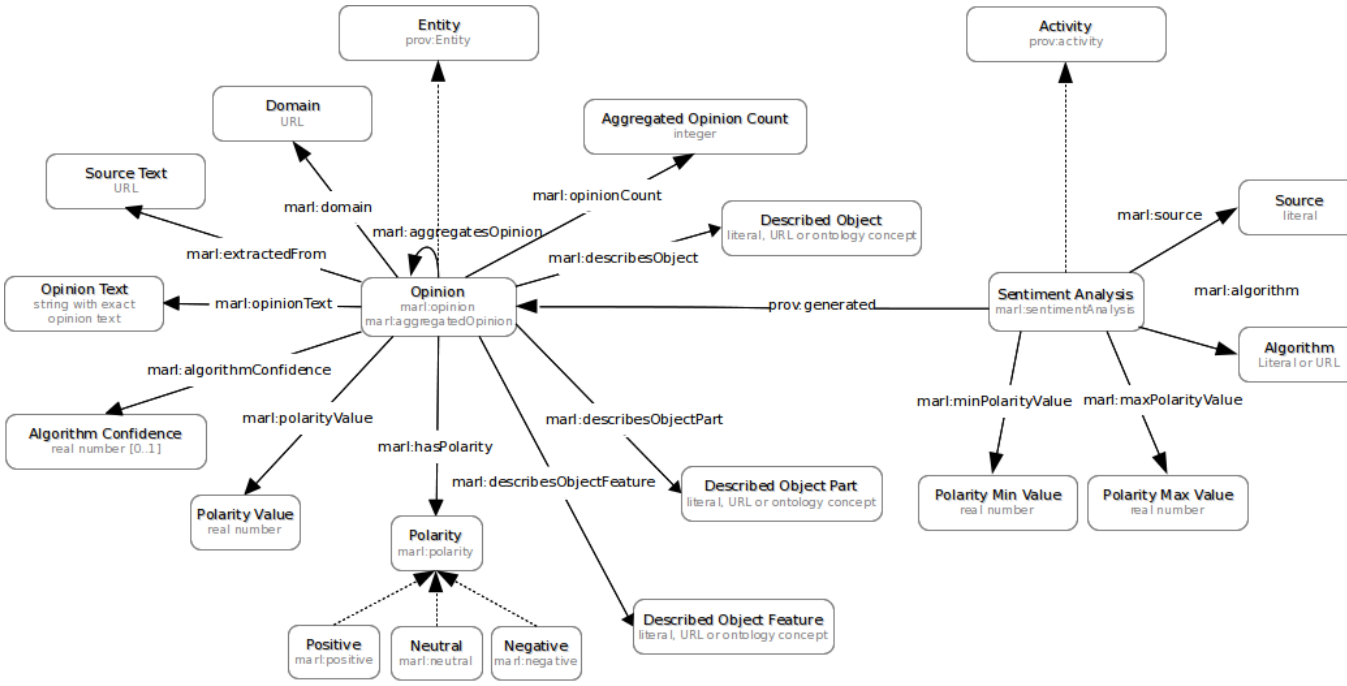

Figure 3.3: Class and Properties Diagram for the Marl Ontology

3.3.2 Onyx Ontology

Onyx[36] aims to complement the Marl Ontology by providing a simple means to describe emotion analysis processes and results using semantic technologies. Such as Marl Ontology, the idea behind Onyx is to represent the results of an emotion analysis service as a sematic format.

The goals of the Onyx ontology to achieve as a data schema are:

- Enable to publish raw data about emotions in user-generated content.
- Deliver schema that will allow to compare emotions coming from different systems (polarity, topics, features).
- Interconnect emotions by linking them to contextual information expressed with concepts from other popular ontologies or specialized domain ontologies.

At its core, the ontology has three main classes: Emotion, EmotionAnalysis and EmotionSet.

Basically, the EmotionAnlaysis acts as the SentimentAnalysis class from the Marl Ontology but it has an important property, the EmotionModel indicates the model in which the Analysis is based. Each EmotionModel will be linked to the different categories it contains (EmotionCategory), the Appraisal or Dimension instances it introduces (through Appraisal and Dimension), etc.This analysis generates an EmotionSet which contains a group of Emotions as a results of the analysis part.

Each Emotion has a EmotionCategory, which is the specific category of emotion such as sadness, anger, joy, etc. The EmotionIntensity corresponds to the numerical value associated to the Emotion. In case that a dimensional model is specified, it can be defined by subclassing Appraisal, Dimension and ActionTendency, whose value should be a float number.

The Onyx class diagram presented in figure 3.4 shows connections between classes and properties used for describing opinions.

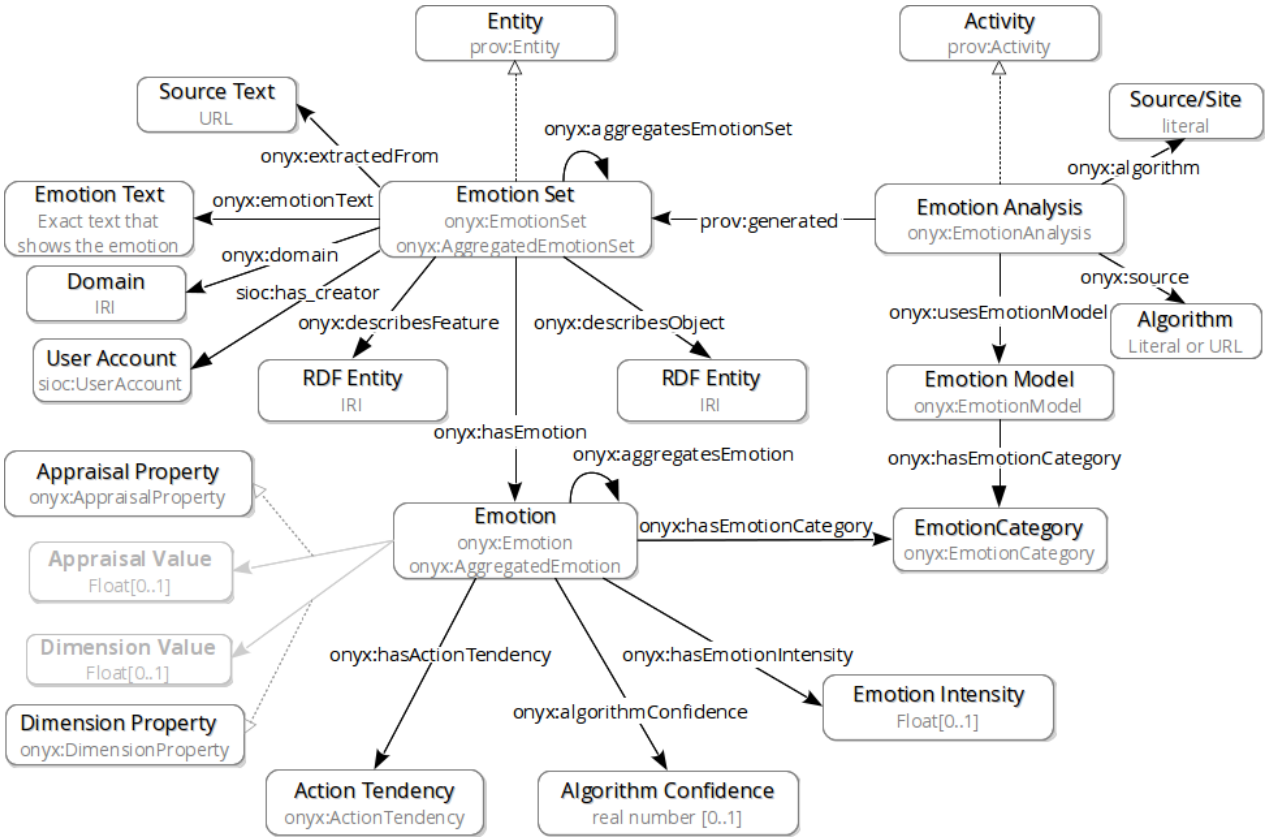

Figure 3.4: Class and Properties Diagram for the Onyx Ontology

CHAPTER 4

Architecture and Methodology

This chapter presents the methodology used in this work. It describes the overall architecture of the project, with the connections between the different components involved on the development of the project.

4.1 Methodology

The aim of the section is to provide an overview of the methodology and phases used for the development of the project. The project has been divide into three main phases in order to achieve a solution for each of the proposed goals. Diagram 4.1 shows an overview of the development phases of this project.

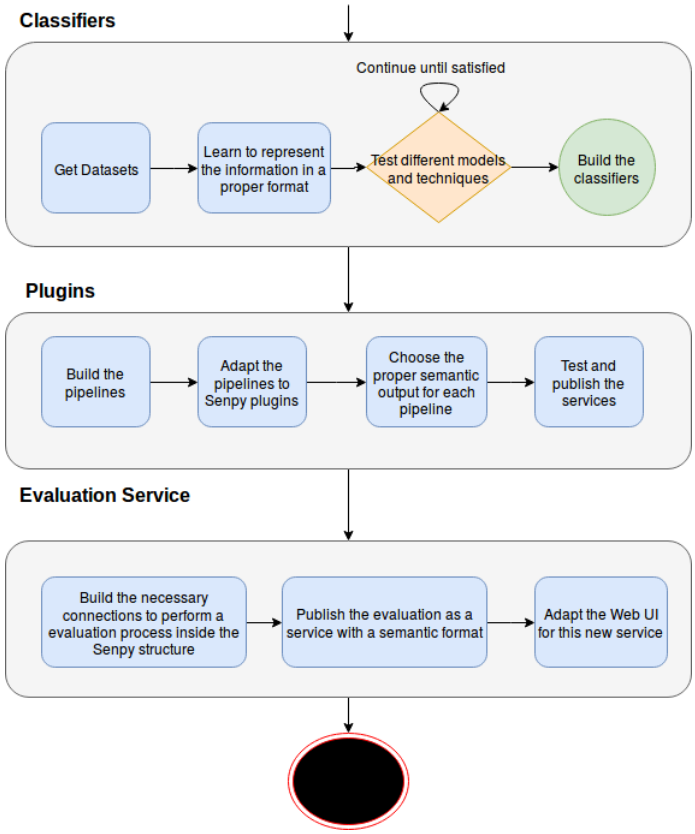

Figure 4.1: Project phases

4.2 Architecture

Figure 4.2 shows all the connections between the different components used in this project.

On the one hand, there is Senpy framework, which will handle the queries of the evaluation service. This queries will contain the information of the plugins and datasets that it is wanted to evaluate. First of all, the evaluation service will request the needed datasets to GSITK DatasetManager. Once it get them, it passes these datasets to the different plugins that are wanted to evaluate. Each one of the plugins has a evaluation handler that will request the evaluation to the correspondent module in GSITK.

On the other hand, there are the plugins that have been developed. This plugins have been incorporated the pipelines as their main component. They also have the evaluate handler.

When a evaluation is finished, each plugin apply the semantic wrapper in order to transform these results into a Linked Data format. The service will publish these results using the API Endpoint or the WEB UI. There are not included in this architecture the main components of Senpy related to the validation, serialization , etc., which can be seen in figure 3.2.

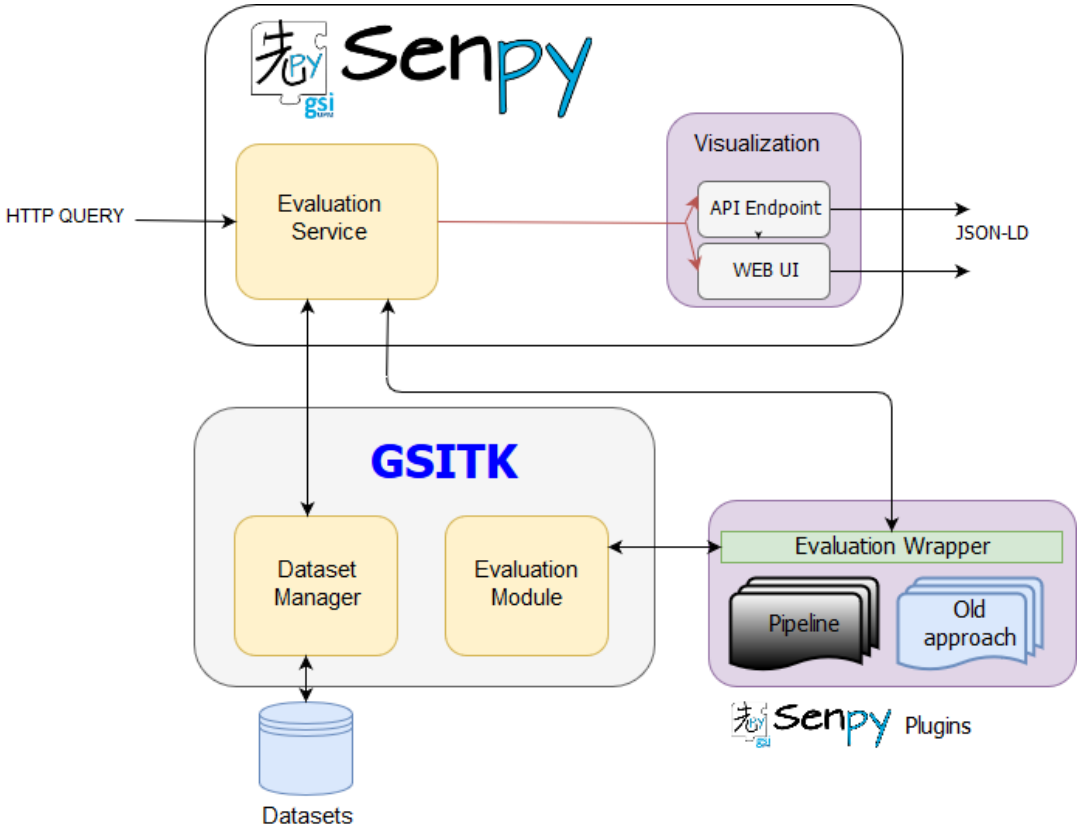

Figure 4.2: General Architecture

CHAPTER 5

Deep learning for Sentiment and Emotion Analysis

The previous sections have described the technologies and theory for developing a text classifiers using the Neural Network techniques.

This section will discuss the two use case: Sentiment and Emotion Analysis applying Deep Learning techniques, which will include the datasets used, the first steps with the neural networks and the use of more advance techniques. The section will provide an overview of the technique and a architecture diagram. Finally, there will be a conclusion with a cooperation over all the different approaches.

5.1 Sentiment Analysis Using Deep Learning Techniques

This section will describe several neural networks that have been selected and implemented using Keras. The aim of this section is to compare the results with conventional machine learning techniques and give a conclusion over all the models.

The datasets used for training and evaluating the classifiers are:

- **Stanford Sentiment Treebank** corpus (STT) is based on a dataset of movie reviews [25]. It consists in a subset of 11855 single sentences. The dataset is divided into train, dev and test sets provided with fine-grained labels (very positive, positive, neutral, negative, very negative). The Stanford Sentiment Treebank is the first corpus with fully labeled parse trees that allows for a complete analysis of the compositional effects of sentiment in language. This dataset will be used as a binary dataset too.

- **IMDB** dataset consists of 25000 movies reviews from IMDB, labeled by sentiment (positive/negative) as a train set, and another 25000 for the test set. This dataset will be used due to its volume is data is higher than the SST.

5.1.1 First steps

First of all, there are some hyperparameters that need to be defined in order to create the first base models. The idea is to provide an overview of this parameters with theirs values and the explanation behind them. Then, the three base models will be presented with an overview of the architecture and the different results of this three base models.

Batch size, defines number of samples that going to be propagated through the network. The main idea is that the network takes the first n samples, where n is the batch size defined previously. The network trains with the n samples and the pick up others for training. Choosing small batches present some advantages over a big batch. They require less memory, since you train network using less number of sample the overall training procedure require less memory. It's especially import in case that the dataset doesn't fit in memory. Typically networks trains faster with mini-batches. That's because they update weights after each propagation. However, the smaller the batch the less accurate estimate of the gradient. For performing these experiments, it has been selected a batch size of **32**.

One of the main problems with neural networks relies on the overfitting. Normally, the models have a large number of free parameters that can describe an amazingly wide range of phenomena. Even if such a model agrees well with the available data, that doesn't make it a good model. It may just mean there's enough freedom in the model that it can describe almost any data set of the given size, without capturing any genuine insights into the underlying phenomenon. When that happens the model will work well for the existing

data, but will fail to generalize to new situations. The true test of a model is its ability to make predictions in situations it hasn't been exposed to before.

There are other techniques which can reduce overfitting, even when there is a fixed network and fixed training data. These are known as regularization techniques. The idea is to use one of this techniques, known as weight decay or L2 regularization. L2 regularization is based on adding an extra term to the lost function, a term called the regularization term.

Moreover, a dropout layer prevents co-adaptation of hidden units by randomly dropping out a proportion p of the hidden units during forward backpropagation [11]. In this project, it has been applied a dropout on the layer after the MLP with a constraint on L2-norms of the weights vector.

Another way to prevent overfitting is to set an appropriate number of iteration. The reason behind this resides on the continue updating of the parameters of the neural network during the training process. For example, if the number of iteration is high, the model will have adapted too well for the training data and it will be useless outside of it. In order to avoid this situation, it can be defined a proper number of iteration or a "stop" that will pause the training process when a condition is reached. For all the experiments, it will be used the second approach. Using the development data as a condition, when the accuracy of this set does not undergo any change, it means that the model is beginning to have overfitting and therefore the training process is stopped in order to extract the model at that this moment.

Loss functions, is another important parameter to decide. Cross entropy is the most used loss function when a classification task is performed. During the back-propagation training, it is wanted to control the output node values to either 1.0 or 0.0 depending on the target values. In the MSE case, the gradient contains a term of (output)*(1-output). As the compute output gets closer and closer to either 1.0 or 0.0 the value of this term will be smaller and smaller. If the gradient is small, the change in weight will be smaller too and the training will be stall out. However, with cross entropy this term is eliminated and the training process is not likely to stall out. Therefore, the loss function will be categorical cross entropy and binary cross entropy (this depends on the dataset that will be used).

Activation function, is the final activation to apply in the output node for obtaining the results of the classification task. As it has been saw in previous section 2.5, there are several activation function but two of them are specially used for classification task: sigmoid and softmax. The sigmoid function will be used in binary classification and the softmax function will be used in categorical classification.

Optimizers, play a very crucial role to increasing the accuracy of the model. At this point, the models have used the RMSprop, which has worked well for all the models, but several optimizers have been tested with the final model to appreciate the different results.

This list provides a summarize of all the parameters that will be used as part of all the models:

- Batch size, 32.
- Loss function, binary and categorical cross entropy.
- Activation function, sigmoid and softmax.
- Optimizer, RMSprop.
- Dropout with L2 regulizer, rate 0.5.

Once all the parameters have been set, the architecture of the base model is described in figure 5.1. $w_1, w_2, w_3, ..., w_N$ represent the word vector in a sentence, whose length is N. $h_1, h_2, h_3, ..., h_N$ correspond to the output of the MLP and it is the hidden vector. The MLP (Multilayer Perceptron) is composed of a Embedding layer, LSTM/GRU/CNN layer, a Dropout layer and the activation function.

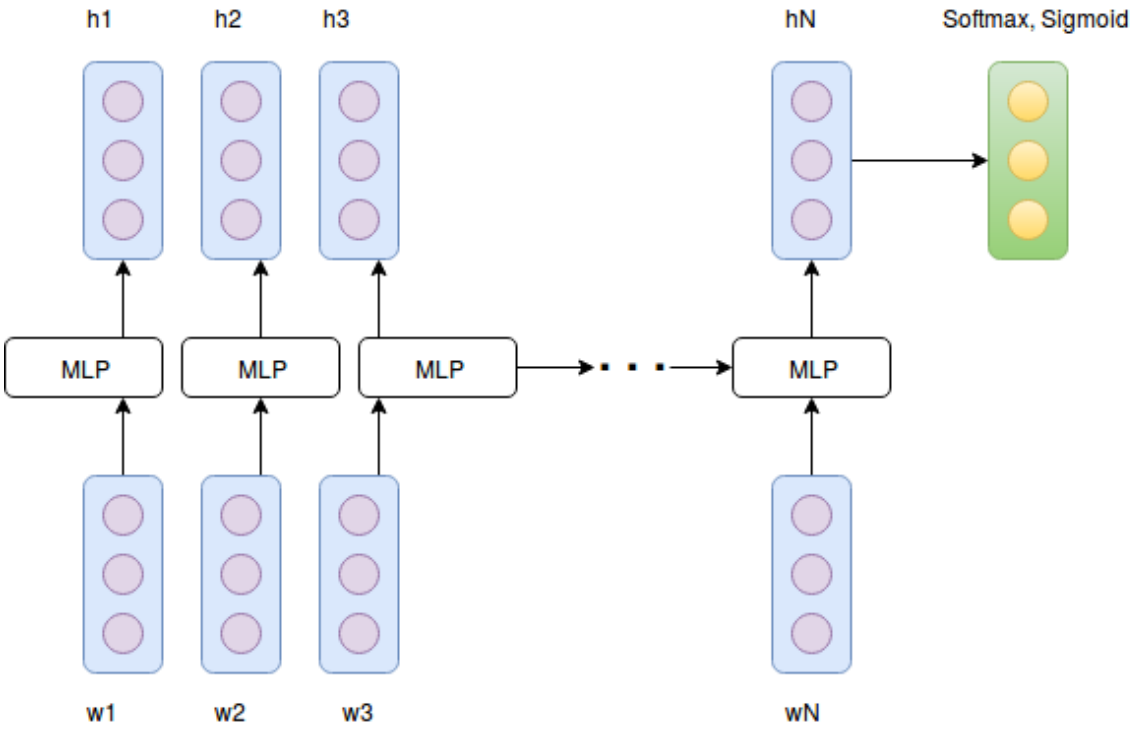

Figure 5.1: Base Model

Next two tables show the results of the three base models: **GRU**, **LSTM** and **CNN**. They have been evaluated using the GSITK library and the SST dataset. As it can be seen the GRU have performed it slightly better than the other two models in both tasks

(binary and 5-labels). GRU may be less prone to overfitting on some small datasets, such as the SST, since it only has two gates while LSTM has three. The idea is that GRU are less computationally demanding hence faster to train and better for smaller datasets while LSTM are better for large datasets where models that retain a longer timestep understanding perform better. This statement will be shown in the next section where a much larger dataset (IMDB) is going to be used for training a GRU and a LSTM networks with an attention layer.

Comparing CNN with GRU and LSTM is a problematic matter, this is like comparing an apples to birds question.A convolutional network is basically a standard neural network that's been extended across space using shared weights. A recurrent neural network is basically a standard neural network that's been extended across time by having edges which feed into the next time step instead of into the next layer in the same time step. There's a kind of similarity between the two but it's pretty abstract.

Intuitively, it can be seen that CNNs are ideal for images and videos while RNNs are ideal for text and speech. Although, as it can be seen, CNNs are quite useful even in text classification. But, as it can be seen in tables 5.1 and 5.2, CNNs performed it worse than the other but very faster (7 times faster). Dispite of this fact, the two approaches could be combined into one network (RNN+CNN) and this possibility would be explore in the next phase.

Table 5.1: Base-model results for binary task

Model	Accuracy	Precision_macro	Recall_macro	F1_weighted	F1_micro	F1_macro	Time/epoch
GRU	0.818781	0.821902	0.818862	0.818352	0.818781	0.818366	25
LSTM	0.808347	0.810585	0.808277	0.80799	0.808347	0.807976	28
CNN	0.802306	0.839308	0.746975	0.790454	0.792754	0.790237	3

Table 5.2: Base-model results for 5-label task

Model	Accuracy	Precision_macro	Recall_macro	F1_weighted	F1_micro	F1_macro	Time/epoch
GRU	0.41448	0.419576	0.371883	0.387326	0.41448	0.362428	23
LSTM	0.413122	0.405718	0.390666	0.408845	0.413122	0.394071	25
CNN	0.397738	0.457578	0.314515	0.252742	0.253441	0.250189	3

Section 3.1.6 talked about the pre-trained word2vec model that has been used for developing the different experiments. At this point, the main base model will be the GRU due to its accuracy. Table 5.3 shows the accuracy of each pre-trained model. Using a pre-trained word2vec model has increase the accuracy of our model, but one of them is given best results than the other one, **Glove**. Thus, this is the pre-trained word2vec that will be use for transforming the words into sequences.

Table 5.3: Word2vec pre-trained models evaluation

Model	Word2Vec	Binary	5-granularity
GRU	Google	0.842394	0.467873
GRU	FastText	0.842394	0.390045
GRU	**Glove**	**0.856123**	**0.473756**
LSTM	Google	0.843493	0.437557
LSTM	FastText	0.808347	0.413122
LSTM	**Glove**	**0.855574**	**0.473303**

Next graphs 5.2, 5.3 and 5.4 have been extracted during the training process. These graphs try to show a general vision of the important variables within the training process for basic embedding and one using the pre-trained word2vec model Glove. The variables that are being monitored are: accuracy, loss and loss of the validation data (val loss). As it can be seen, the training with the pre-trained model starts on a much higher basis with respect to the basic model.

The curve of the pre-trained model is much smoother and has no sudden changes. On the other hand, the curve of the basic model is much more abrupt and hits abrupt changes in its accuracy reaching better results in the training process but worse in general. The answer resides in the other two graphs. While the training loss is decaying, the validation loss is rising. This is because the model is adapting its training parameters only for the training set, therefore the validation set has worse results and validation loss will rise, entering in an overfitting state.

Abscissa axis corresponds to the number of epoch during the training process. This number is five but at the top of this section, the number that has been decided was 20. The reason is the "early stop" that it has been discussed before, when validation loss does not undergo any change or becomes worse, the model gives one more try. In case it gets worse

it will stop the training process to avoid overfitting.

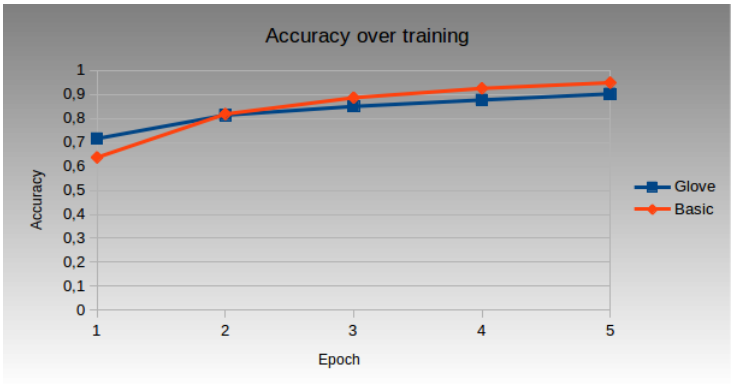

Figure 5.2: Evolution of the accuracy during the training process

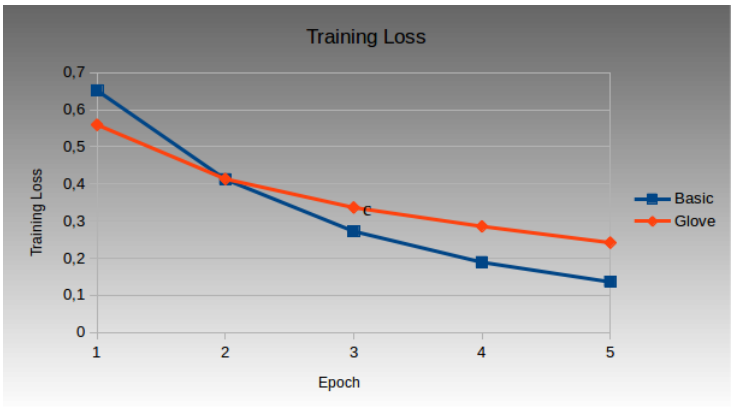

Figure 5.3: Evolution of the loss during the training process

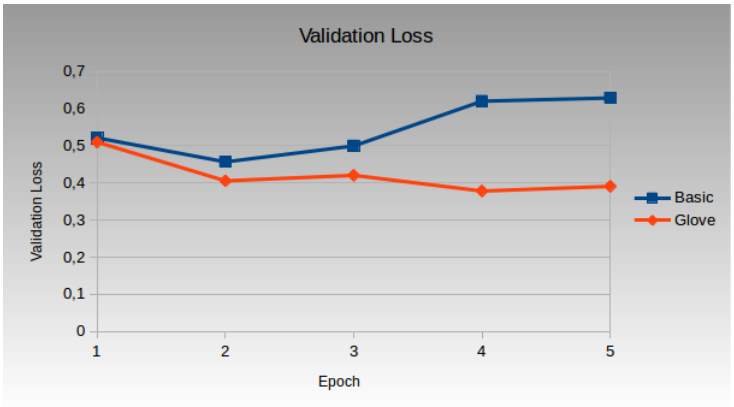

Figure 5.4: Evolution of the validation loss during the traning process

5.1.2 Attention Layer

Section 2.8 describes the mathematical explanation behind an attention mechanism, but let's see the implementation of this mechanism inside the base model architecture.

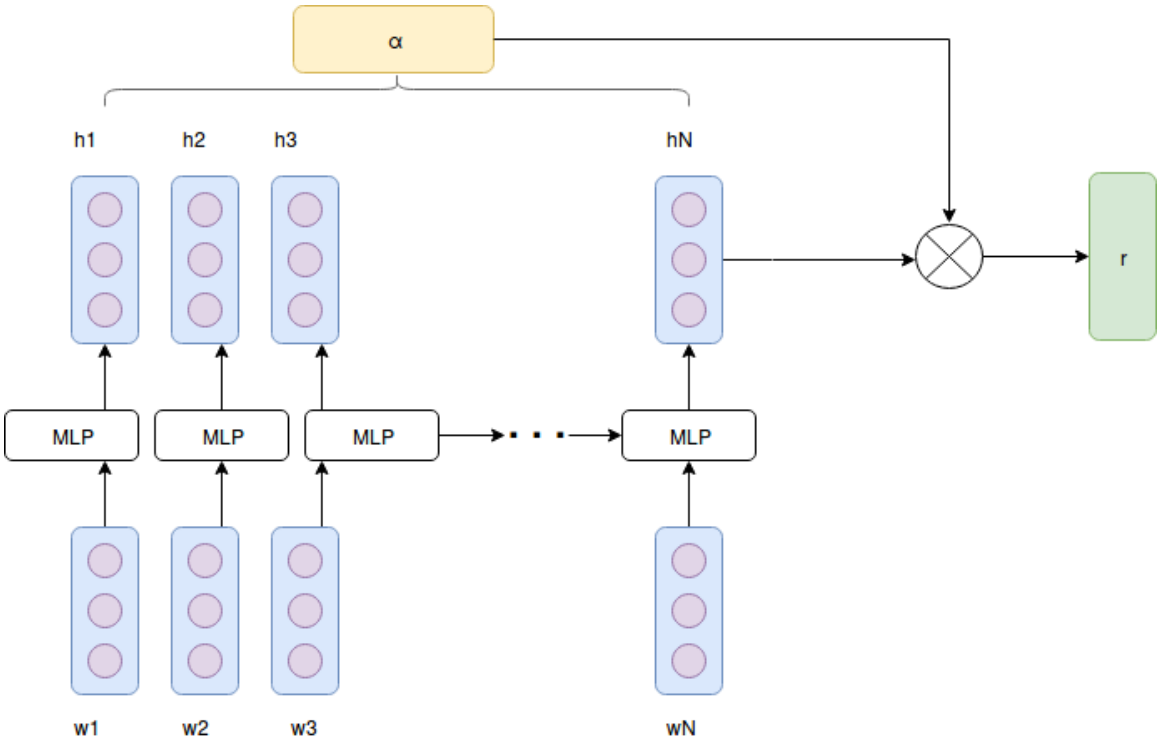

Figure 5.5: Attention Mechanishm inside the model

This would apply attention to the output of the MLP, and also to the output/input passed to the next unit. In this case, the linear output of the neurons would be squashed directly by the softmax/sigmoid. Basically, this attention mechanism allows the network to refer back to the input sequence, instead of forcing it to encode all information into one fixed-length vector.

The results achieved by adding the attention layer are better than the ones from the base models using the Glove pre-trained word2vec model. In table 5.4, the GRU continues being superior than the LSTM network in both categories binary and categorical.

Table 5.4: Attention Layer Results

Model	Word2Vec	Binary	5-granularity
GRU+ATT	**Glove**	**0.873147**	**0.473303**
LSTM+ATT	Glove	0.870597	0.450679

5.1.3 Combining and exploring models

This section will explore the Bidirectional networks and the CNN Recurrent Neural Network. Both models are going to be based on the architecture of section 5.1.1.The reasons behind using this type of networks, is that both complement the basic models of RNNs, making them more complete and better their basic qualities.

Bidirectional RNNs[40] are an extension of traditional RNNs that can improve model performance on sequence classification problems. In problems where all timesteps of the input sequence are available, Bidirectional RNNs train two instead of one RNNs on the input sequence. The first on the input sequence as-is and the second on a reversed copy of the input sequence. This can provide additional context to the network and result in faster and even fuller learning on the problem. Figure 5.6 shows a general overview of this networks.

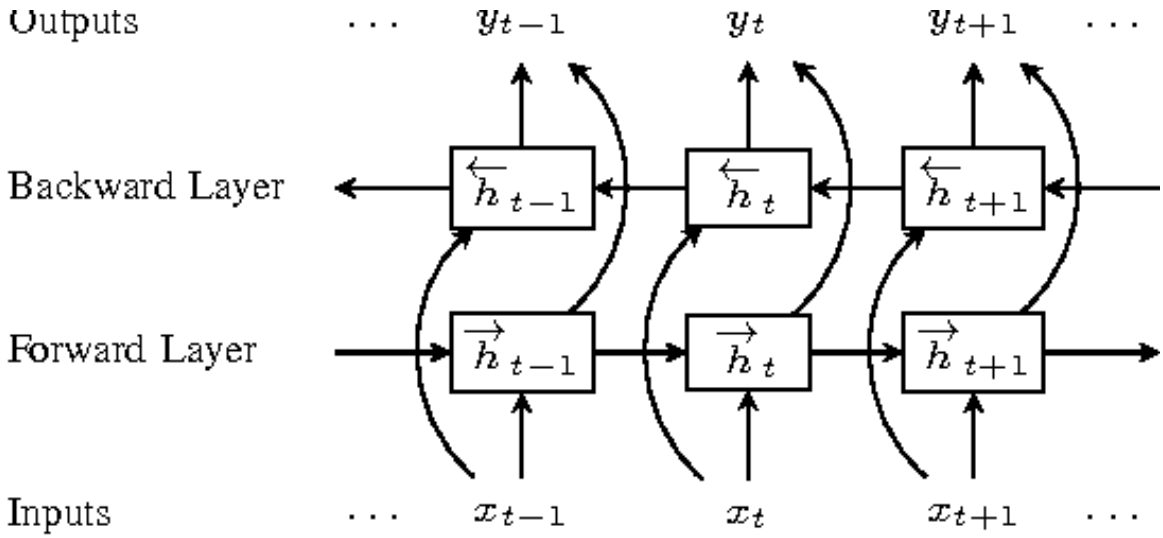

Figure 5.6: Bidirectional Recurrent Neural Network Architecture

Basically, it involves duplicating the first recurrent layer in the network so that there are now two layers side-by-side, then providing the input sequence as-is as input to the first layer and providing a reversed copy of the input sequence to the second, using all available input information in the past and future of a specific time frame. This networks have been used in sentiment analysis task achieving good results [24].

In the other hand, there are the CNN RNN networks. This networks involves using a Convolutional Neural Network (CNN) layers for feature extraction on input data and combine its output with a RNN, which provides sequence prediction. This architecture is used for the task of generating textual descriptions of images. Key is the use of a CNN that is pre-trained on a challenging image classification task that is re-purposed as a feature

extractor for the caption generating problem [45].

Moreover, they have been used on speech recognition and natural language processing problems where CNNs are used as feature extractors for the LSTMs on audio and textual input data. In particular, this articles describes the implementation of this network on the Sentiment Analysis field [46], achieving superior results than other experiments.

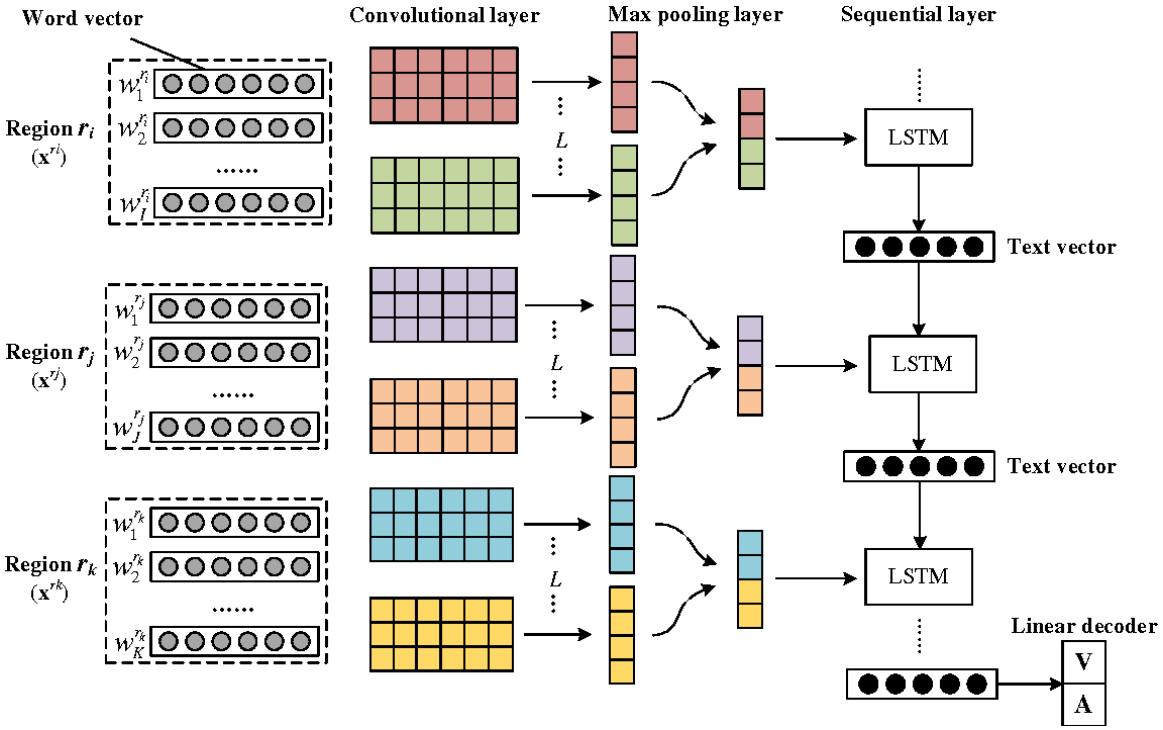

Figure 5.7: C-LSTM Neural Network

Results obtained with the Bidirectional Networks are slightly better than the Attention model by itself. However, the CNN+RNN and the CNN+BiRNN have achieved worse results. This mean that the parameters and all the hyper-parameter phase haven't adapted pretty well to the models. Adapt and improve this network will be a future work for the deep learning analysis.

Table 5.5: Bidirectional and CNRNN results

Model	Word2Vec	Binary	5-granularity
BiGRU+ATT	**Glove**	**0.883147**	**0.488733**
BiLSTM+ATT	Glove	0.872597	0.471041
GRU+CNN+ATT	Glove	0.822076	0.453394
LSTM+CNN+ATT	Glove	0.842394	0.453846

5.2 Emotion Analysis Using Deep Learning Techniques

This project will used two datasets based on the discrete models.

- The "Emotion Intensity in Tweets" dataset from the WASSA 2017 shared task[23]. The dataset was created by extracting tweets following the next rule, for each emotion X, 50 to 100 terms have been selected that were associated with that emotion at different intensity levels.
- The ISEAR dataset was created by student respondents where they were asked to report situations in which they had experienced all of 5 major emotions (joy, fear, anger, sadness, disgust).

The procedure followed in the development of this analysis have been similar to the Sentiment Analysis part. The results will be shown in the table 5.6 with a little explanation about them.

5.3 Conclusion

Table 5.6: Results with all the datasets

Model	SST	SST-5	IMDB	ISEAR	EmoIntensity
GRU+ATT	0.873147	0.473303	0.92128	0.7536	0.834182
BiGRU+ATT	**0.883147**	**0.488733**	0.9436	0.788384	**0.857416**
CNN+BiGRU+ATT	0.822076	0.453394	0.88936	**0.870895**	0.736155
LSTM+ATT	0.870597	0.450679	0.89608	0.794693	0.844048
BiLSTM+ATT	0.872597	0.471041	**0.94688**	0.8188	0.857097
CNN+BiLSTM+ATT	0.842394	0.453846	0.88232	0.860864	0.729153

Table 5.6 describes the results achieved by the most performant models. As a conclusion of the Sentiment Analysis applying deep learning techniques, GRUs variations have manifested a better performance than LSTM variations, obtaning the highest results with the SST, 49% in 5-labels and 88,78% . However, in the IMDB dataset, the LSTM variations have been imposed on the rest of the models obtaining an accuracy of 95%.

On the other hand, the Emotion Analysis has similar results. In the EmoIntesity dataset, the most successfully model is the BiGRU with 86% of accuracy over four labels. However, the ISEAR datasets seems to work better with the CNN+BiGRU combination achieving a 87% over five labels.

GRUs have demonstrated to train faster, and they perform better when there is less training data. On the other hand, LSTM remember longer sequences and as it can be seen they have perform better in big datasets, such as the IMDB.

Although, the dataset is one the problem when this kind of models are being developed. Basically, all the neural nets are constructed to model the underlying distribution of the dataset they are being trained on. The dataset by itself, it is only a partial distribution of all possible case. The ideal case would be to have a large set of training data so that the model can learn and discover possible hidden patterns within these data and thus be able to make a better prediction. This is not possible in the vast majority of cases as it can see but even so, models such as CNN + RNN networks could have made a better adaptation and achieve better results if they had had enough data to adjust their parameters

It is important to note that the vast majority of models have a huge number of pa-

rameters to update during the training process and as complexity is added to the model this number grows exponentially. The aim is to work with a large enough training dataset without going into overfitting during the training process.

On the other hand, neural networks start from an infinite number of hyperparameters to tune and thus adapt the models in the best possible way to the datasets we have. Even so, this process can become quite cost, in terms of processing time, since generating all possible variants for each of the models can become an endless task.

In short, neural networks have shown good results in sentiment analysis, specifically recurrent neural networks. The data used and the application domain are crucial for the development of the models and therefore, it is necessary to always keep in mind both in order to develop models adapted to them.

Finally, the models chosen for developing it as a service has been the Bidirectional GRU for case Sentiment and the Bidirectional Conv1d in Emotion Analysis, due to its good performance. They will be trained with the SST 5-granularity dataset and the ISEAR dataset correspondingly.

CHAPTER 6

Machine Learning as Web Services

This section will describe the transformation of the classifiers in the previous chapter (Chapter 5) into a functional web service. The goal is to use Senpy to publish this classifiers. Senpy automatically handles several tasks (e.g.with the user) and provides a semantic API on top of classifiers and their results. In order to publish a service with Senpy, each classifier has to be converted into a Senpy Plugin. During the development of this project, a new paradigm for creating the Senpy plugins was introduced. As a result, the classifiers have been adapted to match these changes.

In summary, this section will provide an overview of the development of new Senpy plugins based on existing classifiers.

6.1 Introduction

The aim of this section is to develop the analysis services in the Senpy context. As was mentioned during the development of the project a new paradigm has appear in order to make easier the deploy of a classifier as a plugin. This paradigm is described in appendix A. Figure 6.1 shows the part of the project involved in this section.

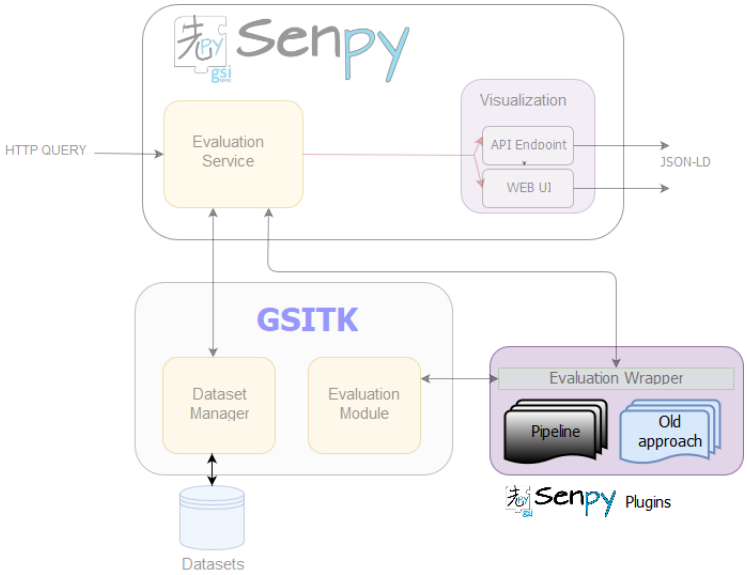

Figure 6.1: General architecture - Plugin development

6.2 Developtment of the plugins

The sections 3.1.2 and 3.1.3 defined the two main technologies used in this section, apart from Senpy.

The Keras library provides a convenient wrapper for deep learning models to be used as classification or regression estimators in scikit-learn. This capability has been used for building the classifiers from the models proposed in the section 5.3.

The section 2.3 explained the transformation done to the plain text for using them as an input of the neural networks classifiers. In order to follow the sklearn context, this transformation has been implemente as a transformer estimator.

Once we have all the necessary parts as sklearn components, they are used for developing a pipeline. The aim is to use this pipeline as the main component of the plugin.

Figure 6.2 shows the previous process as wrappers. The first term is the keras model which is the base of all the plugins. Secondly, we got the sklearn which acts as a interpreter of the keras model, putting a classifier wrapper (based on the sklearn API) above the model.

Also this wrapper is the new implementation of developing plugins. Finally, it is the Senpy wrapper, which publishes the classifiers as plugins. In this wrapper coexists the new plugins with the old ones.

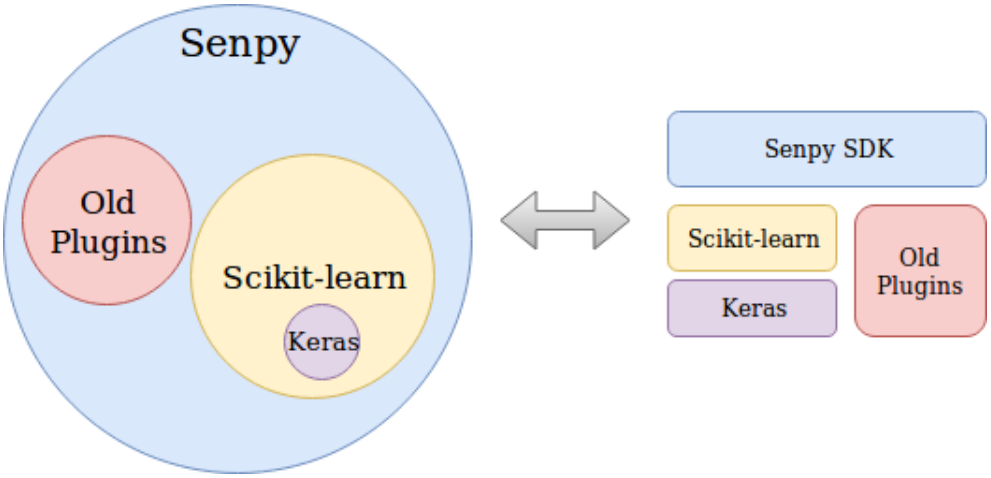

Figure 6.2: Wrappers involved in the development of the plugins

We have developed two plugins:

- Sentiment analysis plugin based on a BidirectionalGRU using the SST dataset. It analyses text and produces a sentiment polarity. The possible polarity values are between $[-1, 1]$.
- Emotion analysis plugin based on a CNN+BidirectionalGRU using the ISEAR dataset. It analyses text and produces five emotion labels, where their values are their probability distribution in the text. The five emotions are: sadness, joy, anger, disgust, and fear.

Both plugins have been cached, in order to reduce its deployment once they have been trained. The code of both plugins can be found in the repository *https://github.com/gsi-upm/senpy-plugins-community*.

CHAPTER 7

Developing Evaluation Services

At this point, Senpy 3.2 has been presented as a powerful tool for creating natural language processing analysis services. The previous section has described the functionality of a plugin and how they have been modified for simplifying the development, the idea is that a developer does not need to understand the basic structure of a plugin while he has a pipeline or something that behaves in the same way (following the sklearn API). Allowing a simple design for developing the plugin and abstracting the developer from the base code.

This section will discuss the development of the evaluation of services, which will allow to evaluate the plugins with different datasets and present the results of this evaluation in a linked data format.

7.1 Architecture

A key aspect of developing a new analysis algorithm is to evaluate it and compare it to others.The contribution of this part of project is going to be focused on the **Evaluation Layer**. The main idea is to implement this evaluation service for evaluate a set of plugins, one or more, with a set of datasets, one or more. Once the results are retrieved,they are going to be published using JSON-LD format.

The evaluation process will be taken to the field by gsitk library 3.1.4. The parameters necessary to carry out an evaluation are at least one pipeline or a similar object and one or several datasets.

First of all, let's see how this new component fits into the overall architecture. As it has been seen previously, Senpy is based on two pillars: the Core and the Plugins. The general idea is that this new evaluation service is present in both. On the one hand, the core itself will be the one that handles the requests to this new service, similar to how it handles them in the analysis part, passing the information of the datasets to evaluate each plugin.

On the other hand, it is in the plugin itself that the call to gsitk is made to perform the evaluation of the plugin with the corresponding datasets. As explained in the previous section, for practical purposes, the plugin would act as a pipeline by itself allowing it to be used by gsitk to carry out the evaluation.

This allows the evaluation process to be carried out both from the core and from the plugin itself. The life cycle of the complete process, from the core to the plugins, will be the following one:

1. The Core will process the requests and seperate the plugins from the datasets. It will load the datasets, that the users have provided in the requests, available at GSITK.

2. Once the datasets are loaded, the Core calls each one of the plugins presented in the requests with all the datasets.

3. The Plugin uses the evaluation module from GSITK, and collect the results. This results will be translated into a JSON-LD format, which will be explained below.

4. The Core collects all the results and presents it at a whole.

The figure 7.1 shows the process described above. The datasets that can be used in the complete process are all the ones available at the Dataset Manager from GSITK. Although, in case that it is wanted to use a external datasets, it can be passed to the plugin directly just calling the evaluation method with these datasets. The datasets has to be in pandas format.

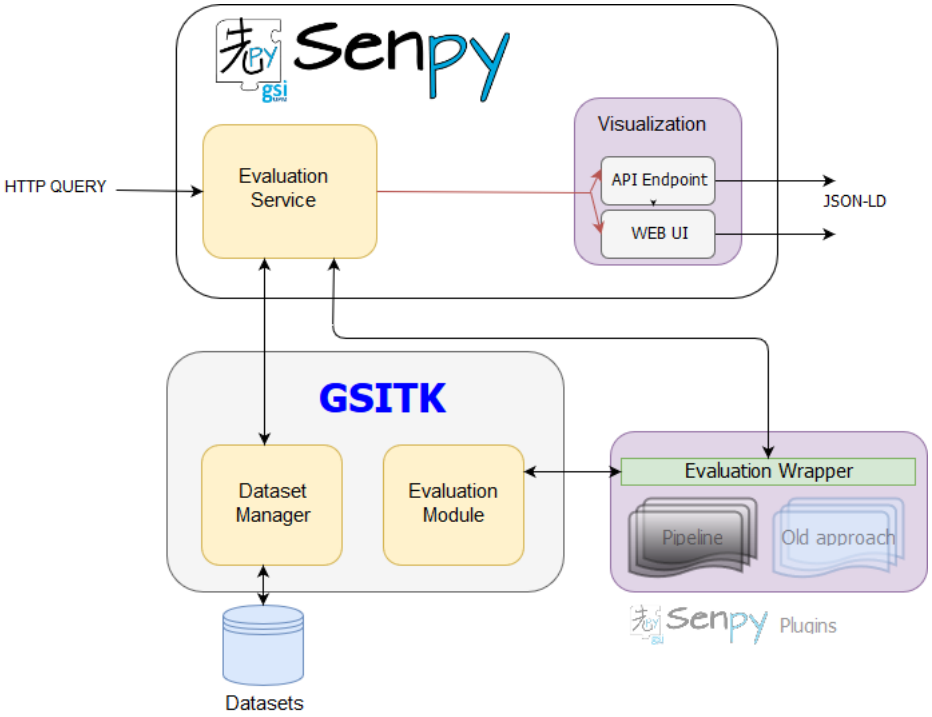

Figure 7.1: General Architecture - Evaluation Process

Evaluation services add a new wrapper above the ones described in Figure 6.2. Figure 7.2 shows the position of the evaluation wrapper in previous figure.

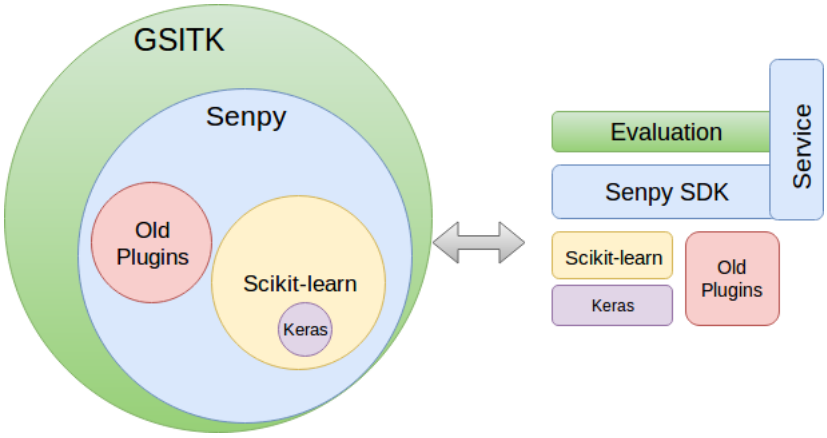

Figure 7.2: Evaluation Wrapper inside the plugin context

As described in the section 3.3, JSON-LD is a good implementation for translating JSON results to a linked data format. Senpy uses a linked data format for publishing the results from the analysis service.

Following this line, the evaluation services is going to publish its services using this structure. Although, the ontology is not published yet, it has already being drafted by the

GSI of the ETSIT-UPM. This project has contributed to translated the Evaluation ontology into the different schemes needed by JSON-LD. The draft of this ontology could be found in appendix B.

7.1.1 API and Playground

Basically, for running an evaluation is needed one or more plugins and one or more datasets. The idea for developing the evaluation API is the same as the one behind the general API from Senpy. They share the same parameters except one the datasets that have been added to the evaluation API.

Following the structure of the Senpy API, all the available datasets have been published as a JSON-LD format under the "api/datasets/". These datasets are all the possible provided by gsitk. Each one of them contains specific information about it. Evaluation service API is under "api/evaluate/" and as it has been talking above, it accepts the same parameters as the normal API and other one under the name datasests. Table 7.1 shows an overivew of the API endpoint and its parameters.

Table 7.1: Evaluation service API

URL	Information	Paremeters
/api/datasets	Returns the information of all the datasets	None
/api/evaluate	Basic endpoint for evaluation service	- algorithms, plugins/plugin to used in the evaluation service - datasets, dataset/datasets to be used in the evaluation service

This is an example response of a query made to the evaluation service.

```
GET http://0.0.0.0:5000/api/evaluate/?algo=sentiment-vader&dataset=vader&
    aggregate=true

{
  "@context": "http://0.0.0.0:5000/api/contexts/AggregatedEvaluation.jsonld
      ",
  "@id": "_:AggregatedEvaluation_1515955865.6111374",
  "@type": "aggregatedEvaluation",
  "evaluations": {
    "@id": "_:Evaluation_1515955866.811627",
    "@type": [
      "StaticCV",
      "Evaluation"
    ],
    "evaluates": "sentiment-vader__vader",
    "evaluatesOn": "vader",
    "metrics": [
      {
        "@id": "Metric0",
        "@type": "Accuracy",
        "value": 0.9126190476190477
      },
      {
        "@id": "Metric1",
        "@type": "Precision_macro",
        "value": 0.6453070416307266
      },
      {
        "@id": "Metric2",
        "@type": "Recall_macro",
        "value": 0.602816474582442
      },
      {
        "@id": "Metric3",
        "@type": "F1_macro",
        "value": 0.6232948736286149
      }
    ]
  }
}
```

Also the Playground provided by Senpy has been updated for allow this new API through a user-friendly interface. This new capability allows the user to evaluate the plugins with several datasets.

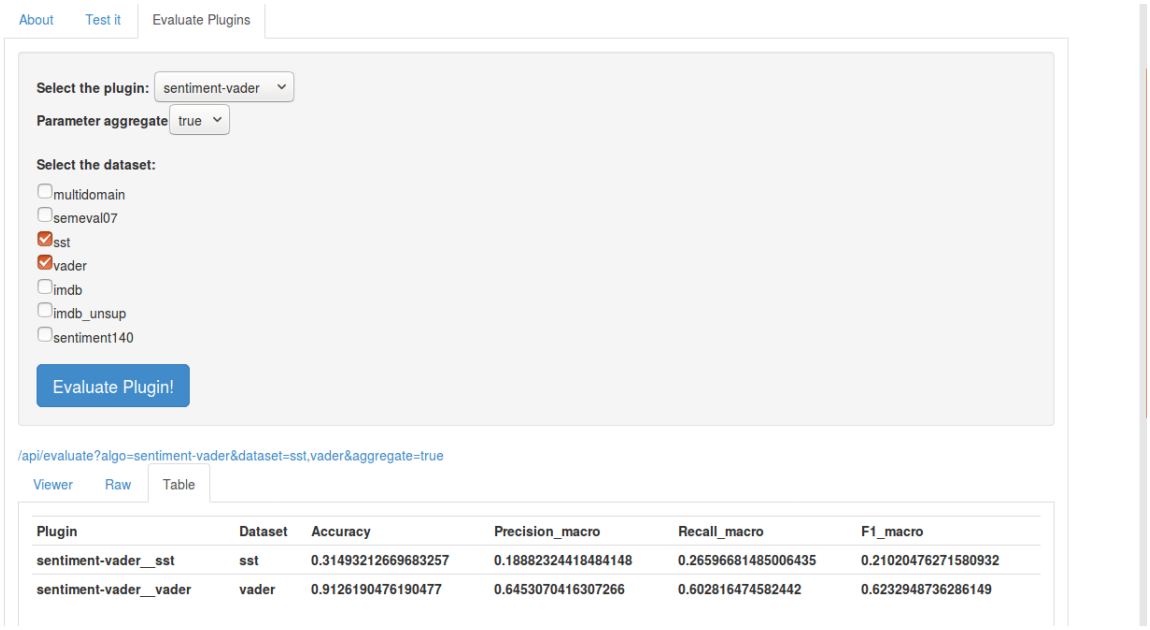

Figure 7.3: Evaluation Playground - Results from the evaluation, table format

Figure 7.4: Evaluation Playground - Results from the evaluation

CHAPTER 8

Conclusions and future work

This chapter will describe the achieved goals done by the master thesis following some the key points developed in the project.

8.1 Achieved Goals

The achieved goals for this project are the following ones:

- We have built two classifiers: one for sentiments and another one for emotions as a result of all the techniques applied. The code of the experiments carried out is public in github under the following url: *https://github.com/NachoCP/TFM-NeuralNetworks*. The repository has several ipython notebooks with the experiments and some python files with the code necessary to build the models.
- We have published two Senpy plugins based on the previous classifiers. This plugins can be found in the open-source repository of Senpy plugins: *https://github.com/gsi-upm/senpy-plugins-community*.
- We have developed an evaluation service, extending the services inside Senpy. This evaluation service will be included inside the Senpy base code.

8.2 Conclusion

The project has fulfilled the proposed objectives, but we must refer to each one of these objectives in to draw a general conclusion.

First of all, developing a classifier using Neural Networks has proven to be a good approach, obtaining fantastic results in sentiment and emotion analysis. Moreover, GRU and LSTM networks have been studied during the development of this project, allowing to extrapolate their differences. Among the main differences, the most prominent is the absence of internal memory by the GRUs, which allow them to act better for small datasets, compared to the LSTMs that works better with larger datasets. This absence of memory also affects the training process, allowing GRU networks to train more quickly.

Additionally, it has been discussed the problematic of the huge number of internal parameters that this kind of networks have, which slow the training process (more parameters, more time required) and the overfitting that they can suffer while training.

On the other hand, these classifiers have been implemented as plugins inside the Senpy framework, publishing them as a service inside a Linked Data context. The new paradigm for developing plugins has been proved to be an easy way to translate from a machine learning technique to a service.

Finally, it has been implemented an evaluation service inside Senpy, providing a way of performing interactive evaluation of the plugins. Due to the amount of datasets and algorithms, having a service and a formal representation of the evaluation carried out by them proves to very useful for comparing the algorithm behaviour with other datasets than the one with which it has been trained. Moreover, it allows to compares different algorithms with the same datasets in order to observe a concrete results or choosing the best algorithm.

APPENDIX A

New Plugin Paradigm

The new paradigm consists in redesign the plugin concept in order to keep it simple and easy to translate from an algorithm or a classifier, isolating the developer from the base code. In order to achieve this, the black box concept has been introduced.

A **Black Box**[1] is a device, system or object which can be viewed in terms of its inputs and outputs (or transfer characteristics), without any knowledge of its internal workings. Its implementation is "opaque" (black). Almost anything might be referred to as a black box: a transistor, an algorithm, or the human brain. However, how can it be translated into our specific domain?. Let's look the image below.

Figure A.1: Black Box basic diagram

[1]Wikipedia Black Box definition: https://en.wikipedia.org/wiki/Black$_b ox$

The Black Box Plugins will delegate the analysis to several methods. The main methods of this Black Box will be: **input**, **predict** and **output**. The workflow will be as follow: the **input** transform the query (entry+params) into data that can feed the **predict** method; the **predict** will use a source (algorithm, external API, rules, etc) for making an analysis; and the **output** will transform the results given by the previous method into an entry that Senpy can handle. This new structure maintains maintains the homogeneity of the entry (enriches the entry with the results of the analysis), allowing it to be used by the entry of another plugin. However, how does this new structure affect a developer?

Once the developer has finished designing its algorithm, either as a pipeline, a classifier or a set of rules applied to an input data, the idea is that he only has to complete the previous methods in order to fit its model into a Black Box structure. In other words, specify if its algorithm's input need some preprocess for feeding it and maps the output to the correct JSON-LD format. Senpy will provide a wrapper layer that will make the connections between these methods and deploy them as a service. Moreover, as it follow the scikit-learn api, this Black Box can be used as a estimator from other Pipeline. The diagram A.2 illustrates the life cycle of this process.

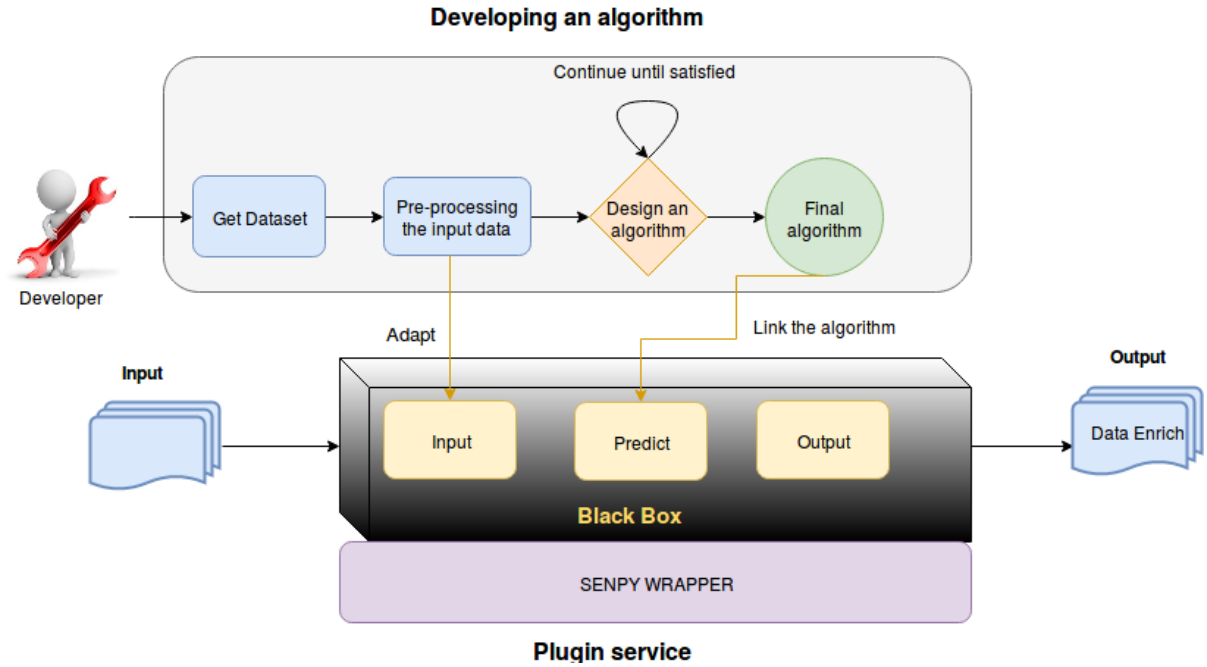

Figure A.2: Life cycle from developing an algorithm to implement the BlackBox

APPENDIX B

Evaluation Ontology

The draft B.1 shows an overview about the Evaluaton ontology.

The main classes of the Evaluation Ontology are: Aggregated Evaluation, Evaluation and Metric. agglutinates the whole set of evaluations, Evaluation is the basic class that defines the concepts and metric is the statistics. These classes are linked by the next way: an AggregatedEvaluation is run on a source, the results are represented as one or more Evaluation instances that contain one or more Metric instances.

The Evaluation provides the general information related to the process, such as the algorithm and the datasets that has been used for that specific evaluation. This two values are under the properties "evaluates" for the algorithm and "evaluatesOn". Moreover, each Evaluation has a set of Metrics under the property "hasMetrics". Each one of the Metrics, provide information about its type (e.g. "accuracy"), its value and the deviation of this value.

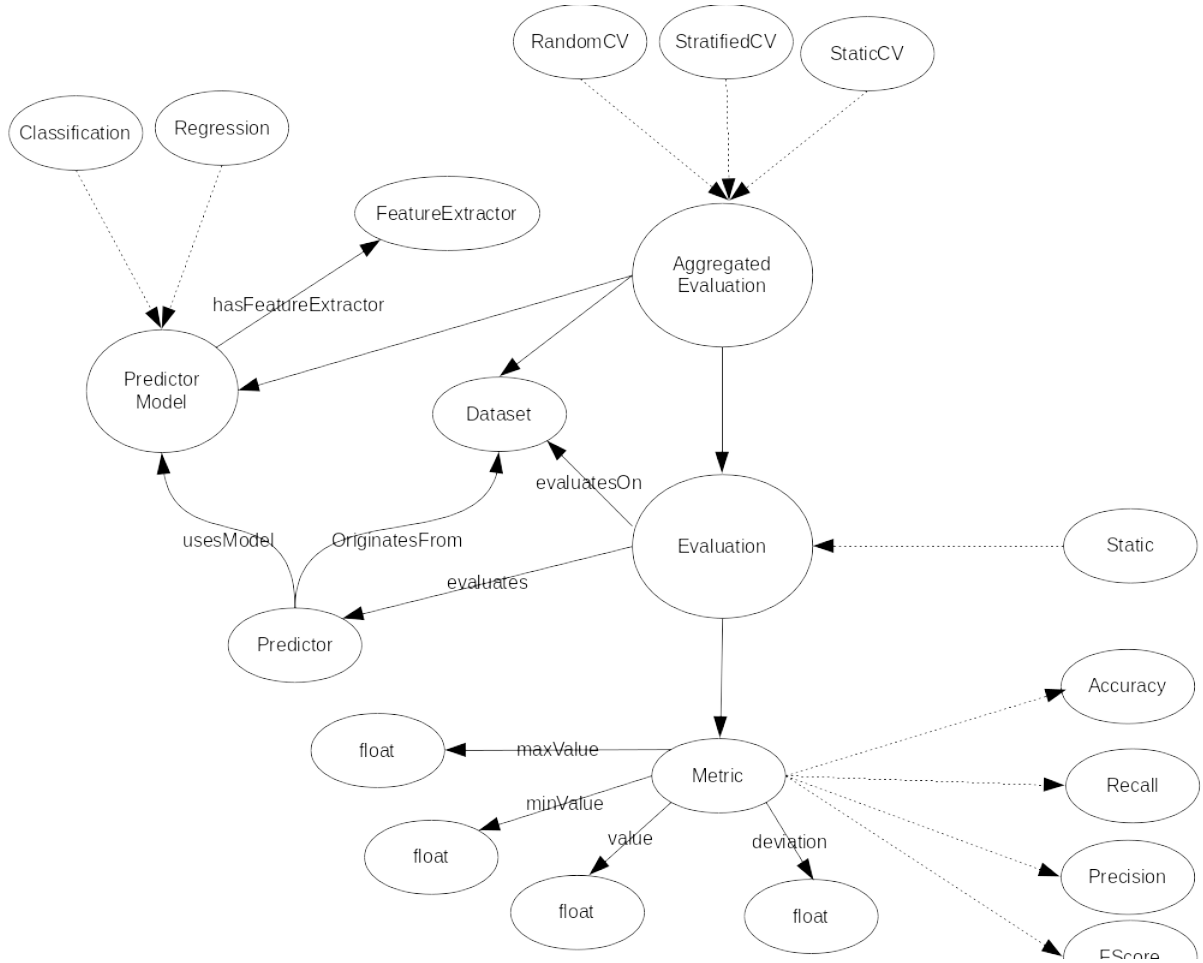

Figure B.1: Evaluation Ontology draft

Bibliography

[1] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

[2] Joan C Borod. *The neuropsychology of emotion*. Oxford University Press, 2000.

[3] Pimwadee Chaovalit and Lina Zhou. Movie review mining: A comparison between supervised and unsupervised classification approaches. In *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*, pages 112c–112c. IEEE, 2005.

[4] Jan Chorowski, Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. End-to-end continuous speech recognition using attention-based recurrent nn: first results. *arXiv preprint arXiv:1412.1602*, 2014.

[5] Ze-Jing Chuang and Chung-Hsien Wu. Multi-modal emotion recognition from speech and text. *Computational Linguistics and Chinese Language Processing*, 9(2):45–62, 2004.

[6] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[7] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

[8] Paul Ekman. An argument for basic emotions. *Cognition & emotion*, 6(3-4):169–200, 1992.

[9] Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

[10] Simon S Haykin, Simon S Haykin, Simon S Haykin, and Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, NJ, USA:, 2009.

[11] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

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

[13] CA Iglesias, JF Sánchez-Rada, G Vulcu, and P Buitelaar. Linked data models for sentiment and emotion analysis in social networks. In *Sentiment Analysis in Social Networks*, pages 49–69. Elsevier, 2017.

[14] William James. *The principles of psychology*. Read Books Ltd, 2013.

[15] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. *Machine learning: ECML-98*, pages 137–142, 1998.

[16] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[17] Filippos Kokkinos and Alexandros Potamianos. Structural attention neural networks for improved sentiment analysis. *arXiv preprint arXiv:1701.01811*, 2017.

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

[19] Yunfei Long, Lu Qin, Rong Xiang, Minglei Li, and Chu-Ren Huang. A cognition based attention model for sentiment analysis. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 473–482, 2017.

[20] Danilo P Mandic, Jonathon A Chambers, et al. *Recurrent neural networks for prediction: learning algorithms, architectures and stability*. Wiley Online Library, 2001.

[21] Albert Mehrabian. Pleasure-arousal-dominance: A general framework for describing and measuring individual differences in temperament. *Current Psychology*, 14(4):261–292, 1996.

[22] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.

[23] Saif M Mohammad and Felipe Bravo-Marquez. Emotion intensities in tweets, 2017.

[24] Amr El-Desoky Mousa and Björn Schuller. Contextual bidirectional long short-term memory recurrent neural network language models: A generative approach to sentiment analysis. In *Proceedings EACL*, 2017.

[25] Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 115–124. Association for Computational Linguistics, 2005.

[26] Bo Pang, Lillian Lee, et al. Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval*, 2(1–2):1–135, 2008.

[27] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics, 2002.

[28] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[29] Rosalind W Picard and Roalind Picard. *Affective computing*, volume 252. MIT press Cambridge, 1997.

[30] Robert Plutchik. A general psychoevolutionary theory of emotion. *Theories of emotion*, 1(3-31):4, 1980.

[31] Jonathan Posner, James A Russell, and Bradley S Peterson. The circumplex model of affect: An integrative approach to affective neuroscience, cognitive development, and psychopathology. *Development and psychopathology*, 17(3):715–734, 2005.

[32] David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.

[33] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA. `http://is.muni.cz/publication/884893/en`.

[34] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[35] Stuart Russell, Peter Norvig, and Artificial Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:27, 1995.

[36] J. Fernando Sánchez-Rada and Carlos A. Iglesias. Onyx: A Linked Data Approach to Emotion Representation. *Information Processing & Management*, 52:99–114, January 2016.

[37] J. Fernando Sánchez-Rada, Carlos A. Iglesias, Ignacio Corcuera-Platas, and Oscar Araque. Senpy: A Pragmatic Linked Sentiment Analysis Framework. In *Proceedings DSAA 2016 Special Track on Emotion and Sentiment in Intelligent Systems and Big Social Data Analysis (SentISData)*, pages 735–742, October 2016.

[38] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[39] Marc Schröder, Hannes Pirker, and Myriam Lamolle. First suggestions for an emotion annotation and representation language. In *Proceedings of LREC*, volume 6, pages 88–92, 2006.

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

[41] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[42] Maite Taboada, Julian Brooke, and Manfred Stede. Genre-based paragraph classification for sentiment analysis. In *Proceedings of the SIGDIAL 2009 Conference: The 10th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 62–70. Association for Computational Linguistics, 2009.

[43] Maite Taboada, Julian Brooke, Milan Tofiloski, Kimberly Voll, and Manfred Stede. Lexicon-based methods for sentiment analysis. *Computational linguistics*, 37(2):267–307, 2011.

[44] Peter D Turney. Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 417–424. Association for Computational Linguistics, 2002.

[45] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[46] Xingyou Wang, Weijie Jiang, and Zhiyong Luo. Combination of convolutional and recurrent neural network for sentiment analysis of short texts. In *COLING*, pages 2428–2437, 2016.

[47] Yequan Wang, Minlie Huang, Xiaoyan Zhu, and Li Zhao. Attention-based lstm for aspect-level sentiment classification. In *EMNLP*, pages 606–615, 2016.

[48] Chung-Hsien Wu, Ze-Jing Chuang, and Yu-Chung Lin. Emotion recognition from text using semantic labels and separable mixture models. *ACM transactions on Asian language information processing (TALIP)*, 5(2):165–183, 2006.

[49] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[50] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pages 2048–2057, 2015.

[51] Chew-Yean Yam. Emotion detection and recognition from text using deep learning. 2015.

[52] Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. Attention-based bidirectional long short-term memory networks for relation classification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 207–212, 2016.