MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE SAAD DAHLAB - BLIDA 1

FACULTE DES SCIENCES

DEPARTEMENT D'INFORMATIQUE

Master's Report

in Computer Sciences

Specialty : Natural Language Processing

Automatic Identification of an Author Based on Writing Style

Presented by:

Tabet Abderraouf and Bouhala Ayoub

Supervided by :

Promoteur Mr Cherif-Zahar

2021-2022

Appreciation

At the end of this work, we thank God in the first place for giving us the strength and courage to bring it to completion.

We would like to express our sincere thanks to our Promoter m.chrife zahar for his unquestionable advice and interests.

Our thanks go to the people who helped us in the realization of this brief.

We extend our sincere thanks to the members of the jury for the interest they expressed in this modest work and agreed to review it.

In the impossibility of mentioning all the names, our sincere thanks go to all those who contributed by their advice to the good outcome of this work.

Finally, we would not dare to forget to thank all the TAL teachers for the enormous work they do in order to create the right conditions for our studies.

Dedication

I dedicate this work to my dear parents

I dedicate this work to my dear family

I dedicate this work to my dear friends

I dedicate this work to my brothers Tb30,Tb10

I dedicate this work to my dear sport athletes

I wish you all happiness

Abstract

It often happens that articles that appear in the press or sometimes entire books are not signed and we know nothing about their author. It also happens that authors attribute to themselves the authorship of an unsigned article or book or, on the contrary, that critics question this authorship.

The idea of the present subject and of being able to attest whether, according to the style of this or that other author, an unsigned article could be from him or not.

Our interest in the subject applies to journalistic articles in Arabic with nationalist or even revolutionary connotations from the pre-revolutionary Algerian period that appeared in Ech-Chihâb and/or El-Bassā'ir.

Unfortunately we didn't manage to found a proper Arabic data concern our topic , therefore we used a English one.

The Ech-Chihâb of IbnBadîs appeared from 1925 to 1939 first in weekly and then monthly form.

El-Bassa ’ir was the organ of the Association of Algerian Muslim Ulemas from 1935 to 1939 and then slipped voluntarily during the Second World War to reappear only in 1947 to be suspended by the colonial administration in 1946.

Both newspapers contained religious, social, and biographical articles with moralistic aims, but also contained sections of national or foreign policy, which are of particular interest to us.

Résumé

Il arrive souvent que des articles parus dans la presse ou parfois des livres entiers ne soient pas signés et on ne sait rien de leur auteur. Il arrive aussi que les auteurs s'attribuent la paternité d'un article ou d'un livre non signé ou, au contraire, que les critiques remettent en question cette paternité.

L'idée du sujet présent et de pouvoir attester si, selon le style de tel ou tel autre auteur, un article non signé pourrait être de lui ou non.

Notre intérêt pour le sujet s'applique aux articles journalistiques en arabe à connotation nationaliste voire révolutionnaire de la période algérienne pré-révolutionnaire parus dans Ech-Chihâb et/ou El-Bassâ'ir.

Malheureusement, nous n'avons pas réussi à trouver une bonne donnée arabe concernant notre sujet , donc nous avons utilisé un anglais.

Le Ech-Chihâb d'Ibn Badîs est apparu de 1925 à 1939 d'abord sous forme hebdomadaire puis mensuelle.

El-Bassa 'ir a été l'organe de l'Association des oulémas musulmans algériens de 1935 à 1939 et a ensuite glissé volontairement pendant la Seconde Guerre mondiale pour réapparaître seulement en 1947 pour être suspendu par l'administration coloniale en 1946.

Les deux journaux contenaient des articles religieux, sociaux et biographiques à but moraliste, mais aussi des sections de politique nationale ou étrangère, qui nous intéressent particulièrement.

الملخص

غالبًا مايحدث أن المقالات المنشورة في الصحافة أوفي بعض الأحيان لا يتم توقيع الكتب الكاملة ولا يُعرف أي شيء عن مؤلفها. يحدث أيضًا أن يدعي المؤلفون أنهم مؤلفون لمقال أو كتاب غير موقع أو،على العكس من ذلك، يشكك النقاد في هذا التأليف..

فكرة الموضوع الحالي والقدرة على إثبات ما إذا كان،وفقًا لأسلوب هذا المؤلف أو ذاك،مقالة غير موقعة يمكن أن تكون له أملا..

اهتمامنا بالموضوع ينطبق على المقالات الصحفية باللغة العربية ذات الدلالات القومية أو حتى الثورية من فترة ماقبل الثورة الجزائرية والمنشورة في الصياد و / أو البصائر..

لسوء الحظ، لم نتمكن من العثورعلى بيانات عربية جيدة لموضوعنا،لذلك استخدمنا واحدة باللغة الإنجليزية..

ظهر ابن باديس الشهاب من عام 1925 إلى عام 1939 في شكل أسبوعي ثم شهريًا..

كانت البساط عير عضوًا في جمعية العلماء المسلمين الجزائريين من عام 1935 إلىعام1939 ،ثم انزلق تطواعية خلال الحرب العالمية الثانية لتظهر مرة أخرى في عام 1947 لتعلق من قبل الإدارة الاستعمارية في عام.1946.

تحتوي كلتا الصحيفتين على مقالات دينية واجتماعية وسيرة ذاتية ذات غرض أخلاقي،ولكن أيضًا تحتوي على أقسام من السياسة الوطنية أو الخارجية،والتي تهمنا بشكل خاص..

Table of contents

List of Figures

Abreviations List

RNN Recurrent Neural Network

LSTM Long Short-Term Memory Neural

GRU Gated Recurrent Unit

TAL Traitement automatique du langage

EAP Edgar Allan Poe

HPL HP Lovecraft

MWS Mary Wollstonecraft Shelley

TALN Le traitement automatique du langage naturel

MIT Massachusetts Institute of Technology

BERT Bidirectional Encoder Representations from Transformers

TPU Tensor Processing Unit

GPU Graphics Processing Unit

List of Tables

Chapter 1 : Introduction

1.1 Work context

If most authors are known, some had to use borrowed names or simply did not sign their articles for fear that the presence of their name in plain language would be material, for the colonial administration, to suspend these two newspapers which were the spearhead of the Muslim reform movement and thereby to destroy the movement which is one of the major factors in the outbreak of the Revolution of November 1, 1954.

1.2 Problematic

In the world of writing production (books, press articles, studies, etc.), it can happen that unscrupulous authors plagiarize the work of other authors. Thus the 18th century sees the advent of the individual, claiming for himself the property of his work. Moreover, the word plagiarism was born in 1697 in the Dictionnaire de Pierre Bayle, while the verb "plagier" would be hatched in 1801 under the pen of the picturesque and fertile Louis Sébastien Mercier who was so plagiarized."[1]

So what is plagiarism?

According to the Robert, plagiarism is “copying an author by wrongly attributing parts of his work”.

A more explicit definition or at least more explanation found on the University of Namur website says that it is the fact of “copying someone’s work or part of it and claiming authorship”.

If copying “part of the work” is a matter of plagiarism, how do you do it with quotations?

Are they plagiarism or are they allowed and, if so, how high?

It would seem that It is only with the printing press that the indicators of quotation are introduced into the text, it is only from this invention that quotation acquires its proper meaning, modern, full, and that it defines a specific category in the practice of the text.”[2]

Therefore, there is no harm in repeating sentences from another author while respecting certain rules, including framing the quotation by quotation marks or placing it in another color or another attribute of the characters used and returning it to the reference from which the quotation.

Logic would like the quotation not to be too long because otherwise it would risk becoming plagiarism.

However, it often happens that a work «pumps» literally whole sandwiches and, therefore, it is no longer an original work but a disguised plagiarism.

There is another problem with plagiarism, which is similar to plagiarism, and that is when we have anonymous writings, of which an author, years after publication, claims to be the author.

These types of problems are not common. However, they do exist and we give some examples below.

When Algeria was in the throes of colonialism, it often happened that talented publicists did not sign their articles or use pseudonyms to avoid problems with the administration whichMight have banned them from publishing anything. Thus we find articles in the Chiheb of Benbadis signed “al-fata azzouaoui”, “kâtibkabîr”, “al mansûr”… For connoisseurs, these names are not enigmatic.

The problem which is at the origin of the idea of subject and a considerable amount of political articles published in the Chiheb and which are not signed outright. Among these articles, some form a kind of cornerstone in the ideology of revolutionary and post-colonial Algeria. The latter show the fierce nationnalism of their author.

In this category there is a collection of three articles which originated from an article by Mr. Ferh²at Abbas who represented the leader of the assimilationists. In his article “France is me!” Ferhat Abbas wrote, “If I had discovered the “Algerian nation,” I would be a nationalist… And yet I will not die for the “Algerian homeland”, because that homeland does not exist. I have not discovered it. I questioned the history, I questioned the living and the dead; I visited the cemeteries: no one told me about it. No doubt I found “the Arab Empire”, the Muslim Empire, which honours Islam and our race. But those empires are extinct. They corresponded to the Latin Empire and the Holy Roman Empire of the medieval period. They were born for a time and a humanity that are no longer ours . . . We have once and for all discarded the clouds and the chimeras to definitively link our future with that of the French work in this country.»[3]

As soon as it appeared, an article entitled "A clear answer!" (كلمةصريحة) appeared in the Chihab where it says:

The Algerian nation is not France, it cannot become it, it does not want to become it, it could not become it even if it wanted to! It is a nation very far from France, in its language, in its uses, in its components, in its religion and it refuses to integrate. It has a clean country and that country is Algeria!”

"إن الأمة الجزائرية ليست هي فرنسا، ولا يمكن أن تكون فرنسا، ولا تريد أن تصير فرنسا، ولا تستطيع أن تصير فرنسا لو أرادت، بل هي أمة بعيدة عن فرنسا كل البعد...، في لغتها، وفي أخلاقها، وعنصرها، وفي دينها، لا تريد أن تندمج ولها وطن معين هو الوطن الجزائري"

Some authors such as Mohamed Salah Ramadhan explain that this quote has become the credo of Muslim nationalists for fifty years and that parents have been trying to teach their children by heart.

The value of this quotation, which concludes the answer to Ferhat Abbas and which led to his «repentance» since he will come to apologize to Benbadis, is very great. It appears in the section reserved in the newspaper for the column usually reserved for Ahmed Tewfik el Madani. The latter, from 1976, published his memoirs in three volumes on a total of about 1490 pages. In the second volume reserved for the militantism of the author from 1925 to 1954 in Algeria where he was one of the most influential members of the Muslim reformist movement, the latter explains how Ferhat Abbas published his article deemed insulting and how he wrote his reply entitled “kalimasariha”. The other tells us in his memoirs how

Benbadis went especially to Algiers to congratulate Tewfik el Madani for this scathing response against the followers of assimilationism and to discuss with him the consequences of

the publication of the article because the administration The colonial was not going to be silent.

As the article in question had appeared in Benbadis' newspaper, and that it was not signed as almost all the articles of Tewfik el Madani, it was only natural that people believe that Benbadis was the author. When Tewfik el Madani asserted that he was the author of it, an outcry took place by some students of Benbadis to refuse Tewfik el Madani his declaration and it resulted in a real war against him which will even lead to the production of a pamphlet.

The primary objective of this work was to repair this enormous injustice by using modern computer techniques to determine the degree of likelihood of the author being Benbadis or Tewfik el Madani.

The only major asset for this would be to start from the simple style of writing to try to detect who is the author.

It is therefore important to define an author's style.

1.3 The style of an author

It is not very easy to describe the style of an author. In the specific case of Mr. Ahmed Tewfik el Madani, Mr. Mohamed Salah Ramadan says that his style is "sahlmoumtanaa", which would translate into "inimitable although simple". In fact, Ahmed Tewfik el Madani does not find in almost all of his written production -which is huge compared to the other actors of Algerian Muslim reformism of the 1930s- outdated words, complex phrases, parables. The language is simple, direct and yet eminently elegant.

Other authors such as Mubarek el Mili and BachirIbrahimi use sadj" (the rhyme) a lot in their writings. They sometimes use unusual or even unknown words.

Here is an example of Bachir el Ibrahimi's pen:

« لو مات المنصف بالأغواط لطافت الجزائر بجثمانه عدة أشواط، ولذهبت فيه مذهب العرب في ذات أنواط، ولغسلته بالعبرات المسفوحة، وكفّنته بألفاف القلوب، ودفنته في مستقرّ العقيدة والواجب من نفوسه

ولو مات بتنس لتاهت فخرا على الثغور، وباهت بيوم موته أيامها في غابرات العصور... ولو مات بأية بقعة من أرض الجزائر لكانت هي تونس نضرة واخضرارا، ولاكتسبت الجزائر بجميع أقطارها شرفا ممّن مات ميتة الشرف فيها، ولقبست معاني من الفداء والتضحية بعد عهدها بمثلها، ولفغمتها نفحة ساطعة من عز الإمارة حرمتها الأنوف الشمّ من أبنائها منذ أيام عبد القادر، ولتسمعت نغمة ساحرة عطّلت آذانها منها من عهد عهيد..."

Also on the lexical level, we note that Ahmed Tewfik el Madani uses common terms in Tunisia, the country that saw him born. Thus we find extressions such as «ذهبلتوه" or خرجتوا". We see him using fairly clean expressions such as نرفععقيرتنا that we only find in him.

On a thematic level, Benbadis writes in the religious field. The two main sections of his writings concern tafsîr and hadith. He also writes biographies of companions of the prophet.

Tewfik el Madani specialized in foreign policy of which he was the master in Tunisia then in Algeria. He also wrote in national politics but also in history. His training at the Zitûna

in Tunis focused on history and he will write countless books and studies and give numerous lectures on historical themes relating to Algeria but also to the Muslim world.

It would have been quite certain that the work would have conclusively determined that the article kalimasariha is of him since it mentions historical themes which the historian juggles with.

Unfortunately, modern techniques such as elearning require a consistent dataset. However, although the production of Tewfik el Madani is enormous, it does not exist in the form of a usable text document, nor does this of Benbadis.

Also, we will remain hungry and try to validate the approach that we will explain on a dataset of English authors.

The work can then be tested as soon as an Arabic dataset is available.

1.4 Objectives

The objective of this work is therefore to study the existing methods for detecting plagiarism or identifying an author by his style.

An analysis of the results published by their authors on the use of this or that method will guide us on the direction to take.

1.5 Memory organization

Chapter I: Introduction

Chapter II: State of the art of related work

Chapter III: State of the art on classification

Chapter IV: Designing an Author Detection Application by Style

Chapter V: Implementation and Testing

Chapter II: State of the art of related work

Chapter II: State of the art of related work

2.1 Introduction

The automatic detection of an author or a writer who writes an article or a text in a journal that identifies itself not his personal information and sometimes it was important to know the writer is all based on the writing style, If the machine can detect one author on the other using deep learning, this type of detection is most effective in identifying author by another or a free writer who just writes a text in a journal by another known writer who relies on intrinsic detection that exploits data from within documents, The detection of authors by study of the writing style of the document is the most common form of intrinsic plagiarism detection , contains problems such as relevant text cutting and collection of stylistic data.

In this chapter we will demonstrate what this based for identification a style of an author, and we will illustrate the experiment on Giono also we will talk about some method used to help at the level of deep learning how to identify a style.

2.2 Pen measurements

2.2.1 definition

Stylometry is a field of linguistics that is based on statistics for stylistic properties of a pen text of an article is based on the analysis of linguistics , detection and automatic grouping of writing style in an article or text these are the pen variables,it is used to identify the style of an author it is possible to calculate, to make factor analyses, statistical forecasts, comparisons.

Stylometry is a branch of linguistics concerned with the quantitative description of the stylistic properties of texts. In some cases, it allows to solve problems of authorship of disputed texts and to discover the probable chronology of the works of a given author. A historical overview of pen measurements shows that there was not a single scientist whose work could be considered decisive in its development. At the same time, the literature on the history of pen work shows that authors treat available material selectively, preferring some researchers while completely ignoring others. WincentyLutosławski (1863-1954) is a good example of a scientist forgotten (or underestimated) by contemporary scholars. However, it was he who coined the term "stylometry" already at the end of the 19th century and defined the principles of this "new science". This article presents and discusses the following questions: the importance of chronology in the interpretation of Platonic philosophy, the definition and objectives of the pen, the most important Platonic chronologies, a description and assessment of Lutosławski’s contribution to the development of pen methodology, and the origins of pen measurement. Finally, we will attempt to (re)determine Lutosławski’s place in the history of language sciences [4].

2.2.2 historic

Since the 19th century, Mendenhall (1887) suggests that by analysing the internal characteristics of a text one can recognize the author. Since then, document pen techniques have made significant advances and research [5] has applied this finding to plagiarism detection. Some of this research focuses on extracting and monitoring the most relevant pen data. Stein and Eissen (2007) and Zamani et al. (2014) monitor the proportions of lexical class use within segments in order to discern which are the most relevant to the author's style. Oberreuter and Velásquez (2013) favour, on the other hand, the frequency of terms as a pen data to be monitored. At the same time, research on learning and classification (Layton et al., 2013) has emerged. They characterize an author's style by a n-gram language model, so they train their module on a corpus of learning authors and apply the resulting model on test runs in order to calculate the most likely author [6].

2.3 Experimentation on Giono

2.3.1 Introduction

What if artificial intelligence can identify a writer's style? And if, automatically, the machine was able to identify the characteristics of a writing.

It is exactly the experimentation that is attempted on Giono , conduct an experimentation on a very large corpus that brings together the novel works of Giono unpublished. two bases were thus constituted by E. Brunet: one brings together the Romanesque works of Giono, where we distinguish the two well-known ways of writing of the Gionian specialists since Hill (1929) on the one hand the pre-Romanesque novelswar, on the other chronicles after 1946 - from the King without entertainment. Critics tend to see even three ways in distinguishing in the Chronicles the Angelo Cycle inaugurated by Le Hussard[7]and grouping together the following works : Le Hussard, Angelo, Crazy Happiness, Half-Brigade Stories [8]. The other corpus that serves as reference counts 50 texts for 25 authors [9].

2.3.2 The prediction stage

2.3.3 Traditional methods

The results obtained with the intertextual distance calculations in Hyperbase applied to the Giono base only, which makes it possible to highlight the groupings of works according to their lexical or syntactic affinities. The texts of the study corpus are compared between them and two to two with as internal standard the whole corpus. The following factorial analysis

(Figure 1) proposes the calculation on the forms of the corpus.

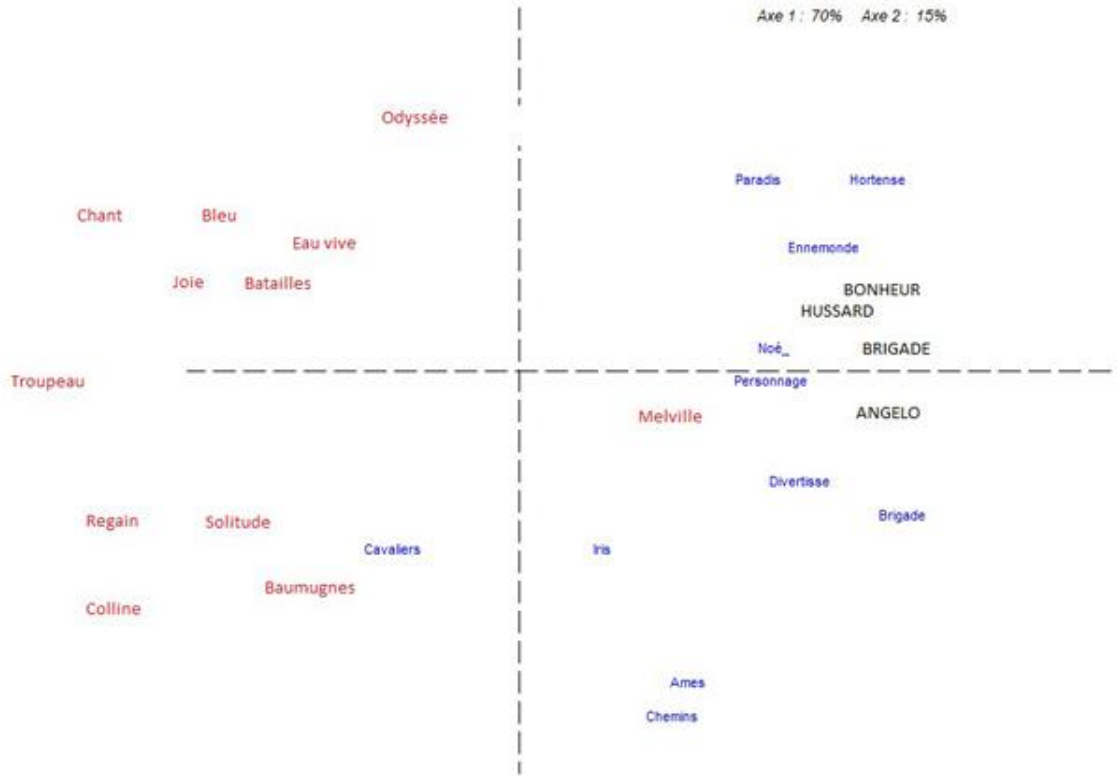

Figure 1: Factorial analysis on forms – GIONO database

We can see the expected distribution between the two ways of Giono (the first in the left quadrants of the graph, the second in the right quadrants). There is no defection of the Great Flock, even if this work corresponds to a war chronicle and, as such, has a hybrid status – belonging to the first chronologically but to the second, a priori, thematically.

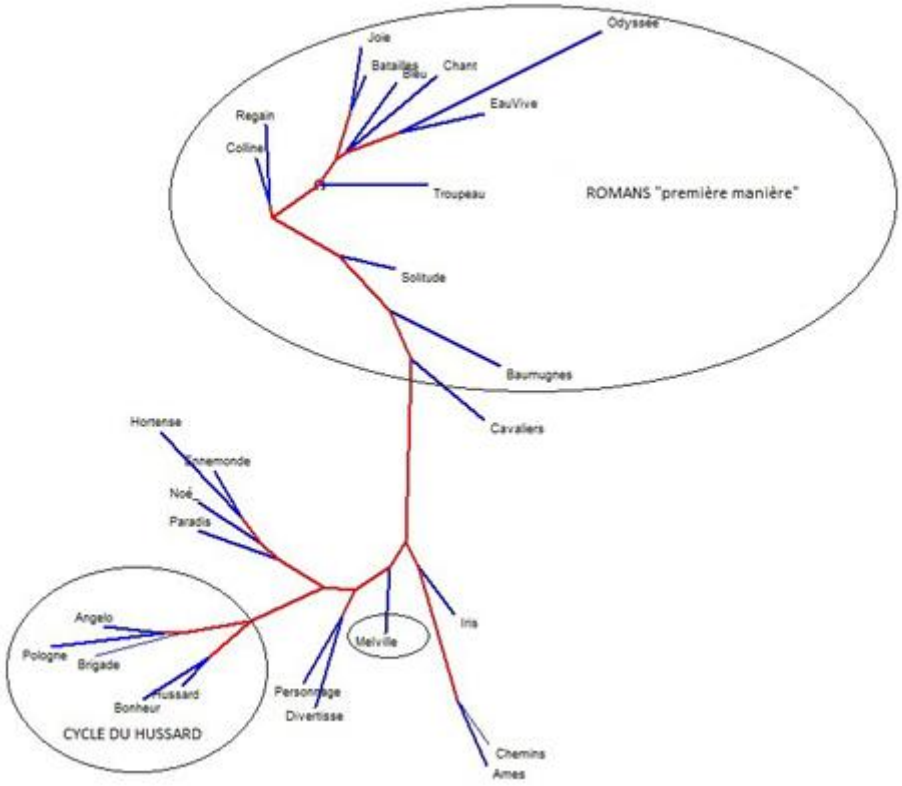

Figure 2: Tree analysis on lemmas – Giono base

The distribution is a little more disturbed when it concerns lemmas (Figure 2).The bipartition remains with a grouping in a bouquet of the Hussard cycle. However, the works seem to affirm their connections according to the lexical fields they develop. Regain and Hill are carried by a single branch for example, while The Great Flock takes some freedom from novels in the first way, like To greet Melville who finds himself isolated on a single branch.

2.3.4 Deep learning, from model to modelling

The work is done from the base of fifty novels; this one is trained with all these texts, in other words the engineer teaches the machine to recognize the works and their authors from the two texts given for each writer; for Giono, the machine has learned to identify The Great Flock and The Hussard on the roof as works written by Giono , It is then to present the other works of Giono and to observe the rate of recognition performed by the machine: this is the first so-called stage of prediction.

We send texts not included in the database and the algorithm states a degree of recognition evaluated in terms of percentages that could be glossed by a formula such as “this text is rather of…”. The algorithm proceeds through a window of three words, which means that the sequentiality is the only criterion and matches are not chosen based on syntactic criteria. Deep learning works in layers and every word is converted into numerical values. Frequency concerns not only forms but sequences, linear sequences of forms.words, which means that the sequentiality is the only criterion and matches are not chosen based on syntactic criteria. Deep learning works in layers and every word is converted into numerical

values. Frequency concerns not only forms but sequences, linear sequences of forms. Deep learning is sensitive to the different levels of granularity of texts – the grammatical value is, for example, independent of the form – and is distinguished from the traditional method by the combination of sequential and frequency, by the identification of complex and multidimensional motifs, by the intertextuality to be discovered. The results for our corpus of studies presented as percentages are gathered in the table below – in red the highest recognition rates.

	Ajar	Aragon	Breton	Camus	LeClézio	Colette	Duras	Emaux	Empéry	Gary1	Gary2	Gide	Giono	Graudoux	Grecq	Malraux	Mammeri	Mauriac	Montherlant	Pérec	Proust	Queneau	Tournier	Vian	Yourcenar	Total
Baumugnes	4.25	6.15	0	5.14	8.6	0.34	2.01	2.23	0.89	1.23	1.56	11.28	**28.94**	0.34	1.9	2.23	1.9	2.46	4.58	0.11	2.68	9.94	0.11	0.67	0.45	*100*
Regain	1.35	3.44	0	1.88	15.42	0.42	3.02	1.35	0.31	0.21	0.52	4.17	**50.94**	0.21	0.21	5.83	1.15	1.88	1.15	0.31	0.52	4.79	0.52	0.21	0.21	*100*
Odyssée	0.2	1.6	0.2	0.9	6.71	0.5	1.7	2.8	1.4	1	0.9	7.61	**8.71**	3.8	3.6	**12.21**	4.7	3.2	**12.71**	2.1	0.9	3.2	3	0.5	**15.82**	*100*
Solitude	1.78	9.63	0.12	3.92	8.2	0.48	1.31	1.9	0.95	1.78	2.73	8.8	**26.75**	0.95	1.9	5.11	2.62	2.14	4.28	0.83	0.83	11.18	0.59	0.59	0.59	*100*
Bleu	3.04	6.21	0.13	4.5	8.23	0.13	4.62	3.86	0.7	2.91	1.71	6.21	**22.17**	3.74	2.03	4.43	5.7	2.79	3.86	0.57	1.08	7.73	1.39	0.82	1.46	*100*
Chant	1.4	2.74	0	2.85	7.15	0	2.47	1.24	0.32	0.38	0.75	3.87	**39.76**	0.48	1.24	2.31	6.99	1.72	9.67	0.21	0.43	8.28	0.86	4.67	0.21	*100*
Joie	2.62	2.74	0.16	3.26	11.68	0	7.98	1.66	0.35	0.41	1.24	5.33	**31.38**	1.34	1.44	2.97	4.4	1.02	7.09	0.32	0.64	8.68	1.44	1.37	0.48	*100*
Batailles	2.32	3.77	0.03	2.71	11.44	0.08	3.8	3.49	0.28	0.95	1.03	6.08	**30.76**	1.17	2.79	3.54	4.72	1.67	4.83	0.33	0.87	8.68	1.4	2.18	1.09	*100*
Melville	2.41	4.56	1.21	4.02	7.37	0.94	4.16	5.09	0.94	3.08	1.47	9.12	**14.88**	2.01	1.74	4.16	3.89	3.08	3.75	2.01	2.82	12.06	1.74	0.8	2.68	*100*
EauVive	2.06	4.16	0.64	3.25	9.96	0.49	3.01	4.52	0.94	2.49	1.64	7.47	**20.77**	1.73	2.28	6.74	3.64	2.19	4.4	2.03	1.4	8.04	2.61	1.18	2.37	*100*
Divertisse	1.76	10.67	0.28	4.99	4.92	0.42	1.97	3.79	0.42	1.26	0.91	10.96	**16.92**	2.74	1.12	4.42	3.72	2.18	5.2	2.46	4.78	10.88	1.05	0.84	1.33	*100*
Noé_	1.17	5.22	1.7	3.24	8.05	0.65	1.9	8.62	1.29	2.79	0.93	8.33	**13.51**	2.95	1.54	7.48	3.11	2.75	4.25	4.49	1.54	8.13	1.74	0.65	3.96	*100*
Paradis	0.37	2.34	1.61	2.56	**12.52**	0.37	7.76	5.56	1.17	1.98	0.37	3.37	**8.78**	2.56	4.39	4.54	2.34	1.46	6.66	2.2	1.98	5.05	**15.89**	1.68	2.49	*100*
Personnage	4.44	**8.42**	0.58	4.56	3.39	1.17	3.39	6.43	0.58	3.86	1.29	**13.68**	**6.43**	1.64	1.99	1.4	1.05	1.75	2.92	**6.67**	5.85	**14.97**	1.75	0.7	1.05	*100*
Ames	3.75	**11.6**	0.16	6.72	7.42	0.39	3.4	5.35	0.55	1.56	1.68	**10.74**	**10.63**	3.32	0.66	2.54	10.12	0.78	7.46	0.78	1.91	7.46	0.23	0.51	0.27	*100*
Chemins	6.04	9.99	0.55	9	5.8	0.12	6.17	3.58	0.86	2.59	3.88	7.83	**15.6**	1.73	0.62	3.95	2.4	0.62	0.68	1.48	1.48	13.32	0.74	0.43	0.55	*100*
Cavaliers	1.93	8.13	0.06	3.45	8.07	0	2.57	2.17	0.35	0.76	1.46	8.25	**30.49**	2.11	2.4	4.27	4.1	1.29	4.1	1.29	0.76	8.37	1.7	1.17	0.76	*100*
Pologne	4.2	2.89	1.03	4.94	2.24	0.47	**13.62**	**6.25**	1.03	**7.18**	1.4	**9.7**	**5.5**	3.45	1.31	1.59	1.87	4.85	**8.77**	1.87	3.54	5.32	2.99	0.28	3.73	*100*
Bonheur	2.3	2.18	0.17	3.47	6.45	0.05	3.32	4.2	0.41	2.16	1.24	5.14	**33.31**	2.04	2.52	4.2	5	2.09	4.29	0.39	0.87	5.82	3.4	1.82	3.15	*100*
Angelo	2.13	1.98	0.91	2.13	2.51	0.15	3.19	2.96	0.38	3.12	1.14	6.69	**50.15**	1.9	1.37	0.91	1.22	1.06	3.88	0.99	2.28	3.65	2.05	0.23	3.04	*100*
Hortense	11.53	2.78	0	0.6	5.57	0	1.99	8.35	0.2	2.78	0.8	5.17	**24.65**	2.78	0.8	2.58	4.97	5.37	5.17	0.2	0.8	2.39	6.36	0.8	3.38	*100*
Brigade	2.87	5.35	0.48	5.64	4.11	0.29	2.87	4.11	0.96	3.63	3.82	15.01	**16.83**	4.11	2.39	3.44	3.73	1.05	3.25	0.86	1.63	8.32	1.24	0.38	3.63	*100*
Ennemonde	0.75	4.16	1.28	1.49	11.73	0.11	3.2	8.85	0.75	2.03	0.53	3.52	**13.75**	3.3	1.49	6.93	3.73	5.01	9.06	3.52	2.03	4.37	1.92	0.85	5.65	*100*
Iris	1.98	4.39	0.25	3.09	4.21	0	1.11	1.24	0.74	1.42	1.67	10.89	**24.75**	2.78	1.3	6.37	5.63	4.64	6	1.86	0.87	9.84	0.74	1.73	2.48	*100*
total	67.29	127.9	11.68	89.33	193.3	8.34	92.33	102.5	17.67	51.82	35.44	194.9	**581.3**	53.31	43.54	125.5	95.13	58.07	131.6	39.03	42.62	193.8	56.23	25.83	61.47	*2500*
moyenne	2.69	5.12	0.47	3.57	7.73	0.33	3.69	4.1	0.71	2.07	1.42	7.79	**23.3**	2.13	1.74	5.02	3.81	2.32	5.26	1.56	1.7	7.75	2.25	1.03	2.46	*100*

Figure 3 : Recce Chart

2.4 The deconvolution step

2.4.1 From deep learning

The deconvolution step leads to the limits of the characterization of a writing. Projecting objects are spotted which justify the successful recognition. The deconvolution step crosses several criteria: Let's take the example of Regain, which has the highest recognition rate.The extract analysed is as follows:

An old roller axle. It is good to have a small box on the chimney, even if, on the small box there is marked pepper. It's good to have this box ready in case we get a chance to get a good mule. That can happen. We'll have to see. One cannot always live on borrowed money. In the path that descends, there is Arsule and his galoshes; one hears them both. Arsulesings.

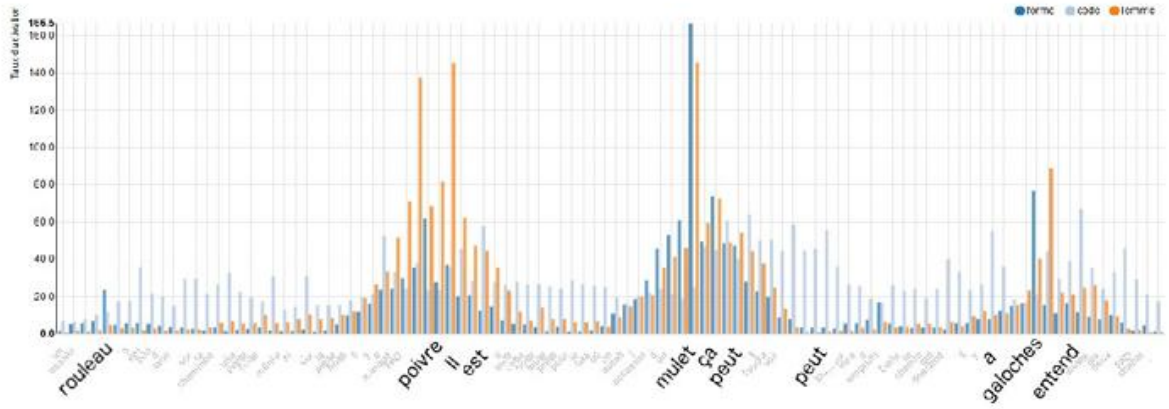

Figure 4 : Histogram – deconvolutionstep

We obtain a three-data histogram - form, code, lemma - and we can sometimes observe one another's variable. It turns out that the projections are on the forms «mule», «ca», «galoches», «pepper», «roll» for the first extract of 100 words that appears, «ploughs», «velvet» for the second as well as on the grammatical codes or grammatical categories «is», «a», "may", "intends" for the first excerpt – meaning that these forms are only retained by virtue of their verbal nature (Figure 4).

2.4.2 Segmentation for indexing

First, the idea is to segment the document. It is important that eachsegment retains meaning in order to be autonomous and therefore to be potentially written by adifferent person. A segmentation in unit of meaning is therefore preferred. Relying onthe work of Zechner et al. (2009) is a pseudo-semantic segmentation that has been retained:

a minimum sentence size (in words). The threshold was set at 15 words, size average of sentences in the French language [10].

The segmentation of documents and articles is treated under a variety of spaces in the literature according to the intended purpose, recognition of texts, in the Domain of information search, we recognize different methods:

- Segmentation into a sequence of words;
- Segmentation in sentences;
- Segmentation intoparagraphs;
- Thematic segmentation;
- Segmentation into logical units reflected in the table of contents;

Segmentation by a series of arbitrary words, it leaves aside the syntactical and semantic aspects of the text.

Segmentation in sentences is not reliable when one waits in response for a part of text that does not require inference work on the part of the user, knowing that the sentence does not have a guarantee of syntactic completeness, In the same way as segmentation into sentences, because of the difficulty of interpreting a paragraph in contexts in which it is attached to a unit preceding it or succeeding it [11].

2.5 Natural Langage Processing

2.5.1 introduction

Today, an increasingly large volume of data on the web is made up, to a very large extent, of textual data that can be analysed and exploited for different purposes. IT has enabled the development of tools to process information and establish solutions. Among the tools and

Computer techniques related to this field, we find the Automatic Natural Language Processing (TALN).solutions. Among the tools and computer techniques related to this field, we find the Automatic Natural Language Processing (TALN). Some languages have been preferred such as English and French, where searches are centralized to give tools for TALN applications. Others, such as the Arabic language, are continuing their research and work to propose robust processing tools for proposing TALN applications for these languages.

2.5.2 Definition of TALN

Automatic Natural Language Processing (TALN) refers to the set of research and methods that aim to allow a machine to automatically understand human language [12]. Much more than just the recognition of terms (or keywords), the TALN aims to “understand” the meaning of the sentences, the ideas that emerge from them, and this in the most optimal and natural way from a human point of view, and to provide an appropriate response without any external intervention being necessary.

It is the computer sub-domain, in particular artificial intelligence (AI), which aims to enable computers to understand and process human language. Technically, the main task of the TALN would be to program computers for the analysis and processing of a huge amount of natural language data.

2.5.3 Historic of TALN

Automatic processing of natural languages was born at the end of the forties of the last century, in a very precise scientific and political context [13].

- Year [1950 - 1968]

The beginnings in automatic processing of natural language began in the 1950s in the United States where the political context, related to the Cold War is conducive to the development of the theme of machine translation.

Between 1951 and 1954 Zelling Harris3 published his most important works of linguistics (distributionist linguistics);

1954: The development of the first (very rudimentary) automatic translator some Russian phrases, selected in advance, were automatically translated into English

1956: Dartmouth Summer School sees the birth of artificial intelligence

1957: N. Chomsky4 publishes his first important works on the syntax of natural languages, and on the relations between formal and natural grammars;

In the late 1960s, Terry Winograd, an MIT researcher, developed a natural language program called SHRDLU, allowing his user to converse with a computer to manage a "world of building blocks" (a blocks world). displaying on one of the first screens, This is the first program that knows how to understand and execute complex orders in natural language.

1962: First conference on translation (Bar-Hillel5) AlPAC6 report;1966: The ELIZA7 system (Weizenbaum);1968: The first (true) translation system (Systran, Russian English);Années [1970 - 1980]

During the 1970s many programmers began to write «conceptual ontologies» is the structured set of terms and concepts representing the meaning of a field of information, whether by the metadata of a namespace or the elements of a domain of knowledge , the purpose of which was to structure the information into data understandable by the computer , This is the case of MARGIE (Schank, 1975), SAM (Cullingford, 1978), PAM (Wilensky, 1978), TaleSpin (Meehan, 1976), SCRUPULE (Lehnert, 1977), Politics (Carbonell, 1979), Plot Units (Lehnert, 1981).

1971: A Smart System in Closed Mode (SHRDLU8), 1976: The METEO translation system developed at the Université de Montréal;

- Year [1990 -2000]

90s :First corpus, statistical approaches in machine learning. Applications use large corpus and statistical methods.

2000s :Use of the World Wide Web as a corpus.

- Since [2000].

In January 2018, artificial intelligence models developed by Microsoft and Alibaba each managed to beat humans in a reading and understanding test at Stanford University.

November 2018, Google launches BERT, a language model [14].

2.5.4 TALN statistic

The statistical TALN comprises all quantitative approaches to automated language processing, including modeling, information theory, and linear algebra [15]. The technology for statistical TALN comes primarily from machine learning and data mining, which involves learning from data from artificial intelligence.

2.5.5 The levels of TALN

The process of automatic processing of linguistic data requires different levels of analysis. The literature refers to morphological analysis, syntactic analysis, semantic analysis and pragmatic analysis (Figure 5). In the following we will briefly describe the different levels of analysis of a natural language text:

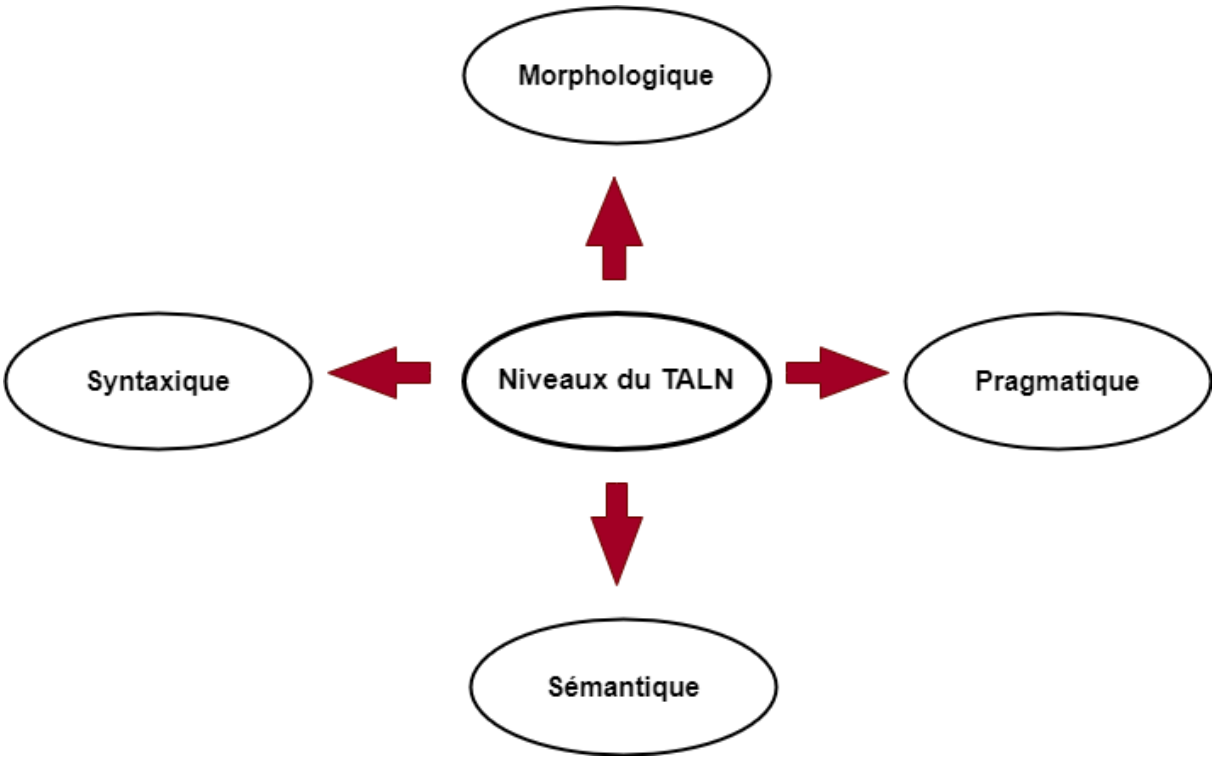

Figure 5: Representation of TALN levels

2.6 sorts of analysis

2.6.1 analysis syntactic

A formal language is defined by its grammar, whereas natural language is not. Indeed, a language is not defined by its syntax, because it is written later and presents only an approximation, from which we speak of syntax model. It is this approximation that makes parsing non-specific and difficult.

Several methods of syntactic analysis have been developed, such as Lemmatisation, Morphology, Morpho-syntactic labeling, Syntactic analysis, Sentence delimitation,

Racinisation, Word separation, but the best known is the notion of formal grammar. It is presented as a set of derivation rules expressing the structure of syntactical entities such as phrase (PH), nominal group (GN), verbal group (GV) and so on. For example, to express that a sentence is composed of a nominal group and a verbal group, we use the PH GN + GV rule. Also, to express that a nominal group is composed of a determinant and a name, we use the GN Det + Name rule. Using this set of rules, it is therefore possible to analyze a number of sentences [16], Figure 6 shows an example of an arborescent representation of the sentence: << the student has written the course>>

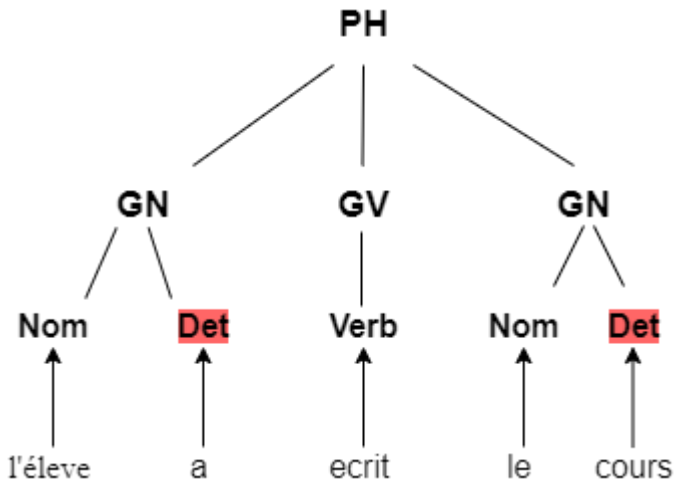

Figure 6: Syntax tree showing an example sentence

2.6.2 analysis semantic

According to Jean VERONIS9 [17], «In an automatic processing system, the analysis of the meaning of sentences usually consists in extracting a simplified, stylized representation of a logicomatic type, which will allow later calculations and reasoning».

Semantic analysis of statements is based on preliminary syntactic analysis. It seeks to construct a formal representation allowing reasoning and thus to infer new information from the information present in the statement. Among the representations is the logic of the proposals. Using logical connectors (such as conjunction “ ”, disjunction “ ”, negation “ etc.) one can form from the proposals complex new proposals. The logic of the proposals is not concerned with the content of the proposals but only with their values of truth. Thus, many phenomena cannot be represented in the logic of predicates.

Another form of representation called "semantic networks" has been proposed, its principle being to represent knowledge in the form of a graph (or network) of concepts. Nodes represent concepts and arcs represent the relationships between these concepts. Several types of relationships between concepts exist such as: EST-UN, SORT-DE, EST-PARTIE-DE, etc.

The use of these concepts requires navigation tools in the graph to understand the meaning of the sentence and the relationships between the different words that make up it.

Semantic networks have been expanded to improve the representation and inference of knowledge [16]. 9Jean Veronis: French born in 1955, Professor of linguistics and computer science, specialist in TAL.

2.6.3 analysis pragmatic

According to J-H. JAYEZ10 [18], Pragmatism concerns the study of the environment of a sentence, at the time when it is issued; it arises from the idea that a sentence (a statement) can take its full meaning only if it is (it) replaced in its original milieu; it is the taking into account of all the conditions of production of a sentence, as it is true that an effective linguistic act can only take place within a certain communication situation». This level of analysis covers everything related to the implicit in the communication. It is therefore the level that poses the most problems to be designed and therefore it is much more complex to establish, which explains why there is little operational realization, which concerns a few applications. We are still far from knowing how to build pragmatic analyzers for the TALN [16].

2.7 Research fields and applications of TALN

The field of automatic natural language processing covers a wide range of research disciplines which can apply skills as diverse as applied mathematics or signal11

2.8 Applications related to production where text editing

Automatic translation: This is one of the most complex problems, says AI-complete, which requires a lot of knowledge, not only linguistic, but also about the world. It is the first research application, active since the 1950s.

Automatic generation of texts: Writing texts that are syntactically and semantically correct, for example to produce weather reports or automated reports.

Automatic Text Summary, Restatement and Paraphrasing: Extracting the relevant content of a text, detecting the most important information, redundancies, in order to generate a coherent text that is humanly credible.

word sense disambiguation: Still an unresolved problem, determining the meaning of a word in a sentence, when it can have several possible meanings, depending on the general context.

spelling correction: in addition to a comparison with dictionary words and a rough search to propose corrections, there are grammatical proofreaders who use semantics and context to correct homophonies.

Conversational Agents, and Question and Answer Systems: Combination of a language comprehension step and then a text generation step.

Detection of co-references and resolution of anaphores: Detection of the connection between several words of a sentence referring to the same subject.

2.9 Conclusion

In this chapter we have presented the state of the art of TALN based on its history, treatment levels, areas of application, which presents many challenges for various fields such as automatic processing of natural language or also searching for information.

To conclude, this notion of pen measurement approach makes it possible to detect different styles of writing within the same text and our contribution despite its limitations allows to automatically group the stylistic phases by author.

The calculations of deep learning have proved to be effective for the recognition of an author: Giono's works have all been recognized (except for about one), after learning about 4 novels by Giono and 2 novels by 25 other contemporary writers.

The more traditional tools to observe the lexical and grammatical properties of the object of study are complementary to deep learning, while waiting for the progress to come regarding the «deconvolution» stage.

Chapter III: State of the art on classification

Chapter III: State of the art on classification

3.1 Introduction

In real life, the pieces of information that the brain processes have an inherent structure and
order, and the organization and sequence of every phenomenon we perceive has an influence on how we treat them. Examples of this include speech comprehension (the order of the words in a sentence), video sequence (the order of the frames in a video), and language translation. This prompted the creation of new models. The most important ones are grouped under the RNN umbrella.

3.2 Definition of RNN

Recurrent Neural Network (RNN) is a class of deep learning based of the works of David Rumelhart in 1986. RNNs are heralded for their ability to process and obtain insights from sequential data. Therefore, video analysis, image captioning, natural language processing (NLP), and music analysis all depend on the capabilities of recurrent neural networks. Unlike standard neural networks that assume independence among data points, RNNs actively capture sequential and time dependencies between data.

One of the most defining attributes about RNNs is parameter sharing. Without parameter sharing, a model allocates unique parameters to represent each data point in a sequence and therefore cannot make inferences about variable length sequences. The impact of this limitation can be fully observed in natural language processing. For example, the sentences to decode are “Kobe Bryant is an incredible basketball player” and “An incredible basketball player is Kobe Bryant”. An ideal model should be able to recognize that ‘Kobe Bryant’ is the basketball player discussed in both sentences regardless the position of the words. A traditional multilayer network in this scenario would fail because it would create an interpretation of the language with respect to the unique weights set for each position (word) in the sentence. RNNs, however, would be more suitable for the task as they share weights across time steps (i.e. the words in our sentence)—enabling more accurate sentence comprehension [19](Figure 7).

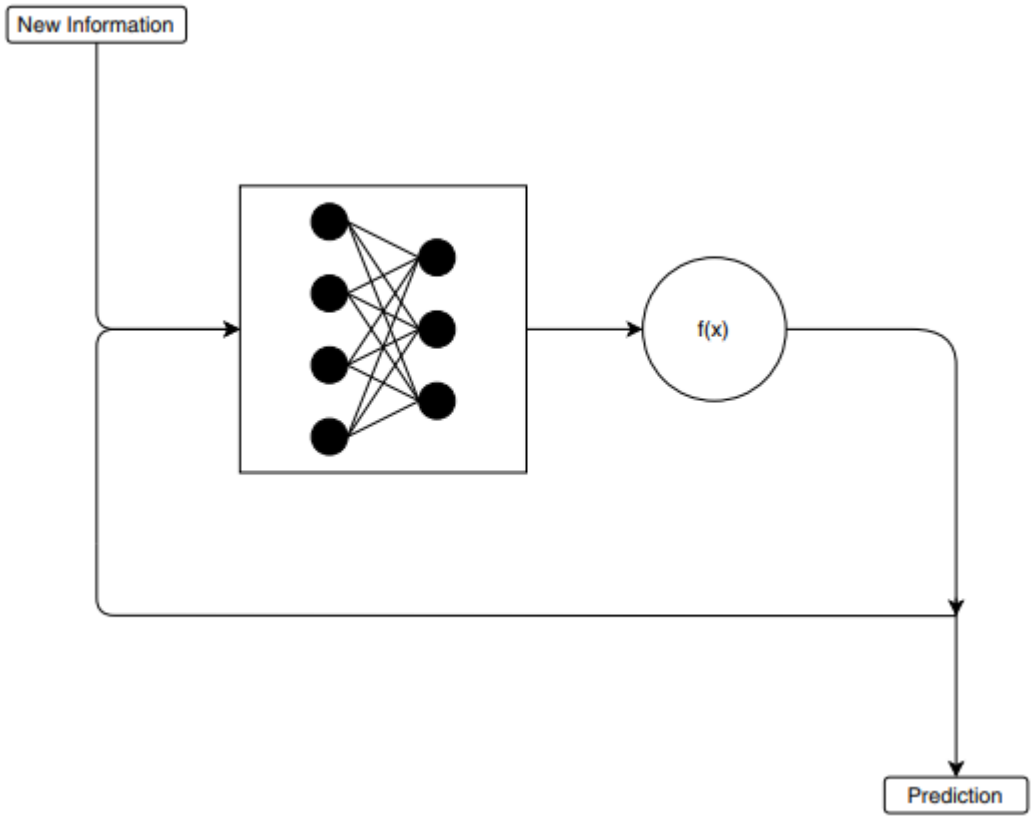

Figure 7: A condensed representation of Recurrent Neural Network (RNN). It is a neural network that recurs over time, which allows information to persist by loops. The f(x) represents some squashing function.

3.3 Basic RNN Cell

Traditional multilayer perceptron neural networks make the assumption that all inputs are independent of each other. This assumption is not true for many types of sequence data. For example, words in a sentence, musical notes in a composition, stock prices over time, or even molecules in a compound, are examples of sequences where an element will display a dependence on previous elements.

RNN cells incorporate this dependence by having a hidden state, or memory, The value of the hidden state at any point in time is a function of the value of the hidden state at the previous time step, and the value of the input at the current time step, that is:

$$htt = (htt - 1)$$

Here, ht and ht-1 are the values of the hidden states at the time t and t-1 respectively, and xt is the value of the input at time t. Notice that the equation is recursive, that is, ht-1 can be represented in terms of ht-2 and xt-1, and so on, until the beginning of the sequence. This is how RNNs encode and incorporate information from arbitrarily long sequences.

We can also represent the RNN cell graphically as shown in Figure 8(a). At time t, the cell has an input x(t) and output y(t). Part of the output y(t) (represented by the hidden state ht) is fed back into the cell for use at a later time step t+1.

Just as in a traditional neural network, where the learned parameters are stored as weight matrices, the RNN's parameters are defined by the three weight matrices U, V, and W, corresponding to the weights of the input, output, and hidden states respectively:

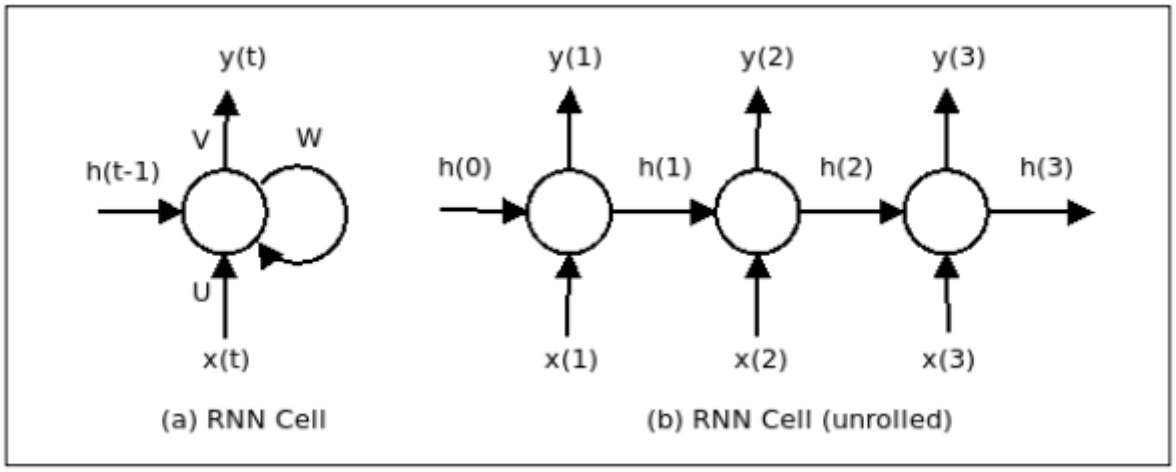

Figure 8: (a) Schematic of an RNN cell; **(b)** RNN cell unrolled

Figure 8(b) shows the same RNN in an "unrolled view". Unrolling just means that we draw the network out for the complete sequence. The network shown here has three time steps, suitable for processing three element sequences. Note that the weight matrices *U*, *V*, and *W*, that we spoke about earlier, are shared between each of the time steps. This is because we are applying the same operation to different inputs at each time step. Being able to share these weights across all the time steps greatly reduces the number of parameters that the RNN needs to learn.

We can also describe the RNN as a computation graph in terms of equations. The internal state of the RNN at a time *t* is given by the value of the hidden vector *h(t)*, which is the sum of the weight matrix *W* and the hidden state *ht-1* at time *t-1*, and the product of the weight matrix *U* and the input *xt*at time *t*, passed through a tanh activation function. The choice of tanh over other activation functions such as sigmoid has to do with it being more efficient for learning in practice.

We have omitted explicit reference to the bias terms by incorporating it within the matrix. Consider the following equation of a line in an n-dimensional space. Here w1through wn refer to the coefficients of the line in each of the *n* dimensions, and the bias *b* refers to the y-intercept along each of these dimensions.

$$y = w1x1 + w2x2 + \cdots + wnxn + bb$$

We can rewrite the equation in matrix notation as follows:

$$y = Wx + b$$

Here *W* is a matrix of shape (*m, n*) and *b* is a vector of shape (*m,1*), where *m* is the number of rows corresponding to the records inour dataset, and *n* is the number of columns corresponding to thefeatures for each record. Equivalently, we can eliminate the vector *b* by folding it into our matrix *W* by treating the *b* vector as a featurecolumn corresponding to the "unit" feature of *W*. Thus:

$$y = w1x1 + w2x2 + \cdots + wnxn + w0 \quad (1)$$
$$= W'X$$

Here *W'* is a matrix of shape (*m, n+1*), where the last column contains the values of *b*, The resulting notation ends up being more compact and (we believe) easier for the reader to comprehend and retain as well.

The output vector *yt*at time *t* is the product of the weight matrix *V* and the hidden state *ht*, passed through a softmax activation, such that the resulting vector is a set of output probabilities:

$$ht = \tanh(Wht - 1 + Uxt)$$
$$yt = \text{softmax}(Vht)$$

Keras provides the SimpleRNN recurrent layer that incorporates all the logic we have seen, as well as the more advanced variants such as LSTM and GRU.

3.4 Multi-layer perceptron

3.4.1 Definition

We present a network with multiple dense layers, Historically, "perceptron" was the name given to a model having one single linear layer, and as a consequence, if it has multiple layers, you would call it a multi-layer perceptron (MLP). Note that the input and the output layers are visible from outside, while all the other layers in the middle are hidden – hence the name hidden layers. In this context, a single layer is simply a linear function and the MLP is therefore obtained by stacking multiple single layers one after the other:

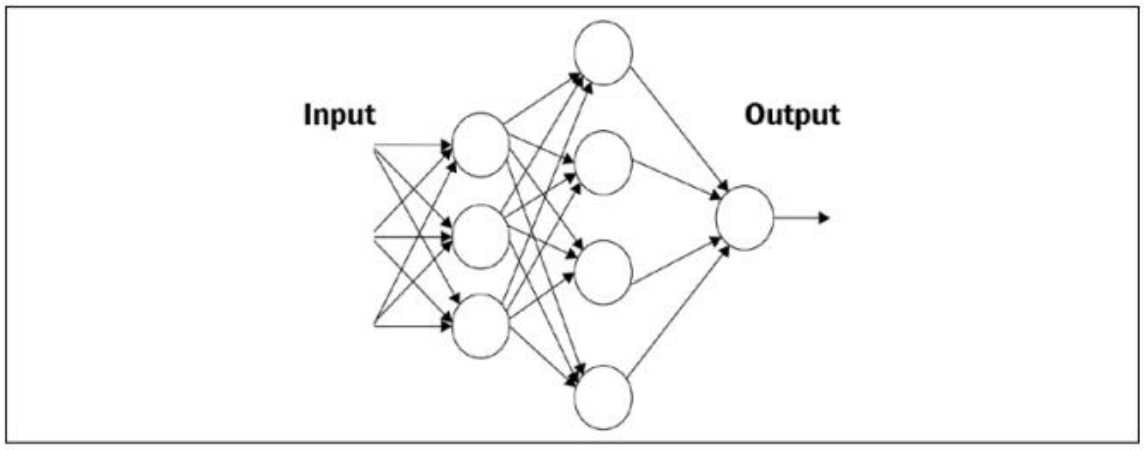

Figure 9: An example of a multiple layer perceptron

In Figure 9 each node in the first hidden layer receives an input and "fires" (0,1) according to the values of the associated linear function. Then, the output of the first hidden layer is passed to the second layer where another linear function is applied, the results of which are passed to the final output layer consisting of one single neuron. It is interesting to note that this layered organization vaguely resembles the organization of the human vision system.

3.5 Activation functions

Sigmoid, Tanh, ELU, LeakyReLU, and ReLU are generally called activation functions in neural network jargon. Those gradual changes typical of sigmoid and ReLU functions are the basic building blocks to develop a learning algorithm that adapts little by little by progressively reducing the mistakes made by our nets. An example of using the activation function $\sigma\sigma$ with (x1 , x2 ,..., xm) input vector, (w1 , w2 ,..., wm) weight vector, b bias, and $\sum$ summation is given in (Figure 10). Note that TensorFlow 2.0 supports many activation functions :

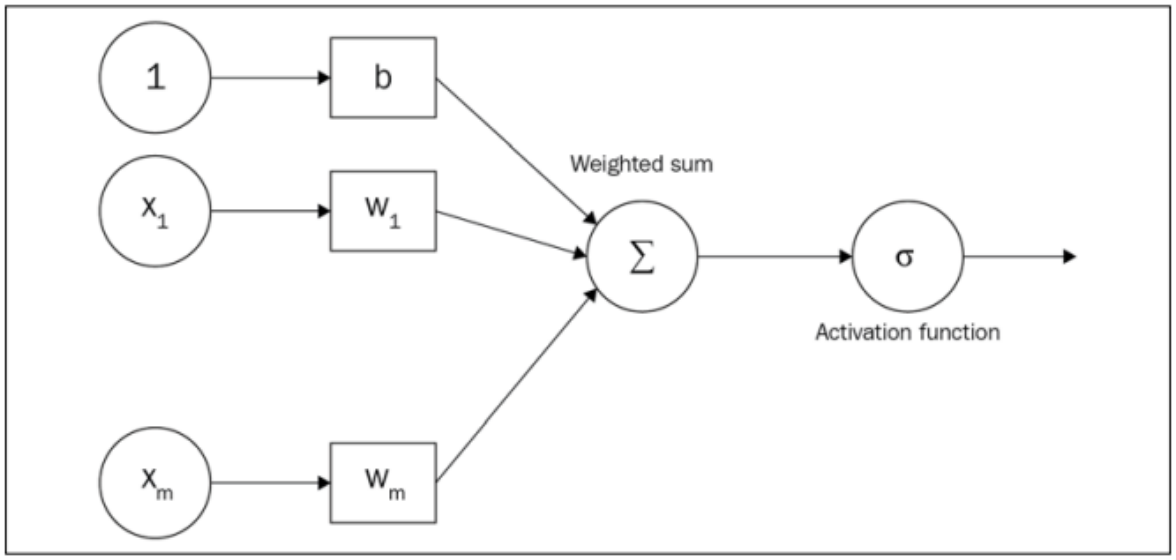

Figure 10 : An example of an activation function applied after a linear function

3.5.1 Activation function – sigmoid

The sigmoid function defined as (x) = 1 / 1+ e^-x and represented in the following figure has small output changes in the range (0, 1) when the input varies in the range $(-\infty, \infty)$. Mathematically the function is continuous. A typical sigmoid function is represented in (Figure 11):

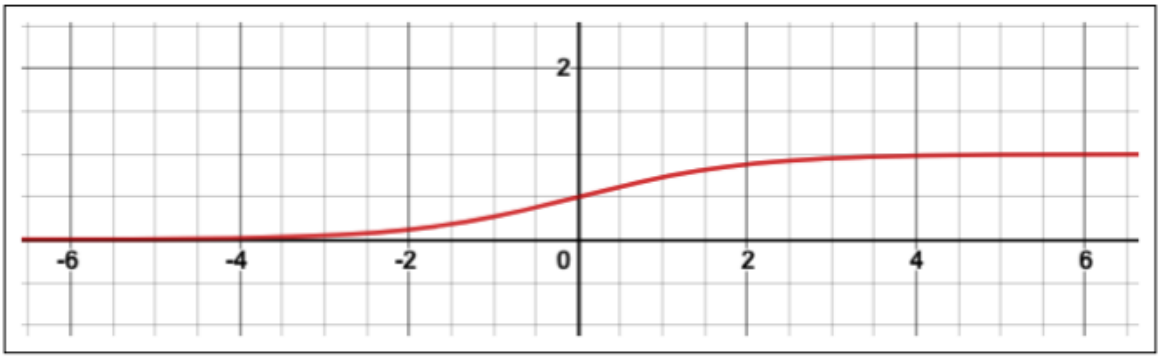

Figure 11: A sigmoid function with output in the range (0,1)

A neuron can use the sigmoid for computing the nonlinear function ($z = wx + b$). Note that if z = wx + b is very large and positive, then e^-z → 0 so (z) → 1, while if z = wx + b is very large and negative e^-z → ∞ so (z) → 0. In other words, a neuron with sigmoid activation has a behavior similar to the perceptron, but the changes are gradual and output values such as 0.5539 or 0.123191 are perfectly legitimate. In this sense, a sigmoid neuron can answer "maybe."

3.5.2 Activation function – tanh

Another useful activation function is tanh. Defined as tanh(z) = e^z – e^-z/e^z – e^-z whose shape is shown in (Figure 12), its outputs range from -1 to 1:

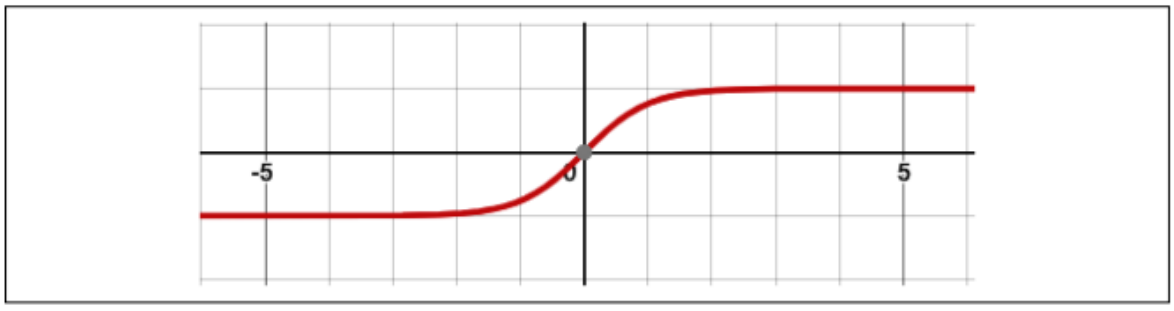

Figure 12: Tanh activation function

3.5.3 Activation function – ReLU

The sigmoid is not the only kind of smooth activation function used for neural networks. Recently, a very simple function named ReLU (REctified Linear Unit) became very popular because it helps address some optimization problems observed with sigmoids. A ReLU is simply defined as f(x) = max(0, x) and the non-linear function is represented in (Figure 13). As you can see, the function is zero for negative values and it grows linearly for

positive values. The ReLU is also very simple to implement (generally, three instructions are enough), while the sigmoid is a few orders of magnitude more. This helped to squeeze the neural networks onto an early GPU:

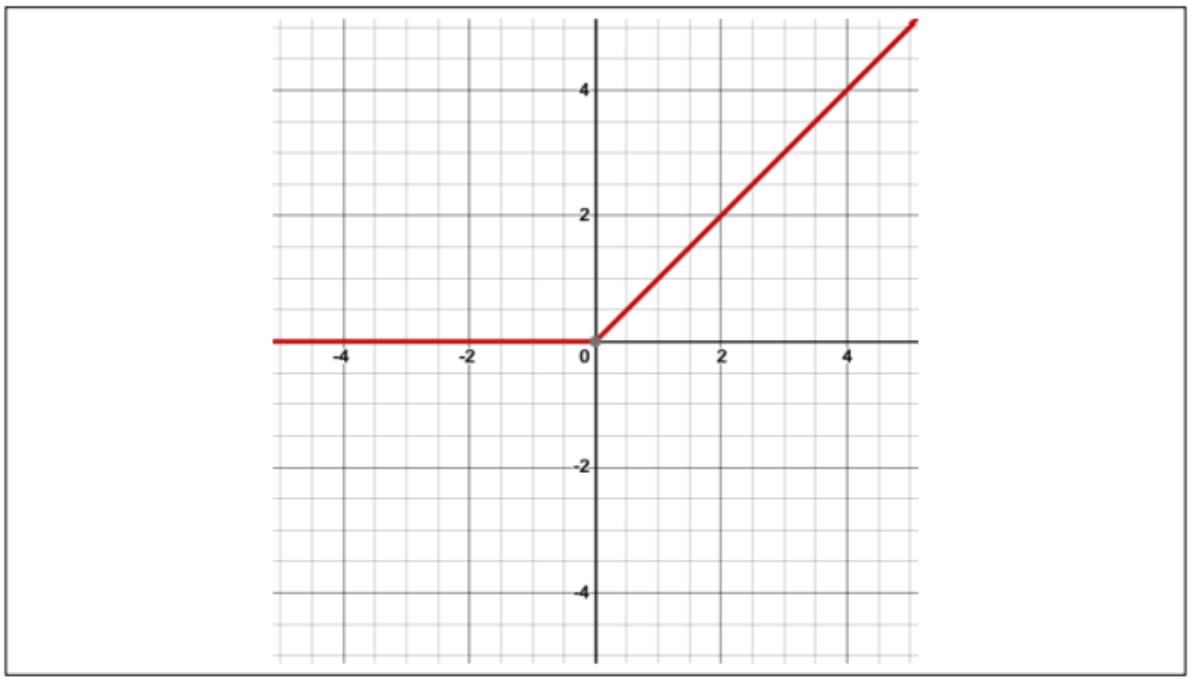

Figure 13: A ReLU function

3.5.4 Two additional activation functions – ELU and LeakyReLU

Sigmoid and ReLU are not the only activation functions used for learning.

ELU is defined as $(\alpha, x) = \begin{cases} \alpha(e^x - 1) & if\ x \leq 0 \\ x & if\ x > 0 \end{cases}$ for $\alpha\alpha > 0$ and its plot is represented in (Figure 14):

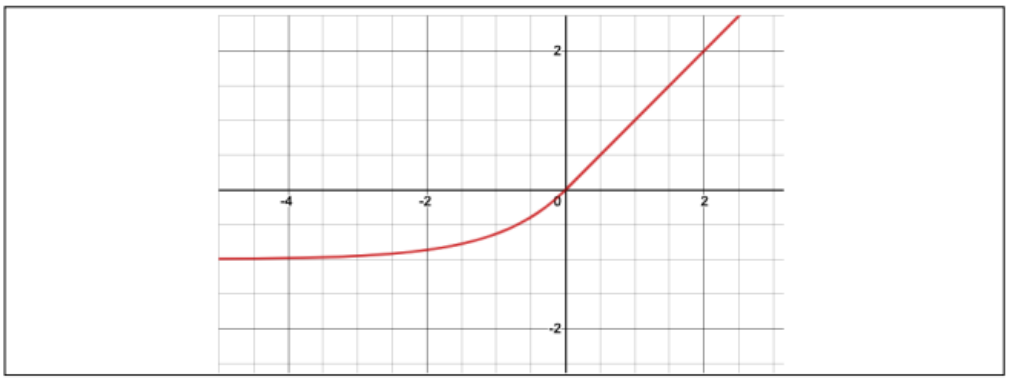

Figure 14: An ELU function

LeakyReLU is defined as $(\alpha, x) = \begin{cases} \alpha x & if\ x \leq 0 \\ x & if\ x > 0 \end{cases}$ for $\alpha\alpha > 0$ and its plot is represented in (Figure 15):

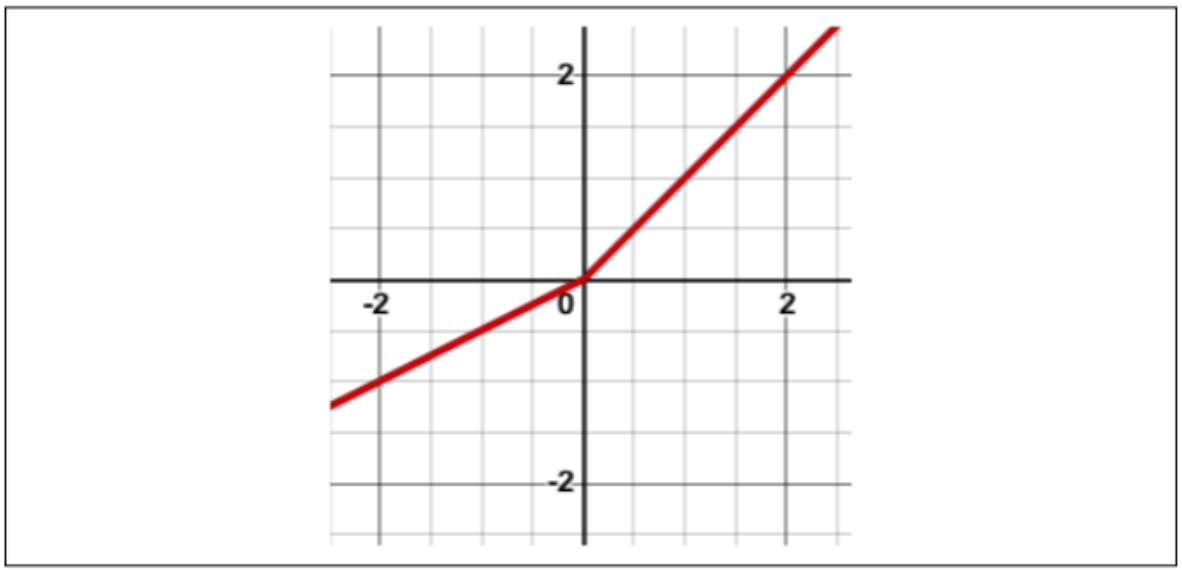

Figure 15: A LeakyReLU function

Both the functions allow small updates if x is negative, which might be useful in certain conditions.

3.6 Back propagation throught time (BPTT)

Just like traditional neural networks, training RNNs also involves backpropagation of gradients. The difference in this case is that since the weights are shared by all time steps, the gradient at each output depends not only on the current time step, but also on the previous ones. This process is called backpropagation through time [20]**.** Because the weights *U*, *V*, and *W*, are shared across the different time steps in case of RNNs, we need to sum up the gradients across the various time steps in case of BPTT. This is the key difference between traditional backpropagation and BPTT.

Consider the RNN with five time steps shown in (Figure 16). During the forward pass, the network produces predictions $\hat{y}t$at time *t* that are compared with the label *yt*tocompute a loss *Lt*. During backpropagation (shown by the dotted lines), the gradients of the loss with respect to the weights *U*, *V*, and *W*, are computed at each time step and the parameters updated with the sum of the gradients:

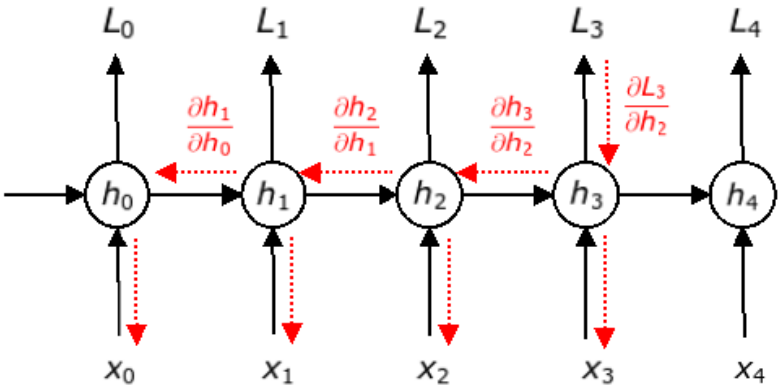

Figure 16 :Backpropagationthrought time.

The following equation shows the gradient of the loss with respect to *W*. We focus on this weight because it is the cause for the phenomenon known as the vanishing and exploding gradient problem.

This problem manifests as the gradients of the loss approaching either zero or infinity, making the network hard to train. To understand why this happens, consider the equation of the SimpleRNN we saw earlier; the hidden state *ht*is dependent on *ht-1*, which in turn is dependent on *ht-2*, and so on:

$$\frac{\partial L}{\partial W} = \sum_t \frac{\partial L_t}{\partial W}$$

Let us now see what happens to this gradient at timestep*t=3*. By the chain rule, the gradient of the loss with respect to *W* can be decomposed to a product of three sub-gradients. The gradient of the hidden state *h2* with respect to *W* can be further decomposed as the sum of the gradient of each hidden state with respectto the previous one. Finally, each gradient of the hidden state with respect to the previous one can be further decomposed as the product of gradients of the current hidden state against the previous hidden state:

$$\frac{\partial L_3}{\partial W} = \frac{\partial L_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial h_3}\frac{\partial h_3}{\partial W}$$

$$= \sum_{t=0}^{3} \frac{\partial L_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial h_3}\frac{\partial h_3}{\partial h_t}\frac{\partial h_t}{\partial W}$$

$$= \sum_{t=0}^{3} \frac{\partial L_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial h_3}\left(\prod_{j=t+1}^{3} \frac{\partial h_j}{\partial h_{j-1}}\right)\frac{\partial h_t}{\partial W}$$

Similar calculations are done to compute the gradient of the other losses *L0* through *L4* with respect to *W*, and sum them up into the gradient update for *W*.We will not explore the

mathfurther in this book, but this WildML blog post [21]has a very good explanation of BPTT, including a more detailed derivation of themath behind the process.

3.7 Vanishing and exploding gradients

The reason BPTT is particularly sensitive to the problem of vanishing and exploding gradients comes from the product part of the expression representing the final formulation of the gradient of the loss with respect to *W*. Consider the case where the individual gradients of a hidden state with respect to the previous one is less than 1.

As we backpropagate across multiple time steps, the product of gradients get smaller and smaller, ultimately leading to the problem of vanishing gradients. Similarly, if the gradients are larger than 1, the products get larger and larger, and ultimately lead to the problem of exploding gradients.

Of the two, exploding gradients are more easily detectable. The gradients will become very large and turn into **Not a Number** (**NaN**) and the training process will crash. Exploding gradients can be controlled by clipping them at a predefined threshold [22]. TensorFlow 2.0 allows you to clip gradients using the clipvalue or clipnorm parameter during optimizer construction, or by explicitly clipping gradients using tf.clip_by_value.

The effect of vanishing gradients is that gradients from time steps that are far away do not contribute anything to the learning process, so the RNN ends up not learning any long-range dependencies. While there are a few approaches to minimizing the problem, such as proper initialization of the *W* matrix, more aggressive regularization, using ReLU instead of tanh activation, and pretraining the layers using unsupervised methods, the most popular solution is to use LSTM or GRU architectures, each of which will be explained shortly. These architectures have been designed to deal with vanishing gradients and learn long-term dependencies more effectively.

3.8 RNN cell variants

In this section we'll look at some cell variants of RNNs. We'll begin by looking at a variant of the SimpleRNN cell: the Long short-term memory RNN.

3.8.1 Long short-term memory (LSTM)

The LSTM is a variant of the SimpleRNN cell that is capable of learning long-term dependencies. LSTMs were first proposed by Hochreiter and SchmidHuber[23] and refined by many other researchers. They work well on a large variety of problems and are the most widely used RNN variant.

As previously mentioned, RNN suffers from a context problem which is attributable to the phenomenon known as the vanishing gradient problem. The vanishing gradient problem occurs when gradient descent is used as an optimization algorithm along with backpropagation [24]. As gap sizes increase between dependencies, the error gradients vanish exponentially and may result in the training of a network to become very slow or even unable to learn

We have seen how the SimpleRNN combines the hidden state from the previous time step and the current input through a tanh layer to implement recurrence. LSTMs also implement recurrence in a similar way, but instead of a single tanh layer, there are four layers interacting in a very specific way. The following diagram illustrates the transformations that are applied in the hidden state at time step *t*.

The line across the top of the diagram is the cell state *c*, representing the internal memory of the unit.

The line across the bottom is the hidden state *h*, and the *i*, *f*, *o*, and *g* gates are the mechanisms by which the LSTM works around the vanishing gradient problem. During training, the LSTM learns the parameters for these gates (Figure 17):

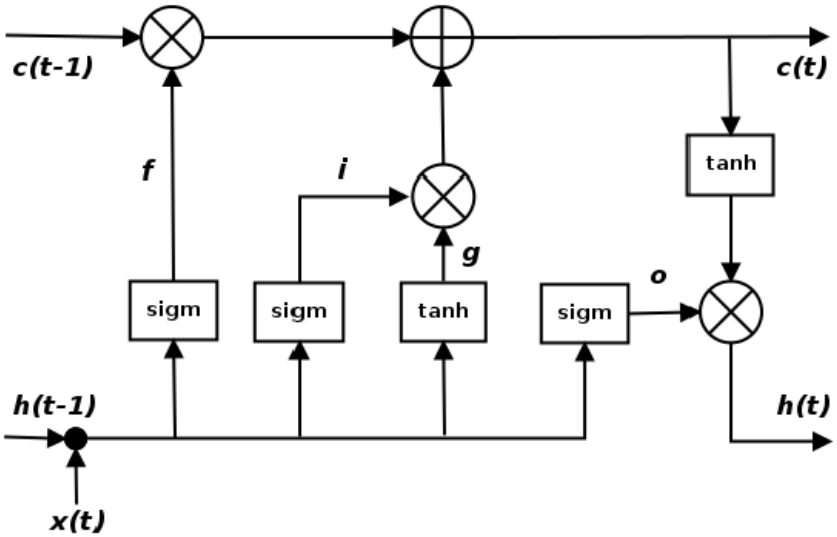

Figure 17 : An LSTM cell

An alternative way to think about how these gates work inside an LSTM cell is to consider the equations for the cell. These equations describe how the value of the hidden state *ht*at time *t* is calculated from the value of hidden state *ht-1* at the previous time step. In general, the equation-based description tends to be clearer and more concise, and is usually the way a new cell design is presented in academic papers. Diagrams, when provided, may or may not be comparable to ones you have seen earlier. For these reasons, it usually makes sense to learn to read the equations and visualize the cell design.

The set of equations representing an LSTM are shown as follows:

$$i = \sigma(W_i h_{t-1} + U_i x_t + V_i c_{t-1})$$
$$f = \sigma(W_f h_{t-1} + U_f x_t + V_f c_{t-1})$$
$$o = \sigma(W_o h_{t-1} + U_o x_t + V_o c_{t-1})$$
$$g = \tanh(W_g h_{t-1} + U_g x_t)$$
$$c_t = (f * c_{t-1}) + (g * i)$$
$$h_t = \tanh(c_t) * o$$

Here *i*, *f*, and *o*are the input, forget, and output gates. They are computed using the same equations but with different parameter matrices *Wi*, *Ui*, *Wf*, *Uf*, and *Wo*, *Uo*. The sigmoid function modulates the output of these gates between 0 and 1, so the output vectors produced can be multiplied element-wise with another vector to define how much of the second vector can pass through the first one.
The forget gate defines how much of the previous state *ht-1* you want to allow to pass through. The input gate defines how much of the newly computed state for the current input *xt*you want to let through, and the output gate defines how much of the internal state you want to expose to the next layer. The internal hidden state *g* is computed based on the current input *xt*and the previous hidden state *ht-1*. Notice that the equation for *g* is identical to that for the SimpleRNN, except that in this case we will modulate the output by the output of input vector *i*.
Given *i, f, o,* and *g*, we can now calculate the cell state *ct*at time *t* as the cell state *ct-1* at time (*t-1*) multiplied by the value of the forget gate *g*, plus the state *g* multipliedby the input gate *i*. This is basically a way to combine the previous memory andthe new input – setting the forget gate to 0 ignores the old memory and setting theinput gate to 0 ignores the newly computed state. Finally, the hidden state *ht*at time *t* is computed as the memory *ct*at time *t*, with the output gate *o*.
One thing to realize is that the LSTM is a drop-in replacement for a SimpleRNN cell; the only difference is that LSTMs are resistant to the vanishing gradient problem., You can replace an RNN cell in a network with an LSTM without worrying about any side effects. You should generally see better results along with longer training times.

3.8.1.1 LSTM Gates

The critical components of the LSTM are the memory cell and its gates. There are different variations of LSTM but they all predominantly include three gates, known as the forget gate, input gate, and output gate. The contents of the memory cell are modulated by the input gates and forget gates. Assuming that both of these gates are closed.the contents of the memory cell will remain unmodified between one time-step and the next.

The gating structure allows information to be retained across many time-steps, and consequently also allows gradients to flow across many timesteps. This allows the LSTM model to overcome the vanishing gradient problem that occurs with most Recurrent Neural Network models. The unfolded graph of an LSTM network can be thought of as a conveyor belt, with the data passing along the from one layer to the next, being altered slightly as it passes through each layer by use of the input and forget gates using linear interactions.

The forget gate is responsible for removing information from the cell state and its goal is to identify which information is no longer useful and may be forgotten. It takes 2 inputs: the Hidden State from the previous memory cell, h(t−1), and the Current Input, x(t), also known as the current cell state at that particular time step.

The inputs are multiplied by weight matrices and a bias is added. After that, a sigmoid function is applied; the sigmoid function is responsible for deciding which values to keep and which to discard. The function outputs a vector with values 0 to 1; a 0 indicates the forget gate wants to forget the information completely while a 1 indicates the forget gate wants to remember the entire piece of information.

The input gate involves a 2-step process and is responsible for deciding what new information will be added to the cell state. Similar to the forget gate, a sigmoid function is applied to h(t−1) and x(t). A hyperpolic tangent function creates a vector of all possible values, ranging from −1 to 1. This vector indicates candidate values which may be added to the cell state.

The output gate selects useful information from the cell state as output in a 3-step process. In the first step, a hyperbolic tangent function is applied to cell state, creating a vector with scaled values from −1 to 1. Step 2 is to use sigmoid function and use the previous hidden state, h(t−1), and x(t) as inputs to create a regulatory filter.

In the final step, the regulatory filter from step 2 is multiplied with the vector from step 1, producing an output and hidden state to the next cell. Using LSTM, the network is able to minimize any long term dependencies and can bridge gaps in data references in excess of 1,000 steps[25] (Fig. 18).

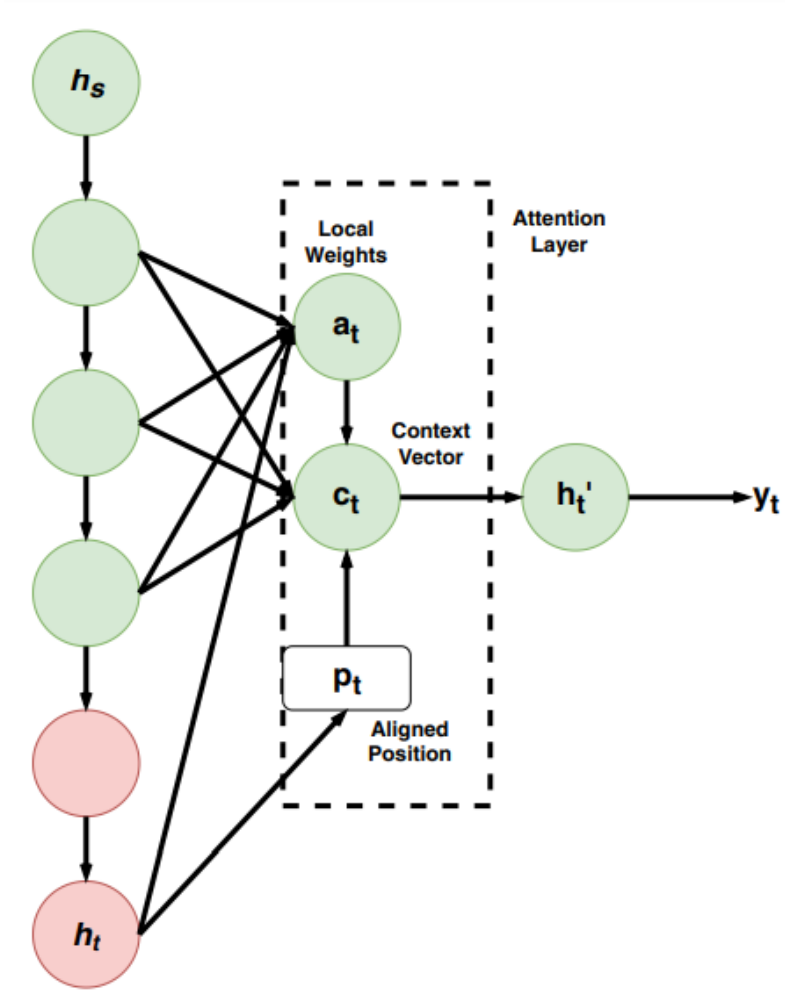

Figure 18 : Attention network

3.8.1.2 Peephole LSTM

The peephole LSTM is an LSTM variant that was first proposed by Gers and Schmidhuber [26]**.** It adds "peepholes" to the input, forget, and output gates, so they can see the previous cell state ct-1. The equations for computing the hidden state ht , at time t, from the hidden state ht-1 at the previous time step, in a peephole LSTM are shown next.

Notice that the only difference from the equations for the LSTM is the additional ct-1 term for computing outputs of the input (i), forget (f), and output (o) gates:

$$i = \sigma(W_i h_{t-1} + U_i x_t + V_i c_{t-1})$$
$$f = \sigma(W_f h_{t-1} + U_f x_t + V_f c_{t-1})$$
$$o = \sigma(W_o h_{t-1} + U_o x_t + V_o c_{t-1})$$
$$g = \tanh(W_g h_{t-1} + U_g x_t)$$
$$c_t = (f * c_{t-1}) + (g * i)$$
$$h_t = \tanh(c_t) * o$$

TensorFlow 2.0 provides an experimental implementation of the peephole LSTM cell. In order to use this in our RNN layers , we need to wrap the cell in the RNN wrapper :

```
hidden_dim = 256
peephole_cell = tf.keras.experimental.PeepholeLSTMCell(hidden_dim)
rnn_layer = tf.keras.layers.RNN(peephole_cell)
```

3.8.2 Gated recurrent unit (GRU)

The GRU is a variant of the LSTM and was introduced by K. Cho [27]. It retains the LSTM's resistance to the vanishing gradient problem, but its internal structure is simpler, and therefore is faster to train, since fewer computations are needed to make updates to its hidden state. The gates for a GRU cell are illustrated in the following diagram (Figure 19):

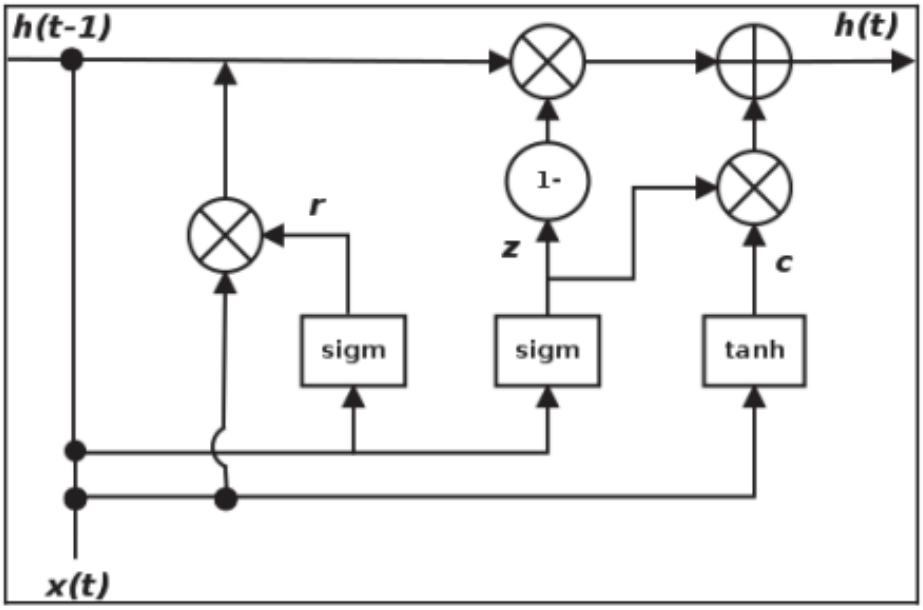

Figure 19 : Illustrate GRU Cell

Instead of the input (i), forgot (f), and output (o) gates in the LSTM cell, the GRU cell has two gates, an update gate z and a reset gate r. The update gate defines how much previous memory to keep around, and the reset gate defines how to combine the new input with the previous memory. There is no persistent cell state distinct from the hidden state as it is in LSTM.

The GRU cell defines the computation of the hidden state ht at time t from the hidden state ht-1 at the previous time step using the following set of equations:

$$z = \sigma(W_z h_{t-1} + U_z x_t)$$
$$r = \sigma(W_r h_{t-1} + U_r x_t)$$
$$c = \tanh\left(W_c(h_{t-1} * r) + U_c x_t\right)$$
$$h_t = (z * c) + ((1 - z) * h_{t-1})$$

He outputs of the update gate z and the reset gate r are both computed using a combination of the previous hidden state ht-1 and the current input xt . The sigmoid function

modulates the output of these functions between 0 and 1. The cell state c is computed as a function of the output of the reset gate r and input xt . Finally, the hidden state ht at time t is computed as a function of the cell state c and the previous hidden state ht-1. The parameters Wz ,Uz , Wr , Ur , and Wc , Uc , are learned during training.

GRU and LSTM have comparable performance and there is no simple way to recommend one or the other for a specific task. While GRUs are faster to train and need less data to generalize, in situations where there is enough data, an LSTM's greater expressive power may lead to better results. Like LSTMs, GRUs are drop-in replacements for the SimpleRNNcell[28].

3.9 RNN variants topologies

RNNs offer yet another degree of freedom, in that it allows sequence input and output. This means that RNN cells can be arranged in different ways to build networks that are adapted to solve different types of problems. (Figure 20) shows five different configurations of inputs, hidden layers, and outputs, represented by red, green, and blue boxes respectively:

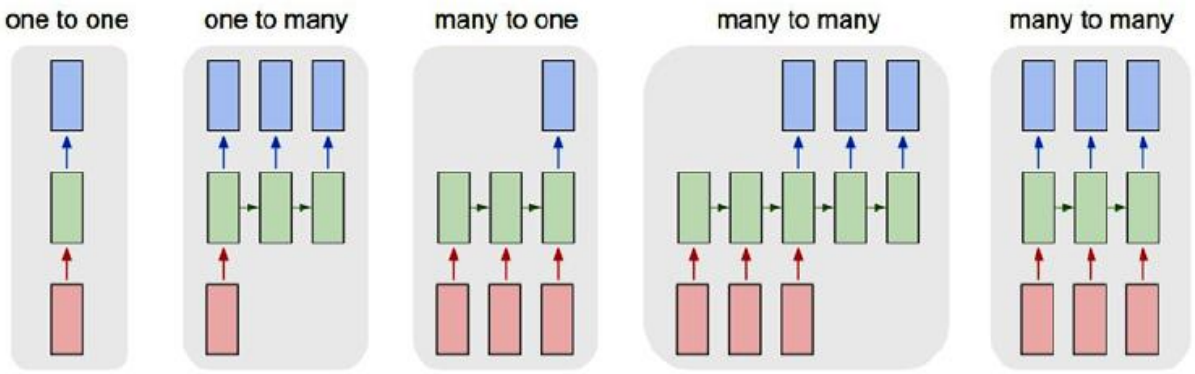

Figure 20 : Common RNN topologies. Image Source: AndrejKarpathy[31].

Of these, the first one (one-to-one) is not interesting from a sequence processing point of view, since it can be implemented as a simple Dense network with one input and one output.

The one-to-many case has a single input and outputs a sequence. An example of such a network might be a network that can generate text tags from images [32]**,** containing short text descriptions of different aspects of the image. Such a network would be trained with image input and labeled sequences of text representing the image tags.

The many-to-one case is the reverse; it takes a sequence of tensors as input but outputs a single tensor. Examples of such networks would be a sentiment analysis network [33]which takes as input a block of text such as a movie review and outputs a single sentiment value.

The many-to-many use case comes in two flavors. The first one is more popular and is better known as the seq2seq model. In this model, a sequence is read in and produces a context vector representing the input sequence, which is used to generate the output sequence.

The topology has been used with great success in the field of machine translation, as well as problems that can be reframed as machine translation problems. Real life examples of the former can be found in [34]and an example of the latter is described in[35]**.**

The second many-to-many type has an output cell corresponding to each input cell. This kind of network is suited for use cases where there is a 1:1 correspondence between the input and output, such as time series. The major difference between this model and the seq2seq model is that the input does not have to be completely encoded before the decoding process begins.

3.9.1 Bidirectional RNNs

Some other RNN architectures include Bidirectional Recurrent Neural Networks (BRNN) and Encoder-Decoder Recurrent Neural Networks (EDRNN). BRNNS deviate from the conventional causal structures utilized by most other RNN frameworks. They make inferences from the current data point in a sequence relative to both past and future data points. This is particularly useful for decoding the meaning of sentences in which each word of the sentence is evaluated in the context of all the values of the sentence. Furthermore, many subtle linguistic dependencies can be extrapolated by considering a word's left and right neighbors.

It is also important to note that many words and phrases used in sentences can have different meanings depending upon the context of the sentence. A bidirectional view enables the model to have a higher probability of correctly extrapolating this context. In addition to NLP, BRNNs are also particularly useful in proteomics—identifying protein sequences from amino acid ordering—as well as in handwriting identification. EDRNN is another versatile RNN framework that allows the RNN to be trained to map an input sequence to variable length output sequences. This framework can be very useful to decode speech as well as to automate responses to speech (Figure 21).

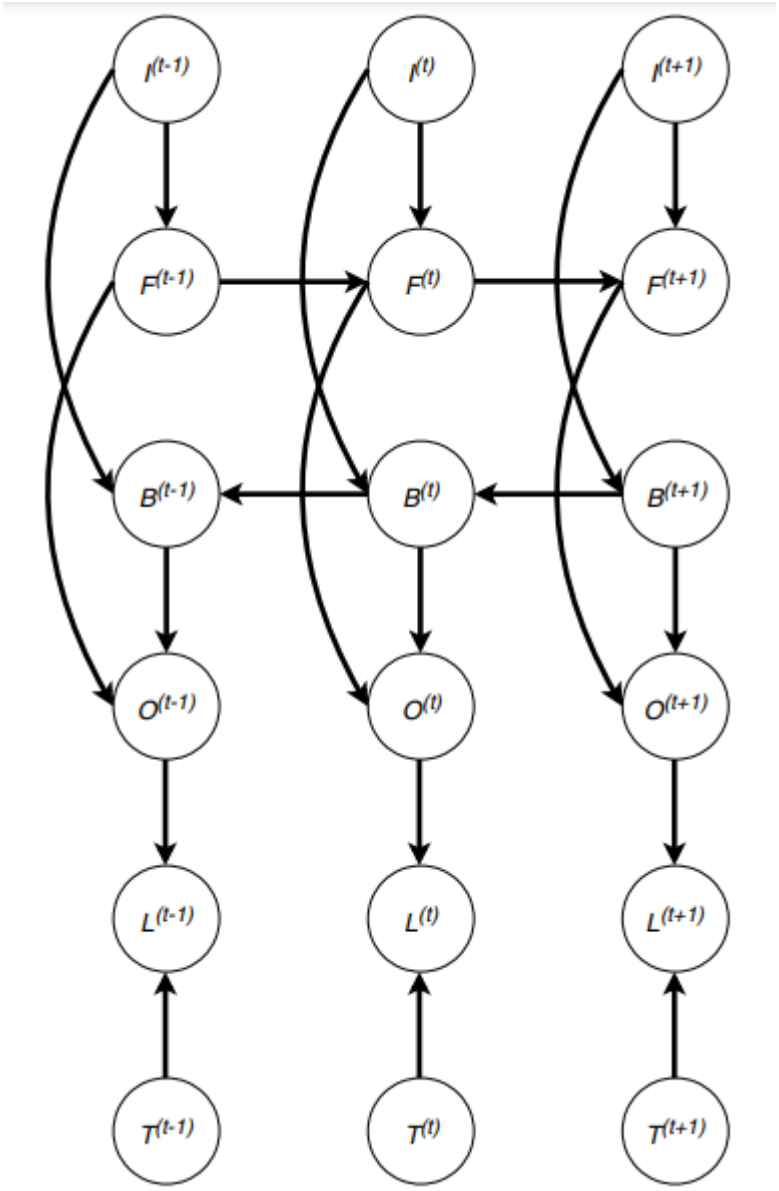

Figure 21 : Bidirectional RNN

We have seen how, at any given time step t, the output of the RNN is dependent on the outputs at all previous time steps. However, it is entirely possible that the output is also dependent on the future outputs as well. This is especially true for applications such as natural language processing where the attributes of the word or phrase we are trying to predict may be dependent on the context given by the entire enclosing sentence, not just the words that came before it.

This problem can be solved using a bidirectional LSTM, which are essentially two RNNs stacked on top of each other, one reading the input from left to right, and the other reading the input from the right to the left. The output at each time step will be based on the hidden state of both RNNs. Bidirectional RNNs allow the network to place equal emphasis on the beginning and end of the sequence, and typically results in performance improvements.

3.9.2 Stateful RNNs

RNNs can also be stateful, which means that they can maintain state across batches during training. That is, the hidden state computed for a batch of training data will be used as the initial hidden state for the next batch of training data. However, this needs to be explicitly set, since TensorFlow 2.0 (tf.keras) RNNs are stateless by default, and resets the state after each batch. Setting an RNN to be stateful means that it can build state across its training sequence and even maintain that state when doing predictions.

The benefits of using stateful RNNs are smaller network sizes and/or lower training times. The disadvantage is that we are now responsible for training the network with a batch size that reflects the periodicity of the data and resetting the state after each epoch. In addition, data should not be shuffled while training the network since the order in which the data is presented is relevant for stateful networks.

3.9.3 Recursive Neural Network

Recursive neural networks, not to be confused with RNNs, are a set of non-linear adaptive models which are used to process data of variable length. They are especially proficient in processing data structure inputs. Recursive networks feed the state of the network back into itself, in what can be viewed as a loop. They are primarily suited for image and sentence deconstruction. The architecture of recursive neural networks enables users to not only identify the constituents of input data but also to quantitatively determine the relationships between them [29].

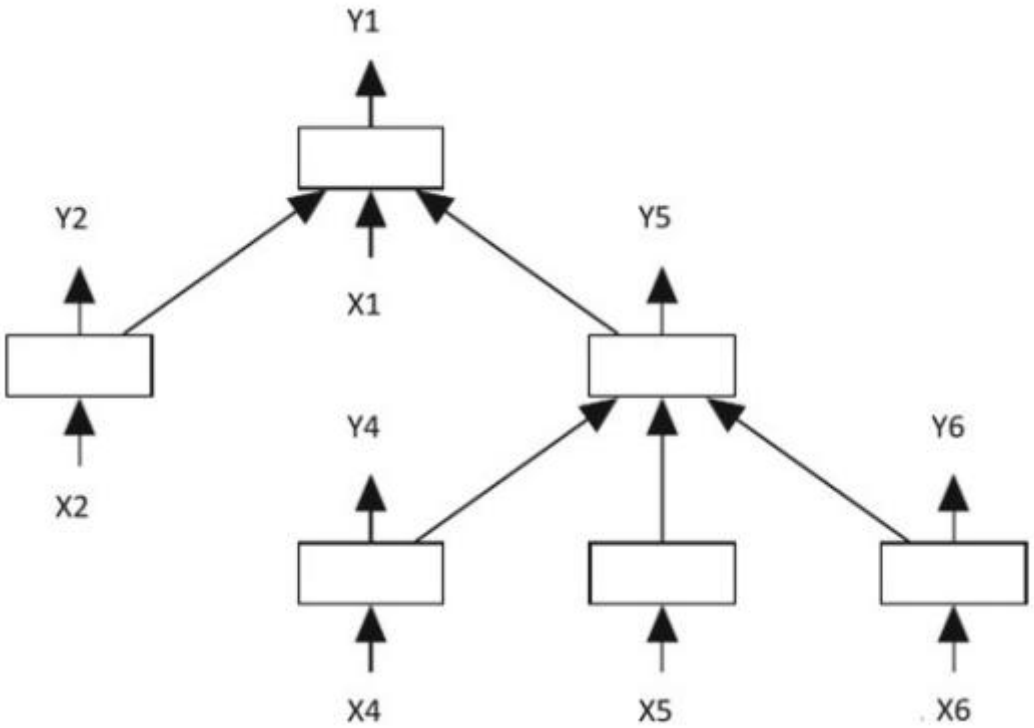

Figure22 : A condensed representation of Recursive Neural Network

This kind Deep Learning Architectures of deconstruction is made possible through a shared-weight matrix and binary tree structure—both of which enable the recursive neural network to extrapolate from varying length sequences of images and words. Furthermore, one

major advantage of recursive nets over recurrent nets is that for a sequence of the same length n the depth (measured as the number of compositions of nonlinear operations) can be drastically reduced from n to log(n) which enables efficient capturing of longterm dependencies [30]**.**. Recursive neural networks are generally known for having a bottom-up feed-forward method and top-down propagation method. Both mechanisms constitute the propagation through structure that is prevalent in most recursive networks (Figure 22).

Two of the most commonly used varieties of recursive networks include the semi-supervised recursive autoencoder and the supervised recursive neural tensor. The recursive autoencoder is used to deconstruct sentences for NLP applications whereas the recursive neural tensor is primarily used for computer vision applications. One drawback common to nearly all recursive neural networks is substantial computational overhead—moreso than recurrent neural networks. Recursive networks are reputed for processing exorbitant amounts of data often containing millions of parameters which results in long training times. As a result, optimization techniques are continuously developed for these architectures; furthermore, the evergrowing sophistication of processors and advancements made in parallel computing enable large-scale deployment of recursive neural networks.

3.10 Conclusion

The incorporation of deep learning models have allowed for large amounts of data to be correlated from multiple modalities. Built to emulate the structure of synaptic connections in the human brain, deep learning architectures are ubiquitously used for feature extraction, pattern analysis, and data abstraction. These models have been shown to perform better and faster than current state-of-the-art analysis techniques through supervised, unsupervised, and semi-supervised learning tasks.

There is a large range of applications that deep learning algorithms could be used for. They can be used to perform classification, data generation, and information understanding. For various fields from autonomous driving to bioinformatics, and medical image processing to assist the medical field in making accurate diagnoses [36]. For example, many CNN architectures are developed for image recognition tasks, including AlexNet and GoogLeNet. LSTM architectures have been designed for natural language processing since they have shown high performance in this application [37]**.** A CNN-based architecture called AtomNet[38] is designed for drug discovery and successfully predicted some novel molecules for Ebola virus Fig. 13. Deep and thorough researches has been done with using different deep learning architectures to analyze multimodality in medical imaging techniques [39].

Chapter IV: Designing an Author Detection Application by Style

Chapter IV: Designing an Author Detection Application by Style

4.1 Definition

As we have seen in the previous chapter all the theories and definition of RNN and the variants cell of RNN(LSTM, GRU ,Linear Classifier),in this chapter we will talk about the necessary basics technologies that we can not avoid , then we look to the diffrents models that we have used and how we trained them.

4.2 Technologies

4.2.1 TensorFlow

TensorFlow is a powerful open source software library developed by the Google Brain team for deep neural networks, the topic covered in this book. It was first made available under the Apache 2.0 License in November 2015 and has since grown rapidly; as of May 2019, its GitHub repository [40] has more than 51,000 commits, with roughly 1,830 contributors. This in itself provides a measure of the popularity of TensorFlow.

Google calls it "an open source software library for machine intelligence," but since there are so many other deep learning libraries like PyTorch [41], Caffe [42], and MxNet[43], what makes TensorFlow special? Most other deep learning libraries – like TensorFlow – have auto-differentiation (a useful mathematical tool used for optimization), many are open source platforms, most of them support the CPU/GPU option, have pretrained models, and support commonly used NN architectures like recurrent neural networks, convolutional neural networks, and deep belief networks, that’s why TensorFlow is the most popular among deep neural network researchers and engineers.

We install TensorFlow 2.0 as the following lines, Only CPU support:
pip install tensorflow==2.0.0-alpha0

With GPU support:
pip install tensorflow-gpu==2.0.0-alpha0

4.2.2 Keras

Keras is a beautiful API for composing building blocks to create and train deep learning models. Keras can be integrated with multiple deep learning engines including Google TensorFlow, Microsoft CNTK, Amazon MxNet, and Theano. Starting with TensorFlow 2.0, Keras has been adopted as the standard high-level API, largely simplifying coding and making programming more intuitive.

4.2.3 Word embedding

Wikipedia defines word embedding as the collective name for a set of language modeling and feature learning techniques in natural language processing (NLP) where words or phrases from a vocabulary are mapped to vectors of real numbers.

Deep learning models, like other machine learning models, typically don't work directly with text; the text needs to be converted to numbers instead. The process of converting text to numbers is a process called vectorization. An early technique for vectorizing words was one-hot encoding, a major problem with one-hot encoding is that it treats each word as completely independent from all the others, since similarity between any two words(measuredby the dot product of the two-word vectors) is always zero.

The following part of code shows how we set our tokenization and preprocessing using Keras

```
def tokenize_and_build_vocab(texts, vocab_size=None, lower=True):
    if vocab_size is None:
        tokenizer = tf.keras.preprocessing.text.Tokenizer(lower=lower)
    else:
        tokenizer = tf.keras.preprocessing.text.Tokenizer(
            num_words=vocab_size+1, oov_token="UNK", lower=lower)
    tokenizer.fit on texts(texts)
```

The next part of code shows how we set our indexation to the data means switch word to index using word-index

```
    tokenizer.word_index = {e:i for e, i in tokenizer.word_index.items()
        if i <= vocab_size+1 }
word2idx = tokenizer.word_index
idx2word = {v:k for k, v in word2idx.items()}
```

4.3 Models

4.3.1 Lstm

if the input is the sequence [c1 , c2 , …, cn], the output will be [c2 , c3 , …, cn+1]. We will train the network for 50 epochs, and at the end of every 10 epochs, we will generate a fixed size sequence of characters starting with a standard prefix

As always, we will first import the necessary libraries and set up some constants , a folder under the data folder, where we will save the weights of the model at the end of every 10 epochs:

```
def load_data(dir_name):
  file_name=os.path.join(dir_name,'data.csv')
  if (os.path.exists(file_name)):
    data=pd.read_csv(file_name,sep=',')
    text=[x for x in data["text"]]
    label=[x for x in data["author"]]
    return text,label
  else:
    return 0,0
```

Next we download and prepare the data for our network to consume, The tf.keras.utils.get_file() function will check to see whether the file is already downloaded to your local drive, and if not, it will download to a datasets folder under the location of the code. We also preprocess the input a little here, removing newline and byte order mark characters from the text.

```
def download_data():
  data=pd.read_csv('data.csv',sep=",")
  text=[x for x in data['text']]
  label=[x for x in data['author']]
  #
  return text,label
```

Now let's read our data in form of csv :

```
def read_data_csv(dir_name,data_name):
  file_name=os.path.join(dir_name,data_name)
  if (os.path.exists(file_name)):
    data=pd.read_csv(file_name,sep=',')
```

Next, we will create our vocabulary. In our case, our vocabulary contains unique characters, composed of uppercase and lowercase alphabets, numbers, and special characters. We also create some mapping dictionaries to convert each vocabulary character to a unique integer and vice versa. As noted earlier, the input and output of the network is a sequence of characters. However, the actual input and output of the network are sequences of integers, and we will use these mapping dictionaries to handle this conversion:

```
if vocab_size is None:
    tokenizer = tf.keras.preprocessing.text.Tokenizer(lower=lower)
else:
    tokenizer = tf.keras.preprocessing.text.Tokenizer(
        num_words=vocab_size+1, oov_token="UNK", lower=lower)
tokenizer.fit_on_texts(texts)
```

The next step is to use these mapping dictionaries to convert our character sequence input into an integer sequence, and then into a TensorFlow dataset. Each of our sequences is going to be 100 characters long, with the output being offset from the input by 1 character position. We first batch the dataset into slices of 101 characters, then apply the split_train_labels() function to every element of the dataset to create our sequences dataset,

which is a dataset of tuples of two elements, each element of the tuple being a vector of size 100 and type tf.int64. We then shuffle these sequences and then create batches of 64 tuples each for input to our network. Each element of the dataset is now a tuple consisting of a pair of matrices, each of size (64, 100) and type tf.int64:

```
def Build_dataset(text_int,label=None,text_size=None,batch_size=64):
```

```
dataset=tf.data.Dataset.from_tensor_slices((text_int,label))
dataset=dataset.shuffle(10000)
```

```
 dataset=tf.data.Dataset.from_tensor_slices(text_int)
```

We combine all of this into a one define function as following :

```
def Build_dataset(text_int,label=None,text_size=None,batch_size=64):
  #
  if text_size is not None:
    test_size=text_size // 3
    val_size=(text_size-test_size) // 10
    #
    dataset=tf.data.Dataset.from_tensor_slices((text_int,label))
    dataset=dataset.shuffle(10000)
    #
    dataset_test=dataset.take(test_size)
    dataset_val=dataset.skip(test_size).take(val_size)
    dataset_train=dataset.skip(test_size+val_size)
    #
    dataset_test=dataset_test.batch(batch_size)
    dataset_val=dataset_val.batch(batch_size)
    dataset_train=dataset_train.batch(batch_size)
    #
    return dataset_train,dataset_test,dataset_val

  else:
    #
    dataset=tf.data.Dataset.from_tensor_slices(text_int)
    #
    return dataset
```

We are now ready to define our network. As before, we define our network as a subclass of tf.keras.Model as shown next. The network is fairly simple; it takes as input a sequence of integers of size 100 (num_timesteps) and passes them through an Embedding layer so that each integer in the sequence is converted to a vector of size 256

(embedding_dim). So, assuming a batch size of 64, for our input sequence of size (64, 100), the output of the Embedding layer is a matrix of shape (64, 100, 256).

The next layer is the RNN layer with 100 time steps. The implementation of RNN chosen is a GRU. This GRU layer will take, at each of its time steps, a vector of size (256,) and output a vector of shape (1024,) (rnn_output_dim). Note also that the RNN is stateful, which means that the hidden state output from the previous training epoch will be used as input to the current epoch. The return_ sequences=True flag also indicates that the RNN will output at each of the time steps rather than an aggregate output at the last time steps.

Finally, each of the time steps will emit a vector of shape (1024,) into a Dense layer that outputs a vector of shape (90,) (vocab_size). The output from this layer will be a tensor of shape (64, 100, 90). Each position in the output vector corresponds to a character in our vocabulary, and the values correspond to the probability of that character occurring at that output position:

```
self.embedding=tf.keras.layers.Embedding(vocab_size,seq_max,mask_zero=True)
#self.droupout1D=tf.keras.layers.SpatialDropout1D(0.2)
self.bidirection=tf.keras.layers.Bidirectional(
    tf.keras.layers.LSTM(64,return_sequences=True)
)
```

```
self.lstm=tf.keras.layers.LSTM(32)
self.dense=tf.keras.layers.Dense(64,activation='relu')
self.droupout=tf.keras.layers.Dropout(0.5)
#self.hiden=tf.keras.layers.Dense(32,activation='relu')
self.out=tf.keras.layers.Dense(out_size,activation='sigmoid')
```

After we wrapped all of this into one function for call it later when we need it, here we made a function for call :

```
def call(self,x):
  #
  x=self.embedding(x)
  #x=self.droupout1D(x)
  x=self.bidirection(x)
  x=self.lstm(x)
  x=self.dense(x)
  x=self.droupout(x)
  #x=self.hiden(x)
  x=self.out(x)
  #
```

And now we print the vocabulary size :

```
print(vocab_size)
```

The result will be as show down :

```
seq_max=300
classes=3
batch_size=128
```

Next we define a loss function and compile our model. We will use the sparse categorical cross-entropy as our loss function because that is the standard loss function to use when our inputs and outputs are sequences of integers. For the optimizer, wewillchoose the Adam optimizer:

```
def loss(labels, predictions):
 return tf.losses.sparse_categorical_crossentropy(
 labels,
 predictions,
 from_logits=True
 )


model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

Finally, we are ready to run our training and evaluation loop. As mentioned earlier, we will train our network for 50 epochs, and at every 10 epoch intervals, we will try to generate some text with the model trained so far. Notice that in order to accommodate a single string prefix, we save the weights after every 10 epochs and build a separate generative model with these weights but with an input shape with a batch size of 1. Hereis the code to do this:

```
num_epochs = 50
for i in range(num_epochs // 10):
 model.fit(
 dataset.repeat(),
 epochs=10,
 steps_per_epoch=steps_per_epoch
```

```
# callbacks=[checkpoint_callback, tensorboard_callback]
 )
 checkpoint_file = os.path.join(
 CHECKPOINT_DIR, "model_epoch_{:d}".format(i+1))
 model.save_weights(checkpoint_file)
 # create generative model using the trained model so far
 gen_model = CharGenModel(vocab_size, seq_length, embedding_dim,
 rnn_output_dim)
 gen_model.load_weights(checkpoint_file)
 gen_model.build(input_shape=(1, seq_length))
 print("after epoch: {:d}".format(i+1)*10)
 print(generate_text(gen_model, "Alice ", char2idx, idx2char))
 print("---")
```

And the output will show like the following :

```
Model: "author_lstm__model_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_1 (Embedding)      multiple                  7783200

 bidirectional_1 (Bidirectio  multiple                  186880
 nal)

 lstm_3 (LSTM)                multiple                  20608

 dense_3 (Dense)              multiple                  2112

 dropout (Dropout)            multiple                  0

 dense_4 (Dense)              multiple                  195

=================================================================
```

However, after about 10 epochs of training , it looks like this :

```
Epoch 1/10
225/225 [==============================] - 487s 2s/step - loss: 0.5565 - accuracy: 0.7699 - val_loss: 0.1883 - val_accuracy: 0.9445
Epoch 2/10
225/225 [==============================] - 474s 2s/step - loss: 0.1613 - accuracy: 0.9477 - val_loss: 0.0594 - val_accuracy: 0.9815
Epoch 3/10
225/225 [==============================] - 471s 2s/step - loss: 0.0718 - accuracy: 0.9772 - val_loss: 0.0355 - val_accuracy: 0.9873
Epoch 4/10
225/225 [==============================] - 474s 2s/step - loss: 0.0414 - accuracy: 0.9873 - val_loss: 0.0200 - val_accuracy: 0.9948
Epoch 5/10
225/225 [==============================] - 474s 2s/step - loss: 0.0344 - accuracy: 0.9892 - val_loss: 0.0164 - val_accuracy: 0.9931
Epoch 6/10
225/225 [==============================] - 477s 2s/step - loss: 0.0271 - accuracy: 0.9920 - val_loss: 0.0269 - val_accuracy: 0.9907
Epoch 7/10
225/225 [==============================] - 473s 2s/step - loss: 0.0244 - accuracy: 0.9923 - val_loss: 0.0092 - val_accuracy: 0.9965
Epoch 8/10
225/225 [==============================] - 484s 2s/step - loss: 0.0153 - accuracy: 0.9953 - val_loss: 0.0125 - val_accuracy: 0.9931
Epoch 9/10
225/225 [==============================] - 482s 2s/step - loss: 0.0162 - accuracy: 0.9952 - val_loss: 0.0337 - val_accuracy: 0.9890
Epoch 10/10
225/225 [==============================] - 461s 2s/step - loss: 0.0172 - accuracy: 0.9947 - val_loss: 0.0047 - val_accuracy: 0.9983
```

4.3.2 GRU

The competition dataset contains text from works of fiction written by spooky authors of the public domain: Edgar Allan Poe, HP Lovecraft and Mary Shelley. The data was prepared by chunking larger texts into sentences using CoreNLP'sMaxEnt sentence tokenizer, the objective is to accurately identify the author of the sentences in the test set.

We are ready to build our network. As usual, we will start by importing the necessary packages:

```
import numpy as np
import os
import shutil
import tensorflow as tf
import nltk
import pandas as pd
```

Now we have to pass throught our preprocessing and tokenization for the data , before that we need to upload and read the data first , the following parts of code shown what we are talking about :

```
def clean_logs(data_dir):
    logs_dir = os.path.join(data_dir, "logs")
    shutil.rmtree(logs_dir, ignore_errors=True)
    return logs_dir
```

Let’s upload the data :

```
def download_data():
  data=pd.read_csv('data.csv',sep=",")
  text=[x for x in data['text']]
  label=[x for x in data['author']]
  #
  return text,label
```

There are 3194 sentences in our dataset. We will then use the TensorFlow (tf.keras) tokenizer to tokenize the sentences and create a list of sentence tokens. We reuse the same infrastructure to tokenize the parts of speech, although we could have simply split on spaces. Each input record to the network is currently a sequence of text tokens, but they need to be a sequence of integers. During the tokenizing process, the Tokenizer also maintains the tokens in the vocabulary, from which we can build mappings from token to integer and back.

```
def tokenize_and_build_vocab(texts, vocab_size=None, lower=True):
    if vocab_size is None:
        tokenizer = tf.keras.preprocessing.text.Tokenizer(lower=lower)
    else:
        tokenizer = tf.keras.preprocessing.text.Tokenizer(
            num_words=vocab_size+1, oov_token="UNK", lower=lower)
    tokenizer.fit_on_texts(texts)
    if vocab_size is not None:
        # additional workaround, see issue 8092
        # https://github.com/keras-team/keras/issues/8092
        tokenizer.word_index = {e:i for e, i in tokenizer.word_index.items()
            if i <= vocab_size+1 }
    word2idx = tokenizer.word_index
    idx2word = {v:k for k, v in word2idx.items()}
    return word2idx, idx2word, tokenizer
```

We see that we could probably get away with setting the sentence length to around 100, and have a few truncated sentences as a result. Sentences shorter than our selected length will be padded at the end. Because our dataset is small, we prefer to use as much of it as possible, so we end up choosing the maximum length.

The next step is to create the dataset from our inputs. First, we have to convert our sequence of tokens in our input and output sequences to sequences of integers. Second, we have to pad shorter sequences to the maximum length of 271. Notice that we do an additional operation on the POS tag sequences after padding, rather than keep it as a sequence of integers, we convert it to a sequence of one-hot encodings using the to_categorical() function. TensorFlow 2.0 does provide loss functions to handle outputs as a sequence of integers, but we want to keep our code as simple as possible, sowe opt to do the conversion ourselves. Finally, we use the from_tensor_slices() function to create our dataset, shuffle it, and split it up into training, validation, and test sets:

```
        ytrue = tf.keras.backend.argmax(ytrue, axis=-1)
        ypred = tf.keras.backend.argmax(ypred, axis=-1)

        mask = tf.keras.backend.cast(
            tf.keras.backend.not_equal(ypred, 0), tf.int32)
        matches = tf.keras.backend.cast(
            tf.keras.backend.equal(ytrue, ypred), tf.int32) * mask
        numer = tf.keras.backend.sum(matches)
        denom = tf.keras.backend.maximum(tf.keras.backend.sum(mask), 1)
        accuracy =  numer / denom
```

```
# split into training, validation, and test datasets
dataset = dataset.shuffle(10000)
test_size = len(sents) // 3
val_size = (len(sents) - test_size) // 10
test_dataset = dataset.take(test_size)
val_dataset = dataset.skip(test_size).take(val_size)
train_dataset = dataset.skip(test_size + val_size)
```

```
# create batches
batch_size = BATCH_SIZE
train_dataset = train_dataset.batch(batch_size)
val_dataset = val_dataset.batch(batch_size)
test_dataset = test_dataset.batch(batch_size)
```

Next, we will define our model and instantiate it. Our model is a sequential model consisting of an embedding layer, a dropout layer, a bidirectional GRU layer, a dense layer, and a softmax activation layer. The input is a batch of integer sequences, with shape (batch_size, max_seqlen). When passed through the embedding layer, each integer in the sequence is converted to a vector of size (embedding_dim), so now the shape of our tensor is (batch_size, max_seqlen, embedding_dim). Each of these vectors are passed to corresponding time steps of a bidirectional GRU with an output dimension of 256. Because the GRU is bidirectional, this is equivalent to stacking one GRU on top of the other, so the tensor that comes out of the bidirectional GRU has the dimension (batch_size, max_seqlen, 2*rnn_output_dimension).

Each timestep tensor of shape (batch_size, 1, 2*rnn_output_dimension) is fed into a dense layer, which converts each time step to a vector of the same size as the target vocabulary, that is, (batch_size, number_of_timesteps, output_vocab_size). Each time step represents a probability distribution of output tokens, so the final softmax layer is applied to each time step to return a sequence of output.

Finally, we declare the model with some parameters, then compile it with the Adam optimizer, the categorical cross-entropy loss function, and accuracy as the metric:

```
class POSTaggingModel(tf.keras.Model):
    def __init__(self, source_vocab_size, target_vocab_size,
            embedding_dim, max_seqlen, rnn_output_dim, **kwargs):
        super(POSTaggingModel, self).__init__(**kwargs)
        self.embed = tf.keras.layers.Embedding(
            source_vocab_size, embedding_dim, input_length=max_seqlen)
        self.dropout = tf.keras.layers.SpatialDropout1D(0.2)
        self.rnn = tf.keras.layers.Bidirectional(
            tf.keras.layers.GRU(rnn_output_dim, return_sequences=True))
        self.dense = tf.keras.layers.TimeDistributed(
            tf.keras.layers.Dense(target_vocab_size))
        self.activation = tf.keras.layers.Activation("softmax")
```

```
  def call(self, x):
      x = self.embed(x)
      x = self.dropout(x)
      x = self.rnn(x)
      x = self.dense(x)
      x = self.activation(x)
      return x
```

```
NUM_PAIRS = None
EMBEDDING_DIM = 128
RNN_OUTPUT_DIM = 256
BATCH_SIZE = 128
NUM_EPOCHS = 50
```

Observant readers might have noticed an additional masked_accuracy() metric next to the accuracy metric in the preceding code snippet. Because of the padding, there are a lot of zeros on both the label and prediction, as a result of which the accuracy numbers are very optimistic. In fact, the validation accuracy reported at the end of the very first epoch is 0.9116. however the quality generated are very poor .

Perhaps the best approach is to replace the current loss function with one that ignores matches where both numbers are zero; however, a simpler approach is to build a stricter metric and use that to judge when to stop the training. Accordingly, we build a new accuracy function masked_accuracy() whose code is shown as follows:

```
def masked_accuracy():
    def masked_accuracy_fn(ytrue, ypred):
        ytrue = tf.keras.backend.argmax(ytrue, axis=-1)
        ypred = tf.keras.backend.argmax(ypred, axis=-1)

        mask = tf.keras.backend.cast(
            tf.keras.backend.not_equal(ypred, 0), tf.int32)
        matches = tf.keras.backend.cast(
            tf.keras.backend.equal(ytrue, ypred), tf.int32) * mask
        numer = tf.keras.backend.sum(matches)
        denom = tf.keras.backend.maximum(tf.keras.backend.sum(mask), 1)
        accuracy =  numer / denom
        return accuracy

    return masked_accuracy_fn
```

We are now ready to train our model. As usual, we set up the model checkpoint and TensorBoard callbacks, and then call the fit() convenience method on the model to train the model with a batch size and epochs:

```
# train
num_epochs = NUM_EPOCHS

best_model_file = os.path.join(data_dir, "best_model.h5")
checkpoint = tf.keras.callbacks.ModelCheckpoint(
    best_model_file,
    save_weights_only=True,
    save_best_only=True)
tensorboard = tf.keras.callbacks.TensorBoard(log_dir=logs_dir)
history = model.fit(train_dataset,
    epochs=num_epochs,
    validation_data=val_dataset,
    callbacks=[checkpoint, tensorboard])
```

A truncated output of the training is shown as follows. As you can see, the masked_ accuracy and *val_masked_accuracy* numbers seem more conservative than the accuracy and *val_accuracy* numbers. This is because the masked versions do not consider the sequence positions where the input is a PAD character:

```
Epoch 1/50
184/184 [==============================] - 480s 3s/step - loss: 0.0566 - accuracy: 0.9911 - masked_accuracy_fn: 0.1302 - val_loss: 0.0064 - val_accuracy: 0.996
Epoch 2/50
184/184 [==============================] - 474s 3s/step - loss: 0.0051 - accuracy: 0.9977 - masked_accuracy_fn: 0.4052 - val_loss: 0.0043 - val_accuracy: 0.997
Epoch 3/50
184/184 [==============================] - 476s 3s/step - loss: 0.0041 - accuracy: 0.9980 - masked_accuracy_fn: 0.4476 - val_loss: 0.0038 - val_accuracy: 0.998
Epoch 4/50
184/184 [==============================] - 481s 3s/step - loss: 0.0038 - accuracy: 0.9982 - masked_accuracy_fn: 0.5057 - val_loss: 0.0037 - val_accuracy: 0.998
Epoch 5/50
184/184 [==============================] - 482s 3s/step - loss: 0.0036 - accuracy: 0.9983 - masked_accuracy_fn: 0.5517 - val_loss: 0.0034 - val_accuracy: 0.998
Epoch 6/50
184/184 [==============================] - 482s 3s/step - loss: 0.0028 - accuracy: 0.9988 - masked_accuracy_fn: 0.6720 - val_loss: 0.0022 - val_accuracy: 0.999
Epoch 7/50
184/184 [==============================] - 484s 3s/step - loss: 0.0018 - accuracy: 0.9993 - masked_accuracy_fn: 0.8102 - val_loss: 0.0014 - val_accuracy: 0.999
Epoch 8/50
184/184 [==============================] - 484s 3s/step - loss: 0.0012 - accuracy: 0.9996 - masked_accuracy_fn: 0.8857 - val_loss: 9.3815e-04 - val_accuracy: 0
Epoch 9/50
184/184 [==============================] - 479s 3s/step - loss: 9.0284e-04 - accuracy: 0.9997 - masked_accuracy_fn: 0.9151 - val_loss: 6.9858e-04 - val_accurac
Epoch 10/50
184/184 [==============================] - 483s 3s/step - loss: 6.5518e-04 - accuracy: 0.9998 - masked_accuracy_fn: 0.9424 - val_loss: 4.7349e-04 - val_accurac
Epoch 11/50
184/184 [==============================] - 479s 3s/step - loss: 5.2431e-04 - accuracy: 0.9998 - masked_accuracy_fn: 0.9544 - val_loss: 3.9755e-04 - val_accurac
Epoch 12/50
```

Finaly the evaluation with test set will show like this :

```
102/102 [==============================] - 88s 849ms/step - loss: 7.0076e-06 - accuracy: 1.0000 - masked_accuracy_fn: 0.9995
test loss: 0.000, test accuracy: 1.000, masked test accuracy: 1.000
```

Here are some examples generated for some random sentences in the test set, shown together with the corresponding ground truth sentences. As you can see :

```
text : the back of the hand is upwards
prediction :  ['EAP']
label :  ['EAP']
text : the drawers of a bureau which stood in one corner were open and had been apparently UNK although many articles still remained in them
prediction :  ['EAP']
label :  ['EAP']
text : evening approached and i beheld the sun set
prediction :  ['MWS']
label :  ['MWS']
text : UNK ivory image but the sight of an UNK pistol calmed them
prediction :  ['HPL']
label :  ['HPL']
text : the inhabitants of our side of the moon have evidently no darkness at all so there can be nothing of the extremes mentioned
prediction :  ['EAP']
label :  ['EAP']
text : his features had perhaps been noble once but were now UNK with the ghastly effects of terrible dissipation
prediction :  ['HPL']
label :  ['HPL']
```

4.3.3 Linear Classifier

we will use the Estimator classifier available in TensorFlow estimator to determine wich text is belong to an author in our data , The classifier Estimator takes in the features and the labels. It converts them to onehot encoded vectors, that is, we have 10 bits representing the output. Each bit can have a value of either 0 or 1, and being one-hot, let’s build our model :

1. The first step is as always importing the modules needed:

```
# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)
```

2. We load the data :

```
def load_data(dir_name):
  file_name=os.path.join(dir_name,'data.csv')
  if (os.path.exists(file_name)):
    data=pd.read_csv(file_name,sep=',')
    text=[x for x in data['text']]
    label=[x for x in data['author']]
    return text,label
  else:
    return 0,0
```

3. Next, we preprocess the data:

```
def my_label_int(labels):
  ints=[]
  for x in labels:
    if x=="EAP":
      ints.append(0)
    elif x=="MWS":
      ints.append(1)
    elif x=="HPL":
      ints.append(2)
  #
  ints=np.array(ints)
  #
  return ints
```

4.Now, we move to the preprocessing for tokenization and indexation:

```
def build_tokenizer(text,vocab_size=None,lowel=True):
  if vocab_size is None:
    tokenizer=tf.keras.preprocessing.text.Tokenizer()
  else:
    tokenizer=tf.keras.preprocessing.text.Tokenizer(
        wum_words=vocab_size,oov_token="UNK"
    )
  tokenizer.fit_on_texts(text)
  if vocab_size is not None:
    tokenizer.word_index={e:i for e,i in tokenizer.word_index if i <= vocab_size+1}
  word2idx=tokenizer.word_index
  idx2word={i:e for e,i in word2idx.items()}

  return word2idx,idx2word,tokenizer
```

5. Use the feature_column module of TensorFlow to define numeric features of size:

```
def build_feature_columns(shape_size):
  fc=[tf.feature_column.numeric_column('x',shape=shape_size)]
  #
  return fc
```

6. Create the logistic regression estimator. We use a simple LinearClassifier. We encourage you to experiment with DNNClassifier as well:

```
linearClassifier=tf.estimator.LinearClassifier(
    feature_columns,
    model_dir='',
    n_classes=3
)
```

7. Let us also build an input_function to feed the estimator:

```
def input_fn(data,label,train):
  if train ==1:
    text_input_fn=tf.compat.v1.estimator.inputs.numpy_input_fn(
        x={"x":data},
        y=label,
        num_epochs=None,
        shuffle=True
    )
  else:
    text_input_fn=tf.compat.v1.estimator.inputs.numpy_input_fn(
        x={"x":data},
        y=label,
        num_epochs=1,
        shuffle=False
    )
  #
  return text_input_fn
```

8. Let's now train the classifier:

```
# train the estimator
linearClassifier.train(
    train_input_fn,
    steps=10
)
```

9. Next, we create the input function for validation data:

```
val_input_fn = tf.compat.v1.estimator.inputs.numpy_input_fn(
 x={"x": eval_data},
 y=eval_labels,
 num_epochs=1,
 shuffle=False)
```

10. Let us evaluate the trained Linear Classifier on the validation data:

```
# input the test
out_input_fn=input_fn(test_data,test_label,0)
# evaluate the estimator
linearClassifier.evaluate(out_input_fn)
```

We get an accuracy of 36.31% after 10 time steps note that since we have specified the time steps, the model trains for the specified steps and logs the value after 10 steps (the number of steps specified). Now if we run train again, then it will start from the state it had at the 10th time step. The steps will go on increasing with an increment of the number of steps mentioned.

The following is the graph of the preceding model (Figure 23):

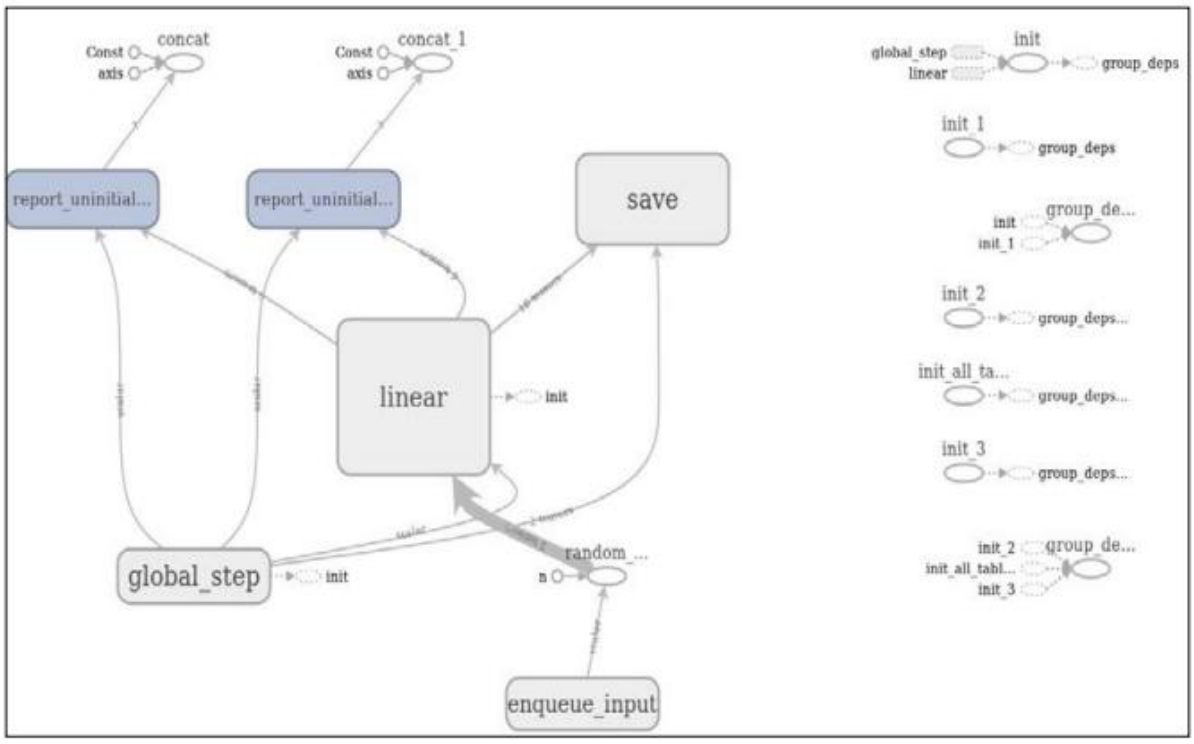

Figure 23 : TensorBoard graph of the generated model

From TensorBoard we can also visualize the change in accuracy and average loss as the linear classifier learned in steps of ten (Figure 24):

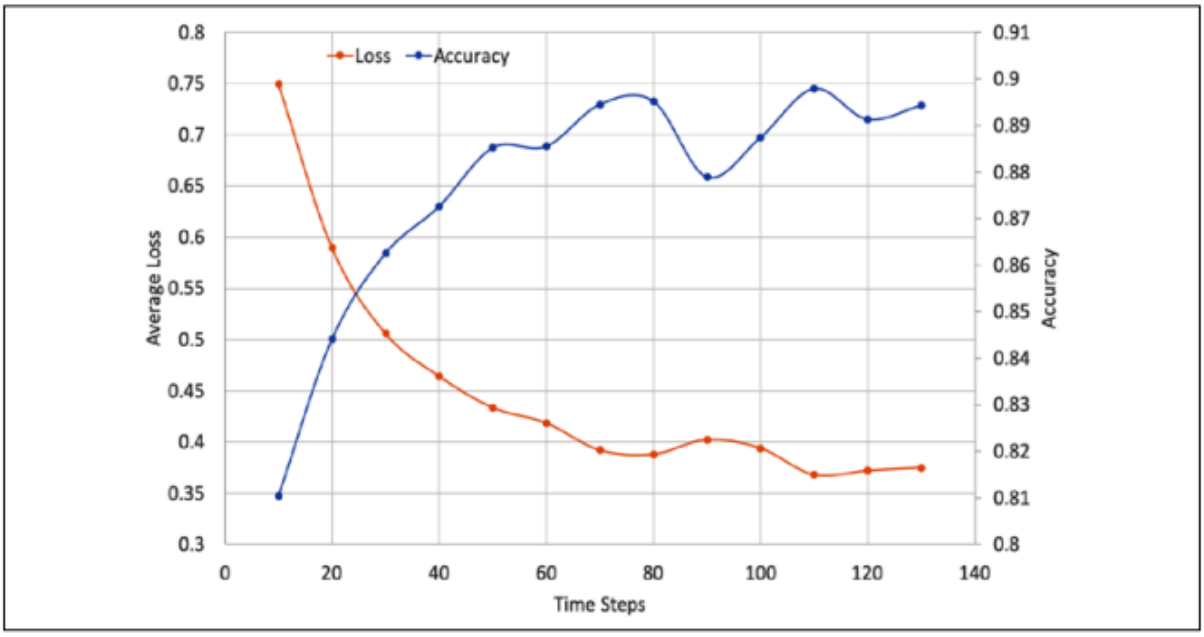

Figure 24 : Accuracy and average loss, visualized

4.3 Conclusion

In this chapter, we have seen the concepts behind distributional representations of words and its various implementations, starting from static word embeddings such as Word2Vec.

We have then looked at improvements to the basic idea, such as subwordembeddings, sentence embeddings that capture the context of the word in the sentence, as well as the use of entire language models for generating embeddings. While the language model-based embeddings are achieving state of the art results nowadays, there are still plenty of applications where more traditional approaches yield very good results, so it is important to know them all and understand the tradeoffs.

We have also seen the two models LSTM and GRU step by step how to create the model and how to load and read the data the way it seems better for building a right model as it has , we have seen also how to preprocessing the data and split it for training and testing , the results that we have seen is different from these three models that we have used.

Chapter V: Implementation and Testing

Chapter V: Implementation and Testing

5.1 Introduction

We have defined our approach to the identification of an author by his style abstract, as well as all the concepts that follows it, and we are nowready to test our application. This chapter will describes the hardware and software environment inwhich we worked on, as well as the test of the application on which we have dealt with it and themeasurementthat we have used.

5.2 hardware environment

Our application has been performed on a machine that has thefollowing characteristics:

- **Brand :** HP ProBook
- **Processor :** Intel(R) Core(TM) i5-2430M CPU @ 2.4GHz 2.4GHz
- **Graphic Card :** Internal Intel(R) HD Graphics 3000
- **Memory :** 8.00 Go
- **Operating System :** Windows 10 Professionel

5.3 Software environment

5.3.1 Python

Python23 is an open source programming language created by Guido programmer van Rossum in 1991. It appeared at the time as a way to automate the elements the more annoying than writing scripts or quickly making prototype applications.In recent years, this programming language has become one of the most widely used in the field of software development, infrastructure management and data analysis.This is a driving force behind the Big Data explosion.

There are two versions of Python: Python 2 and Python 3. The differences between these two versions are multiple. Python 2.x is the older version, which will continue to be supported and receive official updates until 2020. Afterthat date, itwillprobably continue to existunofficially.

Python 3.x is the current version of the language. It brings many new and very useful features, such as better competition control and a more efficient interpreter. However, the adoption of Python 3 has long been slowed down by the lack of supported third-party libraries. Many of them were only compatible with Python 2, which made the transition complicated. However, this problem is now practically solved and there are few valid reasons to continue using Python 2.

The Python language owes its popularity to several advantages that benefit beginners as well as experts, including:

- It is easy to learn and use. Its features are few, this which allows programs to be created quickly and with little effort. In addition, itssyntax is designed to be readable and direct;
- It works on all major operating systems and platformscomputer. Moreover, even if it is clearly not the fastest language,it compensates for its slowness by its versatility;
- Although it is mainly used for scripting and automation, this languageis also used to create professional quality software. Whether it isapplications or web services, Python is used by a large number ofdevelopers to create software.

5.3.2 Google Colab

Google Colab or Colaboratory is a cloud service, offered by Google (free), based on Jupyter Notebook and intended for training and search in machine learning. This platform allows you to train machine learning models directly in the cloud. So without having to install anything on our computer except a browser. Cool, right? Before presenting this magnificent service, we will recall what is a Jupyter Notebook.

Google Colabfile,has three different type of executionthat he must select :

1. None
2. TPU
3. GPU

5.3.3 TensorFlow

TensorFlow is a powerful open source software library developed by the Google Brain team for deep neural networks, the topic covered in this book. It was first made available under the Apache 2.0 License in November 2015 and has since grown rapidly; as of May 2019, its GitHub repository (https://github.com/ tensorflow/tensorflow) has more than 51,000 commits, with roughly 1,830 contributors. This in itself provides a measure of the popularity of TensorFlow.

5.3.4 NumPy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more [https://numpy.org/doc/stable/].

5.3.5 Nltk

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active discussion forum.

Natural Language Processing with Python provides a practical introduction to programming for language processing. Written by the creators of NLTK, it guides the reader through the fundamentals of writing Python programs, working with corpora, categorizing text, analyzing linguistic structure, and more[https://www.nltk.org/].

5.3.6 Matplotlib.pyplot

Pyplot is a Matplotlib module with several simple functions to add elements such as lines, images or texts at the axes of a graphic. Soundinterface is very comfortable, and that's why this module is very used. Matplotlibisfreely distributed under a BSD-style license.

5.3.7 Flask

Flask is a lightweight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It began as a simple wrapper around Werkzeug and Jinja and has become one of the most popular Python web application frameworks.

Flask offers suggestions, but doesn't enforce any dependencies or project layout. It is up to the developer to choose the tools and libraries they want to use. There are many extensions provided by the community that make adding new functionality easy[https://palletsprojects.com/p/flask/].

5.4 Data Set

5.4.1 Description

The competition dataset contains text from works of fiction written by spooky authors of the public domain: Edgar Allan Poe, HP Lovecraft and Mary Shelley.every one of them has three books. The data was prepared by chunking larger texts into sentences using CoreNLP'sMaxEnt sentence tokenizer, the objective is to accurately identify the author of the sentences in the test set.

5.4.2 File Descriptions

- Train.csv : the training set.
- Test.csv : the test set.
- Sample_submission.csv : a sample submission file in the correct form.

5.4.3 Data fields

- Id : a unique identifier for each sentence.
- Text : some text written by one of the authors.
- Author : the author of the sentence (**EAP**: Edgar Allan Poe, **HPL**: HP Lovecraft; **MWS**: Mary Wollstonecraft Shelley).

5.5 The graphical interface of the system

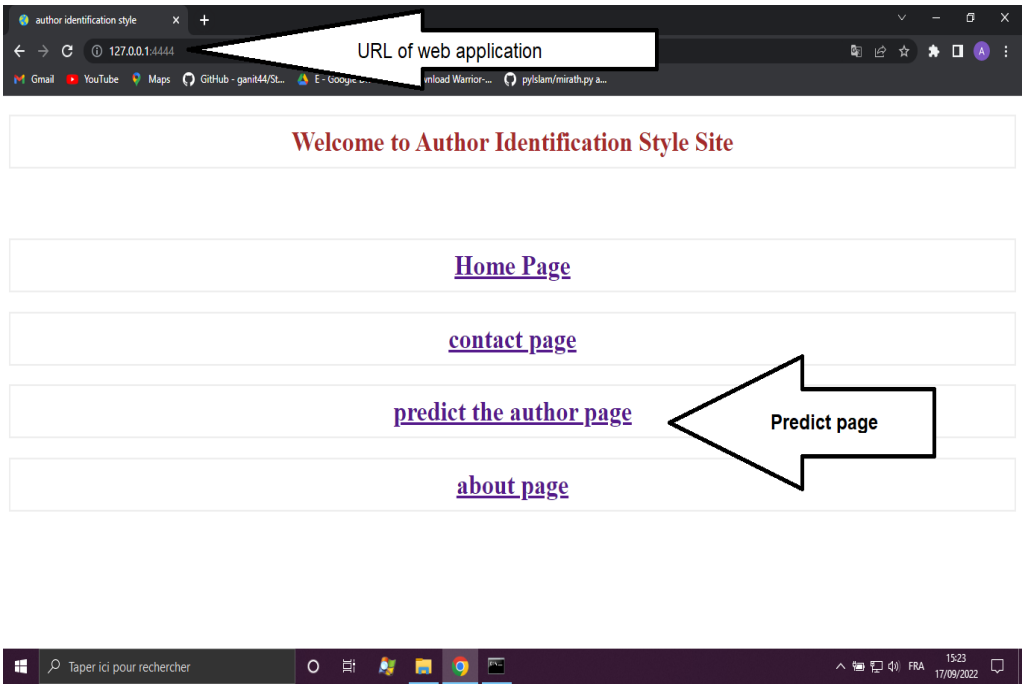

Figure 25 : Home Page of Web application

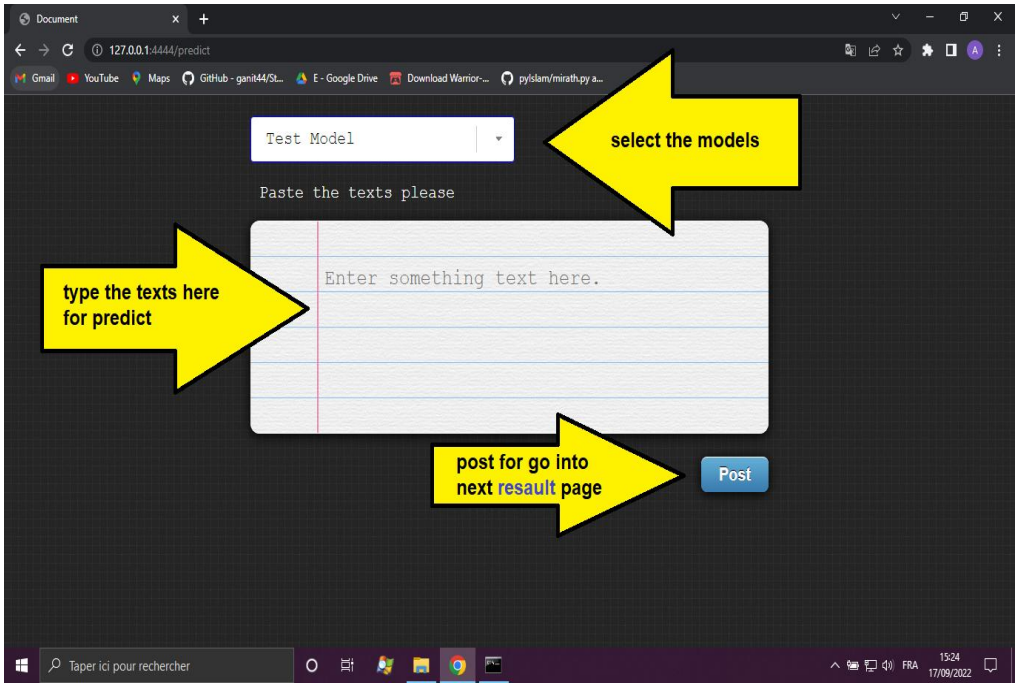

Figure 26 : graphic interface of app

5.5.1 System User Guide

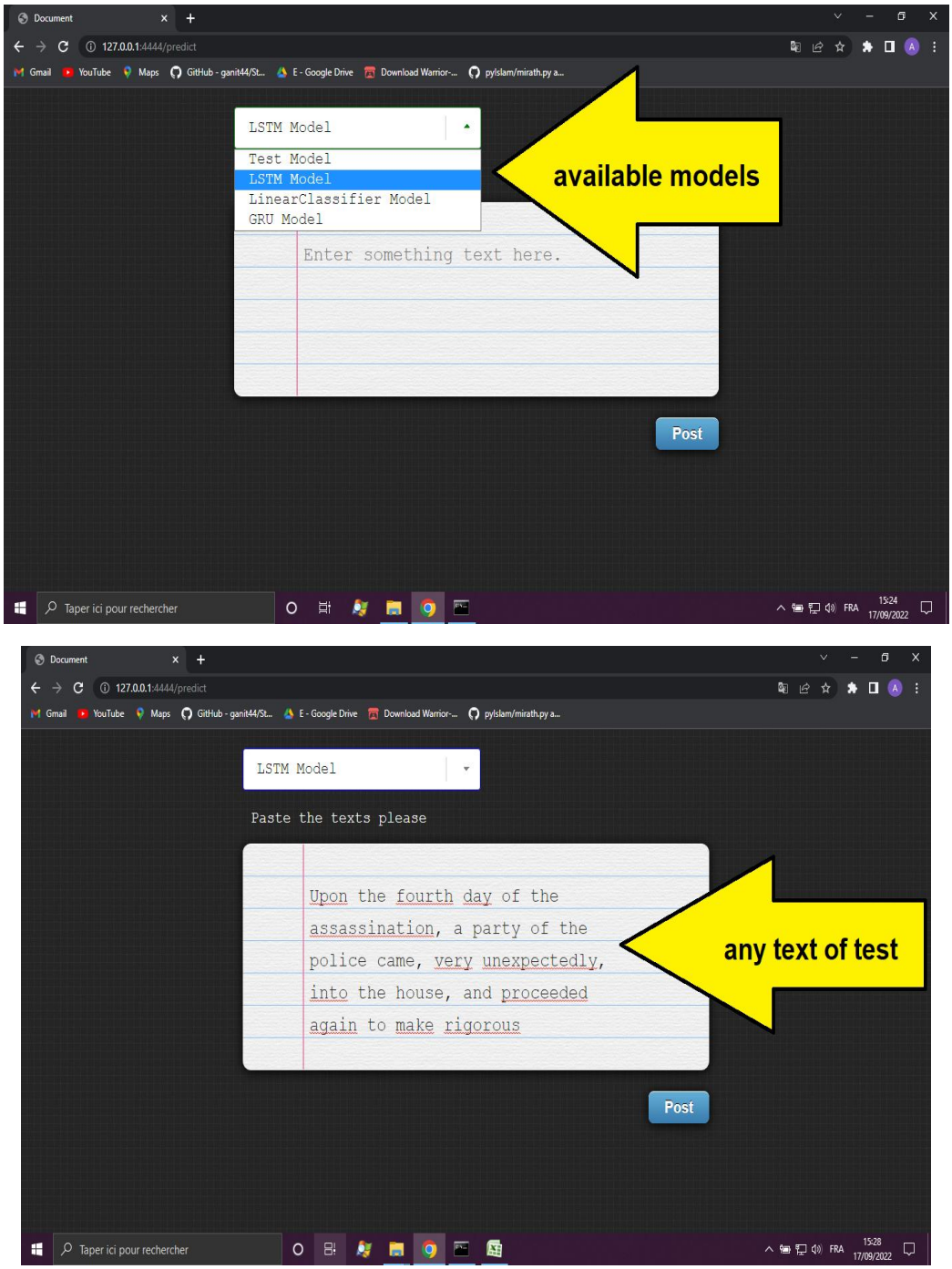

Figure 27 : select the model and text input

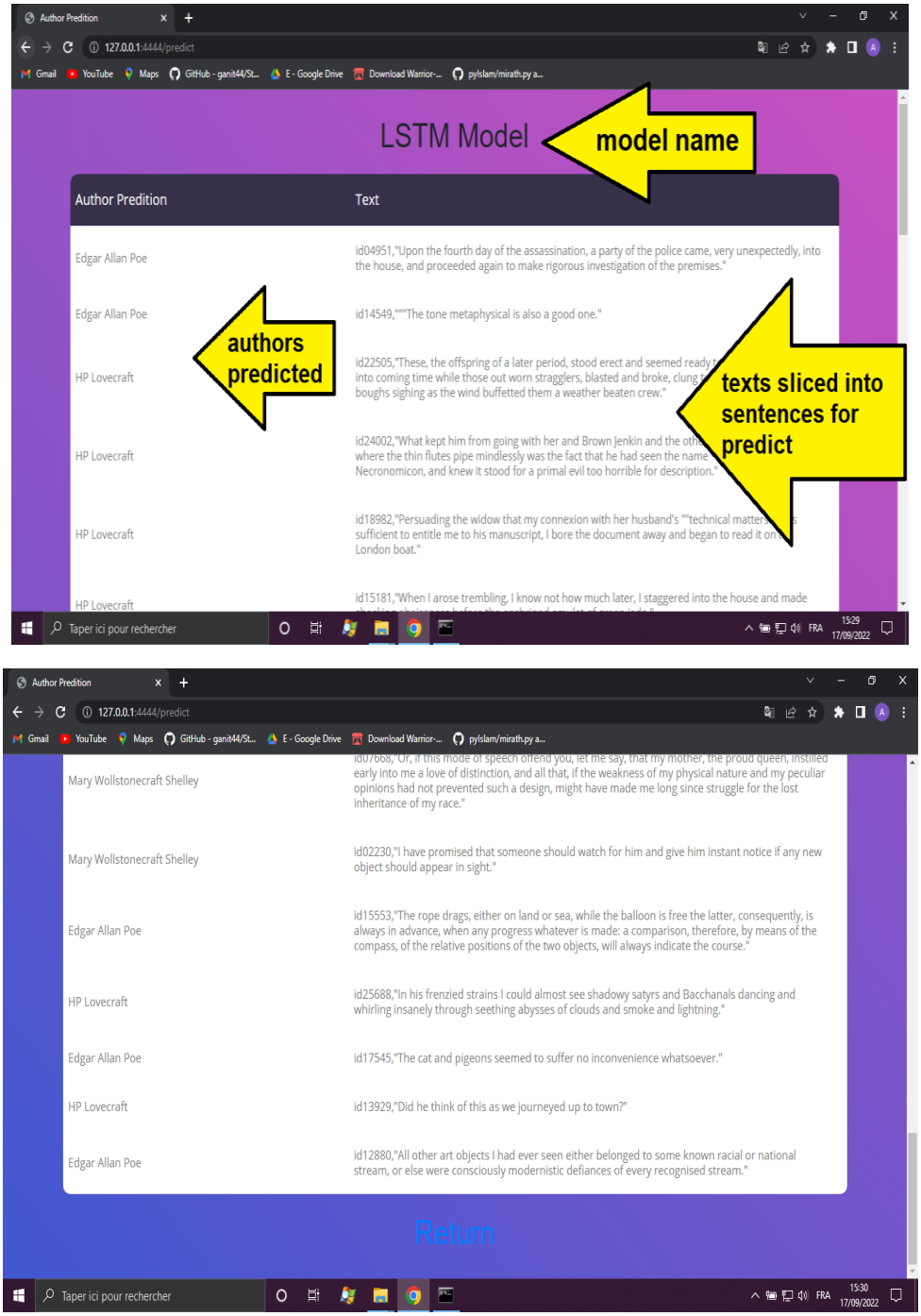

Figure 28: predict result

5.6 Experimentations

5.6.1 Results using LSTM model

Here we show all the results related to LSTM model using the accuracy and loss of train besides of the validation in terms of epoch.

5.6.1.1 Accuracy,loss of train,validation in terms of epoch

Accuracy and loss results are shown in (Figure 29) which shows accuracy. In terms of the epoch, and in the (Figure 30) it shows the percentage of loss in terms of the epoch, then the (Figure 31) shows the convincing results of agreement:

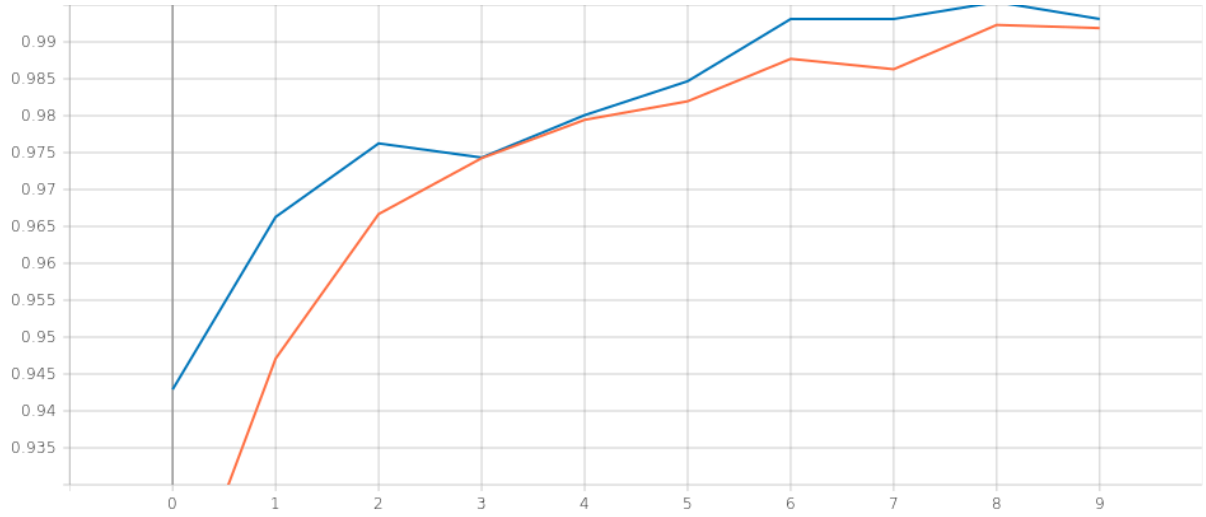

Figure 29: Accuracy of train and validation in terms of epoch

The red graph represents train accuracy, and the blue one represents the validation accuracy in terms of epoch payments.

We notice in the (Figure 29) that at the first epoch, the accuracy rate was above 90%, and this indicates that the training was successful , means it train well. When the ninth batch of epoch was reached, it reached 99%, and the result was perfect.

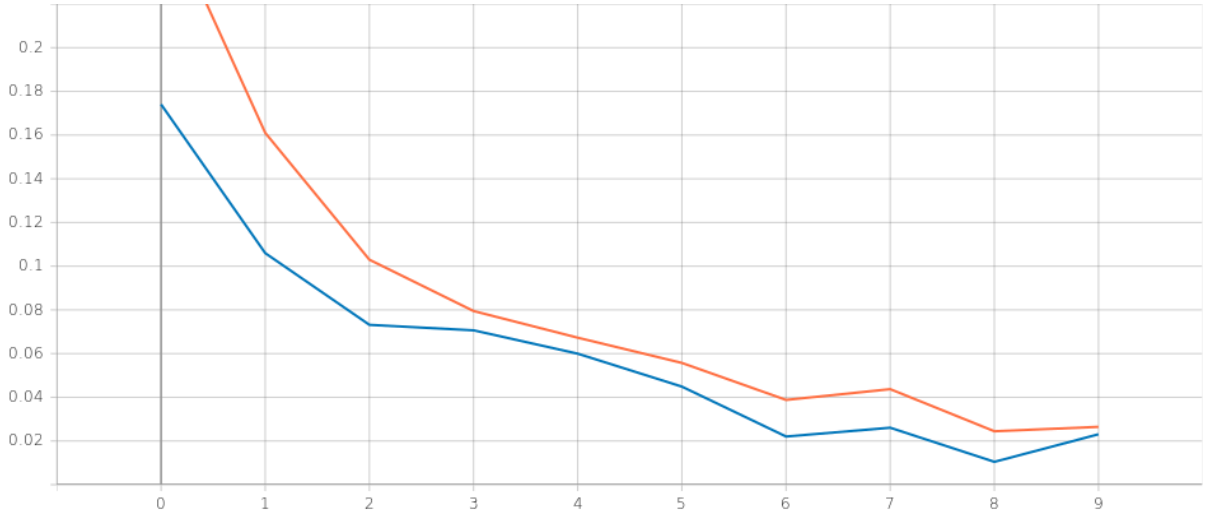

Figure 30:loss of train and validation in terms of epoch

The same colors expressing the above in the (Figure 29) also here in the (Figure 30) symbolize the same thing, but this curve is the percentage of loss in terms of the epoch. We notice in this (Figure 30) that at the first batch, the loss rate was less than 10% at the train and the validation, which indicates that the result is very good from the first batch, and this is thanks to the loss function and the optimizer was an ideal choice, and upon reaching the ninth batch we see The loss rate is less than 2%.

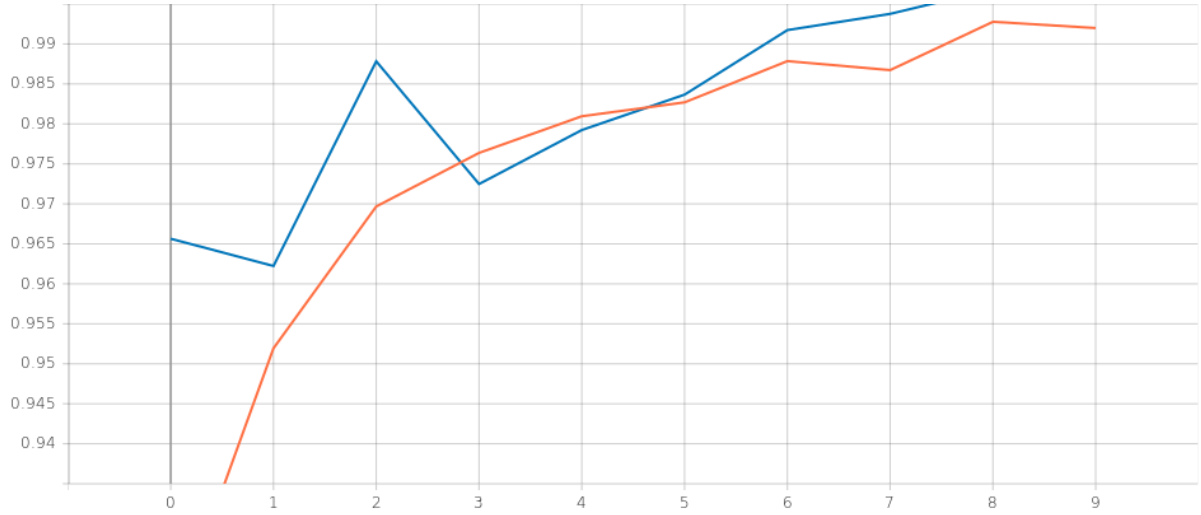

Figure 31: Accuracy masked of train and validation in terms of epoch

As for the percentage of the (Figure 31), the red is the train, the filled data, and the blue is the validation. We see here that the validation starts at the beginning of a high percentage and decreases until it reaches the first batch and increases until it reaches the second batch and decreases and then keeps increasing and also by a percentage For the train it keeps increasing to the end and we notice also that it corresponds to the (Figure 29) Search from the first batch the percentage was above 90% to the last batch above 99% or maybe they quickly reached 100%. And we get the perfect result.

5.6.1.2 Evaluation of lstm model

Here the results of the evaluation are displayed by means of the test data so that we know whether the model provides a realistic result, and this time in terms of iterations, which means the repetition of the batch in only one epoch. In short, the curves here are the percentage of compatibility or loss in one batch only.

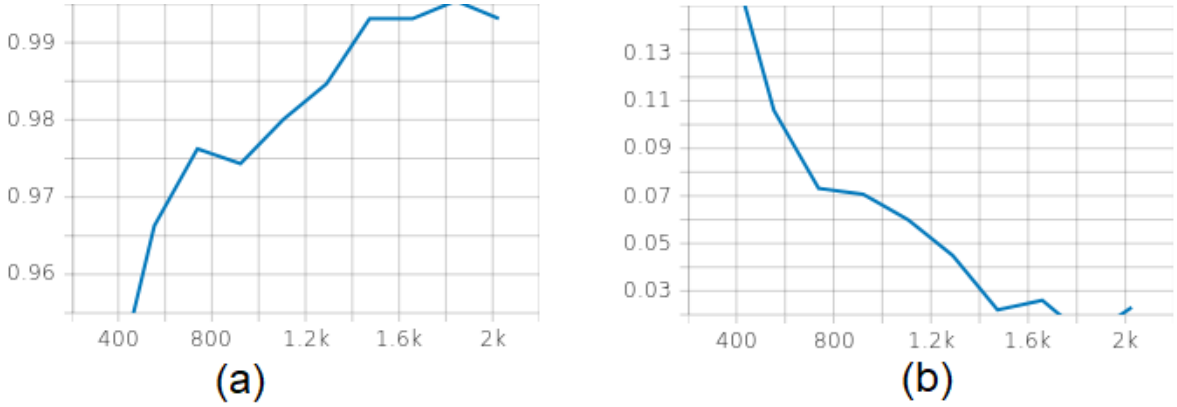

Figure 32: Evaluation of Accuracy (a) and loss (b) in terms of iterations

We note in the (Figure 32(a)) is the percentage of compatibility in terms of frequency. We note the curve is increasing and the percentage has exceeded 99%, and also in the (Figure 32(b)) the percentage of loss is decreasing and has reached zero, which indicates that the evaluation is high and the result is very perfect and the model can be relied on in Predicting the author's name.

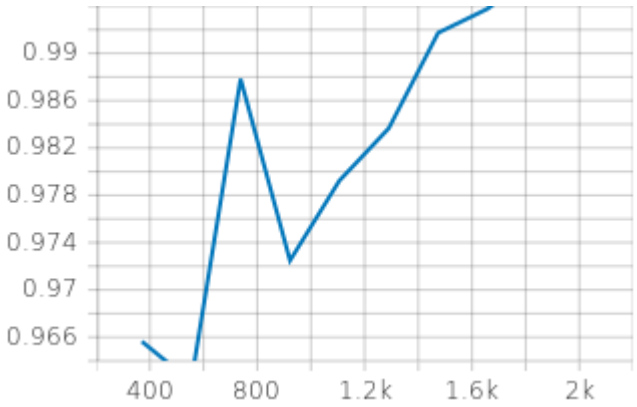

Figure 33 : Evaluation of Accuracy masked in terms of iterations

We also notice in the (Figure 33) the convincing percentage of convincing is decreasing and increasing and decreasing and increasing twice, but in the last batch we see it exceeded 99% and also the evaluation is very high.

```
# evaluate with test set
best_model = AuthorModeLSTM(source_vocab_size,max_seqlen)
best_model.build(input_shape=(batch_size, max_seqlen))
best_model.load_weights(best_model_file)
best_model.compile(
    loss="categorical_crossentropy",
    optimizer="adam",
    metrics=["accuracy", masked_accuracy()])

test_loss, test_acc, test_masked_acc = best_model.evaluate(test_dataset)
print("test loss: {:.3f}, test accuracy: {:.3f}, masked test accuracy: {:.3f}".format(
    test_loss, test_acc, test_masked_acc))
```

```
102/102 [==============================] - 62s 540ms/step - loss: 0.0175 - accuracy: 0.9944 - masked_accuracy_fn: 0.9963
test loss: 0.017, test accuracy: 0.994, masked test accuracy: 0.996
```

Figure 34 : Evaluation of Accuracy masked result

The (Figure 34) shows the result of the evaluation in the form of data and it is the same as what we said in the previous two (Figure 32) and (Figure 33) and this evaluation is perfect as we said in the past.

5.6.2 Results using GRU model

Here we show all the results related to GRU model using the accuracy and loss of train besides of the validation in terms of epoch.

5.6.2.1 Accuracy,loss of train,validationin terms of epoch

Accuracy and loss results are shown in (Figure 35) which shows accuracy. In terms of the epoch, and in the (Figure 36) it shows the percentage of loss in terms of the epoch, and the (Figure 37)shows the convincing results of compatibility.

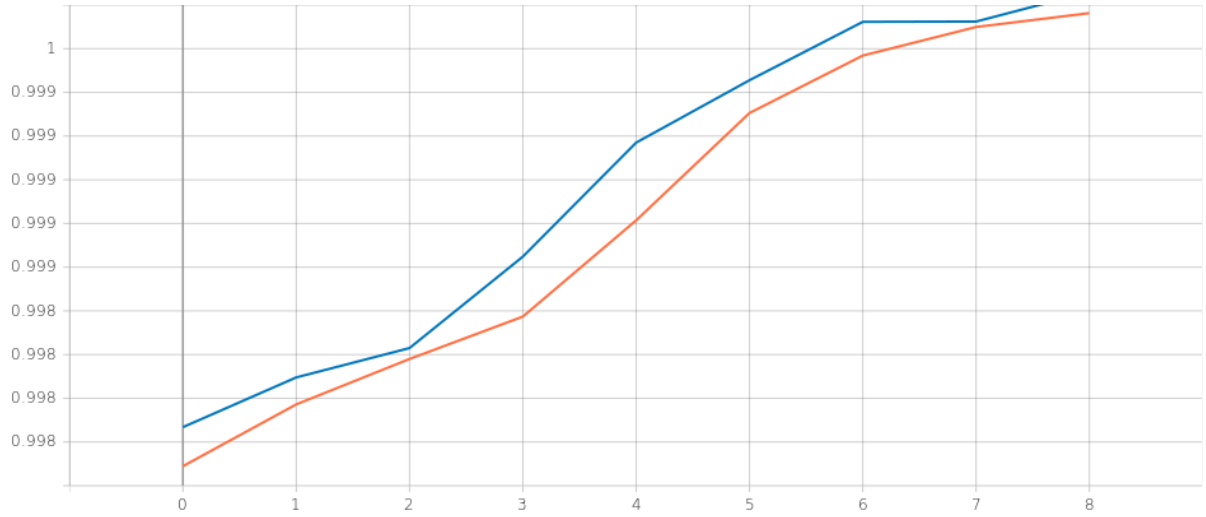

Figure 35 : Accuracy of train and validation in terms of epoch

The red graph represents train accuracy, and the blue graph represents validation accuracy in terms of epoch payments.

We notice in the (Figure 35) that at the first epoch, the accuracy rate was above 99%, and this indicates that the training not was successful 100%. When the ninth batch of epoch was reached, it reached 100% and exceeded it, and the result was not perfect because the over fit.

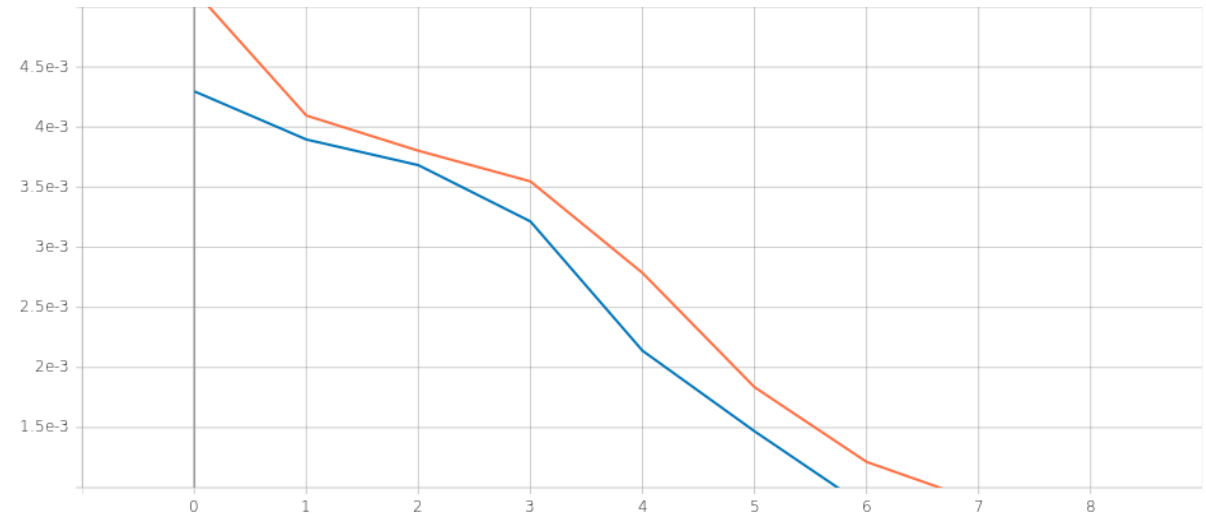

Figure 36:loss of train and validationin terms of epoch

The same colors expressing the above in the (Figure 35)also here in the (Figure 36)symbolize the same thing, but this curve is the percentage of loss in terms of the epoch. We notice in this (Figure 36)that at the first batch the loss rate was less than 0.4% at the train and the validation, which indicates that the result is very good from the first batch and this is thanks to the loss function and the optimizer was an ideal choice, and when reaching the ninth batch we see The loss rate is less than 0.15%.

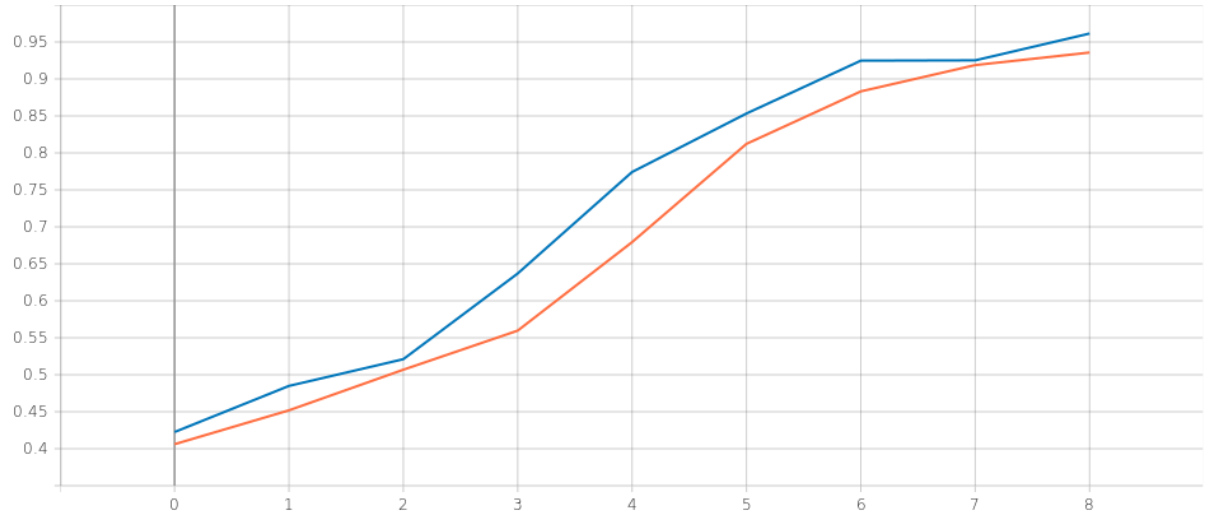

Figure 37: Accuracy masked of train and validation in terms of epoch

As for the (Figure 37), the red is the train, the stuffed data, and the blue is the validation. We see here the two increasing to the end, and we note that it corresponds to the (Figure 36)but here in (Figure 37)from the first batch the percentage was above 45% to the last payment 95%

5.6.2.2 Evaluation of GRU model

Here the results of the evaluation are displayed by means of the test data so that we know whether the model provides a realistic result, and this time in terms of iterations, which means the repetition of the batch in only one epoch. In short, the curves here are the percentage of compatibility or loss in one batch only.

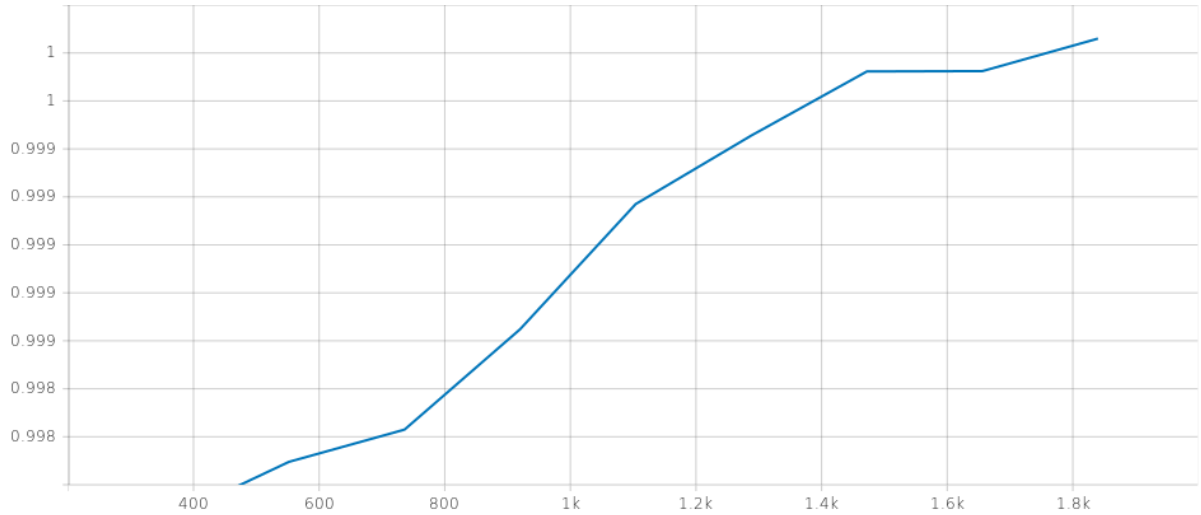

Figure 38 : Evaluation of Accuracy in terms of iterations

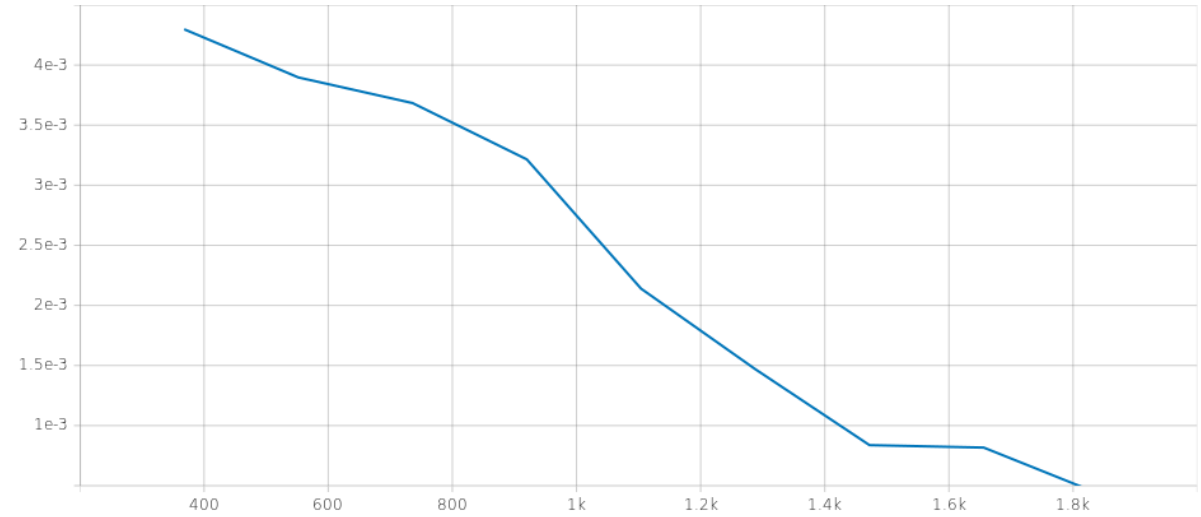

Figure 39 : Evaluation of loss in terms of iterations

We note in the picture (Figure 38)is the percentage of compatibility in terms of frequency. We note the curve is increasing and the percentage has exceeded 100%. Also in the (Figure 39) the percentage of loss has decreased and reached zero, which indicates that the evaluation is high and the result is not perfect and the model maybe can be relied on in predicting the name of the writer.

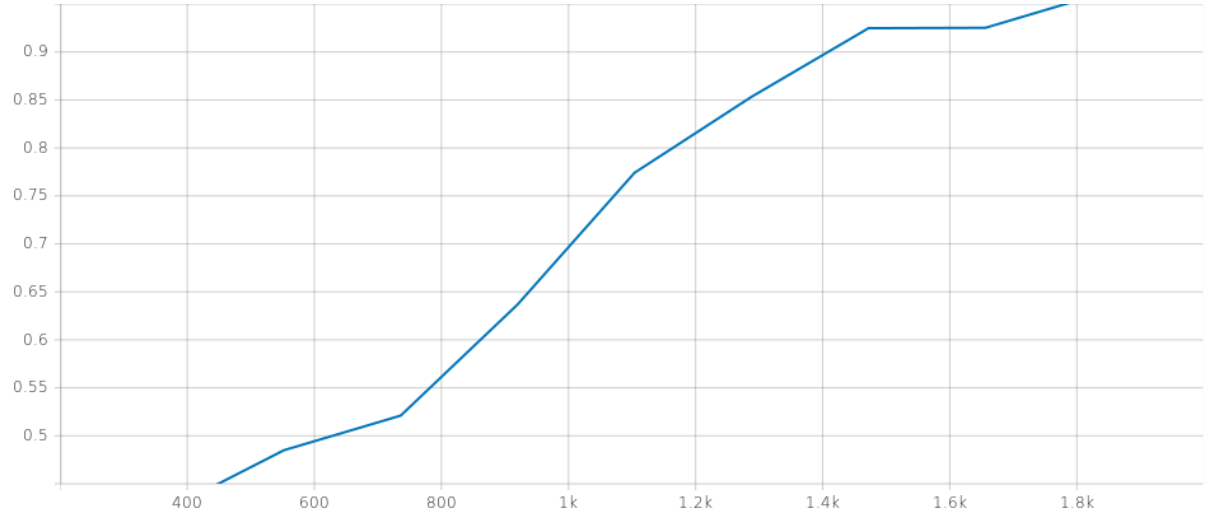

Figure 40 : Evaluation of Accuracy masked in terms of iterations

We also note in the (Figure 40) the convincing percentage of compatibility is increasing, but in the last batch we see it exceeded 99% because the over fit.

```
[ ]  # evaluate with test set
     best_model = POSTaggingModel(source_vocab_size, target_vocab_size+1,
         embedding_dim, max_seqlen, rnn_output_dim)
     best_model.build(input_shape=(batch_size, max_seqlen))
     best_model.load_weights(best_model_file)
     best_model.compile(
         loss="categorical_crossentropy",
         optimizer="adam",
         metrics=["accuracy", masked_accuracy()])

     test_loss, test_acc, test_masked_acc = best_model.evaluate(test_dataset)
     print("test loss: {:.3f}, test accuracy: {:.3f}, masked test accuracy: {:.3f}".format(
         test_loss, test_acc, test_masked_acc))

     102/102 [==============================] - 88s 849ms/step - loss: 7.0076e-06 - accuracy: 1.0000 - masked_accuracy_fn: 0.9995
     test loss: 0.000, test accuracy: 1.000, masked test accuracy: 1.000
```

Figure 41: Evaluation of Accuracy masked result

The (Figure 41) shows the result of the evaluation in the form of data and it is the same as what we said in the previous pictures (Figure 39), (Figure 40), (Figure 41) and this evaluation is perfect as we said in the past.

5.6.3 Results using linear classifiermodel

Let's be honest, there is no interesting result. All the curves in this model are empty and meaningless. Because this model is a type of machine learning, but we have data that we can show the results through

```
[11] # build estimator LinearClassifier
     linearClassifier=tf.estimator.LinearClassifier(
         feature_columns,
         model_dir='',
         n_classes=3
     )

     WARNING:tensorflow:Using temporary folder as model directory: /tmp/tmpcbamsz8r
```

```
[12] # train the estimator
     linearClassifier.train(
         train_input_fn,
         steps=10
     )
```

Figure 42 :model LinearClassifier with estimator

The (Figure 42)shows making a model of the estimator and training it with 10 steps because, frankly, I tried to raise the number of steps, and the results were very bad. I think that 10 is very suitable.

```
[13] # input the test
     out_input_fn=input_fn(test_data,test_label,0)
     # evaluate the estimator
     linearClassifier.evaluate(out_input_fn)

     {'accuracy': 0.40170088,
      'average_loss': 2225.9358,
      'loss': 2225.8008,
      'global_step': 10}
```

Figure 43 : Evaluatemodel LinearClassifier with test data

(Figure 43) shows the final evaluation result with the test data, and the compatibility rate is 40%, and the catastrophic loss rate is 2,225,8008, and the loss rate is almost the same, which indicates that the model is less than 70% in compatibility, so it is not very useful for us in predicting.

5.7 Evaluation measures

To test the quality of any classification system like Support Vector Machines, there's need to perform some evaluation metrics. Support Vector Machines are classification algorithm which I explained briefly in kernels.

The following matrix shows the true or false positive or negative by predicted

		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

Table 1: confusion matrix

5.7.1 definitions

5.7.1.1 Recall

Recall measures the ability of the system to retrieve all documentsIn other words, it measures the proportion of Canadianrelevant documents returned by the system in respect of allrelevant documents contained in the collection. It is expressed by [44]:

$$Recall = \frac{True\ Positive}{False\ Negative + Ture\ Positive}$$

5.7.1.2 Precision

Immediately you can see that Precision indicates how accurate your model is on the positive ones, how many of them are actually positive.

Accuracy is a good measure to determine, when the costs of false positives are high. For example, spam detection by email. In spam detection, a false positive means that an email that is not spam (real negative) has been identified as spam (spam predicted). The email user could lose important emails if the accuracy is not high for the spam detection model.

$$Precision = \frac{True\ Positive}{False\ Positive + True\ Positive}$$

5.7.1.3 F score

Now, if you read many other documents on precision and recall, you can't avoid the other measure, F1 which is a precision and recall function.

The F1 score could be a better measure to use if we have to balance accuracy and recall AND if there is an uneven distribution of classes (large number of actual negatives).

Looking the formula is as follows:

$$f1\_score = \frac{(2\ x\ precision\ x\ recall)}{(precision + recall)}$$

5.7.1.4 Accuracy

Accuracy is a measure that generally describes model performance for all classes. It is useful when all classes are of equal importance. It is calculated as the ratio between the number of correct predictions and the total number of predictions.

$$Accuracy = \frac{True\ Positive + True\ Negative}{False\ Positive + False\ Negative + True\ Positive + True\ Negative}$$

5.7.2 Model Comparison

Models	**Author Name : EAP**			
	Precision	**Recall**	**F1_Score**	**Accuracy**
LSTM	**99.41**	**99.24**	**99.33**	**99.46**
Linear	44.57	51.13	47.62	54.73
GRU	97.13	96.38	96.76	97.39

Table 2: Result of author Edgar Allan Poe (**EAP)**

Models	**Author Name : MWS**			
	Precision	**Recall**	**F1_Score**	**Accuracy**
LSTM	**98.94**	**99.58**	**99.26**	**99.54**
Linear	30.9	15.96	21.05	62.99
GRU	96.16	97.09	96.62	97.91

Table 3: Result of author Mary Wollstonecraft Shelley (**MWS)**

Models	**Author Name : HPL**			
	Precision	**Recall**	**F1**	**Accuracy**
LSTM	**99.65**	**99.2**	**99.43**	**99.67**
Linear	32.24	42.32	36.6	57.72
GRU	97.15	97.21	97.18	98.38

Table 4: Result of author HP Lovecraft (**HPL)**

5.8 Discussion

Overall, models are showing fairly good results based on our data wich contain three different author, we have used a global evaluation measureswich we performe a precision and recall besides of f_score and accuracy.

The LSTM model has invade all the results a reched the top of them in three tables wich proceed a 99% in every evaluation measures, and that means that the model was well prepared wich passed throught all necessary steps , then it trained well.

The GRU model comes after with a great hight results between 96% and 98% wich means was well trained too, therefore the linear classifier was the last with low results because of the base steps that passed throught during the preparation and the training .

5.9 Conclusion

It is very clear in Tables 2,3 and 4 that the result is in favour of LSTM model and this is entirely consistent with the data results, but we noticed that GRU model also had a very high results in the previous graphical results and it is here in the tables that it did not surpass LSTM because the probability that this is due to the reason of the small data but with that, As

for LinearClassifer, its results must have been as bad because it is of the standard ml, and also the data did not provide a satisfactory result.

Conclusion general

The internet become a place for uncounted of unknown documents and articles that contain a very important topics related to anonymous author invades all the domains, It often happens that articles published in the press or sometimes entire books are not signed and nothing is known about their author. It also happens that authors attribute to themselves the authorship of an unsigned article or book or, on the contrary, that critics question this authorship.

Our initial objective during this thesis was to make a web application by exploiting different variants of neural architectures based on automatic recognition of an author.

In this work, we started by explaining some theories that we have talked about based on machine learning. We made a small presentation in order to understand this domain, then we have presented some notions of Automatic identification of the author of an article from the writing style.

Then we discussed about creating Automatic Author Identification of an article from the writing style that based on neural networks, we have covered all the steps necessary to Building a good Automatic Identification model. We have proposed methods that have been used during preprocessing in automatic identification generation such as LSTM and GRU.

Once our approach has been well defined we are ready to put to the test. We presented the datasets of our dataset (EAP: Edgar Allan Poe, HPL: HP Lovecraft; MWS: Mary Wollstonecraft Shelley) which consisted of three different authors , we worked on and processed. And in the end we have completed this work with a summary of the results and tests obtained by our testing on data.

Finally the best rnn cell that has a great results during the test on our data is LSTM.

In the end of this work , we have reached that we could identifier an author based on his style with great precession.

As future works , on propose :

- Develop the concept of author style.
- Test application with dataset that contains more than three authors.
- Test others models of classification.
- Test the application based on different language datasets.

Bibliography

[1]Charles Coustille, *Une histoire du plagiat universitaire*, sur le site « fabula.org ».

[2] Same source as above

[3]L'article a paru dans « *l'Entente* » le 23 février 1936.

[4]Author(s): Adam Pawłowski and ArturPacewicz ,HistoriographiaLinguistica, Volume 31, Issue 2-3, Jan 2004, p. 423 – 447

[5]Stein and Eissen, 2007; Layton et al., 2013; Jayapal and Goswami, 2013

[6]Jérémy Ferrero_, Alain Simac-Lejeune, Automatic detection and grouping of writing style in a text

[7]Voir Robert Ricatte, 1970, Préface Giono, Œuvres complètes, volume 1, Paris, Gallimard, La Pléiade, 1971, p. XLVI.

[8] GIONO, Jean, *Œuvres romanesques complètes*, édition dirigée par Robert Ricatte, Paris, Gallimard, coll. « Bibliothèque de la Pléiade », 6 tomes, 1971-1983.

[9] V .Magri ,Le deep learning comme défi pour identifier le style d'un écrivain : l'exemple de Jean Giono, p 3

[10](Jérémy Ferrero_, Alain Simac-Lejeune, Automatic detection and grouping of writing style in a text

[11]C. Paganelli, La recherche d'information dans des bases de documents techniques en texte intégral. Etude de l'activité des utilisateurs, Thèse de Doctorat en sciences de l'information et de la communication, Université Stendhal de Grenoble 3, 1997, 354p

[12]Zoulikha BENBLAL and Fatima BELOUAFI «Integration of an Arab lemmatizer in the framework of an information search system», Mémoire Master, Adrar, Algérie, 2014

[13]E. Cambria and B. White, "Jumping NLP curves: a review of Natural language processing research", [review article], Computational Intelligence Magazine, IEEE, vol. 9, pp. 48-57, 2014

[14]Toutanova, Kristina, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding" [archive], at arXiv.org, 11 October 2018 (accessed 31 July 2020).

[15] Christopher D. Manning, HinrichSchütze, Foundations of Statistical Natural Language Processing, MIT Press (1999), (ISBN 978-0-262-13360-9), xxxi.

[16]CHELLOUF Athman and AYACHI Yacine, "Proposition et conception d'un système de détection d'événements sur les réseaux sociaux dédié à la langue Arabe", Ecole Militaire Polytechnique, Algérie, 2018

[17]Jean VERONIS, "Informatique et Linguistique" teaching unit INF Z18, Université de Provence, centre informatique pour les lettres et sciences humaines, France, 2001

[18]J-H. JAYEZ, «Compréhension automatique du langage naturel le cas du groupe nominal en français», Masson, 1985

[19]Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press. http://www. deeplearningbook.org (2016)

[20]Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1985).Learning Internal Representations by Error Propagation. Parallel Distributed Processing: Explorations in the Microstructure of Cognition

[21]Britz, D. (2015). Recurrent Neural Networks Tutorial, Part 3 - BackpropagationThrough Time and Vanishing Gradients. URL: http://www.wildml. com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/.

[22]Pascanu, R., Mikolov, T., and Bengio, Y. (2013).On the difficulty of training Recurrent Neural Networks. Proceedings of the 30th International Conference on Machine Learning (ICML)

[23]Hochreiter, S., and Schmidhuber, J. (1997). LSTM can solve hard long time lag problems. Advances in Neural Information Processing Systems (NIPS).

[24](Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. 9(8), 1735–1780 (1997)

[25](Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. 9(8), 1735–1780 (1997)

[26]Gers, F.A., and Schmidhuber, J. (2000). Recurrent Nets that Time and Count. Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN)

[27] Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, by K. Cho, arXiv:1406.1078, 2014

[28] An Empirical Exploration of Recurrent Network Architectures, by R. Jozefowicz, W. Zaremba, and I. Sutskever, JMLR, 2015 and Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling, by J. Chung, arXiv:1412.3555. 2014

[29](Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press. http://www. deeplearningbook.org (2016)).

[30]Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press. http://www. deeplearningbook.org (2016)

[31]Karpathy, A. (2015). The Unreasonable Effectiveness of Recurrent Neural Networks. URL: http://karpathy.github.io/2015/05/21/rnneffectiveness/.]

[32]Karpathy, A., Li, F. (2015).Deep Visual-Semantic Alignments for Generating Image Descriptions.Conference on Pattern Recognition and Pattern Recognition (CVPR).

[33]Socher, et al. (2013).Recursive Deep Models for Sentiment Compositionality over a Sentiment Treebank. Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing (EMNLP)

[34]Bahdanau, D., Cho, K., Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate.arXiv: 1409.0473 [cs.CL].,

[35]Vinyals, O., et al. (2015).Grammar as a Foreign Language.Advances in Neural Information Processing Systems (NIPS).

[36](Hosseini,M.-P.: Proposing a new artificial intelligent system for automatic detection of epileptic seizures. J. Neurol. Disorders 3(4) (2015))

[37]Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in Neural Information Processing Systems, pp. 3104–3112 (2014)

[38]Wallach, I., Dzamba, M., Heifets, A.: AtomNet: a deep convolutional neural network for bioactivity prediction in structure-based drug discovery (2015). arXiv preprint arXiv:1510.02855

[39]Hosseini, M.P., Lau, A., Lu, S., Phoa, A.: Deep learning in medical imaging, a review. IEEE Rev. Biomed.Eng. (2019)).

[40]https://github.com/ tensorflow/tensorflow

[41]https://pytorch.org/

[42]https://caffe. berkeleyvision.org/

[43]https://mxnet.apache.org/

[44] S. Chaudiron, “Evaluation of Information Retrieval Systems”. Hermès, 2004.