الجمهورية الجزائرية الديمقراطية الشعبية

Ministry of Higher Education and Scientific Research

UNIVERSITY YAHIA FARES of MEDEA

Faculty of Science

Department of Mathematics and Computer Science

Specialty:

Systems Engineering and Web Technology

Theme:

Arabic Pos Tagging based in Deep Learning

Presented by :

MOKHBAT Leila

Proposed and directed by :

Mr.HADJIR Ismail

Academic year: 2021-2022

ACKNOWLEDGEMENT

BEFORE ALL I THANK GOD FOR BLESSING ME IN ACHIEVING THIS WORK.

AND MY BELOVED FAMILY FOR THE SUPPORT THEY GAVE ME IN ALL MY ACADEMIC JOURNEY

ALL THE THANKS GO ALSO TO MY SUPERVISOR « MR.HADJIR ISMAIL » WHO HAS GUIDED ME WITH GREAT PROFESSIONALISM AND WHOSE GUIDANCE AND SUGGESTIONS HAD HELPED ME A LOT IN THE FULFILMENT OF THIS RESEARCH WORK.

Summary

Table List

Figures List

General Introduction……I

Chapter 01: Arabic Post Tagging and Deep Learning

Introduction……1

1.1 ***The Arabic Language****……1*

1.1.1 Generation of Arabic Language……1

1.1.2 Definition of Arabic……1

1.1.3 History of Arabic……2

1.1.4 Status of Classical Arabic……3

1.1.5 Modern Standard Arabic……4

1.1.6 Alphabet……5

1.2 Natural Language Processing (NLP)*……5*

1.2.1 A brief History of Natural Language Processing……5

1.2.2 Natural Language Processing Definition……6

1.2.3 Steps of Natural Language Processing ……7

1.2.4 Approaches to Natural language processing……8

1.2.5 Natural Language Processing Applications……9

1.2.6 Major Tasks in NLP……10

1.3 Parts of Speech Tagging (POST Tagging) *……11*

1.3.1 History of Post Tagging……11

1.3.2 Definitions of Post Tagging……14

1.3.3 Techniques for POS Tagging……14

1.3.4 Post Tagging with Hidden Markov Model……15

1.3.5 Part of Arabic Pos Tagging……16

1.3.6 The Post Tagging with NLP……18

1.4Deep Learning*……18*

1.4.1 History of Deep learning……18

1.4.2 Definition of Deep Learning……19

1.4.3 How Deep Learning Works……19

1.4.4 Deep learning Applications……20

1.4.5 Deep Learning Types……22

1.4.6 Popular Deep learning Models……24

1.4.7 How Convolution Neural Networks Work……29

1.4.8 Deep Learning Performance……31

1.5 Neural Networks (NN) *……31*

1.5.1 A Brief History of Neural Networks……31

1.5.2 Neural Network Definition……31

1.5.3 Architecture of Neural Network……32

1.5.4 Why Recurrent Neural Networks ? ……32

1.5.5 Difference between Neural Network and Deep Learning Neural Network……32

1.6 Recurrent Neural Networks*……33*

1.6.1 Recurrent Neural Networks Definition……33

1.6.2 Advantages & Disadvantages of Recurrent Neural Network……34

1.6.3 How Recurrent Neural Networks Work……34

1.6.4 Types of Recurrent Neural Networks……36

1.6.5 Long Short Term Memory (LSTM) ……41

1.6.6 Word Embeddings……*45*

Conclusion……*49*

Chapter 02: Proposed Approach

Introduction……*50*

Proposed Approach Overview……*50*

2 DataSet……*52*

2.1KALIMAT DataSet……*52*

2.2 Categories of the Dataset……*52*

2.3 Creation of Dataset……*53*

2.4 About the Dataset……*54*

2.5 **preprocessing of the data**……*55*

2.5.1 Decomposition of the corpus……55

2.5.2 Vectorize X and Y……58

2.5.3 Pad Sequences……59

2.6 Creation Of RNN……*60*

2.6.1 One hot encoding……62

2.6.2 Creation of Bidirectional LSTM Layers……62

2.6.3 Creation of the GRU models ……64

2.6.4 Training and Testing……65

2.6.5 Compilation the Models……66

Conclusion……*68*

Chapter 03:Implementation and Result

Introduction……69

3.1Development environment: ……69

3.1.1 Python.....69

3.1.2 Google Colab 73

3.1.3 Google Drive.....74

3.2 Implementation of our approach.....75

3.2.1 Dataset.....75

3.2.2 Data download 75

3.2.3 Data Decomposition 76

3.2.4 Vectorizations 77

3.2.5 Pad sequences 78

3.3 Split the Data into Train /Test.....78

3.4 Model Development.....79

3.4.1. Create the first model Bidirectional LSTM.....80

3.4.2 The second models GRU.....83

3.5 Evaluation 84

3.6 Comparison between the two Models.....87

3.7 Test of Models 90

3.7.1 First model (B-LSTM).....90

3.7.2 Second Model(GRU).....91

3.8 Related work.....91

Conclusion.....94

***General Conclusion*.....95**

***Bibliography*.....96**

Tables List

Table2.1Collections with their numbers of articles……..52

Table.2.2 KALIMAT Corpus Statistic. ……..53

Table2.3 Explication of tags……57

Table2.4Each cell denotes the number of times the word is present in the text……58

Table 3.1 Characteristic of Two Models……88

Table 3.2 F1-Score of POS Models Tested By Test Datasets For 40 Epochs. The highest F-Score Is Bolded……..92

Table3.3A hybrid Arabic POS tagging for simple and compound morphosyntactic tags……93

Table 3.4: The accuracy of our POS tagger on the WikiNews dataset(Darwish and Mubarak, against Madamira) ……94

Figures list

Figure 1.1 Alphabet of Arabic Language……5

Figure.1.2.Example for Post Tagging……14

Figure1.3 Noun Part of Speech……16

Figure1.4 Verb Part of Speech……17

Figure1.5Particle Part of Speech……18

Figure1.6 Example Arabic Post Tagging in NLP [……18

Figure.1.7Schematic of Rosenblatt perceptron……25

Figure1.8 Deep Feed Forward Neural Network Architecture……25

Figure.1.9RNN Architecture……26

Figure1.10 Bidirectional RNN Architecture with forward and backward hidden RNN-layers……27

Figure1.11 LSTM unit architecture……28

Figure1.12 GRU Architecture……28

Figur1.13 Generic representation of CNN architecture……30

Figure1.14 Architecture of Simple Neural Network and Deep Learning Neural Network……33

Figure1.15 Processing sequence one by one……35

Figure1.16 Passing hidden state to next time step……35

Figure 1.17 RNN Cell……35

Figure 1.18 Tanh squishes values to be between -1 and 1……36

Figure1.19 Vector transformations without tanh……36

Figure 1.20 RNN Types……37

Figure 1.21 Example One to One ……38

figure1.22 Type of RNN One to One……38

Figure 1.23Example One to Many……..38

Figure 1.24Type of RNN (One-to-many) ……38

Figure1.25 Example Many to One……39

Figure 1.26 Type of RNN (Many-to-one). ……39

Figure1.27 Type of RNN (Many-to-many [Tx=Ty]). ……39

Figure1.28ExampleTx=Ty……39

Figure1.29 Example Tx!=Ty ……40

Figure 1.30 Type of RNN: Many-to-Many [Tx!=Ty] ……40

Figure 1.31 LSTM Gates.……..41

Figure 1.32 LSTM Cell and It's Operations.……42

Figure 1.33Sigmoid squishes values to be between 0 and 1 ……43

Figure 1.34 Forget gate Operations.……43

Figure 1.35Input gate Operations.……..44

Figure 1.36 Calculating cell state.……..44

Figure 1.37 Output gate operations.……..45

Figure 1.38 Word2Vec Training Models. ……47

Figure 1.1 Architecture Overview……50

Figure.2.2 Global Architecture of our Approach……51

Figure.2.3 Example of Article in the Dataset……54

Figure2.4 Example Of Data Preprocessing ……55

Figure2.5 Example of the Decomposition of two Sentences……56

Figure.2.6 Example of Encoding X……59

Figure.2.7 Example of Encoding Y……59

Figure.2.8 Example of Padding Tokens……60

Figure.2.9 Example of Padding Tags……60

Figure 2.10 B-LSTM Architecture……64

Figure2.11 Architecture of GRU Model……65

Figure3.1 Google Drive applications and functions……74

Figure3.2 dataset example from dataset……75

Figure3.3 Connection the Dataset with our Program……76

Figure3.4 Example features and labels……76

Figure3.5 construction of sentences (features)……77

Figure 3.6 Encoding Example……77

Figure3.7 Padded Sequences Examples……78

Figure 3.8 Split the data into Train and Test……79

Figure3.9 Function to post X Shape and Y Shape……79

Figure 3.10 Shape of X_Train and X_Test……79

Figure 3.11 One Hot Encoding the Output ……80

Figure3.12 Execution One Hot Encoding……80

Figure 3.13 Creation the Sequential Model……80

Figure3.14 Creation Embedding Code……81

Figure 3.15 Creation of Bidirectional LSTM Layers……81

Figure 3.16 Creation of Time Distributed and Activation……81

Figure 3.17 Compile and Summary the B-LSTM……82

Figure 3.18 Shows B-LSTM Summary……82

Figure3. 19 Creation of GRU layer……83

Figure 3.20 Shows GRU Summary……83

Figure3.21 Fit of the B-LSTM Model……84

Figure 3.22 Result of Fit for B-LSTM Model……84

Figure 3.23 Function of Evaluation of Accuracy B-LSTM……85

Figure3.24 Result Accuracy of Training the B-LSTM……85

Figure3.25 Result F-scores of Training B-LSTM……85

Figure 3.26Function to Calculate the Evaluation of the Precision and Recall……85

Figure3.27 Result of the Precision and Recall of B-LSTM Model……85

Figure 3.28 Fit of the GRU Model……86

Figure 3.29 Result of Fit for GRU Model……86

Figure 3.30 Function of Evaluation GRU Model……86

Figure3.31 Result of Training GRU Model……87

Figure3.32 Function of Evaluation the Precision and Recall……87

Figure3.33 Result of the Precision and Recall of GRU Model……87

Figure3.34 Function of Comparison the Accuracy ……88

Figure 3.35 Different Loss and Accuracy of two Models……89

Figure 3.36 Comparison of Evaluation of two Models……89

Figure 3.37 Function of Testing the First Model……90

Figure 3.38 Testing in B-LSTM……90

Figure 3.39 Function of Testing the Second Model……91

Figure 3.40 Function of Testing the Second Model(GRU)……91

General Introduction

Natural Language Processing is a field of Artificial Intelligence (AI) which consists of analyzing, understanding and generating natural language text and speech to solve engineering problems and allows people to interact with computers with their spoken speech. POS Tagging is a major step in most NLP applications to determine the proper grammatical tag or syntactic category of a word depending on its context. For example, the words in the sentence: “Ali has the can” are tagged as 'noun', 'verb', 'determiner' and 'noun' respectively. However, some words have more than one tag with respect to the context. For instance, in the previous example the word “can” tagged as a noun, but in another sentence, it can be used as a verb. In the Arabic language, there are three main POS categories: noun, verb and particle[66]

The Arabic language is considered as the most famous language in which it is the sixth language in the word. Nowadays, the majority of nations are dealing with Arabs in the field of business and many foreign people are converting Islam. These two opportunities encourage non-native speakers to learn Arabic language through translation and looking for the interpretation of Quran. This later created a problem in translation because the informatic system face several challenges to differentiate between noun and verb. Also most natural language processing applications like information extraction, machine translation, speech recognition and grammar and spelling checkers, rely on pos tagger, that will become a major focus of research and development.

Arabic POS Tagging is the process of identifying lexical category of the Arabic word existing in a sentence based on its context. POS Tagging is not only limited to word-by-word substitution but it must analyze relationship between words to identify the categories of the words. The most used categories are noun, adverb, verb, adjective and particle. This is done on the basis of words role whether they are in isolation as well as in the sentence. Most words occurring in undiacritized Arabic text have the ambiguity in terms of their part of speech.

Researches show interest in NLP started in 1946 by developing a Machine Translation (MT) system to break the enemy codes during World War. POS tagging can be divided into linguistic taggers and machine learning taggers. Linguistic taggers obtain knowledge from a set of rules or constraints defined by linguists. The second type can be also divided into supervised and unsupervised taggers. Supervised taggers use pre-tagged corpus, select the tag set and then apply it to a specific language. [66]

In this work, we proposed a deep learning approach for Arabic Pos Tagging, to solve the ambiguity and problem of classification of words based on Recurrent Neural Network (RNN). We choose the Bidirectional LSTM, which is a modified version of the RNN because of its credibility and high-performance model in many NLP works. In addition, we make another test with the GRU model, trying to get a fast result. The proposed models of RNN(Bi-LSTM and GRU) will be compared and the model achieving better results will be selected.

➢ Problematic

Many Arabic words have more than one form that can indicate different meanings with the same written form. The first problem that our program will face is through reading written text correctly without the diacritization with absence of Post Tagging such as the word "كتب". It means to write if it is read "kataba". So, It's a verb. When it is read as "kotob" the meaning will change into books which is a noun. The second difficulty is a similarity of word. There are lot of Arabic words that can be ambiguous like some pronouns, which always points to other sentence. Thus. The capability to recognize and identify these different classes words is very important. Also the majority of non-native speakers will find difficulties to guess the meaning of many Arabic word due to several things which effect on the Pos tagging and make it delicate. These include: ambiguous words or phrases, Unknown word, and multipart words. As result, it is very important to recognize, identify and classify the different classes of Arabic words.

➢ Objective

The main objective of this research is to develop an automatic Arabic-language POS Tagging. Using a Deep Learning(DL)approach, based on different models of RNN. The goal was that our system could have the capacity (deducted from the higher accuracy value achieved during train and test steps) to identify the correct Pos Tag from each word in the sentence. The models used in this work be compared to the most famous and performing related works and the best one will be selected.

✓ KEY WORDS

Part-of-Speech, Arabic POS Tagging, Natural Language Processing, Deep Learning, Recurrent Neural Network, Bidirectional Long Short Term Memory, Gated Recurrent Unit

Chapter 01: Arabic Post Tagging and Deep Learning

Introduction

In this chapter we have defined the Arabic language with its characteristics and we have posed the problem especially in the Quran. Labeling the word to its part of a speech tag is based on its context in the sentence so this task is not straightforward as a word may have a different POS tag based on its context in the sentence. so, we will look at using Deep Learning Methods Recurrent Neural Networks for POS tagging.

1.1 The Arabic Language

1.1.1 Generation of Arabic Language

As have seen, Arabic Language has undergone different processes of development through centuries. It has been regarded as the only language among all other old languages, Chinese, Sanskrit, Greek and Latin, which still alive. Most of these languages disappeared or were replaced by the local regional languages (Haeri, 2003). Arabic has been viewed as the "Word of God" (Kalam Allah) and yet all Muslims whether Arabic or non-Arabic speakers are required to sanctify this language for its religious status. Such prestigious position paved the way for Arabic to maintain its purity and vivacity all over the years. On the other hand, Arabic is a diglossic language wherein different varieties are used in different social situations (Ferguson, 1959). The most common varieties are Classical Arabic, the language of the Holy Quran and the legacy of the Arab traditional poetry, Modern Standard Arabic, the constitutionally official language of all Arab countries besides being used in media, education and formal speeches, and the Colloquial Arabic which is used in Arab's day to day communication. Each one of the aforementioned varieties has its sociolinguistic role to play in the Arab societies. Linguistically, both Classical Arabic and Modern Standard Arabic are much more similar than Colloquial Arabic which involves various varieties that vary from one another and from region to region in different linguistic aspects.[1]

1.1.2 Definition of Arabic

A Semitic language originating in the Arabian Peninsula that since the 7th century ad has come to be the principal language of a wide area of the Middle East and North Africa. Modern spoken Arabic consists of many different, often mutually unintelligible dialects, and a modified form of classical Arabic is used as the language of education and administration across the area. [2]

1.1.3 History of Arabic

The History of Arabic has been regarded as a member of Semitic languages which include a number of languages in the Middle East and North Africa. It is originally generated from Afro-Asiatic languages which includes besides Arabic different languages such as Hebrew, Ethiopian and other languages. The first emergence of Arabic as a world language goes back to the seventh century CE. The century of Islam diffusion that followed the death of Prophet Mohammed brought both Islam as a religion and Arabic language to the attention of a world that had possessed only the vaguest notion of what went on in the interior of the Arabian Peninsula (Versteegh, 1997). Abstract: This study tries to highlight the historical development of Arabic throughout the centuries. It also presents some linguistic characteristics of Arabic with a slight comparison to Sana'ani Yemen Dialect. In addition, the study presents the diglossic situation of Arabic through discussing the social role played by both Modern Standard Arabic (henceforth MSA) and the different Arabic dialects in the Arab societies as well as the relationship between diglossia and education. Historically, the study shows that Arabic has been undergone different stages of development and progress over the centuries and the advent of Islam marks the real occurrence of Arabic as a standard language through the revelation of the Holy Quran which was revealed in Arabic. Such a remarkable turning point immensely contributed to transmitting Arabic from the unknown side of the history to be a world language. On the other hand, Arabic has unique linguistic characteristics which make it a distinctive language. Moreover, Arabic shares with other Semitic languages the root of pattern morphology which functions as a generator of Arabic words.

Keywords: Arabic; History; Sociolinguistics; Modern standard Arabic; Arabic dialects; Diglossia.2 English Literature and Language Review, 2015, 1(4): 28-36 29 In his argument about the Arabic emergence, Furthaly (2010) shows that Arabic language evolved from an obscure and non-prestigious language to a major world language after the Islamic conquests, the period after Prophet Mohammed' death. Among Semitic languages, Arabic has been described as the most widely spoken with a number of over 330 million speakers according to the CIA report for 2008 besides being the sacred language of more than a billion Muslims around the world. It is the sixth most widely spoken language in the world and one of the six official languages of the United Nations. Of the 330 million native speakers of Arabic, many millions are Christians and few are Jews. However, the great majority of Arabic speakers are Muslims. Arabic is spoken not only in one variety but rather in different varieties across the Arab World. During the second half of the seventh century, the world witnessed the

foundation of the Islamic Arab Empire which by the beginning of the eighth century, it stretched from Spain to Persia. Such massive dominance contributed to the spread of Arabic as a sacred language being the language of the Holy Quran. In addition, several factors laid the ground for Arabic to spread out of the Arabian Peninsula (Holes, 2004). Internationally, Arabs have been known as traders and migrants. These two factors, trading and migration, created a big size of opportunities of contact between Arabs and non-Arabic speakers in areas such as Iraq, Nile Delta in Egypt, Syria, and Palestine. It could be said that such a kind of contact established a strong base of familiarity with Arabic in such areas. Moreover, the spread of Islam in different parts of the world had far-reaching consequences for the development of Classical Arabic. In the wake of the spread of Islam, Arabic turned from being exclusive only to the Arabian Peninsula to be a dominant language of the Middle East and North Africa (Comrie, 2008). [3]

1.1.4Status of Classical Arabic

Classical Arabic is understood to be the language that developed from the varieties of Bedouin tribes in Arabian Peninsula, as recorded in the pre-Islamic poetry (Ferguson, 1959). Due to Quran was revealed in Arabic, it has a sacred and prestigious position not only among the Arabs but rather all Muslims across the world. All Muslims around the world are required to learn Arabic to be able to carry out their religious acts properly. Farghaly (2012) declares that despites its consistency over 1500 years, Classical Arabic is neither the native nor spoken language of any group; nor is it the language of the contemporary writing. Ennaji (2002) mentions that “the Muslim fundamentalists go even further to claim that only classical Arabic is worth teaching and learning because it reflects Muslim tradition, belief and values” (Ennaji, 2002). He confirms that Arabic is associated with identity, roots, cultural authenticity and tradition. Therefore, Classical Arabic is regarded by many scholars as the “real language” whereas the various colloquial dialects of Arabic are distorted forms of this pure language. It is worth mentioning that the revelation of the Holy Quran in Classical Arabic along with the recognition of Arabic as the “Word of God” (kalimatullah) contributed to getting Classical Arabic a high sacred status among all Muslims irrespective of their regional affiliations and nationalities. Haeri (2003) reports that most classical languages disappeared: Sanskrit came to be replaced by the local regional languages of India such as Hindi, Urdu, Bengali etc, and Latin eventually gave way to the Europeans vernaculars, generation by generation, genre by genre, and domain by domain until even the Vatican stopped required it to be the language of prayers (Haeri, 2003). On the contrary, Classical Arabic could have retained its purity and

linguistic features over 1500 years due to its religious status as a language of Quran being recited daily and mostly five times a day (in the prayers) by all Muslims across the world. Such constant use of Classical Arabic during the frequent religious sermons gave it away to be "eternal" in that it did not die away when its speakers ceased to use it in their day to day communication and instead they had recourse to the use of their various regional vernaculars. Moreover, every Friday, all Muslims are required to gather in the mosques of their localities to listen to the weekly-based oration (Xutbah) delivered in Classical Arabic. Thus, all factors cited above have helped Arabic preserve its dynamic practicality over the years.[3]

1.1.5 Modern Standard Arabic

Modern Standard Arabic is the written language used for all Arabic books, newspapers, street signs, magazines, official documents, and business-related materials. Most educated Arabic speakers are able to use MSA as a "lingua franca" to communicate with one another regardless their nationalities or spoken native dialects. It is almost assimilated and intelligible by all Arabs because most of them have adequate exposure to MSA through media, printed materials, religious practices, and certain work-related or social situations. The Arab children also have a relative mastery of MSA because of learning this variety in schools as well as the frequent exposure to the cartoon films presented in Modern Standard Arabic. MSA is the written norm for all Arab countries as well as the major medium of communication for public speaking and broadcasting (Ryding, 2005). However, the evolution of MSA dates back to the nineteenth century when it dropped some of the more archaic words and phrases and has added new technical and scholarly vocabulary as the times have changed. Versteegh (1997) considers the emergence of Modern Standard Arabic as a rebirth of Classical Arabic. Because it is no one's mother tongue, MSA should be acquired through formal education in schools just like Latin and Sanskrit and would be used in religious sermons, medium of instruction in schools and universities, mass media, newspapers and official speeches. It could also be said that Arabic is a symbolic feature of Arabs unity. Despite the disputes that may occur among Arab states every now and again, Arabic is the concerted element that unifies them linguistically being constitutionally the official language of all Arab countries from the Ocean to the Gulf. [3]

1.1.6 Alphabet There are twenty-eight letters in the Arabic alphabet, shown in the table below. For each letter, the table also gives the transliteration of its name, and an example English word which begins with the sound made by that letter. The correct pronunciation for these is given in1B.[4]

Arabe	Translitéré	Nom	Arabe	Translitéré	Nom
ا	ā	Alif	ض	ḍ	Dād
ب	b	Bā	ط	ṭ	Ta
ت	t	Ta	ظ	ḍ	Dha
ث	th	Tha	ع	‘	‘Ayn
ج	j	Jīm	غ	gh	Ghain
ح	ḥ	Ha	ف	f	Fa
خ	kh	Khā	ق	g	Gaf
د	d	Dal	ك	k	Kaf
ذ	ḍ	Thal	ل	l	Lām
ر	r	Ra	م	m	Mim
ز	z	Zāy	ن	n	Nūn
س	s	Sin	ه	h	Ha
ش	sh	Shīn	و	w (u)	Wāw
ص	ṣ	Sad	ي	y (i)	Ya

Figure 1.1 Alphabet of Arabic Language [5]

1.2 Natural Language Processing (NLP)

1.2.1A brief History of Natural Language Processing

The idea of NLP had emerged from the need for Machine Translation in the 1940s. Then the original language was English and Russian. But the use of other words such as Chinese also came into existence in the initial period of the 1960s. Then a lousy era came for MT/NLP during 1966, this fact was supported by a report of ALPAC, according to which MT/NLP almost died because the research in this area did not have the pace at that time. This condition became better again in the 1980s when the product related to MT/NLP started providing some results to customers.

After reaching in dying state in the 1960s, the NLP/MT got a new life when the idea and need of Artificial Intelligence emerged. LUNAR is developed in 1978 by W.A woods; it could analyze, compare and evaluate the chemical data on a lunar rock and soil composition that was accumulating as a resul1.t of Apollo moon missions and can answer the related question.

In the 1980s the area of computational grammar became a very active field of research which was linked with the science of reasoning for meaning and considering the user ‘s beliefs and intentions.

In the period of 1990s, the pace of growth of NLP/MT increased. Grammars, tools and Practical resources related to NLP/MT became available with the parsers. The research on the core and futuristic topics such as word sense disambiguation and statistically colored NLP, the work on the lexicon got a direction of research. This quest of the emergence of NLP was joined by other essential topics such as statistical language processing, Information Extraction and automatic summarizing. The discussion on the history of NLP cannot be considered complete without the mention of the ELIZA, a chatbot program which was developed from 1964 to 1966 at the Artificial Intelligence Laboratory of MIT. It was created by Joseph Weizenbaum. It was a program which was based on script named as DOCTOR which was arranged to Rogerian Psychotherapist and used rules, to response the questions of the users which were psychometric-based. It was one of the chatbots which were capable of taking the Turing test at that time.**[4]**

Currently, a shift has been made to deep learning. This was due to the dominance of deep learning to perform tasks that are difficult to perform by simply using rules and fixed criteria. When it comes to NLP, there is no way to solve the ambiguity of the language. Single word can have multiple meanings according to its neighboring situation and it is not possible to write a rule or to use a decision tree to represent every possible meaning. Deep learning solves this problem efficiently as it does not require a programmer to provide the rules for deciding rather an algorithm itself deduces the process of mapping an input to an output.**[6]**

1.2.2 Natural Language Processing Definition

Natural language processing (NLP) seeks to model language computationally. So that NLP researchers aim to collect knowledge about how humans understand and use language so that appropriate tools and techniques can be developed to make computer systems understand natural languages and process them to perform the required tasks. The foundations of NLP lie in a number of disciplines, namely computer and information science, linguistics, mathematics, electrical and electronic engineering, artificial intelligence and robotics, and psychology. NLP applications include a number of areas of study, such as machine translation, natural language text processing and summarization, user interfaces, multilingual and cross-language information retrieval (CLIR), speech recognition, artificial intelligence, and expert systems.[7]

Currently, a shift has been made to deep learning. This was due to the dominance of deep learning to perform tasks that are difficult to perform by simply using rules and fixed criteria. When it comes to NLP, there is no way to solve the ambiguity of the language. Single word can have multiple meanings according to its neighboring situation and it is not possible to write a rule or to use a decision tree to represent every possible meaning. Deep learning solves this problem efficiently as it does not require a programmer to provide the rules for deciding rather an algorithm itself deduces the process of mapping an input to an output. [8]

1.2.3Steps of Natural Language Processing

There are 5 phases involved in natural language processing:

- **Morphological and Lexical Analysis**: The lexicon of a language is its vocabulary that includes its words and expressions. Morphology depicts analyzing, identifying and description of structure of words.

•**Lexical Analysis**: It involves dividing a text into paragraphs, words and the sentences.

- **Syntactic Analysis**: This involves analyzation of the words in a sentence to depict the grammatical structure of the sentence. The words are transformed into structure that shows how the words are related to each other Eg. “the girl the go to the school”. This would definitely be rejected by the English syntactic analyzer.
- **Semantic Analysis**: This abstracts the dictionary meaning or the exact meaning from context. The structures which are created by the syntactic analyzer are assigned meaning. There is a mapping between the syntactic structures and the objects in task domain. Eg. “colorless blue idea”. This would be rejected by the analyzer as colorless blue do not make any sense together.
- **Discourse Integration**: The meaning of any single sentence depends upon the sentences that preceeds it and also invokes the meaning of the sentences that follow it. Eg the word “it” in the sentence “she wanted it” depends upon the prior discourse context.
- **Pragmatic Analysis**: It means abstracting or deriving the purposeful use of the language in situations importantly those aspects of language which require world knowledge the main focus is on what was said is reinterpreted on what it actuallymeans. Eg “closethewindow?” should have been interpreted as a request rather than an order.[9]

1.2.4 Approaches to Natural language processing

This section examines NLP methods in terms of foundations, typical techniques, differences in processing and system aspects, strength, flexibility, and suitability for various tasks.

- **Symbolic Approach**: Symbolic approaches perform deep analysis of linguistic phenomena and are based on explicit representation of facts about language through well-understood knowledge representation schemes and associated algorithms. The primary source of evidence in symbolic systems comes from human-developed rules and lexicons.

Symbolic approaches have been used for a few decades in a variety of research areas and applications such as information extraction, text categorization, ambiguity resolution, and lexical acquisition. Typical techniques include: explanation-based learning, rule-based learning, inductive logic programming, decision trees, conceptual clustering, and K nearest neighbor algorithms.

- **Statistical Approach:** Statistical approaches employ various mathematical techniques and often use large text corpora to develop approximate generalized models of linguistic phenomena based on actual examples of these phenomena provided by the text corpora without adding significant linguistic or world knowledge. In contrast to symbolic approaches, statistical approaches use observable data as the primary source of evidence.

Statistical approaches have typically been used in tasks such as speech recognition, lexical acquisition, parsing, part-of-speech tagging, collocations, statistical machine translation, statistical grammar learning, and so on.

- **Connectionist Approach:** Similar to the statistical approaches, connectionist approaches also develop generalized models from examples of linguistic phenomena. What separates connectionism from other statistical methods is that connectionist models combine statistical learning with various theories of representation - thus the connectionist representations allow transformation, inference, and manipulation of logic formulae. In addition, in connectionist systems, linguistic models are harder to observe due to the fact that connectionist architectures are less constrained than statistical ones.

Generally speaking, a connectionist model is a network of interconnected simple processing units with knowledge stored in the weights of the connections between units. For example, one unit might represent the concept "mammal" while another unit might represent the concept "whale". Relations between concepts are encoded by the weights of connections between those concepts. Knowledge in such models is spread across the network, and the connectivity between units reflects their structural relationship.

- **Neural networks Approach:** A major drawback of statistical methods is that they require elaborate feature engineering. Since 2015, the field has thus largely abandoned statistical methods and shifted to neural networks for machine learning. Popular techniques include the use of word embeddings to capture semantic properties of words, and an increase in end-to-end learning of a higher- level task (e.g., question answering) instead of relying on a pipeline of separate intermediate tasks (e.g., part-of-speech tagging and dependency parsing). In some areas, this shift has entailed substantial changes in how NLP systems are designed, such that deep neural network-based approaches may be viewed as a new paradigm distinct from statistical natural language processing. For instance, the term neural machine translation (NMT) emphasizes the fact that deep learning-based approaches to machine translation directly learn sequence-to-sequence transformations, obviating the need for intermediate steps such as word alignment and language modeling that was used in statistical machine translation (SMT). Latest works tend to use non-technical structure of a given task to build proper neural network.[10]

1.2.5 Natural Language Processing Applications

- **Automatic summarization:** Automatic summarization is especially relevant when used to provide an overview of a news item or blog posts, while avoiding redundancy from multiple sources and maximizing the diversity of content obtained.
- **Sentiment analysis:** The goal of sentiment analysis is to identify sentiment among several posts or even in the same post where emotion is not always explicitly expressed. Companies use natural language processing applications, such as sentiment analysis, to identify opinions and sentiment online to help them understand what customers think about their products and services .sentiment analysis understands sentiment in context to help you better understand what's behind an expressed opinion, which can be extremely relevant in understanding and driving purchasing decisions.

- **Text classification:** Text classification makes it possible to assign predefined categories to a document and organize it to help you find the information you need or simplify some activities. For example, an application of text categorization is spam filtering in email.
- **Question Answering:** As speech-understanding technology and voice-input applications improve, the need for NLP will only increase. Question-Answering (QA) is becoming more and more popular thanks to applications such as Siri, OK Google, chat boxes and virtual assistants.[11]

1.2.6 Major Tasks in NLP

This list some of the different researches done in NLP:

- **Automatic summarization:** It produces s an understandable summary of a set of text. It is used to provide summaries or detailed information of text of a known type.
- **Coreference resolution:** It refers to a sentence or larger set of text that determines which words refer to the same objects, example of this is concerned with matching up pronouns with the nouns or names that they link to.
- **Discourse analysis:** The task is identifying the discourse structure of connected text, i.e. the nature of the discourse relationships between sentences e.g. elaboration, explanation, contrast. Another possible task is recognizing and classifying the speech acts in a large set of text e.g. yes and no questions, content question, statements, assertion etc.
- **Machine translation:** Automatically translates text from one human language to another.

Morphological segmentation: Separate words into individual morphemes and identify the class of the morphemes. The difficulty of this task depends greatly on the complexity of the morphology i.e. the structure of words of the language being considered.

- **Named entity recognition (NER):** It describes a stream of text, determine which items in the text relates to proper names, such as people or places, and what the type of each such name or place we are referring to is.

• Other tasks includes

1. **a) Optical character recognition (OCR):** Given an image representing printed text, helps in determining the corresponding or related text.
2. **b) Part-of-speech tagging:** It describes a sentence, determines the part of speech for each word.
3. **c) Parsing:** It refers to the parse tree (grammatical analysis or evaluation) of a given sentence
4. **d) Question answering:** It answers a given human language question and determines its answer. [6]

1.3 Parts of Speech Tagging (POST Tagging)

1.3.1 History of Post Tagging

Brown corpus: Research on part-of-speech tagging has been closely tied to corpus linguistics. The first major corpus of English for computer analysis was the Brown Corpus developed at Brown University by Henry Kučera and W. Nelson Francis, in the mid-1960s. It consists of about 1,000,000 words of running English prose text, made up of 500 samples from randomly chosen publications. Each sample is 2,000 or more words (ending at the first sentence-end after 2,000 words, so that the corpus contains only complete sentences).

The Brown Corpus was painstakingly "tagged" with part-of-speech markers over many years. A first approximation was done with a program by Greene and Rubin, which consisted of a huge handmade list of what categories could co-occur at all. For example, article then noun can occur, but article then verb (arguably) cannot. The program got about 70% correct. Its results were repeatedly reviewed and corrected by hand, and later users sent in errata so that by the late 70s the tagging was nearly perfect (allowing for some cases on which even human speakers might not agree). this corpus has been used for innumerable studies of word-frequency and of part-of-speech and inspired the development of similar "tagged" corpora in many other languages. Statistics derived by analyzing it formed the basis for most later part-of-speech tagging systems, such as CLAWS (linguistics) and VOLSUNGA. However, by this time (2005) it has been superseded by larger corpora such as the 100 million word British National Corpus, even though larger corpora are rarely so thoroughly curated. for some time, part-of-speech tagging was considered an inseparable part of natural language processing, because there are certain cases where the correct part of speech cannot be decided without understanding the semantics or even the pragmatics of the context. This is extremely

expensive, especially because analyzing the higher levels is much harder when multiple part-of-speech possibilities must be considered for each word.

Use of hidden Markov models: In the mid-1980s, researchers in Europe began to use hidden Markov models (HMMs) to disambiguate parts of speech, when working to tag the Lancaster-Oslo-Bergen Corpus of British English. HMMs involve counting cases (such as from the Brown Corpus) and making a table of the probabilities of certain sequences. For example, once you've seen an article such as 'the', perhaps the next word is a noun 40% of the time, an adjective 40%, and a number 20%. Knowing this, a program can decide that "can" in "the can" is far more likely to be a noun than a verb or a modal. The same method can, of course, be used to benefit from knowledge about the following words. more advanced ("higher-order") HMMs learn the probabilities not only of pairs but triples or even larger sequences. So, for example, if you've just seen a noun followed by a verb, the next item may be very likely a preposition, article, or noun, but much less likely another verb**.**

When several ambiguous words occur together, the possibilities multiply. However, it is easy to enumerate every combination and to assign a relative probability to each one, by multiplying together the probabilities of each choice in turn. The combination with the highest probability is then chosen. The European group developed CLAWS, a tagging program that did exactly this and achieved accuracy in the 93–95% range.

It is worth remembering, as Eugene Charniak points out in Statistical techniques for natural *language parsing* (1997), that merely assigning the most common tag to each known word and the tag "proper noun" to all unknowns will approach 90% accuracy because many words are unambiguous, and many others only rarely represent their less-common parts of speech.

CLAWS pioneered the field of HMM-based part of speech tagging but were quite expensive since it enumerated all possibilities. It sometimes had to resort to backup methods when there were simply too many options (the Brown Corpus contains a case with 17 ambiguous words in a row, and there are words such as "still" that can represent as many as 7 distinct parts of speech (DeRose 1990, p. 82)). HMMs underlie the functioning of stochastic taggers and are used in various algorithms one of the most widely used being the bi-directional inference algorithm. Dynamic programming methods In 1987, Steven DeRose and Ken Church independently developed dynamic programming algorithms to solve the same problem in vastly less time. Their methods were similar to the Viterbi algorithm known for some time in other fields. DeRose used a table of pairs, while Church used a table of triples and a method of estimating the values for triples that were rare or nonexistent in the Brown

Corpus (an actual measurement of triple probabilities would require a much larger corpus). Both methods achieved an accuracy of over 95%. DeRose's 1990 dissertation at Brown University included analyses of the specific error types, probabilities, and other related data, and replicated his work for Greek, where it proved similarly effective.

These findings were surprisingly disruptive to the field of natural language processing. The accuracy reported was higher than the typical accuracy of very sophisticated algorithms that integrated part of speech choice with many higher levels of linguistic analysis: syntax, morphology, semantics, and so on. CLAWS, DeRose's and Church's methods did fail for some of the known cases where semantics is required, but those proved negligibly rare. This convinced many in the field that part-of-speech tagging could usefully be separated from the other levels of processing; this, in turn, simplified the theory and practice of computerized language analysis and encouraged researchers to find ways to separate other pieces as well. Markov Models are now the standard method for the part-of-speech assignment. Unsupervised taggers, the methods already discussed involve working from a pre-existing corpus to learn tag probabilities. It is, however, also possible to bootstrap using "unsupervised" tagging. Unsupervised tagging techniques use an untagged corpus for their training data and produce the tag set by induction. That is, they observe patterns in word use, and derive part-of-speech categories themselves. For example, statistics readily reveal that "the", "a", and "an" occur in similar contexts, while "eat" occurs in very different ones. With sufficient iteration, similarity classes of words emerge that are remarkably similar to those human linguists would expect; and the differences themselves sometimes suggest valuable new insights. These two categories can be further subdivided into rule-based, stochastic, and neural approaches. Other taggers and methods: Some current major algorithms for part-of-speech tagging include the Viterbi algorithm, Brill tagger, Constraint Grammar, and the Baum-Welch algorithm (also known as the forward-backward algorithm). Hidden Markov model and visible Markov model taggers can both be implemented using the Viterbi algorithm. The rule-based Brill tagger is unusual in that it learns a set of rule patterns, and then applies those patterns rather than optimizing a statistical quantity. Unlike the Brill tagger where the rules are ordered sequentially, the POS and morphological tagging toolkit RDR POS Tagger stores rule in the form of a ripple-down rules tree.

Many machine learning methods have also been applied to the problem of POS tagging. Methods such as SVM, maximum entropy classifier, perceptron, and nearest-neighbor have all been tried, and most can achieve accuracy above 95%.

A direct comparison of several methods is reported (with references) at the ACL Wiki. This comparison uses the Penn tag set on some of the Penn Treebank data, so the results are directly comparable. However, many significant taggers are not included (perhaps because of the labor involved in reconfiguring them for this particular dataset). Thus, it should not be assumed that the results reported here are the best that can be achieved with a given approach; nor even the best that *have* been achieved with a given approach.

In 2014, a paper reporting using the structure regularization method for part-of-speech tagging, achieving 97.36% on the standard benchmark dataset. [12]

1.3.2 Definitions of Post Tagging

Part-of-speech (POS) tagging simply means labeling words with their appropriate Part-Of-Speech so it explains how a word is used in a sentence

The most basic models in natural language processing are based on Bag of Words which is not the best solution as it fails to capture any syntactic relations between the words.so we can improve this bag of words model using a technique like pos tagging; also, **Post Tagging** is a process to mark up the words in text format for a particular part of a speech based on its definition and context. It is responsible for text reading in a language and assigning some specific token (Parts of Speech) to each word. It is also called grammatical tagging.

Tagging is a kind of classification that may be defined as the automatic assignment of description to the tokens. Here the descriptor is called tag, which may represent one of the part-of-speech, semantic information and so on.**[13]**

Word	Translation	POS
مدرسة	School	NOUN
يقراء	Read	VERB
صغير	Small	ADJ
سعيد	Happily	Adverb

Figure.1.2.Example for Post Tagging [14]

1.3.3 Techniques for POS Tagging

There are various techniques that can be used for POS tagging such as

1. **Rule-based POS tagging**: The rule-based POS tagging models apply a set of handwritten rules and use contextual information to assign POS tags to words.

These rules are often known as context frame rules. One such rule might be: "If an ambiguous/unknown word ends with the suffix 'ing' and is preceded by a Verb, label it as a Verb".

2. **Transformation Based Tagging:** The transformation-based approaches use a pre-defined set of handcrafted rules as well as automatically induced rules that are generated during training.

3. **Deep learning models**: Various Deep learning models have been used for POS tagging such as Meta-Bi LSTM which have shown an impressive accuracy of around 97 percent.

4. **Stochastic (Probabilistic) tagging**: A stochastic approach includes frequency, probability or statistics. The simplest stochastic approach finds out the most frequently used tag for a specific word in the annotated training data and uses this information to tag that word in the unannotated text. But sometimes this approach comes up with sequences of tags for sentences that are not acceptable according to the grammar rules of a language. One such approach is to calculate the probabilities of various tag sequences that are possible for a sentence and assign the POS tags from the sequence with the highest probability. Hidden Markov Models (HMMs) are probabilistic approaches to assign a POS Tag.[15]

1.3.4 Post Tagging with Hidden Markov Model

HMM (Hidden Markov Model) is a Stochastic technique for POS tagging. Hidden Markov models are known for their applications to reinforcement learning and temporal pattern recognition such as speech, handwriting, gesture recognition, musical score following, partial discharges, and bioinformatics.

An HMM model may be defined as the doubly-embedded stochastic model, where the underlying stochastic process is hidden. This hidden stochastic process can only be observed through another set of stochastic processes that produces the sequence of observations.

For example, a sequence of hidden coin tossing experiments is done and we see only the observation sequence consisting of heads and tails. The actual details of the process - how many coins used, the order in which they are selected - are hidden from us. By observing this

sequence of heads and tails, we can build several HMMs to explain the sequence. Following is one form of Hidden Markov Model for this problem, we assumed that there are two states in the HMM and each of the state corresponds to the selection of different biased coin. Following matrix gives the state transition probabilities.[15]

1.3.5 Part of Arabic Pos Tagging

The Arabic POS inventory consists of three main categories Dahdah, 1989; Ghalayini, 2013): Noun (اسم) Verb(فعل) and Particle (حرف)Each one of these categories has many subcategories.

1.3.5.1 Noun Part-of-Speech: A noun conveys lexical meaning (as opposed to grammatical meaning) but gives no indication of time. Nouns are classified according to five main subcategories: definiteness (definite or indefinite), number (Singular, Dual, Plural), gender (Masculine, Feminine), inflection (Derived or Primitive) and declension (Declined or Invariable). Each class contains several tags. **Figure 1.3** presents the hierarchical levels of the noun categories and their tags.

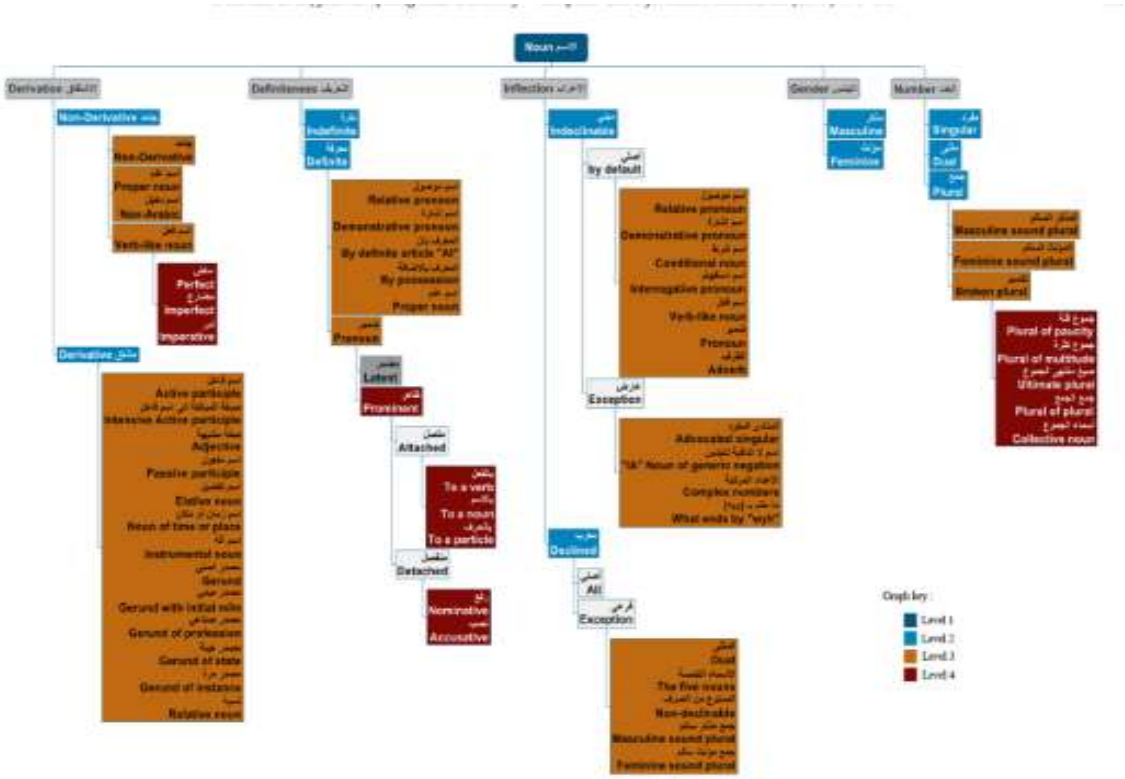


Figure1.3 Noun Part of Speech

- **1.3.5.2 Verb Part-of-Speech:** A verb conveys lexical meaning (as opposed to grammatical meaning), and gives indication of time. Verbs are classified according to two main subcategories: the perfect verb (which consists of another six subcategories: declension, conjugation, inflection, transitivity, voice, and vocalic) and the imperfect verb (which consists of another four subcategories: declension, conjugation, inflection, and vocalic). Further, each category consists of several tags. **Figure 1.4** displays the hierarchical levels of the verb categories and their tags.

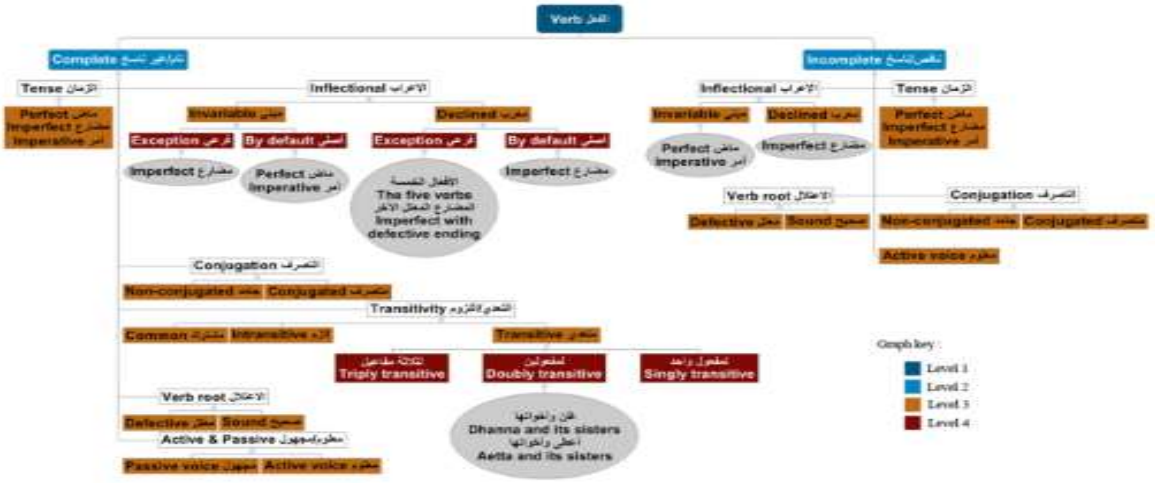


Figure1.4 Verb Part of Speech

- **1.3.5.3 Particle Part-of-Speech:** A particle conveys no lexical meaning, which is conveyed through the parts of speech it relates to. Particles are classified according to two main subcategories: common and specific. The specific class consists of two others subcategories (specific to noun and specific to verb). Moreover, each class consists of several tags. **Figure1.5** displays the hierarchical levels of the particle categories and their tags. [16]



Figure1.5. Particle Part of Speech

1.3.6 The Post Tagging with NLP

the noun and verb phrases. Figure1shows an example of the parse tree of an Arabic sentence after applying the steps one and two.

For the noun clauses, the algorithm looks at the word sequence of each (Bigram) word pair and adds the words matching the following rule (pattern) to the candidate list of concepts.

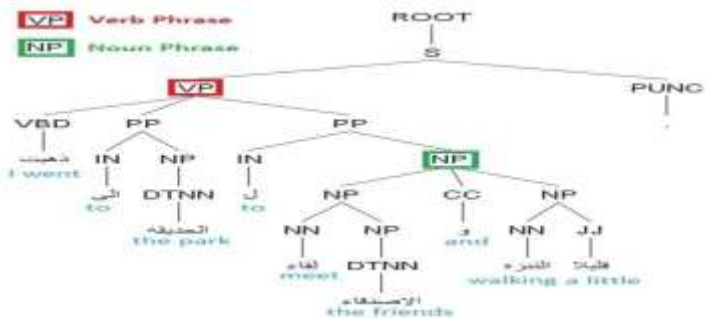


Figure1.6 Example Arabic Post Tagging in NLP [17]

1.4Deep Learning

1.4.1 History of Deep learning

Since the 1950s, a small subset of Artificial Intelligence (AI), often called Machine Learning (ML), has revolutionized several fields in the last few decades. Neural Networks (NN) is as field of ML, and it was this subfield that spawned Deep Learning (DL). Since its inception DL has been creating ever larger disruptions, showing outstanding success in almost every application domain. Figure1 shows the taxonomy of AI. DL which uses either deep

architectures of learning or hierarchical learning approaches), is a class of ML developed largely from 2006 onward. Learning is a procedure consisting of estimating the model parameters so that the learned model(algorithm) can perform a specific task. For example, in Artificial Neural Networks (ANN), the parameters are the weight matrices. DL, on the other hand, consists of several layers in between the input and output layer which allows for many stages of non-linear information processing units with hierarchical architectures to be present that are exploited for feature learning and pattern classification. Learning methods based on representations of data can also be defined as representation learning. Recent literature states that DL based representation learning involves a hierarchy of features or concepts, where the high-level concepts can be defined from the low-level ones and low-level concepts can be defined from high-level ones. In some articles, DL has been described as a universal learning approach that is able to solve almost all kinds of problems in different application domains. In other words, DL is not task specific.[18]

1.4.2 Definition of Deep Learning

Deep learning is a form of machine learning that enables computers to learn from experience and understand the world in terms of a hierarchy of concepts. Because the computer gathers knowledge from experience, there is no need for a human computer operator formally to specify all of the knowledge needed by the computer. The hierarchy of concepts allows the computer to learn complicated concepts by building them out of simpler ones.[19]

1.4.3 How Deep Learning Works

Deep learning neural networks, or artificial neural networks, attempts to mimic the human brain through a combination of data inputs, weights, and bias. These elements work together to accurately recognize, classify, and describe objects within the data.

Deep neural networks consist of multiple layers of interconnected nodes, each building upon the previous layer to refine and optimize the prediction or categorization. This progression of computations through the network is called forward propagation. The input and output layers of a deep neural network are called *visible* layers. The input layer is where the deep learning model ingests the data for processing, and the output layer is where the final prediction or classification is made.

Another process called backpropagation uses algorithms, like gradient descent, to calculate errors in predictions and then adjusts the weights and biases of the function by moving backwards through the layers in an effort to train the model. Together, forward propagation and backpropagation allow a neural network to make predictions and correct for any errors accordingly. Over time, the algorithm becomes gradually more accurate.

The above describes the simplest type of deep neural network in the simplest terms. However, deep learning algorithms are incredibly complex, and there are different types of neural networks to address specific problems or datasets. For exemple :

- **Convolutional neural networks (CNNs):**used primarily in computer vision and image classification applications, can detect features and patterns within an image, enabling tasks, like object detection or recognition. In 2015, a CNN bested a human in an object recognition challenge for the first time.
- **Recurrent neural network (RNNs):** are typically used in natural language and speech recognition applications as it leverages sequential or times series data[21]

1.4.4 Deep learning Applications

Deep learning is currently being utilized in almost every industry,here are some of the examples:

1) **Machine Translation**: Thanks to deep learning, we have access to different translation services. One of the most popular one, Google Translate helps its user to easily translate a language. No need for complicated steps, deep learning has helped this application improve tremendously. From just typing a word, to pronouncing a word, it sure is a big improvement from what is started out to be.
2) **Virtual Assistants**: The most popular application of deep learning is virtual assistants. From the likes Siri, Alexa and Google Assistant, these digital assistants are heavily reliant on deep learning to understand its user and at the same time give the appropriate response in a natural manner. Virtual assistants use deep learning to know more about their subjects ranging from your favorite places to your favorite songs.
3) **Customer Service**: "Chatbots". Deep learning has played a major role in helping businesses by improving customer service and making it more accessible to its customers. Trained with large amount of data, chatbots can understand customer requests and also

guide the customer and help them resolve their problem in a human- like manner. Plus, it saves up customers time and brings down the costs of business.

4) **Self-Driving Cars**: Because of deep learning, self-driving cars do exist and is just going to keep on improving over time. Although it hasn't been made available to public yet, The Uber Artificial Intelligence Labs at Pittsburgh is not only working on making driverless cars, but also integrating food delivery option with the use of this new invention.
5) **News Aggregation**: Deep learning helps develop classifiers that can detect fake or biased news and remove it from your feed.
6) **Digital Marketing**: The application of deep learning in digital marketing helps marketing professionals gauge the effectiveness of their campaigns.
7) **Natural Language Processing**: One of the hardest tasks that human can learn is understanding the complexities associated with language. Whether its semantics, syntax, tonal nuances, expressions or even sarcasms, humans find it hard to easily grasps learning a language. In deep learning, machines are trained to achieve the same thing and develop human-like response and personalized expressions. It is also trying to catch linguistic nuances and answer questions. It is also training machines to build phrases and sentences and capture local word semantics with word embedding.
8) **Colorizing Videos and Images**: Thanks to deep learning frameworks, machines can flaunt their creativity by adding color to old black and white photos and videos.
9) **Personalization**: Have you ever felt that Spotify, Amazon, E-bay, Alibaba and Netflix recommends you exactly the things you like? Deep learning is the main reason for that. It plays a major role in understanding its consumers' behavior and generating recommendations to help them make choices for product and services.
10) **Healthcare**: Deep learning has been playing an important role in medical diagnosis and research. It helps with diagnosis of life-threatening diseases, pathology results and treatment cause standardization and understanding genetics to predict future risks of diseases. Readmissions are a huge problem in the healthcare industry and deep learning is helping to combat this.
11) **Fraud Detection**: The banking and financial sector also benefit from deep learning application especially money transaction is going digital. Furthermore, there are applications under development that will help detect fraudulent credit cards saving billions of dollars of in recovery and insurance of financial institutions.[22]

1.4.5 Deep Learning Types

In this section, some types of Deep Learning are presented:

1.4.5.1 Feed Forward Neural Network

This type of neural network is the very basic neural network where the flow control occurs from the input layer and goes towards the output layer.

These kinds of networks are only having single layers or only 1 hidden layer.

Since the data moves only in 1 direction there is no backpropagation technique in this network.

In this network, the sum of the weights present in the input is fed into the input layer.

These kinds of networks are used in the facial recognition algorithm using computer vision.[23]

1.4.5.2 Convolutional Neural Networks

A convolutional neural network (CNN) is another variant of the feedforward multilayer perceptron. It is a type of feedforward neural network, where the individual neurons are ordered in a way that they respond to all overlapping regions in the visual area.

This is a type of network that constitutes of learning weight and biases. Every input layer is composed of a set of neurons where at every input a dot product is performed and move further with the concept of non-linearity. It is a kind of fully-connected type of network that uses SVM/SoftMax function as a loss function.

CNNs form the core of most sophisticated fancy computer vision applications, facial security features, and automatic number plate recognition. [24]

1.4.5.3 Recurrent Neural Networks

The convolutional model works on a fixed number of inputs, generates a fix-sized vector as output with a predefined number of steps. The recurrent networks allow us to operate over sequences of vectors in input and output. In the case of recurrent neural network, the connection between units forms a directed cycle. Unlike the traditional neural network, the

recurrent neural network input and output are not independent but related. Further, the recurrent neural network shares the standard parameters at every layer. One can train the recurrent network in a way that is like the traditional neural network using the backpropagation method.

Here, calculation of gradient depends not on the current step but previous steps also. A variant called a bidirectional recurrent neural network is also used for many applications. The bidirectional neural network considers not only the previous but also the expected future output. In two-way and straightforward recurrent neural networks, deep learning can be achieved by introducing multiple hidden layers. Such deep networks provide higher learning capacity with lots of learning data. Speech, image processing, and natural language processing are some of the candidate areas where recurrent neural networks can be used. [25]

1.4.5.4 Auto Encoders

An Auto Encoder is an artificial neural network that is capable of learning various coding patterns. The simple form of the autoencoder is just like the multilayer perceptron, containing an input layer or one or more hidden layers, or an output layer. The significant difference between the typical multilayer perceptron and feedforward neural network and autoencoder is in the number of nodes at the output layer. In the case of the autoencoder, the output layer contains the same amount of nodes as in the input layer. Instead of predicting target values as per the output vector, the autoencoder has to predict its inputs.[26]

1.4.5.5 Restricted Boltzmann Machine

A restricted Boltzmann machine (RBM) is a generative stochastic artificial neural network that can learn a probability distribution over its set of inputs.

It is a kind of stochastic neural network that consists of one layer of visible units, one layer of hidden units and a bias unit. The architecture is developed in such a way that each visible unit is connected to all hidden units and bias units are connected to all visible and hidden units. During the learning process, the restriction is developed so that no visible unit is connected with any visible units and no hidden unit is connected with any hidden unit.

1.4.5.6 Deep Belief Networks (DBN)

A deep belief network is a solution to the problem of handling non-convex objective functions and local minima while using the typical multilayer perceptron. It is an alternative type of deep learning consisting of multiple layers of latent variables with connection between the layers. The deep belief network can be viewed as restricted Boltzmann machines (RBM), where each subnetwork's hidden layer acts as the visible input layer for the adjacent layer of the network. It makes the lowest visible layer a training set for the adjacent layer of the network. This way, each layer of the network is trained independently and greedily. The hidden variables are used as the observed variables to train each layer of the deep structure.[26]

1.4.5.7 Recursive Neural Network

A recursive neural network is a kind of deep neural network created by applying the same set of weights recursively over a structured input, to produce a structured prediction over variable-size input structures, or a scalar prediction on it, by traversing a given structure in topological order. Recursive neural networks, sometimes abbreviated as RNNs, have been successful, for instance, in learning sequence and tree structures in natural language processing, mainly phrase and sentence continuous representations based on word embedding.[27]

1.4.6 Popular Deep learning Models

1.4.6.1 Deep Feed-Forward Neural Network (FNN)

Feed-forward neural networks (FNNs) were one of the first and simplest components applied to learn from data using DL paradigms. Inspired from human brain structure, neural networks are formed by connecting a number of levels of nodes called "perceptron" (Rosenblatt, 1958) with weighted connections. Rosenblatt perceptron as it is illustrated in (Figure 6) is a node that takes several binary inputs, x_1, x_2… and produces a single binary output. Resulting networks are otherwise called multi-layer perceptron (MLP) networks structured as input, hidden and output layers.[28]

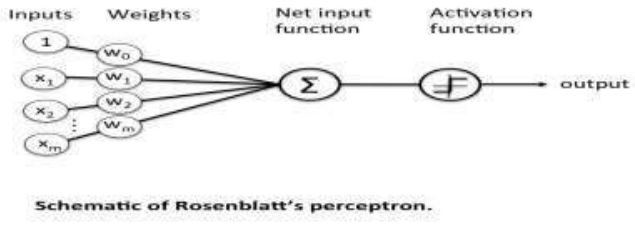


Figure.1.7 Schematic of Rosenblatt perceptron.

The difference between shallow and deep neural networks is the number of layers composing the architecture. Because unlike shallow networks composed of 1 to 3 layers with only a single hidden layer, Deep FNNs are deeper and composed of many hidden layers (Y Bengio, 2009). Figure 8 below illustrates a deep feed-forward neural network composed of four layers where the middle layers are hidden layers.

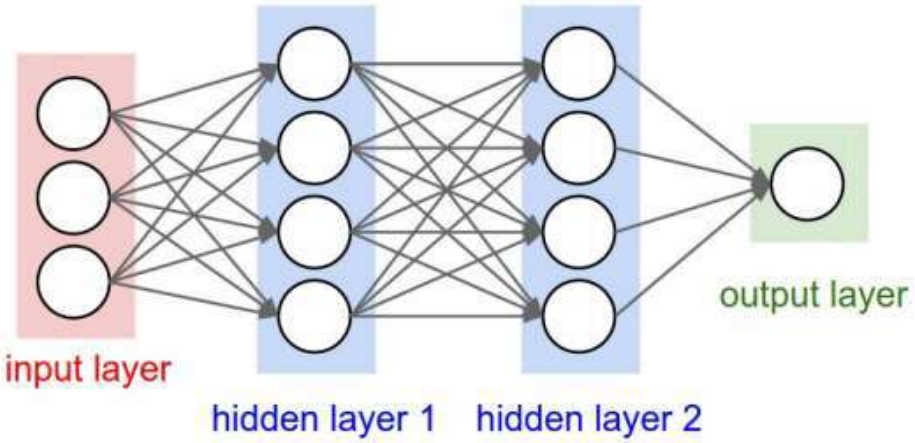


Figure1.8 Deep Feed Forward Neural Network Architecture.

Neural networks are fed with input values and make certain nodes on the input layer to be activated; the resulted signal is broadcasted to the subsequent level. The activation of subsequent level nodes happens if the weighted signal generated at a given level is greater than a given threshold. In first time, weights are randomly initialized then adjusted during training process in order to minimize a certain objective function. This family of networks has been proved to be useful for pattern classification, but less suitable for labeling sequences since it does not take into account the sequence of input data (Dessí et al., 2020)[29]

1.4.6.2 Recurrent Neural Networks (RNNs)

RNNs (Rumelhart et al., 1986) are used for sequential data which involves variable length inputs or outputs such as speech and language and they are adapted for processing data as a sequence (I Goodfellow et al., 2016). In contrast to FNNs which commonly pass the input

data directly from input to output nodes, RNNs have cyclic or recurrent connections among nodes of distinct levels. This makes possible to model the output of the network by taking into account the entire history of the received input data by maintaining in their hidden units an internal memory states or 'state vector'. Recurrent connections connect past data with the one that is currently being processed, simulating a state memory. The forward pass is similar to the one in FNNs. The unique difference is that the activation of a node depends on both the current input and the previous status of the hidden layers. The text is fed into the network by means of vector representations that are recurrently processed. This means that RNNs view a sample text as an ordered sequence of word identifiers, differently from common FNNs working on hand-crafted inputs, e.g., Bag of Words (BOW).(Dessí et al., 2020; LeCun et al., 2015; Nankani et al., 2020) Figure 9 shows a simple architecture of RNN where Xi are input tokens, Wxh are weights for the connections between input layer and hidden layers, Why are weights for connections between hidden layer and output layer, ai are activations of hidden layer neurons and Y_i are outputs.[30]

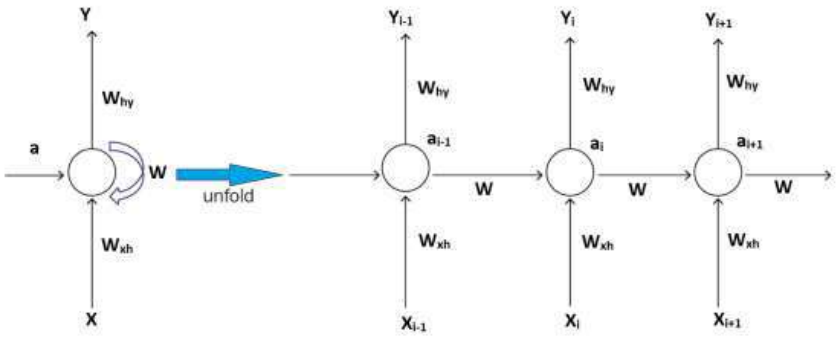


Figure.1.9 RNN Architecture.

In a wide range of applications, data can present patterns from the past to the future and vice versa. For instance, for classifying the sections of a given story, it could be useful to have access to both future and past sections. However, the future content of a text is ignored by common FNNs and RNNs, as they work sequentially. Bidirectional RNNs (BiRNNs) let the network, at a given point in time, to take information from both earlier and later data in the sequence, going beyond the exposed limitations. The idea behind this kind of networks consists of presenting the training data forward and backward by two hidden RNNs which are then combined into a common output layer (Figure 10). This strategy makes possible to find patterns that can be learned from both past and future history of data (Dessí et al., 2020).

Although the main purpose of RNNs is to learn long-term dependencies, theoretical and empirical evidence shows that it is difficult to learn and to store information for very long. In

addition, the training of RNNs proved to be problematic because the backpropagated gradients either grow or shrink at each time step, so over many time steps they typically explode or vanish. (YoshuaBengio et al., 1994)[31]

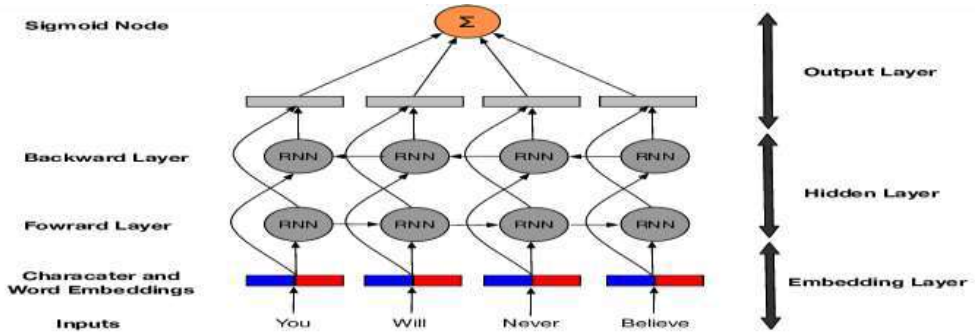


Figure1.10 Bidirectional RNN Architecture with forward and backward hidden RNN-layers.

1.4.6.3 Long Short-Term Memory (LSTM) Network

Long short-term memory (LSTM) networks (Hochreiter&Schmidhuber, 1997) are an RNN extension designed to work on sequence problems by being capable of learning long-term dependencies and that has achieved state-of-the-art results on challenging prediction tasks. LSTM networks employ recurrent connections and add memory blocks in their recurrent hidden layers. These memory blocks save the current temporal state during training and make possible to learn temporal observations hidden in the input data. The fact of using connections as a memory implies that the output of a LSTM network depends on the entire history of the training data, not only on the current input sample. Moreover, using memory blocks allows relating the current data that is being processed with the data processed long before, solving the problem experienced by common RNNs. For this reason, LSTM networks have had a positive impact on sequence prediction tasks. As stated for RNN, a bidirectional layer using two hidden LSTMs can be leveraged to process data both forward and backward. A LSTM has three gates and a memory cell state from which information can flow from one module to another with either some linear interactions or unchanged. The gates are means to remove or add information to the cell states, and they consist of a multiplication operation and a sigmoid layer. The three gates in a LSTM module are input, forget and output gates:

Forget Gate: decides what portion of the previous data to discard.

Input Gate: conjectures what new knowledge is to be updated in the current module.

Output Gate: reflects output of the present module. (Dessí et al., 2020;Nankani et al., 2020)[30]

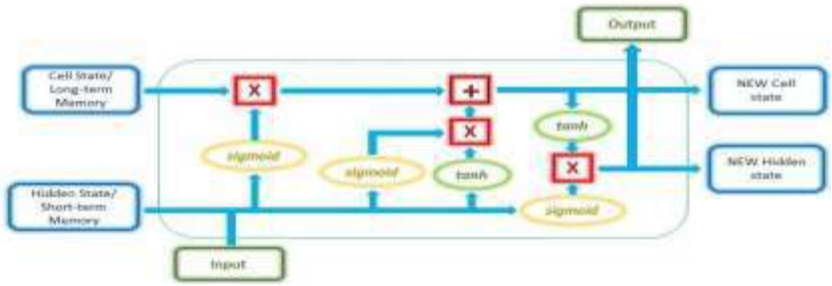


Figure1.11 LSTM unit architecture.[30]

1.4.6.4 Gated Recurrent Units (GRUs)

They are a variant of LSTMs equipped to avoid vanishing and exploding gradient problem with a simpler internal architecture. Introduced by (Cho et al., 2014), GRUs consists of two gates: **Update Gated:** is designed to help learn long-term dependencies from a series of input. **Reset Gate:** is intended to learn short-term dependencies. Unlike LSTMs, GRUs does not maintain an internal memory state different from their hidden state. (Ahmet & Abdullah, 2020) Figure 12 shows the architecture of Gated Recurrent Unit.

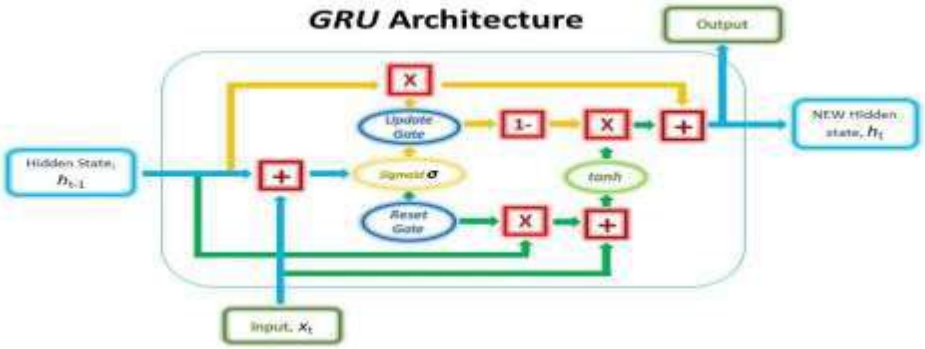


Figure1.12 GRU Architecture.

1.4.6.5 Convolutional Neural Network (CNN)

Convolutional neural networks (LeCun et al., 1989) or CNNs, are a specialized kind of neural network for processing data that has a known grid-like typology. Examples includes time series data, which can be thought as 1-D grid taking samples at regular time intervals, and image data, which can be thought as 2-D grid of pixels. The name "convolutional neural networks" indicates that the network employs a mathematical operation called *"convolution"* which is a specialized kind of linear operation. To be specific, convolutional networks are simply neural network that uses convolutions in place of general matrix multiplication in at least one of their layers. The motivation behind using convolutional networks in place of traditional neural networks is reducing intra-modular network interactions, sharing of parameters and learning of equivariant representations.

Unlike traditional neural networks where every output unit interacts with every input unit, convolutional networks typically have sparse interactions. This is accomplished by using a kernel having a smaller size than input which enables selecting only small, meaningful input features. Parameter sharing in CNNs is related to that in the convolution operation, the same kernel is replicated across the entire input rather than using each element of the weight matrix exactly once for computing the output of a layer as it is applied in traditional nets. This particular form of parameter sharing causes the layer to have a property called *equivariance* to translation which means that if the input changes, the output changes in the same way. In other words, if we apply a transformation to the input, then apply convolution, the result will be the same as if we applied convolution to the input, then applied the transformation to the output. When processing time series data, this means that convolution produces a sort of timeline that shows when different features appear in the input. If we move an event later in time in the input, the exact same representation of it will appear in the output, just later in time. Similarly with images, convolution creates a 2-D map of where certain features appear in the input. If we move the object in the input, its representation will move the same amount in the output. This is useful for when we know that same local function is useful everywhere in the input. (I Goodfellow et al., 2016) Convolutional neural networks (CNNs) have been broadly applied in the computer vision field for treating images, but they are very useful in text classification, especially for shorter-length texts. Moreover, they can capture with high performance relevant subsequences in texts in word-level as proposed in (Kim, 2014), and character-level as suggested in (Zhang et al., 2015). [30]

1.4.7 How Convolution Neural Networks Work

The architecture of a typical CNN is structured as a series of stages of several kinds of layers:

1.4.7.1 Convolutional Layers Units in a convolutional layer are organized in feature maps, within which each unit is connected to local patches in the feature maps of the previous layer through a set of weights called a filter bank. The result of this local weighted sum is then passed through a non-linearity such as a ReLU. A complete convolutional layer is composed of several feature maps (with different weight vectors), so that multiple features can be extracted at each location of the input array. The role of this kind of layers is to detect local conjunctions of features from the previous layer.

1.4.7.2 Pooling or Subsampling Layers: They reduce spatial resolution of the feature map generated by convolution layer by computing a max or an average from $N \times N$ feature map

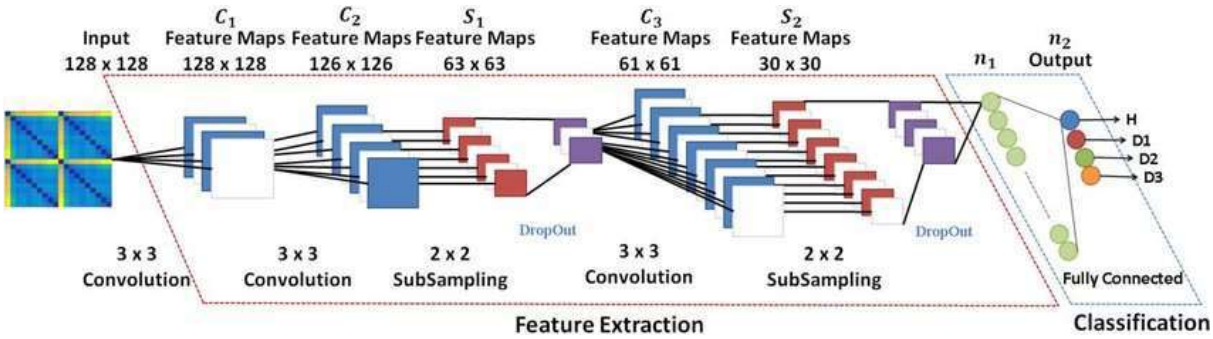
clusters (the process of convolutional and pooling usually repeats several times). The role of the pooling layer is to merge semantically similar features into one.

1.4.7.3 Fully Connected Input Layer:"Flattens" the outputs generated by previous layers to turn them into single vector that can be used as an input for the next layer.

1.4.7.4 Fully Connected Layer: It applies weights over the input generated by the feature analysis to predict an accurate label.

1.4.7.5 Fully Connected Output Layer: It generates the final probabilities to determine a class for the input. (LeCun et al., 1998, 2015)

Figure 1.13 below shows a typical CNN architecture highlighting the layers.



Figur1.13 Generic representation of CNN architecture

IBM has a rich history with machine learning. One of its own, Arthur Samuel, is credited for coining the term, "machine learning" with his research (PDF, 481 KB) (link resides outside IBM) around the game of checkers. Robert Nealey, the self-proclaimed checkers master, played the game on an IBM 7094 computer in 1962, and he lost to the computer. Compared to what can be done today, this feat almost seems trivial, but it's considered a major milestone within the field of artificial intelligence. Over the next couple of decades, the technological developments around storage and processing power will enable some innovative products that we know and love today, such as Netflix's recommendation engine or self-driving cars.

Machine learning is an important component of the growing field of data science. Through the use of statistical methods, algorithms are trained to make classifications or predictions, uncovering key insights within data mining projects. These insights subsequently drive decision making within applications and businesses, ideally impacting key growth metrics. As big data continues to expand and grow, the market demand for data scientists will increase, requiring them to assist in the identification of the most relevant business questions and subsequently the data to answer them.

1.4.8 Deep Learning Performance

The performance of machine and deep learning models increases proportionately with the amount of training data available. Since to make such models generalize better on new data, it is recommended to train them on more data; thus, advances in hardware and the exponential increase in the amount of training data made available are the main reasons of the widespread impact of deep learning.(I Goodfellow et al., 2016; Nankani et al., 2020)[35]

1.5 Neural Networks (NN)

1.5.1 A Brief History of Neural Networks

Neural networks date back to the early 1940s when mathematicians Warren McCulloch and Walter Pitts built a simple algorithm-based system designed to emulate human brain function. Work in the field accelerated in 1957 when Cornell University's Frank Rosenblatt conceived of the perceptron, the groundbreaking algorithm developed to perform complex recognition tasks. During the four decades that followed, the lack of computing power necessary to process large amounts of data put the brakes on advances. In the 2000s, thanks to the advent of greater computing power and more sophisticated hardware, as well as to the existence of vast data sets to draw from, computer scientists finally had what they needed, and neural networks and AI took off, with no end in sight. To understand how much the field has expanded in the new millennium, consider that ninety percent of internet data has been created since 2016. That pace will continue to accelerate, thanks to the growth of the Internet of Things (IoT).[36]

1.5.2 Neural Network Definition

A neural network is a computing model whose layered structure resembles the networked structure of neurons in the brain. It features interconnected processing elements called neurons that work together to produce an output function. Neural networks are made of input and output layers/dimensions, and in most cases, they also have a hidden layer consisting of units that transform the input into something that the output layer can use.[37]

1.5.3 Architecture of Neural Network

Neural Networks are complex structures made of artificial neurons that can take in multiple inputs to produce a single output. This is the primary job of a Neural Network – to transform input into a meaningful output. Usually, a Neural Network consists of an input and output layer with one or multiple hidden layers within.

In a Neural Network, all the neurons influence each other, and hence, they are all connected. The network can acknowledge and observe every aspect of the dataset at hand and how the different parts of data may or may not relate to each other. This is how Neural Networks are capable of finding extremely complex patterns in vast volumes of data. [38]

1.5.4 Why Recurrent Neural Networks?

Recurrent neural networks were created because there were a few issues in the feed- forward neural network:

- Cannot handle sequential data.
- Considers only the current input.
- Cannot memorize previous inputs.

The solution to these issues is the Recurrent Neural Network (RNN). An RNN can handle sequential data, accepting the current input data, and previously received inputs. RNNs can memorize previous inputs due to their internal memory.[39]

1.5.5 Difference between Neural Network and Deep Learning Neural Network

Since Deep Learning and Neural Networks are so deeply intertwined, it is difficult to tell them apart from each other on the surface level. However, there's a significant difference between Deep Learning and Neural Networks.

While Neural Networks use neurons to transmit data in the form of input values and output values through connections, Deep Learning is associated with the transformation and extraction of feature which attempts to establish a relationship between stimuli and associated neural responses present in the brain.[39]

Neural Network is less complex and requires more information about features for performing feature selection and feature engineering method. On the other hand, Deep Learning Neural Network does not require any information about features rather they perform optimum model tuning and model selection on their own.

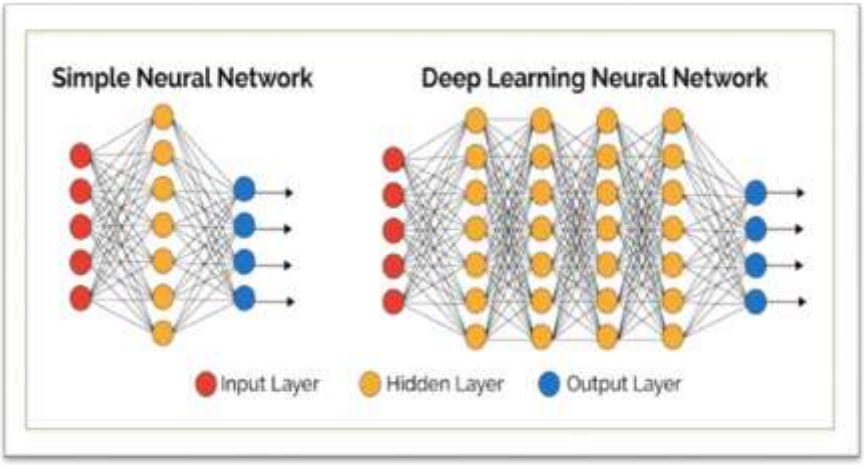


Figure1.14 Architecture of Simple Neural Network and Deep Learning Neural Network.[40]

1.6 Recurrent Neural Networks

1.6.1 Recurrent Neural Networks Definition

A recurrent neural network (RNN) is a type of artificial neural network which uses sequential data. These deep learning algorithms are commonly used for ordinal or temporal problems, such as language translation, natural language processing (nlp), speech recognition, and image captioning; they are incorporated into popular applications such voice search, and Google Translate. recurrent neural networks utilize training data to learn. They are distinguished by their “memory” as they take information from prior inputs to influence the current input and output. While traditional deep neural networks assume that inputs and outputs are independent of each other, the output of recurrent neural networks depend on the prior elements within the sequence. While future events would also be helpful in determining the output of a given sequence.[41]

recurrent neural networks (RNN) are the state of the art algorithm for sequential data and are used by Apple's Siri and Google's voice search. It is the first algorithm that remembers its input, due to an internal memory, which makes it perfectly suited for machine learning problems that involve sequential data. It is one of the algorithms behind the scenes of the

amazing achievements seen in deep learning over the past few years. In this post, we'll cover the basic concepts of how recurrent neural networks work, what the biggest issues are and how to solve them but when do you need to use a RNN?

"Whenever there is a sequence of data and that temporal dynamics that connects the data is more important than the spatial content of each individual frame". Since RNNs are being used in the software behind Siri and Google Translate, recurrent neural networks show up a lot in everyday life. **[44]**

1.6.2 Advantages & Disadvantages of Recurrent Neural Network

1.6.2.1 Advantages

1. RNN can process inputs of any length.
2. An RNN model is modeled to remember each information throughout the time which is very helpful in any time series predictor.
3. Even if the input size is larger, the model size does not increase.
4. The weights can be shared across the time steps.
5. RNN can use their internal memory for processing the arbitrary series of inputs whichis not the case with feedforward neural networks.

1.6.2.2 Disadvantages

1. Due to its recurrent nature, the computation is slow.
2. Training of RNN models can be difficult.
3. If we are using Relu [1] or tanh [2] as activation functions, it becomes very difficult to process sequences that are very long.
4. Prone to problems such as exploding and gradient vanishing.[6]

1.6.3 How Recurrent Neural Networks Work

To understand RNNs properly, you'll need a working knowledge of "normal" feed-forward neural networks and sequential data.

Sequential data is basically just ordered data in which related things follow each other. Examples are financial data or the DNA sequence. The most popular type of sequential data is perhaps time series data, which is just a series of data points that are listed in time order.

An RNN works as follows First words get transformed into machine-readable vectors. Then the RNN processes the sequence of vectors one by one.

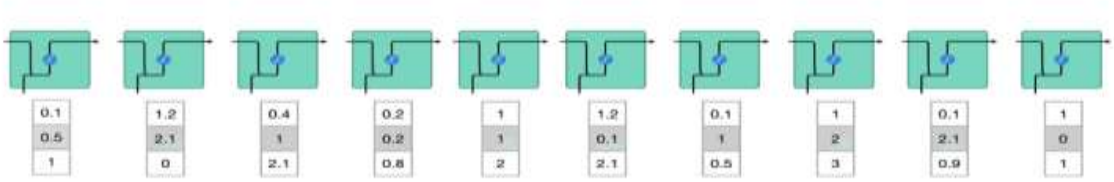


Figure1.15 Processing sequence one by one.

While processing, it passes the previous hidden state to the next step of the sequence. The hidden state acts as the neural networks memory. It holds information on previous data the network has seen before.

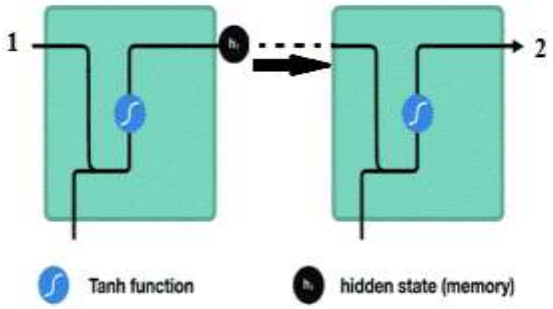


Figure1.16 Passing hidden state to next time step.

Let's look at a cell of the RNN to see how we would calculate the hidden state. First, the input and previous hidden state are combined to form a vector. That vector now has information on the current input and previous inputs. The vector goes through the tanh activation, and the output is the new hidden state, or the memory of the network.

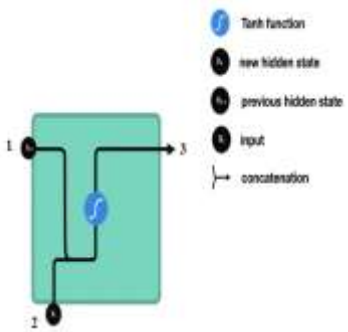


Figure 1.17 RNN Cell.

•Tanh activation:

The tanh activation is used to help regulate the values flowing through the network. The tanh function squishes values to always be between -1 and 1.

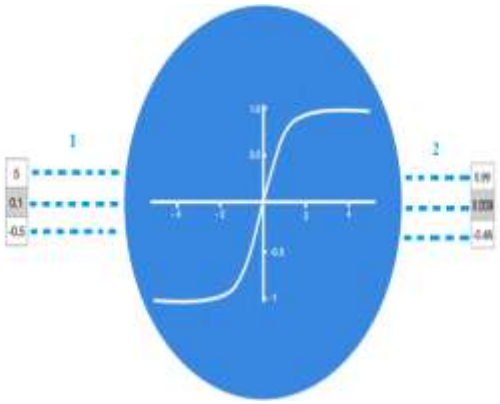


Figure 1.18 Tanh squishes values to be between -1 and 1.

When vectors are flowing through a neural network, it undergoes many transformations due to various math operations. So imagine a value that continues to be multiplied by let's say 3. You can see how some values can explode and become astronomical, causing other values to seem insignificant.

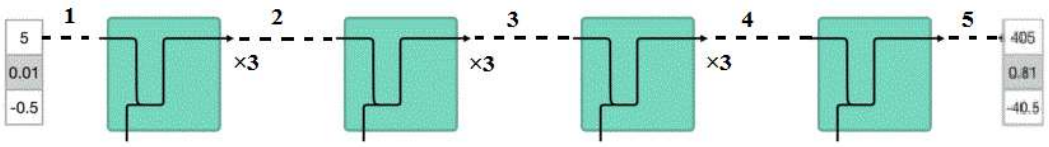


Figure1.19 Vector transformations without tanh.

So that's an RNN. It has very few operations internally but works pretty well given the right circumstances (like short sequences). RNN's uses a lot less computational resources than it's evolved variants, LSTM's and GRU's.[44]

1.6.4. Types of Recurrent Neural Networks

Feed nets map one input to one output, and although we conceptualize recurrent neural networks in this way as an initial concept, they do not actually have this limitation. Instead, their inputs and outputs can vary in length, and different types of RNNs are used for different use cases, such as music generation, sentiment classification, and

machine translation [45]. Different types of RNNs are usually expressed using the following diagrams:

- One to One
- One to Many
- Many to One
- Many to Many

Also note that while feed-forward neural networks map one input to one output, RNNs can map one to many, many to many (translation) and many to one (classifying a voice).

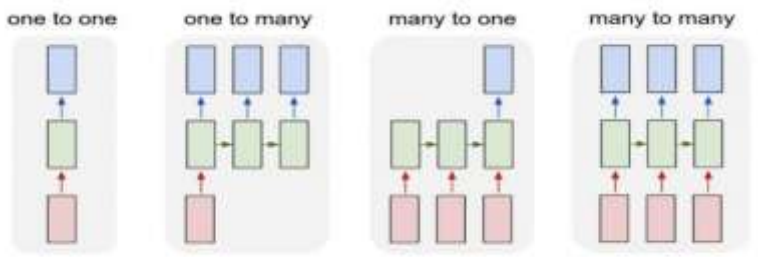


Figure 1.20 RNN Types

Feedforward networks map one input to one output, and while we've visualized recurrent neural networks in this way in the above diagrams, they do not actually have this constraint. Instead, their inputs and outputs can vary in length, and different types of RNNs are used for different use cases, such as music generation, sentiment classification, and machine translation. Different types of RNNs are usually expressed using the following diagrams:

One-to-one: One to One RNN (Tx=Ty=1) is the most basic and traditional type of Neural network giving a single output for a single input, as can be seen in the "figure 1.8". [10]

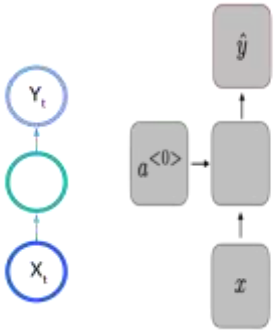
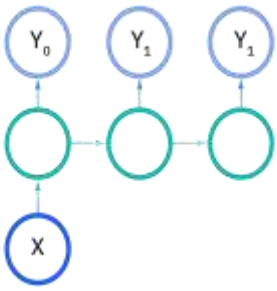


Figure 1.21 Example One to One

figure1.22Type of RNN One to One

One-to-many: One to Many (Tx=1,Ty>1) is a kind of RNN architecture is applied in situations that give multiple output for a single input , as can be seen in the "figure 9". A basic example of its application would be Music generation. In Music generation models, RNN models are used to generate a music piece (multiple output) from a single musical note(single input). [20]



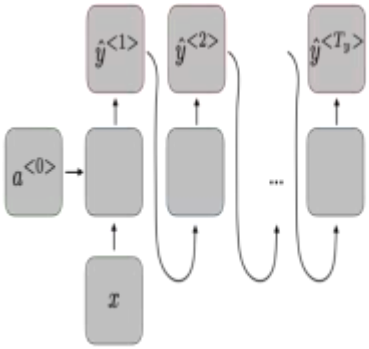


Figure 1.23 Example One to Many

Figure 1.24 Type of RNN (One-to-many).[10]

Many-to-one: Many-to-one RNN architecture (Tx>1,Ty=1) is usually seen for sentiment analysis model as a common example. As the name suggests, this kind of model is used when multiple inputs are required to give a single output, as can be seen in the "figure 10".

Take for example The Twitter sentiment analysis model. In that model, a text input (words as multiple inputs) gives its fixed sentiment (single output). [10]

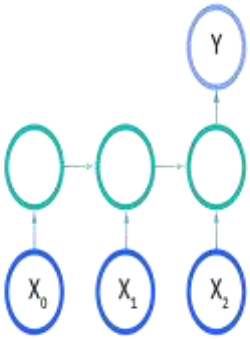


Figure1.25 Example Many to One

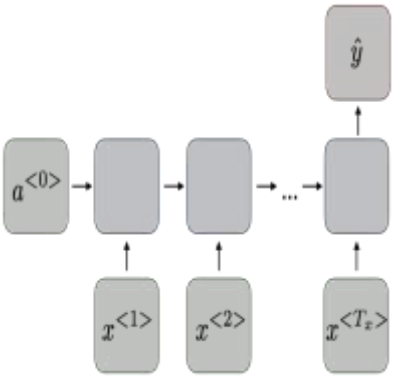


Figure 1.26 Type of RNN (Many-to-one). [45]

Many-to-many: As is pretty evident, Many-to-Many RNN (Tx>1,Ty>1) Architecture takes multiple input and gives multiple output, but Many-to-Many models can be two kinds :

1. Tx=Ty: This refers to the case when input and output layers have the same size , as can be seen in the "figure 11". This can be also understood as every input having a output, and a common application can be found in Named-entity Recognition. [45]

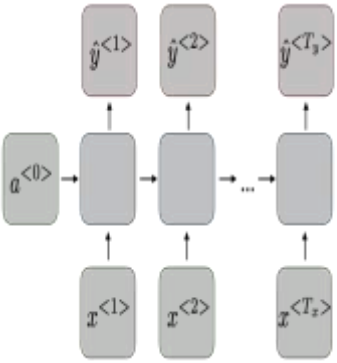


Figure1.27 Type of RNN (Many-to-many [Tx=Ty]).[45]

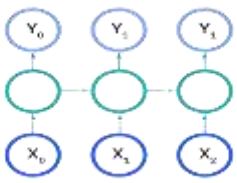


Figure1.28ExampleTx=Ty

Many-to-Many architecture can also be represented in models where input and output layers are of different size, as can be seen in the “figure 12”. and the most common application of this kind of RNN architecture is seen in Machine Translation. For example, "I Believe", the 2 words of the English language translates to only 1 in Arabic, "أعتقد". Thus, machine translation models are capable of returning words more or less than the input string because of a non-equal Many-to-Many RNN architecture works in the background. [45]

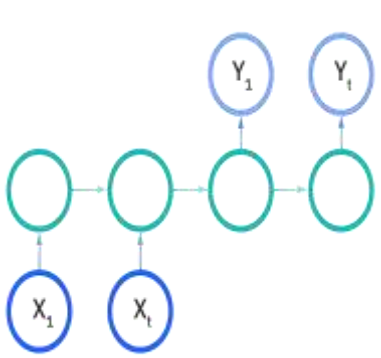


Figure1.29 Example Tx!=Ty[45]

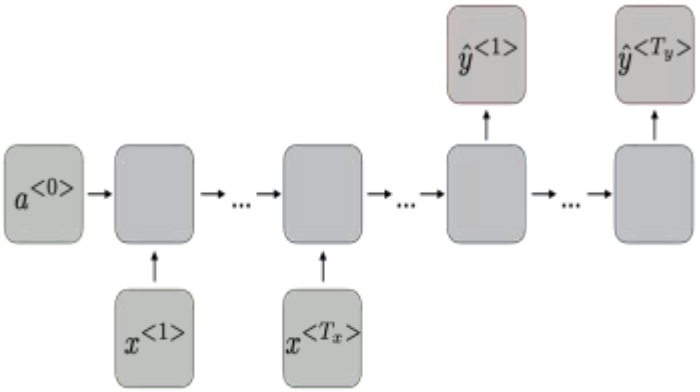


Figure 1.30 Type of RNN : Many-to-many [Tx!=Ty] .

- **Gated recurrent units (GRUs):** This RNN variant is similar the LSTMs as it also works to address the short-term memory problem of RNN models. Instead of using a “cell state” regulate information, it uses hidden states, and instead of three gates, it has two—a reset gate and an update gate. Similar to the gates within LSTMs, the reset and update gates control how much and which information to retain. [45]

1.6.5 Long Short Term Memory (LSTM)

1.6.5.1 LSTM Definition

Long Short-Term Memory (LSTM) networks are a modified version of recurrent neural networks, which makes it easier to remember past data in memory. The vanishing gradient problem of RNN is resolved here. LSTM is well-suited to classify, process and predict time series given time lags of unknown duration. It trains the model by using back-propagation. In an LSTM network, three gates are present: Input gate, Forget gate, Output gate.[47]

1.6.5.2 Workings of LSTM

LSTM was created as a solution for short-term memory. It has internal mechanisms called gates that can regulate the flow of information.

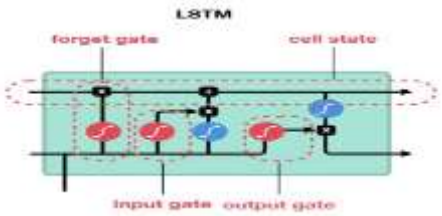


Figure: LSTM



Figure 1.31 LSTM Gates.

These gates can learn which data in a sequence is important to keep or throw away. By doing that, it can pass relevant information down the long chain of sequences to make predictions. Almost all state of the art results based on recurrent neural networks are achieved with these network. LSTM's can be found in speech recognition, speech synthesis, and text generation. You can even use them to generate captions for videos.

Let's start with a thought experiment. Let's say you're looking at reviews online to determine if you want to buy Morning cereal. You'll first read the review then determine if someone thought it was good or if it was bad.

And that is essentially what an LSTM does. It can learn to keep only relevant information to make predictions, and forget non relevant data. In this case, the words you remembered made you judge that it was good.

An LSTM has a similar control flow as a recurrent neural network (As already mentioned in the Recurrent Neural Network section , exactly in "4.5 How a Recurrent Neural Network Works?"). It processes data passing on information as it propagates forward. The differences are the operations within the LSTM's cells.

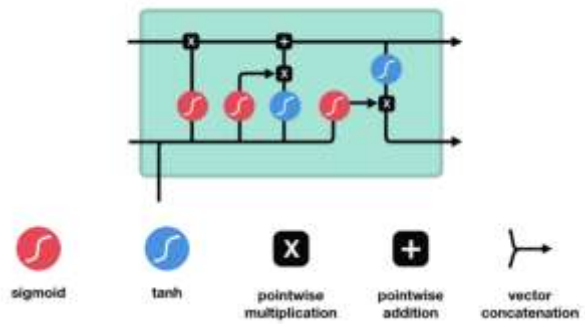


Figure 1.32 LSTM Cell and It's Operations.

These operations are used to allow the LSTM to keep or forget information.

•**Core Concept**: The core concept of LSTM's is the cell state, and it's various gates. The cell state act as a transport highway that transfers relative information all the way down the sequence chain. We can think of it as the "memory" of the network. The cell state, in theory, can carry relevant information throughout the processing of the sequence. So even information from the earlier time steps can make its way to later time steps, reducing the effects of short-term memory. As the cell state goes on its journey, information gets added or removed to the cell state via gates. The gates are different neural networks that decide which information is allowed on the cell state. The gates can learn what information is relevant to keep or forget during training.

Sigmoid: Gates contains sigmoid activations. A sigmoid activation is similar to the tanh activation. Instead of squishing values between -1 and 1, it squishes values between 0 and 1. That is helpful to update or forget data because any number getting multiplied by 0 is 0, causing values to disappears or be "forgotten." Any number multiplied by 1 is the same value

therefore that value stay's the same or is "kept." The network can learn which data is not important therefore can be forgotten or which data is important to keep.

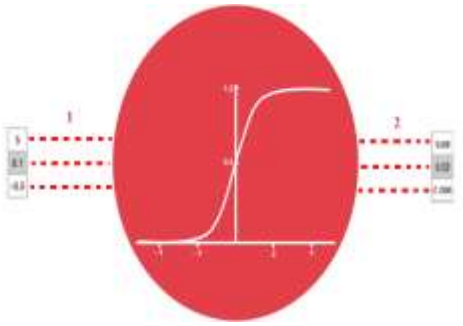


Figure 1.33 Sigmoid squishes values to be between 0 and 1.

Let's dig a little deeper into what the various gates are doing, so we have three different gates that regulate information flow in an LSTM cell. A forget gate, input gate, and output gate.

1) **Forgetgate**: First, we have the forget gate. This gate decides what information should be thrown away or kept. Information from the previous hidden state and information from the current input is passed through the sigmoid function. Values come out between 0 and 1. The closer to 0 means to forget, and the closer to 1 means to keep.

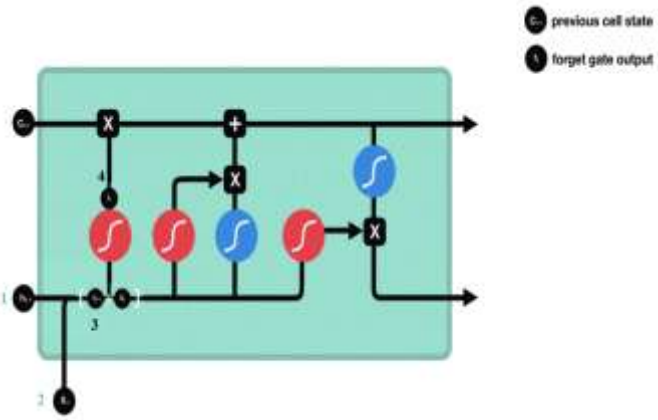


Figure 1.34 Forget gate operations.

2**) InputGate**: To update the cell state, we have the input gate. First, we pass the previous hidden state and current input into a sigmoid function. That decides which values will be updated by transforming the values to be between 0 and 1. 0 means not important, and 1 means important. You also pass the hidden state and current input into the tanh function to squish values between -1 and 1 to help regulate the network. Then you multiply the tanh

output with the sigmoid output. The sigmoid output will decide which information is important to keep from the tanh output.

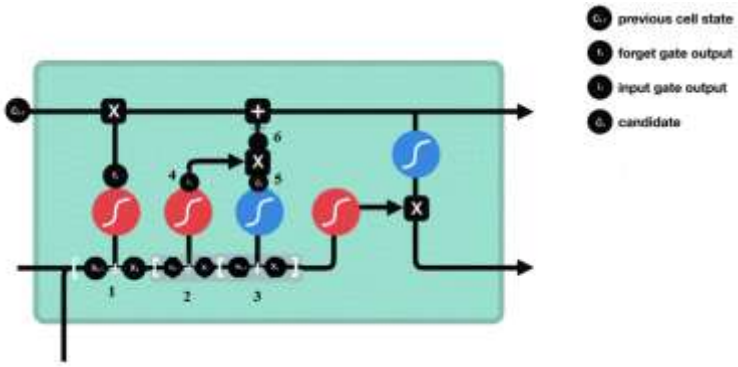


Figure 1.35 Input gate operations.

3) CellState: Now we should have enough information to calculate the cell state. First, the cell state gets pointwise multiplied by the forget vector. This has a possibility of dropping values in the cell state if it gets multiplied by values near 0. Then we take the output from the input gate and do a pointwise addition which updates the cell state to new values that the neural network finds relevant. That gives us our new cell state.

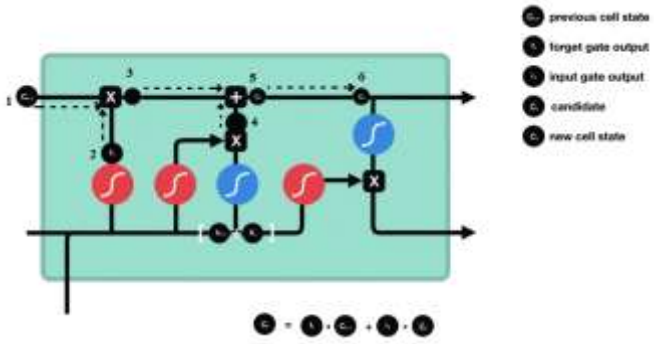


Figure 1.36 Calculating cell state.

4) **OutputGate:**The output gate decides what the next hidden state should be. don't forget that the hidden state contains information on previous inputs. The hidden state is also used for predictions. First, we pass the previous hidden state and the current input into a sigmoid function. Then we pass the newly modified cell state to the tanh function. We multiply the tanh output with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden is then carried over to the next time step.[24]

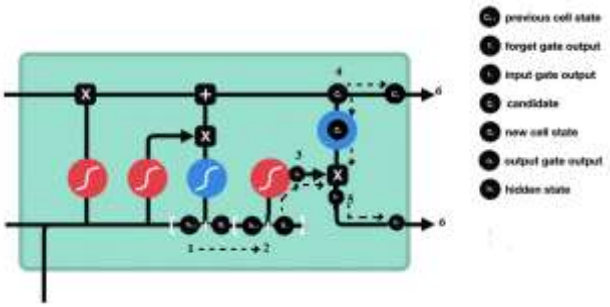


Figure 1.37 Output gate operations.

1.6.5.3 Architecture of LSTM

•**Vanilla LSTM**: Memory cells of a single LSTM layer are used in a simple network structure.

•**Stacked LSTM**: LSTM layers are stacked one on top of another into deep networks.

•**CNN LSTM**: A convolutional neural network is used to learn features in spatial input like images and the LSTM can be used to support a sequence of images as input or generate a sequence in response to an image.
•**Encoder-Decoder LSTM**: One LSTM network encodes input sequences and a separate LSTM network decodes the encoding into an output sequence.

•**Bidirectional LSTM**: Input sequences are presented and learned both forward and backward.

•**Generative LSTM**: LSTMs learn the structure relationship in input sequences so well that they can generate new plausible sequences.[48]

1.6.6 Word Embeddings

1.6.6.1 The Word Embedding Model

Word embeddings are a type of word representation that allows words with similar meaning to have a similar representation. They are a distributed representation for text that is perhaps one of the key breakthroughs for the impressive performance of deep learning methods on challenging natural language processing problems.[48]

1.6.6.2What Are Word Embeddings?

A Word Embedding is a learned representation for text where words that have the same meaning have a similar representation. It is this approach to representing words and documents that may be considered one of the key breakthroughs of deep learning on challenging natural language processing problems.

Word embeddings are in fact a class of techniques where individual words are represented as real-valued vectors in a predefined vector space. Each word is mapped to one vector and the vector values are learned in a way that resembles a neural network, and hence the technique is often lumped into the field of deep learning.

Key to the approach is the idea of using a dense distributed representation for each word.

Each word is represented by a real-valued vector, often tens or hundreds of dimensions. This is contrasted to the thousands or millions of dimensions required for sparse word representations, such as a one hot encoding.

The distributed representation is learned based on the usage of words. This allows words that are used in similar ways to result in having similar representations, naturally capturing their meaning. This can be contrasted with the crisp but fragile representation in a bag-of words model where, unless explicitly managed, different words have different representations, regardless of how they are used.

1.6.6.3 Word Embedding Algorithms

Word embedding methods learn a real-valued vector representation for a predefined fixed sized vocabulary from a corpus of text. The learning process is either joint with the neural network model on some task, such as document classification, or is an unsupervised process, using document statistics. This section reviews three techniques that can be used to learn a word embedding from text data.

1.6.6.3.1 Embedding Layer

An embedding layer, for lack of a better name, is a word embedding that is learned jointly with a neural network model on a specific natural language processing task, such as language modeling or document classification. It requires that document text be cleaned and prepared

such that each word is one hot encoded. The size of the vector space is specified as part of the model, such as 50, 100, or 300 dimensions. The vectors are initialized with small random numbers. The embedding layer is used on the front end of a neural network and is fit in a supervised way using the Backpropagation algorithm.

This approach of learning an embedding layer requires a lot of training data and can be slow, but will learn an embedding both targeted to the specific text data and the NLP task.

1.6.6.3.2. Word2Vec

Word2Vec is a statistical method for efficiently learning a standalone word embedding from a text corpus. It was developed by Tomas Mikolov al. at Google in 2013 as a response o make the neural-network-based training of the embedding more efficient and since then has become the de facto standard for developing pre-trained word embedding .

Two different learning models were introduced that can be used as part of the Word2Vec approach to learn the word embedding they are:

- Continuous Bag-of-Words, or CBOW model.
- Continuous Skip-Gram Model.

The CBOW model learns the embedding by predicting the current word based on its context .The continuous skip-gram model learns by predicting the surrounding words given a current word.

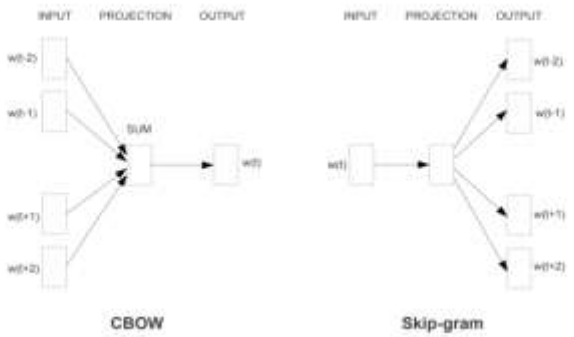


Figure 1.38 Word2Vec Training Models. [49]

Both models are focused on learning about words given their local usage context, where the context is defined by a window of neighboring words. This window is a configurable parameter of the model.

The key benefit of the approach is that high-quality word embeddings can be learned efficiently (low space and time complexity), allowing larger embeddings to be learned (more dimensions) from much larger corpora of text (billions of words).

1.6.6.3.3 GloVe

[2]The Global Vectors for Word Representation, or GloVe, algorithm is an extension to the Word2Vec method for efficiently learning word vectors, developed by Pennington, et al. at Stanford. Classical vector space model representations of words were developed using matrix factorization techniques such as Latent Semantic Analysis (LSA) that do a good job of using global text statistics but are not as good as the learned methods like Word2Vec at capturing meaning and demonstrating it on tasks like calculating analogies

GloVe is an approach to marry both the global statistics of matrix factorization techniques like LSA with the local context-based learning in Word2Vec. Rather than using a window to define local context, GloVe constructs an explicit word-context or word co-occurrence matrix using statistics across the whole text corpus. The result is a learning model that may result in generally better word embeddings. [49]

1.6.6.4 Using Word Embeddings

There are several options when it comes time to use word merging in a NLP project. This section explains those options.

1.6.6.4.1 Learn to embedding

There are two main options when training word embedding:

▪**Independent learning:** where the model is trained to learn embedding, which is saved and used as part of another project task model. This is a good approach if you want to use the same embed on multiple forms.

▪**Co-learning**: where embedding is learned as part of a model for a large task. This is a good approach if you only use embeds for one task.

1.6.6.4.2 Reuse an Embedding

It is common for researchers to make pre-trained word embeddings available for free, often under a permissive license so that can use them on academic or commercial projects. For example, both Word2Vec and GloVe word embeddings are available for free download. These can be used instead of training embeddings from scratch. There are two main options when it comes to using pre-trained embeddings:

▪**Static**: where the embedding is kept static and is used as a component of model.This is a suitable approach if the embedding is a good fit for problem and gives good results .

▪**Updated**: where the pre-trained embedding is used to seed the model, but the embedding is updated jointly during the training of the model. This may be a good option if we are looking to get the most out of the model and embedding task.

Conclusion

In this chapter, a general idea of the Arabic language and there characteristic and the Post Tagging relation between them. After that we talk about Natural Language Processing is presented so that a simplified definition, its brief history, steps, major tasks, approaches and applications are presented. we talk generally about the Arabic language and there characteristic, then we take a general information about part of speech

After that, the Neural Network was discussed, and some information was provided about it, its definition, and a brief history, its construction, components, algorithms. And then more specialization was made in this section, so one of the most widely used types of neural network, which is called the Recurrent Neural Network, was presented. As well as its different architectures.

Chapter 02:

Proposed Approach

- **Introduction**

In this chapter, we will explain the steps of the conception of the proposed model and how it works. The problem of Arabic pos-tagging will be treated through a deep learning model, based on RNN network. Two models were designed and tested: B-LSTM and GRU model. At the end, we compare the performance of the two models.

- **Proposed Approach Overview**

We present below the general architecture giving the possible steps:

- Preprocessing of the Dataset
- Split Train and Test
- Creation of the Model
- Evaluation of the model

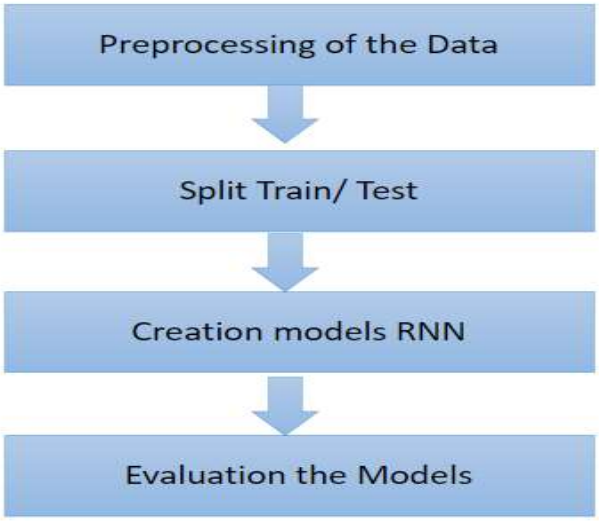


Figure2.1Architecture of Overview

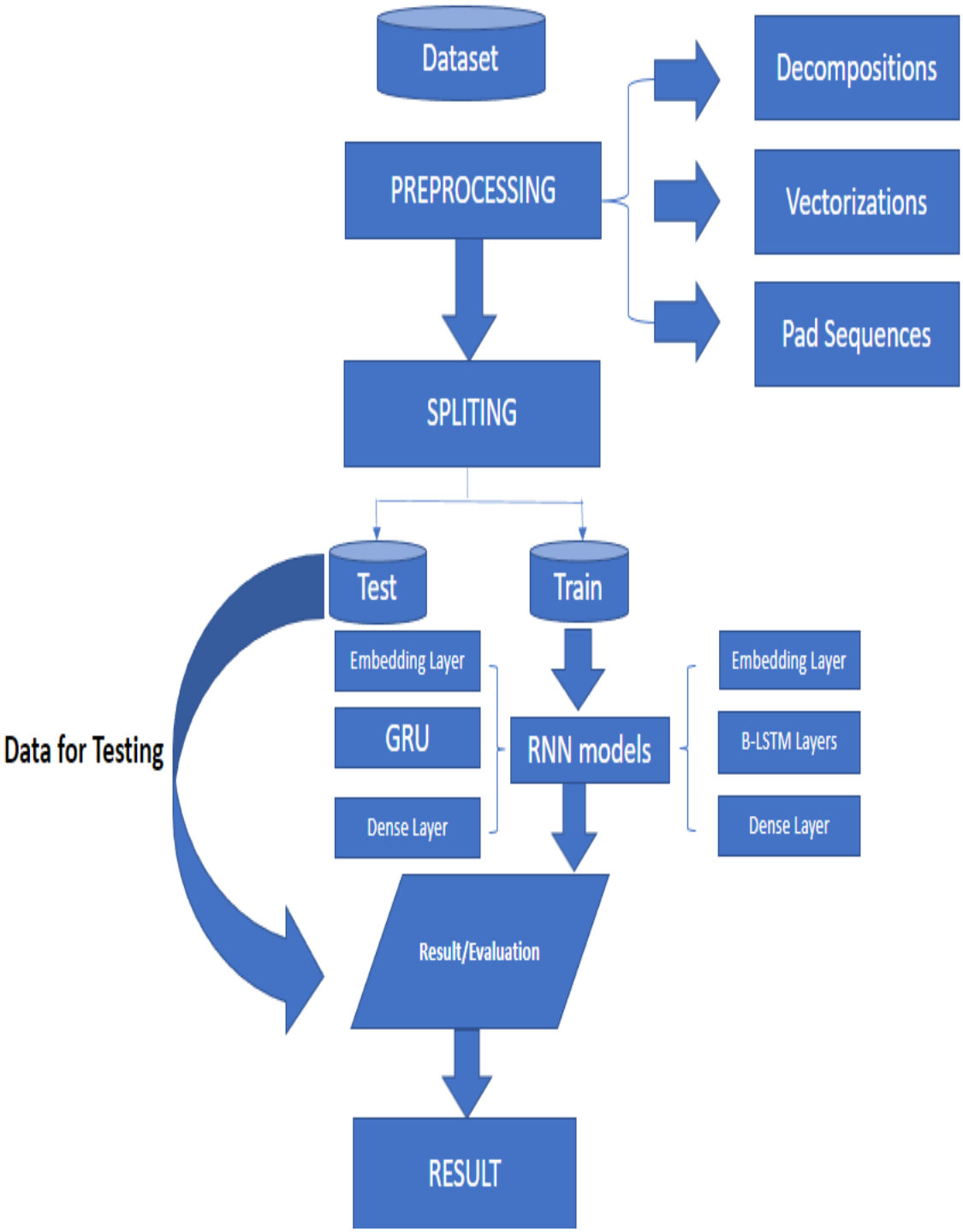


Figure.2.2 Global Architecture of our Approach

2.DataSet

2.1. KALIMAT DataSet

KALIMAT**:** is a multipurpose Arabic corpusthat is composed of:

1) 20,291 Arabic articles collected from the Omani newspaper Alwatan by (Abbas et al. 2011).

2) 20,291 Extractive Single-document system summaries.

3) 2,057 Extractive Multi-document system summaries.

4) 20,291 Named Entity recognized articles.

5) 20,291 Part of Speech Tagged articles.

6) 20,291 Morphologically Analyze articles.[51]

We explain all this informations in the table below:

Collections	**Number of Articles**
The Omani newspaper	20,291
Extractive Single-document system summaries.	20,291
Extractive Multi-document system summaries.	2,057
Named Entity Recognized articles.	20,291
Part of Speech Tagged articles.	20,291
Morphologically Analyze articles.	20,291

Table2.1.Collections with their numbers of articles

2.2. Categories of the Dataset

The data collection articles fall into six categories: culture, economy, local-news, international-news, religion, and sports. Therefore, we are going to present it in this table with their numbers of articles and words.[51]

Categories of the collection articles	Numbers of Articles	Number of words
Culture	2782	1,359,210
Economy	3468	3,122,565
local-news	2035	855,945
international-news	3596	1,460,462
Religion	3860	1,555,635
Sports	4550	9,813,366

Table.2.2. KALIMAT Corpus Statistic. [52]

2.3. Creation of Dataset

The process of creating KALIMAT was applied to the entire data collection (20,291 articles). Firstly, its summarized the document collection using two Arabic summarizers, **Gen–Summ** and **Arabic Cluster-based**.

Gen-Summ is a single document summarizer based on the VSM model that takes an Arabic document and its first sentence and returns an extractive summary. A number of 20,291 system summaries have been generated.

Cluster-based is a multi-document summarizer that treats all documents to be summarized as a single bag of sentences. The sentences of all the documents are clustered using different number of clusters. A summary is created by selecting sentences from the biggest cluster only (if there are two, we select the first biggest cluster). We generated 2,057 multi-document extractive system summaries with a summary for each 10, 100 and 500 articles in each category, in addition to a summary for all the articles in each category.

Secondly, we used an Arabic Named Entity Recognition system (ANER). To annotate the data collection, we followed the Computational Natural Language Learning (CoNLL) 2002 and 2003 shared tasks formed by tags falling into any of the following four categories:

- Person Names: محمود درويش (Mahmoud Darwish).
- Location names: المغرب(Morocco).
- Organization Names: الأمم المتحدة (United Nations).
- Miscellaneous Names: NEs not belonging to any of the previous classes and include date, time, number, monetary expressions, measurement expressions and percentages. ANER system was trained using ANERCorpus (Benajiba et al. 2007), a manually annotated corpus

following the CoNLL shared task. The reason behind choosing ANERCorpus to train our system was that the corpus articles were chosen from Arabic newswires and Wikipedia Arabic, which is quite close to Alwatan's data collection.

Thirdly, we used **Stanford POSTTagger** to annotate the 20,291document collection. The model for Arabic was trained using the Arabic Tree-bank p1-3 corpus based on maximum entropy and using augmented By mapping of ATB tags. The PostTagger identifies 33 part of speeches, using the Treebank project codification such as: Noun (NN), Plural Noun (NNS), Proper Noun (NNP), Verb (VB), Adjective (JJ).

Finally, we applied a morphological analysis process on the data collection using Alkhalil morphological analyzer (Mazroui et al. 2011).

The Analysis was carried out in the following steps: pre-processing (removal of diacritics) and segmentation (each word is considered as [proclitic + stem + enclitic]). Applying Alkhalil analyzer on the data collection we reached an accuracy of 96%. We implemented a Viterbi algorithm to get one solution that is relevant to the context of the analyzed article. We provide KALIMAT for free including the articles, annotated text, entities and summaries to help advancing the work on Arabic NLP.[51]

2.4 About the Dataset

The corpus that we work on contains many articles classified by six collections. We work on one collection which is Part of Speech Tagged article (2782 articles) that contains pair words (token/tag). This picture shows us one of the articles in the Dataset:

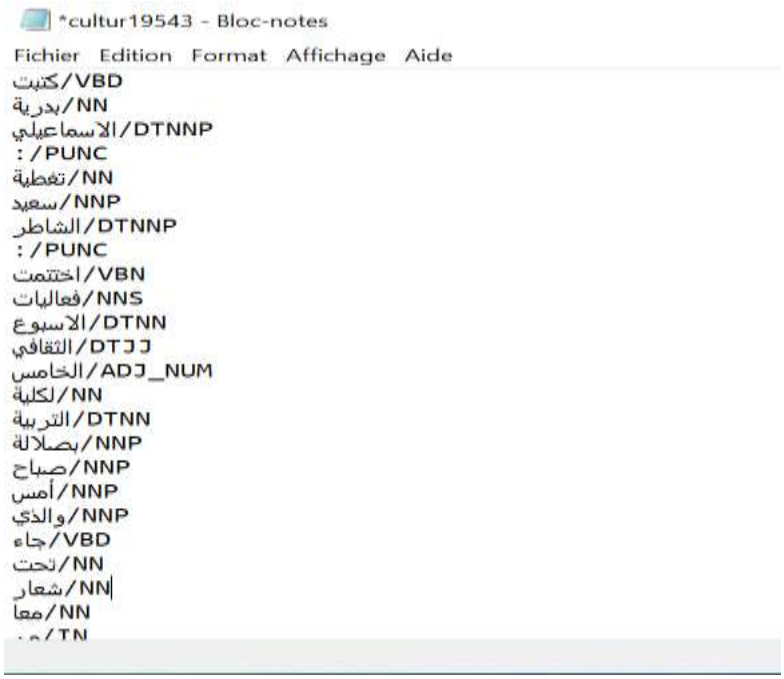


Figure.2.3 Example of Article in the Dataset.

2.5 Preprocessing of the data

In this part of preprocessing, we will perform different steps:

1-dividing data into features and labels

2- vectorization of X and Y

3- pad sequences.

Each data point will be a distinct sentence from the corpus. As this is a many-to-many problems, the input sequence for each data point will contain numerous words. This will be referred as features. In the output sequence, each word will have its own tag. This will be referred as labels. The general work in this part to decompose the data in features and labels as in the following example.

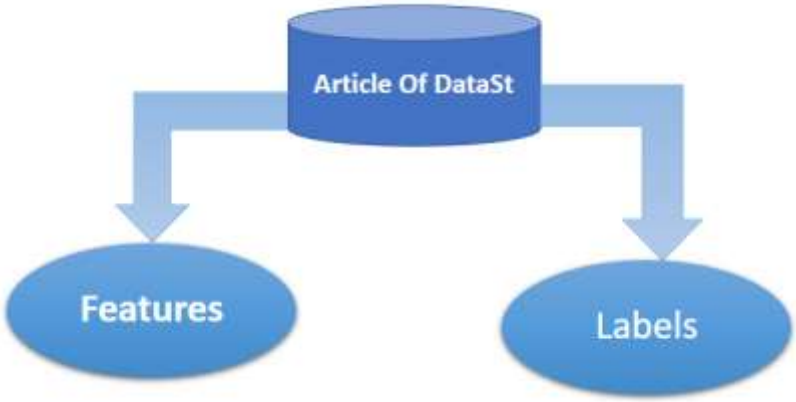


Figure2.4 Example Of Data Preprocessing

2.5.1 Decomposition of the corpus

In this part, When our program reads a foreign words for the first side and finds a punctuation (PUNC) in the other side shows us that the sentence is ended, it adds in the X all the words that are before the backslash « / » and all the words that are after the backslash in Y(the tag of the word in the left side). For example: كتبت/VBD بدرية /NN الاسماعيلي/DTNNP :/PUNC

X ={كتبت بدرية الاسماعيلي}

Y= {DTNNP NN VBD}

The first sentence found will be put in the matrix Inputset(X) and the second tags sentence will be put in the matrix Outputset (Y).

The program will start a new line when there is a punctuation (it means that we have a new sentence) we collect all the words in the dataset to get sentences then it will be representing on this way Inputset/Outputset]. This will be explained in the next figure.

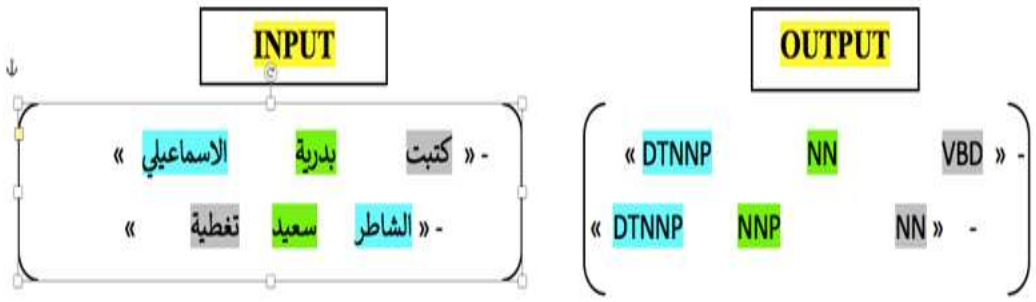


Figure2.5 Example of the Decomposition of two Sentences

The program will limit all sentences by their numbers of words that must be from two to one hundred words (2 to 100). The sentences that contain more than100 words or just one word will be deleted in the matrix (Inputset and Outputset). We represent one sentence from two matrix as following:

```
Inputset=[' ,'تظاهرة ,'فنية ,'استطاعت ,'أن ,'تثبت ,'أن ,'للمسرح
'دورا ,'مهما ,'في ,'إثراء ,'الحركة ,'الثقافية ,'في ,'الوسط ,
'التربوي ,'لمدة ,'ثلاثة ,'أيام ,'متتالية ,'ضمن ,'المهرجان ,
''المسرحي ,'بكلية ,'التربية ,'بنزوى]
```

```
Outputset= ['NN', 'JJ', 'VBD', 'NN', 'VBP', 'NN', 'NN', 'NN', 'JJ',
'IN', 'NN', 'DTNN', 'DTJJ', 'IN', 'DTNN', 'DTJJ', 'NN', 'CD', 'NN',
'JJ', 'NN', 'DTNN', 'DTJJ', 'NN', 'DTNN', 'NNP']
```

Finally, we calculate and display the total number of tagged sentences, vocabulary size and the total number of tags. Also, we displayed all the tags possible to know the different tags of dataset like this:

- Total number of tagged sentences: 18943
- Vocabulary size: 63691
- Total number of tags: 29
- tags are: {'JJ', 'WP', 'VB', 'PRP$', 'DTJJ', 'RB', 'RP', 'VBG', 'JJR', 'DTNNS', 'DT', 'VN', 'PRP', 'NOUN_QUANT', 'WRB', 'NNP', 'UH', 'ADJ_NUM', 'DTJJR', 'VBD', 'CC', 'DTNNP', 'NN', 'CD', 'DTNN', 'VBN', 'NNS', 'VBP', 'IN'}

Here we have a table explaining the Post tags in our work

tags	Description	Example in Arabic
CC	Coordination Conjunction	حروف العطف
CD	Cardinal number	الأول الثاني الثالث
DT	Determiner	المعرفة الاف والام
EX	Existential there	هناك , ذالك، هذا
FW	Foreign word	No word arabic
IN	Proposition	حروف الجر: في ،على،من إذا
JJR	Adjective	الصفة
JJS	Adjective, superlative	اكثر من اجمل من
	List item marker	ادوات النفي لا ليس لم
MD	Model	سوف استطيع
NN	Noun singular	اسماء المفردة
NNS	Noun plural	اسماء الجمع
NNP	Proper noun, singular	اسم علم مفرد
NNPS	Proper noun, plural	
ADJ_NUM	Adjective of number	الاعداد الاصلية:0,1,2.....9
PRP	Personal pronoun	اسماء اشخاص
PRPS	Possessive pronoun	ضمائر الملكية ي ه ك
RB	adverb	الحال
RBR	Adverb, comparative	اقل من
RPS	Adverb, superlative	الافضل
PUNC	Punctuation	علامات الوقف
VB	Verb, base form	فعل اصلي
VBD	Verb, past tense	فعل ماضي
VBG	Verb, present participle	الافعال المضارعة
VBN	Verb, past participle	افعال ماضية مبنية للمجهول
VBP	Verb, non-singular	تصريف الافعال في ضمائر الجمع
WRB	Verb, two-person	تصريف الافعال في ضمائر المثنى
WP	WH-pronouns	التي الذي اللذان
DTNNS	Determinant N, Plural	اسماء مضافة في الجمع
DTJJ	Determinant adjective	اسم موصوف
DTNN	Determinant N, Singular	اسماء مضافة مفردة
NOUN-QUANT	Noun of count	كل بعض نصف ضعف ربع
UH	Interjection	يا ايها ايتها

table2.3 Explication of tags

2.5.2 Vectorize X and Y

The next important step is to convert the text data to numbers, as the machine understands numbers only. There are many ways to convert text to vectors- Count vectorizer, TFIDF vectorizer, and Word Embeddings. In this, we will be discussing Count Vectorizer: This is the basic technique to represent the text data numerically. In this, the unique words (called features) are extracted from the text and then the frequency of each word in the text is calculated. The features extracted from the text are called *Bag of Words.* Each row is a text and every word across all documents is a column. Each cell denotes the count of words in the text. Now, for example, we have 2 sentences:

Text A: "I like to play football"

Text B: "Weather is great today"

After removing stop words, we will be left with- "like play football" and "weather great today". Now apply count vectorizer.

	After removing stopwords					
	football	great	like	play	today	weather
Text A	1	0	1	1	0	0
Text B	0	1	0	0	1	1

Table2.4 Each cell denotes the number of times the word is present in the text

In the count vectorizer, we are just getting the frequency count of the words. Where we are finding the word similar to the extracted words from the count vectorizer, we are going to place 1 there and increase the count whenever that particular word comes again in the text, and elsewhere it will be 0.

The drawback of the count vectorizer is that we cannot infer how important a word is in the text. [53]

The next part of decomposition of the data is encoding all the sentences of two matrix (Inputset and Outputset) into numbers which will have a blind a unique ID for each words and tags as mentioned in the following examples. We can say transfer word to numbers for feeding these inputs (X) and output(Y) to an RNN that oblige to do the vectorization.

This is an example of vectorization of the words:

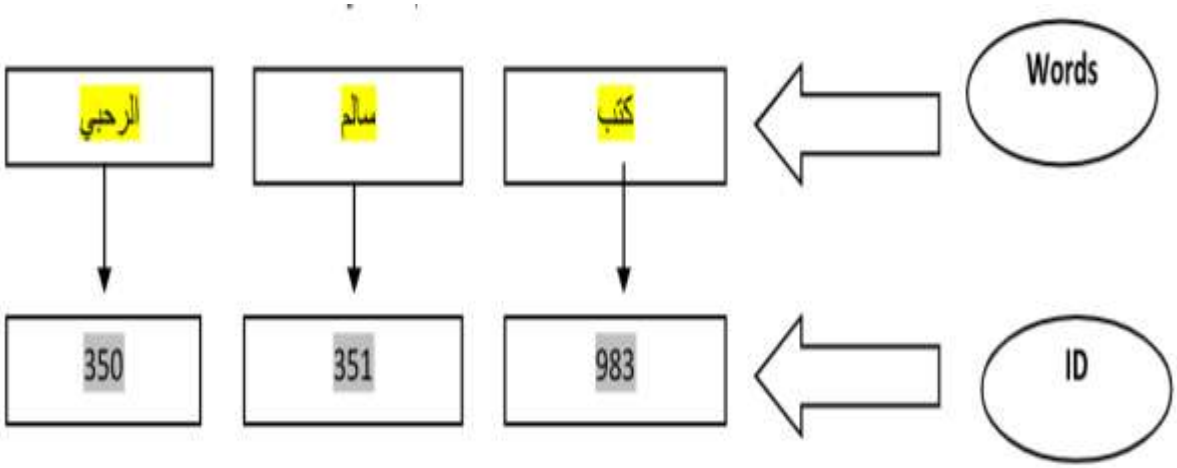


Figure 2.6 Example of Encoding X

This is an example of vectorization of the Post Tags:

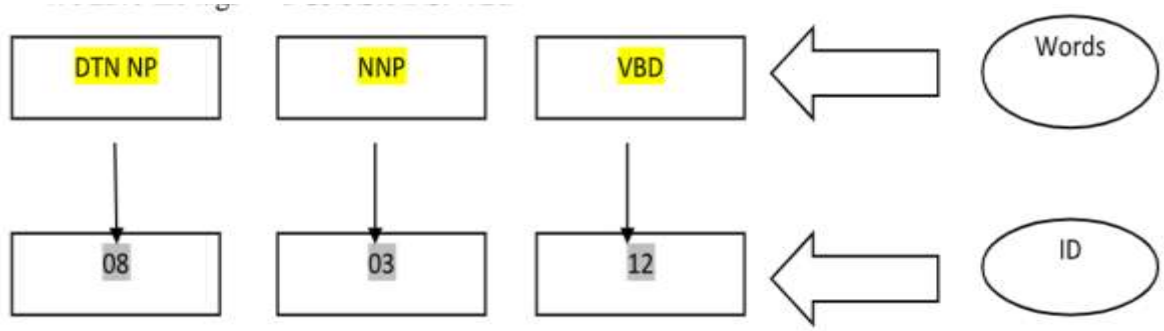


Figure2.7 Example of Encoding Y

2.5.3 Pad Sequences

In this part, after the length of the sentences in the corpus varies. The length of the sentences must be fixed before the input is fed into the RNN model. Because the sentences in the corpus are all various lengths, we cannot dynamically allocate the memory required to process them. As a result, after encoding the data, the following step is to define the sequence lengths. Short sentences must be padded, while large ones must be truncated to a specified length. However, this set length.

we have seen how to create sequences of sentences in the actual data. However, these sequences can be of different length, which is a problem for deep learning models. We need to have a consistent input size for the model. In addition, we can achieve that using padding. Now, TensorFlow, our you can say the Keras API actually offers padding functions that does all the heavy lifting for us. So again, here from Keras preprocessing sequence module we are going to import pad_sequences function.

So first of all, this is a python list, train sentences, I've defined four sentences of different lengths. So 'it will rain', the first sentence contains three words. 'The Weather is cloudy' contains four words, and we have 'Will it be raining today' now the next step is to train this tokenizer using the fit on text method, passing the training sentences, and the last step is to store the word index. The word encoding dictionary that would be generated using the word and the score index attribute, we are using 'pad_sequences'. The function that we have imported above, to this function we are going to pass the sequences that we have generated right above. All right. So, once we run this, we would have the padded sequences stored in this variable pattern. And the (indistinct) sequences, let's print the word encoding dictionary. 'word_index' The next step is to store trainings sentences. Let's print all the training sentences just for our convenience. Then we can print the sequences and we'll compare it with the padded sequences that we have generated using the pad sequences function. Then I am passing this padding parameter which would basically contain the string, which will define do you want a pre-padding or a post-padding, if we look at the padded sequence, you see all of the sequences are all of the length five. And you see the last sentence which had 2, 4 12, 13, 14, 15, as you can see over here [54]:

This is an example of padding the words:

« عام»	«مكان»	« في »	« الناس »	«جلس »
0	0	«التفاحة »	«عمر »	« أكل »

Figure2.8 Example of Padding Tokens

This is an example of padding Postages:

« ADJ»	«NN»	«IN»	«NNP»	«VBD»
0	0	« NN »	« NNP »	« VBD »

Figure2.9 Example of Padding Tags

2.6 Creation Of RNN

There are two key cutting-edge RNN variations that have enabled enormous networks to be trained on real datasets. Although RNNs are capable of addressing a wide range of sequence issues, the challenges of exploding and vanishing gradients that occur during RNN training

make their architecture their deadliest nemesis. The Long, Short Term Memory (LSTM) and the Gated Recurrent Unit (GRU) are two common gated RNN architectures that overcome this problem (GRU). In terms of POS tagging, we'll take a look at all of these models.

Specifically, the problem we are trying to solve is known as sequence modelling where we try to model sequential data like text and learn how to model it. We proposed two different models: Bidirectional LSTM which is more developed then LSTM and GRU to discover which one is the best.

We need to generate a vectorization for each word which is a vector of numbers. To begin, we will encrypt the input and output data which will give each word in the corpus a blind unique ID for input data. Then, we must ensure that the dimensions of the data are what RNN expects before using it. In general, RNN expects to see something like this:

- Shape of X: (#samples, #timesteps, #features)
- Shape of Y: (#samples, #timesteps, #labels)

Depending on the sort of architecture, the shape you employ to feed an RNN can take a variety of forms. The input and output both include the number of timesteps, which is nothing more than the sequence length because the problem we are working on has a many-to-many design. However, the tensor X lacks the third dimension, which is the number of features. Because we are going to employ word embeddings before feeding the data into RNN, we do not need to indicate the third dimension explicitly. That is because when you use Keras' Embedding () layer, the training data is automatically changed to (#samples, #timesteps, #features) where #features are the embedding dimension (and note that the embedding layer is always the first layer of RNN).While using the Embedding layer we only need to reshape the data to (#sample, #timesteps) which is what we have done. But, note that you will need to shape it to (#samples, #timesteps, #features) in case you do not use the embedding layer in keras.

We have the Y matrix (tags/output data) on the other hand. We have twenty-nine POS tags in this execution, each of which is treated as a class and transformed into a one-hot encoding of length twenty-nine. We will Make sure that each sequence of input and output is of the same length.

2.6.1 One hot encoding

The one hot encoding is converting each word to a binary vector with the same length

and marking the specific word with a 1.

```
["Red" "Blue" "White" "Yellow" "Black" "Pink"] White [0 0 1 0 0 0]

Pink [0 0 0 0 0 1]
```
[55]

2.6.2. Creation of Bidirectional LSTM Layers

LSTM: Many attempts have been made to fix the vanishing gradients problem by tweaking vanilla RNNs so that gradients don't expire when sequences get long. The long, short-term memory network, or LSTM, has been the most popular and successful of these initiatives. LSTMs have become so popular that they have nearly superseded vanilla RNNs.

LSTM features an explicit memory unit that retains information essential to learning a task, which is one of the key differences between an RNN and an LSTM. The sole way a normal RNN recalls past information is via updating hidden states over time, but the network lacks an explicit memory to store information. In addition to that, in LSTM, the memory units retain pieces of information even when the sequences are very long.
When you want to assign a sentiment score to a piece of text (say, a customer review), for example, the network can look at the complete review text before awarding a score. In a job like guessing the next word based on the past few typed words, on the other hand, the network does not have access to the words in future time steps while predicting the next word.

Offline and online sequence processing are the terms used to describe these two sorts of operations.

Now, because the network has access to the full sequence before making predictions, why not utilize this task to make the network 'look at the future items in the sequence' during training, in the hopes of making the network learn better

Bidirectional RNNs use this concept to their advantage.

1/ Embedding Layer

The Embedding layer (the first hidden layer) is initialized with random weights and will learn an embedding for all of the words in the training dataset. It is a flexible layer that can be used in a variety of ways, such as:

- - It can be used alone to learn a word embedding that can be saved and used in another model later.
- - It can be used as part of a deep learning model where the embedding is learned along with the model itself.
- - It can be used to load a pre-trained word embedding model, a type of transfer learning.

The Embedding layer is defined as the first hidden layer of a network. It must specify 3 arguments:

o input dim: This is the size of the vocabulary in the text data. o output dim: This is the size of the vector space in which words will be embedded. o input length: This is the length of input sequences.[55]

2/Dense layer

Dense layeris a neural network layer that is connected deeply, which means each neuron in the dense layer receives input from all neurons of its previous layer. The dense layer is found to be the most commonly used layer in the models.

In the background, the dense layer performs a matrix-vector multiplication. The values used in the matrix are actually parameters that can be trained and updated with the help of backpropagation.

The output generated by the dense layer is an ‘m’ dimensional vector. Thus, dense layer is basically used for changing the dimensions of the vector. Dense layers also apply operations like rotation, scaling, translation on the vector. [56]

3/ Output layer

The output layer is the last dense layer, to which the above applies.

In our creation of Bidirectional LSTM. We must to:

- we build two layers of bidirectional LSTM, add LSTM and Add the dense layer for our output in time distributed (output at each sequence) layer then we use SoftMax function for activation for our output
- And we plan their architecture as that:

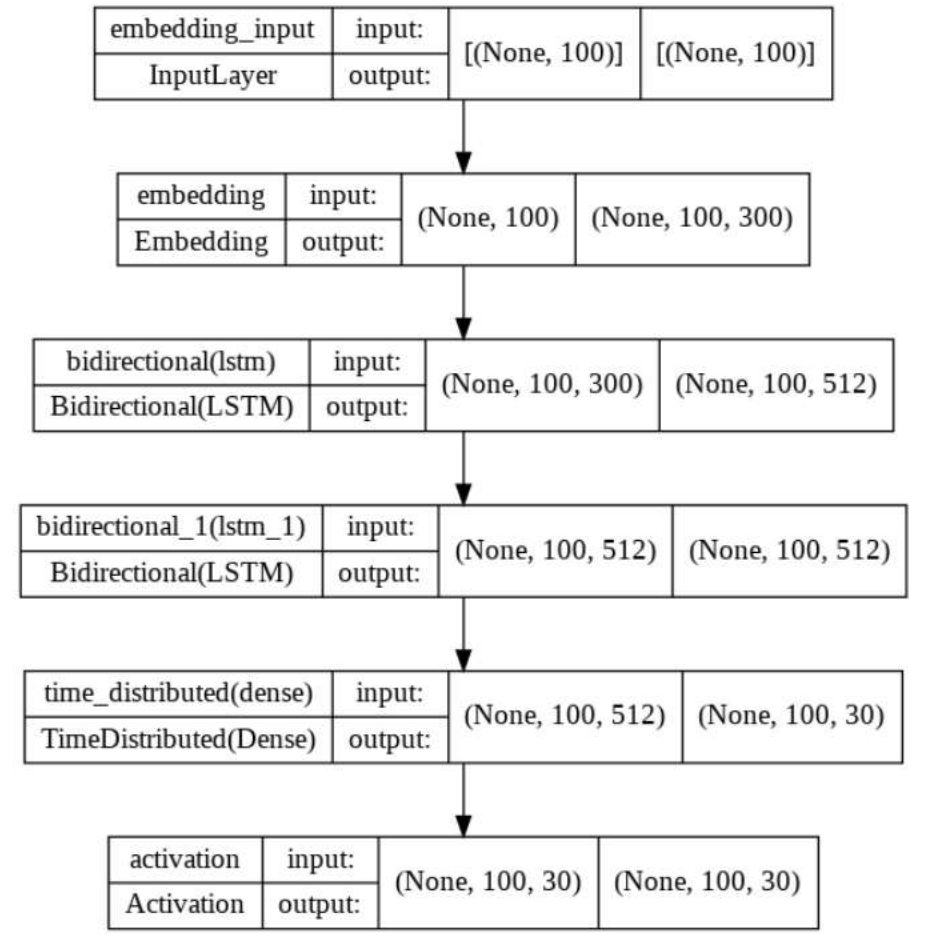


Figure 2.10 B-LSTM Architecture

2.6.3 Creation of the GRU models:

Researchers have attempted to figure out different structures of the LSTM cell regarding the computing costs and the problem of overfitting. The gated recurrent unit is the most popular of these options (GRU). GRU is always easier to train because it is a simpler model than LSTM. Practically, LSTMs and GRUs have nearly replaced ordinary RNNs since they are

more effective and faster to train (despite the larger number of parameters). The following design represents our GRU model architecture.

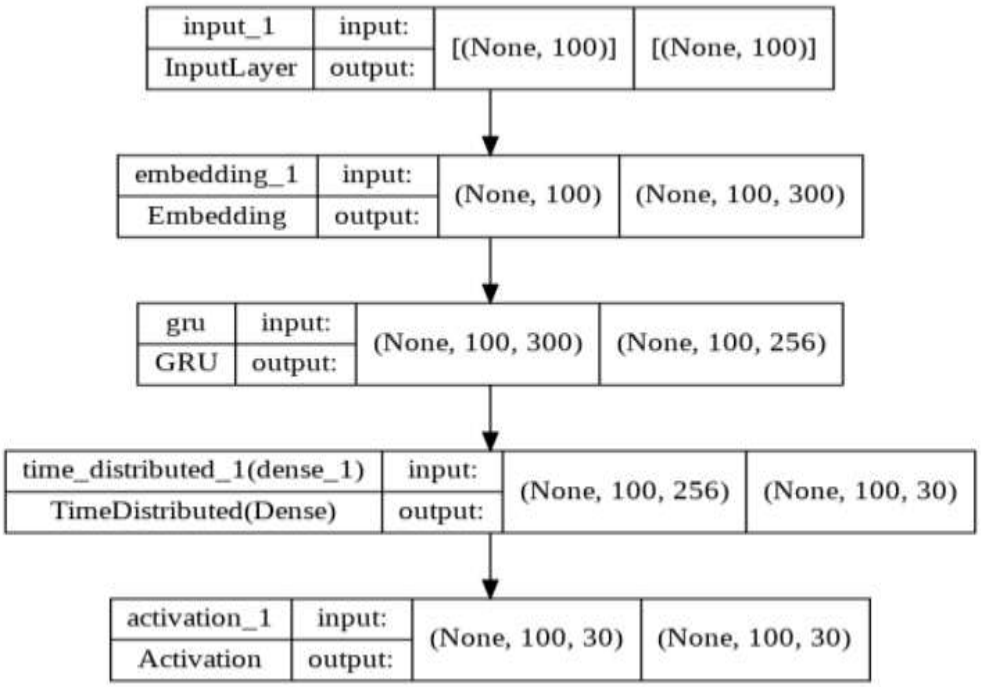


Figure2.11 Architecture of GRU Model

2.6.4 Training and Testing

The preparation of the data is now complete. Let's get started with the modeling by dividing the data into two parts: train and test.

The data contains a little over 2782 articles contain pairs of words and some of the pairs toward the end of the file are constructed many short and long sentences. This is a good number of sentences for developing a small model for Pos Tagging. The complexity of the model increases with the number of words, length of phrases, and size of the vocabulary

Further, we will then take the first (90%) of those as examples for training and the remaining (10%) examples to test the fit model before the creation of each models.

2.6.5. Compilation the Models

After defining a model. Compilation requires a number of parameters to be specified, tailored to training your network. Specifically, the optimization algorithm to use to train the network and the loss.

The type of predictive modeling problem imposes constraints on the type of loss function that can be used. below are some standard loss functions for different predictive model types:

- Regression: Mean Squared Error or mean squared error, for short.
- Binary Classification (2 class): Logarithmic Loss, also called cross entropy or binary Crossentropy.

- Multiclass Classification (>2 class): Multiclass Logarithmic Loss or categorical crossentropy.

The most common optimization algorithm is classical stochastic gradient descent, but Keras also supports a suite of other extensions of this classic optimization algorithm that work well. with little or no configuration. Perhaps the most commonly used optimization algorithms because of their generally better performance are:

- Stochastic Gradient Descent, or sgd.

- Adam, or adam.

- RMSprop, or rmsprop.

We can also specify metrics to collect while fitting your model in addition to the loss function. Generally, the most useful additional metric to collect is accuracy for classification problems (e.g. ‘accuracy’ or ‘acc’ for short). The metrics to collect are specified by name in an array of metric or loss function names. [56]

- We compile GRU models and bidirectional LSTM models Check summary of models

2.6.5.1 Fitting Models:

Once the network is compiled, it can be fit, which means adapting the weights on a training dataset. Fitting the network requires the training data to be specified, both a matrix of input patterns, X, and an array of matching output patterns, y.

The network is trained using the Backpropagation Through Time algorithm and optimized according to the optimization algorithm and loss function specified when compiling the model.

The backpropagation algorithm requires that the network be trained for a specified number of epochs or exposures to all sequences in the training dataset. Each epoch can be partitioned into groups of input-output pattern pairs called batches.

This defines the number of patterns that the network is exposed to before the weights are updated within an epoch. It is also an exigency optimization, ensuring that not too many input patterns are loaded into memory at a time.

- - Epoch: One pass through all samples in the training dataset and updating the network weights. LSTMs may be trained for tens, hundreds, or thousands of epochs.
- - Batch: A pass through a subset of samples in the training dataset after which the network weights are updated. One epoch is comprised of one or more batches.

  Once fit, a history object is returned that provides a summary of the performance of the model during training. This includes both the loss and any additional metrics specified when compiling the model, a progress bar is displayed on the command line for each epoch.

  Training can take a long time, from seconds to hours to days depending on the size of the network and the size of the training data.[55]

2.6.5.2 Evaluating the Models :

Once the network is trained, it can be evaluated. The network can be evaluated on the training data, but this will not provide a useful indication of the performance of the network as a predictive model, as it has seen all of this data before. We can evaluate

the performance of the network on a separate dataset, un seen during training. This will provide an estimate of the performance of the network at making predictions for unseen data in the future.

The model evaluates the loss across all of the test patterns, as well as any other metrics specified when the model was compiled, like classification accuracy. A list of evaluation metrix is returned.[55]

Conclusion

In this chapter, our proposed approach system is described, which aims to design a system capable and we explain, take an example for each steps of programming and plan the architecture of the great models of the Arabic post tagging solution.

Chapter 03:

Implementation and Result

- Introduction:

Implementation is the most important phase after approach used. The choice of development tools greatly influences the cost of programming time as well as the flexibility of the product in question. This phase consists of mentioning: The technological used tools, a description of the work environment and identification of proposed application for our approach.

3.1 Development environment

3.1.1 Python

Python is a high-level, interpreted and general-purpose dynamic programming language that focuses on code readability. The syntax in Python helps the programmers to do coding in fewer steps as compared to Java or C++. The language founded in the year 1991 by the developer Guido Van Rossum has the programming easy and fun to do. The Python is widely used in bigger organizations because of its multiple programming paradigms. They usually involve imperative and object-oriented functional programming. It has a comprehensive and large standard library that has automatic memory management and dynamic features.[36]

3.1.2 Python Features

★ ***Open source:*** Python is publicly available open-source software, anyone can use source code that doesn't cost anything. 2.2.2 Easy-to-learn: Popular (scripting/extension) language, clear and easy syntax, no type declarations, automatic memory management, high-level data types and operations, design to read (more English like syntax) and write (shorter code compared to C, C++, and Java) fast.

★ ***High-level Language:*** High-level language (closer to human) refers to the higher level of concept from machine language (for example assembly languages). Python is an example of a high-level language like C, C++, Perl, and Java with low-level optimization

★ ***Portable***: High level languages are portable, which means they are able to run across all major hardware and software platforms with few or no change in source code. Python is portable and can be used on Linux, Windows, Macintosh, Solaris, FreeBSD, OS/2, Amiga, AROS, AS/400 and many more.

→ ***Object-Oriented***: Python is a full-featured object-oriented programming language, with features such as classes, inheritance, objects, and overloading.

→ **Python is Interactive**: Python has an interactive console where you get a Python prompt (command line) and interact with the interpreter directly to write and test your programs. This is useful for mathematical programming.

→ ***Interpreted:*** A Python program are interpreted, takes source code as input, and then compiles (to portable byte-code) each statement and executes it immediately. No need to compiling or linking.

→ ***Extendable:*** Python is often referred to as a "glue" language, meaning that it is capable to work in mixed-language environment. The Python interpreter is easily extended and can add a new built-in function or modules written in C/C++/Java code.

→ ***Libraries:*** Databases, web services, networking, numerical packages, graphical user interfaces, 3D graphics, others.

→ ***Supports:*** Support from online Python community

3.1.3 Python module used

→ ***RegEx Module:*** A RegEx, or Regular Expression is a sequence of characters that forms a search pattern. RegEx can be used to check if a string contains the specified search pattern.

→ Import the RegEx module: Import re [57]

→ ***string Module:*** The string module contains a number of functions to process standard Python strings.

→ Import the string module: import string [58]

→ ***IO module:*** The IO module provides Python's main facilities for dealing with various types of I/O. There are three main types of I/O: *text I/O*, *binary I/O* and *raw I/O*Text I/O expects and produces str objects. This means that whenever the backing store is natively made of bytes (such as in the case of a file), encoding and decoding of data is made transparently as well as optional translation of platform-specific newline characters.

→ The easiest way to create a text stream is with open(). optionally specifying an encoding

→ Import the IO module: Import IO[59]

- ***The operation system module:*** This module provides a portable way to use operating system-dependent features. If you just want to read or write a file, see open(), if you want to manipulate file paths, see the os.path module, and if you want to read all the lines of all the files from the command line, see the fileinput module. For creating temporary files and directories, see the tempfile module, and for handling high-level files and directories, see the shutil module.
- Notes on the availability of these features:
- The design of native Python modules dependent on the operating system is that the same feature uses the same interface. For example, the function returns information about *path* statistics in the same format (which originated in the POSIX interface).os.stat(path).

★ Extensions specific to a certain operating system are also available through the OS module, but using them is of course a threat to portability.

★ All functions that accept paths or file names accept both *bytes* and *strings*, and if a path or file name is returned, it will be of the same type.

★ On VxWorks, os.open, os.fork, os.execv and os.spawn are not supported.

★ ***Python utf-8 mode:*** The Python UTF-8 Mode ignores the locale encoding and forces the usage of the UTF-8 encoding:

★ Use UTF-8 as the filesystem encoding.

★ sys.getfilesystemencoding() returns.'UTF-8'.

★ locale.getpreferredencoding() returns (the *do_setlocale* argument has no effect).'UTF-8'.

★ sys.stdin, sys.stdout, and sys.stderrall use UTF-8 as their text encoding, with the error handler being enabled for sys.stdin and sys.stdout (sys.stderr continues to use as it does in the default locale-aware mode)surrogateescapebackslashreplace

★ On Unix, os.device_encoding() returns rather than the device encoding.'UTF-8'[60]

- ***Tensorflow module:***TensorFlow is an open-source Python library designed by Google to develop machine learning models and deep learning neural networks.

TensorFlow Recommenders (TFRS) is a library for creating system templates for recommendation tools. It accompanies you throughout the process of creating a system for recommendation tools: data preparation, formulation, training, evaluation and deployment of the model. It is based on the Keras API. Quickly mastered, it nevertheless allows you to create complex models. TFRS allows you to do the following:

- Create and evaluate flexible recommendation recovery models.
- Freely embed item and user information, and context information, in recommendation templates.
- Train multitasking models that together optimize multiple recommendation goals.

→ Import tensorflow as tf[61]

- ***Keras Module:*** Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result as fast as possible is key to doing good research.
- Import the Keras module***:*** from tensorflow import keras [62]
- ***Pandas module:*** panda is a library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and manipulation operations for numerical tables and time series. Pandas is free software underBSDlicense. Its name is derived from the term *Panel Data*, an econometric term for datasets that include observations over multiple periods of time for the same individuals). Its name is also a play on words on the phrase "Python Data Analysis".[63]
- ***NumPy Module:*** NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices.
- Import the NumPy module: import numpy as np [57]
- ***Matplotlib Module:*** Matplotlib is a low-level graph plotting library in python that serves as a visualization utility.

→ Import the matplotlib module: import matplotlib.pyplot as plt [57]

→ ***sklearn Module:*** Scikit-learn is a machine learning library for Python. It features several regression, classification and clustering algorithms including SVMs, gradient boosting, k-means, random forests and DBSCAN. It is designed to work with Python Numpy and SciPy.

→ Import the sklearn module: import sklearn [62]

3.1.2 Google Colab

3.1.2.1 Google Colab definition

Google Collaboratory widely known as Google Colab is an open-source service provided by Google to any person having a Gmail account. Google Colab provides GPU for research to the people who do not have enough resources or cannot afford one. The Google Colab service provides 12.72 GB of RAM and 358.27 GB of hard disk space in one runtime. Every runtime lasts for 12 hours after which the runtime is reset and the user has to establish a connection again. This is to ensure that people do not use the GPU service for crypto currency mining and other illegal purposes. Different run cases are shown below. Once the user opens a Google Colab file, they need to select a runtime type. There are 3 available options for the same:

1. None (that will use the computer's CPU the user is using)

2. GPU

3.-TPU (especially for tensor processing).

3.1.2.2 Benefits of Colab

Besides being easy to use, the Colab is fairly flexible in its configuration and does much of the heavy lifting for you.

- Python 2.7 and Python 3.6 support.

- Free GPU acceleration.

- Pre-installed libraries: All major Python libraries like TensorFlow, Scikit-learn, Matplotlib among many others are pre-installed and ready to be imported.

- Built on top of Jupyter Notebook.

- Collaboration feature (works with a team just like Google Docs): Google Colab allows developers to use and share Jupyter notebook among each other without having to download, install, or run anything other than a browser.

- Supports bash commands.

- Google Colab notebooks are stored on the drive.[57]

3.1.3 Google Drive:

Google Drive is a free cloud-based storage solution that enables users to:

o Storing files: By default, Google gives 15 GB of free storage space across Gmail, Google Drive and Google Photos. If you ever hit that limit, you can always pay for more space by upgrading to a Google One account.

o Sharing files: Files are private until you share them. But at any time, you can invite others to view, edit, or download files and documents.

o Accessing files from anywhere: Files can be accessed from any smartphone, tablet or computer. Offline access is also available.

o Comprehensive search: Drive can recognize images and even text in scanned documents, making searching for files even easier.

o Secure storage: Drive files are protected by an AES256 or AES1283 encryption, the same security protocol used on other Google services.

Here is a breakdown of the most commonly used Google Drive applications:

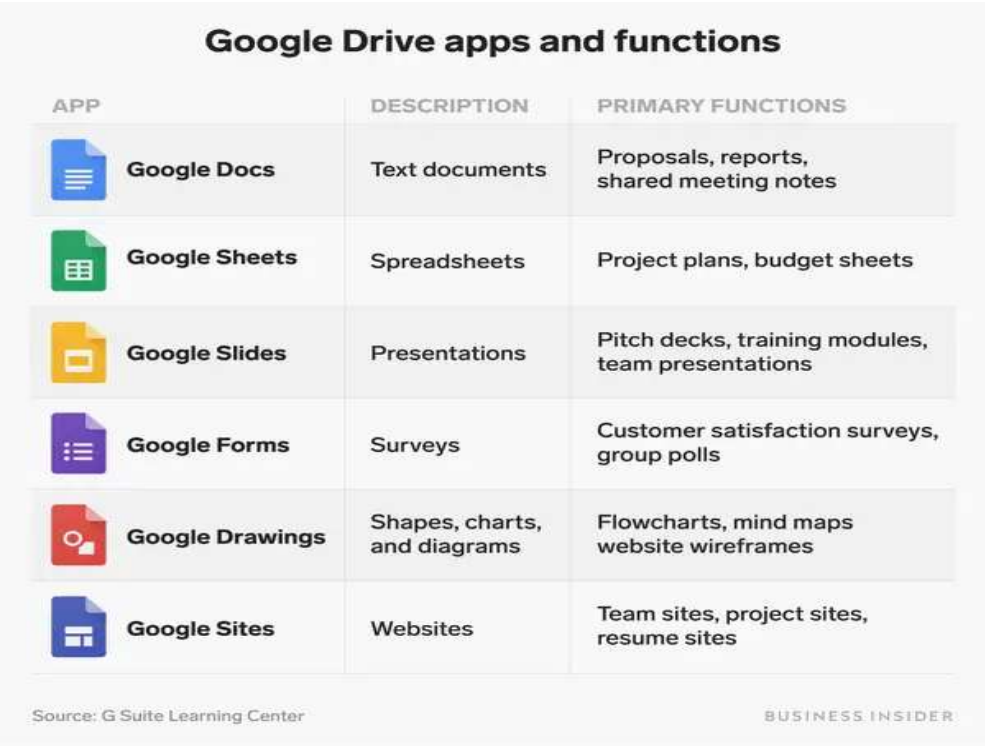


Figure3.1 Google Drive applications and functions.

3.2 Implementation of our approach

3.2.1 Dataset

We used the dataset that is mentioned in the chapter 2.

Arabic dataset contains 2782 articles and each article has numbers of pair words and tags [word in Arabic language + tag in English language]. All articles have 20,500 sentences samples.

3.2.2 Data Download



Figure3.2 Dataset Example from DataSet

Assuming that the data is in the Google Drive, we use the function in the following figure for calling the data that was copied in the Google Drive that must be related to Google Colab.

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')

     Drive already mounted at /content/drive; to attempt to forcibly remount,

[ ]  cd drive/MyDrive/

     [Errno 2] No such file or directory: 'drive/MyDrive/'
     /content/drive/MyDrive

[ ]  import os
     listFiles=[i for i in os.listdir('corpuspostt')  if i.endswith(".txt")]
     import pandas as pd
     listDf=[]
     for i in listFiles:
       df=pd.read_csv('corpuspostt/'+i,delimiter="/",header=None)
       listDf.append(df)
     df = pd.concat(listDf)
     df.head()
```

Figure3.3 Connection the Dataset with our Program

3.2.3Data Decomposition

We construct all the pair words that are in the article into sentences possible up to the punctuation and putting it in X and Y. After we add them into features (Inputset) and labels (Outputset). We illustrate a sample of sentence in the features and labels:

```
inputset= ['التي' 'قد' 'توقع' 'صاحبها' 'في' 'الخطأ' 'و' 'تجعل' 'منه' 'على' 'شفا'
 'حفرة' 'من' 'ارتكاب' 'الجرم' 'الذي' 'يستحق' 'عليه' 'العقاب']
outputset= ['WP' 'RP' 'VBD' 'NN' 'IN' 'DTNN' 'CC' 'VBP' 'NNP' 'IN' 'NN' 'NN' 'IN'
 'NN' 'DTNN' 'WP' 'VBP' 'NN' 'DTNN']
```

Figure3.4 Example Features and Labels

This figure represents just a small part of matrix in the inputset (the phrases moved in features):

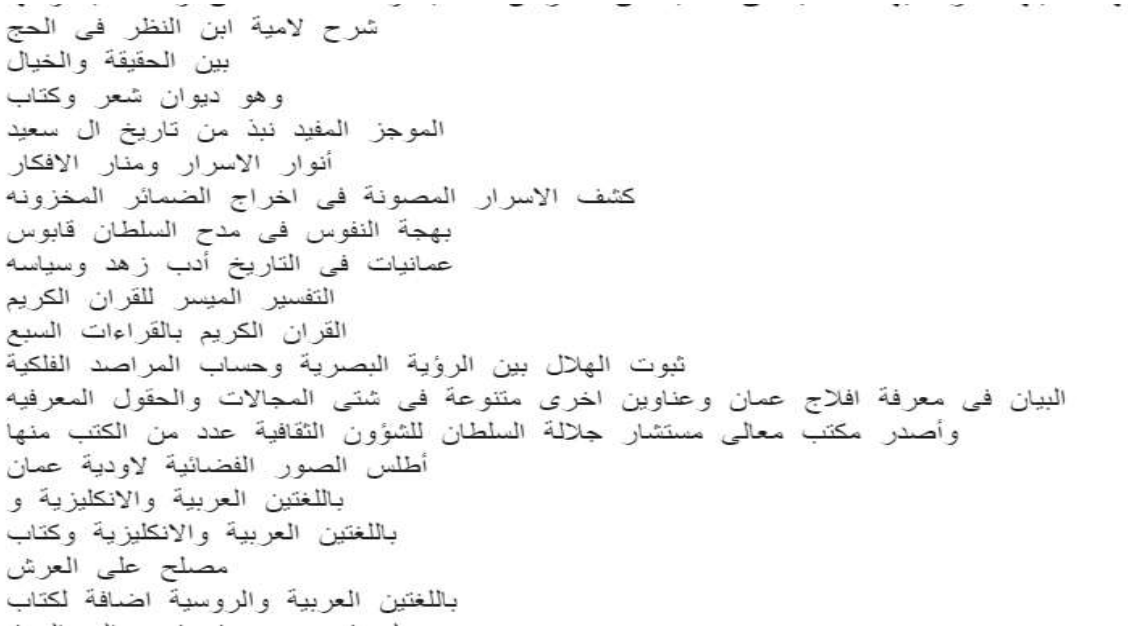


Figure3.5 Construction of Sentences (Features)

3.2.4 Vectorizations

In this part, we turn sentences into vector to word's vector. This operation called by the tokenization in the aim of realizing this task. We used the TensorFlow's tokenizer for making tokenizer then we fit it on the data. After that, we use it to encode all words and all tags in the features and labels by word_tokenizer.texts_to_sequences. We took two examples from our execution as the following figure shows:

```
X1= [32576, 8721, 14603, 25150, 4088, 55361, 31043, 7515, 53129, 35380, 47031, 4088, 49619, 7651, 53517, 55133]
X2= [28618, 5700, 28183, 25150, 14211, 58467, 37097, 21249, 22912, 14380, 50303, 13870, 7267, 35089, 1, 1, 42947, 1, 43449]
Y1= [18, 10, 4, 8, 4, 18, 9, 14, 18, 14, 18, 4, 10, 9, 14, 18]
Y2= [8, 25, 4, 8, 4, 14, 10, 18, 4, 14, 4, 18, 4, 21, 14, 3, 18, 14, 4]
```

Figure 3.6 Encoding Examples

3.2.5 Pad sequences:

In order to carry out this stage to set up a matrix where all of sentences share the same length, it is necessary to complete the vector of the vectorized sentences by zeros when there is an empty space.

We need to import the Tensorflow using the pad sequence and define the max sequence length which is 100 and replace the empty victor with zero as the figure demonstrates:

```
[43032 32616 42984 41902 17950 47420 38272 40787  9226 36229 31274 13338
 55116  3138     1     1 59461     1 51789     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0     0     0
     0     0     0     0]
[10 24 23 16 23 10 22  6 10  6 10 23 24 22  6 10  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0]
```

Figure3.7 Padded Sequences Examples

3.3 Split the Data into Train /Test

The data we work on contains 2256 articles of part of speech and each article contains more than 1000 sentences, but in the preprocessing part, we limited the sentences and got rid of all the sentences that contain more than 1OO words or less than 2 words in sentences. We got 20346 approximately 20500 numbers of sentences.

This is a good number of examples sentences for developing a PosTagging model. The complexity of the model increases with the number of examples, the length of sentences and the vocabulary size.

We will simplify the problem by reducing the dataset to the first 20500 examples in

the file that will be the shortest phrases in the dataset.

Further, we will then take the first 18311 of these as examples for training and the

Remaining 2035 examples for testing to fit our models.

```
# 90% data training, 10% data test
(train_sentences,
 test_sentences,
 train_tags,
 test_tags) = train_test_split(sentences, sentence_tags, test_size=0.1,random_state=42)
```

Figure 3.8 Split the Data into Train and Test

```
import numpy as np
print ("train X=",np.array(train_sentences_X.shape))
print ("test X=",np.array(test_sentences_X.shape))
print ("train Y=",np.array(train_tags_y.shape))
print ("test Y=",np.array(test_tags_y.shape))
```

Figure3.9 Function to Post X Shape and Y Shape

```
train X= [18311   100]
test X= [2035  100]
```

Figure 3.10 Shape of X_Train and X_Test

3.4 Model Development

Here, we will explain our work to create RNN through selecting two models which are: bidirectional LSTM and GRU to compare our result knowing their accuracy result.

3.4.1. Create the first model Bidirectional LSTM

- **One hot encoding:**

At first, we don't need to encode the input in the model but we need to encode tags by to_categorical which are converted into binary vector numerical data for output.

```
from tensorflow.keras.utils import to_categorical

y_train2, y_test2= to_categorical(y_train),to_categorical(y_test)
NUM_CLASSES= y_train2.shape[-1]
NUM_CLASSES
```

Figure 3.11 One Hot Encoding the OutPut

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]]
```

Figure3.12 Execution One Hot Encoding

- **Create the sequential model:**We Import the sequential from keras for create it

```
from keras.models import Sequential
model = Sequential()
```

Figure 3.13 Creation the Sequential Model

- **Create Embedding Layer models:**

We import the Embedding from keras and we initialize the Embedding size=300.

The first, we create embedding layer and usually the first layer in text problems in its input. we take it as the vocabulary size (number of unique words in data). In its output the length of vector for each represented word (Embedding size). In the length of input sequence (max of sequence length)

We add to our model the embedding layer with functions that are in the figure:

```
rnn_model.add(Embedding(input_dim = vocab_size,
                        output_dim = EMBEDDING_SIZE,
                        input_length = MAX_SEQ_LENGTH, ))
```

Figure3.14 Creation Embedding Code

- **Create Bidirectional LSTM:** After we import the bidirectional and LSTM, we add to the model two layers: bidirectional LSTM1 and bidirectional LSTM2 which cotain 256 LSTM cells and return whole sequences for each one with functions that are in the figure:

```
model.add(Bidirectional(LSTM(256, return_sequences=True)))
model.add(Bidirectional(LSTM(256, return_sequences=True)))
```

Figure 3.15 Creation of Bidirectional LSTM Layers

- **Creation of TimeDistribibuted and Dense Layers:**

After we import theTimeDistribibuted and Dense, we create them and add the function SoftMax in the activation because we have many outputs with functions that are in the figure:

```
rnn_model.add(TimeDistributed(Dense(NUM_CLASSES)))
rnn_model.add(Activation('softmax'))
```

Figure 3.16 Creation of TimeDistributed and Activation

- **Compilation of the model:**

In the last part, we compile our model defining the optimizer (the optimization algorithm is **adam**) and loss (the function loss is **categorical_crossentropy**) and for evaluating the performance of classification models with many output metrics (the metrics is **accuracy**)

Next, a summary of the defined model is printed, allowing us to confirm the model

Configuration.

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

model.summary()
```

Figure 3.17 Compile and Summary the B-LSTM

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_2 (Embedding)      (None, 100, 300)          19107600

 bidirectional_2 (Bidirectio  (None, 100, 512)          1140736
 nal)

 bidirectional_3 (Bidirectio  (None, 100, 512)          1574912
 nal)

 time_distributed_2 (TimeDis  (None, 100, 30)           15390
 tributed)

 activation_2 (Activation)    (None, 100, 30)           0

=================================================================
Total params: 21,838,638
Trainable params: 21,838,638
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.18 Shows B-LSTM Summary

3.4.2 The second models GRU:

- Import the Embedding, TimeDistribibuted, Dense and Activation from keras layers for creation our layers.
- Create sequential template on the name model.
- We add the input layer and the embedding layer.
- Add the GRU layer which contain 256 GRU cells and return the sequence. If it True, its means return whole sequences. If return sequences are False, return single output of the end of the sequences.

```
model.add(GRU(256, return_sequences=True))
```

Figure3.19 Creation of GRU Layer

- Add to the model timeDistributed and Dense layer.
- Compile all the model when we created with **categorical_crossentropy** function of loss and **Adam** algorithm of optimization and **accuracy** of metrics.
- Check Summary of the model.
- Get this figure:

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_4 (Embedding)      (None, 100, 300)          19107600

 gru_3 (GRU)                  (None, 100, 256)          428544

 time_distributed_4 (TimeDis  (None, 100, 30)           7710
 tributed)

 activation_4 (Activation)    (None, 100, 30)           0

=================================================================
Total params: 19,543,854
Trainable params: 19,543,854
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.20 Shows GRU Summary

3.5 Evaluation :

We will evaluate all models on the test dataset.

The model1 and models2 should perform very well on the train dataset and they have ideally been generalized to perform well on the test dataset.

- **The First Models (Bidirectional LSTM):**

We train the model of Bidirectional LSTM with 10 epochs.

```
rnn_training = rnn_model.fit(X_train, y_train2, batch_size=128, epochs=10, validation_data=(X_test, y_test2))
```

Figure3.21 Fit of the B-LSTM Model

```
Epoch 1/10
134/134 [==============================] - 20s 108ms/step - loss: 0.5306 - acc: 0.8533 - val_loss: 0.4444 - val_acc: 0.8650
Epoch 2/10
134/134 [==============================] - 13s 100ms/step - loss: 0.3029 - acc: 0.9089 - val_loss: 0.1684 - val_acc: 0.9483
Epoch 3/10
134/134 [==============================] - 13s 100ms/step - loss: 0.0946 - acc: 0.9718 - val_loss: 0.1041 - val_acc: 0.9684
Epoch 4/10
134/134 [==============================] - 13s 99ms/step - loss: 0.0431 - acc: 0.9870 - val_loss: 0.1065 - val_acc: 0.9705
Epoch 5/10
134/134 [==============================] - 13s 98ms/step - loss: 0.0262 - acc: 0.9922 - val_loss: 0.1159 - val_acc: 0.9712
Epoch 6/10
134/134 [==============================] - 13s 97ms/step - loss: 0.0170 - acc: 0.9951 - val_loss: 0.1239 - val_acc: 0.9721
Epoch 7/10
134/134 [==============================] - 13s 98ms/step - loss: 0.0115 - acc: 0.9968 - val_loss: 0.1386 - val_acc: 0.9710
Epoch 8/10
134/134 [==============================] - 13s 98ms/step - loss: 0.0081 - acc: 0.9978 - val_loss: 0.1453 - val_acc: 0.9708
Epoch 9/10
134/134 [==============================] - 13s 99ms/step - loss: 0.0056 - acc: 0.9986 - val_loss: 0.1619 - val_acc: 0.9708
Epoch 10/10
134/134 [==============================] - 13s 99ms/step - loss: 0.0039 - acc: 0.9990 - val_loss: 0.1757 - val_acc: 0.9713
```

Figure 3.22 Result of Fit for B-LSTM Model

```
loss_BLSTM, accuracy_BLSTM = rnn_model.evaluate(X_test, y_test2, verbose = 1)
print("Loss: {0},\nAccuracy: {1}".format(loss_BLSTM, accuracy_BLSTM))
```

Figure 3.23 Function of Evaluation of Accuracy B-LSTM

```
60/60 [==============================] - 1s 16ms/step - loss: 0.1757 - acc: 0.9713
Loss: 0.17571118474006653,
Accuracy: 0.971303403377533
```

Figure3.24 Result Accuracy of Training the B-LSTM

```
[ ]  from sklearn.preprocessing import MultiLabelBinarizer
     from sklearn_crfsuite.metrics import flat_f1_score


     flat_f1_score(y_test,
                   y_pred,average='weighted'
     )
```

```
0.9700057824477661
```

Figure3.25 Result F-scores of Training B-LSTM

```
from sklearn_crfsuite.metrics import flat_precision_score,flat_recall_score  ,sequence_accuracy_score
pre1=flat_precision_score(
    y_test, y_pred,average='weighted'
)
rec1=flat_recall_score(
    y_test, y_pred,average='weighted'
)
print("presicion : ",pre1)
print("recall ",rec1)
print("acc ",accuracy_BLSTM)
```

Figure 3.26 Function to Calculate the Evaluation of the Precision and Recall

```
presicion :  0.9708645471368986
recall  0.9700474934036939
acc  0.9708970785140991
```

Figure3.27 Result of the Precision and Recall of B-LSTM Model

- **The Second Models (GRU):**

```
GRU_training = model.fit(X_train, y_train2, batch_size=128, epochs=10, validation_data=(X_test, y_test2))
```

Figure 3.28 Fit of the GRU Model

```
Epoch 1/10
134/134 [==============================] - 4s 28ms/step - loss: 0.0034 - accuracy: 0.9987 - val_loss: 0.1955 - val_accuracy: 0.9694
Epoch 2/10
134/134 [==============================] - 4s 28ms/step - loss: 0.0033 - accuracy: 0.9988 - val_loss: 0.2010 - val_accuracy: 0.9693
Epoch 3/10
134/134 [==============================] - 4s 28ms/step - loss: 0.0031 - accuracy: 0.9988 - val_loss: 0.2059 - val_accuracy: 0.9690
Epoch 4/10
134/134 [==============================] - 4s 28ms/step - loss: 0.0030 - accuracy: 0.9988 - val_loss: 0.2086 - val_accuracy: 0.9689
Epoch 5/10
134/134 [==============================] - 4s 28ms/step - loss: 0.0029 - accuracy: 0.9989 - val_loss: 0.2116 - val_accuracy: 0.9689
Epoch 6/10
134/134 [==============================] - 4s 28ms/step - loss: 0.0028 - accuracy: 0.9989 - val_loss: 0.2181 - val_accuracy: 0.9689
Epoch 7/10
134/134 [==============================] - 4s 28ms/step - loss: 0.0028 - accuracy: 0.9989 - val_loss: 0.2188 - val_accuracy: 0.9690
Epoch 8/10
134/134 [==============================] - 4s 28ms/step - loss: 0.0027 - accuracy: 0.9989 - val_loss: 0.2244 - val_accuracy: 0.9689
Epoch 9/10
134/134 [==============================] - 4s 28ms/step - loss: 0.0027 - accuracy: 0.9989 - val_loss: 0.2276 - val_accuracy: 0.9687
Epoch 10/10
134/134 [==============================] - 4s 28ms/step - loss: 0.0027 - accuracy: 0.9989 - val_loss: 0.2273 - val_accuracy: 0.9690
```

Figure 3.29 Result of Fit for GRU Model

Next train the model2, we must to evaluate the result of its accuracy, precision and recall.

```
loss_GRU, accuracy_GRU = model.evaluate(X_test, y_test2, verbose = 1)
print("Loss: {0},\nAccuracy: {1}".format(loss_GRU, accuracy_GRU))
```

Figure 3.30 Function of Evaluation GRU Model

```
60/60 [==============================] - 0s 7ms/step - loss: 0.2273 - accuracy: 0.9690
Loss: 0.22727957367897034,
Accuracy: 0.9689921140670776
```

Figure3.31 Result of Training GRU Model

```
[ ] from sklearn_crfsuite.metrics import flat_precision_score,flat_recall_score  ,sequence_accuracy_score
    y_pred=model.predict(X_test).round().argmax(axis=-1)
    y_pred[0]
    pre2=flat_precision_score(
        y_test, y_pred,average='weighted'
    )
    rec2=flat_recall_score(
        y_test, y_pred,average='weighted'
    )
    print("presicion : ",pre2)
    print("recall ",rec2)
    print("acc ",accuracy_GRU)
```

Figure3.32 Function of Evaluation the Precision and Recall

```
presicion :  0.9685720378119153
recall  0.968311345646438
acc  0.9689393043518066
```

Figure3.33 Result of the Precision and Recall of GRU Model

3.6 Comparison between the two Models

When comparing GRU to B-LSTM, there is a reduction in params. As a result, we see a large increase in computing efficiency while the model's performance remains relatively unchanged.

Model GRU:CPU times: user 31.7 s, sys: 1.7 s, total: 33.4s; Wall time: 43 s

Model BLSTM: CPU times: user 38.7 s, sys: 1.14 s, total: 118.4s; Wall time:2m

We can carry out our work by a table that expresses and summarizes our work:

model of Recurrent Neural Network	Bidirectional-LSTM	GRU
Bidirectional LSTM size	256	0
GRU layer size	0	256
Optimization methode	categorical_crossentropy	categorical_crossentropy
Function loss	Adem	Adem
Activation function	softMax	softMax
Percentage of validation split	10%	10%
Processing unit type	Graphical Processing Unit	Graphical Processing Unit
Numbers of traing epochs	10	10
Time of execution	2min	43s
Accuracy	97,130%	96,89%

Table 3. 1 Characteristic of Two Models

We have 97,130% accuracy for the models of Bidirectional LSTM (First model) and 96,89% accuracy for GRU (second model). We can say that there is not a great difference but we can say the first model is better than second model. The representation of the bar proves that.

The accuracy of two models remains nearly the same, but we notice that the time taken by an GRU is faster than B-LSTM

```
import matplotlib.pyplot as plt


plt.bar(["accuracy_BLSTM"," accuracy_GRU"],[accuracy_LSTM,accuracy_GRU])
plt.bar(["loss_BLSTM"," loss_GRU"],[loss_LSTM,loss_GRU])
```

Figure3.34 Function of Compraraison the Accuracy

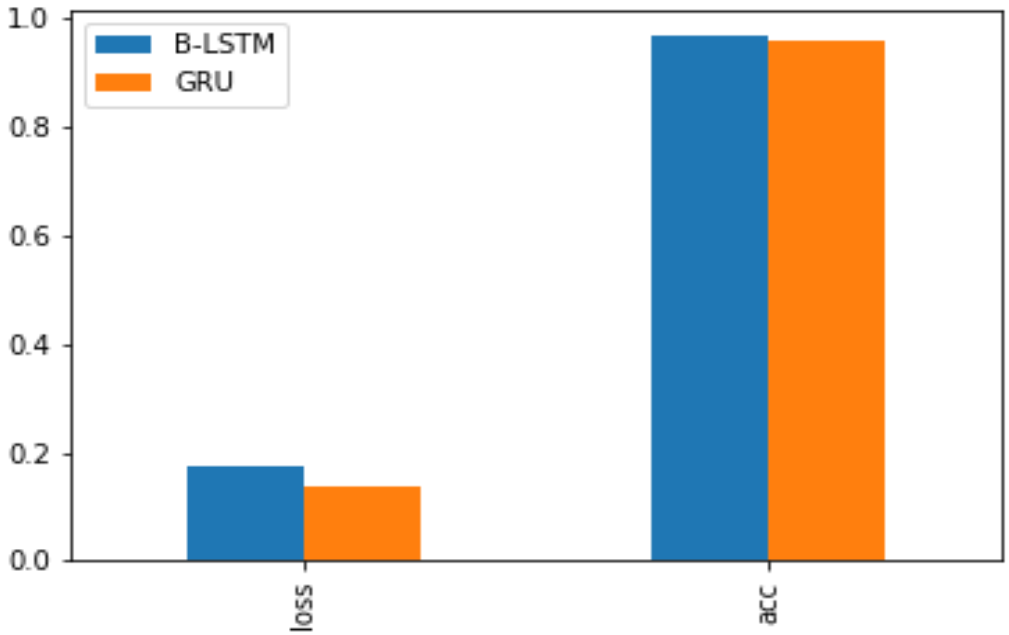


Figure 3.35 Different Loss and Accuracy of two Models

We can also compare evaluation of the two models with precision and recall. First, we notice that the precision of B-LSTM is almost the same as the accuracy. However, the recall decreases a little comparing it to accuracy and precision. Then, we notice a gradual decreas of the precision and recall comparing it to accuracy in the GRU model.

So, we figure out that B-LSTM model is better than GRU model in terms of accuracy, precision and recall. The following bar demonstratethat.

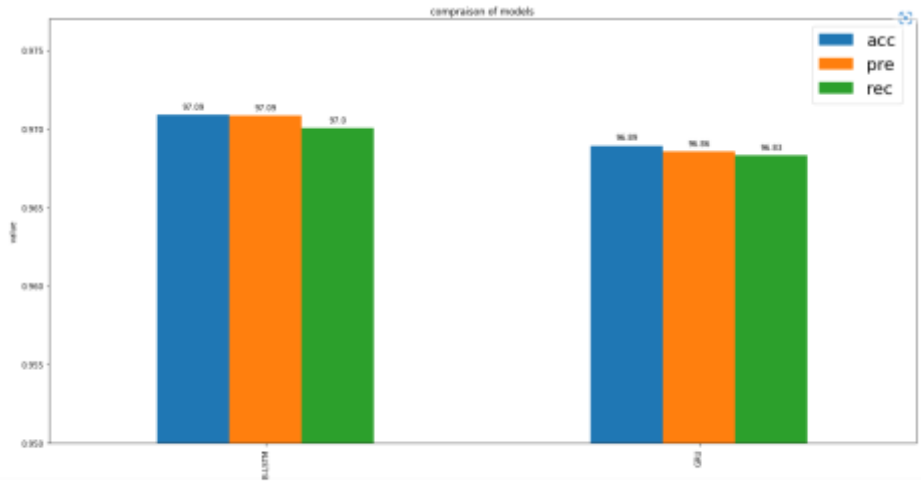


Figure 3.36 Comparison of Evaluation of two Models

3.7 Test of Models:

In this task, we have the full-text revision focused on the development of a part of the system written without diacritization for the Arabic language using a deep learning approach and techniques such as Bi-LSTM and GRU

3.7.1 First model(B-LSTM):

```
pred=rnn_model.predict( test_sentences_X)
pred=pred.argmax(axis=-1)
real=test_tags_y
cpt=0
for i,j,k in zip(test_sentences,real,pred):
  print(j[:len(i)]==k[:len(i)])
  if cpt==10:
    break
  cpt+=1
```

Figure 3.37 Function of Testing the First Model

-Result :

```
[ True  True  True]
[ True  True  True  True  True  True  True  True  True  True  True  True]
[ True  True  True  True  True  True False  True  True  True  True False
  True  True  True  True  True  True  True  True  True  True  True  True
  True False  True  True  True  True]
```

Figure 3.38 Testing in B-LSTM

We have three (03) mistakes in Bidirectional LSTM model in the third sentence.

3.7.2 Second Model(GRU):

```
pred=model.predict( test_sentences_X)
pred=pred.argmax(axis=-1)
real=test_tags_y
cpt=0
for i,j,k in zip(test_sentences,real,pred):
  print(j[:len(i)]==k[:len(i)])
  if cpt==10:
    break
  cpt+=1
```

Figure 3.39 Function of Testing the Second Model

Result:

```
[ True  True  True]
[ True  True  True  True  True  True  True  True  True  True  True  True]
[ True  True  True  True  True  True  True  True  True  True  True False]
  True  True  True  True  True  True  True  True  True  True  True  True
 False False  True  True  True  True]
```

Figure 3.40Function of Testing the Second Model (GRU)

In GRU model, we have 3 mistakes in the third sentence.

We will discuss the results of the different models and tags used in training and testing. The approaches applied to Arabic POS marking have yielded the same number of mistakes but in different places in the same sentence. Even with the same sentences and epochs, they give us the same number of mistakes although both models did not have the same result of accuracy.

3.8 Related work:

As stated earlier, there are many POS marking techniques which are the three types of approaches: The statistical approach, the rules-based approach and the hybrid approach.

In this part we want to shed light on the previous results that deals with Arabic Pos Tagger with the same models or the same corpus for comparison. We have three articles of Arabic Pos Tagging. The first article used our dataset KALIMAT and the second used another dataset. The third article evaluated its results with precision, recall,accuracyand F-measure. We will compare our results with the articles mentioned before.

First article:

Deep learning has been shown to perform well in various natural language processing tasks such as POS, NER, sentiment analysis, language modelling, and other tasks. In addition, it performs well without the need for manually designed external resources or time-consuming feature engineering. In this study, the focus is on using Long Short-Term Memory (LSTM), Bidirectional Long Short-Term Memory (BLSTM), Bidirectional Long Short-Term Memory with Conditional Random Field (BLSTM-CRF), and Long Short-Term Memory with Conditional Random Field (LSTM-CRF) deep learning techniques for tasks in Syntactic level and comparing their performance. The models are trained and tested by using the **KALIMAT** corpus. The obtained results show that a BLSTM CRF model overcame the other models in the NER task. As for the POS task, the BLSTM-CRF model obtained the highest F1-score compared to the other models.[67]

Models	Measurement
	F1
LSTM	80.2%
BLSTM	80.4%
LSTM-CRF	79.7%
BLSTM-CRF	**81.1%**

Table 3.2 F1-Score of POS Models Tested By Test Datasets For 40 Epochs. The highest F-Score Is Bolded.

Though we have used the same corpus(the last update was in 2015) and the same models(B-LSTM), our evaluation of F-measure is more superior than this represented work. Although, this work was done last year and they used 40 epochs in the testing program, we have used only 10 epochs with more effective result which was 90.00%. As consequence, our program can be considered as most effective programs.

Second Article:

We give in this work a comparison between the three systems: the StanfordPOS, the Arabic POS Tagger Library and our system. To do this, we use the global metrics the precision, the recall, the F-measure and the accuracy. We present in Table 6 the results. We note through these results that the best performances for the four criteria are obtained with our system. We achieved an accuracy around 94 %, which exceeds the others with more than 10 %. On the other hand, the F-measure of our system is also greater than those of the other systems with 20 %. Comparing the three systems, we can make the following.[64]

	StanfordPOS (%)	QCRI (%)	Our system (%)
Precision	75.65	73.11	87.30
Recall	54.66	64.62	88.98
F-measure	63.46	68.60	88.13
Accuracy	72.68	84.54	94.02

Table3.3 A hybrid Arabic POS tagging for simple and compound morphosyntactic tags

Our accuracy for two models (B-LSTM=97.13% and GRU=96.89%) is better than the first article's work which has the accuracy between 72.68% and 94.02%.

In addition to that, even the precision(B-LSTM=97.09%; GRU=96.86%) and the recall(B-LSTM= 97.0%; GRU=96.83%) of our work are greater than the first article's work.

Third Article

This paper focuses on comparing between using Support Vector Machine based ranking (SVMRank) and Bidirectional Long-Short-Term-Memory (bi-LSTM) neuralnetwork based sequence labeling in building a state-of-the-art Arabic part-of speech tagging system. Using SVMRank leads to state-of-the-art results, but with a fair amount of feature engineering. Using bi-LSTM, particularly when combined with word embeddings, may lead to competitive POS-tagging results by automatically deducing latent linguistic features. However, we show that augmenting biLSTM sequence labeling with some of the features that we used for the SVMRank - based tagger yields to further improvements. We also show that gains realized using embeddings may not be additive with the gains achieved due to features. We are open-sourcing both the SVM Rank and the bi-LSTM based systems for the research community[65]

State-of-the-art: MADAMIRA	95.3	
System	Truth Segmentation	Farasa Segmentation
SVM^{Rank} (Clitic)	95.9	94.9
SVM^{Rank} (Word)	96.2	95.3
bi-LSTM (Clitic)	94.5	93.5
bi-LSTM (Clitic) w/embeddings	95.0	92.4
bi-LSTM (Clitic) w/features	96.1	95.0
bi-LSTM (Clitic) w/features + embeddings	95.5	94.7

Table 3.4: The accuracy of our POS tagger on the WikiNews dataset (Darwish and Mubarak, 2016) against Madamira

Our work (B-LSTM and GRU models) has achieved the best results of accuracy in both SVM and all Bidirectional LSTM when we compare it with this represented work.

Conclusion:

In this chapter, we explained and summarized all our work and do the test on the two models. We did all our best to develop the whole program using various critical metrics comparing the results of our models, then with other works of the same theme and dataset used.

General Conclusion

Our works was based on a deep learning approach based on various Recurrent Neural Network architectures (Bi-LSTM and GRU) for Arabic texts POS TAGGING, in order disambiguate the majority of ambiguous and Unknown words Arabic text.

The first steps of pre-processing were implemented we have decomposed the data into features and labels. Then, we converted them to number what is called vectorization, also we must have the same length of all sentences this is what we paid to replace the empty vector to 0 that is called the pad sequences.

In our work we have many outputs (29 classes) to converted it in binary classes by one-hot-encoding to inter them in the RNN. After that we splinted the data into parts: 90% for train and the rest for test part. We created our deep learning architecture for Recurrent Neural Networks with two different models: Bidirectional-Long-Short-Term Memory and Gated Recurrent Unit. In the first, we started with creation of Hiddens layers for each model, we created the model as sequential, we created the Embedding layers to create a spatial representation of the word and feed it into a B-LSTM layers and GRU layer that outputs a hidden vector before created the bidirectional LSTM and GRU then we added to our models TimeDistributed and Dense layers activated by SoftMax function. In the end, we compiled and summarized two models. We have a 97.08% accuracy of the Bi-LSTM model and 96.80% precision of GRU model as result, we compare the best models from mistakes and the best percentage of accuracy and lost achieved by the system based on (BI-LSTM). We can say that our system out perform the most famous related works.

Even though this model is already, powerful improvement could be approved:

- We can train it on a larger dataset (KALIMAT).
- We also could increase the number of cells and the number of LSTM layers to two layers in the model bidirectional LSTM, instead of just one layer in the GRU models.
- We could also use a pre-trained integration layer like vectorization and the one hot encoding and padded sequences for embedding layer calculations.
- Finally, we could use the attention mechanism, which is one of the major improvements in natural language processing.

  This work has served us to improve and refine our knowledge in the field of NLP and POS TAGGING. The results obtained largely meet our objectives, set at the beginning of this work and we hope to continue to improve and perfect this system to have better results in the future.

Bibliography

- [1]:https://www.academia.edu/37014446/Arabic_Language_Historic_and_Sociolinguistic_Characteristics
- [2] : https://www.yourdictionary.com/arabic
- [3] :[/ArabicLanguageHistoricandSociolinguisticCharacteristics.pdf]
- [4] :https://www.arabic-studio.com/wp-content/uploads/2014/09/Arabic-Alphabet-Course.pdf
- [5]:https://www.google.com/search?q=les+lettres+en+arabes&rlz=1C5CHFA_enDZ806DZ806&sxsrf=ALiCzsbgDv2v402Q4HbML0pbzz2K1NLvJw:1655576905205&source=lnms&tbm=isch&sa=X&ved=2ahUKEwiUtdTK0Lf4AhXyhP0HHXoRAsUQ_AUoAXoECAEQAw#imgrc=r_LLMYZ4ZU6G9M
- [6] : DevdarshanMishra.Applications of Recurrent Neural Networks (RNNs).Applications of Recurrent Neural Networks (RNNs).Retrieved from https://iq.opengenus.org/applications- of-rnn/ (Poster)
- [7]: CHOWDHURY, Gobinda G. Natural language processing. Annual review of information science and technology, 2003, vol. 37, no 1, p. 51-89. (Article)
- [8]: Johri, P., Kathait, M., Sabharwal, M., Al-Taani, A. T., &Suvanov, S. (2020). atural an ua Pro ssinistor, volution, A filiation and uturork (article)
- [9]:CHOPRA, Abhimanyu, PRASHAR, Abhinav, et SAIN, Chandresh. Natural language processing. International journal of technology enhancements and emerging engineering research, 2013, vol. 1, no 4, p. 131-134. (Article)
- [10]: Liddy, Elizabeth D. "Natural language processing." (2001). (Article)
- [11]: Expert.ai Team.(2020,June 9).Natural Language Processing Applications.Retrieved from https://www.expert.ai/blog/natural-language-processing-applications/ (Blog)
- [12]: https://en.wikipedia.org/wiki/Part-of-speech_tagging#History
- [13]:https://www.tutorialspoint.com/natural_language_processing/natural_language_processing_part_of_speech_tagging.htm
- [14]:https://www.researchgate.net/figure/Example-of-Arabic-POS-tagger_tbl2_325404114
- [15]:https://www.mygreatlearning.com/blog/pos-tagging/
- [16]: Towards_a_standard_Part_of_Speech_tagset(article)

- [17]:https://www.google.com/search?q=arabic+pos+tagging+NLP&tbm=isch&ved=2ahUKEwjzvd2j9bf4AhURLRoKHZFPCboQ2-cCegQIABAA&oq=arabic+pos+tagging+NLP&gs_lcp=CgNpbWcQAzoECCMQJzoHCCMQ6gIQJzoICAAQgAQQsQM6BAgAEAM6CwgAEIAEELEDEIMBOgUIABCABDoECAAQQzoECAAQHlC2NVj5f2DuhAFoAnAAeACAAf4NiAG2S5IBDzAuOS4zLjUtMS4xLjIuMpgBAKABAaoBC2d3cy13aXotaW1nsAEKwAEB&sclient=img&ei=wz-uYrPQKJHaaJGfpdAL&bih=637&biw=1110&rlz=1C5CHFA_enDZ806DZ806#imgrc=w11JzEJGCArB3M
- [18]: https://www.dataversity.net/a-brief-history-of-machine-learning/
- [19]: GOODFELLOW, Ian, BENGIO, Yoshua, et COURVILLE, Aaron. Deep learning.MIT press, 2016
- [20] PAS
- [21]: https://www.ibm.com/cloud/learn/deep-learning
- [22]: Aravind Pai.(2019,JUNE 10).Comprehensive Guide to Text Summarization using Deep Learning in Python.Retrieved from https://laptrinhx.com/comprehensive-guide-to-text- summarization-using-deep-learning-in-python-3082791976/ (Poster)
- [23] :Priya Pedamkar.(2020).7 Awesome Types of Deep Learning Networks - eduCBA.Retrieved from https://www.educba.com/deep-learning-networks/ (Tutorial)
- [24]:Editorial.(2020,March 15).Different types of Deep Learning models explained.Retrieved from https://roboticsbiz.com/different-types-of-deep-learning-models-explained/ (Article)
- [25]:ReLu Function in Python - JournalDev.ReLu Function in Python.Retrieved from https://www.journaldev.com/45330/relu-function-in-python (Tutorial)
- [26]:Editorial.(2020,March 15).Different types of Deep Learning models explained.Retrieved from https://roboticsbiz.com/different-types-of-deep-learning-models-explained/ (Article)
- [27]: KechitGoyal.(2019,DEC 13).Difference Between Deep Learning and Neural Networks.Retrieved from https://www.upgrad.com/blog/deep-learning-vs-neural-networks- difference-between-deep-learning-and-neural-networks/# (Blog)
- [28]: https://github.com/tensorflow/tensor2tensor
- [29]:https://www.analyticsvidhya.com/blog/2021/03/basics-of-neural-network/
- [30]:https://www.kaggle.com/swetaagrawal/formspring-data-for-cyberbullying- detection
- [31]:https://www.sciencedirect.com/science/article/abs/pii/S0360544219322339

- [32]https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challeng
- [35]:https://builtin.com/data-science/recurrent-neural-networks-and-lstm
- [36]: Diana Ramos.(2018,Oct 17):(Last modified on Jul 19, 2021).Real-Life and Business Applications of Neural Networks.Retrieved from https://www.smartsheet.com/neural- network-applications (Poster)
- [37]:IBM Cloud Education.(2020,Sept14).Recurrent Neural Networks.Retrieved from https://www.ibm.com/cloud/learn/recurrent-neural-networks (Poster)
- [38]: upGrad.(2020,MAY 6).Neural Network: Architecture, Components & Top Algorithms.Retrieved from https://www.upgrad.com/blog/neural-network-architecture- components-algorithms/ (Blog)
- [39]:Kechit Goyal.(2019,DEC 13).Difference Between Deep Learning and Neural Networks.Retrieved from https://www.upgrad.com/blog/deep-learning-vs-neural-networks- difference-between-deep-learning-and-neural-networks/# (Blog)
- [40]: Admin M2 IESC.(2018,avril 16).Artificial Intelligence, Machine Learning, and Deep Learning: Same context, Different concepts.Retrieved from https://master-iesc-angers.com/artificial-intelligence-machine-learning-and-deep-learning-same-context-different- concepts/ (Poster)
- [41]: IBM Cloud Education.(2020,Sept14).Recurrent Neural Networks.Retrieved from https://www.ibm.com/cloud/learn/recurrent-neural-networks (Poster)
- [44]: Michael Phi.(2018,Sep 24).Illustrated Guide to LSTM's and GRU's: A step by step explanation.Retrieved from https://towardsdatascience.com/illustrated-guide-to-lstms-and- gru-s-a-step-by-step-explanation-44e9eb85bf21 (Poster)
- [45]: IBM Cloud Education.(2020,Sept14).Recurrent Neural Networks.Retrieved from https://www.ibm.com/cloud/learn/recurrent-neural-networks (Poster)
- [46]: AfshineAmidi,ShervineAmidi.CS 230 - Recurrent Neural Networks Cheatsheet.Recurrent Neural Networks cheatsheet.Retrieved from https://github.com/afshinea/stanford-cs-230-deep-learning/blob/master/en/cheatsheet-recurrent-neural-networks.pdf (GitHub)
- .[47] :Aditi Mittal.(2019,Oct 12).Understanding RNN and LSTM.Retrieved from https://aditi- mittal.medium.com/understanding-rnn-and-lstm-f7cdf6dfc14e (Poster)
- [48] BROWNLEE, Jason. Long short-term memory networks with python: develop sequence prediction models with deep learning. Machine Learning Mastery, 2017. (BOOK)

- [49]:BROWNLEE, Jason. Long short-term memory networks with python: develop sequence prediction models with deep learning. Machine Learning Mastery, 2017.
- [51] https://sourceforge.net/projects/kalimat/
- [52]: POS Tagging for Arabic Text Using Bee Colony Algorithm - ScienceDirect
- [53]:https://medium.com/analytics-vidhya/vectorization-converting-text-to-numbers-20fb1302dde2
- [54]:linkedin.com/learning/deep-learning-foundations-natural-language-processing-with-tensorflow/padding-the-sequences
- [55]: BROWNLEE, Jason. Long short-term memory networks with python: develop sequence prediction models with deep learning. Machine Learning Mastery, 2017. (book)
- [56]: Palash Sharma.(2020,Oct 20).Keras Dense Layer Explained for Beginners. Retrieved from https://machinelearningknowledge.ai/keras-dense-layer-explained-for-beginners/#:~:text=The%20dense%20layer%20is%20a,used%20layer%20in%20the%20models.&text=Thus%2C%20dense%20layer%20is%20basically,the%20dimensions%20of%20the %20vector (Poster)
- [57]Python Tutorial - W3Schools.Python Tutorial Retrieved from https://www.w3schools.com/python/default.asp (Tutorial)
- [58] LUNDH, Fredrik. Python standard library. " O'Reilly Media, Inc.", 2001
- [59]io — Core tools for working with streams — Python 3.10.4 documentation
- [60] os — Diverses interfaces pour le système d'exploitation — Documentation Python 3.10.4
- [61]TensorFlow Recommenders
- [62]AboutKeras
- [63]fr.wikipedia.org/wiki/Pandas#:~:text=Pandas%20est%20une%20bibliothèque%20écrite%20pour%20le%20langage,Pandas%20est%20un%20logiciel%20libre%20sous%20licence%20BSD
- [64]it created bye: N. Ababou and A. Mazroui, Received: 27 February 2015 ; Accepted: 12 August 2015 ; Published online: 8 October 2015; Springer Science+Business Media New York 2015,A hybrid Arabic POS tagging for simple and compound morphosyntactic tags (springer.com)
- [65]:**Arabic POS Tagging: Don't Abandon Feature Engineering Just Yet** written by: Kareem Darwish;,Hamdy Mubarak, Ahmed Abdelali and Mohamed Eldesoukifrom Qatar Computing Research Institute (QCRI) HBKU, Doha,

Qatar.Arabic POS Tagging: Don't Abandon Feature Engineering Just Yet (aclanthology.org)

- [66]:POS Tagging for Arabic Text Using Bee Colony Algorithm - ScienceDirect (article)
- [67]:A Comparative Study on Various Deep Learning Techniques for Arabic NLP Syntactic Tasks written by:1/Shaima A Abushaala from Misurata University; 2/Mohammed M Elsheh from Libya Academy-Misurata it received Date: 07 December 2021; Revised Date: 09 January 2022; Accepted Date: 21 January 2022.