الجامعة الأردنية

نموذج التفويض

أنا أحمد نزيه عبد الرحمن المجدوبه، أفوض الجامعة الأردنية بتزويد نسخ من رسالتي /أطروحتي للمكتبات أو المؤسسات أو الهيئات أو الأشخاص عند طلبهم حسب التعليمات النافذة في الجامعة.

التوقيع:

التاريخ:

The University of Jordan

Authorization Form

I am Ahmad Nazeih Abdel Rahman Almajdoubah authorize the University of Jordan to supply copies of my Thesis/ Dissertation to libraries or establishments or individuals on request, according to the University of Jordan regulations.

Signature:

Date:

INVESTIGATING ENCODER-DECODER RECURRENT NEURAL NETWORK FOR DIACRITIZING ARABIC TEXT AND CORRECTING SPELLING MISTAKES

By

Ahmad Nazieh Almajdoubah

Supervisor

Dr. Gheith Abandah, Prof.

This Thesis was Submitted in Partial Fulfillment of the Requirements for the Master's Degree in Computer and Network Engineering

Faculty of Graduate Studies

The University of Jordan

August, 2021

Committee Decision

This Thesis (Investigating Encoder-Decoder Recurrent Neural Network for Diacritizing Arabic Text and Correcting Spelling Mistakes) was Successfully Defended and Approved on _____

Examination Committee	**Signature**
Dr. Gheith Abandah (Supervisor) Prof. of Computer Engineering	----------
Dr. Iyad Jafa (Member) Prof. of Computer Engineering	----------
Dr. Mohammad Abdel-Majeed (Member) Assist. Prof. of Computer Engineering	----------
Dr. Ali Al-Haj (Member) Prof. of Computer Engineering (Princess Sumaya University for Technology)	----------

Acknowledgment

I thank **Allah** for guiding me and giving me patience, the ability and the power to complete this thesis.

I wish to express my appreciation and great thanks to my supervisor **Prof. Gheith Abandah** for his support, patience, efforts and guidance to complete this thesis.

Thanks are due to my thesis **committee members** for their cooperation, comments and support.

Thanks to all **Computer Engineering Department staff** for their support and valuable comments.

Special thanks to **Prof. Amer Badarnh** who dedicated himself to enrich our knowledge in different topics.

Special great thanks to **Dr. Ashraf Suyyagh** for his help and support in my work and great thank to **Dr. Ahmad Ayen Abd Allah** who gave me the considerable energy to complete my thesis. And I wish to give all my special thanks to my special friend **Basheer Alkahteeb** for his massive support.

I owe thanks to all my friends, classmates and my colleagues who supported me all the time.

My thanks for everyone who helped and supported me during my study.

Lastly, I want to thank my parents, my wife and my children for their patience and for prays in all my study stages.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

Term	Description
AI	Artificial Intelligence
BiLSTM	Bidirectional Long-Short Term Memory
BRNNs	Bidirectional Recurrent Neural Networks
CA	Classical Arabic
CNN	Convolutional Neural Network
CSM	Correcting Spelling Mistakes
DER	Diacritic- Error-Rate
DNNs	Deep Neural Networks
FNN	Forward Neural Network
FSTs	Finite State Transducers
GNN	Gated Recurrent Neural Network
GPUs	Graphical Processing Units
GRU	Gated Recurrent Unit
LM	Language Model
LSTM	Long Short-Term Memory
ML	Machine Learning
MLP	Multilayer Perceptron
MSA	Modern Standard Arabic
NER	Named Entity Recognition
NLP	Natural Language Processing
NMT	Neural Machine Translation
NNs	Neural Networks
OOV	Out-of-Vocabulary

RNNs	Recurrent Neural Networks
SMT	Statistical Machine Translation
SVM	Support Vector Machine
TF	Tensor Flow
WER	Word-Error-Rate

Investigating Encoder-Decoder Recurrent Neural Network for Diacritizing Arabic Text and Correcting Spelling Mistakes

By

Ahmad Nazieh Almajdoubah

Supervisor

Dr. Gheith Abandah, Prof.

Abstract

Nowadays, while different dialects have affected the quality of Arabic language, the need for models that are able to correct spelling mistakes or to diacritize Arabic text is increasing. This work aims to investigate encoder-decoder recurrent neural networks' models, to solve the diacritization and spelling detection and correction problems. We performed many experiments on multiple models, and concertrate on three models (Transformer model, encoder-decoder sequence-to-sequence Model, encoder-decoder sequence-to-sequence-Attention Model) that show promising performance. The success of these models comes either from using embedding layer in both encoder and decoder portions of the model, or using the attention mechanism as a base architecture or as an attention layer of the model. The best BLEU-score result recorded in the diacritization is 64.96, and in correcting spelling mistakes is 76.12 using the *LDC ATB3* dataset. When using *Tashkeela* dataset, the BLEU-score is 61.02 in diacritization, whereas in correcting spelling mistakes it is 63.97. This work, depending on the BLEU-score results, suggests that the aforementioned models can be used to solve other sequence-to-sequence problems. We hope that the results of this work will contribute to improving the comuunication through Arabic texts by both native and non-native Arabic speakers.

Chapter One

Introduction

1. Introduction

Arabic is one of the Semitic languages, and it has more than 422 million world speakers ranking it the fifth most spoken world language. Spelling mistakes in the Arabic language occur due to user hand typing or optical character recognition (OCR) software and machine translated documents (Noaman *et al.*, 2016). On the other hand, the diacritizing of Arabic text is an important way to introduce the words in the correct meaning, which in turn makes the understanding of the context easier. In this chapter, we first introduce a background of the diacritization and correcting spelling mistakes (CSM) problems. Then we are illustrating the problem statement followed by motivations and objectives. The contribution of this thesis is introduced before we end this chapter by presenting thesis organization.

1.1 Background

Many researchers studied and analyzed the problems of CSM and diacritization, in the following lines, we will illustrate these problems to provide a background about the problems that we solved.

Abandah *et al.*, (2015) investigated the Arabic language in social networking and mobile phone communications in Jordan. They analyzed the collected sample content of Arabic language from four social network sources and mobile phones. These resources are Twitter, Facebook, blogging sites, news sites and messages from mobile phones. The results of analyzing these contents reflected the percentage of different errors in Arabic language in these samples. The results were as follows: The presence of bilingualism problem in Facebook was 14% English contributions, whereas in Twitter it was 24%. The percentage of English words in the analyzed Arabic samples was 6.4%. Arabic using English letters and numerals (Arabizi) was 13.2% of the samples. Colloquial Arabic was used in 55.4% of the samples.

Al Ameri (2015) aimed in his research to diagnose students' Arabic spelling mistakes of teacher training institutes. The researcher chose the students of the institutes for the preparation of teachers in Baghdad, for the academic year (2013-2014), and the number of twelve institutes. Seven of these institutes were chosen intentionally because they included only the fifth stage students of Arabic language department to withdraw the sample. The researcher selected 100 male and female students, who constitute 45% of the research community; 40 male and 60 female students. The researcher prepared two tools for evaluation, namely (achievement test for grammar, and the spelling Arabic text). After the researcher verified the validity of the tests, their stability and level of difficulty, he applied them to the research sample for the period from (24/3/2013) to (14/4/2014). The results obtained of his study is illustrated in Table 1.1.

Zahui *et al.*, (2017) proposed in his work to classify spelling error into three main categories, and each category contains numerous different mistakes. His classification was:

a) Phonetic transcription errors

These errors are caused when there is a poor reproduction between speech (phonetic aspect) and writing. For example, a phonetic reproduction error of (ضفضعة / ضفدعة: frog), which is caused by automatic detection system (see Table 1.2). The automatic detection system is used for learning Second Language Learners (SLLs). This system is a lexical error diagnosis system, it has the ability to interact with learner using two main types of test items: supply-type; requiring the learners to write a few words, and selection-type (Magdy *et al.*, 2007).

Table 1.1: The frequency of spelling errors that the students have signed (Al Ameri, 2015)

No.	Subject	Repetition	Percentage
1	Not distinguishing between the forms of *Tanween* in expressive cases	76	%76
2	Writing intermediate *Hamza* on *Alif*	73	%73
3	Writing *Alif Maqsora* instead of *mamdoda*	71	%71
4	Writing *Damma* instead of *Waw*	70	%70
5	Writing *Alif mamdoda* instead of *Maqsora*	70	%70
6	Failure to write *Alif* after the *Waw-group* in verbs	67	%67
7	Writing *Taa maftoha* instead of *marbota*	67	%67
8	Writing *Hamza* at the end of the word on *Waw*	64	%64
9	Writing intermediate *Hamza* single on the line	64	%64
10	Writing intermediate *Hamza* on *Waw*	64	%64
11	Writing *Thaa* instead of *Daa*	59	%59
12	Dropping *Hamzet wasel*	59	%59
13	Writing *Daa* instead of *Thaa*	53	%53
14	Writing *Damma* instead of *Waw*	53	%53
15	Writing *Hamza* at the end of the word on *Alif*	51	%51
16	Writing *Yaa* instead of *Kasra*	51	%51
17	Dropping *Alif* of *(AL)- definition* when connected to a preposition or ampersand	51	%51
18	Writing Hamza at the end of the word singly	47	%47
19	Writing *Alif* instead of *Fatha*	47	%47
20	Writing *Hamza* at the end of the word on *Yaa*	47	%47
21	Writing *Daa* or *Thaa* instead of *Thal*	45	%45
22	Writing *Sad* instead of *Seen*	43	%43
23	Write the word in two parts	43	%43
24	Provide or delay in the letters of a single word	39	%39
25	Dropping *Lam* before the *"solar letter"*	38	%38
26	Dropping *Hamza* after *(AL)- definition*	37	%37
27	Writing *Taa marbota* instead of *maftoha*	37	%37
28	Dropping *(AL)- definition* from the word	37	%37
29	Writing *Fatha* instead of *Alif mamdoda* or *Maqsora*	33	%33
30	Write the word that is not written as it is pronounced	33	%33
31	Writing intermediate *Hamza* on *Yaa*	30	%30
32	Writing *Non* instead of *Tanween* different types	29	%29
33	Writing *Hamza* at the beginning of the word	28	%28
34	Writing misplaced *Alif* after *waw*	25	%25
35	Writing *Taa* instead of *Dal*	25	%25
36	Writing *Haa* instead of *Fatha*	22	%22
37	Writing *Thal* instead of *Tha'*	22	%22
38	Writing *Haa* instead of *Alif mamdoda* or *Maqsora*	22	%22
39	Writing *Kasra* instead of *Yaa*	21	%21

Table 1.2: Example of phonetic transcription error (Zahui *et al.*, 2017)

Correct Word	ة	ع	**د**	ف	ض
Wrong Word	ة	ع	**ض**	ف	ض

Other phonetic errors caused by the speaker when he delete specific letters despite of the existing of these letters in the real words. For example, the word "الشمس" (sun) can be written "اشمس" or "مكتبة" (library) can be written "مكتبه".

b) Typographical errors

This type of mistakes is caused by scanning document with erroneous optical recognition, or keyboard typing error. These types of errors are major ones since the rate of such type of errors can reach one error per word (Hovermale *et al.*, 2009). For example, switching between the letter "خ" and the letter "ح" in the sentence "إنَّ كلامك مخرج" instead of "إنَّ كلامك محرج".

Another group of errors are classified as "Typographical errors", these groups of errors are sometimes called soft spelling mistakes or sub-standard spellings. The categories of such letters are: (1) Hamzah different shapes (ا, أ, إ, آ, ئ, ء, ؤ). (2) Haa and taa marboutah (ه, ة). (3) Alif maqsoura and yaa (ى, ي). The writer of these letters in different words might miss-spell them, which causes many mistakes in the words (Attia *et al.*, 2016).

c) Morphosynthetic errors

These errors happen when there is miss understanding of the gender and the number that suits the word. For example, using the word "هذه" instead of "هاتان" in the sentence " هذه البنتان تلعبان بالكرة" whereas the correct sentence is "هاتان البنتان تلعبان بالكرة" (Zahui *et al.*, 2017).

There are a few letters in Arabic that are regularly incorrectly spelled utilizing variations, moreover, some specialists think that it is helpful to totally make these variations vague (standardized) (Azmi *et al.*, 2019). To handle these types of errors, machine learning (ML) approaches outperformed other approaches, and investigating the different approaches may lead to improve the accuracy of correcting spelling mistakes (Shaalan *et al.*, 2010).

Diacritics, on the other hand, provide the Arabic characters the correct way to be pronounced. This will lead to determine the whole word pronunciation; and as a result the word meaning. Any incorrect changing of the letter diacritics will cause an error, or at least an ambiguity in the meaning. From this point, to make a correct diacritization for un-diacritized text, it is important to improve the existing systems by testing more alternatives, or searching for different innovative solutions. The investigation of ML approaches that are used for diacritization is an important researching area, since the different approaches could be improved to achieve faster and more accurate systems (Abandah *et al.*, 2015).

1.2 Problem Statement

The process of CSM is an important element for delivering a correct text in different languages. In Arabic language, the problem of spelling mistakes has a large impact on using the language by native and non-native Arabic users, as well as its impact on the spread of this language in various means of communication. Similarly to CSM, the process of diacritization has the same effect on Arabic language.

In this research, we will investigate the existing methods that deal with Arabic text diacritization, which will be the base for next steps to solve the CSM. Bearing in mind that, the problem with short sequences has been solved in different approaches, whereas the one with long sequences is still under research. In this work, the problem with long

sequences will be investigated. The existing approaches of Recurrent Neural Networks (RNNs) with Encoder/Decoder (E/D) will be surveyed and evaluated, particularly approaches with attention that perform well with long sequences.

The proposed ML solution using RNN with E/D will be tested on two datasets that are used in related research. These two sets are *LDC-ATB3* and *Tashkeela*.

Arabic diacritization using RNNs is now well researched with good accuracy when the input and output lengths are the same, we will try to tackle Arabic diacritization using E/D networks. The success in the previous step using best-found approaches will be applied to solve CSM.

The purpose of this work is to present the best models, after testing the existing models that are used to solve sequence-to-sequence problems, which achieved the best results in solving the diacritization and spelling mistakes problems.

1.3 Motivations

The exponential use of the computer and means of communication (social networks, instant messaging and e-mail), makes the automatic spelling correction a key theme in natural languages processing (NLP). The delivery of a correct meaning is an important role that word plays. If the meaning transmitted wrongly by mistaken word or sometimes by undiacritized word, misinterpretation or misunderstanding of the meaning will be reflected, especially for the ones who are recent with Arabic language, such as children or new Arabic learners (Zahui *et al.*, 2017). By offering the systems that can check the Arabic words text, the non-native Arabic speakers or the new users of this language, will have the confidence to use the language easily. On the other hand, the existence of systems that are capable of diacritizing Arabic text will make the understanding of the

context easier. Consequently, the content of Arabic language will spread continuously on the internet.

The learning of Arabic language is important for many reasons. Firstly, it is used in different countries all over the world. By making the learning mechanism of the language as easy as possible, the spreading of Arabic language between different states will rapidly increase. The tools that help to speak, write, and listen to the language correctly, will be an important factor to enhance the number of Arabic language users. Secondly, because the Arabic language is used in different fields in native Arabic countries, such as Learning, News, Official treatments, regional conferences and cultural exchanges. It is important for native speakers to get over their lack of speaking or writing Arabic words correctly. Therefore, by providing the well-revised content using an accurate tool that guarantees the correct text and correct diacritization of words, the content of Arabic language will be increased, and the communication using this language will be wide-spread in many different regions and between different nations (Wali and Gargouri, 2015).

1.4 Objectives

The main objectives of this thesis project are:

1- To survey the existing RNNs, particularly Encoder/Decoder solutions that have features such as attention to deal with long sequences.
2- Experiment with alternative solutions to select the best ones that can solve adding diacritics problem, when the input and output sequences are not of the same length.
3- Using the results of Step 2 to solve the correcting spelling mistakes problem.

1.5 Contribution

Since there is a lack of different types of Arabic language corpus, as well as the shortage of ML programs that deal with the Arabic language compared to other languages, in

addition to the fact that the Arabic language is considered as one of the most difficult languages in the world because it is considered a language rich in terms, the contribution of this study was to solve the problems of spelling mistakes and diacritizing Arabic text. Specifically, the contributions are as follows:

- The first contribution is presenting ML models that has the ability to correct the spelling mistakes with efficient results, which in turn help the users of Arabic language to communicate easily using this language.
- The second contribution is producing models that can diacritize Arabic text, and as a result, increasing the understanding of different Arabic texts, which in turn making Arabic language effortless to use.
- The third contribution is to provide a comparison between different type of models that can be used to solve CSM or diacritization problems or any other sequence-to-sequence problems.
- The last contribution is to introduce models that have the ability to solve both CSM and diacritization problems at the same time.

1.6 Thesis Organization

We organized our thesis in to seven chapters, the outline of each chapter is as follows:

This chapter introduces the background of the problems discussed, then the problem statement and motivations are presented to illustrate the problem and the importance of solving it, after that the objectives are explained and finally the contribution of this work is illustrated.

Chapter two presents a review of the extent literature that relevance to the subject of this thesis including overview for RNN in NLP, diacritizing Arabic text and CSM. The different approaches that were used during multiple periods were presented. The

chronological development in various models is introduced to illustrate the different previous work through the years.

Chapter Three investigates the concept of deep learning for natural language processing and the different machine learning general methods and concepts. The neural networks, recurrent neural network, encoder-decoder framework and sequence-to-sequence architecture are illustrated.

Attention in natural language processing is the main concept that is discussed in chapter four. The meaning of attention is explained, the idea of attention is illustrated and attention mechanisms were debated. The end suction of this chapter explains the "Transformer Model" in details.

Chapter five presents the methodology used for both diacritizing Arabic text and CSM. Experimental environments are illustrated, datasets are explained and training methodology is discussed and analyzed.

Chapter six reveals the results of each experiment for both diacritizing Arabic text and CSM. The results are discussed and compared to each other to receive a general view of the values gained and to determine the best models. We also compared our results with related work.

Chapter seven is the chapter where the conclusion of the different executed experiments is discussed. The comparison between both CSM and diacritization is illustrated to conclude the relationships between the multiple implemented experiments. The completion of the different ideas which manipulated in this work is mentioned as the future work.

Chapter Two
Related Work

2. Related Work

In our thesis, different approaches have been discussed. According to each approach, many different related works were accomplished by many researchers. In the following lines, every approach will be mentioned with its related works.

2.1 RRNs in NLP

Early NLP structures had been primarily, based totally in most cases on manually written rules, which is exhausting and time-consuming and do not have the ability to cover numerous linguistic phenomena. A statistical Language Model (LM) was introduced in the 1980s to overcome the previous method. In this model, the product of conditional probability of the following word, taking into consideration its predecessors, is divided to represent a word sequence probability, which is called context or a history of context. Despite of that the LM model has pros, it moreover has cons, that is; this model is barely has the ability to learn the extremely many parameters. So, the need of approximation model is immerged to beat this problem. This introduced the creation of N-gram model as approximation method which were used in many NLP problems by different researchers.

Although LM has been used to solve many problems and gave a promised results at that time, it has a disadvantage. The disadvantage represented as that the number of probabilities which the N-gram model creates is much more than the actual ones that appear in the training corpus; such that it creates the whole probabilities from 0 to N-grams, which is not reflecting the actual situation. To solve this problem, techniques which are called “Smoothing Techniques” were used. The aim of using these techniques is to determine the probability of events that do not appeared in the corpus used in the training phase, and to decrease the probability of events that appeared in it. But the dilemma did not end there, another problem raised when using larger corpora for

universal language models, which is considered as a dimensionality problem. To grasp the idea of this problem, for instance, if we want to model an N-gram model which has a corpus of 20,000 vocabulary size, there will be a number of $20{,}000^{N-1}$ free parameters.

The solution for this problem was by introducing the NNs and after that the RNNs (Elman, 1990). These networks afforded language modeling in continuous space, and moreover have the ability to learn features automatically and in continuous representation (Jing *et al.*, 2019).

Because RNNs have some limitations in dealing with problems especially the ones that have long-term dependencies, gating mechanisms have been developed to reduce some restrictions of the basic RNN, which produced two prevalent RNN types: Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) and gated recurrent unit (GRU) (Cho *et al.*, 2014a).

The RNNs, have been used in many diverse types of ML, and achieved a very promising results in several fields. The great results gained of using RRNs for solving various ML problems encouraged the researchers to use it to solve different NLP problems. There are different types of NLP problems which were researched by vast variety of researchers, such as: Automatic Summarization, Machine Translation, Discourse Analysis, Morphological Segmentation, Sentiment Analysis, Part Of Speech Tagging, Text Completion, Question Answering, CSM, Diacritization etc.

To discuss the term of RNN in NLP, we present the different techniques that have been used to handle the different types of NLP problems. The following lines will present some research which suggested different general models used to handle diverse NLP problems.

2.1.1 Character-Aware Models

The first type of techniques will be presented is “Character-Aware Techniques”. The approaches that are used in this type are: character-level approach, word level approach or a combination of these two approaches; depending on the problem that we want to deal with.

Mikolov *et al.*, (2012) found that in the case of Out-Of-Vocabulary (OOV) word problem, it is beneficial to use the character-level model which has the ability to improve the layout of unknown or uncommon words, because the structural similarities between words are revealed by character features. On the other hand, the character-level has the ability to minimize the training parameters because of the small Softmax layers with the output of the character-level.

Nevertheless, the character level model revealed, that time, that it is worse than word-level model when used with different NLP problems, and it is challenging to train highly accurate character-level model. The reason behind this is that the character-level model to have the ability to predict the next word correctly, it must take into consideration a long history for predicting. Depending on the previously mentioned facts, most researchers headed to combine between both the character-level and word-level approaches to receive the advantages of both two approaches. As a result, many solutions are proposed to overcome the different problems by using a combination of these two approaches.

Kim *et al.*, (2015) used the character-level along with word-level in his proposed model. The idea in their model was that they performed to use the character-level features after organizing them as a base for the word-level model. On other words, they extracted the features from the character-level model to generate a word by word using Convolutional Neural Network (CNN), and for receiving these extracted features, they used LSTM to receive words in the word-level. With this technique, they outperform other previous

techniques when they use many different natural languages (Russian, Czech, German, French, Spanish and Arabic).

Miyamoto and Cho (2016) suggested RNN model with LSTM units that uses both character-level and word-level techniques. Their model was used to predict the next word depending on all preceding words. In the proposed model, the inputs of the character-level approach are transformed to a vector representations of words by using bidirectional LSTM (BiLSTM). And the inputs of the word-level approach are converted into high-dimensional space by using a word lookup table. The resulted vector representations of words are utilized in the LSTM model that predicts the next word depending on all the given preceding words.

Hwang and Sung (2017) used a hierarchal RNN model to solve the problem of character-level Neural networks (NNs). In their proposed model, the RNN architecture was comprised of various models which have different time scales. And by using this model they have achieved results that outperformed previous models even though a 30% number of parameters are reduced.

Verwimp *et al.*, (2017) suggested a model that combines both character and word level models which called character-word LSTM-RNN model. This model produces the concatenated vectors that produced from the character-level and word-level approaches. The character-aware model is directly used the character-level model as a feature extractor for the word-level model. As a result, the main model will have a rich character-word information that used for prediction. The results achieved in this model were a value of 2.77% relative improvement on English language with the same number of parameters when compared to a baseline model, and with a value of 4.57% on Dutch language.

Gumilang and Purwarianti (2018) tested a character-level features and word-level features on different types of topologies of RNNs (Bidirectional RNN (BRNN), LSTM

and GRU) compared with topology of CNNs. The researchers used two datasets that were gained from social media: one in English language and the other in Indonesian language. The different topologies were used to test these two datasets with character-level features and word-level features for text classification. The results presented, after applying different experiments, revealed that in the problem of text classification the word-level model outperforms character-level model in small-sized datasets, whereas the character-level model achieved more promising results than word-level model when the dataset is containing millions of data.

Ataman *et al.*, (2019) presented a model which is more efficient in computation for the solution of character-level Neural Machine Translation (NMT). The presented model is a hierarchal decoding architecture used as a machine translation task from English language to other five languages. The translations generated from this model are posteriorly produced at the level of characters and words. The hierarchal NMT model is consists of character embedding layer, character-level BiRNN, Word-level RNN, attention mechanism and character-level RNN. The results achieved from applying this model, that the proposed model can score more accurate results in translation of the sub word-level NMT model with using fewer parameters. The proposed model moreover revealed better capacity in learning longer-distance grammatical and contextual dependencies than the standard character-level NMT model.

Bansal and Lobiyal (2020) introduced a method that combines between both character and word attention information which made a notable improvement in the NMT model. The main idea in their proposed method is to use two attention mechanisms; the first attention mechanism was used to take the character-level attention input as a sequence of character. The second attention used the word-level to capture the words generated from the first character-level attention. The combining of these two attention mechanisms gave

the encoder, which is the first part of the model, the ability to simultaneously encode the information. On the other part, which is the decoder, the information is decoded depending on word-level. The results gained proved that the suggested model outperformed the English-Hindi baseline system which was presented as a language pair system.

2.1.2 Encoder-Decoder Models

The idea of using encoder-decoder model in machine learning was proposed on machine translation by (Neco and Forcada, 1997). After six years, (Bengio *et al.*, 2003) suggested a language model which depends on NNs. This language model enhanced the data sparsity problem of the traditional statistical machine translation (SMT) models. The effort that made of this work created a fertile land for other researchers to use the NNs in different machine translation approaches.

Kalchbrenner and Blunsom (2013) suggested a new approach for machine translation which is considered as the birth of NMT because it was the first approach that using deep learning NNs to map between natural languages. The proposed structure was an end-to-end encoder-decoder approach. In their model, a recurrent continuous translation model is suggested to present a purely sentence-level translation model. They proposed different models (RCTMs) that had the ability to improve the translation and to decrease perplexities of the models when compared to reference translations. The presented models were distinguished by their flexibility because of their sensitivity of the continuous representations to conditioning information.

With the time, the problem of dealing with long sequences arises. When using RNN models, the problem of vanishing gradients was one of the most disadvantages of using such models, which made the performance of these models unpromising (Pascanu *et al.*, 2013). There were some solutions proposed to overcome this problem, one of them is

LSTM, which were used in multiple approaches to solve different problems and achieved very promising results. In the year of 2014, GRU is another promising solution proposed by (Cho *et al.*, 2014a) which was used in various kinds of NLP problems and reflected a promising result too, but with more efficiency in consuming resources compared to same problems when it was used instead of LSTM models in some applications. The following lines will represent some of these models and their results.

Sutskever *et al.*, (2014) alongside with Cho *et al.*, (2014b) improved an approach called sequence-to-sequence model which uses RNN for both encoder and decoder parts of the model, and they introduced the LSTM models in the NMT Approaches. By using this idea, the problem of exploding gradients is solved Therefore that the long-distance dependencies is captured by the model, and the sentences with long sequences is processed with better results.

Zhou *et al.*, (2016) suggested a new type of linear connections to represent NMT. The new type was a fast forward connection depending on LSTM networks and an interlaced bidirectional approach for stacking the multiple LSTM layers. The proposed model achieved a promising result using the BLEU-score with a value of 36.3 without using attention mechanism and a value of BLEU-score equals to 37.7 with attention presented in the model on the WMT'14 English-to-French corpus. And with a value of 40.4 for the BLEU-score they outperform other models that time after making model ensample and after handling unknown words. The proposed model moreover had the ability to deal with WMT'14 English-to-German task.

Britz *et al.*, (2017) presented a massive exploration of NMT architectures. In the work presented by them, they solved the problem of the previous architectures that were taking long time to complete the training process, which in turn is costly by consuming resources. The solutions proposed are to present the first large-scale analysis of the hyper-

parameters of the NMT architecture. Moreover, they released an open-source framework for NMT which granted the researchers the ability to apply the experiments with new techniques and regain state of the art results.

Platanios *et al.*, (2018) suggested a simple improvement to the existing NMT models, but this simple touch presented a great result in these models. The proposed approach adopted the standard NMT system architecture without any change, and the idea was to add a new component to the standard architecture. The new component is called contextual parameter generator (CPG). The CPG role was to generate the parameters of the system (for instance, weights in the NNs). This component has the ability to accept source language embeddings and target language embeddings as input, and then produce the parameters for both the encoder and the decoder. The results of applying this idea revealed that the system had the ability to use monolingual data for training step and the system moreover implemented zero-shot translation.[1] The results moreover presented an exceeding results in performance for both of IWSLT-15 and IWSLT-17 corpuses. The last result was that the relationships between languages was revealed by learning the language embeddings.

Hammerton (2003) was the first who used LSTMs in the field of Named Entity Recognition (NER). He proposed a model which was the superior in that time, despite of the small architecture of the model's network because of the lack of affordable computation power and time. Adding to that, the models for sophisticated numeric vector for words were not presented. The results achieved from this model were better than base model in German language and exceeded, with slight value, the English language base model.

[1] Zero-shot translation means that the model has the ability to translate between language pairs even though there is no explicit training data.

After a long period, in the year of 2016, Kuru *et al.*, (2016) suggested a model called CharNER, which is for language independent NER a character-level tagger. In this model, to extract character-level representations LSTMs were used. The model is considering a sentence as a series sequence of characters. Instead of each word, the model outputs a tag distribution for each character. By using the character-level tags, the word-level tags are gained. The results observed from this model referred that, to use characters as the initial representation is more effective than using words.

Yang *et al.*, (2016) proposed a model that used bidirectional GRU as a deep hierarchal RNN for sequence tagging. The suggested model solved both multi-task joint training and cross-lingual in a unified manner. The GRU units are employed in both word and character levels to encode context and morphology information, and the resulted tags are predicted by a conditional random field layer. Their model was language independent, task independent and feature engineering free. The model achieved a dominant result compared to other models on multiple benchmark tasks and on different languages. The tasks are including NER, chunking and part of speech tagging.

Tran *et al.*, (2017) presented a model which is consisting of stack residual LSTM and biased trainable decoding which is connected to a neural NER model. In the proposed model, the word features are obtained from character-level RNN and word embeddings. The work in the suggested model achieved a state-of-the-art results that time in both English and Spanish languages using the CoNLL 2003 Shared Task NER dataset.

Akbik *et al.*, (2018) proposed a model which gives a single word different embeddings according to its contextual use. They used a pre-trained BiLSTM character language model to perform NER. For evaluation purposes, the researchers applied a comparative evaluation with other embeddings to find that their embeddings are very useful in

downstream tasks. This model achieved F1-score results outperform other previously models in the same task with both German and English NER.

2.1.3 Attention Models

Attention was to begin with presented in NLP for machine translation assignments by Bahdanau *et al.* (2014). In any case, the thought of impressions had as of now been proposed in computer vision by Larochelle and Hinton (2010), taking after the perception that natural retinas focus on significant parts of the optic cluster, whereas determination falls off quickly with whimsy. The term visual attention got to be particularly prevalent after Mnih *et al.* (2014) essentially outflanked the state of the craftsmanship in a few pictures classification errands as well as in energetic visual control issues such as question following due to an engineering that may adaptively select and after that handle a grouping of locales or areas at tall determination and utilize a continuously lower determination for encourage pixels. As we mentioned before, Bahdanau *et al.* (2014) was the first who used attention mechanism in NLP which is known as Bahdanau attention or additive attention. The usage of attention alleviates the encoder from the load of receiving to encode all data in the input sentence right into a fixed-length vector. By using this new approach, the data may be unfolded at some stage in the series of annotations, which may be selectively retrieved via way of means of the decoder accordingly. More details about additive attention will be presented in Section 4.2.1.

Luong *et al.*, (2015) presented new attention model techniques; global attention and local attention. In the global attention, the model takes into consideration all words in the input sentences, whereas in local attention the model just concentrates on a subset of the input sentences at a time. The two proposed attention mechanisms were applied on the problem of translation from English language to German language and vice versa. The local attention achieved extra 5.0 BLEU-score over the other models that are using techniques

other than attention. Moreover, the proposed model, which used different previously mentioned suggested techniques, achieved a BLEU-score of 25.9 when applying WMT'15 English to German translation task. When comparing this result with the best results gained by other systems that time, an improvement of 1.0 BLEU-score is recorded. The techniques of attention mechanisms proposed by Bahdanau *et al.*, (2014) and Luong *et al.*, (2015) were as the seeds for other researchers to improve different models in diverse fields. In the field of NLP, the use of attention mechanisms enhanced vast various models.

Tu *et al.*, (2016) proposed a converged-based NMT model. In NMT, the use of attention mechanisms enhanced the model by jointly learning to align and translate. In NMT-attention models, the problem of ignoring the previous alignment information is immerged. Converged-based NMT model was proposed to overcome this problem. The idea was to create a vector and preserve it to keep track of all attention history. After that, the attention history of this vector is fed to the attention model to increase the adjusting of attention model, and as a result improving the NMT model translation. The results revealed that both the alignment quality and translation quality are significantly enhanced, when compared to the standard attention based NMT.

The idea of attention mechanism was used in different models to enhance the results gained from the various models in divers NLP applications. A turning point in using attention technique was proposed in 2017 in a paper with title "Attention Is All You Need", which was the first idea of using attention as a main mechanism in the model to solve problems.

Vaswani *et al.*, (2017) proposed a new attention technique in the general form of NMT model as an encoder-decoder model. The new suggested architecture moreover known as "Transformer", referring to the new idea that depends on attention mechanism to

transform the input language to other output language. The researchers used WMT 2014 English-to-French corpus and received a BLEU-score of 41.8 after training a single model on eight Graphical Processing Units (GPUs) for three days and a half. Another translation task was applied using the new suggested technique by applying WMT 2014 English-to-German translation task, and the results outperformed the best announced results that time by 2.0 BLEU-score points. The model moreover revealed its superiority in quality by applying more parallelism while manipulating data, and reduced the execution time for training compared to other previous models. More details about the Transformer model are illustrated in Section 4.3.

Tang *et al.*, (2018) analyzed encoder-decoder attention mechanisms in NMT models with respect to word sense disambiguation. In machine translation problem, to recognizing the ambiguous words is one of the hardest obstacles that faces the model. Most models deal with such problem by adding a tag that refers to this unknown words. Tang *et al.*, (2018) assumed that there is more attention is paid by the model to context when ambiguous words are translated. The distribution patterns for attention moreover were explored after translating the vague words. The result of analyzing these points reveals that the concentration of attention distribution occurred by attention mechanism is applied on the ambiguous word itself ignoring the context that the word is a part of, which as a result affects the model translation performance. Adding to the previous, the NMT model does not rely on attention mechanism to recognize or deal with unknown words. The researchers suggested that, to solve this problem the NMT model could use the encoder hidden state by encoding the contextual information that are belonging to the word sense disambiguation problem. In the attention mechanism in the Transformer model after analyzing multiple experiments, the first numbers of layers have the potential to learn

aligning source and target tokens. Whereas the last numbers of layers have the ability to learn to receive the features from the related-unaligned context tokens.

Medina and Kalita (2018) proposed a parallel attention mechanism in NMT. The proposed mechanism is to run stacked encoding branches from encoder-decoder attention focused hierarchy in parallel. The parallel mechanism in turn, will decrease many sequential operations and consequently the training time will be decreased. The suggested parallel attention mechanism was applied on the Transformer model, which replaced the sequential attention mechanism with parallel one. This replacement decreases the time spent in the training process and moreover increased the BLEU-score recorded after executing this model. The translation process was applied on English-to-French and English-to-German translation tasks, and the results revealed state of the art results that time.

Guo *et al.*, (2019) tried to improve the results achieved when using the Transformer model; they suggested Star-Transformer model which is an improvement for Transformer model. The new proposed model took a star-shaped topology instead of fully-connected structure. The star-shaped topology reduced the complexity of the system from quadratic to linear one. This new topology preserved the capacity to capture both long-range dependency and local composition. The experiments were applied on different 22 datasets, and the Star-Transformer achieved state-of-the-art results, when applied on humbly-size datasets compared to the standard Transformer model. Moreover, the use of the new proposed topology decreased the training time on the different used datasets compared to Transformer model training time.

Zhang *et al.*, (2020) depended on the observation, that there is a similarity between the context vectors which generated by an attention mechanism to predict the target words when the target words are different, they suggest an innovative GRU-attention model for

NMT which they called G*Att*. The proposed method used GRU to update the input representations with the former decoder state, the role of G*Att* is to allow translation-sensitive input representations which then contribute to distinctive context vectors. The researchers moreover suggested an alternative of G*Att* by exchanging the input order of the source representations with the previous decoder state to the GRU. The experiments were executed using WMT14 English-German, WMT17 English-German, and NIST Chinese-English translation tasks. The results revealed that the proposed GAtt-NMT model outperformed the vanilla attention-based NMT with a significant improvement. The researchers moreover provided that the analyses on the context vectors and attention weights illustrated the capability of the G*Att* in improving the representations' discriminating capacity and handling the overtranslation problem.[2]

Shi *et al.*, (2021) improved NMT by using sentence alignment learning, such that the sufficiency in machine translation is enhanced by conveying the semantic knowledge from bilingual sentence alignment learning. In more details, a discriminator which learns to evaluate sentence aligning score over translation candidates was designed. This discriminator is containing gated self-attention based sentence encoders. For better capturing of lexical evidence from bilingual sentence pairs, the discriminator is trained with an n-pair loss. Moreover, the researchers proposed a sentence alignment-aware decoding method and an adversarial training approach for NMT which transfer the learned semantic knowledge of the discriminator to NMT models. The translation tasks applied were English-to-German, Uyghur-to-Chinese, and Chinese-to-English. The scored results illustrated the superiority of this model in the three mentioned translation tasks when compared to the baseline NMT models.

[2] Overtranslation problem referes to repeatedly predicting the same target words.

Dou *et al.*, (2021) suggested approach which applies attention forcing in machine translation. The proposed approach has the ability to guide a sequence-to-sequence model with referenced attention and generated output history. Consequently, the model become able to overcome its mistakes without any need for classifier or schedule. Because of multi-modal and discrete nature of the NMT output space, the researchers added a selection mechanism to vanilla attention forcing, which for each pair of training data, selects the appropriate training approach. The experiment results clarified that the use of attention forcing improved the overall translation quality and moreover the diversity of the translations.

2.2 Diacritization

The interest in Arabic diacritization methods has recently risend and is researched ursing different approaches of ML. Compared to other NLP topics, we can notice some lack of research in this problem. The reason for the rareness of studies, is that, nowadays the MSA is used without diacritics and the diacritics are used in a small area of fields. Another reason is that the rarity of the well-prepared datasets that belong to Arabic diacritics and Arabic language in general, was another obstacle for researchers to deal with such problem. In the next lines, the related work that belongs to Arabic diacritics will be explained.

The interest in diacritics started from the concerning in the language vowels. El-Sadany and Hashish (1988) suggested a rule-based method that achieved the vowelization by using morphological analyzer. In the year of 2004, a rule-based grapheme to sound transformation architecture was proposed by (El-Imam, 2004).

The rule-based methods have many disadvantages; one of them is that for any usable language, these languages have the property of changing nature, and as a result, new rules must be introduced. Another important obstacle for such languages, especially for Arabic

language, it is not possible to keep the rules up-to-date and applying them on other Arabic dialects.

Gal (2002) used a bigram Hidden Markov Model to apply the diacritization approach. We can say that in this approach a white-space restricted word-based model is used to restore just a subset of all diacritics or vowels.

Vergyri and Kirchhoff (2004) restored diacritics in Arabic conversation by joining between contextual and morphological data with a sonic signal. This method treated the diacritization problem as an unsupervised tagging case. Each word in the unsupervised tagging is tagged as one of different suggested forms introduced by the Buckwalter's morphological analyzer (Buckwalter, 2002). The tag sequences is learned using the expectation maximization algorithm.

Because of the drawbacks of rule-based methods, many different researchers used another new method to manipulate the problem of diacritization. A hierarchical top-down example-based architecture was suggested by (Emam and Fisher, 2005). The approach proposed initially search the training data hierarchically to look for a matching between sentences. If there is a matched sentence, the whole statement is used. If there is no matching, by using a hierarchal procedure, the search for matching phrases is done, then the search for words to determine diacritics. If the model could not find any match of the previous top-down steps, every word in the statement is diacritized using character n-gram models.

After few years, (Nelken and Shieber, 2005) suggested a weighted finite state machine-based algorithm. In their approach, larger morphological units, characters, and words, were used. Comparing with past studies, this method is considered more advanced in terms of integrating different information sources and introducing the problem as a search task in a consolidated framework. Adding that the high scores of accuracy results that

been scored by this approach when compared to other previous models. Necessary to add that the algorithm enables a character-based generative diacritization scheme just for words that do not appear in the data used for training. In their paper, the diacritics (*shaddah* and *sokoon*) not mentioned as predicted diacritics using their method.

Although the strategies proposed for diacritic restoration have been developed and made strides over time, they are still constrained in terms of scope and precision.

Zitouni *et al.*, (2006) suggested in their method to receive back the foremost comprehensive list of the diacritics that are utilized in any Arabic content. Their proposed strategy varies from the past approaches within the way the diacritization problem is defined and since numerous data sources are integrated. The problem of diacritic retrieving is presented as a sequence classification problem, such that for any given sequence of characters, the aim is to determine the suitable diacritic for each character. Their approach is depended on "Maximum Entropy" technique (Berger *et al.*, 1996). In their model, they assumed that sequence classification is one of the problems that can be solved using the maximum entropy techniques. This can be done by transforming the activation scores into probabilities by using a softmax function and using a search algorithm called "The standard dynamic programming", which known as "Viterbi search". They presented an Arabic diacritic restoration statistical model which is based on the maximum entropy technique. Their model had the ability to join different sources of information which are: segment-based features, lexical features, and part-of-speech features. With a view to mimic real-world applications, segment-based and part-of-speech features are both have been generated by distinct statistical systems. The results achieved in that time revealed that, the joining between aforementioned sources of information is marvelously achieve a promising result and great scores.

A new idea of NLP partial diacritization is presented by (Diab *et al.*, 2007). In their work, they defined different diacritization schemes, which were suggested depending on the naturally locations of distributed diacritics in Arabic text. The experiment was to examine the differences in applying full-diacritization versus partial-diacritization of a phrase-based SMT context. The results revealed that the full diacritization is not beneficial for SMT, and for partial diacritization there was no significant changing in the output context of the SMT.

Habash and Rambow (2007) called their new system "The MADA-D System". They tried to improve their previous model "Morphological Analysis and Disambiguation of Arabic" (MADA) by training a group of taggers for individual linguistic features that are parts of the full morphological tag. As they mentioned, the number of morphological tags can differ relying on the way that they use to count, and it is in the range between 2,000 and 20,000 tags. In their work, they follow what (Hajic *et al.*, 2005) suggested about the features that must be taken into consideration such as: nunation, mood and case. The reason behind that these features are very important for determining the correct diacritics. In the second step, for their tool of machine learning, they used the support vector machine (SVM) tool of (Giménez and Marquez, 2004) which is faster and slightly less accuracy of the abandoned tool. The last step was to present a particular module for settling residual uncertainty after the fundamental vagueness is done. The model they proposed, as classifier model, was trained on the same training set that was used by (Zitouni *et al.*, 2006); which was *ATB3-tarin* subset with 288,000 words. For developing and testing, a separated set *ATB3-devtest* with 52,000 words was used. The results of their work revealed that a diacritizer which employs a lexical resource can beat a profoundly optimized diacritization framework that draws on a huge number of features. The

disadvantage of their system was the large influence of the system with existing of a number of unknown words.

In the year of 2009, the hybrid systems idea emerged to overcome the drawbacks of the previous models. Rashwan *et al.*, (2009) proposed a hybrid system for automatic Arabic diacritization. The system is a two-layer stochastic approach to automatically diacritize Arabic text. The first layer in the model determines the most expected diacritics for the context by selecting a series of a full-form Arabic word diacritizations with a top marginal probability by using m-gram probability estimation and a wide A* lattice search. The second layer operates when OOV full-form happens. Each Arabic word is factorized by the second layer into probable morphological type (suffix, prefix, pattern, and root). Then the use of, lattice search and m-gram probability estimation, takes a rule to select from the affordable diacritization the most appropriated sequence of diacritization. Necessary to add that the dataset that they used is not popular in the era. The results reported from their work were: 2.1% WER in the morphological errors and 8.1% WER in the syntactical errors. They concluded that the master plan to grasp usable results when solving full-Arabic diacritization problem is to combine between linguistic factorization and statistical methods.

Another hybrid approach was proposed in the same year by (Shaalan *et al.*, 2009). In order to learn their model, the different right use of each diacritic, a large fully Arabic diacritized corpus and an Arabic lexicon was used. The architecture of the hybrid model depended on SVM-statistical prioritized techniques, bigram and lexicon retrieval. The model moreover dialed with the diacritics of the last letter of the word, moreover called “case-endings”, as an isolated post processing task utilizing syntactic data. The idea of the model was to apply all the three techniques: bigram, SVM-statistical technique and lexicon retrieval, on an Arabic sentence without diacritics. Every technique will process

the sentence and suggest the diacritics for every word. The chosen of the word diacritics depends on the agreement between those techniques for at least two of them. If the agreement is not achieved, the diacritics is applied using lexicon retrieval technique as a first priority, then using bigram for the second priority technique and lastly using SVM-statistical technique as the final priority. A word without any diacritic is the output of the system, if there are no results gained from any previously mentioned techniques. The results of their work were better than other systems in the same problem that time. The results for both, diacritic error-rate (DER) and word error-rate (WER), metrics precedes the MADA-D system which was proposed previously by (Habash and Rambow, 2007). And the adjustments of case-ending algorithm have improved the performance.

Rashwan *et al.*, (2010) used a long-horizon n-gram probability system and A* lattice search to determine the top marginal probability of a context of Arabic word diacritizations that is chosen with the appropriate diacritics using the first mode that they proposed. The second mode is operated when the words are OOV. In this mode, the OOV words are factorized to all its probable morphological constituents, and then back to using the first mentioned technique to determine the most probably diacritics. Whereas the second mode accomplishes a distant superior scope of the exceedingly derivative and inflective Arabic language, the first mode is faster to memorize, i.e., yields way better disambiguation comes about for the same estimate of training corpora, particularly case-ending diacritics. The hybrid mode of the previously mentioned modes is proposed and tested by (Rashwan *et al.*, 2010) and the results were better than other models of both (Habash and Rambow, 2007) and (Zitouni *et al.*, 2006). For morphological diacritization, the results in WER-metric were: 3.1% for the proposed hybrid model, 5.5% for (Habash and Rambow, 2007) and 7.9% for (Zitouni *et al.*, 2006). And the results for all-text diacritization including the case-endings were: 12.5%, 14.9% and 18% respectively.

In the year of 2012, (Bahanshal and Al-Khalifa, 2012) introduced an approach to evaluate Arabic diacritization systems. They used three available and free-to-use diacritization systems; KACST Arabic Diacritizer (KAD) (Alghamdi *et al.*, 2010), Mishkal Diacritizer and Sakhr Diacritizer (Bahanshal and Al-Khalifa, 2012). They assumed a simple formula which taking the accuracy as a ratio between correct diacritics gained by the tested system for a specific text, to the actual amount of the diacritics in the sample text which was selected from the Holy Quran and some Arabic poems. The results revealed that KAD is outperformed other tested methods with accuracy rate of 94.5%. They have mentioned that KAD has one drawback, which is the model revealed a tendency to provide a "*socoon*" diacritic to long vowels. The system mentioned previously have a weakness point; the testing step was applied on a small amount of test data which may not reflect the real performance of the tested systems.

Said *et al.*, (2013) proposed a hybrid approach for Arabic diacritization. The approach is designed as a pipeline of different components, each of these components has the ability to determine particular side of the vowel restoration problem. The diacritization of the text passes through three stages. The first stage is preprocessing, the second stage is that for each word, valid morphological structures are generated. From these structures, a network of analyses is built by joining the analyses of both rule-based and statistical morphological analyzers. The last step is the disambiguation, which is used to introduce the text with its diacritics as well as case-endings diacritics. The aforementioned system was combining data-driven techniques with rule-based ones to achieve the solving of diacritization problem. The results were 4.4% WER for the morphological diacritization ignoring case-endings, and when case-endings were included in the full-diacritization problem the WER was 11.4%. These results outperformed other precedent systems' results with an absolute 1.1 decreasing in WER.

In the year of 2014, (Pasha *et al.*, 2014) introduced "MADAMIRA", which is a model used for morphological understanding and solving Arabic language ambiguity problem by joining two previously proposed models; MADA (Habash and Rambow, 2005; Habash *et al.*, 2009; Habash *et al.*, 2013) and AMIRA (Diab *et al.*, 2007). In this system, the same main architecture that was used in MADA is used, but with some extra improvements to the model inspired by AMIRA model. The Java implementation of the system added more extensibility, portability, and robustness to it. As well as the system became faster than other systems. MADAMIRA system was not just for diacritizing Arabic text, it is used for disambiguation, morphological analysis, lemmatization, POS tagging, tokenization, glossing, stemming, named-entity recognition and base-phrase chunking. MADAMIRA benefits from rapid, linear SVMs performed by using Liblinear (Fan *et al.*, 2008). The model was trained on the all parts of ATB-dataset 1, 2 and 3. The results of this model revealed more accuracy than the two combined architecture separately, and a high speed performance, which was measured in words processed per second.

Rashwan *et al.*, (2015) suggested a deep learning approach with confused sub-set resolution (CSR) architecture to automatically diacritize Arabic text. In this model, by using deep neural networks (DNNs) the problem of diacritization of Arabic text is resolved, the CSR method is added to enhance the classification accuracy, and Arabic POS framework is used too. The used dataset was *LDC Arabic Tree Bank*, which is standard dataset. And another dataset, which was collected by the researchers from different resources, is moreover used in the training stage using the proposed model. The results achieved by the system gave an error improvement of nearly 4% when using features of the text; like determining the identity of the last character, which revealed its superiority over state-of-the-art systems that time.

Abandah *et al.* (2015) proposed an automatic Arabic language text diacritization method by using a sequence transcription way. The approach that they proposed is converting a full text with no diacritics, to a fully-diacritized text. They used a deep BiLSTM network. The aim of this network is to make a full use of long-range context in the two input directions and enhance the abstractions of high-level build linguistic. The average DER was 2.09% and the average WER was 5.82%. These results were gained after executing the training process on samples of 11 different books.

Chennoufi and Mazroui (2016) studied the impact of large training corpus and morphological analysis on the performance of Arabic diacritization. In their work, different developed hybrid systems, which were proposed for Arabic text automatic diacritization, were introduced and evaluated. These evaluated systems are: Alkhalil Morpho-Sys1, which was regarded as one of the most important morphological analyzers that time (Bebah *et al.*, 2011). The second evaluated system was the second version of Alkhalil Morpho-Sys1. The enhancement of Alkhalil Morpho-Sys2 compared to the previous version was in language sources and the steps of analysis (Boudchiche *et al.*, 2014). The third evaluated system was AraMorph, which is the Java version of BAMA (Buckwalter, 2002). The first effect studied was the morphological impact, then the learning saturation study was searched. The authors used four different metrics for evaluation. The first two metrics determine the wrongly diacritized words, in the first metric with taking into account the last character diacritic (WER1) and in the second metric with ignoring that diacritic (WER2). The second two metrics were the same but for characters DER1 and DER2 respectively. Depending on the experiments and comparisons, the authors concluded that the scores can be enhanced by acting on two steps: The first step belonging with the level of linguistic analysis; to minimize the error rate, when taking into consideration the last character of the word, they integrated other

syntactic information given by Alkhalil Morpho-Sys2. The second step is related to corpus; firstly, by correcting spelling errors, and secondly by enriching the corpus with other types of texts.

Metwally *et al.* (2016) proposed a multi-layered approach for handling the issue of Arabic content diacritization. The model is a hybrid of the morphological analysis approach, the sequence labelling approach, and the machine translation approach. The framework predicts both the unavailable morphological and syntactic diacritics at high precision. It uses first-order Hidden Markov Model (HMM) as well as an exterior morphological analyzer for morphological diacritization to realize high precision in addition to high scope. The WER of the morphological diacritization accomplished by the model was 4. 3%. To be added, they utilized a CRF based classifier for the syntactic diacritization which accomplished a syntactic WER of 13.7%. One of the most interesting points of their proposed approach, that time, is that its operation depends basically on machine learning methods, which makes the approach appropriate to any dialect that uses diacritics with nearly no changes.

Darwish and Mubarak (2016) proposed an open-source tool which was written completely in Java. This tool was called “Farasa”. The suggested tool composed of four sub-models: POS-tagger, dependency parser, segmentation-tokenization module and Arabic text diacritizer. Using linear kernels, the architecture is depending on SVM-ranking. In that time, Farasa tool outperform other systems that used for diacritizing Arabic text (Darwish and Mubarak, 2016).

Fashwan and Alansary (2017) suggested an automatic diacritization model for MSA text, which they called “SHAKKIL”. The model is tried to handle the problem of diacritization using two levels: syntactic processing level and morphological level. The first level was responsible for detection the case ending diacritic for each word relying on a rule-based

approach representing the shallow-parsing technique. On the other hand, the other level uses four layers; rule-base morphological disambiguation layer, uni-morphological form layer, statistical-based disambiguation layer and OOV layer. The proposed system took the advantage of taking the benefits of both statistical approach and rule-based approach. The authors used an unwell-known corpus which makes their results just belonging to their corpus. Using their model and the corpus they used the WER was 4.56%, and the DER was 1.88% both for morphological. For syntactic, the declared percent of WER was 9.36%.

Zerrouki and Balla (2017) prepared a corpus called “Tashkeela”, which is one of the most interesting corpuses for Arabic language. The corpus is a diacritized Arabic text collected from 97 books that contains either modern or classical Arabic language. From this number of books, 75 million words are contained in this corps. The corpus could be used as a dataset different problem; features and data extraction, disambiguity mechanism and automatic diacritics models. The last finding to be added, the corpus is available freely for use.

Abandah and Arabiyat (2017) introduced a hybrid approach with RNNs to diacritize Arabic text. This approach works as follow: The input text is processed using MADAMIRA, but after removing the diacritics before the processing step. MADAMIRA produces the corresponding diacritics to the words which are divided to their including parts (suffix, prefix, and stem) by analyzing the undiacritized input text. The next step is the sequence reproduction, which is applied after stripping the un-accepted segments and diacritics, then the sequence is entered in to the RNN. Inside the RNN, the partially-diacritized text is processed to produce the fully diacritized text. The last step is the post processing of the text, which manipulates some wrong diacritics and letters before making the final comparison between the output texts of the system with the input diacritized text.

The results of the aforementioned approach proofing that the hybrid approach is preceded the statistical approach in diacritization problem with improvement in WER by 26% and DER by 34%. The comparison made was with the best results for other systems in that time using the benchmark dataset “LDC ATB3”. Another impressive result achieved in the proposed model was the speed up of the time taking for training and diacritizing the words in the input text.

Moumen *et al.*, (2018) tried to solve Arabic diacritization problem using GRU. The approach which he proposed was a simple gated recurrent neural network (GNN). The idea of this project was to compare the influence of using GRU instead of LSTM in the same neural network architecture. The problem of diacritizing Arabic text was used to test this model. The dataset used for training the model was part of “Tashkeela” corpus. And the authors followed the same architecture of (Zitouni *et al.*, 2006) in dividing the dataset. The metrics used for the evaluation were two metrics; DER (ALL), which represents the DER over all diacritics, and DER (Last), which represents DER over only the last character. The results revealed that GRU could be used, as LSTM, to solve Arabic diacritization problem, and the metrics used for evaluation revealed the closeness of the values of the gained results. To be added, GRU outperformed LSTM by decreasing the amount of time used for execution.

Mubarak *et al.* (2019) suggested a model which manipulates the Arabic text and uses a consolidated character-level sequence-to-sequence deep learning approach. This model is used to solve both case endings and core-word diacritization. They use standard neural machine translation setup to divide the words into characters. The proposed model achieved a WER of 4.94%. Necessary to add that, the approach that proposed in their article is not clear to be understood, because the model was not explained in a practical

way. Another weakness of the study is that the dataset that was used is differ from other well-known datasets of this field.

Abandah and Abdel-Karim (2019) proposed a novel approach to make an automatic diacritization for Arabic text, which is faster and more accurate of the other known automatic diacritization approaches. They used four BiLSTM Layers to receive the needed diacritized text. By adjusting and fixing the used network's parameters, such as: network-size, network-architecture and hyper-parameter, the results were very promising. With the values of DER 1.97 on the larger new *Tashkeela* dataset, and 2.46% on the *LDC ATB3* dataset. The percentage of improvement rate gained was 47%, which overcomes the previously published results.

AlKhamissi *et al.*, (2020) suggested a hierarchical recurrence model for dealing with Arabic diacritization problem. The idea was to use two different recurrence hierarchy levels; one operates on the character level, and the other operates on the word level. To connect between the aforementioned levels, a cross-level attention module is used. The task approach is a softmax layer which recounts valid combinations of diacritics. A recurrent decoder is connected to the architecture to improve the results. To enhance the final result, the researchers used some techniques such as: majority voting and sentence dropout. The results for the experiments achieved 5.34% for WER, which outperforms *Tashkeela*.

Al-Thubaity *et al.* (2020) suggested a bidirectional long-short term memory neural network with conditional random field model for recovering Arabic text diacritics. The model uses sequence-to-sequence schema without using any other schemas such as dictionary or morphological analyzers. The proposed model was trained on four different datasets: Penn Arabic Treebank (ATB), the Holy Quran, King Abdulaziz City for Science and Technology text-to-speech (KACST TTS) dataset and Sahih Al-Bukhary. The result

of the model was 2.37% for DER on *ATB* (1, 2 and 3) dataset which outperforms other previous models. The results moreover reflected that the size of dataset, that used to train the model, affected the value of metrics used, and the quality of diacritization have a large impact on the accuracy of the model.

Madhfar and Qamar (2020) proposed three deep learning models for solving the Arabic text diacritization problem. The first model is used as a baseline model. The model consists of six layers; embedding layer, three BiLSTM layers, fully connected layer and finally softmax layer. This baseline model was used to test the dataset used in the experiments and to reveal how this model dealt with the problem of diacritization. The second model is an encoder-decoder model which was adjusted from Tacotron (Wang *et al.*, 2017). The encoder part of this model was the same as Tacotron, whereas the decoder and attention type are different. The third proposed model relies on the encoder part of the text-to-speech-approach, taking into consideration the best results achieved by this approach. The corpus used in this work was "Tashkeela" with some preprocessing to the text included in the dataset. The results revealed that the third model outperforms the other proposed models in all metrics used in the experiments, and this model was faster than other two models in training.

2.3 Correcting Spelling Mistakes

Detecting and correcting spelling mistakes is one of the problems that gained the interest of NLP researchers from an early stage. Many researchers, using different approaches, tried to handle this problem. With the time, the accuracy of the correcting spelling mistakes increased, and the detecting error is becoming faster rather than more accurate (Attia *et al.*, 2016).

Damerau (1964) was one of the earliest researchers who addressed this issue based on four edit operations (insertion, deletion, substitution, and modification). However, the

limitation of memory and computation possibilities were an obstacle for his work at that time.

Church and Gale (1991) used probability scores, considering word bigram probabilities, depending on a noisy channel model to rank a series of spelling nominees. Kukich (1992) categorized the work on spelling mistakes into three parts: (1) Error detection, (2) Separated word correction, (3) Context-sensitive correction. Brill and Moore (2000) attempted to enhance the noisy channel model by learning common string-to-string corrections. Delden *et al.* (2004) had the ability to correct spelling mistakes using supervised and unsupervised methods of machine learning, especially the mistakes of word splitting and merging.

Han and Baldwin (2011) proposed a method for normalizing the ungrammatical words in the short messages of Twitter. By using SMT for the choosing of candidates, they build a normalization dataset. Wu *et al.* (2013) used the same method that Han and Baldwin (2011) used, but for Chinese language. They built a spelling mistake detection and correction model using a decoder based on the SMT model for correction.

The case of spell checking and spelling mistakes correction for Arabic language has been investigated by many researchers. Shaalan *et al.* (2003), Atwell (2012), and Shaalan *et al.* (2013) classified and characterized the spelling mistakes in Arabic.

Haddad and Yaseen (2007) suggested a hybrid model that uses morphological knowledge to device morphographic principles to determine the word recognition and non-word correction process. They proposed two probabilistic measures for correcting mistakes: Pattern-Root Predictive Value and Root-Pattern Predictive Value. But the testing of the system performance was not represented.

Magdy *et al.* (2007) proposed a system for diagnosing lexical error for second language learners; Arabic Lexical Error Diagnosis System (ALEDS). The authors categorized the

lexical errors in to three types of errors: errors in semantic or word choice, errors at the interface of lexical and grammar and errors in the word formation. The suggested system uses edit-distance mechanisms as well as constraint relaxation techniques to identify lexical errors that learners fall in. In the proposed system, NLP tools such as error analyzer, morphological generator, and morphological analyzer, were included to diagnose and manipulate ill-formed input text. The techniques used in the system allowed users to send a feedback, and the system were able to provide error-specific diagnosis.

Hassan *et al.* (2008) improved a language independent model that uses an automated finite state to suggest a nominee correction with a determined edit distance from the mistaken word. For assigning scores to the candidates and choosing the best correction in the afforded context, a word-based language model is used after the generation of candidates. The full form entries Arabic dictionary consists of 526,492 entries, and the number of mistakes tested on it was 556 mistakes. The system they proposed was not compared to any other systems, rather than they did not explained whether the test mistakes were actual errors extracted from generated text or real ones.

Al-TAAni *et al.* (2009) try to solve one of the important aspects in Arabic language errors that must be taken into consideration; checking agreement between numerals and counted objects in any text of Arabic language. They proposed Arabic numeral checker has the ability to handle the previously mentioned problem of the numbers in range (1 to 999,999,999,999). The presented system relays on a deep morphological examination by using finite state transducers (FSTs), to generate the expected numeral and the transformation process from digit of numbers to written text numbers. By using FSTs, the size of the lexicon is marvelously reduced, and the results became more accurate. An index-based approach is applied for sorting morphism to make the access and restoring of morphisms in the FST faster. To be mentioned the results were compared with

commercial software and the proposed system outperform it. Lastly, the proposed model could be used as a standalone system, or a part of more generalized systems that handling Arabic text problems.

Shaalan *et al.* (2010), suggested a model for non-native Arabic learners which analyses and corrects spelling errors. The researchers recommend a two-layer spell-checking architecture dealing with spelling error detection and correction. The suggested error detection mechanism is applied on best of Buckwalter's Arabic morphological analyzer to illustrate the capability of the proposed approach to recognize conceivable spelling mistakes. The adjustment mechanism adopted a rule-based edit distance algorithm. Rules are planned in depending on common spelling mistake patterns made by Arabic learners. Error correction used a different filtering approach to propose last adjustments. The approach utilizes semantic data given in working out questions in arrange to attain profoundly precise discovery and correction of spelling mistakes made by non-native Arabic learners. At long last, the proposed approach was assessed using real test information and promising comes about were accomplished.

Shaalan *et al.* (2012), for spelling correction, they used the Noisy Channel Model trained on word-based unigrams. Comparing with Microsoft Spell Checker, their system performance was very bad.

Alkanhal *et al.* (2012) improved a spelling mistake detection and correction model for Arabic language data entry errors. The overfitting of their results made their work undependable. The overfitting problem was because their work was tested on developed set. Another disadvantage of their system was the small size of the dictionary that they used; (427,000) words.

Zribi and Ahmed (2013) suggested an idea to implement their proposed approach on a distributed mechanism, and they called it multi agent system (MAS). The proposed

system can detect semantic errors in Arabic text. In their work, the researchers applied four editing operations on the text: deleting a letter, changing the order of two consecutive letters, interchanging a letter by another one and adding extra letter to the word. And this was applied to dictionary words. By using automatically built forms called “the number of lexically neighboring words”, the number of correct forms were calculated for each individual word. The results reflected the degree of similarity between words. Comparing these results for Arabic languages with other two languages; English and French, the researchers found that the average number of neighboring forms reaches the value of 26.5, which is a large value compared with 3 and 3.5 for English and French respectively. The previous mentioned method is used to calculate the probability of receiving a correct word when the word is mistaken. With a percent of 5.79, the probability of an Arabic word is 10 times of English word and 14 times of French word respectively. Because the difficult of capturing semantic errors in Arabic language compared to other languages, the researchers suggested an architecture that joining with different methods relies on linguistic and statistical information within MSA. The results reflected a competent performance of the system based on the used metrics; recall and precision. By combining the different methods, the distributed system recovered from the shortage of each method working separately as well as achieving a great result.

Jeblee *et al.* (2014) proposed a system for automatic Arabic error correction, and it was presented in ANLP-QALB 2014; shared task on automatic text correction for Arabic. The system joined between three parts: statistical language modeling approaches, rule-based linguistic approaches and machine translation-based methods. The system used different experiments to examine its ability to correct Arabic language errors. During these experiments the metrics, (precision, recall and F1-score), were used to compare its results with a spellchecker system used as a baseline model. With a result of 74.73% for F-score

metric, when the improved set of a QALB corpus is used without punctuation, the system outperformed the baseline system. When the punctuation errors are taken into consideration, F-score decreases to 65.42%. Despite of these results, the proposed model still outperforms many other systems.

Attia *et al.* (2014) described a hybrid Arabic spelling and punctuation correcting system (HASP). Firstly, for punctuation errors correction the proposed system used conditional random field (CRF) classifier. Secondly, for capturing errors and producing and filtering candidates, the system used word list (or open-source dictionary). Thirdly, to select the best candidates an n-gram language approach was used. Lastly, for the text normalization a group of deterministic rules was used. The researchers moreover provided a useful result after number of experiments belonging to word alignment and its effect on spelling error correction at the character level. The proposed system used the QALB dataset and used recall, precision, and F-measure as metrics. The results of the system were useful ones, but not very promising.

AlShenaifi *et al.* (2015) proposed a hybrid cascade model for Arabic spelling error detection and correction. This system was a participant in the second shared task on automatic Arabic error correction (QALB-2015 shared task). The proposed system included many components, each individual component can process a specific type of spelling error. These components are: Statistical based, lexicon based, and rule based. These components were arranged in a cascade style. The main model moreover consists of two main systems: distance-based model and probabilistic-based model. Through the experiments applied using the proposed model, the idea of using a cascading arrangement for the different components achieved an enhancement in the results, which were: 0.51 for recall, 0.67 for precision and 0.58 for F1-score.

Rozovskaya *et al.* (2015) introduced a summary about the second QALB shared task on automatic text correction for Arabic QALB-2015. The second shared task was an extension to the previous one (QALB-2014), which concentrate on errors correction on the Arabic text made by native Arabic speakers. In QALB-2015, in addition to the previous task in QALB-2014, the interest moreover became on the correction of Arabic text errors which made by non-native Arabic speakers. The training dataset and development data is provided to the participants. Despite of this, the participants are freely to use any other data for improving their models. The official metrics that used for the competition are Precision, recall and F1-score. The model which was used as a base model to compare the suggested models with, was MADAMRA (Pasha *et al.*, 2014). The participated teams were eight teams which suggested twelve systems to take a part in the completion. The results were very promising with F1-score of 72.87 for the winning system.

Attia *et al.* (2016) improved their own dictionary of Arabic words to contain 9.2 million words. The mistake types were analyzed by enhancing the error model and establishing an edit distance re-ranker. They moreover developed the language model by studying the level of noise in various data sources and choosing the best subset to train the system on. The system which they proposed performs amazingly better than OpenOffice Ayaspell 3.4, Google Docs (tested in April 2014) and Microsoft Word 2013.

Mars (2016) proposed a system for words error detection and correction in Arabic language. The suggested model is a sequential set of different architectures, which are: (rule-based, lexicon-based and statistical-based) architectures. The researcher divided the spelling errors in Arabic text in to two main categories: 1) Non-words error, which is any word that does not exist in Arabic language. 2) Real-word error; where the word is valid however unseemly within the context. In his work, the researcher depends on different

resources to overcome the problems of error detection and correction. In non-word error, a hybrid pipeline approach is used. For error detection, a dictionary of Arabic word and named-entity list were used. Necessary to add that both dictionary and the named-entity list were prepared by the researcher. For error correction, many approaches were used to receive a better result, and these approaches are: MADAMIRA model for error correction, space issue model, rule-based with name-entity error corrector model, ranking alternative candidates model, Levenshtein distance model and choose the gold correction model, which was made manually for each spelling error. The test dataset was composed of different articles that were gathered from various resources to provide a test dataset of 53,607 tokens. The number of spelling errors extricated was 2067. The author used precision, recall and F1-measure as metrics. Compared to other systems, the model outperformed the former systems with 67% F1-score.

Zahui *et al.* (2017) introduced (*EL-mossahih V1.0*), which is a hybrid model used for detection and correction of typographical and phonetic transcription errors in Arabic text. The model uses different methods to handle these errors, the methods used are: a language model, phonetic transcription, permutation technique, neighborhood technique and dictionary. The learning corpora that used in the proposed system was collected from different resources which were not all mentioned by the researcher. The model was tested multiple times in different number of words each time. In every experiment in the detection phase, the percentage number of the wrong words detected by the proposed system, to the number of words with an error is registered, and after multiple experiments the average value is calculated. The average value calculated in this phase was 74.75%. The same procedure is applied to the correction phase, but the ratio was taken using the number of wrong words corrected by the model, to the number of wrong words detected by the model. The average value calculated this time was 80.22%.

Madi and Al-Khalifa (2018) introduced a work-in-progress work for enhancing a web-based tool which performs Arabic grammatical error detection. The idea presented for the model is that the user can write some Arabic words in a sentence or few sentences in a text area in the browser, the text will be sent to a web server which includes a deep learning model for detecting the grammatical errors. The retuned-back text will indicate the wrong words by contrasting the word color or underlying it, and the error type will be determined for the user to provide him the ability to correct it. The system will not correct the errors as it is just a detection model. The deep learning models that will be examined are: RNNs, LSTMs and BiLSTMs. Necessary to add that, the QALB corpus will be used in both training and testing phases.

Azmi *et al.* (2019) proposed a method to correct spelling error called real-word spelling error. This type of error relays on the using the correct word in the sentence not the word itself. Because the word itself is correct and the typical spell checker will accept it. For example, letter (ت) is replaced by the letter (ق) resulting in a correct but erroneous sentence (لذيذ كالقمر), meaning "tasty like moon", whereas the intended sentence was (لذيذ كالتمر), "tasty like date (fruit)". We see that both words (القمر ، التمر) are correct words, but the use of word (القمر) in the sentence instead of word (التمر) gives incorrect meaning and hence, an erroneous sentence. The authors propose a language model that uses word and stem n-gram (n=1-3) as well as machine learning for the detection step. This phase achieved a precision of 83.5% and a recall rate of 99.2%. For the correction phase, an n-gram was used and the authors report a 98% correction accuracy result.

Al-Hagery *et al.* (2019) explained a hybrid approach for cleaning misspelling and missing Arabic words in data warehouse. The proposed model is relying on decision tree induction (DTI), as well as Arabic misspelling detection and correction model (AMDCM) which is used for the multi-level cleaning. The model consists of sub-systems and procedures

which introduce the resulted data without errors, these sub-systems are: Arabic misspelling detection approach, Arabic misspelling correction approach, system for cryptic values, model for dummy values, approach for unification naming styles and improvement hybrid algorithm used for cleaning missing and misspelling Arabic data. The dataset used contained seven attributes for students in Computer College. The researchers presented the data before and after processing, but they did not refer to any metrics.

Mahmoud *et al.* (2019) recorded a patent for a novel Arabic spell checking error model. A data driven mistake model was unveiled by the researchers. The model is based on mistake designs found at the morphemes level. An architecture is produced by error-correction designs generator and is put away in an error-correct patterns database (ECPD). The ECPD is utilized in conjunction with a correction candidates' generator (CCG) to supply a list of redress candidates for a given mistake word. The error model can learn the sorts and shapes of the language patterns from a clarified corpus. The error approach can be utilized to analyze the sort of error and can provide candidates adjustments for wide ranges of Arabic spelling mistakes. The QALB corpus used as dataset in the different training experiments and test phases. The results were very promising, and the proposed model had the ability to suggest different words for mistaken word to choose the correct word from.

Madi and Al-Khalifa (2020) proposed error detection system for Arabic text using neural sequence labeling. The error detection in MSA text is processed using multiple experiments on different approaches of neural network models. The corpus was created by the researchers, and it contained just 620 sentences. So, they used cross-validation on the data to apply the evaluation stage on the model. Necessary to add that the max length of the words in the used dataset was just 17 words. The researchers proposed three

architectures to be researched on the suggested dataset; namely: (Simple RRN, LSTM and BiLSTM). The aforementioned approaches were examined with different values of hyper-parameters to receive the best accuracy with chosen values. The base model that was used to compare the different models with, was Microsoft Word 2007. The results revealed that BiLSTM outperforms the other models in precision and F-score with values of 80.74 and 81.55 respectively. On the other hand, with the value of 93.8 of recall the LSTM model achieved the highest score. Necessary to add that the researchers tested the different approaches on 50 sentences of QALB corpus and LSTM outperforms other models in both precision and F-score, whereas the baseline model defeated the other models with a value of 91.96 in recall metric.

Alkhatib *et al.* (2020) introduced a systematic approach for spelling error detection and correction, and for grammatical error detection and correction at the word level. The proposed model uses Bi-LSTM and word embedding and replaces the decoder with a polynomial network model classifier that used for error detection. The corpus used in the experiments was collected and processed by the researchers from multiple resources. To make it suitable for text error detection and correction, each ill-word or mistaken one was validated, annotated, and manually checked out by a specialist. The result for the well-prepared corpus was receiving a 15 million fully inflected Arabic words to be used in the training phase. The system evaluation step was performed using a test set which was a dataset enhanced by Open Source Arabic Corpora (OSAC) (Saad and Ashour, 2010). The system achieved an F-measure of 93.89% compared to other two known tools; *Ayaspel* version 3.4 and *Microsoft Office* 2013, which achieved 76.4% and 53.8 respectively.

The researchers pointed that they used intra-attention mechanism which improved the performance of the model even with some fail in individual cases.

Chapter Three
Deep Learning for Natural Language Processing

3. Deep Learning for Natural Language Processing

To explain the concept of deep learning in NLP, we firstly must understand the idea of natural language processing, and how the distinct models process the different tasks of ML that belongs to NLP. In this chapter, we started with defining and explaining the concept of NLP. Then, the language as probability model is explained in the second section of this chapter. In the third section, neural networks are presented to make a deep understanding to the next section, which presents RNNs. The fourth section discusses RNNs and explains how it works. Moreover, to complete the general idea, the encoder-decoder framework is discussed in the last section of this chapter.

3.1 Natural Language Processing

Natural language processing is considered as a wide field of information that processes the computational algorithms to analyze model and produce natural human language. Within the age of the omnipresent World Wide Web and the unstoppable rise of unstructured content data, NLP must be one of the foremost critical advances. It covers numerous diverse tasks such as text summarization, automatic parsing, named entity and machine translation. NLP is exceedingly associated with Artificial Intelligence (AI) science and frequently is considered as its part.

Typically, the NLP pipeline included numerous task-specific steps from pre-possessing to the ultimate application. In later years, this conventional approach was greatly moved by the modern deep leaning-based approaches, which have accomplished exceptional performance in different NLP tasks. This approach infers a preparing of end-to-end models without manual feature interpolation.

One of the most important steps in NLP applications, is to pre-process the data that will be used as an input data to the model. The initial manipulating of the source data will

significantly affect the results of the overall model. The general steps for pre-processing the input data as follows:

1. Cleaning the data (capitalization, noisy data removal, stop-word removal, etc.).
2. The normalization of the data (lemmatization, stemming).
3. Annotation of the input sequence (part-of-speech tagging, tokenization, etc.).
4. Analysis of the data (counting, data summarizing, data categorizing, etc.).

The previous steps are valuable for each system, but for instance, when we are dealing with the neural network systems, tokenization is the most important step for data pre-processing. In tokenization method, the text is split into diverse tokens, which makes it easily processed by the NLP system in sequential order. By using tokenization, a fixed vocabulary and one hot encoding is used to encode an input sequence to a set of vectors that can be processed by a neural network-based engine.

The most common neural networks approach that uses tokenization is word-tokenization. In such approach, the input text is divided into pieces which distinguished by punctuations or spaces. Despite of the simplicity and fastness of this method, it just only operates well with fixed vocabulary, and the summarization becomes an open-vocabulary problem. Necessary to add that this method is performing badly with the languages that are morphologically rich.

Character tokenization is another method that emerged to overcome the weaknesses of previous aforementioned one. From its name, the vocabulary in this approach is equals to the alphabet of the language, because the tokenization here divides the text into alphabets. This method was not widely used in NLP models despite of its ability to recognize new words. The reason behind this was, if we take into consideration the probability distribution of the character conditioned of having the precedence character, this

distribution would be almost the same as discrete uniform that needs larger models to understand the language structure. On the other hand, large models will drop down their accuracy of the previously built models comparing it with the usage of word tokenization because of the needs for more data.

Within the later past, a new tokenization method called “Byte-Pair Encoding” was broadly used in the community of NLP, particularly in machine translation research. This method overcomes the problem of “out-of-vocabulary” by encoding unique and unknown words as units of series of sub-words. The idea of this approach that different classes of words in the same language can be used as a combination of different meaningful units. For instance, the compositional, morphological and phonological rules of the language present different loanwords, cognates and compounds. Byte-pair encoding can capture the most common series of characters in the language by initiating a compact vocabulary.

3.2 Language as Probability Model

The foremost common task of NLP deals with building the language's models. This errand is profoundly related to the abstractive content summarization issue and its understanding, is fundamental for managing with the cutting-edge summarization models. Language modeling is often seen by statistical-based NLP as unsupervised distribution estimation from a set of illustrations each composed of variable length groupings of tokens. Each token is accepted to be left-conditioned of the previous tokens shaping the conditional distribution over vocabulary. Because pf the nature of sequential ordering, the product of the conditional probabilities is mostly calculated to receive the joint probability (Radford *et al.*, 2019).

Equation 3.1 illustrates the previous idea.

$$p(x) = \prod_{i=1}^{n} p(s_i | s_1 , \dots , s_{i-1}) \tag{3.1}$$

Where, function p represents the joint probability and S_i represents the *i-th* token. The key strategy for producing distributed representations of tokens, which moreover called embeddings, is the language model. These embeddings will be later used in determined NLP problems. In natural languages, each sequence is manipulated in two directions: from right-to-left and from left-to-right. To achieve this, mostly pair of distinctive models are trained, and then joined together to attain the best accurate results. However, lately, the explicit bidirectional conditioning is obtained by using the idea of conditioned token masking (Devlin *et al.*, 2018).

This makes us to re-write equation (3.1) in the new following formulation:

$$p(x) = \prod_{i=1}^{n} p(s_i | s_1, \dots, s_{i-1}, s_{i+1}, \dots, s_n) \quad (3.2)$$

Recently, counting models can form any precise language representation. Unfortunately, this formulation would use infeasible amount of time and memory. So, in practical way, some complex functions are used to overcome this problem. These functions are trained on a corpus of texts to receive the approximation of language models. Nowadays, deep learning models have achieved very promising performance in language processing, which make them as a standard approach for this task.

3.3 Neural Networks

Inspired by the biological neural network, the artificial neural network was basically advanced as a mathematical model. In biological neural network, the electrical signals are transferred between neurons, which makes the needed orders to be translated to do the required action. The same procedure is applied in the artificial neural networks but in a mathematical manner; such that, the inputs are processed to receive the desired output.

The mathematical model of the biological neural network was firstly proposed by McCulloch and Pitts in the year of 1943 (Dash and Dubey, 2012). The previous mentioned model had the capability to solve binary issues without having the ability to

learn. The first artificial neural network, which has the potential to change its own weights, was first presented in 1957 by Rosenblatt. This network has the ability to enhance its weights, and as a result the ability to learn. This artificial neural network called “Perceptron”. The perceptron works as a linear binary classifier that sets a threshold function on a linear series of the input features. The threshold function in the perceptron could be replaced with other non-linear activation functions (*e.g.*, tanh, RELU, sigmoid, etc.), which are used in most different approaches of neural networks. Figure 3.1 represents the simple perceptron.

The idea of perceptron opened the minds to improve the use of this simple network to more innovative one. The Multilayer Perceptron (MLP) consists of a group of perceptrons, each group is in a connected layer with other layers. The MLP should at least consists of three layers; the input layer, the hidden layer that has a non-linear activation function for each neuron, and the output layer. Necessary to add that the neurons are in parallel in each layer. Figure 3.2 illustrates MLP.

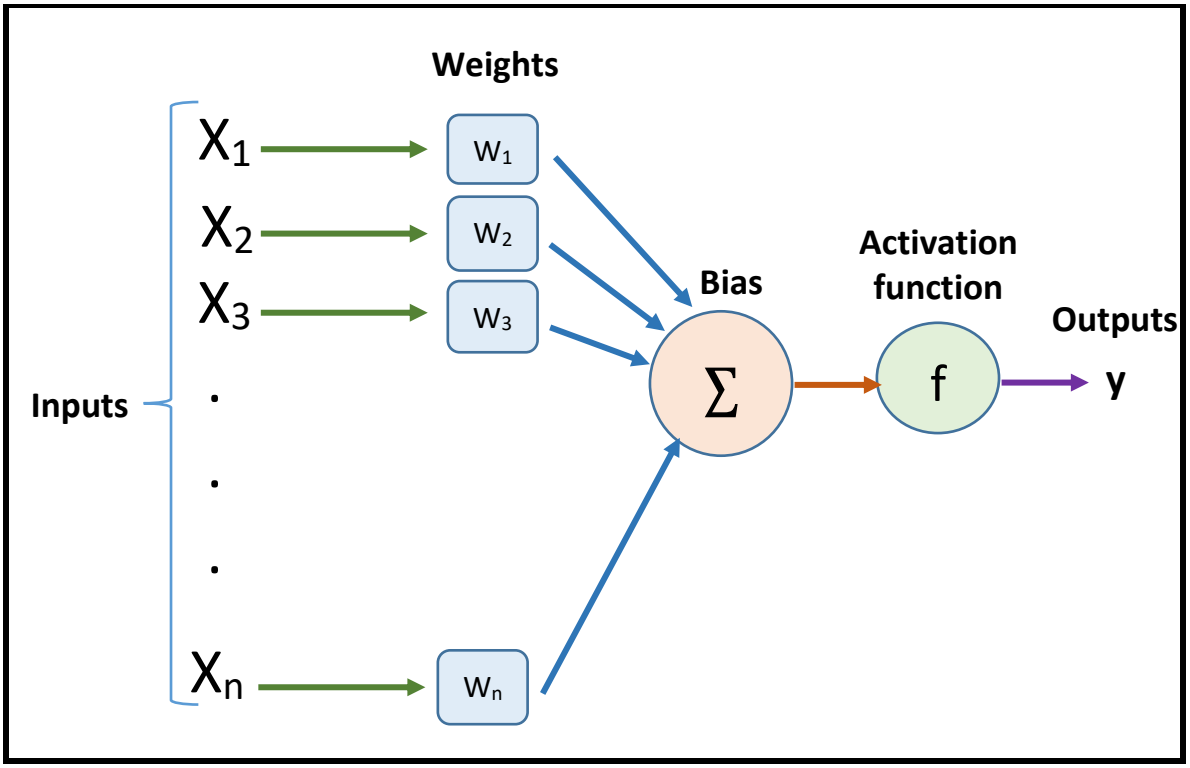


Figure 3.1: Simple Perceptron

A universal function approximator is used to mathematically prove the MLP. Theorem 3.1 illustrates the theorem of universal approximation:

Theorem 3.1 A feed-forward network with one hidden layer holding a restricted number of neurons, can approximate continued functions on consolidated subsets of Rn, under moderate presumptions on the activation function (Pinkus, 1999).

Assuming a neural network is used as a classifier, this means in theoretical manner that for a huge number of neurons by using a learning algorithm which achieves a global minimum, in accurate method, the differentiation between non-linear devisable data can be achieved using a forward neural network (FNN). In other words, to find an algorithm for an arbitrary network that can achieve a global optimum is impossible. So, a back-propagation algorithm, which utilizes the gradient descent optimization of chain rule and loss function, is usually used to train the FNN to receive the local minimum. Which in turn ensuring to be very close to the global minimum (LeCun *et al.*, 2015). The previous algorithm has a drawback called: vanishing gradient problem. This problem appears when the neural network reaches a specific depth. The cause of aforementioned problem is that the weights are being freezed when there is a continuously decrease in the gradient of the deep layers. To overcome this problem, many techniques were suggested, for example, residual connections, hardware updates, and receiver activation function.

3.4 Recurrent Neural Networks

The ordinary FNN considers that all inputs are of the same size and are uniformly structured. Moreover, because of the truth that the continuous data has a sequential nature, the FNNs will not be able to process this data easily.

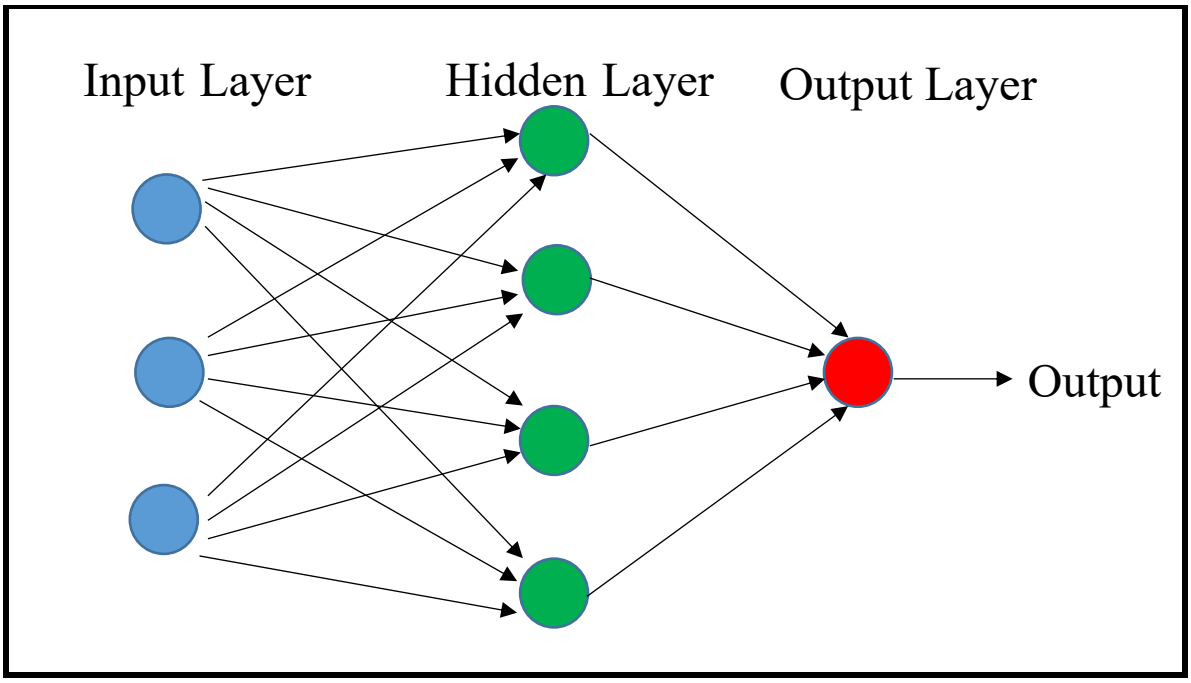


Figure 3.2: Multilayer Perceptron

We can find sequential data in many forms, such as: text, videos, speech, time-series, etc. In our work, we will focus on text processing methods, which have a close nature to our problem, such that, the text we will deal with is different in length for each part. Necessary to add that the using of standard forward neural networks in such problems will consume numerous using of computational resources, time, and memory, which in turn make it useless. Because of all previously mentioned reasons, a different type of architecture of neural network is emerged. The new type can deal with sequential processing, and it reflected promising results when it is used with different sequential problems. This type is called “Recurrent Neural Network” (RNN). Figure 3.3 views the general structure for FNN and RNN.

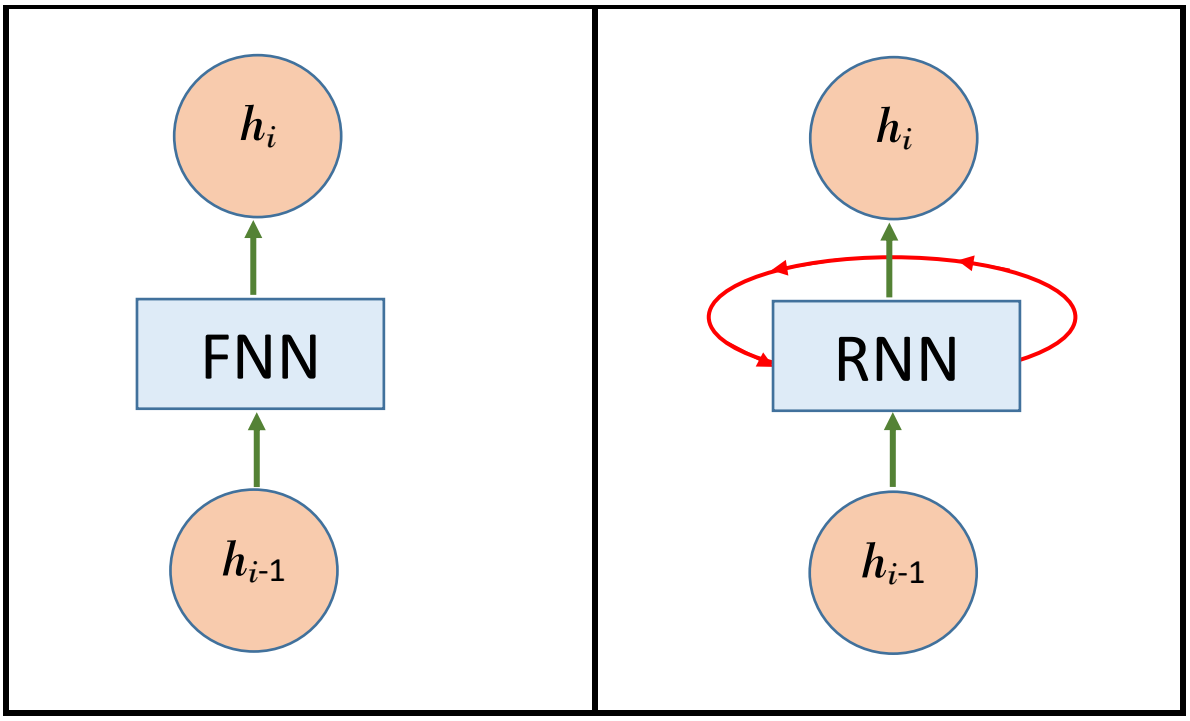


Figure 3.3: The general architecture of the FNN and RNN

The strength point of the RNN lies in its ability to keep the results from the previous step. This makes the advantage of recalling the order of the elements, rather than in any length, the input context is achieving the modeling of the input dependencies. For each output of the network, a conditioned mechanism is applied depending on the previous computations by using a combination of a hidden state from the current hidden state with the previous iteration. For every token of the input sequence and the new update of the network's weights, a recursive procedure is repeated. To be formulated, the deterministic state condition function for basic RNN is illustrated in Equation 3.3:

$$h_t^i = f(W_{n,n}^a \, h_t^{i-1} + \, W_{n,n}^b \, h_{t-1}^i) \tag{3.3}$$

Where function *f* is the non-linear function, *i* is the layer's number and $W_{n,n}^a$, $W_{n,n}^b$ are the learnable matrices.

The disadvantage of the standard RNN is that it just processes the sequence in unidirectional way. For each new token, the network looks in reverse to the past states to find any inter-tokens connections. Nevertheless, it is apparent that a sequence's

components can relate to the following tokens as well as to past. Thus, by practicing, a different approach of RNN that processes every part of the input sequence in both directions, which as a result, achieves more informational text representation. This type of RNNs Therefore called bidirectional recurrent neural networks (BRNNs). To simplify the idea, we can conceder BRNN as two of RNNs such that the hidden states for each distinct position in the input sequence are combined to introduce a final bidirectional representation.

The standard RNN has a defect that is “vanishing gradient problem”. This problem hinders the network from dealing with long sequences. Another problem, which is close to the aforementioned problem, that the influence of the past input is decreased and disappeared with increasing the length of the sequence. This causes the inability to train the RNNs on long sequences or using them to generate long ones. The problem of handling long sequences has been solved using different methods. The idea of keeping the needed data in a long-term memory is applied by LSTM (Hochreiter and Schmidhuber, 1997). LSTM as a long-term memory is controlled by three gates; input gate, forget gate and output gate. This long-term memory is regulated using the additional cell state. The controlling of the input flow to the cell is organized by the input gate. Forget gate determines, in the context, whether to keep the previous information or not. In addition, the regularization of the output flow is governed by the output gate. Figure 3.4 describes the architecture of the LSTM cell. Another recently proposed cell is GRU. We can say that GRU is an LSMT cell but without output gate and has a less parameters than LSTM (Cho *et al.*, 2014).

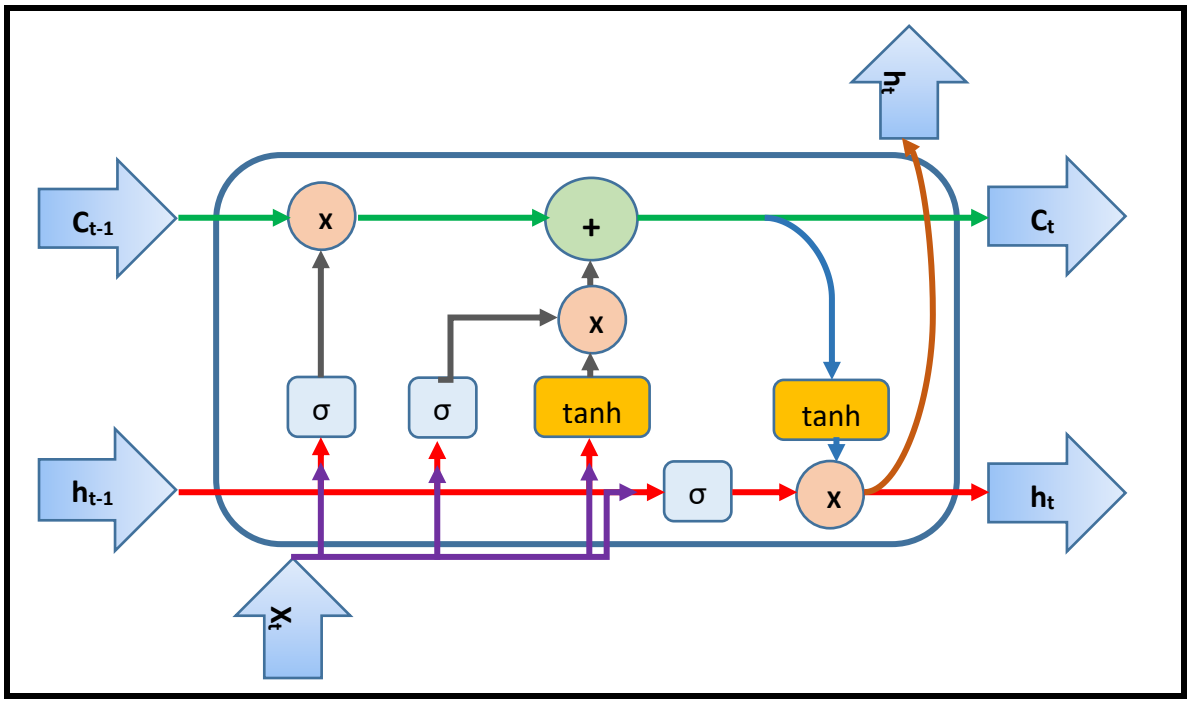


Figure 3.4: Architecture of the LSTM cell

The previous mentioned cells had the ability to score a well dominant results in many fields, but the improvements and innovative visions in the field of ML is continued to discover and apply different approaches such as Attention-based models which achieved a prominent result in different types of ML problems. Attention in NLP is explained in the next chapter.

3.5 Encoder-Decoder Framework

The simple approach for the sequence-to-sequence model is to have only single neural network. The input sequence of this architecture is a series of input text, whereas the processed output divided by special tokens. In this approach, the input sequence differs in length from the output sequence which diminishing the effectiveness of a simple use of a single network. As a result, most models are built using encoder-decoder approach. In the encoder-decoder framework, two networks are trained; the encoder is used to encode the input sequence, whereas the decoder is used to produce the output text. Figure 3.5 illustrates the general structure of the Encoder-Decoder model. In the starting step, the embedding matrix maps the input sequence to the distributed representations. After

that, the distributed representations are transformed to the encoder-hidden states, which are contextual distributed representations. These hidden states reflect some information about the sequence in each vector. Encoders, as used in different models, depends on different approaches and the most usable ones are different types of RNNs. Because of the problematic in dealing with sequential processing, attention-based and convolutional models are recently replacing the basic RNNs. The output from the hidden states of the encoder is inferred by the decoder. The decoding process occurs in a sequential manner, such that, each recently generated token is separately conditioned on the sequence of the previously generated tokens. Its architecture is mostly based on the same principle as the encoder. On the initial step, the decoder calculates the vector that holds the information essential for the generation of the next token. Taking into consideration that the decoder is previously having the set of hidden states of the encoder and the embedding vectors of the output draft. The next step is that the vector of probabilities over the vocabulary is mapped by the generator layer. In the aforementioned step, all vector's components are restricted to the interval (0, 1) and add them up to "one" to make them able to be interpreted as probabilities. Moreover, this is made by the helping of the normalization technique "Softmax". The next decoding iteration is using the embedding of the largest probability token value as an input. Finally, after reaching the end of the sequence token, the generation will be stopped (Cho *et al.*, 2014).

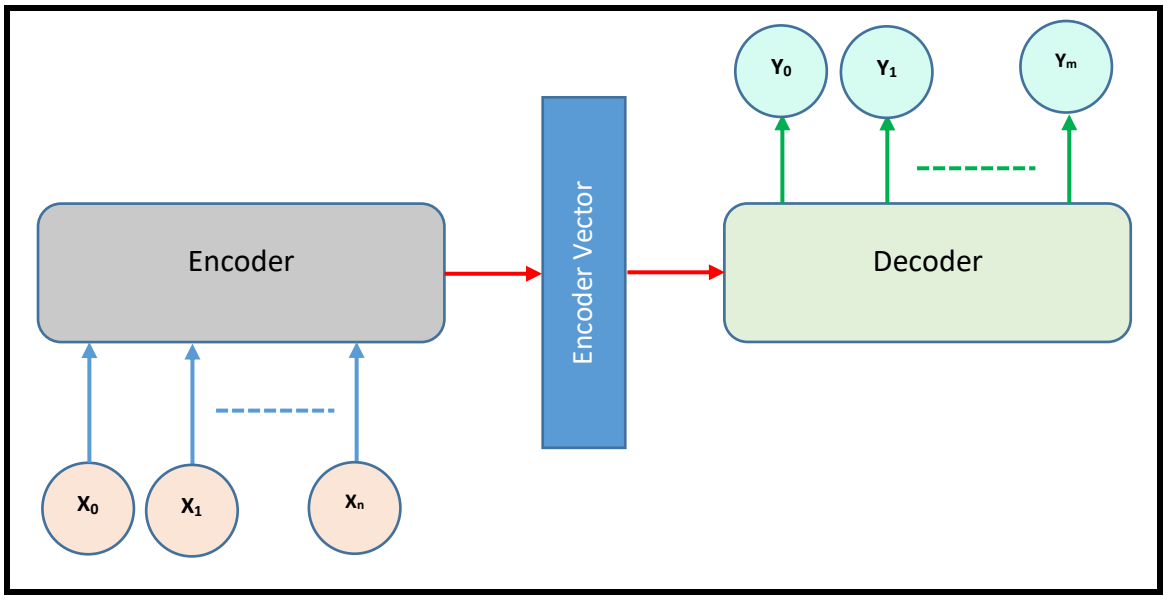


Figure 3.5: Encoder-Decoder Model Architecture

3.6 Sequence-to-Sequence Architecture

The typical RNN usually has a fixed-length input and output vectors; such that, both lengths of the input and output vectors are the same. However, in some cases such as machine translation, speech recognition, diacritization and CSM, input and output sequences are not of the same length. As a result, sequence-to-sequence architecture is proposed to overcome this problem.

Sequence-to-sequence models considered as generative neural networks models that taking a string of inputs of *n*-length and producing a string of outputs of *m*-length. There are two processes for the sequence-to-sequence model: encoding process and decoding process. In the encoding process, which is essentially an RNN model, the input sequence $a = (a_1, a_2, \ldots, a_T)$ is entered to the encoder. Then, only the hidden state of the last layer (h_t) is saved, which is a vector usually called *sentence embedding* or *context vector* (c). This differs from the simple RNN model of that; in the simple RNN, the output of each state is considered. Equation 3.4 represents the hidden state of the last layer (h_t), and Equation 3.5 illustrates the *context vector* (c).

$$h_t = LSTM(a_t, h_{t-1}) \tag{3.4}$$

$$c = tanh(h_T) \tag{3.5}$$

where, h_t is a hidden state at time t, and c is the context vector of the hidden layers of the encoder.

On the other hand, by feeding the decoder of the *context vector* (c) and all preceding predicted values ($b_1, b_2, \ldots, b'_{t-1}$), it can predict the next value (b'_t). Figure 3.6 illustrates a sequence-to-sequence model representation of three time steps. The input sequence is (a_1, a_2, a_3).

From this figure, we note that each part of input sequence (a_1, a_2, a_3) is fed into the embedding layer of the encoder, which represent each part of the input sequence in a fixed length vector of defined size. The produced vector from the embedding layer is a dense one, which having real values instead of 1's and 0's. The fixed length of word vectors enables us to reproduce words in a better representation along with reduced dimensions. In other words, the embedding layer behavior is look like a lookup table; the keys of the table are the words, whereas the dense word vectors are the values. The encoder and decoder parts of the sequence-to-sequence model may contain any type of RNN; we will consider the type of RNN is LSTM.

In each time step, layers of recurrent unit accept an input token, which is used to gather the related information and produce a hidden state. Each unit combines between each hidden state and the input. On the other hand, this unit returns the output, which is discarded, and it returns a new hidden state. It is valuable to add that, the last hidden state (h_3 in in Figure 3.6) represents the encoder vector. This vector is containing all the information from the previous RNN units, and it is the entirely information the decoder will get from the input.

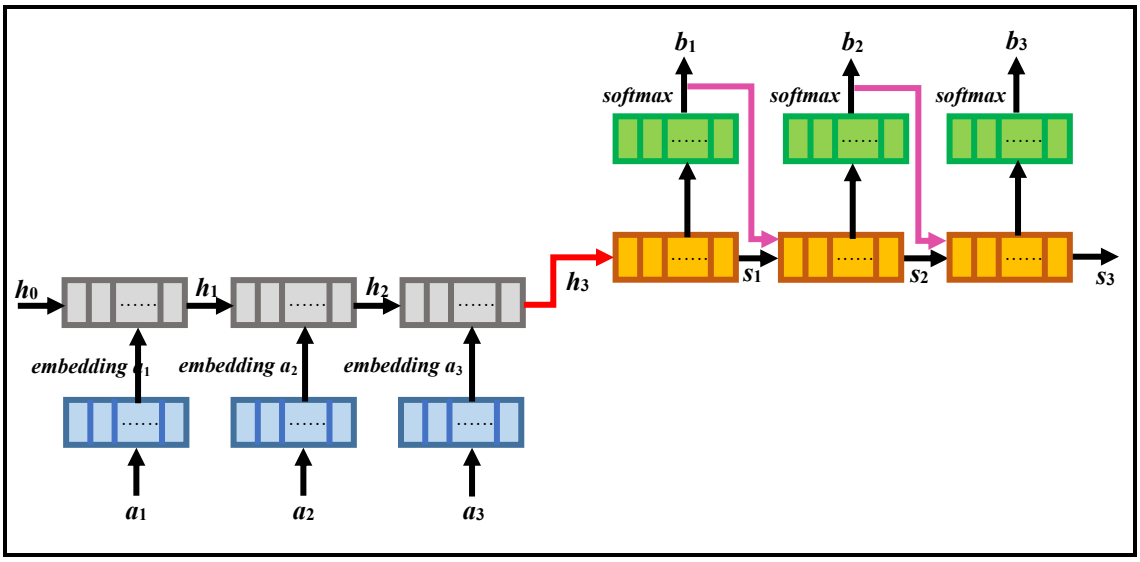


Figure 3.6: Sequence-to-sequence model representation for three time steps

In the decoder part, at time step *t*, each unit brings out an output. The encoder vector will be the hidden state of the first unit, whereas the remaining units receive the hidden state from the previous unit. By using a softmax function to calculate the output, the probability for every token in the output vocabulary is obtained (Sutskever *et al.*, 2014).

Chapter Four
Attention in Natural Language Processing

4. Attention in Natural Language Processing

In numerous issues that include the dealing with natural languages, the components composing the source content are characterized by having each a diverse significance to the assignment at hand. For occurrence, in aspect-based assumption investigation, cue words, such as “sad” or “happy,” can be significant to a few perspectives beneath thought, but not to others. In machine translation, a few words within the source content may be unessential within the translation of another word. In a visual question-answering assignment, foundation pixels may well be not important in replying to an address with respect to a protest within the closer view but pertinent to questions with respect to the scenery.

Seemingly, compelling arrangements to such issues ought to factor in an idea of significance, therefore as to center the computational resources on a confined set of imperative components. One possible approach would be to tailor arrangements to the specific genre at hand, in arrange to way better misuse known regularities of the input, by highlight designing. For case, in the argumentative examination of enticing expositions, one may decide to provide uncommon accentuation to the ultimate sentence. Nevertheless, such an approach is not continuously reasonable, particularly in case the input is long or exceptionally information-rich, such as in content summarization, where the yield is the condensed form of a conceivably long content arrangement. Another approach of expanding popularity amounts to machine learning the pertinence of input elements. In that way, neural designs seem consequently weigh the relevance of any locale of the input and consider such a weight while performing the most task. The prevailing solution to this issue is a mechanism that impressively solve the problem, known as “Attention” (Galassi *et al.*, 2020).

4.1 What is Attention

In English Language, when we say “attention” to someone, we mean that he has to pay his alertness on specific finding. The same purpose of attention mechanism is used in ML techniques. The rationale of this operation is to permit the generator at each cycle to consider all encoder covered up states whereas paying more consideration to the foremost pertinent tokens. In other words, it replaces the set of covered up states of the encoder by the set of weighted averages of these vectors. The algorithm of attention is the following:

1- Outline linearly the contextualized embedding vectors into the set of vectors that called queries, keys, and values.
2- Apply the attention score function to the keys and inquiries to calculate the attention dispersion. The dissemination is standardized by the softmax function.
3- Utilizing attention distribution to calculate the context vector as a weighted entirety of the values.

Equation 4.1 presents the matrix form of the algorithm:

$$Attention(Q; K; V) = softmax(score(K;Q))V \quad (4.1)$$

The difference in attention types depends on the type of score-function used. That is, by changing the formula of scoring function, the attention type is differ from the others, and makes a new technique, which results a different procedures and values. Table 4.1 illustrates different attention score functions. When we compare the use of these different types of score functions, we can see that dot-product is outperforming the other functions in terms of space and time efficiency. The most used types of them are dot-product function and additive function.

Table 4.1: Different score functions for attention

Attention	Score function	Source
General	$score(k_i, q_i) = k_i^T \, Wa \; q_i$	(Luong *et al.*, 2015)
Additive	$score(k_i, q_i) = \beta_a^T \; tanh \; (Wa \; [k_i; \; q_i])$	(Bahdanau *et al.*, 2014)
Content-base	$score(k_i, q_i) = cosine \; [k_i; \; q_i]$	(Graves *et al.*, 2014)
Dot-Product	$score(k_i, q_i) = k_i^T \; q_i$	(Luong *et al.*, 2015)
Scaled Dot-Product	$score(k_i, q_i) = \frac{k_i \; q_i}{\sqrt{d_k}}$	(Vaswani *et al.*, 2017)
Concatenate	$score(k_i \; , q_i) = \; v_a^T \; \tanh \; (W_a[k_i \; ; \; q_i])$	(Luong *et al.*, 2015)

4.2 Attention Mechanisms

When we are talking about attention in NLP, we notice that there is an initial exploration by variety of influential papers (Bahdanau *et al.*, 2014), (Sukhbaatar *et al.*, 2015), a rapid improvement of new attention models and efficient designs resulted, coming about in a profoundly broadened structural scene. Moreover, many types of attentions are used in different names despite of that they are the same. For example, the concepts of inner attention (Wang *et al.*, 2016) and word attention (Wu *et al.*, 2018) are ostensibly the same. Clearly, distinctive researchers to characterize diverse concepts, hence assist including to the uncertainty within the writing have presented the same terms. For illustration, the term context vector is utilized with various meanings by Bahdanau *et al.* (2014), Yang *et al.* (2016) and Wang *et al.* (2019). So, in the following paragraphs, the effective and well-known attention types will be explained.

4.2.1 Additive Attention

As we explained in chapter three, the encoder of neural network scans and encodes a source sentence into a fixed-length vector. A decoder at that point yields an interpretation from the encoded vector. The total encoder–decoder framework, which comprises of the encoder and the decoder for a dialect combine, is mutually prepared to maximize the likelihood of a redress interpretation given a source sentence. A potential issue with this encoder–decoder approach is that a neural network ought to be able to compress all the needed data of a source sentence into a fixed-length vector. This may make it troublesome for the neural network to manage with long sentences, particularly those that are longer than the sentences within the preparing corpus (Cho *et al.*, 2014). This was the motive for Bahdanau *et al.* (2014) in their well-known paper to suggest the using of attention mechanism in NLP, and to be the first who introduced an attention approach for manipulating NLP.

The idea of this attention is to enhance the machine translation of sequence-to-sequence model by implementing attention through aligning the decoder with the relevant input sentences. Figure 4.1 illustrates the idea of additive attention.

In this model, the conditional probability is defined as we can see in Equation 4.1:

$$p(y_t \mid y_1 , \dots , y_{t-1} , X) = g(y_{t-1} , s_t , c_t) \tag{4.1}$$

Here s_t represents an RNN hidden state for time, which is calculated as follows:

$$s_t = f(s_{t-1} , y_{t-1} , c_t) \tag{4.2}$$

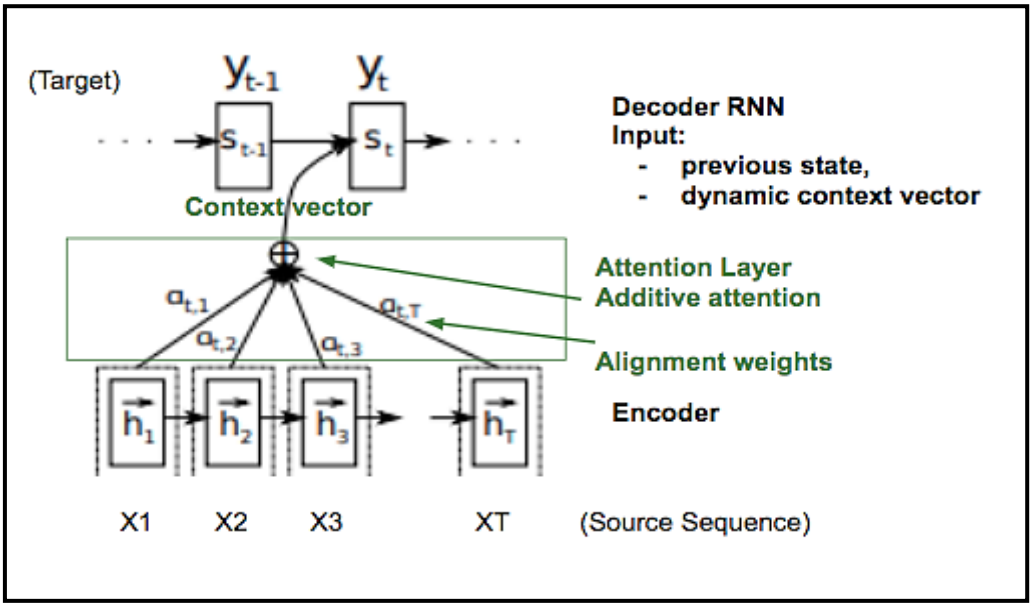


Figure 4.1: Additive Attention or Bahdanau Attention (Bahdanau *et al.*, 2014)

To be noted, the probability in the previous mentioned equations is conditioned on a unique vector c_t for every target word y_t. Which is differ from the encoder-decoder existing approach. The context vector c_t relies on a series of annotations $(h_1 , \dots , h_{Tx})$ to which an encoder connects to the input sentence. For each h_j annotation information about all the input sequence is collected, with a concentrate focusing around the t-th word of the input sequence. A weighted sum of the annotations h_j is calculated to generate the context vector as illustrated in Equation 4.3.

$$c_t = \sum_{j=1}^{T_x} a_{tj}\, h_j \tag{4.3}$$

For each annotation h_j the weight a_{tj} is calculated using Equation 4.4:

$$a_{tj} = \frac{\exp\,(e_{tj})}{\sum_{k=1}^{T_x} \exp\,(e_{tk})} \tag{4.4}$$

where:

$$e_{tj} = a(s_{t-1} , h_j) \tag{4.5}$$

Equation 4.5 represents the previous mentioned alignment model that presents how close the output at position *t* to the inputs around position *j*. As we can see from equation 4.5,

the score value is relying on the hidden state (S_{t-1}) of the RNN, and the (h_j) of the input sequence.

4.2.2 Global Attention

The key idea of the global attention model is to keep all the hidden states of the encoder to build the context vector c_t. The mutable-length alignment vector a_t, which have the size of time steps of the source, is calculated by taking a comparison between each source hidden state $\bar{h}_s$ and the current hidden state of the target h_t:

$$a_t(s) = align(h_t, \bar{h}_s) = \frac{\exp(score(h_t, \bar{h}_s))}{\sum_{\acute{s}} \exp(score(h_t, \bar{h}_{\acute{s}}))} \tag{4.6}$$

For score function, a "content-based" function is used with different three choices. Equation 4.7 presents these choices:

$$score(h_t, \bar{h}_s) = \begin{cases} h_t^T \bar{h}_s & dot \\ h_t^T W_a \bar{h}_s & general \\ v_a^T \tanh(W_a[h_t ; \bar{h}_s]) & concatenate \end{cases} \tag{4.7}$$

To complete the needed attention mechanism for effective processing in NLP, the "location-based" function was suggested. The target hidden states h_t are used to calculate the alignment scores. Location-based function is illustrated in equation 4.8:

$$a_t = softmax(W_a h_t) \tag{4.8}$$

By considering the alignment vector as weights, the weighted average over all the input hidden states is calculated to fetch the context vector c_t.

To differentiate the global attention from the previous mentioned additive attention, there are some specifications for the global one. Firstly, the global attention simplifies the use of hidden states by using them in both encoder and decoder LSTM top layers. Whereas, in additive attention the bi-directional backward with forward hidden states of the encoder's source was combined with decoder unidirectional non-stacking hidden states.

Secondly, the path for attention model calculations in global attention is easier to be done; the path for calculations simply goes from h_t to a_t then to c_t and finally to $\tilde{h}_t$. Then the prediction is made as mentioned in the previous lines. On the other hand, the calculation in additive attention goes from h_{t-1} to a_t then to c_t after that to h_t, to continue to max-out layer and deep output layer. Last of all, additive attention just using one alignment function, whereas global attention uses three different functions, which achieved more promised results (Luong *et al.*, 2015). The global attention model is illustrated in Figure 4.2.

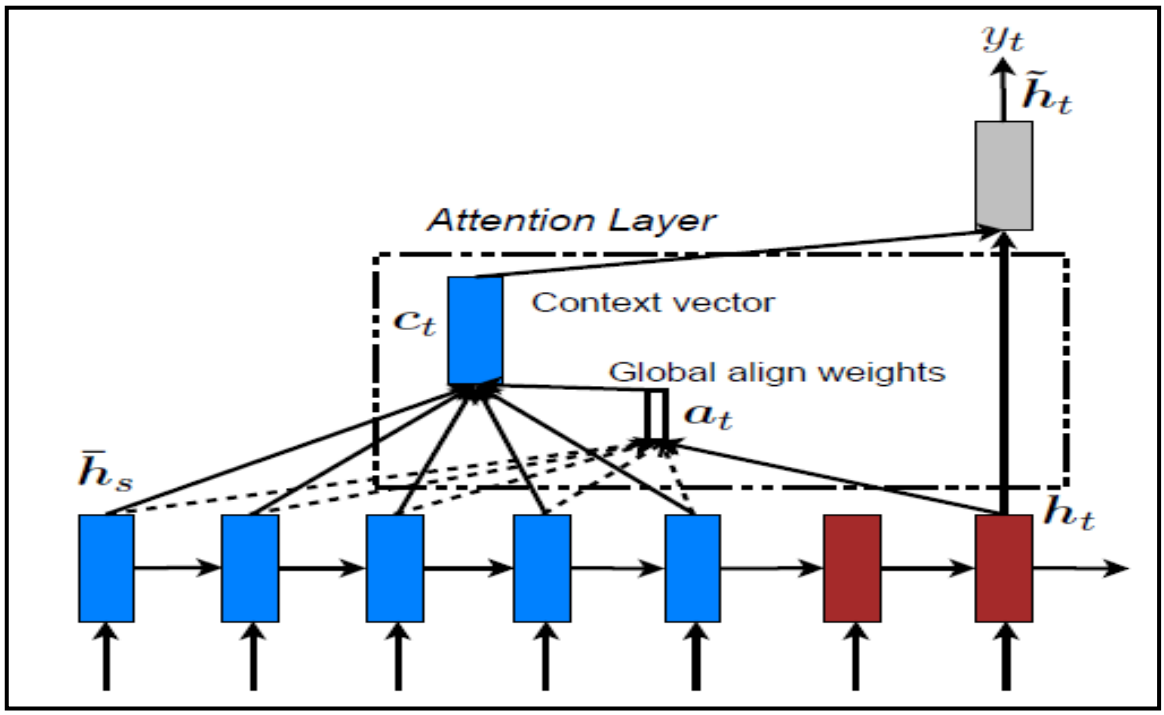


Figure 4.2: Global Attention (Luong *et al.*, 2015)

4.2.3 Local Attention

The global attention features a downside that it needs to go to all words on the source side for each target word, which is costly and can possibly render it unreasonable to decipher longer groupings such as paragraphs or documents. To address this lack, the local attention mechanism is proposed. In local attention, the focusing is just centered on subset of the source positions per target word. The technique used in local attention is to centers on a little window of context and it must be differentiable. This approach has an advantage of maintaining a strategic distance from the costly computation caused within the soft attention and, at the same time, is less demanding to train than the hard attention model.

In details, an aligned position p_t is firstly generated by the model at time *t* for each target word. By using a window of a set of hidden states, a weighted average over the set is calculated to provide the context vector c_t. The window used is: $[p_t - D\ , p_t + D]$, where *D* is experimentally selected (Luong *et al.*, 2015). In this model, the fixed-length vector a_t is proposed, and a different two sub-types are considered. The first one is called "monotonic alignment" or "local-m", and the second one is "predictive alignment" or "local-p".

In local-m, the input and output sequences are considered to be monotonically aligned by setting ($p_t = t$). The Equation 4.6 is used to define the alignment vector a_t. But in local-p, the aligned position is not considered as in local-m. The following equation (Equation 4.9) illustrates how the model predicts the aligned position:

$$p_t = S.sigmoid(v_p^T \tanh(W_p h_t)) \tag{4.9}$$

Where, v_p and W_p are two learned parameters of the model, which used to predict positions. The input sentence length is presented using S. To force the alignment points

to be near p_t, a Gaussian distribution is used, and it was centered around p_t. So, the alignment weights are calculated using Equation 4.10:

$$a_t(s) = align(h_t, \bar{h}_s)\exp\left(-\frac{(s-p_t)^2}{2\sigma^2}\right) \quad (4.10)$$

The alignment function of Equation 4.6 is used as the alignment function of Equation 4.10. By looking at Equation 4.10, we can notice that *s* is an integer, whereas p_t is a real number. The standard deviation (σ) is set to be equals to ($\frac{D}{2}$). Figure 4.3 illustrates the local attention model.

In the next lines, the idea of attention mechanism is used in more effective manners depending on other types of attention by combining one or more type in an innovative structure, or by using attention in practical innovatory designs.

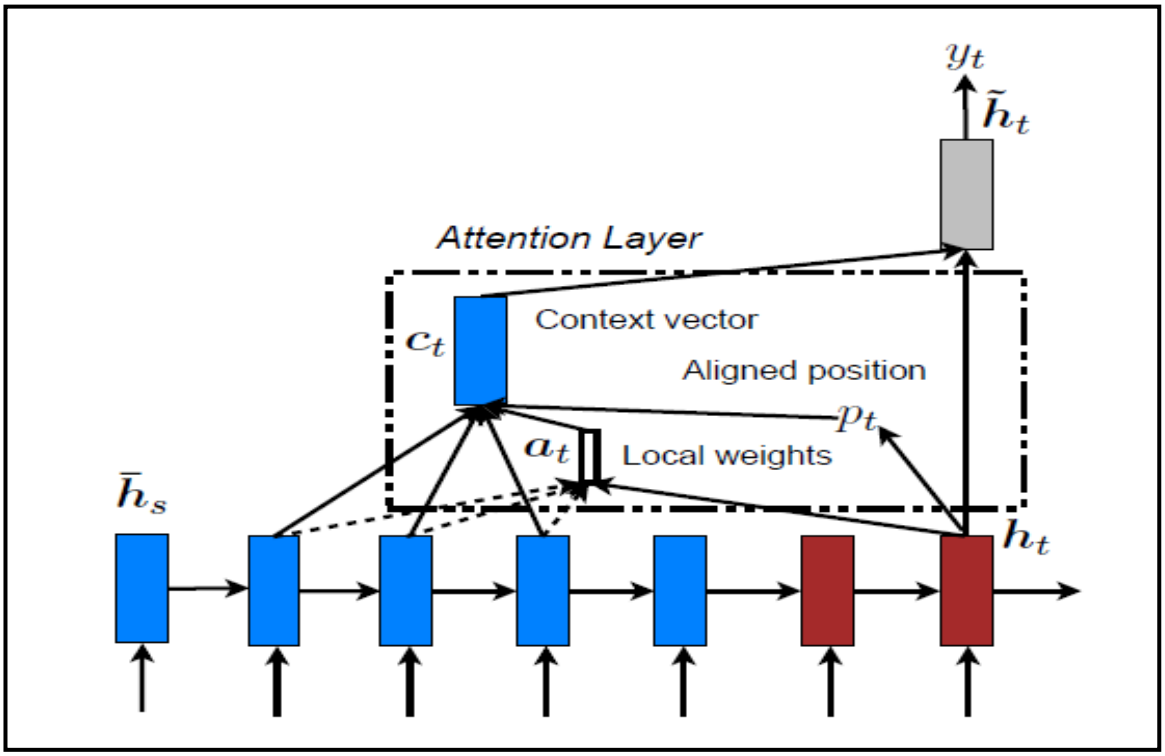


Figure 4.3: Local Attention (Luong *et al.*, 2015)

4.3 Transformer

In the year of 2017, an important turning point took place in the field of sequence-manipulated models by introducing the “Transformer” model. Encoder-decoder models were used in NLP, machine translation and any other sequence-to-sequence problems. The idea of attention was used in the encoder-decoder models to overcome the lack of these models to process longer sequences or to find dependencies between spatially separated elements in a sequence. But the long time taking in the execution and the extensively used resources in both encoder-decoder models and some of encoder-decoder models with attention leaded to produce transformer model. The main idea of the transformer is to use attention as a main component in the proposed model, the clear evidence for that is the title of the paper that presented the model; “Attention Is All You Need”. In the following lines the proposed model will be illustrated.

4.3.1 Transformer Architecture

The transformer architecture used the same main architecture of the models that deal with sequence-to-sequence problems, which is the encoder-decoder architecture. The difference here is that transformer model uses the attention mechanism as a base component to process the processed problems. Therefore, the next explanation will reflect the model’s architecture in an encoder-decoder manner.

4.3.1.1 Encoder Stack

In the transformer base model, the encoder has six identical layers (N=6), each layer composed of two sub-layers. The first layer is a multi-head self-attention, and the second one is a position wise feed-forward fully connected network layer. Each of the previously mentioned layer is followed by normalization layer. The output’s dimension of each sub-layer (d_{model}) is set to 512 to facilitate the residual connections. As we can see, the basic

architecture in the encoder is the attention mechanism. Figure 4.6 illustrates the encoder part in the left part of the model.

4.3.1.2 Decoder Stack

In the decoder stack, we can notice that it is the same as encoder stack of number of stacks which is six too. The layers are the same with some differences. The two sub-layers presented in the encoder are presented in the decoder too. There is a third sub-layer called masked multi-head attention which is multi-head attention sub-layer over the output of the encoder stack. Another difference from the encoder stack is the self-attention sub-layer. This was modified to ensure that the output predictions gained from position *x* are dependent on just the positions less than *x*. This is done by making an offset by one position of the embedding's output sub-layer.

4.3.2 Attention in Transformer

Attention mechanism is being used in the transformer model in a way that ensuring the increase of model's efficiency and reducing the time execution, compared to other models. The main idea of the transformer is to parallelize the processing of input sequences to enhance the beneficial using of attention mechanism as well as dealing with large datasets. The next lines will explain the main effective components in the transformer model.

4.3.2.1 Multi-Head Attention

As a recent technique that receives the advantages of previous attention techniques, multi-head attention was proposed by Vaswani *et al.* (2017). This technique played an important role in NLP and improved the output results, if it compared to the existing results, in the same field. In more details, this model uses the idea of linearly calculating keys (d_k), queries (d_k) and values (d_v) n-times, leaving the previous strategy that used a function for single attention mechanism. For each different step of linear calculation, a parallel

process of the attention function takes an action to produce different versions of d_v; which are dimensional scaled dot-product attentions. The multi-head attention output is gained through concatenating the attention values of the attention function for each step mentioned previously. The following two equations, reflect the procedure of this technique:

$$\text{Attention}(Q,K,V) = softmax\left(\frac{Q\,K^T}{\sqrt{d_k}}\right)V \tag{4.11}$$

$$MultiHead(Q;K;\,V\,) = Concat(head_1;\ldots;\,head_n)W^O \tag{4.12}$$

where $head_i = Attention(QW_i^Q\,,\ KW_i^K\,,\ VW_i^V)$, and $W_i^O, W_i^Q, W_i^K, W_i^V$ are matrices.

To grasp the idea of multi-head attention as simple as possible, considering that we have scaled dot-product attention, which represented in Equation 4.11, and by using multiple of this attention in parallel with number of layers equals the number of paralleled attentions, we finally have what we called “multi-head attention”. Figure 4.4 illustrates the scaled dot-product attention as explained in the original paper, and Figure 4.5 explains the idea of doubling the aforementioned attention mechanism and how it is used in parallel manner. The multi-head attention is the base idea for the transformer model that will be explained in the next section.

4.3.2.2 Multi-Level Attention

Clearly from its name, we can conclude that there are multi-levels of attention used. To demonstrate the idea behind this technique, let us assume that we have two-level attention mechanism. The attention values of keys (k) and query (q) are calculated using Equation 4.11. For simplification, we will assume the attention is calculated as $Attention(q,k,k)$. The output of this attention will have the same dimension of the query, which gives level number one. In level number two, the output of level number one is treated as a new query.

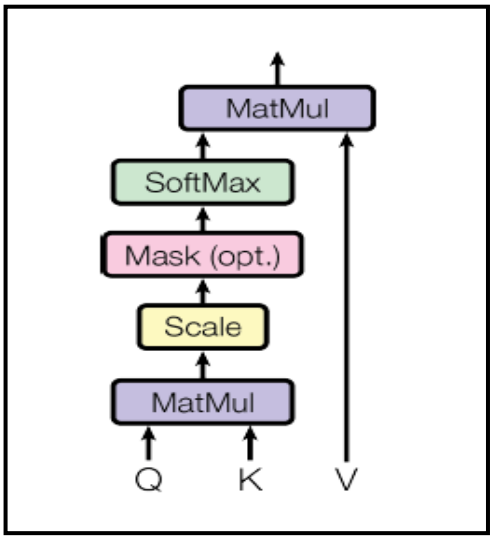


Figure 4.4: Scaled Dot-Product Attention (Vaswani *et al.*, 2017)

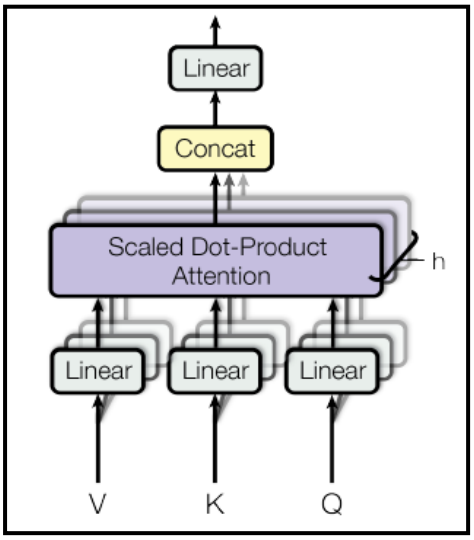


Figure 4.5: Multi-Head Attention (Vaswani *et al.*, 2017)

The attention value is calculated for the input sequence or (k) again. The result of this procedure is: $Attention(Attention(q, k, k), k, k)$. When the second attention finishes its calculations, the higher level of internal features of the words of the input text is became learned.

Quotes from this idea, Vaswani *et al.* (2017) duplicated the attention mechanism (m-times) over the source sequence, aiming to learn their proposed model the higher-level features. In this model, the (m-level) attention technique makes the source sequences to change level by level, to be more suitable for extracting more features and more acceptable as a decoder input sequence. Therefore, the aforementioned model is known as "Transformer Model". Figure 4.6 illustrates the building mechanism of the transformer.

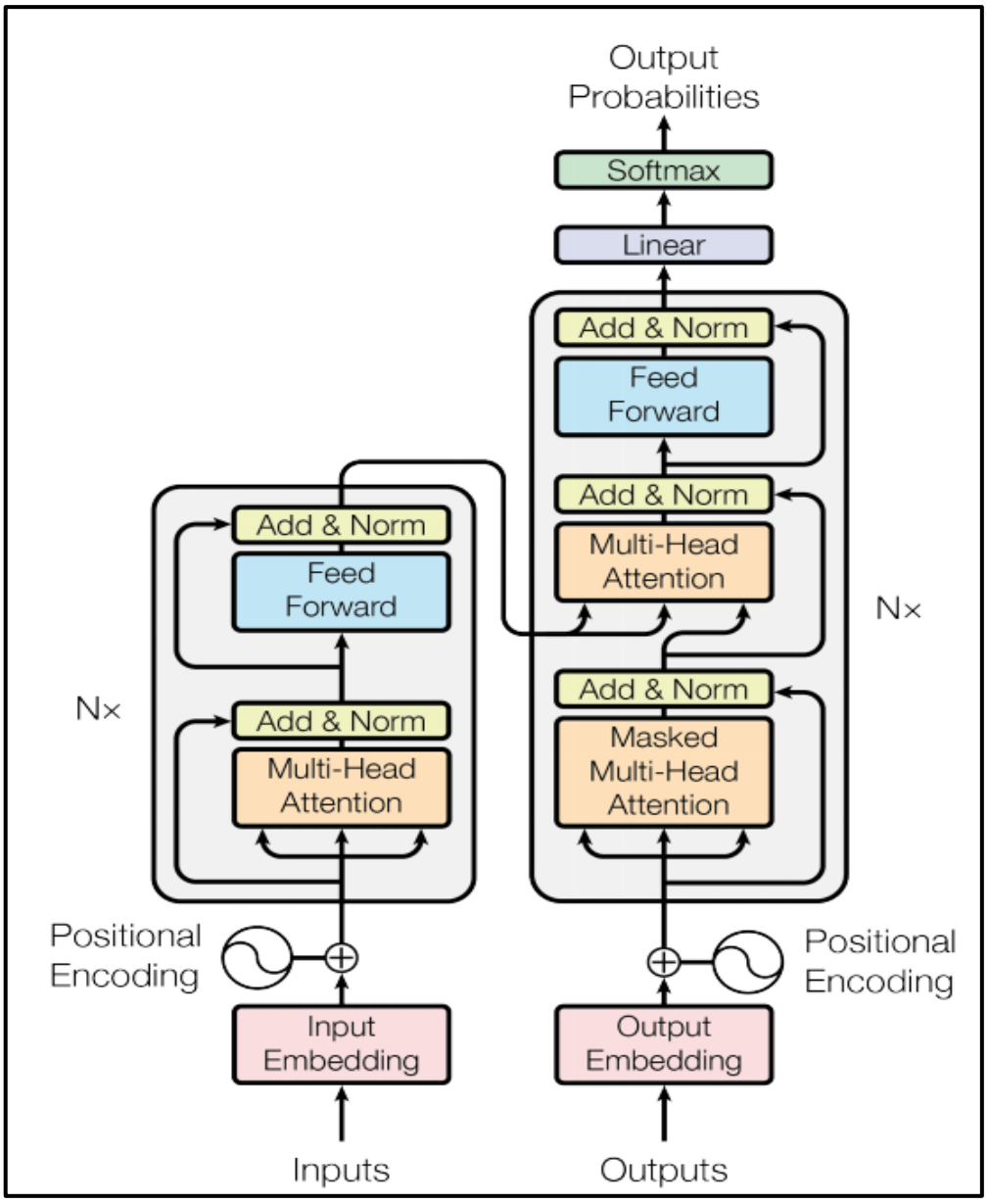


Figure 4.6: Transformer-model architecture (Vaswani *et al.*, 2017).

Chapter Five
Methodology

5. Methodology

In this chapter the used environments, that were the places where the programs' codes executed, will be explained. Then the datasets that have been used in the different experiments will be illustrated. Finally, the training methodology, which used for both diacritizing Arabic text and CSMs will be clarified in detail with the different executed models.

5.1 Experimental Environment

In the Experimental Environment's section, we firstly explain the computational resources, and then we illustrate the programming environment that has been used for executing the experiments. The following sub-sections represent them.

5.1.1 Computational Resources

The recent deep learning methods and the proposed ML models require high specification computation resources to be executed. The ordinary CPUs cannot afford this demand to these models. Therefore, the idea of using machines that are more powerful to manage these models is emerged. GPUs are the solution for this obstacle. To use an acceptable environment for training ML model, you need a machine with one or more GPUs and acceptable amount of RAM. This depends on the model's architecture and the amount of data to be trained on.

In our experiments, which concern in solving diacritization and CSM problems, we started to use a server with (Intel(R) Xeon(R) – CPU) and 32 GB of RAM. Unfortunately, this server with its specifications needed more than three days to execute the first proposed model, and sometimes it halts the execution. Therefore, we researched for a solution for this issue. The solution was to use platforms that offer powerful computational resources for using CPUs and GPUs, and surely can execute ML program

codes. These resources are “Google Colab (Google Collaboratory, 2017) and Kaggle (Kaggle, 2014)” kernels.

Both environments provide the ability for the user to benefit from the cloud computational resources for executing and training ML models. These environments are costly free for executing various ML codes with the provided resources of CPUs, GPUs, RAMs, and storage space. The affordable free resources were not enough to execute the different models we used in our work, therefore, needed to use (Google Colab Pro+ account) with more powerful resources. In this account the available used GPUs are different. We stick to the best affordable GPU (Tesla V100-SXM2) to execute our models. Table 5.1 represent the computational resources’ specifications that we used in our experiments.

5.1.2 Programming Environment

One of the most used programming languages in ML is Python. It gives the ability, with various libraries, to build diverse models in different ML sectors. The ease of use and powerful programming capabilities made us to use it in our work. The python version used is 3.7.10. To use a powerful open-source software library that increase the ability to execute our proposed models, TensorFlow (TF) library were used. TF, which based on differential programming and dataflow, is a symbolic math library became a base for many of ML models. TF is used in the models we proposed as a main library in different programs’ codes. The version of TF used is 2.4.1.

There are many other libraries used, but each program code uses part of them or different types of them depending on the architecture of each model. The idea of using any library is to befit from the built-in functions that are used to process the analyzed data. As it is hard to mention each of these libraries, we mentioned the main library which is a basic one for building the models.

Table 5.1: Specifications for computational resources

Parameter	**specification**
GPU	Tesla V100-SXM2
Available RAM	50 GB
Disk Space	147 GB
Max execution time	24 hours
Max idle time	1.5 hours

5.2 Datasets

The data sets that have been used to investigate the different solutions or approaches for solving CSM and diacritization problems are:

(a) Linguistic Data Consortium's (LDC) Arabic Treebank (LDC2010T08) or more precisely, LDC's Arabic Treebank Part 3 (ATB3) v3.2

This dataset is a newswire stories from An Nahar Lebanese publication. It is composed of 599 distinct stories, and it is a sample for Modern Standard Arabic (MSA) (Zitouni *et al.*, 2006). This dataset is used for training the model and then for evaluating and testing it. The last 90 newswire stories were used for evaluating and testing, whereas the previous 509 stories, which are chronologically ordered, were used to train the suggested models. In other words, 3,857 is the number of used sequences evaluation, and the training was on 22,170 sequences (Abandah and Abdel-Karim, 2019).

(b) Tashkeela dataset

This dataset is an example of Classical Arabic (CA), it contains text that represents samples of CA and Holy Quran. This modified dataset is composed of 55k lines chose from the original dataset after removing the file formatting errors, and fixing diacritization problems (Fadel *et al.*, 2019).

Tashkeela dataset is divided in to 2,500 lines for validation, and another 2,500 lines for testing. Before that, a part of 50,000 lines is used for training the different models (Abandah and Abdel-Karim, 2019). Table 5.2 summarizes the different characteristics of the two datasets.

5.3 Training Methodology

In the training methodology, we firstly illustrate how the data sets used in the experiments were prepared. In the second section, the description of the used models is presented. The following lines will present these two sections.

Table 5.2: Datasets characteristics (Abandah and Abdel-Karim, 2019)

Dataset	Word Count	Sequence Count	Letter per Word	Word per Sequence
LDC ATP3	305 K	26,027	4.6	11.3
Tashkeela	2,312 K	55 K	4.0	42.1

5.3.1 Dataset Preparation

In both used datasets (*LDC ATB3* and *Tashkeela*), we determine the maximum length of input text to be 100 characters. To achieve this, we process the used datasets to wrap the text when it is around 100 characters long. To be more obvious, the text wrapping will be applied for the text when we reach 100 character long followed by new line character. If this condition is not applied, a point for wrapping text is searched by; firstly, find a proper punctuation mark (for instance, full stop or comma). Or secondly, finding any punctuation mark. Or lastly, break at space. This criterion is applied within 50 characters after the first 100 character. After applying the aforementioned process, the number of samples in the *LDC ATB3* dataset used for training and validation reached 31886 samples. The maximum sequence length for inputs was 100 and the maximum sequence length for outputs is 175 in the diacritization problem. For CSM problem, the number was 101 and 100 for the maximum sequence length for inputs and the maximum sequence length for outputs respectively. Whereas, in *Tashkeela* dataset, 170,795 samples are used for training and validation, and 8,228 samples for testing. In the diacritization problem when using *Tashkeela* dataset, the maximum sequence length for inputs was 104 and the maximum sequence length for outputs was 192. For CSM problem, it was 106 for both maximum sequence input length and maximum sequence output length. It is important to mention that in the *LDC ATB3* dataset, the validation set is also used for testing which is (0.149) of the 31,886 samples.

The dataset preparation methodology applied on diacritizing Arabic text differ from that was applied on CSM. Each type of these two problems has its own method to process the available datasets (*LDC ATB3* and *Tashkeela*). For diacritization, the dataset is preprocessed using a code, which removes the diacritics from the source data. Then it builds the new dataset in pairs that consists of the undiacritized-first part of the pair with

its diacritized-second part. This dataset is used for training the proposed models on the diacritization problem.

In CSM problem, the process was more complicated. Firstly, we preprocessed the dataset by removing the diacritics, and then we made the erroneous words by replacing one correct character for each word with a wrong one. We applied the process of creating erroneous words on (60%) of the input sequence. The new dataset created is used as a training dataset for the proposed models. Because the number of errors in Arabic language is large, the errors applied to the input text are illustrated in Table 5.3.

As we can see from the table, the errors applied are the commonly occurring errors in Arabic text of native Arabic speakers and non-native Arabic speakers. These errors are applied randomly on the dataset to reflect the quality of proposed models in solving such problems. Necessary to add that the use of these “errors applied” considered as a sample of errors, which means that we can use another error types and the idea will be the same. We just use this sample of errors to test the models and reflect the results gained from each model. Therefore, we can ensure that by using other types of errors, the results will be close to the results we achieved.

As we can see from the table, the errors applied are the commonly occurring errors in Arabic text of native Arabic speakers and non-native Arabic speakers. These errors are applied randomly on the dataset to reflect the quality of proposed models in solving such problems. Necessary to add that the use of these “errors applied” considered as a sample of errors, which means that we can use another error types and the idea will be the same. We just use this sample of errors to test the models and reflect the results gained from each model. Therefore, we can ensure that by using other types of errors, the results will be close to the results we achieved.

Table 5.3: Added spelling mistakes

No.	Original	Replacement as mistake
1	Demonstrated pronouns (أدوات الإشارة) ''هذا'' ''هذه'' '' هؤلاء '' ''ذلك'' ''أولئك''	Adding (*Alef*: ا) ''ألف'' ''where it is pronounced but not written'' ''هاذا'' ''هاذه'' '' هاؤلاء '' ''ذالك'' ''أولائك''
2	(*Thaa:* الظاء) ''ظ''	(*Thaa:* الضاد) ''ض''
3	(*Taa Matbota*: تاء مربوطة) ''ة''	(*Taa Maftoha*: تاء مفتوحة) ''ت''
4	(*Hamza after Yaa*: الهمزة بعد الياء) ''يء''	(*Alif Maqsora with Hamza above*: همزة على ألف مقصورة) ''ىٔ''
5	(*Alif*: ألف) ''ا''	(*Alif Maqsora*: ألف مقصورة) ''ى''
6	(*Alif Maqsora*: ألف مقصورة) ''ى''	(*Alif*: ألف) ''ا''
7	(*Hamzat*: همزات) ''أ'' ''إ'' ''ء'' ''ئ'' ''ؤ''	Changed Randomly

5.3.2 Models Description

As previously mentioned, the models proposed in this work are firstly used to investigate diacritization problem and tackling it. Then the same models are used to solve the problem of CSM. In the next lines, each suggested model would be described with its hyper-parameters. The general backgrounds and different mechanisms used in the models were discussed in the previous chapters. In this section, we present the different models' hierarchy and technical points of each model as well as the hyper-parameters determined for each model.

5.3.2.1 Encoder-Decoder-2L Model

The first proposed model, which we suggested, will be the base model to the other suggested models. Therefore, that we can compare the other models with it. The reason to choose this model is that it represents a basic model's architecture that is; it consists of an encoder-decoder hierarchy. After number of trails to determine the general specifications of this model, we decided to make a "two- LSTM layer encoder-decoder" model. The reason of that is, with the different applied experiments on the datasets used,

the model with two-layer achieved the best results compared with the model when changing the number of layers other than two. Necessary to add that, the increasing of the number of layers may affect the models in a positive way, but in our model adding extra layers, increase the resources consumption without any good improvements.

The suggested Encoder-Decoder-2L model starts with input_1 layer, the output of this layer is the input of a masking layer that used to mask the input which in turn be skipped in all downstream layers. The encoder_Lstm_1 layer is following the masked layer to provide its output for both encoder_Lstm _2 and decoder_Lstm _1 layers. Input_2 layer will moreover supply the decoder_Lstm _1 layer with the input values. The output of both decoder_Lstm _1 and encoder_Lstm _2 layers will be fed to the decoder_Lstm _2 layer, which in turn supply the dense layer with its output. Lastly, the output of dense layer represents the final output of this model. Figure 5.1 represents the model structure.

The model hyper-parameters are as follows:

- Architecture: Encoder-Decoder Model
- Number of layers: 6
- Batch size for training: 64
- Latent dimensionality of the encoding space: 256
- Dropout rate: 0.2

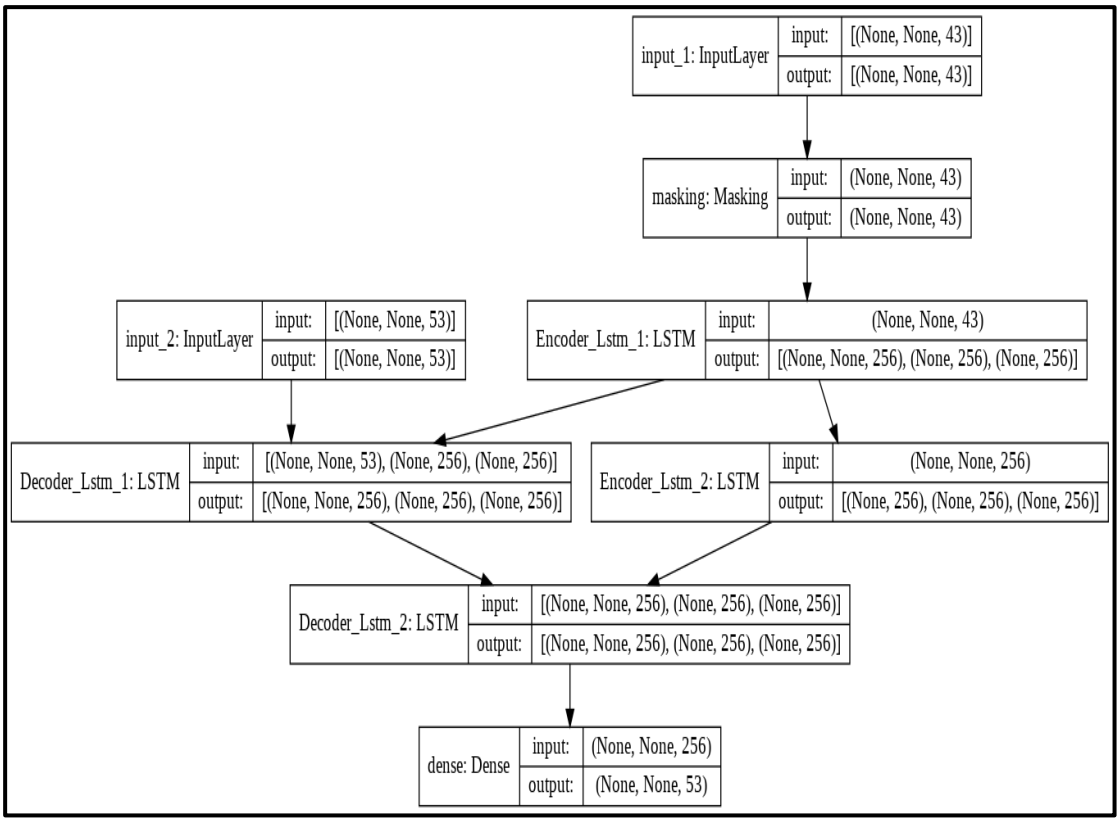


Figure 5.1: Encoder-Decoder-2L Model Architecture

5.3.2.2 Encoder-Decoder-2L-Attention Model

The second proposed model used attention mechanism, which explained in Section 4. The model needs to direct its attention to the diacritics in first problem or mistaken word in the second problem. Therefore, an attention layer is added to provide the model increased ability to deal with such problems. The attention layer is added between decoder_Lstm_1 layer and decoder_Lstm_2 layer while the remaining layers remain the same as in the Encoder-decoder-2L Model. The added attention layer increased solely the accuracy, but it did not provide the needed results. The attention type used in this model is additive attention that explained in Section 4.2.1. The hyper-parameters for this model are the same as the base model except for the number of layers, which became seven layers. Figure 5.2 illustrates the second model architecture.

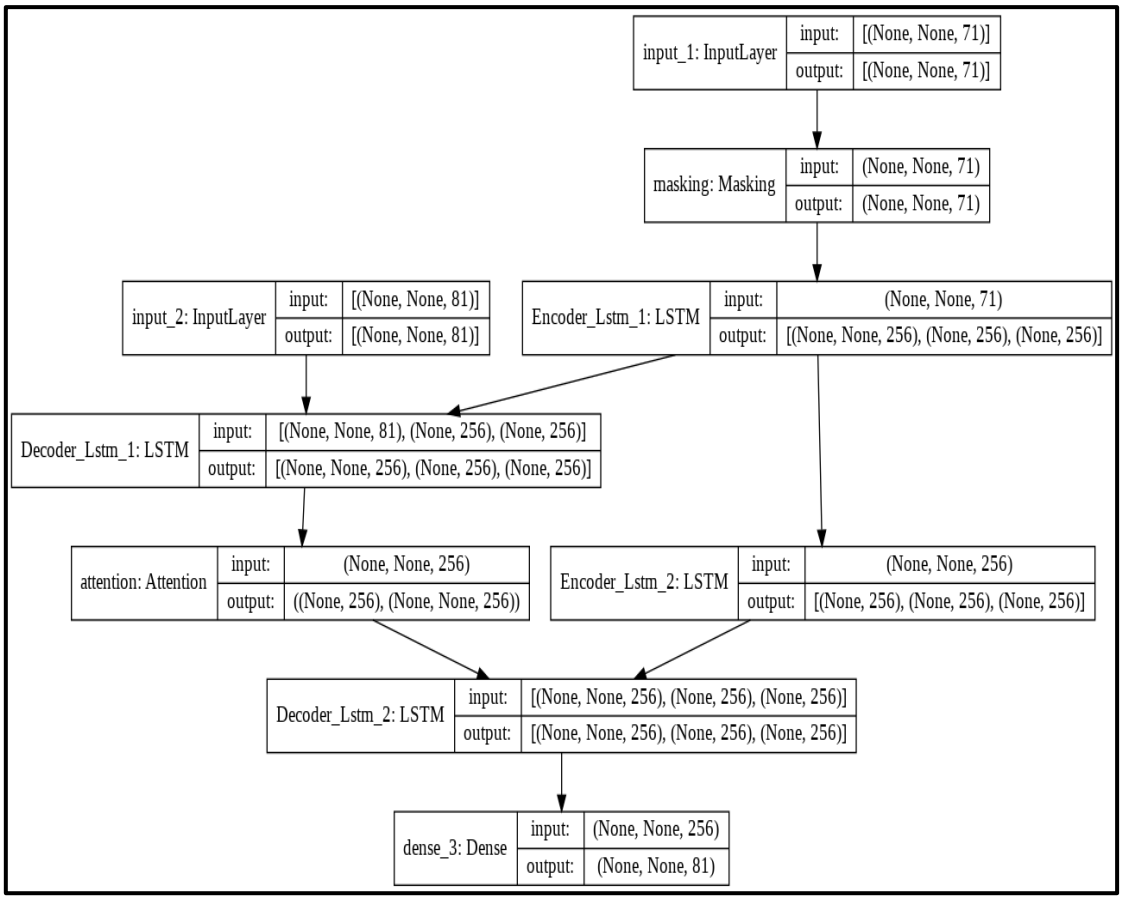


Figure 5.2: Encoder-Decoder-2L-Attention Model Architecture

5.3.2.3 Transformer Model

The third tested model used for solving diacritizing Arabic text and CSM is the transformer model that explained previously in Section 4.3. As we mentioned, the attention mechanism is used as a base architecture in the model. There are two types of attention used: Scaled dot-product attention and multi-head attention. Figure 4.6 illustrates the Transformer Model architecture. To solve the problem of focusing the attention on to specific characteristics regardless the others, the multi-head attention layer is used to capture the important features of the used dataset in the model, and as a result understanding the dataset properties. In our work, the Transformer Model had the ability to grasp the diacritization problem and the model had the ability to solve it. The same result gained when applying the CSM problem using the Transformer Model; that is, the model had the ability to understand the corpus used in the training phase and had

the ability to correct the errors in the erroneous used text. The results will be illustrated in the next chapter. Necessary to add that, Transformer Model has two types: Transformer Basic Model and Transformer Big Model. The model used in our experiments is the Transformer Basic Model, because it can provide good results with less computation resources. The reason behind not using Transformer Big Model is that it needs much more high resources' specifications, which will be hard to be available. Another reason is that the basic model achieved good results. The parameters and hyper-parameters of the used transformer model are as follows:

- Architecture: Transformer Base Model
- Number of layers: 6
- Batch size for training: 64
- Number of attention heads (H): 8
- Model dimensionality: 1,024
- Dropout rate: 0.1

5.3.2.4 E-D-Attention-Concatenation Model

In this model, the main hierarchy as encoder-decoder model is used but with different number of layers and with changing some types of layers. This model consists of nine layers; the embedding layer, three encoder -LSTM layers, one decoder-embedding layer, one decoder-LSTM layer, one attention layer, then a concatenation layer to combine the output of decoder-LSTM layer with the output of attention layer, and finally a time distributed dense layer to produce the desired output of this model. This model performed bad results compared to other models. Figure 5.3 represents the architecture of the model.

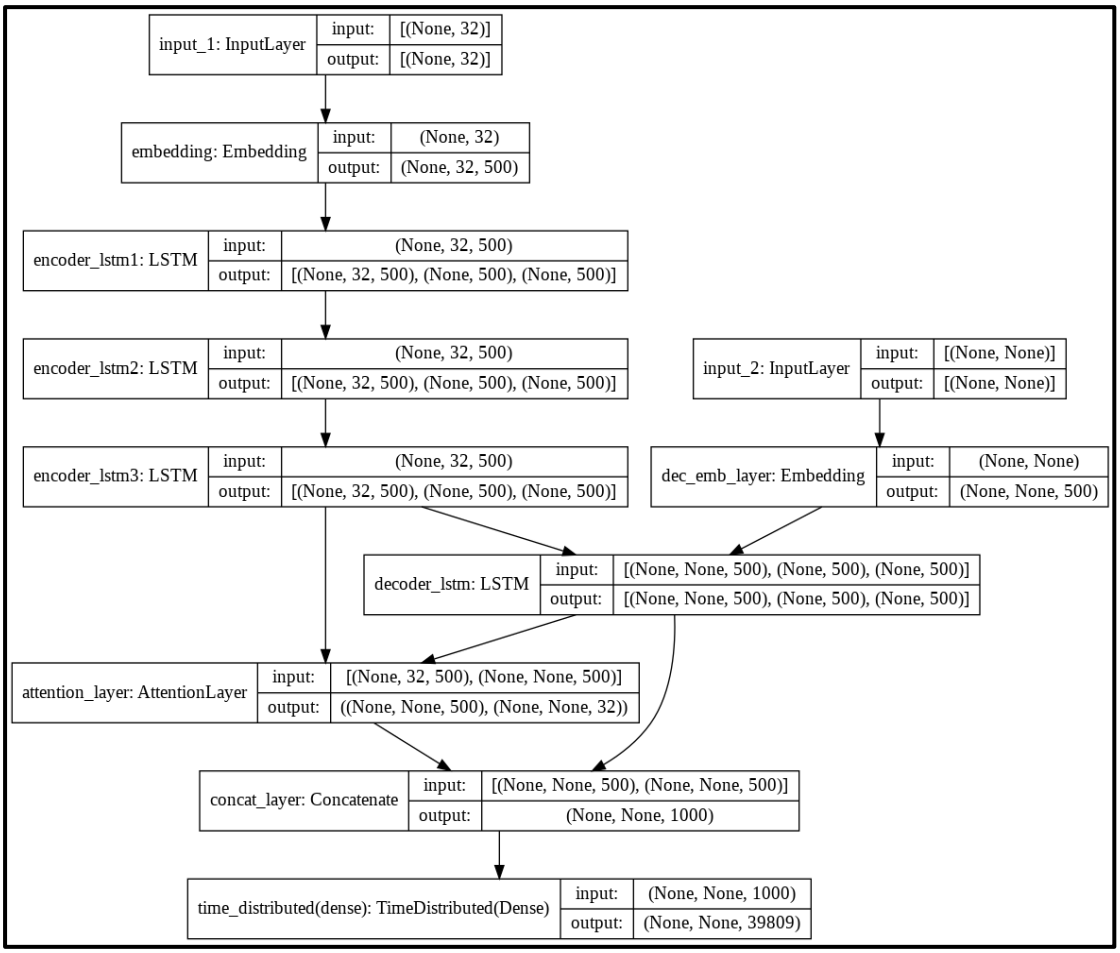


Figure 5.3: E-D-Attention-Concatenation Model Architecture

The parameters and hyper-parameters of this model are the following:

- Architecture: Encoder-Decoder with Attention Model
- Number of layers: 9
- Latent dimensionality of the encoding space: 500
- Batch size for training: 64
- Dropout rate: 0.2

5.3.2.5 E-D-Seq-to-Seq Model

This model is one of the most two models that achieved good results. The model takes the idea of neural machine translation to resolve the problems of both diacritization and CSM. The model basic architecture is encoder-decoder model. The layers used to represent this model are as follows: In the encoder, there are two layers; an encoder-

embedding layer and an encoder-LSTM layer. Whereas, in the decoder we have three layers; decoder-embedding layer, decoder-LSTM layer and decoder-dense layer. In this model, the embedding layers use a character embedding to create the embedding vectors for each word in the dataset used in the training process. The idea in this model, which made the model one of the best two models, is to use embedding layer in both encoder and decoder. The embedding layer role is to transform every word from text to a defined size of a fixed length vector. The produced vector is a dense vector, but it has real values instead of 1's and 0's. This representation of fixed-length word vectors reduced dimension and made the representation of words more understandable for the used model. If this layer presented in both encoder and decoder, the model would have the ability to learn the embedding for each word in the used corpus as well as to have the ability to understand the relations between the input and output sequences of the model. Using this idea in this model produced very good results. Figure 5.4 represents the proposed model architecture. The parameters and hyper-parameters assigned for this model are as follows:

- Architecture: Encoder-Decoder Model
- Number of layers: 5
- Embedding dimensionality: 128
- Hidden dimensionality: 1,024
- Batch size for training: 64

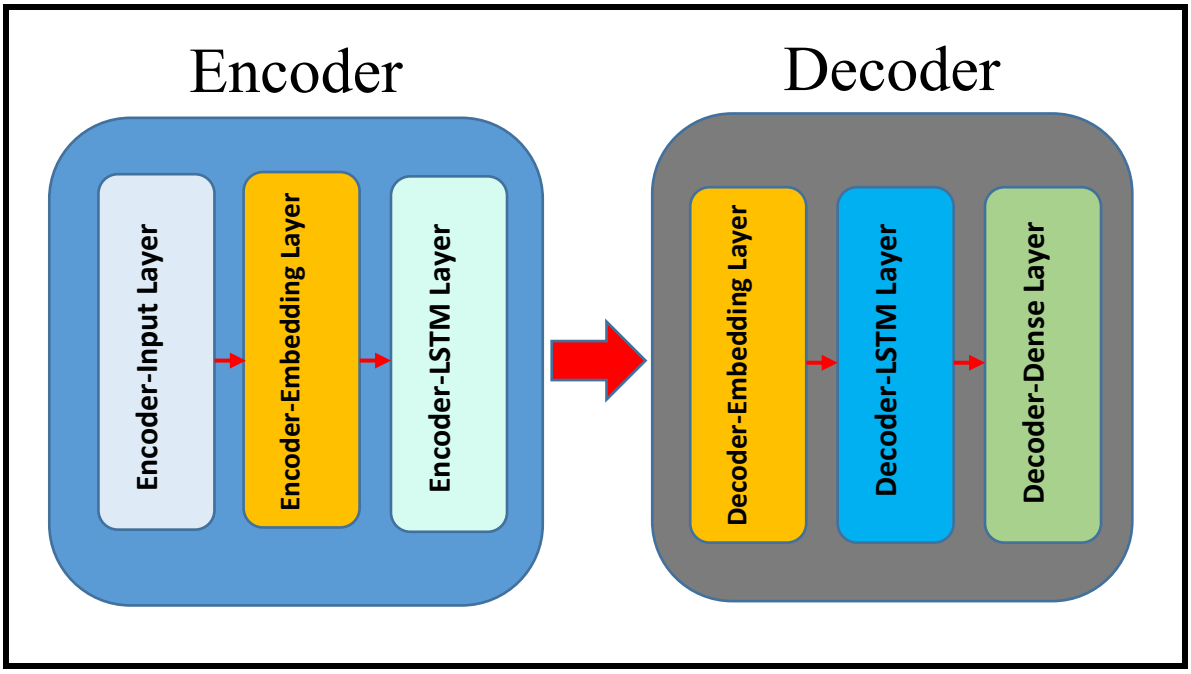


Figure 5.4: E-D-Seq-to-Seq Model Architecture

5.3.2.6 E-D-Seq-to-Seq- Attention Model

This model is the second one of the two proposed models that achieved the best results in solving the problems of diacritizing Arabic text and CSM. The idea of this model is to add attention layer, which applies attention mechanism, to E-D-Seq-to-Seq Model. As we mentioned before, the attention mechanism can add more concentration on specific features of the input text to determine the relationships between different types of these features, and as a result receiving the needed output. It is necessary to add that, the attention mechanism must be used correctly in a way that the model concentrate on the correct features, which increase the ability of the model to solve the problems. The attention mechanism used in this model is (Loung attention), which is illustrated in Section 4.2. This model, with E-D-Seq-to-Seq Model, achieved great results compared to other models. The results and discussion will be presented in the next chapter. The parameters and hyper-parameters of this model are the same as the E-D-Seq-to-Seq Model, except for the number of layers, which is six layers.

Chapter Six
Results and Discussion

6. Results and Discussion

In this chapter, the results of the proposed models with the different executed experiments will be explained. Firstly, the results of the diacritizing Arabic text will be presented, discussed, and evaluated. Then the results that belong to CSM problem will be illustrated too. As we mentioned before, the models that used to solve diacritization problem are used moreover to solve CSM problem. Therefore, the same models will be tested for the first solved problem with its results, and then with the second solved problem with its related results. Lastly, a comparison between the results gained from these two different problems will be held to clarify the different results collected from these models with the different problems.

6.1 Results and Discussion in Diacritization

For the diacritization problem, the two datasets are used in all models in the same procedure; that is, the undiacritized part of the corpus is the input of the model, and the diacritized part is the desired output text of the used model. In the following lines, every model will be discussed according to the results achieved by training the two datasets mentioned previously.

The first proposed model is the Encoder-Decoder-2L Model. We considered this model to be as the base model that we compared other models to it. After different experiments and tuning the hyper-parameters of this model, the results gained were not good results compared to other proposed models. In this model, the early stopping technique is used with patience of 5 epochs. This means that the training steps will wait for 5 epochs after the best gained monitored metric is achieved without any improvement; (in our early stopping technique, the validation accuracy is monitored). In our experiments with this model, we applied the training on 100 epochs, and the early stopping was in epoch number 60 in *LDC ATB3* dataset, whereas the early stopping in the *Tashkeela* dataset was in epoch

number 64. The time taken for training the *LDC ATB3* corpus was 1,358 seconds with average epoch execution time 23 seconds per epoch. In addition, for the *Tashkeela* dataset the average epoch execution time taken in seconds per epoch was 117 with total execution time of 7,504 seconds. Table 6.1 illustrates the results.

According to these results, the registered values are not considered acceptable for a model to solve such problems, especially the value of BLEU-score, which reflects that this model is not able to generate acceptable diacritized text. Depending on this, the work for improving this model will go forward by introducing the next model.

Table 6.1: Evaluation results of the models used to solve diacritization problem (bold numbers indicate the best results)

Model	Dataset	Loss	Accuracy	BLEU-score	Best Epoch	Execution time (seconds)	Average execution time (seconds/epoch)
Encoder-Decoder-2L	*LDC ATB3*	0.3130	0.3595	10.93	60	1,358	23
	Tashkeela	0.1417	0.5617	15.47	64	7,504	117
Transformer	*LDC ATB3*	**0.0137**	**0.9598**	**53.12**	249	887.6	4
	Tashkeela	**0.0098**	**0.9707**	**61.40**	284	4,260	15
E-D-Seq-to-Seq	*LDC ATB3*	0.0238	**0.9953**	**55.39**	25	2,250	90
	Tashkeela	0.0110	**0.9982**	**63.99**	85	16,065	189
E-D-Seq-to-Seq-Attention	*LDC ATB3*	0.0111	**0.9934**	**61.02**	18	2,070	115
	Tashkeela	0.0056	**0.9967**	**64.96**	85	19,380	228

The un-promised results obtained from the previous model, introduced the idea to present this model; “Encoder-Decoder-2L-Attention Model”. By adding the attention layer to the previous model, we can receive better performance, especially when dealing with long sequences. Bahdanau attention (Bahdanau *et al.,* 2014) is the attention mechanism that we used in this model. This attention performs good improvement to the models that are used to solve sequence-to-sequence problems, like the problem of diacritizing Arabic text. By tuning the hyper-parameters and executing many experiments, there was slightly improvement in the accuracy result in *LDC ATB3*. Nevertheless, in the *Tashkeela* corpus the results for both accuracy and loss were better than the results recorded for the *LDC ATB3* dataset. Unfortunately, the BLEU-score value was *zero* for both datasets, which is an indicator that, the attention mechanism worked in negative way. When we revised the code and the results, we noticed that the code is correct, but the results were very bad. We noticed that the model can concentrate its attention on the diacritics regardless the text itself. Therefore, the output gained was not understandable text. To be added, the technique of early stopping is moreover used to finish the execution when there is no improvement in the validation accuracy that is monitored by this technique. The results of this model will not be presented or compared with other models due to the bad results recorded.

The third tested model is the Transformer Model. In the *LDC ATB3* corpus, it took 249 epochs to train the model to reach the stability state of the model with total time for training equals to 887.6 seconds, with average epoch execution time 4 seconds. Whereas the number of training epochs in *Tashkeela* dataset increased to reach 284 epochs, which made the epochs’ training to last for 4,260 seconds. The average epoch execution time was 15 seconds. We notice that the average execution time per epoch in this model, in both used datasets, is the least compared to other proposed models. That is because the

Transformer Model architecture reduces the execution time because of its parallelism technique in manipulating data. The results gained from this model were good results and are illustrated in Table 6.1.

In the E-D-Attention-Concatenation Model, when dealing with *LDC ATB3* corpus, the accuracy score was bad compared to other tested models. The result recorded of loss was moreover not good. The value of BLEU-score was disappointing, which scored just 1.06. The use of *Tashkeela* corpus slightly improved the results, but it remains not good values. The results recorded from this model will not be presented or compared to other results.

In the E-D-Seq-to-Seq Model, the results were very good in both used datasets. When we used *LDC ATB3* dataset, the time taken for training this model was 2,250 seconds with average value for executing each epoch of 90 seconds. In the *Tashkeela* dataset, the total execution time was 16,065 seconds, with average execution time of 189 seconds per epoch. The remaining results are presented in Table 6.1.

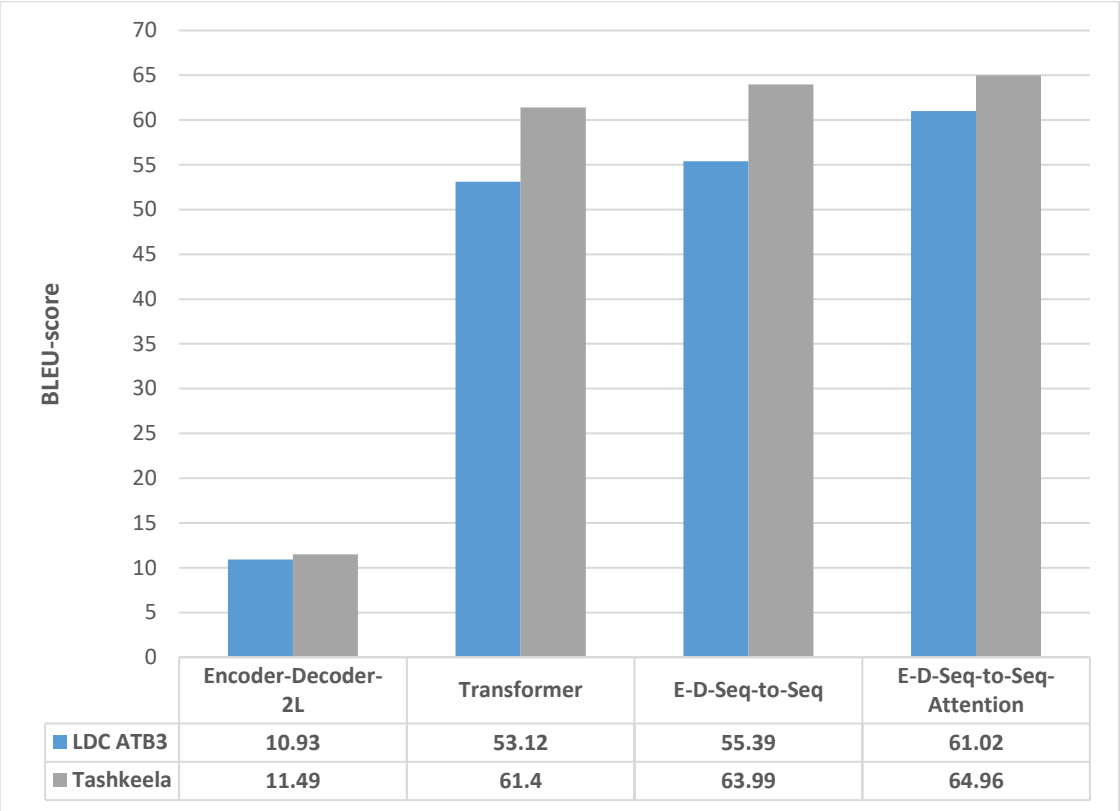
The good results achieved from the previous model, gave us the motive to experiment with the E-D-Seq-to-Seq-Attention Model. In this model, in the *LDC ATB3* corpus, the total execution time for training phase was 2,070 seconds, where the average execution time was 115 seconds per epoch. On the other hand, when using the *Tashkeela* corpus, the total execution time was 19,380 seconds, and the average execution time was 228 seconds per epoch. The rest of results are illustrated in Table 6.1.

By looking to Figure 6.1, which represents the BLEU-score for different models that were used to manage diacritizing Arabic text, we can notice that the Encoder-Decoder-2L Model, which we took it as a base model, gave BLEU-score results good as a point of start results. The value of BLEU-score in the *LDC ATB3* dataset was 10.93 and in the *Tashkeela* dataset result was 15.47. In the era of ML, these results are considered as not

good, but as we mentioned, this model is considered as a base model for other models to be compared with.

The remaining three models (Transformer Model, E-D-Seq-to-Seq Model and E-D-Seq-to-Seq- Attention Model), were the models that achieved the best proposed models' results in solving the diacritization problem. The Transformer Model is the model with the least BLEU-score value of the best three models. This model is differed from the other models of the techniques that it uses in its architecture, that it uses attention mechanism as the base component of the model. The description of the model is presented in Section 4.3. It is important to add that, the good results reflect the model's ability to solve diacritization problem.

The second model of the best proposed models is the E-D-Seq-to-Seq Model. This model uses in its architecture an embedding layer in both the encoder and decoder hierarchy of the model. The existence of embedding layers in both parts, gave the proposed model the ability to understand the relation between the input sequence and the output sequence and to connect between the related characteristic of both input and output sequences, in addition to reduce the dimensionality of the used embedding vectors. The use of this structure in the system has achieved good results; because by using embedding layer in both the encoder and decoder, the model can compare between two vectors with minimum dimensionality that the embedding layers present. This idea granted this model the ability to achieve the good results. This model scored BLEU-score in the *Tashkeela* dataset better than the result scored in the *LDC ATB3* dataset. Figure 6.1 reflects the results.



	Encoder-Decoder-2L	Transformer	E-D-Seq-to-Seq	E-D-Seq-to-Seq-Attention
LDC ATB3	10.93	53.12	55.39	61.02
Tashkeela	11.49	61.4	63.99	64.96

Figure 6.1: BLEU-score in diacritization problem for different models

As we can see in the bar-chart of Figure 6.1, the E-D-Seq-to-Seq- Attention Model is the best model that scored the best values, compared to other tested models. With a value of 61.02 for *LCD ATB3* corpus and a value of 64.96 for *Tashkeela* corpus. This model is an improvement of the preceding model we used (E-D-Seq-to-Seq Model).

The good results achieved by the E-D-Seq-to-Seq Model, pushed us to use the attention mechanism aiming to improve the results gained from this model. In the E-D-Seq-to-Seq-Attention Model, Loung attention (Luong *et al.*, 2015) is used to enhance the results. By using attention mechanism, the model improved its ability to diacritize the Arabic text accurately, and to improve the model's performance in solving the problem. The achieved BLEU-score result reflects the benefit earned to the model by using attention mechanism. Figure 6.1 illustrates the results.

To make a comparison of our work with others, we introduce our results using the best three models. In Table 6.2, we compared our results using accuracy metric with great

previously announced result using BiLSTM model introduced by Abandah and Abdel-Karim (2019) and other recently work presented by Qin *et al.*, (2021). In Abandah and Abdel-Karim (2019) work, they used three models (Encoder/Decoder model, Unidirectional LSTM and Bidirectional LSTM), and they apply the experiments using the same datasets we used (*LCD ATB3* and *Tashkeela*). The results in the table reflect that the best models' results in their work (Bidirectional LSTM), were the same as in the Transformer model in the *LCD ATB3* dataset. Whereas it outperforms the Transformer result in the *Tashkeela* dataset.

Table 6.2: Best diacritization results of our work and related work (bold numbers indicate the best results)

Model	Accuracy	
	LDC ATB3	***Tashkeela***
Encoder/Decoder (Abandah and Abdel-Karim, 2019)	33.0 %	36.0 %
Unidirectional LSTM (Abandah and Abdel-Karim, 2019)	81.0 %	88.0 %
Bidirectional LSTM (Abandah and Abdel-Karim, 2019)	**96.0 %**	**98.0 %**
AraBERT-BiLSTM (Qin *et al.*, 2021)	94.8 %	94.4 %
ZEN 2.0- BiLSTM (Qin *et al.*, 2021)	94.9 %	94.7 %
AraBERT-Transformer (Qin *et al.*, 2021)	94.9 %	94.5 %
ZEN 2.0- Transformer (Qin *et al.*, 2021)	95.1 %	94.9 %
Transformer (this work)	**96.0 %**	**97.1 %**
E-D-Seq-to-Seq (this work)	**99.5 %**	**99.8 %**
E-D-Seq-to-Seq-Attention (this work)	**99.3 %**	**99.7 %**

Qin *et al.*, (2021), recently tried to solve the diacritization problem using two models (AraBERT (Antoun *et al.*, 2020) and ZEN 2.0 (Song *et al.*, 2021)), they used BiLSTM and Transformer (Vaswani *et al.*, 2017), with the both proposed models to model the contextual information. This idea is used to solve the diacritization problem. In Table 6.2, we can note that none of the proposed models achieved superior results compared to ours or compared to Abandah and Abdel-Karim (2019). As we can see that, our best models (E-D-Seq-to-Seq and E-D-Seq-to-Seq-Attention) outperform others work in all used datasets. We can notice these results in Table 6.2.

6.2 Results and Discussion in CSM

The aforementioned models were used to solve the problem of diacritizing Arabic text. This problem is considered as a sequence-to-sequence problem. The problem of CSM is moreover considered as a sequence-to-sequence problem, which makes it beneficial to apply the models used to solve the diacritization problem in solving the CSM problem. In this section, the results of the models that used to process the text with spelling errors will be presented and discussed.

As the previous diacritization problem, in CSM problem we used the Encoder-Decoder-2L Model as the base model. The model in its performance of solving CSM problem was close to its performance in solving the diacritization problem in *LDC ATB3* dataset, with a better performance of the model's results in *Tashkeela* dataset. In this model, the early stopping technique is used, the same as used in the diacritization problem. The early stopping happened in epoch number 58 in *LDC ATB3* dataset, but it was in epoch number 66 in the *Tashkeela* dataset. The time taken for training the *LDC ATB3* dataset was 1,073 seconds, where this time was in *Tashkeela* dataset 5,866 seconds. The average epoch execution time was 19 seconds per epoch for *LDC ATB3* corpus and 89 seconds per epoch

for *Tashkeela* dataset. The results achieved from this model with other measured values are presented in Table 6.3.

In the Encoder-Decoder-2L-Attention Model, after applying many experiments with changing the parameters and hyper-parameters, the results we gained were bad for both *LDC ATB3* and *Tashkeela* datasets, the BLEU-score was *zero* in both datasets. This result reflects that, this model was not able to handle the problem of CSM. The disappointing results in the BLEU-score values were because that, the attention mechanism used (Bahdanau attention (Bahdanau *et al.,* 2014)), gave a negative influence on the model. In other words, the adding of attention layer directed the training process to a wrong way of training by making the model not to understand the dataset features, and to focus on building the words of the output text but in a bad output shape. The output text was a rubbish data and the model did not have the ability to interpret the input text to produce the desired output text.

Table 6.3: Evaluation results of the models used to solve CSM problem (bold numbers indicate the best results)

Model	Dataset	Loss	Accuracy	BLEU-score	Best Epoch	Execution time (seconds)	Average execution time (seconds/epoch)
Encoder-Decoder-2L	*LDC ATB3*	0.3771	0.3868	6.49	58	1,073	19
	Tashkeela	0.1262	0.6338	11.26	66	5,866	89
Transformer	*LDC ATB3*	**0.0016**	**0.9600**	**41.32**	256	768	3
	Tashkeela	**0.0011**	**0.9688**	**48.06**	293	3,223	11
E-D-Seq-to-Seq	*LDC ATB3*	0.0166	**0.9942**	**52.77**	35	3,045	87
	Tashkeela	0.0098	**0.9984**	**61.77**	128	22,400	175
E-D-Seq-to-Seq-Attention	*LDC ATB3*	0.0049	**0.9974**	**58.97**	28	2,828	101
	Tashkeela	0.0044	**0.9982**	**63.12**	132	27,984	212

In the Transformer Model, the results were good. In the *LDC ATB3* dataset, the training stage spent 768 seconds to finish the training. The average execution time was 3 seconds per epoch. The value of both loss and accuracy were good results and refer that this model can solve the CSM problem. In *Tashkeela* dataset, the results were better and there was improvement in the loss and accuracy values. The average execution time per epoch was 11 seconds, which is more than the average execution time of *LDC ATB3* dataset. The total execution time was 3,223 seconds. In this model, we notice that the average time is the least in both datasets compared to other proposed models. The results are presented in Table 6.3.

In the E-D-Attention-Concatenation Model, the results were disappointing, as the results in the diacritization problem. In the *LDC ATB3* corpus, the recorded BLEU-score was less than one, which reflects that the model did not learn to solve the problem in a right way. In *Tashkeela* dataset, the results were not in better conditions. The BLEU-score was less than one as well, which supports the previously mentioned note about this model that, it is not suitable to solve such problems.

The E-D-Seq-to-Seq Model is the first of two best models that achieved very good results in CSM problem. When using the *LDC ATB3* corpus, it took 35 epochs with total time of 3,045 seconds and average execution time of 87 seconds per epoch to finish the training process. In the other corpus (*Tashkeela*), the time spent to finish the training stage was 22,400 seconds and the average execution time was 175 seconds per epoch. After training 128 epochs in this model, the recorded results reflect the efficiency of using this model in solving CSM problem. All gained results are written down in Table 6.3.

The good results scored in the previous model, gave the scope to enhance the model by adding attention mechanism to improve these results. The last suggested model (E-D-

Seq-to-Seq-Attention Model) outperformed other proposed models' results. In the *LDC ATB3* dataset, the model trained with total execution time of 2,828 seconds to finish the training process. The average execution time per epoch was 101 seconds. In the *Tashkeela* dataset, it took for the model 132 epochs to finalize the training stage. The average time spent for execution was 212 seconds per epoch with a total execution time of 27,948. The aforementioned results with other results are illustrated in Table 6.3.

To compare different results of models in the CSM problem, Figure 6.2 illustrate the BLEU-score for different models used to solve this problem. The same models used to solve the diacritization problem are used to deal with CSM problem. As we can see in the Figure 6.2, the same behavior of models is noticed when observing the recorded results; the worst models in the diacritization problem were the same as in the CSM problem, which are: (Encoder-Decoder-2L-Attention Model and E-D-Attention-Concatenation Model). These results are expected, because both diacritization problem and CSM problem are sequence-to-sequence problems. Necessary to add that, the behavior of the Encoder-Decoder-2L-Attention Model was bad as in the diacritization problem. The output was rubbish data as it appeared in the previous diacritization problem. The reason of that because the model did not have the ability to understand the relationship between the input sequence and desired output sequence. In the second worse model (E-D-Attention-Concatenation Model), the results gained for both used datasets in BLEU-score were less than 1, which reflects that, this model did not have the ability to solve the problems of CSM and diacritization. We have to mention that the results for the worst two models will not be compared to other models.

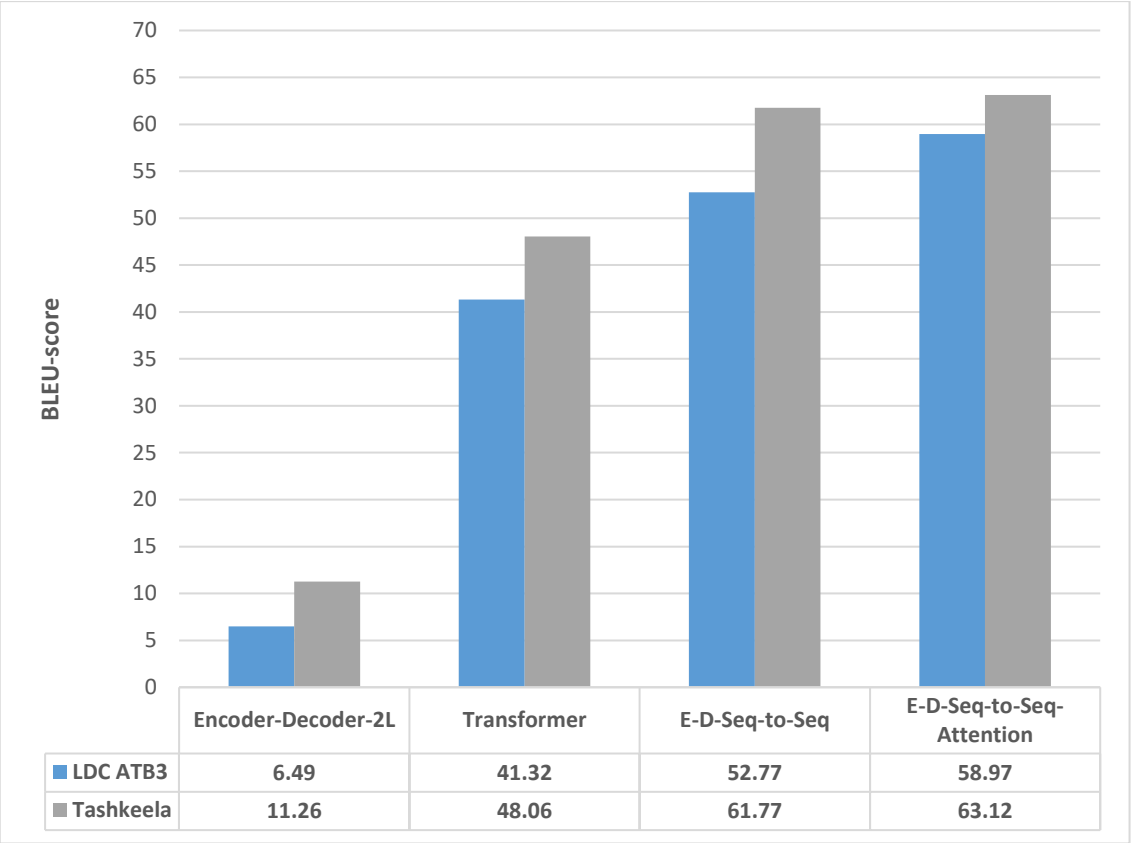


Figure 6.2: BLEU-score in CSM problem for different models

The third suggested model (Encoder-Decoder-2L Model) presented poor results in the *LDC ATB3* dataset which was 6.49 Bleu-score, which reflects that the model did not have the ability to solve the CSM problem. The result achieved when using the *Tashkeela* dataset supports our claim. In this dataset, the scored BLEU-score was 11.26, which is bad result and reflects that the model is not able to understand the CSM problem.

As the previous problem of diacritizing Arabic text, in the CSM problem the three best models are the same: Transformer, E-D-Seq-to-Seq and E-D-Seq-to-Seq-Attention models. The Transformer Model was the last in the three models in BLEU-score value with a value of 41.32 in the *LDC ATB3* corpus and a value of 48.06 in *Tashkeela* corpus. We can note that, in this model the value scored for the *Tashkeela* dataset is better than that scored for the *LDC ATB3* dataset, this because that the *Tashkeela* corpus is rich in number of samples (more text corpus) than *LDC ATB3* corpus, and the model could

understand this corpus as it is CA dataset (*Tashkeela* dataset) more than other one (*LDC ATB3* dataset), which is a MSA dataset.

The E-D-Seq-to-Seq Model scored results, which are good values in the BLEU-score metric; the values achieved are: 52.77 in the *LDC ATB3* dataset and 61.77 in the *Tashkeela* dataset. This model had the ability to solve the CSM problem in an efficient way and in both CA and MSA corpuses. We can notice that in the *Tashkeela* dataset, which is a MSA corpus, the model has BLEU-score value more than the one in the *LDC ATB3* corpus. This because the same reasons we mentioned in the Transformer Model.

The last model, which scored the best results of the overall models, is the E-D-Seq-to-Seq-Attention Model. By adding attention mechanism (Luong attention (Luong *et al.*, 2015)), this model achieved a BLEU-score value of 58.97 in the *LDC ATB3* dataset, which is a good result. In *Tashkeela* dataset, a value of 63.12 BLEU-score is achieved, which is a very good result. By looking to these results, we conclude that this model can deal with the CSM problem and has the ability to solve this problem in an effective manner. As we mentioned previously, the existence of embedding layer with the architecture of this model, created a mechanism that granted this model the ability to solve this sequence-to-sequence problem efficiently. Figure 6.2 illustrates the aforementioned results.

To evaluate our work, we compared it with related work. Table 6.4 present the best three models' results of our work compared to others work in CSM problem. We compared our results with Azmi *et al.*, (2019) results and Alkhatib *et al.*, (2020) results.

Azmi *et al.*, (2019) proposed error detection and correction model. For the detection phase, they combine between language model and ML, whereas, for correction step they use language model. The accuracy results of their work divided the created tested errors in to two types, dependent on the location of error word: Distance1, which is one edit

distance away from correct word. Moreover, distance2, which is two edit distances. The error density was 15% on 10,000 sentences. In addition, the results presented in two groups: Group1, which has the distance1 error, and group2 which has a combination of distance1 (80%) and distance2 (20%). All the previous results are illustrated in table 6.4. Alkhatib *et al.*, (2020) introduced a general approach, which investigates using deep neural networks for Arabic text error detection. They improved a systematic architecture for error detection and correction at a word level. The model they proposed consists of word embedding, BiLSTM memory mechanism and polynomial network (PN) classifier instead of a decoder. The accuracy result of this model is presented in Table 6.4. As we can see in Table 6.4, BiLSTM-PN scored a non-comparative result, whereas the value of (Group1-distance1) scored a great result. Despite of this, our best two models (E-D-Seq-to-Seq and E-D-Seq-to-Seq-Attention) outperforms all these models.

Table 6.4: Best CSM results of our work and related work

(bold numbers indicate the best results)

Model	***LDC ATB3***
Group1-distance1 (Azmi *et al.*, 2019)	**98.0 %**
Group2-distance1 (Azmi *et al.*, 2019)	93.8 %
Group2-distance2 (Azmi *et al.*, 2019)	84.2 %
BiLSTM-PN (Alkhatib *et al.*, 2020)	93.9 %
Transformer (this work)	**96.0 %**
E-D-Seq-to-Seq (this work)	**99.4 %**
E-D-Seq-to-Seq-Attention (this work)	**99.7 %**

6.3 Comparing Results of Models in Diacritization and CSM

In the previous sections, Section 6.1 and Section 6.2, the results of used models for diacritization problem and CSM problem are illustrated and discussed. In this section, the models used will be compared in the same corpus with these two problems. The dataset of *LDC ATB3* as a MSA dataset will be discussed first. When we look at Figure 6.3, we can see that Encoder-decoder-2L Model scored poor results compared to other proposed models. The remaining three models were the models that scored great results in both diacritization and CSM problems. In the *LDC ATB3* dataset, Transformer Model achieved BLEU score result in the diacritization problem better than the result scored in the CSM problem. The same tendency of these result is noticed in the E-D-Seq-to-Seq Model and E-D-Seq-to-Seq-Attention Model. As we can see in Figure 6.3, Transformer model achieved the least BLEU-score of the best three models, followed by E-D-Seq-to-Seq Model and then E-D-Seq-to-Seq-Attention Model in the scored values respectively. In *Tashkeela* dataset, it was the same behavior for the best three models; that is, the best BLEU-score value was for the E-D-Seq-to-Seq-Attention Model followed by less value for E-D-Seq-to-Seq Model, and the least value was scored for Transformer Model (Figure 6.4 present the results). It is important to add that, the BLEU-score value for all previously mentioned models, scored more values in *Tashkeela* corpus than *LDC ATB3* corpus. That is, in each model the value of BLEU-score recorded in *Tashkeela* dataset, is more than the value scored in the *LDC ATB3* dataset. The results are illustrated in Figure 6.3 for *LDC ATB3* dataset and Figure 6.4 for *Tashkeela* dataset.

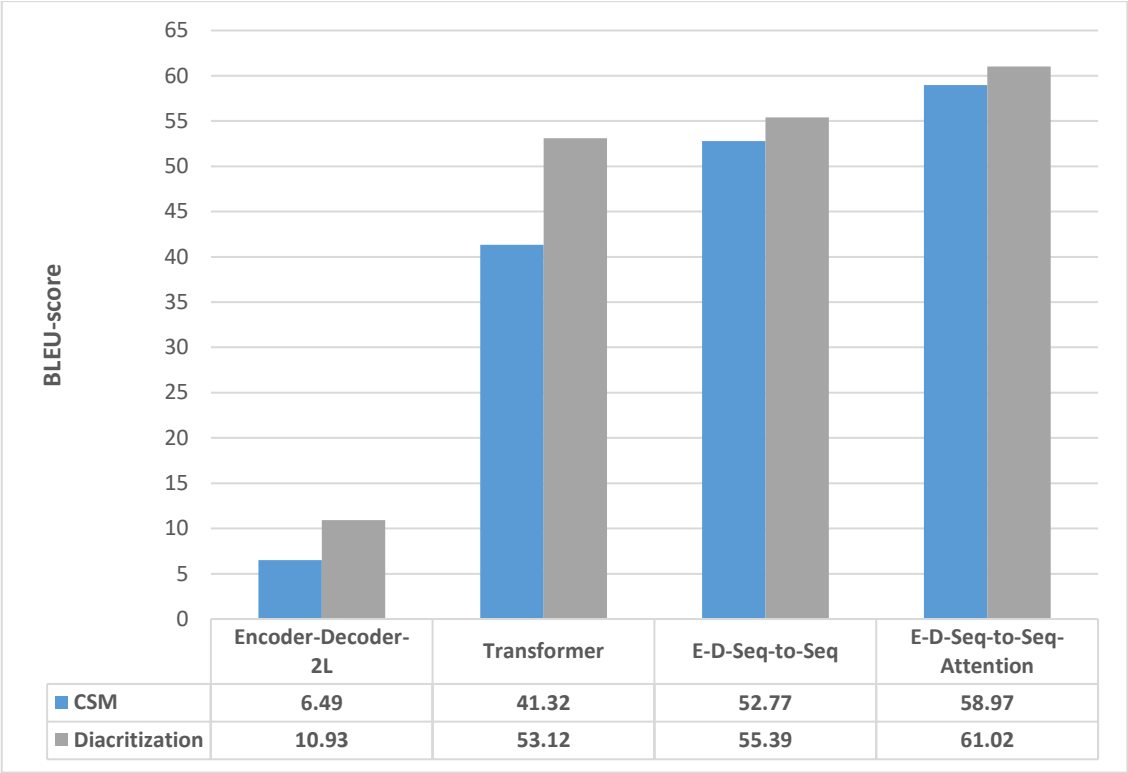


Figure 6.3: BLEU-score in both diacritization and CSM problems for *LDC ATB3* dataset using different models

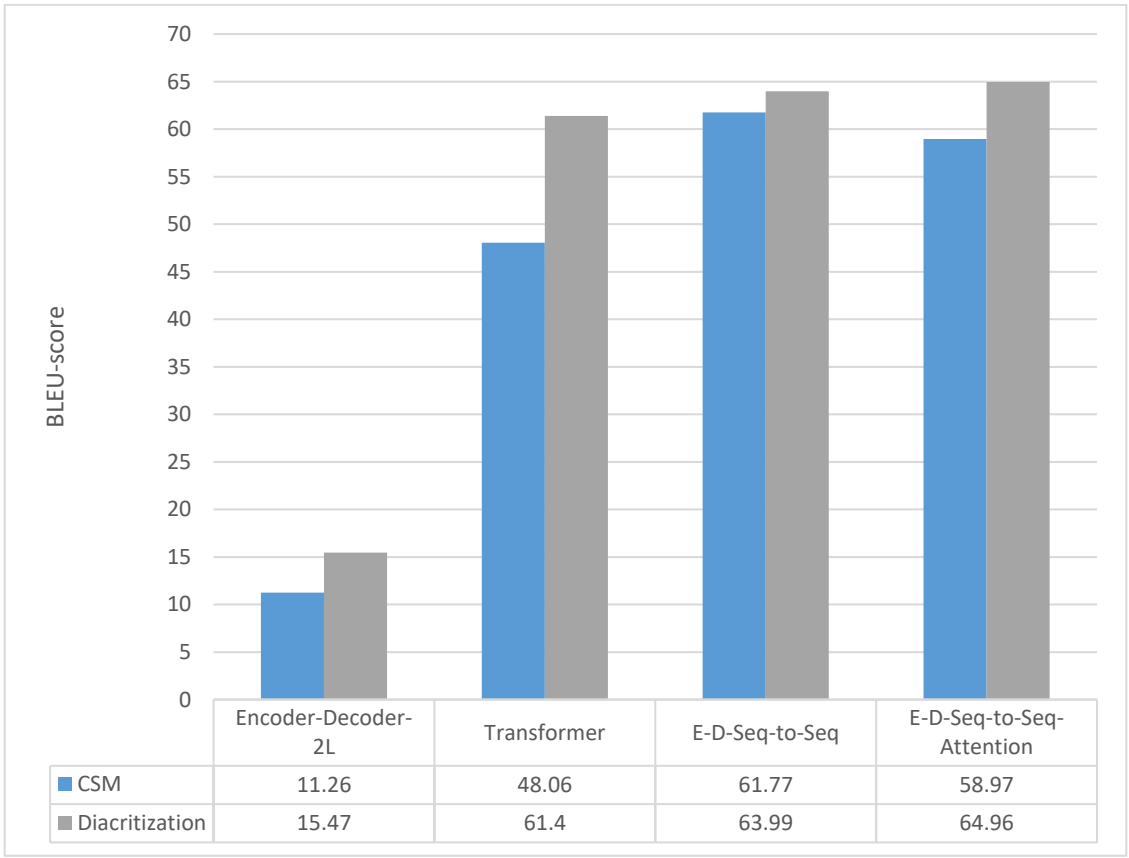


Figure 6.4: BLEU-score in both diacritization and CSM problems for *Tashkeela* dataset using different models

Extra work has been done; the success in the best two models in both diacritization and CSM problems (E-D-Seq-to-Seq Model and E-D-Seq-to-Seq-Attention Model), pushed us to test these models to solve the CSM and diacritization problem in a simultaneous manner. Table 6.5 presents the results achieved using the *LDC ATB3* and *Tashkeela* corpuses. As we can see that both models achieved good results using the two used corpuses. The first used model is the E-D-Seq-to-Seq Model. This model scored BLEU-score values of 59.47 for the *LDC ATB3* dataset and 65.12 for *Tashkeela* dataset. The recorded results reflected that this model had the ability to solve the problem that combines between CSM and diacritization problems successfully. In addition, this model has the ability to benefit from diacritization to increase the understanding of the problem to output the needed sequence.

Table 6.5: Evaluation results of the models used to solve CSM and Diacritization problem

Model	Dataset	Loss	Accuracy	BLEU-score	Best Epoch	Execution time (seconds)	Average execution time (seconds/epoch)
E-D-Seq-to-Seq	*LDC ATB3*	0.0096	0.9932	59.47	33	3,630	110
	Tashkeela	0.0032	0.9990	65.12	138	16,698	121
E-D-Seq-to-Seq-Attention	*LDC ATB3*	0.0097	0.9983	62.16	28	3,929	140
	Tashkeela	0.0026	0.9991	67.73	72	20,376	283

The other model is the E-D-Seq-to-Seq-Attention Model. This model achieved a BLEU-score for the *LDC ATB3* dataset of 62.16, the using of *Tashkeela* corpus in this model scored 67.73 BLEU-score value. In this model, the existing attention mechanism increased the model accuracy and ability to solve the diacritization and CSM problems simultaneously. As in the problems of diacritization and CSM, when we solved them separately, this model gave the best results of the proposed models. In this problem (CSM-Diacritization problem), the model had the ability to solve it in good BLEU-score results. Figure 6.5 presents the aforementioned two models with the BLEU-score results for the CSM-Diacritization problem.

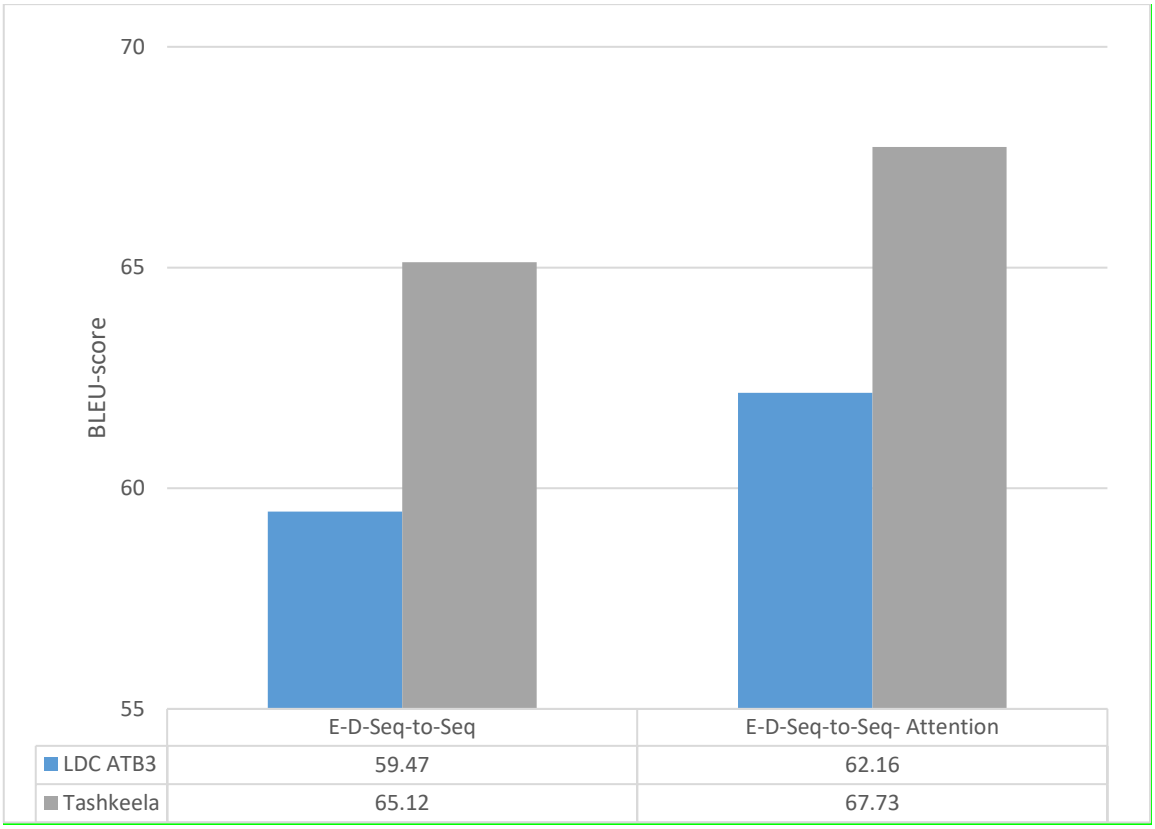


Figure 6.5: BLEU-score of the best models used for solving CSM andDiacritization problem in different dataset used

Chapter Seven
Conclusion and Future Work

7. Conclusion and Future Work

In this chapter, we firstly conclude our work with different conclusions that reflect the benefits and results we gained from this work. Then we present the future work that will be searched and completed in the future.

7.1 Conclusion

In our work, we aimed to investigate the existing methods to solve the problem of diacritizing Arabic text, and these methods were used to deal with the CSM problem. There are six different models that we used to test these two problems. Three of these models achieved the best results of the investigated models. These models are Transformer model, E-D-Seq-to-Seq Model and E-D-Seq-to-Seq-Attention Model. The aforementioned models achieved good BLEU-score results in both diacritization and CSM Problems. The best result scored was 64.96 using *LDC ATB3* dataset in the E-D-Seq-to-Seq-Attention Model in the diacritization problem. In addition, the best result scored in the CSM problem was 76.12 using *LDC ATB3* dataset in the same model. Necessary to add that the problems of both diacritizing Arabic text and CSM are considered as sequence-to-sequence problems, and the three previously mentioned models had the ability to solve these problems with amazingly recorded results. We notice that in the E-D-Seq-to-Seq Model, the existence of the embedding layer in both the encoder and decoder parts of the model, enhanced the performance of this model, and hence competent scores were achieved. In our work, the E-D-Seq-to-Seq-Attention Model improved the value of the BLEU-score of the E-D-Seq-to-Seq Model by using attention mechanism in the model's architecture. Another model, which is Transformer Model, benefit from using the attention mechanism as a main structure of the model without using any other mechanisms, and it scored good results.

7.2 Future Work

As future work, the good results gained in the Diacritization and CSM problems using the proposed models, opened the way to think of proposing a complete model, which can take the colloquial Arabic text and translate it to eloquent Arabic text. Furthermore, we are working on solving the CSM problem and adding diacritics to the text in a simultaneous manner.

References

Abandah, G. A., Graves, A., Al-Shagoor, B., Arabiyat, A., Jamour, F., & Al-Taee, M. (2015). **Automatic diacritization of Arabic text using recurrent neural networks.** International Journal on Document Analysis and Recognition (IJDAR), 18(2), 183-197.

Abandah, G., & Abdel-Karim, A. (2019). **ACCURATE AND FAST RECURRENT NEURAL NETWORK SOLUTION FOR THE AUTOMATIC DIACRITIZATION OF ARABIC TEXT.** Jordanian Journal of Computers and Information Technology (JJCIT), 6(2).

Abandah, G., & Arabiyat, A. (2017, October). **Investigating hybrid approaches for Arabic text diacritization with recurrent neural networks.** In 2017 IEEE Jordan conference on applied electrical engineering and computing technologies (AEECT) (pp. 1-6).

Akbik, A., Blythe, D., & Vollgraf, R. (2018, August). **Contextual string embeddings for sequence labeling.** In Proceedings of the 27th international conference on computational linguistics (pp. 1638-1649).

Alghamdi, M., Muzaffar, Z., & Alhakami, H. (2010). **Automatic restoration of Arabic diacritics: a simple, purely statistical approach.** Arabian Journal for Science and Engineering, 35(2), 125.

Al-Hagery, M. A., Alreshoodi, L. A., Almutairi, M. A., Alsharekh, S. I., & Alkhowaiter, E. S. (2019). **A hybrid Technique for Cleaning Missing and Misspelling Arabic Data in Data Warehouse.**

Alkanhal, M. I., Al-Badrashiny, M. A., Alghamdi, M. M., & Al-Qabbany, A. O. (2012). **Automatic stochastic Arabic spelling correction with emphasis on space insertions and deletions.** IEEE Transactions on Audio, Speech, and Language Processing, 20(7), 2111-2122.

AlKhamissi, B., ElNokrashy, M. N., & Gabr, M. (2020). **Deep Diacritization: Efficient Hierarchical Recurrence for Improved Arabic Diacritization.** arXiv preprint arXiv:2011.00538.

Alkhatib, M., Monem, A. A., & Shaalan, K. (2020). **Deep Learning for Arabic Error Detection and Correction.** ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP), 19(5), 1-13.

AlShenaifi, N., AlNefie, R., Al-Yahya, M., & Al-Khalifa, H. (2015, July). **ARIB@ QALB-2015 shared task: a hybrid cascade model for Arabic spelling error detection and correction.** In Proceedings of the Second Workshop on Arabic Natural Language Processing (pp. 127-132).

Al-TAAni, A. T., Wedian, S. A., & Darwish, O. M. (2009). **Arabic numerals checker: checking agreement between numerals and counted objects in the Arabic language.** International Journal of Computer Processing of Languages, 22(04), 341-357.

Al-Thubaity, A., Alkhalifa, A., Almuhareb, A., & Alsanie, W. (2020). **Arabic Diacritization Using Bidirectional Long Short-Term Memory Neural Networks with Conditional Random Fields.** IEEE Access, 8, 154984-154996.

Antoun, W., Baly, F., & Hajj, H. (2020). **Arabert: Transformer-based model for Arabic language understanding.** arXiv preprint arXiv:2003.00104.

Ataman, D., Firat, O., Di Gangi, M. A., Federico, M., & Birch, A. (2019). **On the importance of word boundaries in character-level neural machine translation.** arXiv preprint arXiv:1910.06753.

Attia, M., Pecina, P., Samih, Y., Shaalan, K., & Van Genabith, J. (2016). **Arabic spelling error detection and correction.** Natural Language Engineering, 22(5), 751-773.

Azmi, A. M., Almutery, M. N., & Aboalsamh, H. (2019). **Real-word errors in Arabic texts: A better algorithm for detection and correction.** IEEE/ACM Transactions on Audio, Speech, and Language Processing, 27(8), 1308 – 1320.

Bahanshal, A. O., & Al-Khalifa, H. S. (2012, August). **A first approach to the evaluation of Arabic diacritization systems.** In Seventh International Conference on Digital Information Management (ICDIM 2012) (pp. 155-158). IEEE.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). **Neural machine translation by jointly learning to align and translate.** arXiv preprint arXiv:1409.0473.

Bebah, M. O. A. O., Meziane, A., Mazroui, A., & Lakhouaja, A. (2011, May). **Alkhalil morpho sys.** In 7th International computing conference in Arabic.

Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. (2003). **A neural probabilistic language model.** The journal of machine learning research, 3, 1137-1155.

Berger, A., Della Pietra, S. A., & Della Pietra, V. J. (1996). **A maximum entropy approach to natural language processing.** Computational linguistics, 22(1), 39-71.

Boudchiche, M., Mazroui, A., Bebah, MOAO, Lakhouaja, A., & Boudlal, A. (2014, February). **The Morphosyntaxic Analyzer AlKhalil Morpho Sys 2.** In 1st National Doctoral Day on Engineering of the Arabic Language, (JDILA'14) (pp. 1-5).

Brill, E., & Moore, R. C. (2000, October). **An improved error model for noisy channel spelling correction**. In Proceedings of the 38th Annual Meeting on Association for Computational Linguistics (pp. 286-293).

Britz, D., Goldie, A., Luong, M. T., & Le, Q. (2017). **Massive exploration of neural machine translation architectures.** arXiv preprint arXiv:1703.03906.

Buckwalter, T. (2002). **Buckwalter Arabic morphological analyzer version 1.0**. Linguistic Data Consortium, University of Pennsylvania.

Chennoufi, A., & Mazroui, A. (2016). **Impact of morphological analysis and a large training corpus on the performances of Arabic diacritization.** International Journal of Speech Technology, 19(2), 269-280.

Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014a). **On the properties of neural machine translation: Encoder-decoder approaches.** arXiv preprint arXiv:1409.1259.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014b). **Learning phrase representations using RNN encoder-decoder for statistical machine translation.** arXiv preprint arXiv:1406.1078.

Church, K. W., & Gale, W. A. (1991). **Probability scoring for spelling correction**. Statistics and Computing, 1(2), 93-103.

Damerau, F. J. (1964). **A technique for computer detection and correction of spelling errors**. Communications of the ACM, 7(3), 171-176.

Darwish, K., & Mubarak, H. (2016, May). **Farasa: A new fast and accurate Arabic word segmenter.** In Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16) (pp. 1070-1074).

Dash, Y., & Dubey, S. K. (2012**). Quality prediction in object oriented system by using ANN: a brief survey.** International Journal of Advanced Research in Computer Science and Software Engineering, 2(2).

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). **Bert: Pre-training of deep bidirectional transformers for language understanding.** arXiv preprint arXiv:1810.04805.

Diab, M., Hacioglu, K., & Jurafsky, D. (2007). **Automated methods for processing Arabic text: from tokenization to base phrase chunking.** Arabic computational morphology: Knowledge-based and empirical methods. Kluwer/Springer.

Dou, Q., Lu, Y., Manakul, P., Wu, X., & Gales, M. J. (2021). **Attention Forcing for Machine Translation.** arXiv preprint arXiv:2104.01264.

El-Imam, Y. A. (2004). **Phonetization of Arabic: rules and algorithms.** Computer Speech & Language, 18(4), 339-373.

El-Sadany, T., & Hashish, M. (1988, April). **Semi-automatic vowelization of Arabic verbs.** In 10th NC Conference, Jeddah, Saudi Arabia.

Elman, J. L. (1990). **Finding structure in time.** Cognitive science, 14(2), 179-211.

Emam, O., & Fischer, V. (2005). **Hierarchical approach for the statistical vowelization of Arabic text.** U.S. Patent Application 10/948,443.

Fadel, A., Tuffaha, I., Al-Jawarneh, B., & Al-Ayyoub, M. (2019). **Arabic Text Diacritization Using Deep Neural Networks**. arXiv preprint arXiv:1905.01965.

Fan, R. E., Chang, K. W., Hsieh, C. J., Wang, X. R., & Lin, C. J. (2008). **LIBLINEAR: A library for large linear classification.** The Journal of machine Learning research, 9, 1871-1874.

Fashwan, A., & Alansary, S. (2017, April). **SHAKKIL: an automatic diacritization system for modern standard Arabic texts.** In Proceedings of the Third Arabic Natural Language Processing Workshop (pp. 84-93).

Gal, Y. A. (2002, July). **An HMM approach to vowel restoration in Arabic and Hebrew.** In Proceedings of the ACL-02 workshop on Computational approaches to semitic languages.

Galassi, A., Lippi, M., & Torroni, P. (2020). **Attention in Natural Language Processing.** IEEE Transactions on Neural Networks and Learning Systems.

Giménez, J., & Marquez, L. (2004). SVMTool: **A general POS tagger generator based on Support Vector Machines.** In In Proceedings of the 4th International Conference on Language Resources and Evaluation.

Google Colaboratory. (2017). Retrieved 27 January 2020, from <https://colab.research.google.com>

Graves, A., Wayne, G., & Danihelka, I. (2014). **Neural turing machines.** arXiv preprint arXiv:1410.5401.

Gumilang, M., & Purwarianti, A. (2018, October). **Experiments on character and word level features for text classification using deep neural network.** In 2018 Third International Conference on Informatics and Computing (ICIC) (pp. 1-6).

Guo, Q., Qiu, X., Liu, P., Shao, Y., Xue, X., & Zhang, Z. (2019). **Star-transformer.** arXiv preprint arXiv:1902.09113.

Habash, N., & Rambow, O. (2005, June). **Arabic tokenization, part-of-speech tagging and morphological disambiguation in one fell swoop.** In Proceedings of the 43rd annual meeting of the association for computational linguistics (ACL'05) (pp. 573-580).

Habash, N., & Rambow, O. (2007, April). **Arabic diacritization through full morphological tagging.** In Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers (pp. 53-56).

Habash, N., Rambow, O., & Roth, R. (2009, April). **MADA+ TOKAN: A toolkit for Arabic tokenization, diacritization, morphological disambiguation, POS tagging, stemming and lemmatization.** In Proceedings of the 2nd international conference on Arabic language resources and tools (MEDAR), Cairo, Egypt (Vol. 41, p. 62). stemming and lemmatization.

Habash, N., Roth, R., Rambow, O., Eskander, R., & Tomeh, N. (2013, June). **Morphological analysis and disambiguation for dialectal Arabic.** In Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (pp. 426-432).

Haddad, B., & Yaseen, M. (2007). **Detection and correction of non-words in Arabic: A hybrid approach**. International Journal of Computer Processing of Oriental Languages, 20(04), 237-257.

Hajic, J., Smrz, O., Buckwalter, T., & Jin, H. (2005). **Feature-based tagger of approximations of functional Arabic morphology.** In Proceedings of the Workshop on Treebanks and Linguistic Theories (TLT), Barcelona, Spain.

Hammerton, J. (2003). **Named entity recognition with long short-term memory.** In Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003 (pp. 172-175).

Han, B., & Baldwin, T. (2011, June). **Lexical normalisation of short text messages: Makn sens a# twitter**. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1 (pp. 368-378).

Hassan, A., Noeman, S., & Hassan, H. (2008). **Language independent text correction using finite state automata**. In Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-II.

Hochreiter, S., & Schmidhuber, J. (1997). **Long short-term memory.** Neural computation, 9(8), 1735-1780.

Hovermale, D. J., & Mehay, D. N. (2009). **Real-word Spelling Correction for CALL**. In Presentation for the Sixth Midwest Computational Linguistics Colloquium.

Hwang, K., & Sung, W. (2017, March). **Character-level language modeling with hierarchical recurrent neural networks.** In 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 5720-5724).

Jeblee, S., Bouamor, H., Zaghouani, W., & Oflazer, K. (2014, October). **CMUQ@ QALB-2014: An SMT-based System for Automatic Arabic Error Correction.** In Proceedings of the EMNLP 2014 Workshop on Arabic Natural Language Processing (ANLP) (pp. 137-142).

Jing, K., & Xu, J. (2019). **A survey on neural network language models.** arXiv preprint arXiv:1906.03591.

Kaggle: Your Machine Learning and Data Science Community. (2014). Retrieved 16 March 2020, from <https://www.kaggle.com/>

Kalchbrenner, N., & Blunsom, P. (2013, October). **Recurrent continuous translation models.** In Proceedings of the 2013 conference on empirical methods in natural language processing (pp. 1700-1709).

Kim, Y., Jernite, Y., Sontag, D., & Rush, A. (2016, March). **Character-aware neural language models.** In Proceedings of the AAAI conference on artificial intelligence (Vol. 30, No. 1).

Kukich, K. (1992). **Techniques for automatically correcting words in text**. Acm Computing Surveys (CSUR), 24(4), 377-439.

Kuru, O., Can, O. A., & Yuret, D. (2016, December). **Charner: Character-level named entity recognition.** In Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers (pp. 911-921).

Larochelle, H., & Hinton, G. E. (2010). **Learning to combine foveal glimpses with a third-order Boltzmann machine.** In Advances in neural information processing systems (pp. 1243-1251).

LeCun, Y. (2015). Yoshua bengio, and Geoffrey hinton. **Deep learning.** In: Nature 521 (7553), 436-444.

Luong, M. T., Pham, H., & Manning, C. D. (2015). **Effective approaches to attention-based neural machine translation.** arXiv preprint arXiv:1508.04025.

Madhfar, M., & Qamar, A. M. (2020). **Effective Deep Learning Models for Automatic Diacritization of Arabic Text.** IEEE Access.

Madi, N., & Al-Khalifa, H. S. (2018). **A proposed Arabic grammatical error detection tool based on deep learning.** Procedia computer science, 142, 352-355.

Madi, N., & Al-Khalifa, H. (2020). **Error Detection for Arabic Text Using Neural Sequence Labeling.** Applied Sciences, 10(15), 5279.

Magdy, M., Shaalan, K., & Fahmy, A. (2007, December). **Lexical error diagnosis for second language learners of Arabic**. In Proceedings of the Seventh Conference on Language Engineering, Egyptian Society of Language Engineering (ELSE) (pp. 5-6).

Mahmoud, S. A., Al-Khatib, W. G., & Alnethary, T. (2019). **Arabic spell checking error model.** U.S. Patent No. 10,515,148. Washington, DC: U.S. Patent and Trademark Office.

Mars, M. (2016, July). **Toward a Robust Spell Checker for Arabic Text.** In International Conference on Computational Science and Its Applications (pp. 312-322). Springer, Cham.

Medina, J. R., & Kalita, J. (2018, December). **Parallel attention mechanisms in neural machine translation.** In 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA) (pp. 547-552). IEEE.

Metwally, A. S., Rashwan, M. A., & Atiya, A. F. (2016, July). **A multi-layered approach for Arabic text diacritization.** In 2016 IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA) (pp. 389-393). IEEE.

Mikolov, T., Sutskever, I., Deoras, A., Le, H. S., Kombrink, S., & Cernocky, J. (2012). **Subword language modeling with neural networks.** Preprint (http://www. fit. vutbr. cz/imikolov/rnnlm/char. pdf), 8, 67.

Mnih, V., Heess, N., & Graves, A. (2014). **Recurrent models of visual attention.** Advances in neural information processing systems, 27, 2204-2212.

Moumen, R., Chiheb, R., Faizi, R., & El Afia, A. (2018, May). **Arabic diacritization with gated recurrent unit.** In Proceedings of the International Conference on Learning and Optimization Algorithms: Theory and Applications (pp. 1-4).

Mubarak, H., Abdelali, A., Sajjad, H., Samih, Y., & Darwish, K. (2019, June). **Highly effective Arabic diacritization using sequence to sequence modeling**. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers) (pp. 2390-2395).

Neco, R. P., & Forcada, M. L. (1997, June). **Asynchronous translations with recurrent neural nets.** In Proceedings of International Conference on Neural Networks (ICNN'97) (Vol. 4, pp. 2535-2540).

Nelken, R., & Shieber, S. (2005). **Arabic diacritization using weighted finite-state transducers.** In Proceedings of the 2005 ACL Workshop on Computational Approaches to Semitic Languages. Association for Computational Linguistics.

Noaman, H. M., Sarhan, S. S., & Rashwan, M. (2016). **Automatic Arabic spelling errors detection and correction based on confusion matrix-noisy channel hybrid system**. Egypt Comput Sci J, 40(2), 54-64.

Pasha, A., Al-Badrashiny, M., Diab, M. T., El Kholy, A., Eskander, R., Habash, N., ... & Roth, R. (2014, May). **Madamira: A fast, comprehensive tool for morphological analysis and disambiguation of Arabic.** In LREC (Vol. 14, No. 2014, pp. 1094-1101).

Pascanu, R., Mikolov, T., & Bengio, Y. (2013, May). **On the difficulty of training recurrent neural networks.** In International conference on machine learning (pp. 1310-1318). PMLR.

Pinkus, A. (1999). **Approximation theory of the MLP model.** Acta Numerica 1999: Volume 8, 8, 143-195.

Qin, H., Chen, G., Tian, Y., & Song, Y. (2021). **Improving Arabic Diacritization with Regularized Decoding and Adversarial Training.** In Proceedings of the Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing.

Platanios, E. A., Sachan, M., Neubig, G., & Mitchell, T. (2018). **Contextual parameter generation for universal neural machine translation.** arXiv preprint arXiv:1808.08493.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). **Language models are unsupervised multitask learners.** OpenAI blog, 1(8), 9.

Rashwan, M. A., Al Sallab, A. A., Raafat, H. M., & Rafea, A. (2015). **Deep learning framework with confused sub-set resolution architecture for automatic Arabic diacritization.** IEEE/ACM Transactions on Audio, Speech, And Language Processing, 23(3), 505-516.

Rashwan, M. A., Al-Badrashiny, M. A., Attia, M., Abdou, S. M., & Rafea, A. (2010**). A stochastic Arabic diacritizer based on a hybrid of factorized and unfactorized textual features.** IEEE Transactions on Audio, Speech, and Language Processing, 19(1), 166-175.

Rashwan, M., Al-Badrashiny, M., Attia, M., & Abdou, S. (2009, September). **A hybrid system for automatic Arabic diacritization.** In The 2nd international conference on Arabic language resources and tools (pp. 54-60).

Rokaya, M. (2015). **Arabic semantic spell checking based on power links**. International Information Institute (Tokyo). Information, 18(11), 4749.

Rozovskaya, A., Bouamor, H., Habash, N., Zaghouani, W., Obeid, O., & Mohit, B. (2015, July). **The second qalb shared task on automatic text correction for Arabic.** In Proceedings of the Second Workshop on Arabic Natural Language Processing (pp. 26-35).

Saad, M. K., & Ashour, W. M. (2010). **OSAC: Open source Arabic corpora.** In 6th ArchEng Int. Symposiums, EEECS (Vol. 10).

Said, A., El-Sharqwi, M., Chalabi, A., & Kamal, E. (2013, June). **A hybrid approach for Arabic diacritization.** In International Conference on Application of Natural Language to Information Systems (pp. 53-64). Springer, Berlin, Heidelberg.

Shaalan, K. F., Attia, M., Pecina, P., Samih, Y., & van Genabith, J. (2012, May). **Arabic Word Generation and Modelling for Spell Checking.** In LREC (pp. 719-725).

Shaalan, K., Aref, R., & Fahmy, A. (2010, March). **An approach for analyzing and correcting spelling errors for non-native Arabic learners**. In 2010 The 7th International Conference on Informatics and Systems (INFOS) (pp. 1-7).

Shaalan, K., Bakr, H. M. A., & Ziedan, I. (2009, March). **A hybrid approach for building Arabic diacritizer.** In Proceedings of the EACL 2009 workshop on computational approaches to semitic languages (pp. 27-35).

Shi, X., Huang, H., Jian, P., & Tang, Y. K. (2021). **Improving neural machine translation with sentence alignment learning.** Neurocomputing, 420, 15-26.

Song, Y., Zhang, T., Wang, Y., & Lee, K. F. (2021). **ZEN 2.0: Continue Training and Adaption for N-gram Enhanced Text Encoders.** arXiv preprint arXiv:2105.01279.

Sukhbaatar, S., Weston, J., & Fergus, R. (2015). **End-to-end memory networks.** Advances in neural information processing systems, 28, 2440-2448.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). **Sequence to sequence learning with neural networks**. Advances in NIPS.

Tang, G., Sennrich, R., & Nivre, J. (2018). **An analysis of attention mechanisms: The case of word sense disambiguation in neural machine translation.** arXiv preprint arXiv:1810.07595.

Tran, Q., MacKinlay, A., & Yepes, A. J. (2017). **Named entity recognition with stack residual LSTM and trainable bias decoding.** arXiv preprint arXiv:1706.07598.

Tu, Z., Lu, Z., Liu, Y., Liu, X., & Li, H. (2016). **Modeling coverage for neural machine translation.** arXiv preprint arXiv:1601.04811.

Van Delden, S., Bracewell, D., & Gomez, F. (2004, November). **Supervised and unsupervised automatic spelling correction algorithms**. In Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration, 2004. IRI 2004. (pp. 530-535).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). **Attention is all you need.** In Advances in neural information processing systems (pp. 5998-6008).

Vergyri, D., & Kirchhoff, K. (2004). **Automatic diacritization of Arabic for acoustic modeling in speech recognition.** In Proceedings of the workshop on computational approaches to Arabic script-based languages (pp. 66-73).

Verwimp, L., Pelemans, J., & Wambacq, P. (2017). **Character-word LSTM language models.** arXiv preprint arXiv:1704.02813.

Wali, W., & Gargouri, B. (2015). **Supervised learning to measure the semantic similarity between Arabic sentences.** In Computational Collective Intelligence (pp. 158-167). Springer, Cham.

Wang, B., Liu, K., & Zhao, J. (2016, August). **Inner attention based recurrent neural networks for answer selection.** In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (pp. 1288-1297).

Wang, X., Hua, Y., Kodirov, E., Hu, G., & Robertson, N. M. (2019, July). **Deep metric learning by online soft mining and class-aware attention.** In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 33, pp. 5361-5368).

Wang, Y., Skerry-Ryan, R. J., Stanton, D., Wu, Y., Weiss, R. J., Jaitly, N., ... & Saurous, R. A. (2017). **Tacotron: Towards end-to-end speech synthesis.** arXiv preprint arXiv:1703.10135.

Wu, J. C., Chiu, H. W., & Chang, J. S. (2013, December). **Integrating dictionary and web N-grams for Chinese spell checking**. In International Journal of Computational Linguistics & Chinese Language Processing, 18(4), 17-30.

Wu, L., Tian, F., Zhao, L., Lai, J., & Liu, T. Y. (2018, April). **Word attention for sequence to sequence text understanding.** In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 32, No. 1).

Xie, Z., Avati, A., Arivazhagan, N., Jurafsky, D., & Ng, A. Y. (2016). **Neural language correction with character-based attention**. arXiv preprint arXiv:1603.09727.

Yang, Z., Salakhutdinov, R., & Cohen, W. (2016). **Multi-task cross-lingual sequence tagging from scratch.** *arXiv preprint arXiv:1603.06270*.

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., & Hovy, E. (2016, June). **Hierarchical attention networks for document classification.** In Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies (pp. 1480-1489).

Zahui, A., Elhor, W., & Cheragui, M. A. (2017, May). **EL-Mossahih V1. 0: A hybrid approach for detection and correction of typographical and phonetic transcription errors in Arabic texts**. In 2017 8th International Conference on Information Technology (ICIT) (pp. 774-779).

Zerrouki, T., & Balla, A. (2017). **Tashkeela: Novel corpus of Arabic vocalized texts, data for auto-diacritization systems.** Data in brief, 11, 147.

Zhang, B., Xiong, D., Xie, J., & Su, J. (2020). **Neural machine translation with GRU-gated attention model.** IEEE transactions on neural networks and learning systems, 31(11), 4688-4698.

Zhou, J., Cao, Y., Wang, X., Li, P., & Xu, W. (2016). **Deep recurrent models with fast-forward connections for neural machine translation.** Transactions of the Association for Computational Linguistics, 4, 371-383.

Zitouni, I., Sorensen, J. S., & Sarikaya, R. (2006, July). **Maximum entropy based restoration of Arabic diacritics**. In Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics (pp. 577-584).

Zribi, C. B. O., & Ahmed, M. B. (2013). **Detection of semantic errors in Arabic texts.** Artificial Intelligence, 195, 249-264.

المصادر والمراجع

(1) عبد محسن حمد العامري. (2015). **الأخطاء الإملائية الشائعة لدى طلبة معاهد إعداد المعلمين والمعلمات.** مجلة كلية الاسلامية الجامعة، (33)، 445-474.

بحث في الشبكات العصبية المكررة (المشفر / فاك التشفير)

لتشكيل النصوص العربية وتصحيح الأخطاء الإملائية

إعداد

أحمد نزيه المجدوبه

المشرف

الأستاذ الدكتور غيث عبنده

ملخص

في الوقت الحاضر، بينما أثرت اللهجات المختلفة على جودة اللغة العربية، تزداد الحاجة إلى نماذج قادرة على تصحيح الأخطاء الإملائية وتشكيل النص العربي. يهدف هذا العمل إلى التحقيق في نماذج الشبكات العصبية المتكررة لوحدة فك التشفير والتشفير، لحل مشاكل التشكيل والكشف الإملائي وتصحيحها. التجارب المختلفة التي تم تطبيقها على نماذج متعددة، رشحت ثلاثة نماذج (نموذج المحول، نموذج تسلسل وحدة فك التشفير إلى تسلسل، نموذج تسلسل وحدة فك التشفير إلى تسلسل الانتباه). يعتمد نجاح هذه النماذج إما على استخدام طبقة التضمين في أجزاء كل من وحدة التشفير وفك التشفير من النموذج، أو استخدام آلية الانتباه كعمارة أساسية أو كطبقة انتباه للنموذج. أفضل نتيجة BLEU مسجلة في التشكيل هي 64.96، وفي تصحيح الأخطاء الإملائية هي 76.12 باستخدام مجموعة بيانات LDC ATB3. عند استخدام مجموعة بيانات Tashkeela، حققت درجة BLEU 61.02 في التشكيل، بينما في تصحيح الأخطاء الإملائية، كانت القيمة 63.97. يشير هذا العمل، اعتمادًا على نتائج درجة BLEU، إلى أنه يمكن استخدام النماذج المذكورة أعلاه لحل مشاكل التسلسل إلى التسلسل الأخرى. ستساهم نتائج هذا العمل بشكل فعال في زيادة فهم النصوص العربية من الناطقين باللغة العربية وغير الناطقين بها.