HYBRID APPROACH FOR AUTOMATIC ARABIC TEXT DIACRITIZATION USING RECURRENT NEURAL NETWORKS

By
Saba Amin Al-Qudah

Supervisor
Dr. Gheith Ali Abandah, Prof

This Thesis was submitted in Partial Fulfillment of the Requirements for the Master's Degree of Science in Computer Engineering and Networks

School of Graduate Studies
The University of Jordan

Dec, 2016
COMMITTEE DECISION

This Thesis (Hybrid Approach for Automatic Arabic Text Diacritization using Recurrent Neural Networks) was successfully defended and approved on ----------

Examination Committee	**Signature**
Prof. Gheith Abandah, (Supervisor) Prof. of Computer Architecture	----------
Dr. Iyad Jafar, (Member) Assoc. Prof. of Digital Image Processing	----------
Dr. Omar Al-Kadi (Member) Assoc. Prof. of Pattern Recognition	----------
Dr. Ali Al-Haj (Member) Assoc. Prof. of Digital Image Processing (Princess Sumaya University for Technology)	----------

DEDICATION

To my parents for their prayers and endless support

To my supporter and my beloved Mutaz

To my little girls Kindah and Ellen

To auntie Um Motasem

ACKNOWLEDGEMENT

I would like to thank my supervisor Prof. Gheith Abandah for his advices, suggestions, help and patience during our work on this thesis. I highly appreciate his efforts. He is an excellent mentor and he taught me how to do research correctly. My gratitude is beyond words.

I am grateful to Prof. Nizar Habash and Eng. Anas Shahrour for their help and cooperation while we were working on the data preparation stage.

List of Contents

List of Figures

List of Tables

List of Abbreviations

ASR	Automatic Speech Recognition
BAMA	Buckwalter Arabic Morphological Analyzer
BPC	Base Phrase Chunker
BPTT	Back Propagation Through Time
CA	Classical Arabic
CUDA	Compute Unified Device Architecture
DER	Diacritic Error Rate
DNN	Deep Neural Networks
DP	Dynamic Programming
DRNN	Deep Recurrent Neural Network
ECA	Egyptian Colloquial Arabic
FFN	Feedforward Networks
GPU	Graphics Processing Units
HMM	Hidden Markov Models
LDC ATB3	Linguistic Data Consortium's Arabic Treebank Part 3
LSTM	Long Short-Term Memory
MADA	Morphological Analysis and Disambiguation of Arabic
MSA	Modern Standard Arabic
NLP	Natural Language Processing
POS	Part of Speech
RNN	Recurrent Neural Networks
SVM	Support Vector Machines
TTS	Text-To-Speech
WER	Word Error Rate

HYBRID APPROACH FOR AUTOMATIC ARABIC TEXT DIACRITIZATION USING RECURRENT NEURAL NETWORKS

By
Saba Amin Al-Qudah

Supervisor
Dr. Gheith Ali Abandah, Prof

ABSTRACT

Arabic is a Semitic language and one of the oldest languages in the world. In this language, the letters of the same word can have many different diacritization marks and this leads to different words with different meanings. Also, the lack of these diacritics makes this language ambiguous for reading by non-native speakers and for processing by Arabic automated system such as Automatic Speech Recognition (ASR), and Text-to-speech (TTS).

Adding diacritics to Arabic text is an important step for Arabic Natural Language Processing (NLP) applications and many researchers have developed tools to automatically diacritize Arabic text because performing diacritization manually is inefficient and time consuming.

Many methods in the literature have been developed to solve the automatic diacritization problems using rule-based, statistical, and hybrid methods. The hybrid approach combines rule based approach with statistical approach in order to exploit the advantages of these two approaches and get better results.

One of the most important statistical approaches that are used to solve the diacritization problem is the sequence transcription approach that uses Recurrent Neural Networks (RNN) of deep bidirectional Long Short Term Memory (LSTM) architecture. In this thesis, we propose a hybrid approach to automatically diacritize Arabic text. We use a full morphological and syntactical analyzer called MADAMIRA which is one of the most popular tools in this field and we investigate the use of Compute Unified Device Architecture (CUDA) RecurREnt Neural Network (CURRENNT) library for solving a sequence labeling problems and to speedup RNN training. To our knowledge, CURRENNT is considered as the first publicly-available tool that has parallel implementation of deep bidirectional LSTM RNN architecture. Also, we study the effect of adding linguistic information to the input sequences of RNN at three different levels.

Our proposed system achieves state-of-the-art results over the best reported hybrid system. Using LDC's Arabic Treebank Part 3 corpus, we achieve a diacritic error rate of 2.39%, and a word error rate of 8.4%. When the case ending diacritics were not included, we achieve a diacritic error rate of 0.78%, and a word error rate of 2.3%. This approach reduces the diacritic error rate by 34% and the word error rate by 26% over the best published results.

CHAPTER I

Introduction

This chapter demonstrates Arabic diacritics, diacritization types, importance of Arabic text diacritization, research objectives and contributions and thesis outline.

1.1 Arabic Diacritics

The Arabic language consists of 28 basic alphabet letters and eight basic diacritization marks. Diacritization marks insert phonetic information to Arabic alphabet letters. The location of these marks could be above or below the letter. Diacritics can be classified into three categories: short vowels, nunation, and syllabification marks (Azmi and Almajed, 2013). Table 1 represents the eight basic diacritization marks. The first category consists of three short vowels that could be inserted on any constant of the word. The second category consists of three nunation diacritics or double case ending diacritics that only inserted on the last letter of the word. The third category consists of shaddah and sukon. Shaddah or germination diacritic can be joined with any other diacritics and is pronounced like consonant doubling. Sukon indicates that the letter does not have vowels. The fifth column represents Buckwalter transliteration, which is the standard encoding that represents Arabic characters and diacritics for computers (Habash, et al., 2007). For the Buckwalter transliteration codes of Arabic characters and diacritics, see appendix A.

Arabic language has two forms: classical and modern. The Classical Arabic (CA) denotes the pure language that is used in literary texts and Quran. The Modern Standard Arabic (MSA) is based on CA but with the addition of recent words to meet modern needs and challenges (Farghaly and Shaalan, 2009).

Table 1. The basic Arabic diacritization marks

Diacritic Types	Diacritic name	Diacritic shape	Pronunciation	Buckwalter Transliteration
Short vowels	Fatha	◌َ	/a/	A
	Damma	◌ُ	/u/	U
	Kasra	◌ِ	/i/	I
Nunation (Double case ending)	Tanween fath	◌ً	/an/	F
	Tanween damm	◌ٌ	/un/	N
	Tanween kasr	◌ٍ	/in/	K
Syllabification marks	Shaddah	◌ّ	consonant	~
	Sukon	◌ْ	vowel absence	O

1.2 Diacritization Types: Lexemic and Inflectional

Arabic is highly inflective and derivative language so that the same Arabic word can have different meanings and pronunciations depending on the related diacritics. The diacritics can be classified into two kinds (Habash and Rambow, 2007): lexemic diacritics and inflectional diacritics. Lexemic diacritics differentiate between lexemes that have same spelling, as shown in Table 2. The form (كتب) has nine possible diacritized dictionary forms and these lexemes have different meanings depending on the diacritics (Kirchhoff, et al., 2002). The first word (كَتَبَ) is an active verb and the second word (كُتِبَ) is passive verb and third word (كُتُب) is a noun and so on. These lexemic diacritics are morphology-dependent.

Table 2. The possible diacritized forms of the word (كتب)

#	script	romanized	meaning
1	كَتَبَ	kataba	he has written
2	كُتِبَ	kutiba	he had written
3	كُتُب	kutub	books
4	كَتْب	katb	a script
5	كَتَّبَ	kattaba	he made s.o. write
6	كُتِّبَ	kuttiba	to make s.o. write
7	كَتِّبْ	kattib	make s.o. write
8	كَتَبَّ	katabba	like slicing
9	كَتَبّ	katabb	like the slicing

Inflectional diacritics differentiate between different syntactic rules of the same lexeme and usually appears on the final consonant of the word. These diacritics are syntax-dependent. For example, Figure 1 represents the importance of the correct diacritization. In the first example, the syntactic diacritic of the word إِبْرَاهِيمَ is "Fatha" since its "object" in the parsing tree, while in the second example the same word إِبْرَاهِيمُ has "Damma" since its "subject" in the parsing tree. This example shows that incorrect syntactic diacritics on the last consonant of the words (ابراهيم) and (الناس) completely changed the meaning.

Ibrahim is afraid from people	أرهبَ **النَّاسُ إِبْرَاهِيمَ**
People are afraid from Ibrahim	أرهبَ **النَّاسَ إِبْرَاهِيمُ**

Figure 1. An example on the importance of inflectional diacritics

1.3 Importance of Arabic Text Diacritization

Arabic is the language of Quran and currently considered as the fourth most widely spoken language in the world. It is the official language of 58 countries, and the estimated number of Arabic speakers is 267 million (Lewis et al., 2016).

Diacritization is considered as one of the Arabic Natural Language Processing (NLP) applications in which researchers develop tools to diacritize the texts automatically because performing diacritization manually is inefficient and time consuming. However, Native speakers are able to restore diacritics of script based on the context and their understanding of the grammar and the lexicon of Arabic. The lack of diacritics results in a confusions for beginning readers, sufferers of dyslexia, and non-native speakers. Also, applications as Automatic Speech Recognition (ASR) and Text-to-speech (TTS) need diacritization to correctly process their data since these systems will not be able to know the exact meaning of undiacritized words. Moreover, machine

translation from and to Arabic needs diacritization to obtain the correct translation. Another reason for the diacritization is the information retrieval, when we search for an Arabic word we will have many irrelevant words in the search results (Azim and Almajed, 2013).

We note that most of Arabic text is written without diacritization marks and this leads in some times to ambiguity even for the natives because they need to consider the meaning through the context. Adding diacritization marks will remove the ambiguity of Arabic sentences (Azim and Almajed, 2013).

Making diacritization system for Arabic is a complicated task because Arabic language is highly inflectional and derivational. Text without diacritics can result in considerable ambiguity, for example: (عَقْد→ "contract", عِقْد → "necklace", عَقَّد → "complicate") (Said, et al., 2013). Also, the orthographic representation of the Arabic word is not enough to correctly pronounce the word so we need diacritics.

For all the reasons that we discussed above, we identified the importance of automatically adding diacritization marks to Arabic text.

1.4 Research Objectives and Contributions

The use of neural networks in this field can be considered as an emulation of the brain behavior of a native speaker when attempting to add diacritics to undiacritized text. The human brain is a Recurrent Neural Network (RNN) i.e. a network of neurons with feedback connections that can learn many tasks. This was our motivation to work on a novel approach that uses RNN to automatically adding diacritization marks to Arabic text (Pineda, 1987).

We can use RNN to make sequence transcription task and to map the input

sequences to output sequences. The network is trained under supervised learning where we give the input sequence, which is the undiacritized Arabic text, and the target sequence, which is the fully diacritized Arabic text. Once this is done, we can use RNN for diacritizing undiacritized text.

Using RNN approach is more powerful and biologically more reasonable than other statistical approaches such as Hidden Markov Model (HMM) (Gal, 2002) because it is unable to deal with complex memories of previous input, Feedforward Networks (FFN) or Support Vector Machines (SVM) (Burges, 1998) because they depend only on the current input. However, RNN benefits from the contextual information of the input sequences and it achieves a wonderful results when it's used in a sequence problems such as handwriting recognition (Abandah, et al., 2014) and speech recognition (Graves, et al., 2013).

Our novel approach is to use RNN to add the missing diacritics to Arabic text. The beginning of this work was when Abandah et al. (2015) proposed statistical approach that uses RNN to automatically add diacritics to Arabic text. To the best of our knowledge, this approach was the most accurate statistical approach reported. Then, they proposed a hybrid system (Arabiyat, 2015) that consists of a rule based approach, which uses Buckwalter Arabic Morphological Analyzer (BAMA) (Buckwalter, 2004), and the statistical approach that uses RNNLIB library (Graves, 2008). They achieved about 24% Diacritic Error Rate (DER) improvement and 15% Word Error Rate (WER) improvement over the best reported hybrid approach (DER and WER metrics are explained in details in Section 5.1). They used BAMA morphological analyzer but it did not implement the morphological segmentation in its best capability so we need more better rule based diacritization and tokenization approach and also RNNLIB library is slow so we can't train large data.

The main contribution of this thesis is to develop a hybrid system that improves partial diacritization and tokenization of data by using full morphological and syntactical analyzer like MADAMIRA morphological analyzer (Pasha, et al., 2014). There was cooperation with Prof. Nizar Habash (Nizar Habash's Home Page) and he gave us the required data after running MADAMIRA tool on it and also we used Compute Unified Device Architecture (CUDA) RecurREnt Neural Network (CURRENNT) library (Weninger, et al., 2015) instead of RNNLIB library since this library is fast because it uses CUDA and Graphics Processing Units (GPU).

The objectives and contributions of this project are summarized as follows:

1. Investigating the use of CURRENNT library, which is a recurrent neural network library for sequence labeling problems that supports GPUs through NVIDIA's CUDA, to automatically diacritize Arabic texts.
2. Building an accurate system to add all Arabic diacritics (fatha, kasra, damma, the three tanweens, shadda, and the sukoon) and comparing it with the best published results in this field.
3. Improving diacritization accuracy by combining rule based approach, which applies the full morphological and syntactical analysis, with CURRENNT statistical approach. And then using post-processing techniques to correct some issues of CURRENNT output.

Finally, we think that this thesis is an important addition to the field of automatic diacritization of Arabic text and provides a good approach for computer support of Arabic language.

1.5 Thesis Outline

The rest of this thesis is structured as follows. Chapter 2 represents a literature review of previous related approaches that were developed to solve diacritization problem. Chapter 3 explains the technologies used in our work including RNN that we used to solve diacritization problem. Also, MADAMIRA tool is described which is widely used and mentioned in previous works. Chapter 4 describes the methodology of our work which includes our workload, data processing, sequence transcription, and RNN training parameters. Chapter 5 describes the experiments that we do in this work, results of our proposed system, and a comparison with state-of-the-art systems is made. Finally, chapter 6 shows conclusions and suggestions for future work.

CHAPTER II

Literature Review

Automatic diacritization of Arabic text is an active area in the current NLP researches (Hifny, 2012). Most NLP applications, such as ASR, need fully diacritized texts for training phase. Also, the lack of diacritics will produce many irrelevant words and each word will have different meaning and different pronunciation. Even though, there are many diacritization techniques used to handle this problem, there still a room to enhance the accuracy of adding diacritics to Arabic words (Zayyan, et al., 2016). The techniques used in solving diacritization problem can be classified into three approaches: rule-based, statistical, and hybrid approaches (Azmi and Almajed, 2013). The following sections review the best approaches that were used in this field.

2.1 Rule-based Approaches

A rule-based approach was the first approach that used to solve this problem. It depends on morphological analyzers, dictionaries, and grammar rules. This approach gives good results when a strong linguistic knowledge is available (Azmi and Almajed, 2013).

El-Sadany and Hashish (1988) used a dictionary, analyzer, and a grammar module that consists of morphophonemic and morphographemic rules. El-lmam (2004) proposed a system that transcribes a letter to sound based on a set of Arabic dependent rules with a help of dictionary that consists of exception words. Shaalan (2010) proposed a tool that consists of Arabic morphological and syntax analyzer and this morphological analyzer depends on the augmented transition network technique to determine context relations between stems and affixes (prefix, suffix or both). However, the rule based approaches suffered from difficulty in dealing with up-to-date rules, the

existence of Arabic dialects, and new words are always appeared in living language (Azmi and Almajed, 2013).

2.2 Statistical Approaches

This approach doesn't need any linguistic knowledge or use of tools like morphological analyzer but needs a large and fully diacritized dataset (Azmi and Almajed, 2013).

Gal in (Gal, 2002) proposed a statistical approach that used the HMM to restore short vowel diacritics of Arabic texts. It depends on the contextual correlation between words and consists of hidden states which represent the diacritized words of the training corpus, and for each state there is observation which represents the undiacritized form of the word that exists in this state. Then Viterbi algorithm is used to find the most suitable diacritics for these observations. This model achieved a word accuracy of 86%. The errors of this model caused by the small corpus size since it used Qur'an as corpora.

Kirchhoff et al. (2002) focused on developing an automatic diacritizer for dialectal Arabic speech (Arabic oral conversations) since most of ASR focused on MSA. They used the LDC CallHome of Egyptian Colloquial Arabic (ECA) dialectal corpus. This corpus contains script-based transcription (without diacritic) and Romanized-based transcription (with diacritics). The result showed that training on Romanized transcription is significantly better than training on script transcription. They notice that MSA and ECA are completely different and the using of out-of-corpus text data on ECA was unsuccessful. They obtained a WER ranged from 9% to 28% on MSA text depending on whether case ending is including or not and a WER of 48% on ECA text.

Hifny (2012) proposed diacritization system that uses statistical n-gram language model, Dynamic Programming (DP) algorithm, and smoothing techniques. N-gram language model is used to add probability score for the diacritized Arabic word sentences of the undiacritized input sentence to distinguish these diacritized sentences. Dynamic programming algorithm is used to find the most possible diacritized sentence. Different smoothing techniques are used to solve the problem of unseen n-grams in the training data. Hifny used trigram language model and this didn't handle long linguistic dependency. This system was trained and tested on Tashkeela corpus which contains Islamic religious heritage books (Zerrouki, 2011) and achieved a WER of 8.9% when case endings were included and WER of 3.4% without case endings.

Abandah et al. (2015) proposed diacritization system that uses RNN statistical approach and depends on the deep bidirectional long short term memory architecture in order to deal with dependency between the words in long sentences and in both directions. It uses RNN sequence transcription to automatically add diacritics to Arabic text. To the best of our knowledge, this approach is the most accurate statistical approach reported. Also, it outperforms the most important hybrid approaches. However, the using of RNN in this field has never been proposed before. They achieved on LDC's Arabic Treebank Part 3 (LDC ATB3) corpus (Maamouri et al., 2004), a WER of 9.07% and a DER of 2.72% respectively and achieved on Tashkeela corpus, a WER of 5.82% and a DER of 2.09% respectively. This improvement in accuracy was achieved because of the big size of this corpus.

2.3 Hybrid approaches

This approach combines rule based approach with statistical approach. Vergyri and Kirchhoff (2004) examined the benefits of using several knowledge sources (acoustic, morphological, and contextual) to automatically diacritize Arabic texts and

the effect of their combination. Using BAMA all possible diacritization and morphological analyses of a given Arabic text were produced, then they trained unsupervised tagger to assign probability to all diacritized forms produced by this analyzer, and then the Expectation Maximization (EM) algorithm is used to learn the tag sequences. They used two different corpora, the FBIS corpus of MSA speech and the LDC CallHome ECA corpus because they examined the use of Arabic dialectal speech and MSA. They did not model the shadda diacritic and achieved WER of 27.3% and diacritization error rate of 11.5% when case endings were included

Nellken and Shieber (2005) proposed a new algorithm to restore diacritics using a probabilistic model defined as weighted finite state transducers and simple morphological model. Their basic model consists of cascade transducers. The Language Model (LM) learns the weights from diacritized words of the training set and used it to select the most probable undiacritized word sequence. The Spelling (SP) transducer divides the word into its component letters for the following transducer that operate on the letter based. The Diacritic Drop (DD) replaces diacritics with the empty string. The final transducer unknowns (UNK) to deal with the words that appeared in test input and were not existed in the training data. Then Viterbi decoding used to restore diacritics. Independence of the transducers was the problem of this approach. This system was trained and tested on LDC's Arabic Treebank of diacritized news stories (Part 2) and achieved a WER of 23.61% and a DER of 12.79% when case endings were included and without case endings, the results were 7.33% and 6.35% respectively.

Zitouni et al. (2006) proposed diacritization system that uses statistical model based on the maximum entropy framework which allows the system to used different sources of information such as lexical, segment-based, and Part of Speech (POS) features. Because they don't have a morphological lexicon, the segment based features

were generated by statistical Arabic morphological analysis using WFST approach and also the POS features were generated by a parsing model that depends on the maximum entropy. All these features are then combined in the maximum entropy framework to restore the missing diacritics. This system doesn't exploit long-range context dependencies since maximum entropy framework deals with each state independent of other states. Also it was trained and tested on the LDC ATB3 and achieved a WER of 18% and a DER of 5.5% when case endings were included and without case endings, the results were 7.9% and 2.5% respectively.

Habash and Rambow (2007) proposed diacritization system that consists of two subsystems: Morphological Analysis and Disambiguation of Arabic (MADA), and the standard n-gram language model. The MADA subsystem uses BAMA morphological analyzer to get all possible analyses of a word, then fourteen SVM predictors are used to narrow this list. The standard n-gram language model used to select one solution from the narrowed list. The best results they reported were by using trigram lexeme model. This approach is limited by depending on trigram language models that don't exploit long-range context dependencies. Also it achieved on LDC ATB3, a WER of 14.9% and a DER of 4.8% when case endings were included and without case endings, the results were 5.5% and 2.2% respectively.

Rashwan et al. (2011) proposed diacritization system that consists of unfactorized system (based on dictionary) and factorized system (based on morphological analyzer). This approach consists of two phases: off-line and on-line phases. In the off-line phase, a dictionary model and a statistical language model are built. In the on-line phase, the input words are searched in the dictionary. If the word is found, the dictionary will retrieve all its diacritized forms (lattice). This lattice is disambiguated to predict the most likely diacritics via A* lattice search algorithm and n-

gram probability estimation. Then it concatenated with the words that were not founded in the dictionary and passed to the factorizing module. This module factorized the words that were undiacritized in the first layer into its possible morphological components (prefix, root, pattern and suffix), and again uses n-gram probability estimation and A* lattice search algorithm. This approach is limited by depending on trigram language models to determine the most probable diacritics. Also this system achieved on LDC ATB3 v1.0, a WER of 12.5% and a DER of 3.8% when case endings were included and without case endings, the results were 3.1% and 1.2% respectively.

Said et al. (2013) proposed diacritization system that consists of three phases. In first phase, they used automatic corrector and tokenizer. In second phase, they used statistical and rule based analyzer to generate all possible morphological analysis for each input word. In third phase, they used POS tagger, which use HMM algorithm, to select the most suitable analysis based on context and to solve the ambiguity of last letter diacritic. Also, they used out of vocabulary diacritizer to diacritize the words that are not analyzed by the morphological analyzer like foreign names. This system achieved on LDC ATB3, a WER of 11.4% and a DER of 3.6% when case endings were included and without case endings, the results were 4.4% and 1.6% respectively.

Shahrour et al. (2015) proposed diacritization system that combines syntactic analysis with morphological tagging. Syntactic analysis helps in dealing with syntactic case diacritics on words final and morphological tagging worked on lexical diacritics that exist on the word stems. This approach shows the importance of using automatic syntactic analysis in improving morphological analysis and word diacritization, since it provides better prediction of case and state features using statistical parsing and manual syntactic rules. They used the Penn Arabic Treebank (PATB, parts 1, 2 and 3), and for accuracy they used two metrics: Diac (percentage of correctly fully diacritized words)

and ALL (percentage of correctly prediction for full morphological analysis of words) and reported the result on ALL Words and Nominals. This approach achieved Diac accuracy improvement on ALL Words by 2.5% absolute and on Nominals by 5.2% absolute.

Arabiyat (2015) proposed a hybrid approach that combines a rule based approach with Abandah et al. (2015) statistical approach. They used BAMA morphological analyzer to provide morphological analysis and partial diacritization of the data. Then they used this data for RNNLIB, which is a recurrent neural network library that used to solve sequence labeling problems (Graves, 2008) and is implemented using deep bidirectional Long Short-term Memory (LSTM) cell. Also preprocessing corrections techniques were used to solve some issues that appeared when using BAMA and post processing corrections techniques were applied to the RNN results in order to improve the accuracy result. This hybrid approach provides the best results compared to all state-of-the-art hybrid diacritization models and achieved on LDC ATB3, a WER of 9.66% and a DER of 2.74% when case endings were included and without case endings, the results were 3.95%and 1.24% respectively. In this approach, BAMA did not implement the morphological segmentation in its best capability so they suggest the using of POS tagger to choose one specific analysis of BAMA solutions based on context.

Zayyan et al. (2016) proposed diacritization system for MSA text that combines multi-lexical level statistical approach with knowledge-based approach. There are three lexical levels: word level, morphemes level, and letter level. In word level, they use four statistical n-gram models: four-gram, tri-gram, bigram, and unigram models. The first three models consist of two sub-models: right-context, used to improve diacritization accuracy of the word by considering the previous history of the given word and left-

context, will consider the next words of the given word (the number of previous and next words depend on n-degree). In morphemes level, they select prefix and suffix depending on the largest matching number of letters and also they used the four statistical n-gram models and the sub models on the morphological units. In letter level, each letter in the word is considered as single units and they used the four statistical n-gram models and the sub models on letters units. They used the Nemlar written corpus and Le Monde Diplomatic corpus from European language resources association; these two corpora are constructed from different articles of different fields like news, sports, scientific etc. also, these two corpora are developed using automatic and manual reviewing techniques which give 99% diacritization accuracy for these corpora. When they combine the multi-lexical models, this system achieved a WER of 7.1% and a DER of 3.9% when case endings were included and without case endings, the results were 5.1% and 2.7% respectively.

Metwally et al. (2016) proposed diacritization system that consists of three layers. The first and second layers predict the morphological diacritics while the third layer predicts the syntactical diacritics. The first layer uses first-order HMM to select the best probable sequence that contains the morphologically diacritized words with their POS tags but HMM can handle only the previously seen words so unseen words will be undiacritized and they solved this problem using second layer. The second layer uses Standard Arabic Morphological Analyzer (SAMA) to consult it regarding the possible analysis of the undiacritized word of the first layer. SAMA generates a list of the possible analysis and each analysis contains one possible diacritization with its POS tag and they take the first analysis of SAMA. The third layer applies the Conditional Random Fields (CRF) (Lafferty, et al., 2001) to add for every word one syntactic

diacritic. This system achieved on LDC ATB3, a morphological WER of 4.3% and a syntactic WER of 9.4%.

In this thesis, we proposed a hybrid approach to automatically diacritize Arabic text; we used a full morphological and syntactical analyzer with CURRENNT library. We gain amazing results and they are reported in Chapter 5.

CHAPTER III

Technologies Used

3.1 Recurrent Neural Networks (RNN)

Human brain spread the stored information over many neurons and when one attempts to memorize something, this information will be collected from all these neurons. Scientists benefit from this and emulate human brain using artificial neural networks. This solved many problems and mainly the problems that need learning and decision making.

Using neural networks in NLP field is rare (Khedher, 1999). However, many researchers get amazing results when using neural networks (Jamro, et al., 2016). Khedher (1999) wrote these beautiful sentences: “The main reason why it seems that the Arabic text processing seems to be suitable for neural network application is that people from their early age are trained to talk properly. Why neural networks cannot be trained similarly? Of course, proper and enough data is necessary”, “The Arabic language as one of the most advanced living languages, can utilize neural networks techniques in many aspects” and “It is strongly recommended to consider seriously the use of neural networks in research of Arabic language processing”.

There are two types of learning algorithms for the neural network: supervised and unsupervised learning (Svozil, et al., 1997). In this work, we use supervised learning algorithm since it can conclude a general function based on the training data and it will be able to test data in a reasonable way. Our training data is part of LDC ATB3 benchmark and consists of a set of training examples. Each example consists of

two fields: input and target fields. Input field sentences are generated after removing diacritics from the same sentences of the target field. The supervised learning algorithm depends on training sentences to conclude a general function that can be used in the new sentences. The importance of inferring a general function by the supervised learning algorithm is when the test set is constructed from new data that is not founded in the train set. Validation set is used to validate the performance of the learning algorithm during training data and the training is stopped based on the validation error rate to avoid overfitting. Model overfitting occurs when training error rate decreases and testing error increases and this may happened due to noise or when the number of representative samples in the training set is few (Tan, et al., 2005).

In this work, we use sequence transcription approach to automatically diacritize Arabic text. RNN will transcribe the input sequence of MADAMIRA to produce a fully diacritized output sequence. RNNs are used in solving diacritization problem since they use the past history of input sequences, which mean that they will benefit from the context of input sequences when mapping input sequence to output sequence (El Hihi and Bengio, 1995).

3.1.1 Feedforward neural network vs. Recurrent neural networks

The feedforward neural network is artificial neural network where the data is transferred from input layer to the output layer in only one direction and it will pass through the hidden layers (if exist). Input layer contains one neuron per feature and the output layer contains one neuron per class. This network doesn't have any cycles or loops as shown in Figure 2. Backpropagation algorithm is used in training this network and to update the weight of this neural network by computing the difference between

the desired and actual outputs and then these differences will propagate back to the input layer (Graves, 2008).

For the standard feedforward network, we can compute the output vector sequence *y = (y1, . . . , yT)* when we have the input sequence *x = (x1, . . . , xT)* by using the following equations:

$$h_t = H(W_{ih}x_t + b_h) \quad (1)$$

$$y_t = (W_{ho}h_t + bo) \quad (2)$$

where the *W* terms denote weight matrices (e.g., W_{ih} is the input-hidden weight matrix), the *b* is bias vectors (e.g., b_h is hidden bias vector), and *H* is the hidden layer activation function and in most case its sigmoid function (Graves, et al., 2013).

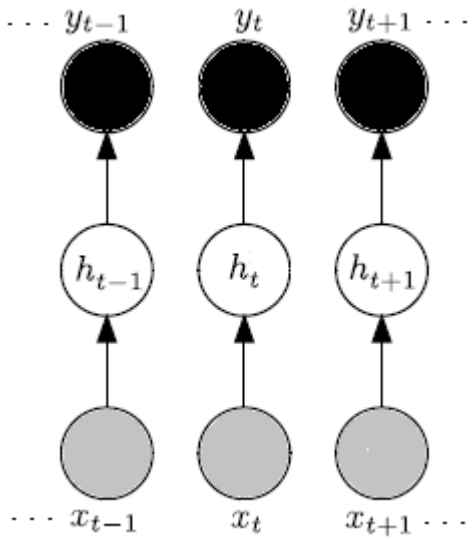

Figure 2. A simple feed forward neural network

The Recurrent neural network (RNN) is a type of artificial neural network and it works on the current inputs and on the previous history of the hidden layer for each time step. This network has cycle unlike feedforward neural network. Back Propagation

Through Time (BPTT) is a gradient technique used for training RNN and the gradient of error function is computed using current and previous inputs (Graves, 2008).

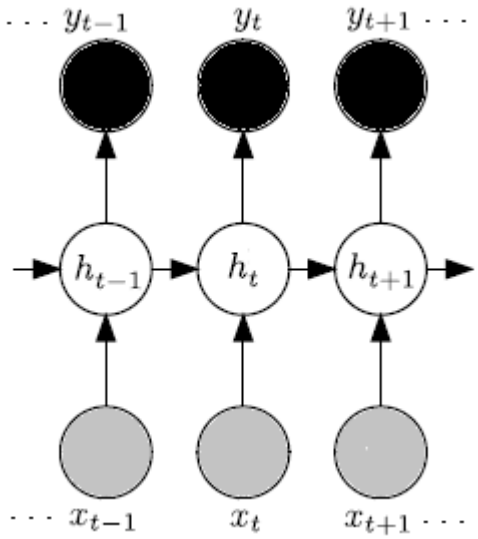

Figure 3. A simple recurrent network

Figure 3 represents a simple recurrent network which is called Elman network (Elman, 1990). It is designed to learn sequences by the addition of "context units" which are external input that used to update the hidden layer. So the hidden layer will be activated by the input and the "context units". Then the hidden layer will feed forward to activate the output layer. Also the hidden layer will feed back to activate the "context units".

We compute the output layer, *y*, by using the following equations (Abandah, et al., 2015):

$$h_t = \mathcal{H}\left(W_{ih}\, x_t + W_{hh}\, h_{t-1} + b_h\right) \tag{3}$$

$$y_t = W_{ho} h_t + b_o \tag{4}$$

RNNs architecture, which can make sequence learning task, get state-of-the-art results in a handwriting recognition (Graves et al., 2008), text generation (Sutskever et al., 2011) and language modelling (Mikolov et al., 2010).

3.1.2 Long-Short Term Memory (LSTM)

The long short-term memory is used to solve the vanishing gradient issue that appeared when using recurrent networks. To solve this issue, they used memory cells to store information instead of using neurons (Hochreiter and Schmidhuber, 1997). LSTM architecture, that uses memory cells to store information, is better in accessing and dealing with long-range context data. LSTM has made state-of-the-art results in solving sequence processing problems such as speech recognition (Graves, et al., 2013).

Long short-term memory cell consists of self-connections memory cells and three multiplicative units or gates (Input gate, forget gate and output gate) as shown in Figure 4. Input gate is used to store or write information. Output gate is used to retrieve or read information. Forget gate is used to forget or reset operations for the cell. So these gates control the flow of information over the cell. The gradient will not vanish because the self-connection has a value of one and the memory cell will keep remembering the first input as long as the input gate is closed and the forget gate is opened (Graves and Schmidhuber, 2005).

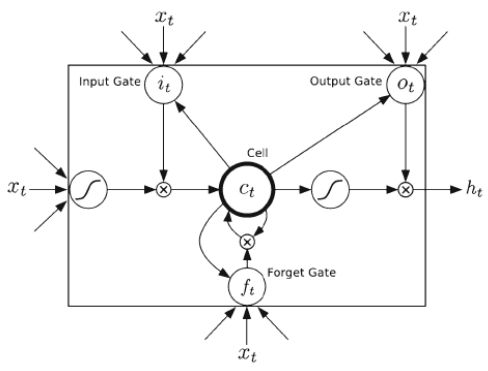

Figure 4. Long short-term memory cell

3.1.3 Bidirectional Recurrent Neural Networks (BRNN)

RNNs show their ability to deal with sequential data that has dependency between data points and these data points are close to each other (Robinson, 1994). In general, RNNs architecture receive one input vectors *Xt* at a time and can predict Y_{tc} based on the use of all the available input data until the current time t_c (i.e. *Xt, t=1,2,…, tc*) .

RNN deals with the future input information that coming after *tc* by delaying the output to a certain amount of *G* time frames up to *Wtc+G*. Theoretically, *G* could be very large since it must deal with all the available future information, but in practice, if *G* is too big the prediction results will be dropped (Robinson, 1994).

Bidirectional Recurrent Neural Network (BRNN) solves this issue of RNN since it can deal with the past and the future input information. The structure of BRNN is achieved by splitting the state neurons of RNN into two parts; the first part is forward states and it is used for the positive time direction i.e. from t=1 to T and the second part is the backward states and it is used for the negative time direction i.e. from t=T to 1. There are no connections between the outputs of the forward states and the inputs of backward states. Also, the data of both states will deliver to the same output layer as shown in Figure 5.

The forward pass of BRNN structure is the same as RNN but with some differences that the input sequence is given in opposite directions to the two hidden layers and the output layer will change only if the hidden layers of both directions have processed all input sequence (Schuster and Paliwal, 1997).

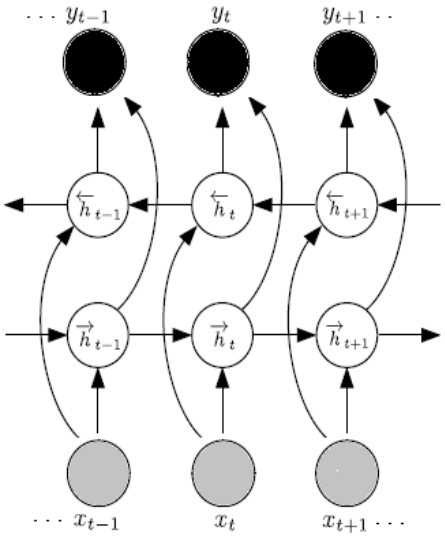

Figure 5. Bidirectional Recurrent Neural Network

We compute the output sequence *y* of BRNN by iterating the backward layer from *t=T* to *1*, the forward layer from *t=1* to *T*, and then updating the output layer:

$$\overrightarrow{h_t} = \mathcal{H}(W_{x\vec{h}}\, x_t + W_{\vec{h}\,\vec{h}}\overrightarrow{h_{t-1}} + b_{\vec{h}}) \tag{5}$$

$$\overleftarrow{h_t} = \mathcal{H}\left(W_{x\overleftarrow{h}}\, x_t + W_{\overleftarrow{h}\overleftarrow{h}}\overleftarrow{h_{t+1}} + b_{\overleftarrow{h}}\right) \tag{6}$$

$$y_t = W_{\vec{h}y}\overrightarrow{h_t} + W_{\overleftarrow{h}y}\overleftarrow{h_t} + b_o \tag{7}$$

where $\overrightarrow{h_t}$ denotes the forward hidden sequence, and $\overleftarrow{h_t}$ denotes the backward hidden sequence (Abandah, et al., 2015) (Graves, et al., 2013).

BRNN architecture will help in automatic diacritization task since the whole sentence will transcribe and this architecture can deal with the input sequences in both directions so the output sequence will depend on the whole input sequences (Abandah, et al., 2015).

3.1.4 Deep recurrent neural network

Deep neural networks (DNN) mean that the networks consist of several hidden layers. Each layer is concerned to solve part of the problem and the output of one layer is the input to the next layer as shown in Figure 6. The features of higher layers will be complicated because they are building from lower layers. Also the final layer will be responsible for the final output (Hermans and Schrauwen, 2013). DNNs have proved their capability to solve sequence problems, for examples: speech recognition (Graves, et al., 2013) and handwritten digit recognition (Ciresan, et al., 2010) and also get state-of-the-art results in solving many problem in natural language processing field, for examples: information retrieval (Huang, et al., 2013) and machine translation (Cho, et al., 2014).

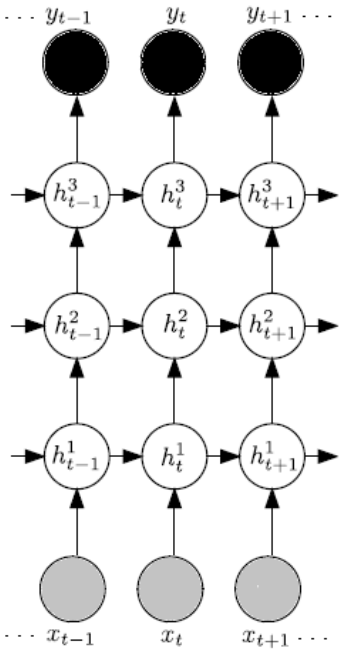

Figure 6. Deep RNN

DNN can deal with complicated and non-linear relationships and its hierarchical architecture can solve RNN problem where the information moves through only one layer to get the final results. Deep Recurrent Neural Network (DRNN) means that the

networks consist of several hidden layers and each layer is a recurrent neural network. The higher layer of DRNN can deal with complicated data with fewer units since the features of higher layer will be constructed from input features of lower layers (Hermans and Schrauwen, 2013).

The hidden vector sequences h^n and the network outputs y_t are calculated as shown below, Assuming that all N layers in the stack have identical hidden layer function.

$$h_t^n = \mathcal{H}\left(W_{h^{n-1}h^n} h_t^{n-1} + W_{h^n h^n} h_{t-1}^n + b_h^n\right) \quad (8)$$

Where $h^o = x$. The network outputs y_t are

$$y_t = W_{h^N y} h_t^N + b_o \quad (9)$$

In this work, we use deep Bidirectional LSTM architecture i.e each hidden layer is LSTM nodes and every hidden layer takes its input from both forward and backward layers of lower level (Abandah, et al., 2015). This architecture is shown in Figure 7.

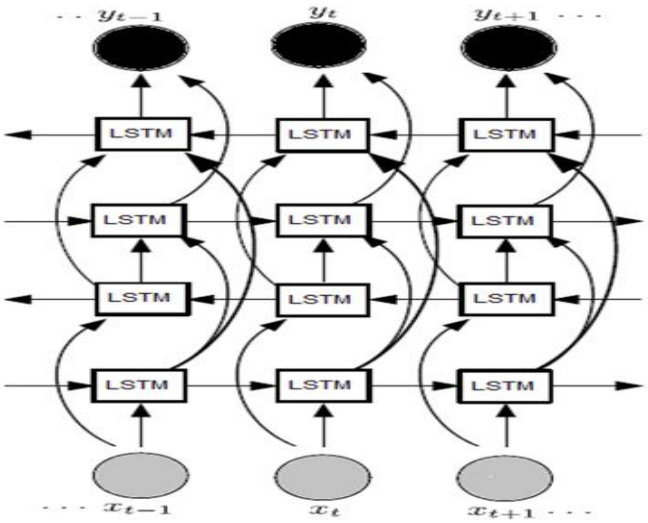

Figure 7. Deep bidirectional LSTM architecture

3.2 Some of the Most Important Arabic Morphological Analyzers

In this section, i will present the meaning of some linguistic features that we may use in this thesis and then i will present some of the most important Arabic morphological analyzer that was referenced in the literature such as Buckwalter Arabic Morphological Analyzer (BAMA), Morphological Analysis and Disambiguation of Arabic (MADA), and MADAMIRA (combines MADA and AMIRA toolkits) that we used in this work.

3.2.1 Linguistic Features of The Morphological Analyzer

Morphological analyzers, such as BAMA and MADA, can provide all possible morphological information for each input word. Each analysis consists of a set of linguistic features. I want to explain the meaning of some linguistic features that we may use in this thesis such as Part of Speech (POS), Gender, Number, Voice, Base Phrase Chunker (BPC), Case, pattern, Stem, Clitics. POS means the syntactic location of the Arabic word such as verb, noun, adjective, and so on. Gender means Arabic word is used for male or female. Number means Arabic word is singular, dual or plural. Voice means Arabic word is active or passive. BPC which determines the base phrase of the word that exists in the sentence, for example: verbal and nominal phrase. Case is a specific feature for the nouns and it means that the word can be genitive or normative and so on. Pattern is a model that is used to build the stems from root by adding some of pattern templates and will give different words with different meaning based on the used templates. The three basic characters of pattern are (f ف) is used for first letter of root, (E ع) is used for second letter of root, and (l ل) is used for third letter of root. For example: (k t b) ك ت ب is a root and the stems will be generated after adding pattern such as write (كَتب), writer (كاتب), and books (كُتب) and so on (Azmi and Almajed, 2013).

Clitics are morphemes that indicate grammatical information such as "بمدارسهم" 'with their schools', the stem is "مدارس" 'schools', the proclitic is 'b'ب, and enclitic is 'their' "هم" (Boudchiche, et al., 2016) .

3.2.2 Buckwalter Arabic Morphological Analyzer (BAMA)

BAMA morphological analyzer generates all possible diacritized and morphological analysis solutions for every word. It provides only the lexemic diacritics since it can't predict the inflectional diacritics. BAMA consists of three components: the lexicon, the compatibility tables and the analysis algorithm. It has three separate lexicons for prefixes, stems, and suffixes. Each word in these lexicons contains undiacritized and diacritized forms, its morphological category, and its English meaning. Also, it has three compatibility tables: Arabic stems, Arabic prefixes, and Arabic suffixes, these three tables are associated with the three lexicons to make stems, prefixes, and suffixes compatible (prefix-stem, stem-suffix and prefix-suffix) in order to determine which morphological categories are permitted to occur. In the analysis algorithm, Perl code uses the three lexicon files and the three compatibility tables in order to generate all possible diacritized and morphological analysis solutions for every word (Farghaly and Shaalan, 2009). BAMA may give wrong analysis due to Arabic proper names or transliterated foreign names that are not listed in the lexicons. Figure 8 shows BAMA analysis for the word يكتبون with three different meanings and pronunciations.

```
INPUT STRING: يكتبون
LOOK-UP WORD: yktbwn
  SOLUTION 1: (yakotubuwna) [katab-u_1 ]
ya/IV3MP+kotub/VERB_IMPERFECT+uwna/IVSUFF_SUBJ:MP_MOOD:I
    (GLOSS): they (people) + write + [masc.pl.]
  SOLUTION 2: (yukotabuwna) [katab-u_1 ]
yu/IV3MP+kotab/VERB_IMPERFECT+uwna/IVSUFF_SUBJ:MP_MOOD:I
    (GLOSS): they (people) + be written/be fated/be destined + [masc.pl.]
  SOLUTION 3: (yukotibuwna) [>akotab_1 ]
yu/IV3MP+kotib/VERB_IMPERFECT+uwna/IVSUFF_SUBJ:MP_MOOD:I
    (GLOSS): they (people) + dictate/make write + [masc.pl.]
```

Figure 8. BAMA solutions for the word **يكتبون**

3.2.3 Morphological Analysis and Disambiguation of Arabic (MADA)

MADA is a toolkit for Arabic tokenization, diacritization, morphological disambiguation, POS tagging, and stemming. It was built for MSA and later MADA-ARZ version was built for Egyptian Arabic. For each input word, a list of every possible morphological interpretation is produced. Then it made a prediction for each word based on the context using SVMs, used to classify the word and to predict a certain morphological features for each word as POS, and using N-gram language models (Pasha, et al., 2014). Each analysis of MADA consists of the diacritized form, its morphological features, and its English glossary. Also, it is constructed based on BAMA database and it makes all of the morphological ambiguity, tokenization, diacritization and POS in one stroke (Habash and Rambow, 2005). Also MADA uses a disambiguation module to specify the correct POS tag in a specific text (Farghaly and Shaalan, 2009).

3.2.4 MADAMIRA (combines MADA and AMIRA toolkits)

MADAMIRA is a fast and comprehensive tool for morphological analysis and disambiguation of Arabic that combines MADA and AMIRA. MADAMIRA is the new version of MADA; this version is more robust, portable, easy to use and maintain, and

faster. It is implemented in Java, which provides much more speed than Perl and allows new features to include with the existing code, also provides XML and HTTP support that did not exist in MADA or AMIRA. The AMIRA toolkit includes a tokenizer, POS tagging, and BPC. AMIRA and MADA are implemented in Perl. AMIRA uses a multi-step approach while MADA treats most of morphological interpretation in one stroke. Also MADA provides a deeper analysis and slower speed than AMIRA. AMIRA provides additional utilities as BPC that is not supported by MADA. MADAMIRA follows the same general design as MADA with some additional components from AMIRA (Pasha, et al., 2014). The MADAMIRA online web (Pasha, et al., 2014) includes four tab panels: diacritized forms, tokenized forms, parts-of-speech, and lemmas. For each input word, a box displays all the morphological features for that word including POS, case, gender, number, and gloss. For Arabic word سياسه (seyassah), it displays POS: noun, case: genitive, gender: feminine, number: singular, and gloss: politics as shown in Figure 9.

Figure 9. The MADAMIRA online web displays all the morphological features of Arabic word سياسه (seyassah)

3.3 GPU, CUDA and CURRENNT

In this section, i will present introduction about GPU and CUDA and then i will explain CURRENNT library that we use to transcribe sequences.

3.3.1 Graphics Processing Units (GPU) and Compute Unified Device Architecture (CUDA)

NVIDIA is the name of the company which invents GPU and CUDA. It considered as one of the first and most important gaming hardware manufacturers. GPU is a specialist processer or accelerator card exists in most modern pc and designed to speed up computations by doing parallel tasks at same time or accelerate the same task if it can be done in parallel and also it runs the sequential part of task on CPU. The GPU architecture is built in a suitable way for doing data parallelism.

CUDA is a programming language used to program GPU and it is C/C++ Application Programming Interface (API) that accelerates code on NVIDIA's GPU. This language works by writing functions for GPU which is called kernels that can be executed in parallel and for several times on all parameters that need this kernel. CUDA is designed to run on parallel architecture and to execute large numbers of threads and this threading model makes it perfect for solving parallel problems because it splits problem into grid of block and each block consists of multiple threads and can run in any order (Cook, 2012).

3.3.2 CUDA RecurREnt Neural Network (CURRENNT)

In this work, we do our experiments by using the open source software library CURRENNT which is CUDA enabled recurrent neural network library used for labeling sequential data and it supports the parallel implementation of deep LSTM RNN

architecture by using NVIDIA's GPUs to accelerate the training phase of RNN. Since CURRENNT supports deep bidirectional LSTM architecture it will be perfect choice for our experiment. To our knowledge, CURRENNT is considered as the first publicly-available tool that has parallel implementation of a deep LSTM RNN architecture. This indicates that we can speedup the training of Bidirectional LSTM RNN architecture. The problem of training RNN was occurred due to its inefficient implementation because of the limited parallelism that resulting from the time dependencies. CURRENNT benefits from mini-batch learning to make parallel weight update of all sequences and also it supports classification and regression tasks (Weninger, et al., 2015).

CURRENNT has two options "parallel sequences and stochastic". The option "parallel sequences" is used to speed up computations by splitting the whole data set into fractions (mini-batches) and then processed the sequences within mini-batch in parallel. The option "stochastic" controls whether weight updates will be after each mini-batch, or after full batch mode, or for online learning. Also, CURRENNT automatically stops the training when the validation set error does not improve for a number of epochs in order to prevent overfitting.

The parallel implementation of Deep LSTM-RNNs with N layers architecture that supported by CURRENNT is explained below (Weninger, et al., 2015). Given an input sequence x_t, we compute the output sequence y_t by iterating the following equations in forward pass (from t = 1 to T):

$$\mathbf{h}_t^{(0)} := \mathbf{x}_t, \tag{10}$$

$$\mathbf{h}_t^{(n)} := \mathcal{L}_t^{(n)}\left(\mathbf{h}_t^{(n-1)}, \mathbf{h}_{t-1}^{(n)}\right), \tag{11}$$

$$\mathbf{y}_t := \mathcal{S}\left(\mathbf{W}^{(N),(N+1)}\mathbf{h}_t^{(N)} + \mathbf{b}^{(N+1)}\right) \tag{12}$$

where **W** is the weight matrices and **b** is the bias vectors (the superscripts used for layer indices) and $\mathbf{h}_t^{(n)}$ is the hidden feature representation of *t* time frame and in the level *n* units (*n* start from 1 to N). The input layer is the 0-th layer and the output layer is the *N* + 1-th layer. *S* is the output layer function and $\mathbf{L}_t^{(n)}$ is the composite LSTM activation function. Then each unit has a state variable $\boldsymbol{c}_t$ and the hidden layer activations for the state variables is scaled by $\mathbf{o}_t^{(n)}$ which is the activations of the output gates:

$$\mathbf{h}_t^{(n)} = \mathbf{o}_t^{(n)} \otimes \tanh(\mathbf{c}_t^{(n)}), \quad (13)$$

$$\mathbf{c}_t^{(n)} = \mathbf{f}_t^{(n)} \otimes \mathbf{c}_{t-1}^{(n)} + \mathbf{i}_t^{(n)} \otimes \tanh\left(\mathbf{W}^{(n-1),(n)}\mathbf{h}_t^{(n-1)} + \mathbf{W}^{(n),(n)}\mathbf{h}_{t-1}^{(n)} + \mathbf{b}_c^{(n)}\right) \quad (14)$$

where $\otimes$ is element-wise multiplication and the state is scaled by a forget gate (Gers et al., 2000) with dynamic activation $\mathbf{f}_t^{(n)}$. $\mathbf{i}_t^{(n)}$ is the activation of the input gate that adjust the flow of the feedforward and recurrent connections. There are dependencies between layers and time step as shown above so the parallel computation of feedforward activations cannot be done across layers and the parallel computation of recurrent activations can't be performed across time steps. To be able to increase the degree of parallelism, they introduce data fractions of certain size *P* from *S* sequence and these data fractions will work in parallel and each one have *T* time steps. State matrix $\mathbf{C}^{(n)}$ of n-th layer is shown below:

$$\mathbf{C}^{(n)} = [\mathbf{c}_{1,p}^{(n)} \cdots \mathbf{c}_{1,p+P-1}^{(n)} \cdots \mathbf{c}_{T,p}^{(n)} \cdots \mathbf{c}_{T,p+P-1}^{(n)}] \quad (15)$$

where $\mathbf{c}_{t,p}^{(n)}$ refers to the state of sequence *p* in layer *n* at time *t*. To do the feedforward computation for all time steps and *P* sequences in parallel, we can do pre-multiplication with $\mathbf{W}^{(n-1),(n)}$. To do the recurrent computation, we can use $\mathbf{W}^{(n),(n)}$ to update $\mathbf{C}^{(n)}$ from

left to right. This matrix structure gives memory locality for the data of one time step and for bidirectional layers we will repeat this matrix structure for each layer.

In training, the backward pass for the hidden layers is occurred by dividing the matrix of weight changes into a part that will propagate to the previous layer and a recurrent part that will propagate to the previous time step; this will make a parallel implementation of the backpropagation through time (BPTT) algorithm (Weninger, et al., 2015).

Our training data consists of a set of training examples and each example is a pair of sequences (undiacritized sentence, diacritized sentence) and the network was trained to be able to classify each input char with the corresponding diacritized version. A softmax output layer is the best choice to do classification tasks because it ensures that the sum of output values equal to one since it defines a probability distribution over all possible output characters, and the network was trained to minimize the cross-entropy of this distribution with the target labels.

Our neural network has the following structure:

- Input layer: the number of input neurons determine by the number of input class.
- Hidden layers: we did many experiments with different number of layers to determine the optimal number of hidden layers. The type of this layer was bidirectional LSTM
- Feed forward output layer: which use softmax activation function. The number of output neurons determine by the number of target class.

- Post output layer: which is the last layer in the network and used to evaluate the cross entropy objective function during forward pass and give the error to output layer during backward pass. The size of this layer determines by the number of target class and the type of this layer is multiclass classification.

The network was trained using mini-batch steepest descent learning i.e. weight updates after every fractions (the number of parallel training sequences that we used in this work is 30) and with momentum 0.9 and learning rate of 0.0001 and the weights were initialized with normal distribution that have mean =0 and sigma=0.1. The GPU we work on is NVIDIA GTX 580 with 1.5 GB of RAM and the GPU specification (Nvidia Geforce Page) is shown in Table 3. The training was stopped at the lowest validation error rate.

During training, each sequence is given to this network to give a prediction for each character and the errors are back-propagated to update the weight of each layer. During testing, the trained network (the learned weights) and the test sequences are used in a forward pass mode in order to produce a prediction over the labels.

Table 3. GPU specification of NVIDIA GTX 580

GPU Engine Specs	
CUDA Cores	512
Graphics Clock (MHz)	772 MHz
Processor Clock (MHz)	1544 MHz
Texture Fill Rate (billion/sec)	49.4
Memory Specs	
Memory Clock	2004 MHz (4008 data rate)
Standard Memory Config	1536
Memory Interface	GDDR5
Memory Interface Width	384-bit
Memory Bandwidth (GB/sec)	192.4

CHAPTER IV

Methodology

4.1 Introduction

For complete diacritization, the diacritization process of Arabic text can be classified into two types: morphological and syntactic diacritization (Metwally, et al., 2016). Our diacritization system consists of two stages: first, we used MADAMIRA morphological analyzer to extract some of the morphological and syntactical diacritics. Then, we will use this data for CURRENNT statistical approach to predict the rest of diacritics so we will produce a fully diacritized text.

Our proposed diacritization system has two phases to produce diacritized text: training and production phases as shown in Figure 10 and 11. In training phase, we make data encoding in order to have a suitable data format i.e. we have two fields: input sentences which are undiacritized and target sentences which are diacritized. Then, we used MADAMIRA morphological analyzer that can produce the morphological analysis and the segmentation for each word in the sentence. So, we will get partially diacritized and partially tokenized data (the word consists of prefix, stem, and suffix). Then we make text corrections, to fix some words of MADAMIRA output and to prepare the data for RNN sequence transcription stage. After that, we do four RNN experiments to transcribe the input sentences of MADAMIRA and then RNN is trained to predict the rest of diacritics. In production phase, we apply the same steps but we use the testing data and we use the trained RNN networks to predict diacritics. Then we do Post processing corrections to enhance the accuracy of RNN results. We use these two metrics to calculate diacritization accuracy: DER and WER, since state-of-the-art approaches use these metrics and we can compare our result to them (explained in Section 5.1).

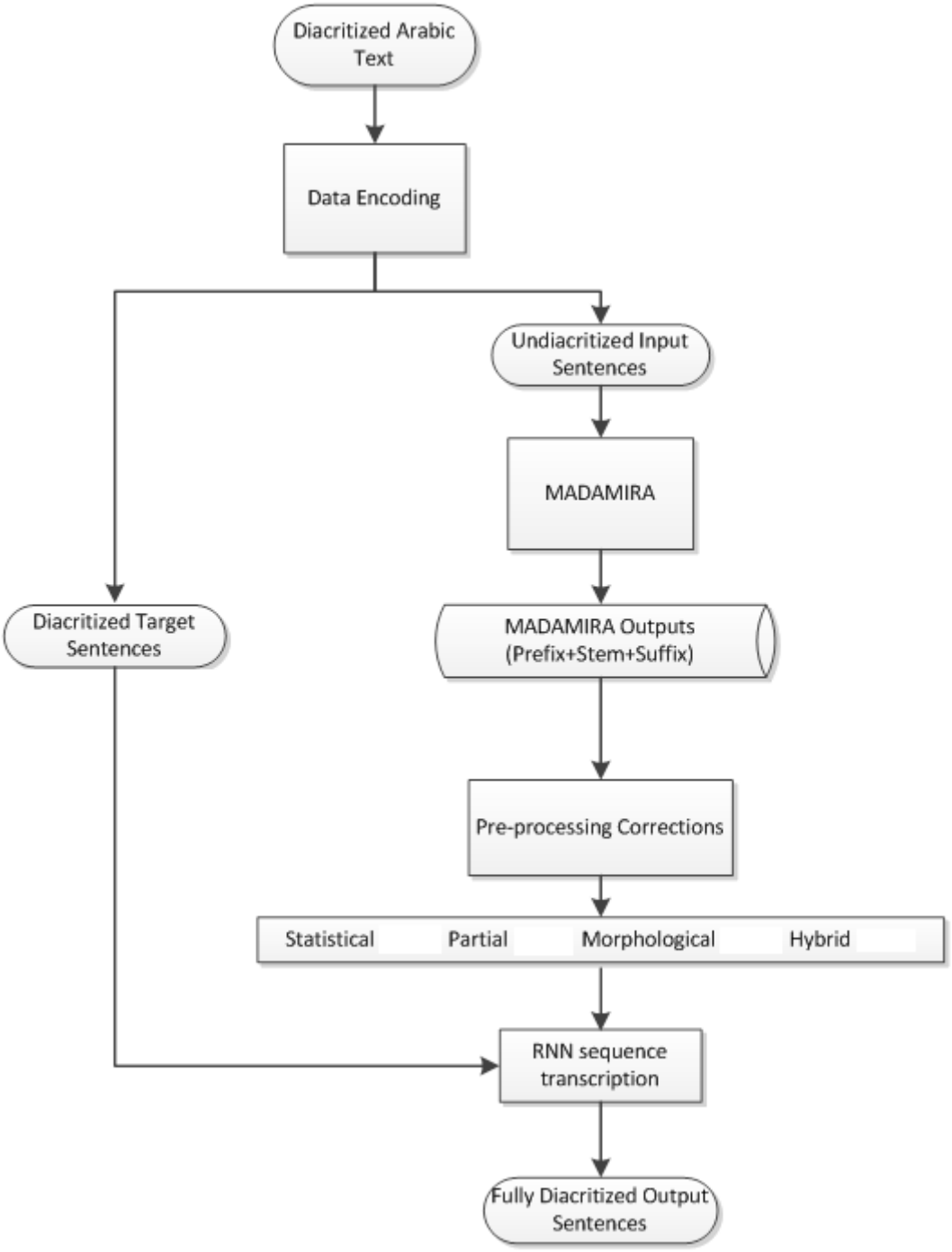

Figure 10. The schematic diagram of the proposed diacritization system (Training phase).

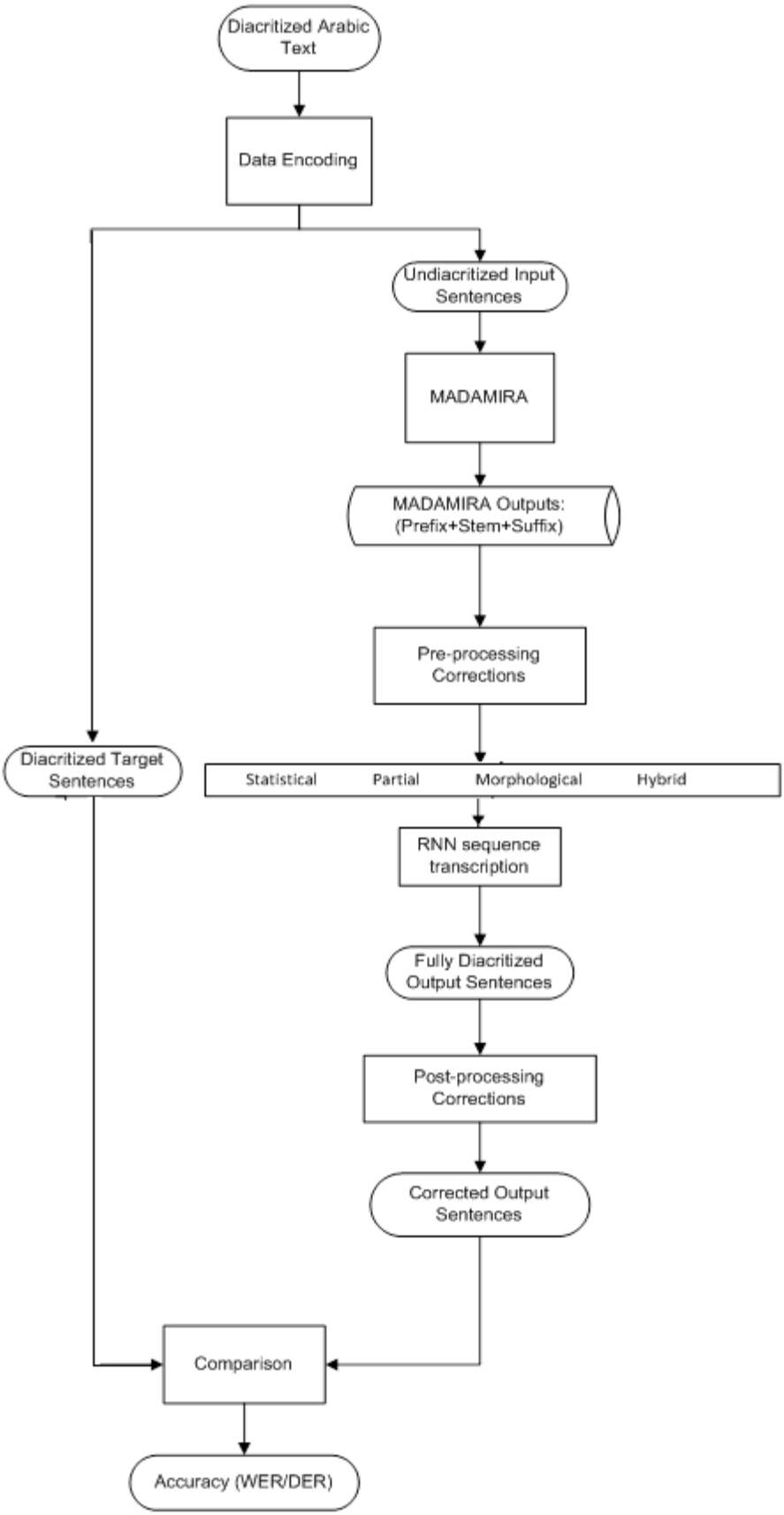

Figure 11. The schematic diagram of the proposed diacritization system (Production phase)

4.2 Data

Our experimental data was from LDC's Arabic Treebank of diacritized news stories of Part 3 v3.2 and catalog number LDC2010T08 ((Maamouri, et al., 2004). The LDC ATB3 corpus is a benchmark and consists of 599 news stories of An-Nahar Lebanese newspaper that were released in 2002 and it includes the inflectional diacritics.

The LDC ATB3 corpus is partially diacritized since 39.8% of the corpus is not diacritized as shown in Table 4. Also 5.4% is the percentage of letters that have two diacritics i.e Shaddah and another diacritic. The average number of words per sentence is 11.31 i.e. more than eleven words are in one sentence so this indicates the long relations between the words that exist in one sentence. In this work, we use deep bidirectional LSTM and this architecture fits these situations.

Table 4. Statistics of LDC ATB3

Criterion	**Value**
Size	305 K words
Letters per word	4.64
Words per sentence	11.31
No diacritics	39.8%
One diacritic	54.8%
Two diacritics	5.4%

This benchmark is widely used in the automatic Arabic text diacritization field so we can compare our results to the best published result of this field. Also, we can know the effect of using partially diacritized inputs on the diacritization accuracy.

4.3 Data Pre-processing

We make a few pre-processing steps to make the data suitable for RNN sequence transcription.

4.3.1 Data Encoding

We prepare our experimental data for RNN training and testing by having one sentence per line and this is achieved by splitting the data into more than one sentences based on some punctuation marks (Abandah, et al., 2015). Each sentence is available in two fields: input and target fields. Comma is used to separate the two fields. Target field is the diacritized sentences of the LDC ATB3 corpus and the input field is generated after removing all diacritics from target field. This preparation is useful for the supervised learning of RNN.

Unicode system gives each letters and diacritics separate encoding (see appendix A for Unicode codes of Arabic character). This is called "one-to-many" letter encoding and is used with one-to-many network (Abandah, et al., 2015). For example, the word **طَلَبَ** has two field record of input and target i.e. **"طلب", "طَلَبَ"** and is encoded as follows:

"طلب", "طَلَبَ"

"0637 0644 0628"," 0637 064E 0644 064E 0628 064E"

Using separate encoding for diacritics will make diacritized sequence length longer than undicritized sequence length and will add some difficulty. Abandah et al. (2015) proposed the using of "one-to-one" letter encoding so the letter and its diacritics will encode to one encoding. This will make target sequence length equal to the input sequence length.

The Unicode codes of the 36 Arabic letters start from 0x0621 to 0x063A and from 0x0641 to 0x064A and also diacritics have hexadecimal codes from 0x064B to 0x0652. To use "one-to-one" encoding, we begin with the Unicode code of the letter then we clear the most significant 8 bits of the code and then we shift the lower 8 bits of the code to the left four bits. Then in case the letter is not followed by any diacritics we used this shifted code. But in case we have one or two diacritics we will OR the shifted code of the letter with the bit codes of its diacritics that are shown in Table 5. If the letter is followed by shaddah diacritic, then bit 3 will be set (bit code 1000) and if the letter is followed by a diacritic other than shaddah, then bits 0 through 2 will be set depending on the bit codes of the diacritic.

Table 5. The Binary bit codes and hexadecimal Unicode of Arabic diacritics

Diacritic	**Unicode**	**Bit code**
No diacritic	-	0000
Fathatan	0x064b	0001
Dammatan	0x064c	0010
Kasratan	0x064d	0011
Fataha	0x064e	0100
Damma	0x064f	0101
Kasra	0x0650	0110
Sukun	0x0652	0111
Shadda	0x0651	1000

We use the following formula to find a unique code "L" of the letter. However, l represents the Unicode value and the diacritics d_1 and d_2 represent bit codes.

$$L = \begin{cases} (l \wedge 0x00ff \ll 4) & no\ diacritics \\ (l \wedge 0x00ff \ll 4) \vee d1 & one\ diacritics \\ (l \wedge 0x00ff \ll 4) \vee d1 \vee d2 & two\ diacritics \end{cases} \quad (16)$$

Then we will encode the previous example to decimal format: “0880 1088 0640”, “0884 1092 0644”. So, we will have “one-to-one” mapping between the input codes and the output codes.

4.3.2 Using MADAMIRA

We used MADAMIRA morphological analyzer to extract partially diacritized and tokenized words and then we prepared this data in a format suitable for RNN sequence transcription i.e each line of file contains one sentence with two fields input and target. Output of MADAMIRA appears as shown in Figure 12.

The MADAMIRA file begins with the word SENTENCE which contains the input file sentences that exist in one line and they will be in Buckwalter encoding. Then MADAMIRA starts with the first word in the SENTENCE and then makes for this word prediction of diacritization, lexeme, aspect, and case...etc. There is more than one prediction for this word but we work on the top analysis of MADAMIRA output so we have the final results of this word as appears in the line that starts with star. This line was generated based on the best analysis of the word that depends on the context so it will take in concern the different morphological features such as lexeme, gender, part of speech and case and so on, to be able to produce diacritics for this word.

Star line starts with confidence degree that indicates how much this analyzer sure about the prediction of diacritics, and we work based on this confidence degree when we take the diacritization of the word. Then prediction for the second word will be produced until the end of the sentence and so on.

We will have partially diacritzed data from MADAMIRA based on this confidence degree since we ignore diacritization of any words with confidence degree less than the confidence degree that we choose. For example: if we choose 95%

confidence degree, we will discard the diacritization of the words that have confidence degrees less than the chosen confidence degree i.e. the word “AlnA}b” النائب has a confidence degree of 89% so we will take it without diacritics but if we choose the confidence degree of value 70% then we will take this word with its diacritics “Aln~A}ibu” النّائِبُ.

```
;;; SENTENCE AtlE AlnA}b AlEAm Altmyyzy EdnAn EDwm ElY AltHqyqAt AljAryp fy HAdv AxtfA'
 Almhnds fy AltnZym Almdny wdyE >by rASd qbl >rbEp >yAm
;;WORD AtlE
;;LENGTH 4
;;OFFSET 0
;;SVM_PREDICTIONS: AtlE diac:AiT~alaEa lex:{iT~alaE asp:p cas:na enc0:0 gen:m mod:i
num:s per:3 pos:verb prc0:0 prc1:0 prc2:0 prc3:0 stt:na vox:a
*0.891518 diac:AiT~alaEa lex:{iT~alaE_1 bw:{iT~alaE/PV+a/PVSUFF_SUBJ:3MS gloss:examine;study
sufgloss:he/it_[verb] pos:verb prc3:0 prc2:0 prc1:0 prc0:0 per:3 asp:p vox:a mod:i gen:m
num:s stt:na cas:na enc0:0 rat:na source:lex stem:{iT~alaE stemcat:PV
---------------
;;WORD AlnA}b
;;LENGTH 6
;;OFFSET 0
;;SVM_PREDICTIONS: AlnA}b diac:Aln~A}ibu lex:nA}ib asp:na cas:n enc0:0 gen:m mod:na
 num:s per:na pos:noun prc0:Al_det prc1:0 prc2:0 prc3:0 stt:d vox:na
*0.892456 diac:Aln~A}ibu lex:nA}ib_1 bw:Al/DET+nA}ib/NOUN+u/CASE_DEF_NOM gloss:deputy;
delegate;vice- pregloss:the sufgloss:[def.nom.] pos:noun prc3:0 prc2:0 prc1:0
prc0:Al_det per:na asp:na vox:na mod:na gen:m num:s stt:d cas:n enc0:0 rat:y source:lex stem:nA}ib stemcat:N/ap
---------------
;;WORD AlEAm
;;LENGTH 5
;;OFFSET 0
;;SVM_PREDICTIONS: AlEAm diac:AlEAm~u lex:EAm~ asp:na cas:n enc0:0 gen:m mod:na num:s per:na
 pos:adj prc0:Al_det prc1:0 prc2:0 prc3:0 stt:d vox:na
*0.892456 diac:AlEAm~u lex:EAm~_1 bw:Al/DET+EAm~/ADJ+u/CASE_DEF_NOM gloss:general;common;public
pregloss:the sufgloss:[def.nom.] pos:adj prc3:0 prc2:0 prc1:0 prc0:Al_det per:na asp:na vox:na
```

Figure 12. An example of MADAMIRA output file

Also, in order to have tokenized data (each word divided into prefix, suffix, or both) we deal with the tokenization file produced by MADAMIRA that gives undiacritized and tokenized data. There are several tokenization schemes available with MADAMIRA, we used ATB_EVAL scheme which make clitics tokenization but doesn’t separate the (al) determinant (Pasha, et al., 2014). The "+" sign indicates the separation between the word components as shown in Figure 13.

The tokenization is dependent on the analysis of MADAMIRA and sometimes there is a difference between the words of previous MADAMIRA file that we took from

it the partial diacritization of data and their morphological segmentation in the tokenization file. Since we don't have confidence degree in this tokenization file, we need to handle these issues by working on normal cases, exception cases, and failed to transform cases.

```
ATlE AlnA}b AlEAm Altmyyzy EdnAn EDwm Ely AltHqyqAt AljAryp fy HAdv AxtfA'
Almhnds fy AltnZym Almdny wdyE Aby rA$d qbl ArbEp AyAm
w+AfAdt mSAdr mTlEp Ely AltHqyq An+h yjry AltwsE fy+h mE vlAvp EnASr mn qwy
AlAmn AldAxly AHylwA Ely mfrzp AltHry l+AstyDAH+hm TbyEp AlElAqp Alty trbT+hm b+Almxtfy
w+Hty AlAn tEtqd AlmSAdr An AsbAb AlHAdv qd tkwn $xSyp
w+*krt An Aby rA$d lA yntmy Aly Ay tnZym Aw Hzb
w+rAt An nql syArp+h Aly brytAl fy AlbqAE kAn b+qSd Altmwyh
w+fhm An+h yjry AlEml Ely Edd mn AlAtSAlAt AlAxyrp Alty tlqy+hA Aw Ajry+hA Aby
rA$d Ely jhAz+h Alxlywy
ElmA An+h Evr Ely h*A AljhAz mlqy fy bldp rwmyp
w+fy AlmjAl nfs+h
ASdr nqyb Almhndsyn fy byrwt SbHy AlbsAT AlbyAn AlAty
" mrt AyAm Ely AxtfA' Alzmyl Almhnds wdyE Aby rA$d mn dwn An ytm Alk$f Hty AlsAEp En mkAn wjwd+h
An+y AnA$d AlslTAt AlAmnyp Alrsmyp Alk$f fy AsrE mA ymkn En mkAn wjwd Almhnds Aby rA$d
w+An Ay sw' qd yTAwl lA smH Allh AHd AlzmlA' Almhndsyn hw b+mvAbp AEtdA' Ely Aljsm Alhndsy AllbnAny
b+kAml+hw+tAml nqAbp Almhndsyn fy byrwt fy An yEwd Almhnds Aby rA$d Aly *wy w+zmlA'+h slymA mEAfy "
tqdm AlmHAmy ywsf lHwd AmAm Almjls AlEdly b+m*krp Tlb fy+hA b+wkAlp+h En sbEp mdEyn fy mlf mjzrp
AlAwnyskw AnDmAm mwkly+h AlmdEyn Aly AldEwy AlEAmp
w+hm k+lyr Ebd Allh Al$ryAq w+f&Ad Hlym AlfgAly w+nAzk bTrs SwmA w+wfA' dAwd Abw Ezy w+zyAd slymAn
AljAmws w+bsAm slymAn AljAmws wrl+y slymAn AljAmws
k+*lk
Tlb dEwp Sndwq AltEwyDAt l+mElmy AlmdArs AlxASp k+$xS mEnwy w+mstql b+wAsTp mn ymvl+h qAnwnA AmAm
AlmHAkm
" b+Sfp+h ms&wlA b+AlmAl En AfEAl AlmwZf ldy+h AHmd mnSwr
w+l+ylzm+h b+AltkAfl w+AltDAmn AlAlzAmAt AlmAlyp Alty s+yqr+hA mjls+km Alkrym l+AlmstdEyn
sndA Aly AlmAdp 0 mn qAnwn AlEqwbAt w+l+AHkAm qAnwn ASwl AlmHAkmAt AljzA}yp "
```

Figure 13. An example of MADAMIRA tokenization output file

We have normal cases i.e. the "+" sign is inserted in the correct location when we have similar characters and similar number of characters between the corresponding words of both files. For example: the diacritized word is waAafAdat (وَأَفادَت) and the tokenized word from tokenization file is w+AfAdt (و+أفادت) so in this case, we have similar characters and similar number of characters, we will get after combining them wa+AafAdat (وَ+أَفادَت). We need the data to be partially diacritized and partially tokenized to be able to do our experiments (explained below) and to study the effect of adding this linguistic information on the results of our RNN library.

We have exception cases i.e. we can insert the "+" sign in the correct location based on some rules, that will deal with different characters and different number of

characters between the corresponding words of both files. I will discuss some of these rules. There is a rule to handle the extra A in the token word, for example: the diacritized word is fa>axobaruwniy (فَأَخْبَرُونِي) and the tokenized word is f+>xbrwA+ny (ف+أخبروا+ني) so, we will have after combining them based on this rule fa+>axobaruw+niy (فَ+أَخْبَرُو+نِي). There is a rule to handle the char that has shadda diacritic, which means the doubling of the char, for example: the diacritized word is min~A (مِنّا) and the tokenized word is mn+nA (من+نا) so we will have after combining them min~+A (مِنّ+ا). There are also a rules to handle A,y characters, and y,Y characters, since some tokens in the tokenization file are return to the base form. Example of A,y characters case: the diacritized word is fa>tAh (فَأَتاه) and the tokenized word is f+>ty+h (ف+أتي+ه) so we will have after combining them fa+>tA+h (فَ+أتا+ه). Example of y,Y characters case: the diacritized word is warawaY (وَرَوَى) and the tokenized word is w+rwy (و+روي) so we will have after combining them wa+rawaY (وَ+رَوَى).

We have failed to transform cases which contain all the words that are not corresponding to the above two cases and have incorrect tokenization, so we will discard the insertion of “+" sign for these words and this means that we will have partially tokenized data. For example: the diacritized word is bisomi (بِسْمِ) and the tokenized word is b+bsm (ب+بسم) which is incorrect so we will not insert “+" to the diacritized word.

In this work, we use the output of MADAMIRA to build four experiments (Statistical, Partial Diacritization, Morphological, and Hybrid). In Statistical experiment, the input sequences are not diacrtized and also are not tokenized, for example:

"<*n fAlHkwmp t$Er btfwq AstvnA}y"

In Partial Diacritization experiments, the input sequences are not tokenized but partially diacritized from MADAMIRA, for example:

"<i*ano faAlHukuwmapu ta$oEuru bitafaw~uqK AisotivonA}iy~K"

In Morphological experiments, the input sequences are not diacrtized but partially tokenized, for example:

"<*n f+AlHkwmp t$Er b+tfwq AstvnA}y"

In Hybrid experiments, the input sequences are partially diacritized and partially tokenized from MADAMIRA, for example:

"<i*ano fa+AlHukuwmapu ta$oEuru bi+tafaw~uqK AisotivonA}iy~K"

4.3.3 Text Correction

We make text corrections for some words of MADAMIRA output, when we concatenate the input sequence of MADAMIRA analyzer with the target sequence, as we discussed below.

4.3.3.1 Letter Correction

When we concatenate the input sequence with the target sequence we must make sure that MADAMIRA output letters will be exactly same as target letters. We notice that MADAMIRA changes some words letters, for example: MADAMIRA output word is "الهدي" instead of "الهدى". So, we fix this problem by making letter correction that checks each letter of input word with each letter of the corresponding target word.

4.3.3.2 Target Normalization

For Morphological and Hybrid experiments, we need to ensure one to one mapping between input and target sequences because they make morphological division for the words. So, we normalize the target sequence to be tokenized in the same way that the input sequence is tokenized. Using the same example of the morphological experiments that we used in Section 4.3.2, the input and target sequences were:

"<*n f+AlHkwmp t$Er b+tfwq AstvnA}y","<i*ano faAlHukuwmapu ta$oEuru bitafaw~uqK AisotivonA}iy~K"

And it will be after we do target normalization as following:

"<*n f+AlHkwmp t$Er b+tfwq AstvnA}y","<i*ano fa+AlHukuwmapu ta$oEuru bi+tafaw~uqK AisotivonA}iy~K"

4.4 Sequence Transcription

In this work, we use CURRENNT library to transcribe sequences. The architecture of this library is the deep bidirectional LSTM and this is useful for sequence transcription problem. Speech recognition (Graves, et al., 2013) and handwriting recognition (Abandah, et al., 2014) are two examples of using this architecture to solve sequence transcription problems and also they achieved state-of-the-art results.

We use CURRENNT library to train and test the sequences that were produced by using the “one-to-one” letter encoding. So, the input and target sequences had the same lengths.

4.5 RNN Training Parameters

We follow many researchers of automatic diacritization field in splitting the LDC ATB3 corpus into two sets: training, and test. In this split, test set will be same as validation set rather than using different set for it. So, we will be able to compare our results with previous researchers results such as (Zitouni, et al., 2006) (Habash and Rambow, 2007) (Rashwan et al., 2011) (Said, et al., 2013), etc.

LDC ATB3 corpus consists of 599 news stories from An Nahar Newspaper. The training set consists of the first 509 news stories i.e. approximately the first ten months of the news stories (~242 K words). The test set consists of the remaining 90 news stories and this is approximately 15% of the corpus i.e. approximately the last two months news stories of the same year (~42 K words). The validation set will be same as test set. In this work, when we prepared LDC ATB3 data, there was a field which contains the diacritized version of the words. So, we extracted these words to form LDC ATB3 sentences, but sometimes this field will have the word ”none”, which means that this word is available without having the diacritized versions, so we don’t take these words.

4.6 Data Post-processing

We use post-processing techniques to overcome some issues in the output sequences of the RNN sequence transcription and to achieve better accuracy.

4.6.1 Letter Correction

We correct any error in the letters of the RNN output sequences to be same as the letters of input and target sequences. And this correction will not improve diacritization accuracy since we correct letter not diacritics but this will give better

output sequences. For example, you could have the word "قُوت" in the output sequence instead of the word "قُوى".

4.6.2 Sukun Correction

We ignore sukun diacritic of the target and output sequences when we counting diacritization errors because there are different ways for using this diacritic i.e some writing styles don't use sukun and consider any letter without diacritic as having sukon and some other writing styles use sukon to show that the letter doesn't have vowel. For LDC ATB3, This correction makes a reduction on the DER by 3.7%.

4.6.3 Fatha Correction

We correct any letters in the output sequences, which have diacritic other than fatha to fatha diacritic. In case this diacritic precedes Alef, Alef Maksura, and Taa Marbuta letters since according to Arabic orthographic system we have always fatha in that case. For LDC ATB3, This correction makes a reduction on the DER by 1.4%.

4.6.4 Dictionary Correction

Dictionary is built from diacritized words that exist in the training set and is indexed by the undiacritized word. We check if RNN output word exists in the dictionary by using the undiacritized form. If the RNN output word is founded in the dictionary and has the same diacritics we keep this word and we don't change it. Also we don't change the output word if it is not founded in the dictionary but in case the output word is founded in the dictionary and don't match any diacritzed word we select from dictionary the diactrized version that has smallest edit distance and we correct this word. If the output word and dictionary word differ only in the last letter diacritics we

don’t change output word. For LDC ATB3, This correction makes a reduction on the DER by 1.24%.

CHAPTER V

Experiments and Results

Our proposed diacritization system consists of using MADAMIRA morphological analyzer to add linguistic information to the input sequence and then we used CURRENNT to do sequence transcription task and to predict a fully diacritized output sequences. Our workload is LDC ATB3 corpus because the state-of-art approaches of automatic diacritization field use this workload. We split this corpus into two sets: training and testing. The validation set is same as testing set. We do many experiments using CURRENNT to come up with the best configurations that will give the best results.

5.1 Accuracy evaluation

There are two metrics that are used in the literature to calculate diacritzation accuracy: diacritization error rate (DER) and word error rate (WER). DER represents the percentage of letters with wrong diacritics, and WER represents the percentage of word that has at least one letter of wrong diacritics. We calculate these metrics for each experiment from the output sequences of CURRENNT and for both cases with and without last ending diacritics. We calculate the accuracy of LDC ATB3 under the same condition that was used in previous works such as (Zitouni et al., 2006), (Habash and Rambow, 2007), (Rashwan et al., 2011), and (Said et al., 2013). The following are the conditions:

1. Words, numbers, and punctuators are all considered in calculating accuracy.
2. Each character or digit in a word can have diacritics.

3. For DER calculation, in case the letter has more than one diacritic then you have one of two choices: you need to consider all of them or you can consider them as one error.
4. In case the target letter was undiacritized then the diacritics of the same letter in the output sequence will be ignored.

5.2 RNN Tuning Experiments

We try to work on a network architecture that to some extent will be close to the network architecture that was used in (Abandah, et al., 2015) because they get state-of-the-art results and they used the same workload. In this work, we use different library that is fast because it uses GPU and to our knowledge, it considered as the first publicly-available tool that has parallel implementation of a deep LSTM RNN architecture.

We train and test our data using "one-to-one" transcription network which is the best for automatic diacritization and in this network the input and target sequences have one to one correspondence because character and diacritic have one code (Abandah, et al., 2015). The following subsections present the experiments we do to tune RNN.

5.2.1 Selection of confidence degree

We did several experiments to determine the confidence degree that will give us the optimal accuracy. Confidence degrees indicate how much MADAMIRA analyzer sure about the prediction of diacritics (it was explained in section 4.3.2). In all experiments, we work on 250 neurons depending on the network architecture that was used in (Abandah, et al., 2015) because they get state-of-the-art results in this field and our work is considered as extension for their works.

We worked on the partial and hybrid experiments with two hidden layers of size 250 and for all confidence values (10, 20, 30, 40, 50, 60, 70, 80, 88, 90, and 100) and

we calculate DER and WER for all of them. Figures 14, 15, 16 and 17, show the DER and WER values of the partial and hybrid experiments of two hidden layers of size 250. We noticed that if we want to take the partial diacritization from MADAMIRA based on high confidence degrees, such as 100% and 90%, then the diacritized words will be few so we will get many undiacritized words. Also, this was noticed from the figures that the error rates for high confidence values were high.

We noticed that if we want to take the partial diacritization from MADAMIRA based on low confidence degree, such as 10%, then we will have many diacritized words and the error rates will be reduced compared to the error rates of high confidence values. However, if we want to take the partial diacritization from MADAMIRA based on intermediate confidence degree, such as 60%, then we will have many diacritized words and in the same time the MADAMIRA will be highly sure from the prediction of this diacritization and we will ignore the diacritized words that MADAMIRA was not sure about. We found that 60% confidence value of the hybrid experiment gave us the lowest value of DER and WER so we used 60% for the following experiments.

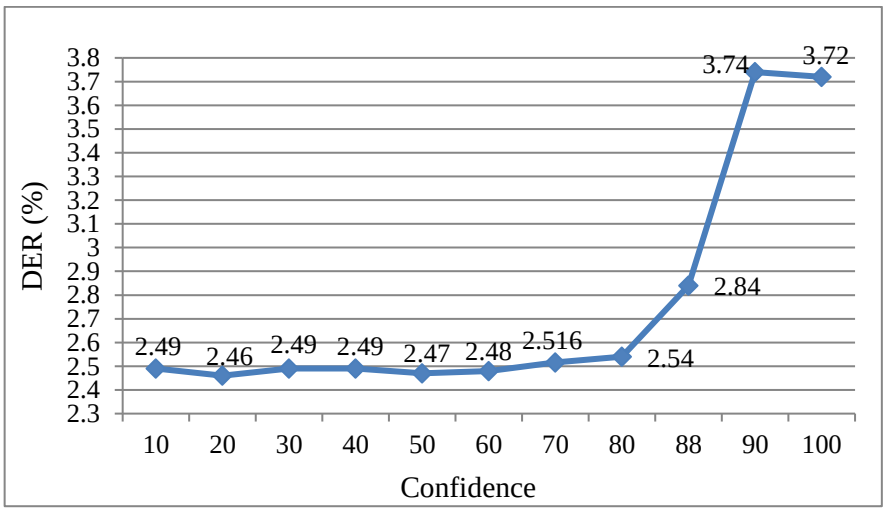

Figure 14. DER values of the partial experiment that has two hidden layers of 250 neurons.

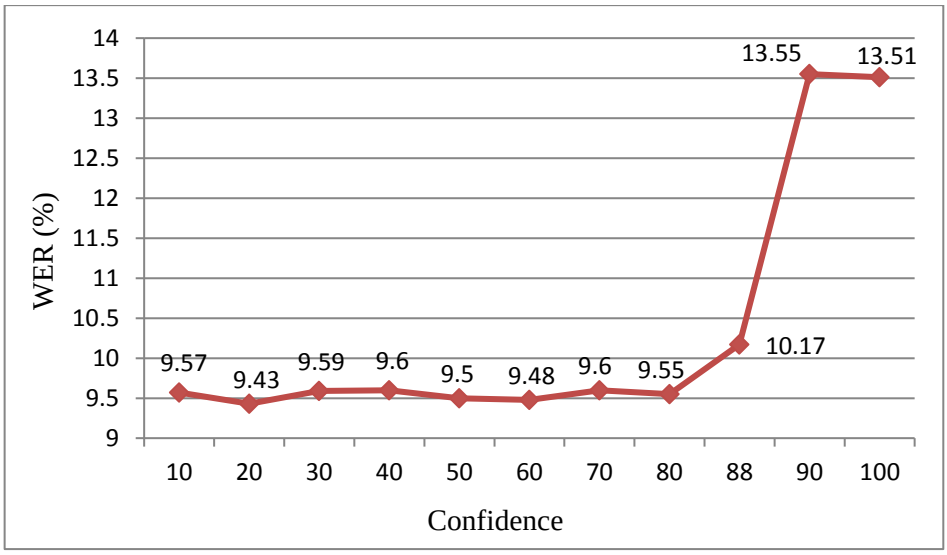

Figure 15. WER values of the partial experiment that has two hidden layers of 250 neurons.

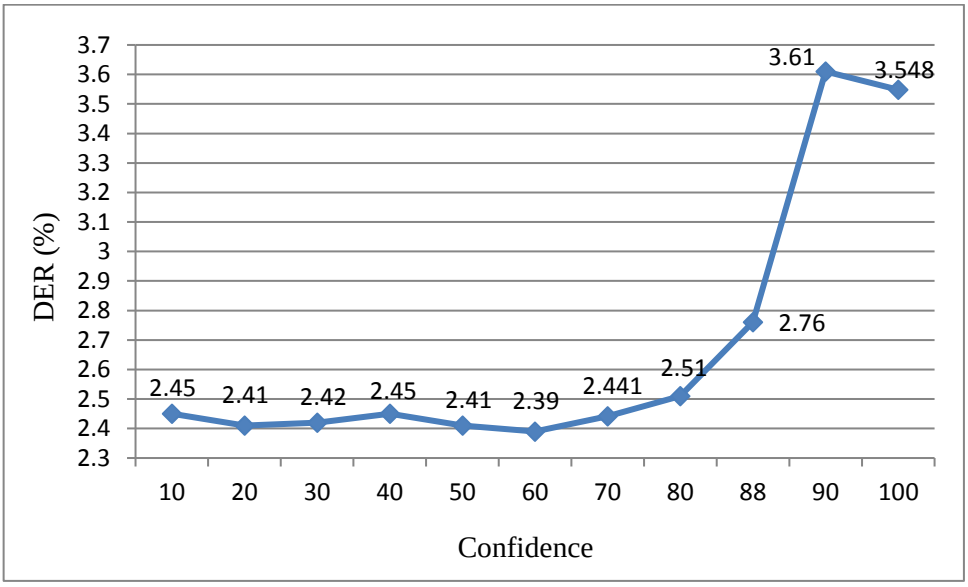

Figure 16. DER values of the hybrid experiment that has two hidden layers of 250 neurons.

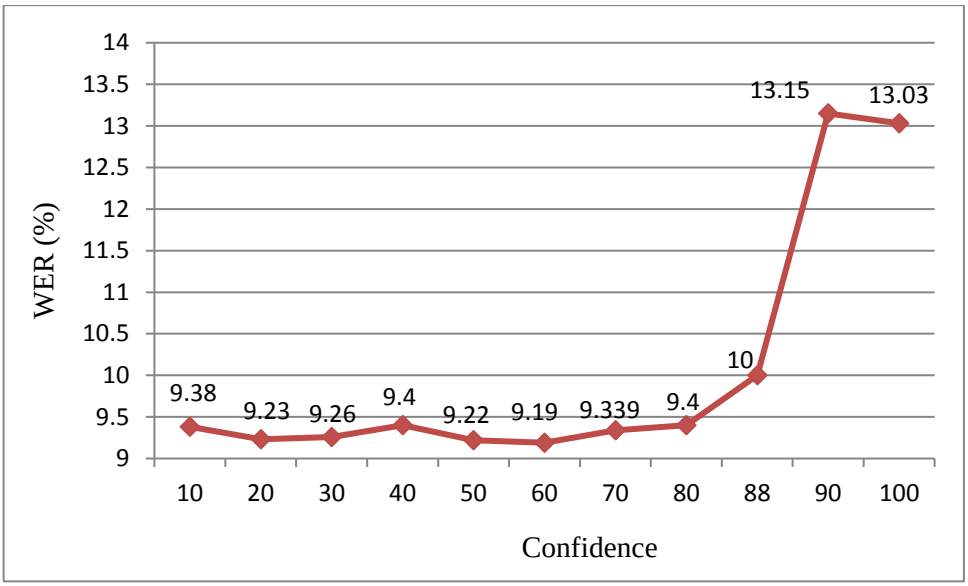

Figure 17. WER values of the hybrid experiment that has two hidden layers of 250 neurons.

5.2.2 Network size

We have tested the effect of changing the number of hidden layers on error rates. We know that the number of hidden layers depend on the hardness of the problem. Here we need to solve diacritization problem. In this problem, we deal with character then with words and finally with sentences so this indicates that we need to have deep neural network.

We did several experiments to determine the optimal number of hidden layers that we will use. We worked on the partial and hybrid experiments of 60% confidence. We calculated the classification error rate (the rate of number of diacritzation error to the total number of symbols). Figure 18 and 19 show the classification errors rate of the two experiments after we changed the number of hidden layers from one layer to four layers. We have noticed that increasing number of layers reduce the classification errors. Also, the error rate of the hybrid and partial experiments are close to each other and the hybrid experiment gives better result comparing with the partial experiment.

This is predictable since the input sequences for hybrid experiment are partially diacritzed and tokenized and the input sequences for partial experiment are only partially diacritzed. The benefit we have from tokenized words will solve few cases of complex words and this will help RNN so we will get more improvement for hybrid experiment over partial experiment.

For partial experiment, we achieve the same classification errors when we use three and four hidden layers. For hybrid experiment, we achieve a little improvement when using three and four hidden layers. We need to determine which one are the best three or four layers. So, we need to know the training time for the hybrid experiment because changing number of hidden layers from one layer to four layers gave us a little improvement. So, we didn't need to further increase the number of hidden layers and we need to know the training time taken by three and four layers configuration.

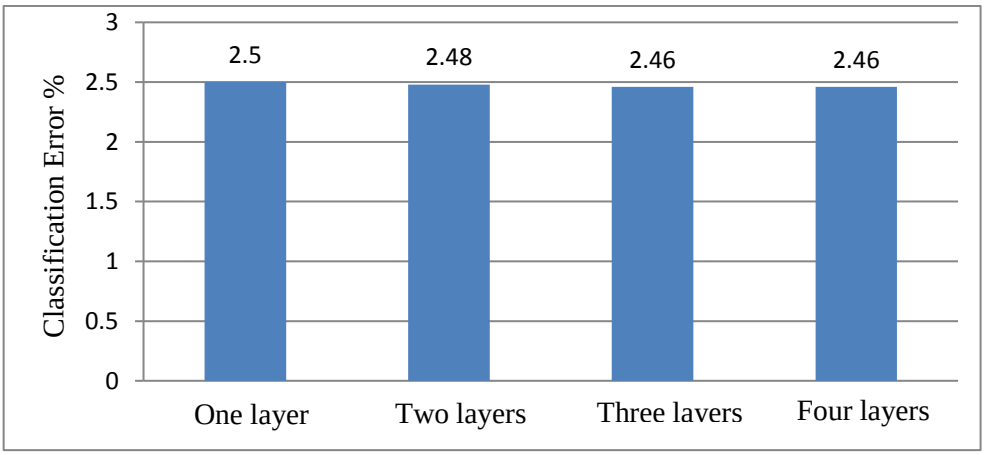

Figure 18. Effect of changing number of hidden layers on the partial experiment of 60% confidence degree and each hidden layer consist of 250 nodes.

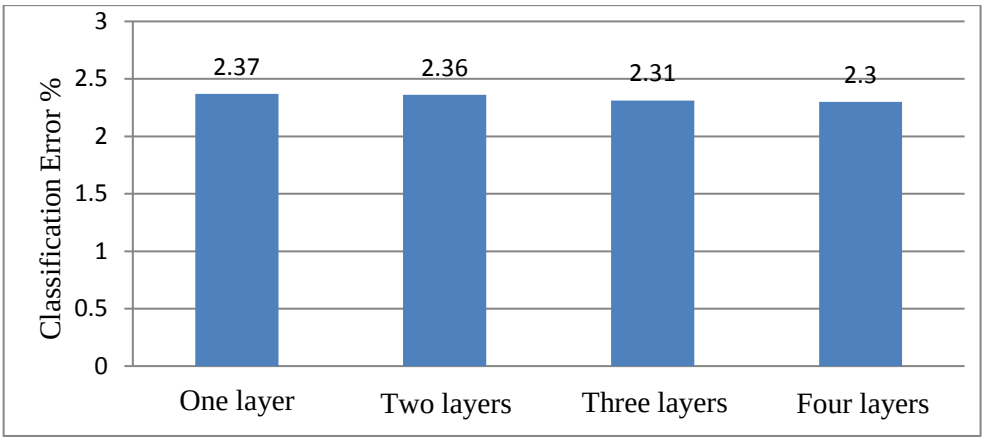

Figure 19. Effect of changing number of hidden layers on the hybrid experiment of 60% confidence degree and each hidden layer consist of 250 nodes.

5.2.3 Training and Testing Time

In this work, we are interested in the training time because in (Abandah, et al., 2015) the training time of RNNLIB was very long and this time is important to be able to work on large data sets and to enhance the accuracy. Also, the testing time is important because we need to know how much time it will take for diacritizing undiacritized texts.

We measured the training and testing times for the four experiments when we have different number of layers (from one layer to four layers), as shown in Figure 20 and 21. The training and testing times of four layers configuration achieved the higher value because more layers (more weights to train) will take more time to train and the same with testing time, it will grow. Also, we noticed that testing times gave little improvement when we change the number of layers.

From Figure 20 and 21, we conclude that the hybrid experiment of 3-layers configuration is the best when we consider the accuracy (classification errors rates is 2.31) and the training time (2 hours and 42 minutes). We noticed from Figure 19 that one layer configuration is not good as three or four layers since we need deep neural

network configuration and three layers provide good improvement over one and two layers configuration. Also, the four layer configuration is slower than three layers configuration so we consider 3-layers configuration.

Therefore, we adopt the one-to-one transcription method, the best confidence value is 60% and the optimal number of layers for this problem is three hidden layers each with 250 nodes.

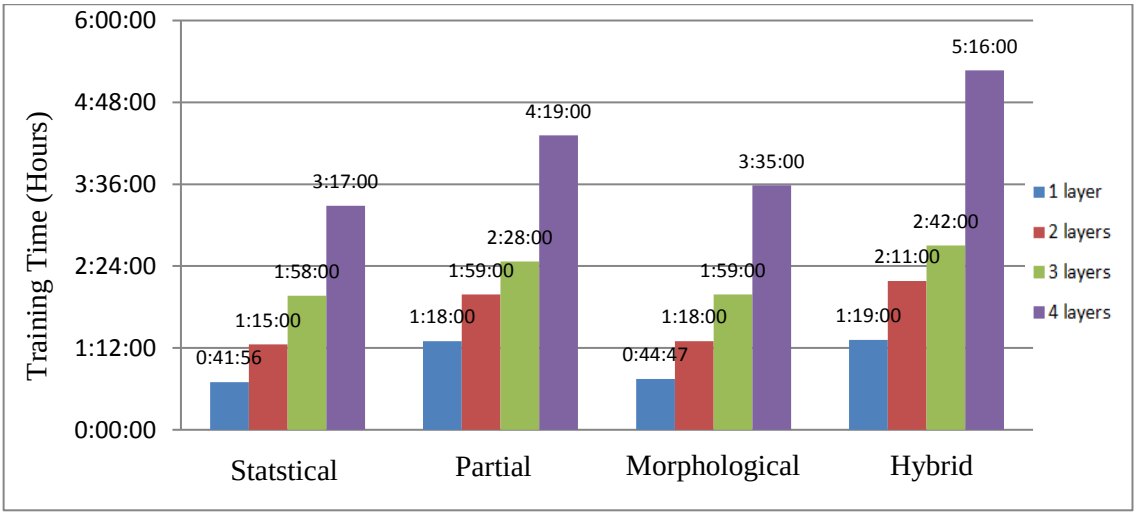

Figure 20. Training time of the four experiments when we change the number of layers from 1 layer to 4 layers.

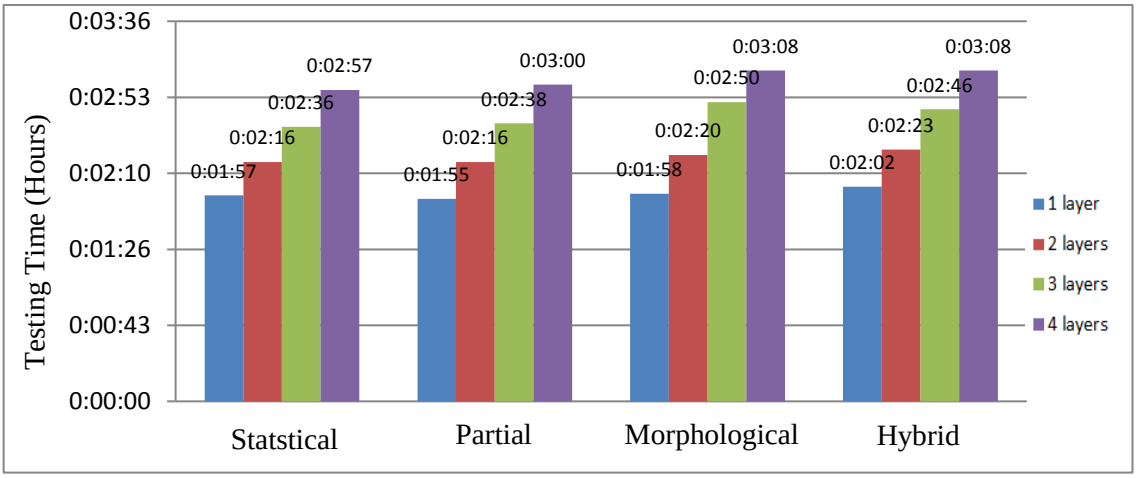

Figure 21. Testing time of the four experiments when we change the number of layers from 1 layer to 4 layers.

5.3 Experiments Results

In this section we present the results of the four experiments that we discussed in Section 4.3.2, we need to study the effect of adding linguistic information to the input sequences of RNN at three levels and also in case we don't have any linguistic information, the following are our four experiments:

1. Statistical Experiment
2. Partial Diacritization Experiment
3. Morphological Experiment
4. Hybrid Experiment

Table 6 shows the results of the four experiments using LDC ATB3 corpus. DER_all and WER_all are the error rates when all diacritization errors are calculated. DER_ilast and WER_ilast are the error rates when ignoring the diacritization errors in the last letter of each word. The last raw shows the difference between DER_all and DER_ilast. All these experiments are trained using the same splitting of training and testing set that is described in Section 4.5 and all these results are calculated after applying post-processing corrections which we described in Section 4.6.

Table 6. Diacritization results of the four experiments using LDC ATB3 corpus.

Accuracy	Statistical	Partial	Morphological	Hybrid
DER_all	3.62	2.46	3.40	2.39
WER_all	12.24	8.67	11.54	8.40
DER_ilast	1.70	0.80	1.63	0.78
WER_ilast	5.36	2.35	5.17	2.30
DER_last	1.92	1.65	1.77	1.61

We have noticed that the diacritization error of all experiments decrease when we ignore the diacritization error of the last letters of each word. This is predictable

because predicting the syntactic diacritics of the last letter is much hard than predicting the morphological diacritics. The syntactic diacritics depend on the word location in the parsing tree. Also, as we noticed that the WER is affected by the diacritic of last letter because if the word has a suffix then the syntactic diacritics will not appear on the last letter of the word but it will appear on the stem such as ” تحكُمُها”. DER_last refers to the rate of last-letter diacritization errors to the diacritization errors of all letters. About 92%, 65%, 77%, and 61% of the errors are due to case ending diacritics and these percentages are for statistical, partial, morphological, and hybrid experiments respectively.

5.4 Post-processing Contribution

We use post-processing techniques to improve diacritization error rates. Table 7 shows the impact of the post-processing techniques on DER calculation. We notice that sukun correction has the highest percentage among other techniques because LDC ATB3 is partially diacritized. Fatha correction reduces DER values with 1.4%. It is not a significant value but it will contribute in reducing the error rates.

Table 7. The effect of the post-processing techniques on DER reduction.

Technique	Reduction %
Sukun correction	3.7
Fatha correction	1.4
Dictionary correction	1.24
Total	6.34

The low contribution of dictionary correction is due to its construction from the diacritized words variants that exist in the training set so it will not have the new words and vocabulary of the last two months stories that exist in the test set. If we use a bigger

dictionary that contains all the words with all diacritized words variants, we could have a better contribution of dictionary correction.

5.5 Discussion

We found that the hybrid experiment achieves the best results among the other three experiments because it benefits from partial diacritization and tokenization information that were added to the input sequences and this will help RNN in predicting the rest of the diacritics.

Then the partial experiment takes the second order because it adds partial diacritization information to the input sequences and this will help RNN. This is predictable because comparing to the hybrid approach it provides a little information.

Morphological experiment is better than statistical experiment because it adds morphological information (tokenization) to the input sequences and this will help RNN but statistical approach doesn't have any linguistic information.

Figure 22 and 23, show DER and WER values of the four experiments. We notice that the DER value of the statistical experiment is 3.62 and when we use the hybrid approach, which uses CURRENNT library and MADAMIRA morphological analyzer, the DER value was reduced to 2.39. This is indication that this hybrid system is good because CURRENT will speedup the training time since its use CUDA and MADAMIRA will add morphological info to CURRENT, all of these will help in reducing the DER value and will help in training large dataset.

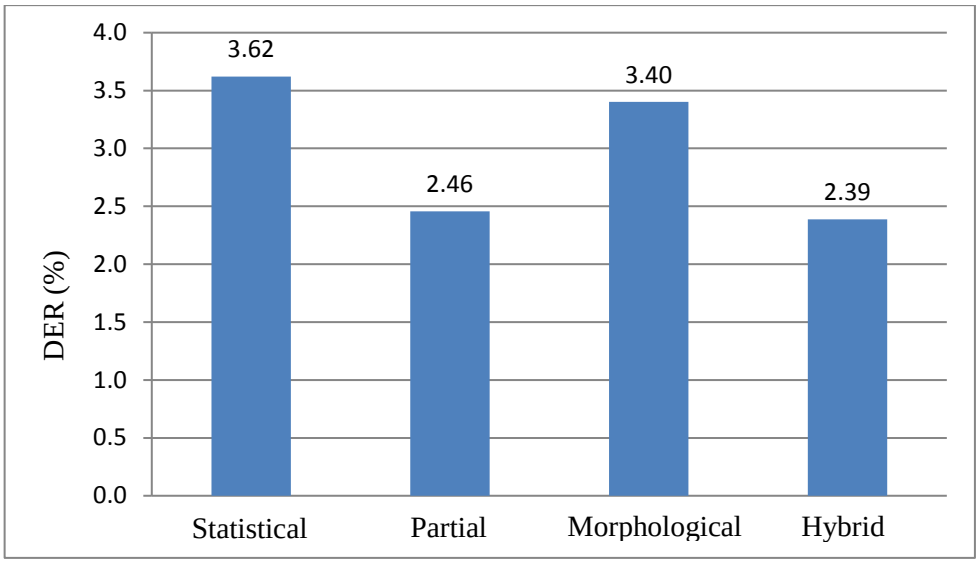

Figure 22. DER results of the four experiments.

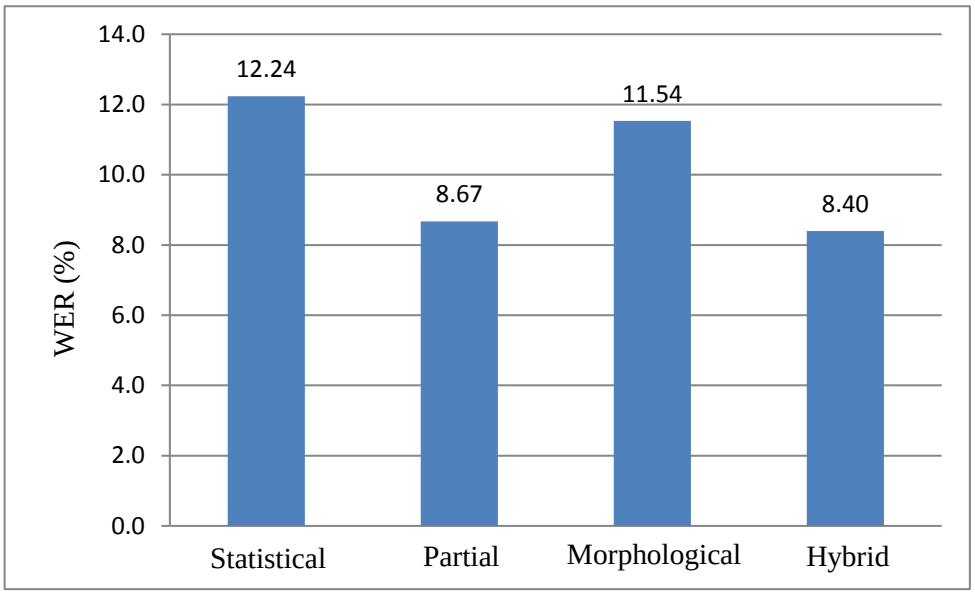

Figure 23. WER results of the four experiments.

Figure 24 shows the classification error of the four experiments. The classification error of experiment four is the best result because it has more linguistic information comparing with other experiments.

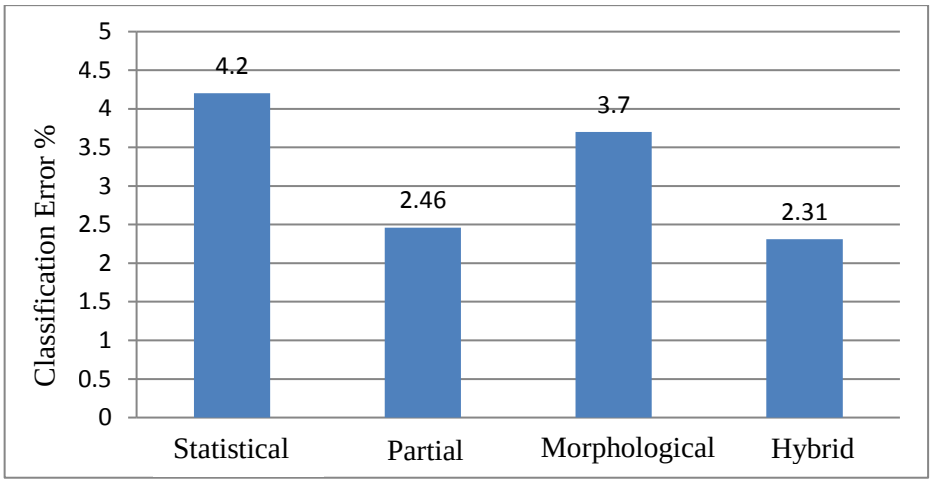

Figure 24. RNN Classification error rates for all experiments.

5.5.1 Comparison with state-of-art systems

Table 8 shows the accuracy results of the best published system in the field of automatic diacritization of Arabic text and our system. Zitouni et al. (2006), Habash and Rambow (2007), Rashwan et al. (2011), Said et al. (2013), and Arabiyat (2015) are all using hybrid approaches that combine rule based approach with statistical approach. Abandah et al. (2015) use statistical approach depends on the deep bidirectional LSTM architecture. All these systems use LDC ATB3 benchmark to provide a fair comparison. Also, we compare our system to Metwally et al. (2016) which is a recent research in this field and its hybrid approach that uses LDC ATB3 benchmark.

Table 8. Comparison between our diacritization system results with related works.

Systems	All Diacritics		Ignore Last		DER Last
	DER	WER	DER	WER	
Zitouni et al. (2006)	5.5	18	2.5	7.9	3
Habash and Rambow (2007)	4.8	14.9	2.2	5.5	2.6
Rashwan et al. (2011)	3.8	12.5	1.2	3.1	2.6
Said et al. (2013)	3.6	11.4	1.6	4.4	2
Abandah et al. (2015) (statistical approach)	2.72	9.07	1.38	4.34	**1.34**
Arabiyat (2015) (hybrid approach using RNNLIB):					
Statistical experiment	3	10.36	1.42	4.52	1.58
Partial experiment	2.74	9.66	1.24	3.95	1.5
Morphological experiment	2.89	10.17	1.36	4.43	1.53
Hybrid experiment	2.8	9.92	1.26	4.07	1.54
Metwally et al. (2016)	-	13.7	-	4.3	-
This Work:					
Statistical experiment	3.62	12.24	1.70	5.36	1.92
Partial experiment	2.46	8.67	0.80	2.35	1.65
Morphological experiment	3.40	11.54	1.63	5.17	1.77
Hybrid experiment	**2.39**	**8.40**	**0.78**	**2.30**	1.61

As shown in Table 8, our experiments give the best results compared to all the state-of-the-art hybrid approaches. Our hybrid experiment achieved the best results among the other three linguistic-added information experiments and gives 12% DER improvement and 7% WER improvement over the best reported result of the statistical approach of Abandah et al. (2015) and 34% DER improvement and 26% WER improvement over the best reported result of the hybrid approach of Said et al. (2013). Also, the statistical approach of Abandah et al. (2015) achieved the best result of DER_Last value.

When we compare our system with the hybrid system of Arabiyat (2015) that use RNNLIB, our system gives 13% DER improvement and 13% WER improvement over the best result of the partial experiment that uses RNNLIB. Also, our hybrid experiment that uses CURRENNT library and MADAMIRA morphological analyzer achieved DER improvement of 15% and WER improvement of 15% over the hybrid

experiment that uses RNNLIB library and BAMA morphological analyzer of Arabiyat (2015).

Our statistical and morphological experiments give DER of 3.62 and 3.40, respectively. The statistical and morphological experiments of Arabiyat (2015) give DER of 3 and 2.89, respectively. It's predictable to have different results since we are using a different library that has a special feature of using CUDA. We achieved close results that differ in a few percent but the most important thing that we achieved better results in DER and WER values while using the hybrid experiment.

When we compare our system with the recent research Metwally et al. (2016) we notice that the WER of this system gives higher value than our WER values and also this WER value is higher than Said et al. (2013). However, when we compare the value of the morphological WER of Metwally et al. (2016) with Said et al. (2013) and Abandah et al. (2015) we notice that it's close to them. When we compare this value with our system it is better than statistical and morphological experiments.

5.5.2 Comparison between our Hybrid Approach and Arabiyat (2015) Hybrid Approach

In this section, we compare this work with Arabiyat (2015) work as shown in Table 9 below. The main goal of both works was to improve the accuracy of diacritization by using hybrid approach. This work used MADAMIRA morphological analyzer and Arabiyat's work used BAMA morphological analyzer. These two tools were totally different and produced different output. MADAMIRA output file and how we extract partially diacritized and tokenized data from it was explained in details in section 4.3.2. Regarding BAMA, it generated all possible diacritized and tokenized solutions for every word (there is a figure of BAMA Solution in section 3.2.2). Then,

they made for each word an array of prefixes, stems, and suffixes for all solutions. They take the matched diacritization in all solutions with the morphological segmentation and they discarded the diacritics in case they had different diacritics for same letter in all solutions.

BAMA didn't produce the syntactic diacritics but MADAMIRA can produce it. Also, BAMA provide tokenization for the words in the same file but MADAMIRA provide tokenization in separate file.

Table 9. Comparison between our hybrid approach and Arabiyat (2015) hybrid approach.

	This work	Arabiyat (2015)
Morphological Tool	MADAMIRA	BAMA
RNN Library	CURRENNT	RNNLIB
Text Correction	- Letter correction - Target normalization	- Space normalization - Extra alef removal - Letter correction - Out of vocabulary conversion - Target normalization

This work used text correction to solve some issues between the output of MADAMIRA and target sentences such as letter correction and target normalization (explained in section 4.3.3). Arabiyat's work used text correction such as "space normalization" to remove the extra space in BAMA's result and ensure one to one mapping between the output of BAMA and target sentences, "extra alef removal" to delete the extra alef in BAMA's result, "letter correction" to deal with BAMA's issues such as incorrect letters, increasing or decreasing the number of letters and replication for some words, "out of vocabulary conversion" is used to replace the numbers that is generated by BAMA when it can't analyze proper and foreign names with the correct names from target sequence, and "target normalization".

Both works used the output of their morphological analyzer to build the four experiments (statistical, partial, morphological and hybrid experiments).The input sequences for the hybrid experiments in both works are partially diacritized and tokenized from the morphological analyzer that both works used. The input sequences for the morphological experiments in both works are partially tokenized. The input sequences for the partial experiments in both works are partially diacritized. The input sequences for the statistical experiments in both works are not diacritized and not tokenized. Also, both works used the same "one-to-one" encoding and encoded the Arabic text into decimal format (explained in section 4.3.1).

For sequence transcription, this work used CURRENNT library (C++/CUDA) and Arabiyat's work used RNNLIB library (C++). Since both works used different library, we need different data conversion script to convert the decimal file to format suitable for both library. Also, RNNLIB supported weight noise regularization option, its specific feature of RNNLIB and used to solve overfitting problem of RNN training, but CURRENNT library didn't support that.

Regarding the post-processing techniques, which were used to correct some errors in the output sequences of RNN libraries, both works used the same techniques such as letter correction, sukon correction, fatha correction and dictionary correction. These techniques were explained in details in section 4.6.

We conclude that for both works, the post-processing techniques were the same but all pre-processing techniques were different because this work used different tools to provide partially diacritized and tokenized data and used different library to train the input sequences.

Figure 25 and 26 show the results of both works. The hybrid experiments of this work provides 13% DER improvement and 13% WER improvement over the best result of the partial experiment of Arabiyat's work.

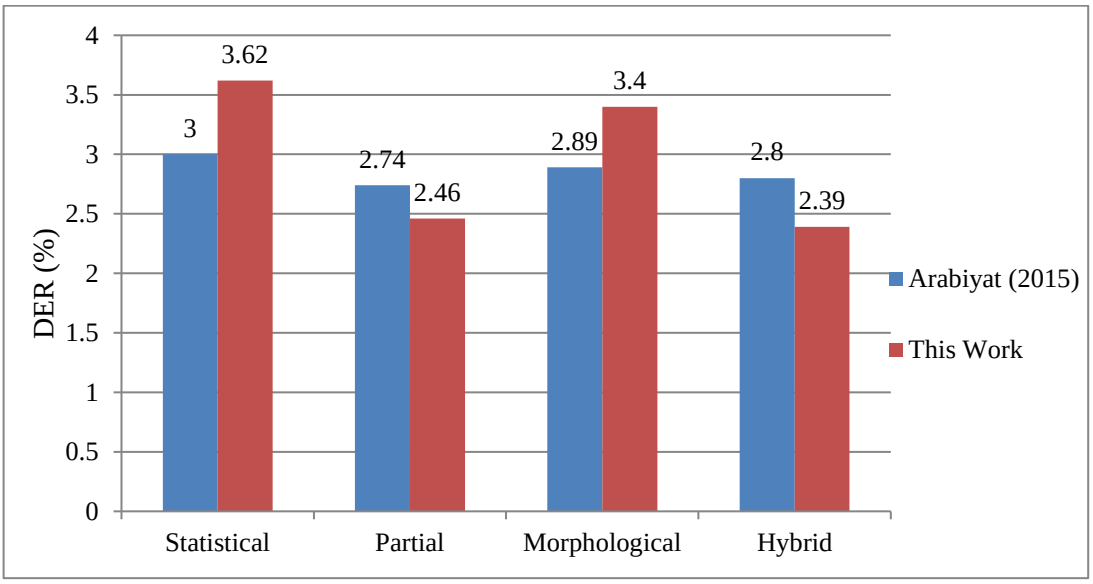

Figure 25. DER values of our hybrid approach and Arabiyat (2015) hybrid approach.

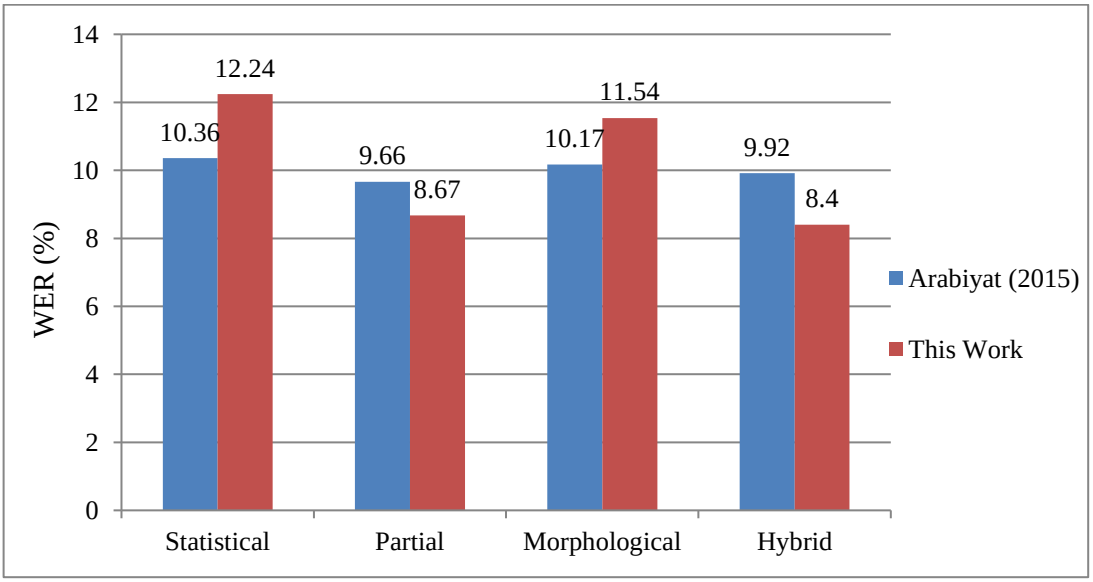

Figure 26. WER values of our hybrid approach and Arabiyat (2015) hybrid approach.

5.5.3 Acceleration using CURRENNT

In this section, we compare the training time taken by RNNLIB library, a single CPU and pure C++ library by Graves (2008), with the training time taken by CURRENNT library, a C++/CUDA based library implemented by Weninger et al. (2015). However, this training time was calculated for the statistical experiment as shown in Table 10. The two hidden layers of both libraries were Bidirectional LSTM and each of size 250. Also, both libraries were used for sequence labeling problem and were trained and tested on LDC ATB3.

Table 10. Comparison between the two libraries using the training time.

Libraries Names	Training Time
RNNLIB	17 Days
CURRENNT	1 hour and 15 minutes

We found that CURRENNT has provided a 326x speedup in deep Bidirectional LSTM training over RNNLIB because it supports GPU implementation of deep LSTM RNN, benefits from mini-batch learning and can process many sequences in parallel. This is indication that CURRENNT will help in processing a large database and enhance accuracy since experimental evidence shows that the accuracy will improve when we increase the size of training data (Abandah, et al., 2015).

5.5.4 Error Analysis

In this section, we show the distribution of word errors based on the number of diacritization errors in each word and if there is a diacritic error in the last letter of word or not. Table 11 shows the distribution of the errors for all experiments. One% means the percentage of the words that have only one diacritization error. Two% means the

percentage of the words that have two diacritization errors. Three+% means the percentage of the words that have three or more errors.

Table 11. The distribution of word errors for all experiments.

Experiment	Errors per word	One%	Two%	Three+%	Total %
Statistical	Last letter correct	24.07	8.27	1.82	34.17
	Error in last letter	55.60	7.40	2.83	65.83
Partial	Last letter correct	12.23	5.66	1.39	19.28
	Error in last letter	72.91	5.11	2.70	80.72
Morphological	Last letter correct	25.25	8.04	1.58	34.87
	Error in last letter	55.17	6.99	2.96	65.13
Hybrid	last letter correct	12.04	5.61	1.33	18.98
	Error in last letter	72.69	5.46	2.87	81.02

We have noticed that the results of the hybrid and partial experiments are closed to each other and the results of the statistical and morphological experiments are closed to each other. We get the best improvement of the results when we do the hybrid and partial experiments comparing with the result of the statistical and morphological experiments.

The table shows that the percentages of the words that are having one diacritization error are about 80% for the statistical and morphological experiments and about 85% for the hybrid and partial experiments. The percentages of the words that are having two errors are about 15% for the statistical and morphological experiments and about 11% for the hybrid and partial experiments. The percentages of the words that are having three or more errors are about 4% for all experiments.

Also, the percentages of the words that are having errors in the last letter are about 65% for the statistical and morphological experiments and about 80% for the hybrid and partial experiments. This is an indication that the predicting of syntactic diacritics is hard (Zitouni, et al., 2006) and will increase the DER and WER values.

We have manually checked 200 error samples. Table 12 shows the target sequences and the output sequences of six error samples and we underline the words that have error. We observed that about 28% of the words that have diacritic error are a valid Arabic verb or noun. For example, the target word verb of sample 1 is تُؤَمِّنُ (provide) and the output word verb is تُؤْمِنُ (believe). The output word verb of sample 2 is تَمَّت (completed) and the target word verb is تَمُتُّ (related or relevant). The output word noun of sample 3 مَلائِمَ is not a valid Arabic word and the target word noun is مُلائِمٍ (suitable).

Table 12. Sample sequences that have errors

Sample	Target Sequence	Output Sequence
1	كَما تُؤَمِّنُ بَعْضَ المُساعَدَةِ فِي الرِّعايَةِ الصِّحِّيَّةِ	كَما تُؤْمِنُ بَعْضَ المُساعَدَةِ فِي الرِّعايَةِ الصِّحِّيَّةِ
2	وَهِيَ لا تَمُتُّ بِصِلَةٍ لِأَرْكانِ الدِّينِ الإِسْلامِيِّ الحَنِيفِ	وَهِيَ لا تَمَّت بِصِلَةً لِأَرْكانِ الدِّينِ الإِسْلامِيِّ الحَنِيفِ
3	مُلائِمِ لِإِطْلاقِ الشَّبابِ	مَلائِمَ لِإِطْلاقِ الشَّبابِ
4	فِي لُبْنانَ وَمُصِرِّ وَالعِراقِ	فِي لُبْنانَ وَمِصْرَ وَالعِراقِ
5	هُوَ مَصْدَرُ فَخْرِنا وَاِعْتِزازِنا	هُوَ مَصْدَرٌ فَخْرُنا وَاِعْتِزازُنا
6	وَذَكَّرَ بِمُبادَراتِ الأَمانَةِ العامَّةِ لِلْمَدارِسِ	وَذَكَرَ بِمُبادَراتِ الأَمانَةِ العامَّةِ لِلْمَدارِسِ

Sample 4 represents an example of diacrtization errors due to the target words of test sample. The target word is مُصِرِّ (insistent) and the output word is مِصْرَ (Egypt). We observed that about 2% of the words that have diacritic error are due to target words.

Sample 5 represents an example of diacrtization errors due to the complex words that have prefix, suffix, or both. The target word in sample 5 is the noun فَخْرِنا (honor) and it has pronoun suffix "naa" نا. Predicting diacritics of these complex words are hard because we must be sure from diacritics of the last letter of stem and the last letter of suffix and we have noticed that about 44% of words in this sample are complex words.

Sample 6 shows an example of predicting shadda diacritic. We have noticed that about 6% of the words in this sample are having shadda diacritic relative to all diacritics

and about 22% of output words can't predict the shadda diacritic of target words. But in some cases output words may wrongly predict shadda even that the target word didn't have shadda.

We also do experiments to see the effect of having shadda in input sequences because shadaa diacritic is harder to predict than other diacritics. We noticed that some Arabic texts contain shadda diacritic even though other diacritics are missing. We ran two experiments on LDC ATB3 using statistical experiment. In first run we have only shadda in input sequences (true shadada) and in the second run we don't have any diacritics and we want to predict all diacritics including shadda (predicted shadda). The results are shown in Figure 27. We found that having shadda in input sequences gives a better classification error rate with an improvement of 5% but we must predict shadda because it's one of the eight diacritics that we want to predict.

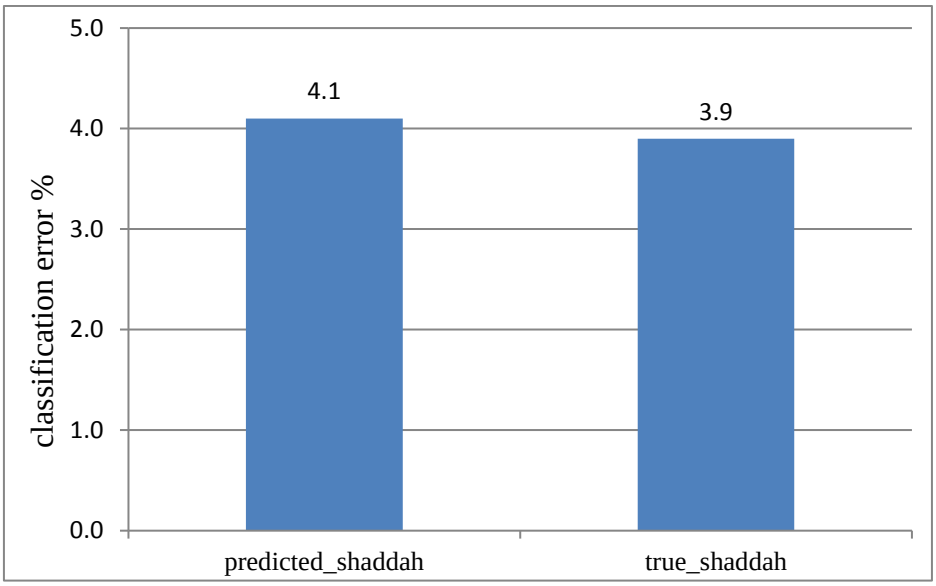

Figure 27. The effect of shadda diacritic on classification error.

CHAPTER VI

Conclusions and Future Work

The process of automatically adding diacritcs to Arabic text is called automatic diacrtization. Its importance lies in reaching to the correct understanding and analysis of the text because using different diacritical marks over the same word constants result in a different meaning and pronunciation for this word. For example: ذهب (this word is without diacritic) may means (ذَهَبْ) gold or (ذَهَبَ) went but adding diacritics will solve this issue.

The techniques used in solving the diacritization problem can be classified into three approaches: rule-based, statistical, and hybrid approaches. To the best of our knowledge, Abandah et al., (2015) proposed for the first time the using of RNN sequence transcription method to solve diacrtization problem with deep bidirectional LSTM architecture. In order to improve diacritization accuracy, we can use morphological analyzer that will add linguistic information to the input sequences and help RNN transcription learning.

In this work, we use MADAMIRA morphological analyzer to have partially diacritized and tokenized data and then we use this data for CURRENNT, the first publicly-available tool that performs parallel implementation of deep LSTM RNN architecture, to produce a fully diacritized data and to speedup training phase. The deep bidirectional LSTM architecture is used to deal with dependency between words in long sentences and in both directions. We found that the best configuration was the using of one-to-one transcription method with three hidden layers (each 250 neuron) and the best confidence value was 60%.

Our diacritization system achieved state-of-the-art results on LDC ATB3 and gives 34% DER improvement and 26% WER improvement over the best reported hybrid system of Said et al. (2013).

In this work, we used some post processing correction techniques to improve diacritization accuracy. In future, we can add more post processing correction techniques by using additional rules of Arabic text diacritization such as shaddah and sukun diacritics don't appear at initial letter of word, tanween diacritics appear only at last letter of word, (آ ,ا and ى) letters don't have any diacritics, (ء ,ة and إ) letters don't have shadda diacritic, (و ,ا and ي) letters are preceding by letters that have diacritics similar to the vowel i. e, fatha, damma and kasrah, respectively. Also we will consider the using of CURRENNT library and MADAMIRA morphological analyzer when we work on a large training corpus. We have noticed the impact of using this hybrid approach on reducing the DER value of the statistical experiment from 3.62 to 2.39 when we use this hybrid approach. CURRENT will speedup the training time of Bidirectional LSTM and it's a great tool for sequence transcription. On LDC ATB3 data, we achieved a 326x speedup over the using of RNNLIB library. One of CURRENT's drawbacks, it is difficult to compile with modern version of compilers, uses old CUDA runtime, and not actively developed.

Our research of the computer processing of Arabic text is important and useful because we have many commercial applications of Arabic automatic diacritization system and we must be sure from these applications. Also, we noticed that many earlier researches of diacritization problem were done by non-Arabic speakers and they solved this problem using statistical approaches such as Gal in (Gal, 2002), (Vergyri and Kirchhoff, 2004), (Nelken and Shieber, 2005) and others. We appreciate their works but

we are Arabic native speakers and we can understand this language so we must serve our language (the language of Quran) and this language is used by approximately 1.6 billion Muslim in the world. We must be aware and honest in serving this language.

REFERENCES

Abandah, G. 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**, Available on doi:10.1007/s10032-015-0242-2 © Springer-Verlag Berlin Heidelberg.

Abandah, G. A., Jamour, F. T., and Qaralleh, E. A. (2014), Recognizing handwritten Arabic words using grapheme segmentation and recurrent neural networks. **International Journal on Document Analysis and Recognition (IJDAR)**, 17(3), pp. 275-291.

Arabiyat, A. (2015), Automatic Arabic Text Diacritization Using Recurrent Neural Networks. Unpublished Master Thesis, University of Jordan, Amman, Jordan.

Azmi, A. and Almajed, R. (2013), A survey of automatic Arabic diacritization techniques. **Natural Language Engineering**, Available on CJO doi:10.1017/S1351324913000284.

Boudchiche, M., Mazroui, A., Bebah, M. O. A. O., Lakhouaja, A., and Boudlal, A. (2016). AlKhalil Morpho Sys 2: A robust Arabic morpho-syntactic analyzer. **Journal of King Saud University-Computer and Information Sciences**, Available on doi:10.1016/j.jksuci.2016.05.002.

Buckwalter, T. (2004), Buckwalter Arabic Morphological Analyzer, v2.0 edn, **Linguistic Data Consortium**, Philadelphia.

Burges, C. J. (1998), A Tutorial on Support Vector Machines for Pattern Classification, **Data mining and Knowledge Discovery**, 2(2), pp.121-167.

Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. (2014), Learning phrase representations using rnn encoder-decoder for statistical machine translation, **arXiv** preprint arXiv:1406.1078.

Ciresan, D. C., Meier, U., Gambardella, L. M., and Schmidhuber, J. (2010), Deep, big, simple neural nets for handwritten digit recognition, **Neural computation,** 22(12), pp. 3207-3220.

Cook, S. (2012). **CUDA programming: a developer's guide to parallel computing with GPUs**. 1st edition. San Francisco, CA: Morgan Kaufmann Publishers.

El Hihi, S. and Bengio, Y. (1995), Hierarchical Recurrent Neural Networks for Long-Term Dependencies, Proceedings of the **NIPS,** pp. 493-499.

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

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

El-Sadany, T. and Hashish, M. (1988), Semi-Automatic Vowelization of Arabic Verbs, Proceedings of the **10th National Computer Conference**, Saudi Arabia, pp. 725-732.

Farghaly, A. and Shaalan, K. (2009), Arabic natural language processing: Challenges and solutions. **ACM Transactions on Asian Language Information Processing (TALIP),** 8(4), 14.

Gal, Y. (2002), An HMM Approach to Vowel Restoration in Arabic and Hebrew, Proceedings of the **Workshop on Computational Approaches to Semitic Languages**. Philadelphia, pp. 27–33.

Gers, F. A., Schraudolph, N. N., and Schmidhuber, J. (2003), Learning precise timing with LSTM recurrent networks, **The Journal of Machine Learning Research**, 3, pp.115-143.

Graves, A. (2008), **Supervised Sequence Labelling with Recurrent Neural Networks**. PhD thesis, Technical University Munich, Munich, Germany.

Graves, A., Liwicki, M., Bunke, H., Schmidhuber, J., and Fernández, S. (2008), Unconstrained on-line handwriting recognition with recurrent neural networks. In **Advances in Neural Information Processing Systems (NIPS**), pp. 577-584.

Graves, A., Mohamed, A. R., and Hinton, G. (2013), Speech recognition with deep recurrent neural networks, Proceedings of the **Acoustics, Speech and Signal Processing (ICASSP)**, IEEE International Conference, pp. 6645-6649.

Graves, A. and Schmidhuber, J. (2005), Framewise phoneme classification with bidirectional LSTM and other neural network architectures, **Neural Networks,** 18(5-6), pp. 602-610.

Habash, N. and Rambow, O. (2007), Arabic Diacritization Through Full Morphological Tagging, Proceedings of the **North American Chapter of the Association for Computational Linguistics (NAACL)**, pp. 53-56.

Habash, N., Soudi, A., and Buckwalter, T. (2007), On Arabic transliteration, In **Arabic computational morphology,** Springer Netherlands**,** pp. 15-22.

Hermans, M. and Schrauwen, B. (2013), Training and analysing deep recurrent neural networks, Proceedings of the **Neural Information Processing Systems,** pp. 190-198.

Hifny, Y. (2012), Smoothing techniques for Arabic diacritics restoration, Proceedings of the **12th Conference on Language Engineering (ESOLEC '12)**, Cairo, Egypt.

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

Huang, P. S., He, X., Gao, J., Deng, L., Acero, A., and Heck, L. (2013), Learning deep structured semantic models for web search using clickthrough data, Proceedings of the 22nd **ACM international conference on Conference on information & knowledge management,** pp. 2333-2338.

Jamro, W. A., Shaikh, H., and Mahar, J. A. (2016). Comprehensive Analysis of Neural Network Techniques in Computational Linguistic Applications. **Asian Journal of Engineering, Sciences and Technology**, 15.

Kheder, M. (1999), Use of Neural Networks in Arabic Text Transliteration. Proceedings of the 4th **International Conference on Recent Trends in Computer Science Applications & Information Systems,** Philadelphia University Amman, Jordan, pp. 13-14.

Kirchhoff, K., Bilmes, J., Das, S., Duta, N., Egan, M., Ji, G., ... and Vergyri, D. (2002), Novel approaches to Arabic speech recognition: report from the 2002 **Johns-Hopkins summer workshop**, ICASSP'03, Hong Kong, vol.l, pp. 344-347.

Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In Proceedings of **the eighteenth international conference on machine learning, ICML**, Vol. 1, pp. 282-289.

Lewis, M. P., Simons, G. F., and Fennig, C. D. (2016), **Ethnologue: Languages of the World**, 19th edn. SIL International, Dallas, Tex. Online version: http://www. ethnologue. com.

Maamouri, M., Bies, A., Buckwalter, T., and Mekki, W. (2004), The Penn Arabic treebank: Building a large-scale annotated Arabic corpus, Proceedings of the **NEMLAR conference on Arabic language resources and tools,** Cairo, Egypt, pp. 102-109.

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

Mikolov, T., Karafit, M., Burget, L., Cernocky, J., and Khudanpur, S. (2010), Recurrent neural network based language model. In **Eleventh Annual Conference of the International Speech Communication Association**, pp. 1045–1048.

Nelken, R. and Shieber, S. (2005), Arabic Diacritization Using Weighted Finite-State Transducers, Proceedings of the **Workshop on Computational Approaches to Semitic Languages**. University of Michigan, Ann Arbor, pp. 79–86.

Nizar Habash's Home Page. Last accessed November 1th 2016. Available from **http://www.nizarhabash.com/**.

Nvidia Geforce Page. Last accessed December 28th 2016. Available from **http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-580/specifications**

Pasha, A., Al-Badrashiny, M., Kholy, A. E., Eskander, R., Diab, M., Habash, N., Pooleery, M., Rambow, O., and Roth, R. (2014). MADAMIRA: A Fast, Comprehensive Tool for Morphological Analysis and Disambiguation of Arabic. In Proceedings of the **Language Resources and Evaluation Conference (LREC)**, Reykjavik, Iceland., Vol. 14, pp. 1094-1101. Online version of MADAMIRA: **http://nlp.ldeo.columbia.edu/madamira/**.

Pineda, F. (1987), Generalization of back-propagation to recurrent neural networks. **Physical review letters**, 59(19), 2229.

Rashwan, M. A., Al-Badrashiny, M. A., Attia, M., Abdou, S. M., and Rafea, A. (2011), A Stochastic Arabic Diacritizer Based on a Hybrid of Factorized and Unfactorized Textual Features, **IEEE Transactions on Audio, Speech, and Language Processing**, vol.19, no.1, pp.166-175.

Robinson, A (1994), An application of recurrent nets to phone probability estimation, **Neural Networks, IEEE Transactions**, 5(2), pp.298-305.

Said, A., El-Sharqwi, M., Chalabi, A., and Kamal, E. (2013), A hybrid approach for Arabic diacritization. A Hybrid Approach for Arabic Diacritization. Proceedings of the **Natural Language Processing and Information Systems,** Springer Berlin Heidelberg, pp. 53-64.

Schuster, M. and Paliwal, K. (1997), Bidirectional recurrent neural networks, **Signal Processing, IEEE Transactions** on, 45(11), pp. 2673-2681

Shaalan, K. (2010), Rule-based approach in Arabic natural language processing, **International Journal on Information and Communication Technologies (IJICT)**, Serial Publications 3(3):11–9.

Shahrour, A., Khalifa, S., and Habash, N. (2015), Improving Arabic Diacritization through Syntactic Analysis, Proceedings of the 2015 **Conference on Empirical Methods in Natural Language Processing**. Lisbon, Portugal, pp. 1309–1315.

Sutskever, I., Martens, J., and Hinton, G. E. (2011). Generating text with recurrent neural networks. In Proceedings of the **28th International Conference on Machine Learning (ICML-11)**, pp. 1017-1024.

Svozil, D., Kvasnicka, V., and Pospichal, J. (1997). Introduction to multi-layer feed-forward neural networks. **Chemometrics and intelligent laboratory systems**, 39(1), pp. 43-62.

Tan, P. N., Steinbach, M., and Kumar, V. (2006**). Classification: basic concepts, decision trees, and model evaluation**. Introduction to data mining, 1st edn., pp. 145-205. Boston: Pearson Addison Wesley.

Vergyri, D. and Kirchhoff, K. (2004), Automatic Diacritization of Arabic for Acoustic Modeling in Speech Recognition, Proceedings of the **20th International Conference on Computational Linguistics**. Geneva, pp. 66–73.

Weninger, F., Bergmann, J., and Schuller, B. (2015), Introducing CURRENNT: The Munich Open-Source CUDA RecurREnt Neural Network Toolkit**. Journal of Machine Learning Research**, vol.16, pp.547-551. Software available from: **https://sourceforge.net/projects/currennt/**.

Zayyan, A. A., Elmahdy, M., binti Husni, H., and Al Ja'am, J. M. (2016). Automatic diacritics restoration for modern standard arabic text. In **Computer Applications & Industrial Electronics (ISCAIE), 2016 IEEE Symposium on. IEEE**, pp. 221-225.

Zerrouki, T. (2011), Tashkeela: Arabic vocalized text corpus, Retreived April 5, 2015, from **http://aracorpus.e3rab.com/.**

Zitouni, I., Sorensen, J. S., and Sarikaya, R. (2006), Maximum entropy based restoration of Arabic diacritics, Proceedings of the **21st International Conference on Computational Linguistics** and 44th Annual Meeting of the Association for Computational Linguistics (ACL), Sydney, Australia, pp.577-584.

Appendix A

Unicode and Buckwalter transliteration of Arabic characters.

Arabic Character	Shape	Unicode	Buckwalter
Arabic Letter HAMZA	ء	U+0621	"
Arabic Letter ALEF with MADDA above	آ	U+0622	|
Arabic Letter ALEF with HAMZA above	أ	U+0623	>
Arabic Letter WAW with HAMZA above	ؤ	U+0624	&
Arabic Letter ALEF with HAMZA BELOW	إ	U+0625	<
Arabic Letter YEH with HAMZA above	ئ	U+0626	}
Arabic Letter ALEF	ا	U+0627	A
Arabic Letter BEH	ب	U+0628	B
Arabic Letter TEH MARBUTA	ة	U+0629	P
Arabic Letter THE	ت	U+062A	t
Arabic Letter THEH	ث	U+062B	v
Arabic Letter JEEM	ج	U+062C	j
Arabic Letter HAH	ح	U+062D	H
Arabic Letter KHAH	خ	U+062E	x
Arabic Letter DAL	د	U+062F	d
Arabic Letter THAL	ذ	U+0630	*
Arabic Letter REH	ر	U+0631	r
Arabic Letter ZAIN	ز	U+0632	z
Arabic Letter SEEN	س	U+0633	s
Arabic Letter SHEEN	ش	U+0634	$
Arabic Letter SAD	ص	U+0635	S
Arabic Letter DAD	ض	U+0636	D
Arabic Letter TAH	ط	U+0637	T
Arabic Letter ZAH	ظ	U+0638	Z
Arabic Letter AIN	ع	U+0639	E
Arabic Letter GHAIN	غ	U+063A	g
Arabic Letter FEH	ف	U+0641	f
Arabic Letter QAF	ق	U+0642	q
Arabic Letter KAF	ك	U+0643	k
Arabic Letter LAM	ل	U+0644	l
Arabic Letter MEEM	م	U+0645	m
Arabic Letter NOON	ن	U+0646	n
Arabic Letter HEH	ه	U+0647	h
Arabic Letter WAW	و	U+0648	w
Arabic Letter ALEF MAKSURA	ى	U+0649	Y
Arabic Letter YEH	ي	U+064A	y

Arabic Character	Shape	Unicode	Buckwalter
Arabic FATHATAN	ً	U+064B	F
Arabic DAMMATAN	ٌ	U+064C	N
Arabic KASRATAN	ٍ	U+064D	K
Arabic FATHA	َ	U+064E	a
Arabic DAMMA	ُ	U+064F	u
Arabic KASRA	ِ	U+0650	i
Arabic SHADDA	ّ	U+0651	~
Arabic SUKUN	ْ	U+0652	o

طريقة هجينة للتشكيل الآلي للنصوص العربية باستخدام الشبكات العصبونية ذات التغذية الراجعة

إعداد
سبا امين عبدالرحمن القضاه

المشرف
الدكتور غيث علي عبندة

ملخــص

اللغة العربية هي لغة سامية وواحدة من أقدم اللغات في العالم. في هذه اللغة، يمكن أن يكون لأحرف الكلمة الواحدة العديد من علامات التشكيل المختلفة، وهذا سيؤدي للحصول على كلمات مختلفة في المعنى. أيضا، فإن عدم وجود هذه الحركات تجعل هذه اللغة غامضة للقراءة من قبل غير الناطقين بها وللمعالجة من قبل النظام الآلي مثل التعرف الآلي على الكلام ، وتحويل النص المكتوب إلى كلام.

إضافة علامات التشكيل إلى النص العربي هو خطوة مهمه في مجال معالجة اللغات الطبيعية مثل اللغه العربية وقد قام العديد من الباحثين بعمل العديد من الأبحاث لتطوير نظام التشكيل الآلي للنصوص العربية لأن التشكيل اليدوي يعتبر غير فعال ومضيعة للوقت.

الطرق المستخدمة في مجال التشكيل يمكن تصنيفها إلى ثلاثة طرق: طرق تعتمد على قواعد النحو و الصرف، طرق إحصائية، وهناك النوع الهجين. النوع الهجين يجمع بين المعرفة اللغوية والطرق الإحصائية من أجل استغلال مزايا هاتين الطريقتين والحصول على نتائج أفضل.

واحدة من أهم الطرق الإحصائية التي تستخدم لحل مسألة التشكيل هي طريقة تحويل المتسلسلات التي تستخدم الشبكات العصبونية ذات التغذية الراجعة والمبنيه بإستخدام الشبكات العميقة ثنائية الإتجاه ذات وحدات تخزين طويلة وقصيرة الأمد. في هذا البحث، تم تقديم مقترح جديد للتشكيل الآلي للنصوص العربية بإستخدام الطريقة الهجينة. قمنا باستخدام محلل صرفي يدعى مدى ميرا و الذي يعتبر من اهم المحللات الصرفية المستخدمة في هذا المجال وقمنا أيضا باستخدام أداة تستخدم الشبكات العصبونية ذات التغذية الراجعة والتي عالجت مشكلة التشكيل بطريقة تحويل المتسلسلات وأيضا قامت بتسريع تدريب الشبكات العصبونية. على حد علمنا، تعتبر هذه الاداة كأول أداة تتوفر لديها المقدرة على تسريع تدريب الشبكات العميقة ثنائية الإتجاه ذات وحدات تخزين طويلة وقصيرة الأمد. وأيضا قمنا بدراسة أثر إضافة معلومات لغوية للشبكات العصبونية على ثلاثة مستويات مختلفة.

حقق النظام المقترح لدينا نتائج هي الأكثر دقة لغاية الآن في مجال أنظمة التشكيل الهجينة وذلك باستخدام كتاب جمعية البيانات اللغوية الجزء الثالث. لقد حقق هذا النظام نسبة خطأ في التشكيل على مستوى الحرف 2.39% ونسبة خطأ على مستوى الكلمة 8.4%.واذا لم نحتسب حركة التشكيل على الحرف الأخير فإن نسبة الخطأ على مستوى الحرف تصبح 0.78% و 2.3% على مستوى الكلمة. واستطاع أن يقلل نسبة الخطأ على مستوى الحرف بمقدار 34% وعلى مستوى الكلمة بمقدار 26% مقارنة مع أفضل النتائج المنشورة في هذا المجال.