TOWARDS A ROBUST UNDERSTANDING OF ARABIC WORD SENSE

by

ALI ALKhatlan

B.S., University of King Abdulaziz, Computer Science, KSA, 2008

M.S., University of Colorado at Denver, Computer Science, USA, 2013

A dissertation submitted to the Graduate Faculty of the

University of Colorado Colorado Springs

in partial fulfillment of the

requirement for the degree of

Doctor of Philosophy

Department of Computer Science

2019
This dissertation for Doctor of Philosophy degree by

Ali Alkhatlan

has been approved for the

Department of Computer Science

by

Jugal Kalita, Chair

Jonathan Ventura

Khang Nhut Lam

Rory Lewis

Sudhanshu Semwal

12/18/2019

Date
ABSTRACT

Word Sense Disambiguation (WSD) is a task which aims to identify the meaning of a word given its context. This problem has been investigated and analyzed in depth in English. However, work in Arabic has been limited despite the fact that there are half a billion native Arabic speakers. In this work, we present multiple approaches for the problem of WSD in Arabic, inspired by recent developments and successes in learning word embeddings with approaches such as GloVe, and Word2vec. The primary shortcoming of word embeddings is the single vector representation of a word’s meaning, although many words are polysemous. Our first contribution in this work is to computationally obtain an embedding for each sense, using an Arabic WordNet (AWN) to overcome the problem of WSD. We also compute word semantic similarity giving thought to multiple Arabic stemming algorithms. We tackle WSD from a different point of view that solves WSD by restoring Arabic diacritics to reduce the complexity of word disambiguation process. We make available a large pre-processed corpus that is ready to be used for further experiments and a WSD test data based onAWN¹, seeking to fill gaps in Arabic NLP (ANLP) to draw closer to the level of English NLP.

¹Word embeddings, and WSD test data are available in our GitHub https://github.com/LincLabUCCS/Arabic-Word-Sense-Disambiguation
DEDICATION

To my mother, father, and future family.
ACKNOWLEDGMENTS

First and foremost, I sincerely praise and thank God for giving me great support throughout my research. Second, I submit my gratitude to my beloved parents and close friends for their passionate support on this journey. Finally, I deeply thank my advisor Dr. Jugal Kalita for his effort and inspiration. This work is complete due to his daily mentorship and contributions.
# TABLE OF CONTENTS

## CHAPTER

I Introduction  
1.1 Overview ............... 1  
1.2 Motivation ................ 3  
1.3 Research Contributions .......... 4

II Literature Review  
2.1 State-of-the-art ............ 6  
2.1.1 Learning Representation .......... 7  
2.1.1.1 Word2vec .............. 8  
2.1.1.2 GloVe .............. 10  
2.1.2 Recurrent Neural Network ........ 10  
2.1.2.1 RNN Architectures .......... 11  
2.1.2.2 Attention Structure .......... 12  
2.1.2.3 Applications of RNN ........ 14  
2.2 Arabic Word Sense Disambiguation ........ 15  
2.2.1 Dictionary and Knowledge based Approach .......... 15  
2.2.2 Lesk Algorithm ............ 15  
2.2.3 Lesk Algorithm in Arabic .......... 16  
2.2.4 Supervised Methods .......... 17  
2.2.5 Other Approaches .......... 18
2.3 Conclusion ......................................................... 20

III Multi-Word Sense Embedding 21

3.1 Arabic Resources .............................................. 23
  3.1.1 Text Corpus ................................................. 23
  3.1.2 Arabic WordNet ............................................. 24
3.2 Stemming the Corpus and Word Vector Initialization ................. 25
  3.2.1 Stemming Algorithms ...................................... 25
  3.2.2 Word Vector Initialization ............................... 26
3.3 Evaluation of Word Similarity ................................ 28
3.4 Proposed Algorithms for Sense Embeddings ......................... 30
  3.4.1 RETROFIT Algorithm .................................... 30
  3.4.2 ArabSenEmb Algorithm ................................ 31
3.5 WSD System and Test Data .................................. 32
  3.5.1 WSD System ............................................... 32
  3.5.2 WSD Test Data ............................................ 33
3.6 Result .................................................................. 34
3.7 Discussion and Conclusion ...................................... 35

IV Attention-based Sequence Learning Model for Arabic Diacritic Restoration 37

4.1 Introduction ...................................................... 37
4.2 Characteristics of Arabic ....................................... 40
4.3 Related Work ..................................................... 41
4.4 Approach .......................................................... 43
6.2.1.4 Modified Version of RETROFIT Algorithm  . . . . . . . 65
6.2.1.5 Modified Version of ArabSenEmb Algorithm . . . . . . . 66

6.2.2 Combining Word-level and Character-level Sequence to Sequence process for Arabic Diacritic Restoration . . . . . . . . . . . . . . . 66

6.2.2.1 Future Extension . . . . . . . . . . . . . . . . . . . . . 68

6.3 Conclusion of the Dissertation . . . . . . . . . . . . . . . . . . . . . . . . 69

References  71
<table>
<thead>
<tr>
<th>TABLE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Corpus statistics for each individual file and the combined file showing the origin country, the number of tokens, the number of unique tokens, and the file size for each.</td>
</tr>
<tr>
<td>3.2</td>
<td>Spearman rank order correlation between human relatedness judgment and model similarity. Bold and underlined scores show the best results overall, and underlined scores show comparable results.</td>
</tr>
<tr>
<td>3.3</td>
<td>Obtained results for the WSD task for both no stemming and Light10 vectors for both algorithms.</td>
</tr>
<tr>
<td>4.1</td>
<td>Statistics of the corpus used.</td>
</tr>
<tr>
<td>4.2</td>
<td>The DER of all diacritic (All) and at word ending (End) for LSTM and GRU with attention (Attentive) and without attention mechanism. Bold indicates the best result among the others in the same layer, bold and underscore indicate the best result overall.</td>
</tr>
<tr>
<td>4.3</td>
<td>The best result reported by related work using DER-All.</td>
</tr>
<tr>
<td>5.1</td>
<td>Possible diacritic marks for the Arabic word ( ).</td>
</tr>
<tr>
<td>5.2</td>
<td>Diacritic Procedure of the Sentence (waHat<del>aY Al—na taEotaqidu Al-maSAdiru &gt;an</del>a &gt;asobAba AIHAdivi qado takuwnu $axoSiy~apF ) . . .</td>
</tr>
<tr>
<td>5.3</td>
<td>All models, outputs of the diacriticized sentence (faAloEamalu Alr<del>iyaADiy</del>u SaEobN wa$saA}ikN wayaHotaAju AIY jaladN waSaborN). Bold characters/diacritic labels indicate the output mistakes.</td>
</tr>
</tbody>
</table>
5.4 Accuracy of WSD using all Arabic diacritic restoration models. . . . . . 59
LIST OF FIGURES

FIGURE

2.1 Submission counts of NLP papers in top well-known NLP conferences extracted from [1] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
2.2 General Architecture of Word2vec extracted from [2] . . . . . . . . . . . 9
2.3 General Architecture of Multiple RNN Architectures RNN, LSTM, and GRU. [3] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
2.4 Simplified Lesk algorithm for the Word Bank [4] . . . . . . . . . . . . . . 16
3.1 Output of our WSD system showing each synset with its cosine similarity to the context vector. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 33
4.1 Illustration of our attentive BiRNN model, showing the structure including the embedding, BiRNN, attention and output layers. . . . . . . . . . . . . . 46
4.2 The Transformer Inputs and Outputs. . . . . . . . . . . . . . . . . . . . . . 46
4.3 Arabic Diacritic Restoration Using Transformer Architecture. . . . . . . . 47
4.4 The confusion matrix of our 3-layers Attentive-GRU . . . . . . . . . . . . 52
6.1 The main concept of our proposed word-character level combination. . . . . 68
CHAPTER I

INTRODUCTION

1.1 Overview

Word Sense Disambiguation (WSD) is the ability to automatically determine the correct sense (meaning) of a word in context. The task of WSD has received a lot of attention in English. However, the amount of work in other languages such as Arabic is still limited. The global importance of Arabic should provide motivation to spur additional in-depth studies. This is especially true because we believe that the availability of tools and systems in other languages would dramatically affect the performance of downstream systems in English.

The significance of Arabic comes from the fact that Arabic is the fifth most widely spoken language with around half a billion speakers. In Arabic, the main source of ambiguity is the abundant necessity of diacritical marks to guide pronunciation [5], although the bulk of Arabic writing ignores the explicit use of diacritics, introducing ambiguity in the perceived meaning to even fluent native speakers. For example, the meaning of an Arabic
word (علم) varies greatly based on the types and positions of the usually unwritten diacritics (علم, science)(علم, Flag) (علم, understood) and (علم, teach).

WSD is a must-solve problem in NLP, which is considered an AI-complete problem [6]. In fact, WSD has received a great amount of effort over several decades, and there are multiple approaches to handle it. These approaches can be classified as knowledge-based, supervised, and unsupervised [6]. The WSD problem is not an end task, but it is an intermediate step for many applications [7]. As a result, it is likely that improvement in this core issue would have a great impact on a variety of applications such as Information Retrieval (IR) [8], Machine Translation (MT) [9], and Text Classification [10].

Recent advances in learning distributed representations of words (also called word embeddings) have gained considerable interest, popularity and use. Each word embedding is represented by a low-dimensional vector that is hopefully sufficient to encode its semantic and syntactic features. The use of word embeddings is currently ubiquitous, especially in explicitly computing semantic similarity between word pairs, making it beneficial for a variety of downstream applications [2, 11–14]. Word embeddings have been successfully deployed as inputs for tasks such as Sentiment Analysis [15], Information Retrieval [16] and Machine Translation [17].

Notwithstanding the success achieved by the use of embeddings, there is a single embedding for each word, which is not useful for performing computation when many words naturally have multiple meanings (senses) and are polysemous. Several research efforts attempt to extend available word embeddings to represent multiple senses as distinct vectors. Using word embeddings to disambiguate words in context is a natural choice because of their outstanding success in word representation. To the best of our knowledge,
all of the related published research has been dedicated to English. We refer to a few here [18–23].

Furthermore, Arabic is different from English in the way that written Arabic contains vowels expressed as diacritics. One of the main causes of word ambiguity in the Modern Standard Arabic (MSA) is missing diacritical marks. This habit in the bulk of Arabic writing encourages us to tackle the AWSD problem by restoring the missing diacritics. With the most recent advanced techniques in solving sequence to sequence problems, we can consider Arabic diacritic restoration as a sequence to sequence problem. Implementing different possible configurations to restore the missing diacritics will be our focus in Chapter IV.

1.2 Motivation

We know of no research work focused on the WSD task that employs word embeddings for Arabic. Our motivation for this work comes from two observations. First, the success in sense representation in English encourages us to perform similar work in Arabic for further validation. Second, we seek to use the power of word embeddings to overcome several Arabic WSD issues, as given below.

- Some Arabic WSD systems are based on the use of local context features to disambiguate words [24, 25]. We believe that these features are not sufficient to describe word senses, and that global context should be taken into account.

- Although the local context features are supposed to originate from the definition of each meaning, extracted completely or partially from a dictionary, Arabic lacks a good
digital dictionary. As a consequence, these approaches are not likely to be effective for downstream systems.

- Finally, several systems are based on supervised methods, which require sense-tagged annotated corpora. However, it is highly expensive and time consuming to develop such resources.

To overcome these issues, this work depends on two successful word embedding methods, namely Word2vec [2] and GloVe [14] to produce global contexts for words. We are also constrained to use the AWN [26], the only available electronic sense inventory database for Arabic.

1.3 Research Contributions

Research conducted in the field of Arabic NLP is still limited. Our work seeks to fill gaps in Arabic NLP to draw it closer to the level of English NLP. In this work, we produce two important resources involved in a ready-to-use large Arabic corpus which contains of a collection of texts extracted from MSA news, and a WSD test dataset which will be available for public use. In particular, we address the problem of word sense disambiguation in Arabic, with and without diacritic restoration.

We analyze word semantic similarity while taking into account multiple Arabic stemming algorithms. Because of the fact that Arabic is a highly inflectional language with a rich morphology, our idea is to present multiple experiments for this task in order to in-
vestigate if stemming improve performance of word semantic similarity compared with no
stemming text.

We present two approaches for producing multiple sense embeddings. Both ap-
proaches exploit the single embedding representation produced during the word semantic
similarity task. We also use the AWN as our sense inventory.

Finally, we present multiple approaches for Arabic diacritic restoration considering
this problem as a sequence to sequence problem. Multiple approaches with multiple config-
urations and architectures are implemented and evaluated. We eventually use these diacritic
models to restore the missing diacritics and use the diacritic restoration as an intermediate
step in performing WSD.

We believe that our work will partially fill the gaps between Arabic NLP and English
NLP. Our large, ready-to-use text corpus, and WSD test dataset will benefit the community
of Arabic NLP, alleviating the lack of such resources, and hopefully will become gold
standard for WSD task. In addition, we perform analysis of the most popular and well-
known Arabic stemming algorithms through the task of word semantic similarity. This will
enhance performance of real time applications such as information retrieval systems. Our
WSD approaches will improve Arabic NLP to overcome its current shortcomings. Finally,
we propose multiple approaches that exploit the more recent advancements in NLP using
seq2seq approaches involving in BiRNN, and the Transformer architecture.
CHAPTER II

LITERATURE REVIEW

In this chapter, we will review the current state-of-the-art methods and techniques in the field of NLP. We will also review existing methods and approaches for the problem for Arabic word sense disambiguation. In regard to the state-of-the-art, we will present only the techniques which we will use in our dissertation. The current state of the art relevant to our dissertation can be divided into three categories, namely learning representations and Recurrent Neural Network (RNN). The approaches found in the literature regarding to AWSD can be divided into three categories, namely dictionary and knowledge-based methods, unsupervised methods, and other statistical approaches.

2.1 State-of-the-art

In the following sections, we review the state-of-the-art techniques used in our dissertation: Learning representation, Recurrent Neural Network, and attention based structure.
2.1.1 Learning Representation

Representation learning in NLP refers to ways to extract features or information from unlabelled text. The goal of representation learning is to obtain a semantic meaning of words and represent them in a dense vector, alternatively called word embedding. Dense vectors are low dimensional vectors which are more efficient representations, unlike the older techniques where semantic meaning of a word was represented in a sparse vector using techniques such as Term-document and term-term matrices, using Pointwise Mutual Information (PMI) and Term frequency-inverse document frequency (TF-IDF).

Word embeddings represent words in low dimensional numerical vectors, which then result in words that have similar meaning appearing near each other in the vector space. These embeddings use a predefined vector space of size, say 300, which is much smaller than the size of the vocabulary. According to Goldberg [27] states the main advantages of dense vectors as “One benefit of using dense and low-dimensional vectors is computational: the majority of neural network toolkits do not play well with very high-dimensional, sparse vectors. However, this is just a technical obstacle, which can be resolved with some engineering effort. The main benefit of the dense representations is in generalization power: if we believe some features may provide similar clues, it is worthwhile to provide a representation that is able to capture these similarities”

Recently, there has been a rapid increases of the use of neural networks in Artificial Intelligence in areas such as computer vision, and NLP. In the recent years, the use of neural networks in NLP task has gained a huge amount of interest, producing outstanding results. This trend started around 2013 after a paper published by Mikolov et al., [2] from Google,
following another paper by Pennington et al., [14] from Stanford. Figure 2.1 shows the rapid growth of interest in NLP in the preeminent NLP conferences.

![Figure 2.1: Submission counts of NLP papers in top well-known NLP conferences extracted from [1]](image)

**2.1.1.1 Word2vec**

Word2vec proposed by Mikolov et al., [2] is a main breakthrough in the field of NLP in recent years. It aimed to obtain distributed representations of words in a vector space, using a neural network architecture. Word2vec follows the distributional hypothesis that words that tend to appear in the same context tend to have similar meanings. Thus, these word embeddings try to capture the characteristics of the surrounding words. Mathematical operations such as finding how those words embeddings are similar can be done using any similarity measurements such as cosine similarity or Euclidean distance.

Figure 2.2 shows the general concept of word2vec, which comes in two models, Continuous Bag of Words (CBOW) and Skip-gram. Both architectures are fully connected neural networks containing one hidden layer. The intuition behind this architecture is that rather than following the approach of counting how many times the word \( w \) appears in
the context of another word. Instead, the models are trained explicitly to compute the probability of the word $w$ appear in the same context where the word appears.

CBOW learns to predict the probability of a word given its context; this context can be determined by the user. The input layer is a 1-hot representation of the surrounding words of the word for which that the model attempts to find the probability of. The middle layer is the sum of all input context words. The last layer is a softmax layer that computes the probability of the words in the vocabulary dictionary and sum it to 1.

Skip-gram follows the same concept as CBOW, but Skip-gram reverses the direction of training compared to CBOW. The goal of Skip-gram is to predict the context words given a word. Skip-gram attempts to find the probability of context words given a word. The input layer is a 1-hot representation of a word, the hidden layer is the same as of CBOW. The last layer is the softmax that assigns probability of the correct target words, context words.
2.1.1.2 GloVe

GloVe stands for Global Vector for word representation, and is another famous word vector representation algorithm introduced by Pennington et al. [14]. It is a count-based model which combines both global matrix factorization and a local context window resulting in a global log-bilinear regression model with a weighted least-squares cost function. GloVe is different from word2vec regarding the main concept of obtaining the representation vector. GloVe is a count-based approach unlike word2vec which is a predictive model.

GloVe is based on a co-occurrence matrix of words which is first constructed a big matrix that counts how many times a word appears in the context of other words. The number of dimensionality in this case, of course, is big. Matrix factorization then takes a place in order to reduce the dimensionality of the co-occurrence matrix, obtaining a new matrix of words and their associated features.

Word embeddings like Word2vec and GloVe are very essential in NLP tasks, leading to great improvements. These embeddings can be used explicitly or implicitly for many NLP tasks. For a good review of the applications of word embeddings, we refer the reader to a recent and comprehensive survey [28]

2.1.2 Recurrent Neural Network

Recurrent Neural Networks (RNN) represent one of the major deep learning architectures used in NLP. It solves problems that need sequential data by applying the same computation at each time step (token). Each output in each time step is dependent on the previous time steps; this is why it is called recurrent. The general framework of RNNs is
to receive sequence of data. Each RNN network produces a fixed-size vector for the current sequence, which is fed the next token in the same sequence. The ultimate objective of RNN is to capture interesting information from the input sequence and map it to the correct output for prediction.

2.1.2.1 RNN Architectures

RNNs come in different models, each of which attempts to overcome certain shortcoming. A Simple RNN as shown in the leftmost diagram in Figure 2.3, is the simplest version of RNN architectures. It simply takes into account the current input at this current time step $x_t$ and the previous time output $h_{t-1}$, passing them through an activation function. Long Short Term Memory (LSTM) and Gated Recurrent Units (GRU) both handles the problem of vanishing gradients which makes difficult for Simple RNNs to remember information from the past layers. The use of LSTMs and GRUs handle this problem through gates, which control information flows through the cell. An LSTM has three gates: input, output and forget gate; on the other hand, GRU has two gates: reset and update gates. GRU and LSTM will be discussed in depth in Chapter V.

Figure 2.3: General Architecture of Multiple RNN Architectures RNN, LSTM, and GRU. [3]
2.1.2.2 Attention Structure

Attention mechanisms represent one of the breakthrough concepts in the field of deep learning in recent years to overcome long distance dependencies. This idea was first was born with the machine translation task, implemented in the encoder-decoder framework [29]. Since an encoder receives a series of inputs and then encodes the information contained in the entire sequence in one fixed vector, this approach suffers from the incapability of memorizing long distance dependencies.

The first work that invented this approach was for machine translation by Bahdanau et al [29]. Attention mechanisms attempt to produce a certain alignment between the input and the output texts. The main idea behind attention mechanisms is to mimic human behavior by giving special attention to certain parts of the input when producing a target output. This approach allows one to augment the context vector by looking at not only the information extracted from the inputs, but also taking into account individual tokens of the decoder.

The machine translation task used the first attention mechanism [29] to improve the performance even over long distances. The encoder was a bi-RNN allowing to read entire input forward and backward. This attention mechanism propagates an attentive score for each decoder token for each individual target output, providing the model the capability to judge which individual input is how important to this current output. Eventually, the model is able to enrich the context vector with the information regarding the entire sequence as well as the target outputs. Another attention mechanism approach for machine translation was developed by Luong et al., [30], who proposed two attention approaches; global and
local approaches. The global approach looks at all source positions and the local one attends to only a subset of source positions at a time. Both approaches showed noticeable improvement in machine translation as well as many other NLP tasks.

Attention mechanisms can work well for any NLP task that falls into seq2seq paradigm, such as text summarization. Rush et al. [31] applied a similar approach on abstractive text summarization, which conditions each output in the summary to the input sentence. This approach showed significant performance improvement compared to several baselines. A similar approach for abstractive text summarization was developed by Nallapati et al. [32] using an attentive encoder decoder architecture.

The paper titled Attention Is All You Need [33] was a major breakthrough in NLP and has gained much popularity. It presented a novel architecture in 2017 by Google using attention mechanisms, totally disposing of recurrences and convolutions in the encoding step to solve the sequence transduction task. The architecture of Transformer was born for the machine translation task, which outperforming baseline models, being more parallelizable and requiring less time for training. The Transformer architecture maps a sequence of inputs to a sequence of outputs using stacked self-attention units in both the encoder and decoder.

Since the publication of this work, the Transformer architecture has produced the state-of-the-art results. It has been used in popular transformer architectures like BERT [34] and GPT [35], for Natural Language Understanding (NLU) and Natural Language Generation.
2.1.2.3 Applications of RNN

RNNs have gained a great deal of interest in the past few years because of the ability to model sequence to sequence (Seq2seq) data type, as well as being dynamic regarding the input types whether they are characters, words, sentence or paragraphs. Many NLP tasks have benefited from the use of RNNs with significant improvements, reporting the results as the state of the art.

It is difficult to list all applications of RNN in the field, however we list some of the most popular tasks in NLP such as named entity recognition (NER), sentiment analysis, part of speech tagging (POS), and word sense disambiguation (WSD).

Chiu and Nichols [36] proposed a method for name entity recognition, using word- and character-level features using a hybrid bidirectional LSTM and CNN architecture. Another work for NER was performed by Lample et al. [37] using BiLSTM and conditional random fields. Sentiment analysis is a problem in NLP that has benefited from the RNN structure. Lai et al. used RNNs and a CNN for sentiment analysis, RNNs aimed to capture contextual information and the CNN to represent the text [38]. Another well-known approach for sentiment analysis used LSTMs to represent levels of granularity including words, sentences, and eventually whole documents in vector space [39].

Part-of-speech tagging is another useful task and recently it has been treated as a sequence labelling problem. Plank et al. [40] performed POS tagging with bi-LSTMs with word, character, and unicode byte embeddings across 22 languages, achieving the state-of-
the-art performance. Wang et al. [41] proposed to use bi-LSTM with word embeddings for POS tagging, achieving a state-of-the-art performance on the Penn Treebank WSJ test set. Last but not least, word sense disambiguation was treated as a sequence labelling approach using an RNN architecture. Raganato et al. [42] proposed a WSD end to end task applying bi-LSTMs and an encoder decoder architecture. They showed that sequence learning for WSD is an effective approach obtaining the state-of-the-art performance.

2.2 Arabic Word Sense Disambiguation

In the following sections, we review the literature on the subject of (AWSD). AWSD approaches range from dictionary based approach, supervised approach, and other statistical approach.

2.2.1 Dictionary and Knowledge based Approach

A dictionary based approach utilizes a dictionary or a knowledge inventory such as WordNet to disambiguate words. This approach was applied to Arabic with the well-known Lesk algorithm [43]. Both the original algorithm and the modified version were adapted for Arabic. In the following paragraphs, we explain the original Lesk algorithm as well as the implementation of the algorithm in Arabic.

2.2.2 Lesk Algorithm

The aim of the Lesk algorithm is to identify senses of ambiguous words in context by looking at overlaps between the definitions of senses with the terms in context. It follows the assumption that neighbor words in a given context may share the same topic. The
simple Lesk algorithm uses a machine readable dictionary to identify the senses of an ambiguous word by their definition words. The disambiguation process is to calculate the overlap between each sense with the context of neighbor words. The sense which is chosen is the sense that has the highest overlap count. Figure 2.1 shows an example of the Lesk algorithm with the word bank.

![Figure 2.1: Simplified Lesk algorithm for the Word Bank][4]

The sentence in the example presented in Figure 2.1 contains two words, *deposits* and *mortgage* that overlap with the first meaning of the word *bank*. All possible meanings, gloss, and examples were extracted from the English WordNet.

### 2.2.3 Lesk Algorithm in Arabic

Both the original Lesk algorithm and modified versions have been implemented in Arabic. Two studies analyzed this approach. The following presents both studies.

Zouaghi et al. [44] developed an Arabic word sense disambiguation system that uses the Lesk algorithm along with information retrieval measures. The candidate sense was chosen by the overlap between the current use context and words in each sense definition of the word to be disambiguated. The authors used 10 ambiguous words in their experiments. The glosses (definitions) of these ambiguous words were extracted from Al-Mu'jam...
al-Wasit [45], a non-digitized Arabic dictionary. However, Al-Mu‘jam al-Wasit does not have sufficient glosses for each sense; therefore, AWN was used to extract synonyms for ambiguous words. Also, salient words, which are words appearing significantly more often in the use context of an ambiguous word sense were extracted from the used context. The accuracy of the proposed approach was 78%.

An other study conducted by Zouaghi et al. [25] deployed both the original Lesk algorithm and modified versions. In this work, both Al-Mu‘jam al-Wasit and AWN were used, and fifty ambiguous words were used for evaluation. The modified versions used graph measures such as Wu and Palmer [46], Resnik [47], Chodorow and Leacock [48], Jiang and Conrath [49], and Lin [50]. When there was an ambiguous word in a sentence, AWN was used to present all the synsets that correspond to this ambiguous word; then the graph measures were used to determine the synset to be chosen by computing the distance between the sentence words and the glosses of each synset. Synset that had the highest measure. In this work, the accuracy of the original Lesk algorithm was 59%, and that of the modified version was 67% with the Chodorow and Leacock measure.

The main drawback of this approach is using the Al-Mu‘jam al-Wasit dictionary. This dictionary is not electronically readable, which makes this approach unusable for real time usage. Also, some synsets in this dictionary do not have sufficient glosses, which makes this approach highly not reliable.

2.2.4 Supervised Methods

A supervised method requires a readable training data; therefore, for WSD the text training sample must be manually tagged with the appropriate sense for each ambiguous
Elmougy et al. [7] performed WSD experiments for Arabic using the Naive Bays classifier. In this work, the training set was collected from the World Wide Web containing 10 sample texts for the training phase and 10 testing samples for the testing phase for each ambiguous word. Each word’s sense was tagged manually by the author. During the training phase, the Naive Bays classifier was used to convert each sense of an ambiguous word into a numerical representation for the disambiguation phase. The disambiguation phase was performed using the output module of the training phase to compute a score for each sense of the ambiguous words and to choose the most appropriate sense. The accuracy of this work was 76%.

The main shortcoming of the supervised approach is the need for manually tagging the sense of the candidate ambiguous words; this is costly and time consuming. In addition, the number of the candidate ambiguous words for training is limited whereas the Arabic language has hundreds of thousands of words. This approach may achieve high accuracy, but is not really effective and reliable.

2.2.5 Other Approaches

Zouaghi et al. [24] presented a hybrid approach for Arabic WSD combining an unsupervised and a knowledge-based method. The lexical resources used in this work were from both Al-Mu’jam al-Wasit and AWN for 10 ambiguous words containing definitions (glosses) for each meaning of the words. The use of both resources overcame the problem that the dictionary does not contain sufficient by detailed glosses for each meaning. Moreover, to enhance each gloss, signatures consisting of surrounding words for each meaning...
were extracted from a corpus using the tf-idf measure. This approach achieves precision of 79%, and 71% F-score measure.

Bakhouche et al. [51] used the Ant Colony Optimization (ACO) algorithm [52] for Arabic WSD. ACO is a technique that probabilistically aims to solve computational problems by mimicking the movement of ants in finding good paths to a food source from a colony. The first application of ACO was for the Travelling Salesman Problem [53], and it was first adopted for the problem of English WSD, yielding better and faster results [54]. In this work, the dictionary was constructed by mapping the AWN to the English WN to overcome the lack of synsets’ definitions in AWN. The authors mimicked the ACO for Arabic WSD using both Lesk and ACO for local and global optimization respectively. This approach achieved a performance of 80% accuracy.

Menai [55] proposed an evolutionary approach with several genetic and memetic algorithms. The author used AWN as a sense inventory. The genetic algorithms attained a better performance compared to the memetic algorithms and the Naive Bays classifier, with an accuracy of 79%.

The most recent work proposed by Bouhriz et al. [56] differs from most of the previous approaches in that it is not limited to using only local context features defined by the words in the neighborhood of the ambiguous word. It took into consideration global context features for the ambiguous words, extracted from the full text. For the WSD process, both context vectors and local vectors were used to determine the candidate sense. The AWN was used as a lexical resource for senses. This approach obtained an accuracy of 74%.
2.3 Conclusion

Word sense disambiguation is one of the fundamental tasks in NLP which has been receiving a significant attention throughout the research history. Many approaches and methods have been proposed and performed to increase the performance. Recent advancement in NLP such as word embeddings has been the flavour of solving tasks such as WSD. This dissertation extends this phenomenal to other language such as Arabic.
CHAPTER III

MULTI-WORD SENSE EMBEDDING

In this chapter, we introduce our contributions on two popular NLP tasks, namely word semantic similarity and word sense disambiguation. The first part of the chapter presents work on the word semantic similarity task. The main objective for this part is to analyze the efficiency of the usage of Arabic stemming algorithms with no stemming of the text. In order to achieve this, there are multiple preprocessing steps that must be completed. The following describes the steps.

- **Preparing the text corpus as input data.** Our experiments require a large amount of input data in order to have meaningful outputs. The lack of a large, preprocessed and ready to use corpus is a considerable disadvantage at this time.

- **Stemming the corpus using multiple available Arabic stemming algorithms.** Because Arabic is highly inflectional and derivational in nature, we preprocess the entire corpus with several stemming algorithms.

- **Initializing word vectors.** Word vectors are produced for the original corpus (without stemming) and for each stemming algorithm. We use two word embedding
techniques for learning word vectors. The first one is a neural network learning model that includes the Skip-gram (SG) and Continuous Bag of Words (CBOW) approaches [2]. The second one is a global matrix factorization model, namely GloVe [14]

- **Evaluating word vectors.** Word vectors are semantically evaluated taking into account all word vectors including unstemmed vectors and each stemming algorithm’s vectors produced in the previous stage, a total of 12 experiments.

The second task we perform is word sense disambiguation that uses the vectors produced previously. Our objective is to extend the single vector to multiple vectors for each word’s senses. The following are the steps performed to achieve the WSD task.

- **Initializing sense-specific vectors.** Sense-specific vectors are produced based on AWN senses and the obtained sense-agnostic (single) vectors. Two algorithms are used and tested against each other.

- **Developing a WSD system.** For evaluation, we perform a WSD task based on the context associated with the ambiguous words. We should note that the field lacks a gold standard test dataset. Our test dataset is manually constructed and is available in our github for public use ¹.

This chapter describes both tasks in detail. Our work described briefly in Chapter VI focuses on the same problem of WSD. We aim to improve the accuracy of WSD proposing different approaches using multiple semantic similarity measures between synsets in the AWN.

¹https://github.com/LincLabUCCS/Arabic-Word-Sense-Disambiguation
3.1 Arabic Resources

To achieve our goal, our experiments require two Arabic resources: A large text corpus to produce the word embeddings, and a lexical database that contains words and senses to serve our purpose of producing sense embeddings. Thus, we use the AWN.

3.1.1 Text Corpus

In recent years, the increased availability of very large amounts of data has influenced all aspects of Artificial Intelligence (AI). In NLP, a text corpus is a substantial and fundamental data resource necessary for all corpus-based studies that build a variety of NLP models. To ensure better performance in learning word embeddings, the use of a large-scale corpus is recommended [57].

A publicly available Arabic corpus has been extracted from Arabic news websites that follow Modern Standard Arabic (MSA) [58]. To the best of our knowledge, this is the largest available Arabic corpus with over a billion and a half words. The corpus is extracted from 10 different news websites from 8 different Arab countries, ensuring a great diversity.

Text preparation is a crucial and time consuming stage in text analysis. Our objective is to preprocess the corpus and provide it as a ready-to-use resource for data analysis tasks. We perform the following for each individual file and then concatenate all files.

- Remove all XML tags and extract the body containing the news content only, omitting other entries such as dates, URLs, IDs and titles.
- Remove all English characters.
- Remove Arabic stopwords such as (كذلك , من , على , etc.)
• Remove diacritics such as Fattha (ٌ), Shaddah (ٍ), Damma (َّ), etc.

• Remove non-alphabetical characters.

• Convert digits to the symbol $Num$.

• Normalize beginning Alif (ِ) to plain Alif (ا), end letter (ی) to (َ),

(ٌ) to (ٍ). Statistics on the corpus after the preprocessing are shown in Table 3.1.

Table 3.1: Corpus statistics for each individual file and the combined file showing the origin country, the number of tokens, the number of unique tokens, and the file size for each.

<table>
<thead>
<tr>
<th>News Name</th>
<th>Country</th>
<th># of Tokens</th>
<th># of Unique Tokens</th>
<th>File size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alittihad</td>
<td>United Arab Emirates</td>
<td>138.980M</td>
<td>872.218K</td>
<td>1.6 GB</td>
</tr>
<tr>
<td>Almasry Alyoum</td>
<td>Egypt</td>
<td>92.462M</td>
<td>699.266K</td>
<td>1 GB</td>
</tr>
<tr>
<td>Almustaqbal</td>
<td>Lebanon</td>
<td>131.405M</td>
<td>897.164K</td>
<td>1.5 GB</td>
</tr>
<tr>
<td>Alqabas</td>
<td>Kuwait</td>
<td>231.696M</td>
<td>1.151M</td>
<td>2.6 GB</td>
</tr>
<tr>
<td>Echorouk Online</td>
<td>Algeria</td>
<td>39.800K</td>
<td>500.039K</td>
<td>500 MB</td>
</tr>
<tr>
<td>Riyadh</td>
<td>Saudi Arabia</td>
<td>265.677M</td>
<td>1.311M</td>
<td>3 GB</td>
</tr>
<tr>
<td>Saba News</td>
<td>Yemen</td>
<td>15.462K</td>
<td>228.044K</td>
<td>200 MB</td>
</tr>
<tr>
<td>Saudi Youm</td>
<td>Saudi Arabia</td>
<td>229.741M</td>
<td>1.179M</td>
<td>2.7 GB</td>
</tr>
<tr>
<td>Techreen</td>
<td>Syria</td>
<td>93.695M</td>
<td>848.156K</td>
<td>1.1 GB</td>
</tr>
<tr>
<td>Youm7</td>
<td>Egypt</td>
<td>254.353M</td>
<td>939.967K</td>
<td>3 GB</td>
</tr>
<tr>
<td>Entire Corpus</td>
<td>8 Countries</td>
<td>1.493 B</td>
<td>2.931.250 M</td>
<td>17 GB</td>
</tr>
</tbody>
</table>

### 3.1.2 Arabic WordNet

WordNet [59] is an electronic lexical database that groups English words which tend to share the same meaning in a synonym set called synset. Synsets are connected through lexical relations such as hypernyms, hyponyms and antonyms. Each synset is associated with a definition and usage examples. The English WordNet is a widely used resource for a variety of NLP tasks.
Because of the important role English WordNet has played, languages such as Arabic started their own WordNet projects. The Arabic WordNet (AWN) [26, 60] follows the design and content of the English WordNet. This resource does not have a high coverage of used words as the English WordNet, and also lacks synsets’ definitions and usage examples. This resource counts at present 23,481 words organized into 11,269 synsets [56]. In this work, we construct glosses for each synset based on synonyms and hypernyms associated with it.

3.2 Stemming the Corpus and Word Vector Initialization

Arabic words are founded on a large number of bases that usually consist of three letters. These bases determine the meanings of words, allowing words to be grouped by meaning according to the bases they share. Words are given context such as noun, verb, singular, or plural by expanding or altering bases according to established patterns. New words can therefore be added to the Arabic language according to these conventions [60].

Because of the latter property of Arabic, our goal is to analyze performance of the word semantic similarity task taking into account Arabic stemming algorithms, using word embeddings.

3.2.1 Stemming Algorithms

Stemming is the automatic process of reducing variants of morphological terms to their stem or root. We evaluate multiple Arabic stemming algorithms based on popularity of usage [61] and availability, so that we have access to their implementation.
The first Arabic stemming algorithm we test is the Khoja stemmer\(^2\) [62], one of the most successful and widely used stemmers, as reported by Taghva [63]. This stemmer aims to find the root of an Arabic word by first stripping prefixes and suffixes, and then tries to find the root, by consulting a root word dictionary.

Another root extraction stemmer similar to Khoja, but one that does not employ a root dictionary is the ISRI Arabic stemmer, developed by Information Science Research Institute [63]. This stemmer aims to find the root of an Arabic word by matching the targeted word with a predefined set of Arabic patterns.

Finally, another successful and well-known Arabic stemmer is The Light10 stemmer\(^3\) developed by Larkey et al. [64]. This stemmer tries to find the root of an Arabic word by removing affixes from a predefined affix set. The Light10 stemmer has become widely used in Arabic information retrieval.

We use the candidate stemmers on the entire corpus. Technically, we tokenize each word in the corpus passing it to each stemmer and returning the output to separate files for each stemmer.

### 3.2.2 Word Vector Initialization

After the stemming stage, we compute the word vectors for each stemmed corpus as well as with no stemming (No_Stem). For this purpose, we use two recent word embedding techniques, Word2Vec [2] implementing both Continuous Bag of Words (CBOW) and Skip-gram (SG) models, and GloVe [14] based on both global co-occurrence matrix factorization and a local context window.

\(^2\)Java implementation of Khoja is at http://zeus.cs.pacificu.edu/shereen/research.htm

\(^3\)Arabic Light10 stemmer implementation is at https://github.com/MiladAlshomary/light10stemmer
The training objective of CBOW is to produce word vectors that are efficient at predicting a target word given its context. Formally, given a corpus $T$ with a sequence of words $w_1, w_2, \ldots, w_T$, the objective of CBOW is to maximize the average of the log probability of observing the target word $w_t$ provided its context words $w_{t+j}$ restricted by a fixed window size of $n$ training context words.

$$\frac{1}{T} \sum_{t=1}^{T} \left( \sum_{-n \leq j \leq n, j \neq 0} \log p(w_t|w_{t+j}) \right)$$  \hspace{1cm} (1)$$

The SG model does the inverse of CBOW. In SG, we predict the context words given a target word. Therefore, a slight change in the training objective occurs using the same notations as CBOW:

$$\frac{1}{T} \sum_{t=1}^{T} \left( \sum_{-n \leq j \leq n, j \neq 0} \log p(w_{t+j}|w_t) \right).$$  \hspace{1cm} (2)$$

The other word embedding technique we use is GloVe, a model that combines both global matrix factorization and a local context window resulting in a global log-bilinear regression model with a weighted least-squares cost function $J$ [14].

$$J = \sum_{i,j=1}^{V} f(X_{ij}) \left( w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij} \right)^2$$  \hspace{1cm} (3)$$
Here $w_i$ and $b_i$ represent word vector and bias, respectively, of the main word. $\tilde{w}_j$ and $\tilde{b}_j$ represent vector and bias of the context word, respectively. $X_{ij}$ is a co-occurrence matrix representing the number of times that word $w_i$ appears in the context of word $w_j$. A weighting function $f(X_{ij})$ assigns a lower weight for pairs that appear either extremely or rarely in the same context.

We initialize the word vectors using the previous word embedding models for each Arabic stemmed corpus. For both CBOW and SG, we set the parameters as 5 for word window size, 400 for vector dimensions, and allow 5 negative samples. GloVe’s parameters are 10 for words window size, 400 for vector dimension, and 100 for $x_{max}$. These vectors are evaluated against a word semantic similarity task.

### 3.3 Evaluation of Word Similarity

We evaluate our word embeddings on the task of word semantic similarity as shown in Table 3.2. Our goal is to analyze both word embedding methods and Arabic stemmers that exhibit comparable performance in capturing the semantic relationships between words. We hypothesize that high quality word representation will lead to better word sense disambiguation.

For evaluation, we use the word semantic similarity dataset constructed by Faaza et al. [65], the only available dataset. This dataset contains of 70 Arabic word pairs.

The results shown in Table 3.2 show Spearman rank order correlation between human similarity judgment and similarity obtained from the model. In order to obtain the result for each stemmer, we must stem the word pairs in the dataset using the stemmer. Thus,
Table 3.2: Spearman rank order correlation between human relatedness judgment and model similarity. Bold and underlined scores show the best results overall, and underlined scores show comparable results.

<table>
<thead>
<tr>
<th>Model</th>
<th>No_Stem</th>
<th>Light10</th>
<th>ISRI</th>
<th>Khoja</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBOW</td>
<td>72.26</td>
<td>70.2</td>
<td>51.13</td>
<td>40.58</td>
</tr>
<tr>
<td>SG</td>
<td>75.48</td>
<td>75.62</td>
<td>59.44</td>
<td>52.56</td>
</tr>
<tr>
<td>GloVe</td>
<td>73.58</td>
<td><strong>78.11</strong></td>
<td>65.49</td>
<td>58.41</td>
</tr>
</tbody>
</table>

we construct a distinct dataset for each stemmer. Then, we compute the cosine similarity between word vectors corresponding to this stemmer. We calculate Sperman rank correlation between the obtained cosine similarity that corresponds to the stemmer and the distinct dataset for the same stemmer. For the No_Stem case, we compute Spearman correlation between the original dataset and word vectors produced from the original corpus.

The similarity results show that Light10 with GloVe produces the best score overall. Light10 and No_Stem yield reasonable scores across all word embedding models compared to ISRI and Khoja. We expected the result to be highly diverse using the stemmed vectors. However, Light10 shows slightly better performance in capturing semantic relationships. We attribute this to the high number of word frequencies in our corpus as shown in Table 1 (# of Tokens compared to # of Unique Tokens). Thus words are well parameterized even though words are not stemmed.

We believe that the reason for the low scores after using ISRI and Khoja is because of the nature of these algorithms. The extensive computations these algorithms perform to obtain the roots of the words might reduce words that have completely different meanings to the same root word that was predominant during training. On the other hand, Light10 is a light stemmer that does not aim directly to root the target words, but tries to remove
affixes. We think because of the nature of its stemming process which stems variants of a word to a common word, leads to a better score.

3.4 Proposed Algorithms for Sense Embeddings

In this section, we present the proposed algorithms for initializing sense specific embeddings. Our hypothesis is that multiple meanings of a word can be found by considering both the main word and its neighbors obtained by considering synonyms, and hypernyms words. The AWN is used for identifying words senses (synsets) as well as for obtaining word senses’ glosses.

3.4.1 RETROFIT Algorithm

In this algorithm, we follow the RETROFIT algorithm developed by Jauhar et al. [19]. We outline the important parts of the algorithm below. The AWN can be considered a directed graph, which we call the ontology $\Omega = \{ S_{\Omega}, E_{\Omega} \}$, where $S_{\Omega}$ represents semantic objects or synsets, and $E_{\Omega}$ represents edges or semantic relations among the synsets. Given a vocabulary of word embeddings $V$ and ontology $\Omega$, the objective is to infer a sense-specific embedding $\hat{V}$ that is maximally consistent with both $V$ and $\Omega$ by finding the minimum of the following objective function:

$$D(\hat{V}) = \sum_{ij} \alpha \| \vec{w}_i^j - \hat{w}_i^j \|^2 + \sum_{ij, i'j'} \beta_r \| \hat{w}_i^j - \hat{w}_{i'j'} \|^2$$  \hspace{1cm} (4)
where, \( \vec{w}_i \) is the vector of the word type \( i \), \( \vec{w}_{ij} \) is the sense-specific vector for the word \( i \) corresponding to the synset \( j \), and \( \vec{w}_{ij} \) denotes the sense-specific vector of a neighbor. \( \alpha \) and \( \beta_r \) are hyper-parameters that score the importance of the neighbor words for a sense. The objective function aims to narrow the divergence between sense-specific vectors of neighbor nodes as well the separate sense-agnostic vector from its sense-specific vectors. Each specific-sense embedding is iteratively updated until convergence as follows:

\[
\vec{w}_{ij} = \frac{\alpha \vec{w}_i + \sum_{i'j' \in N_{ij}} \beta_r \vec{w}_{ij'}}{\alpha + \sum_{i'j' \in N_{ij}} \beta_r}.
\] (5)

\( N_{ij} \) denotes the set of neighbors of \( ij \). The original RETROFIT paper used synonyms, hypernyms, and hyponyms as a synset’s neighbors assigning \( \beta_r = 1.0, 0.5 and 0.5 \), respectively. In this work, we consider synonym and hypernym semantic relations assigning \( \beta_r = 1.0 \) and 0.5, respectively.

### 3.4.2 ArabSenEmb Algorithm

Our second algorithm, the ArabSenEmb is similar to the RETROFIT in that we consider the same semantic relations involving synonyms and hypernyms. However, it requires less computation. Given the same notations, we produce sense-specific embeddings as follows:

\[
\vec{w}_{ij} = \frac{\vec{w}_i + \sum_{i'j' \in N_{ij}} \vec{w}_{ij'}}{N + 1}.
\] (6)
Our sense-specific vector $\vec{w}_{ij}$ is computed by adding up the sense-agnostic vector $\vec{w}_i$ and vectors of the neighbors $\vec{w}_{ij'}$ divided by the number of neighbors and plus one for the sense-agnostic word type $N+1$. We propose the ArabSenEmb to investigate whether the extensive computations performed by RETROFIT produce better sense-specific embeddings. We note that for both algorithms we discard synsets containing compound words since some of them have completely different meanings from the sense in which we are interested. The implementation of this algorithm is shown in Appendix 0.1.

3.5 WSD System and Test Data

After producing the sense embeddings, we need to validate and compute the efficiency of our algorithms. To do so, we present a WSD system that by its nature is a downstream application. We also develop a test datasets that is based onAWN.

3.5.1 WSD System

Our WSD system attempts to compute the appropriate meaning of an ambiguous word given its context. Given a context containing a sequence of words $w_1,w_2,w_3,\ldots,w_d,\ldots,w_n$, where $w_d$ is an ambiguous word, the system lists all synsets representing this ambiguous word along with their similarity to the context. Finally, the system chooses the synset that has the maximum similarity among synsets. More formally,

$$\arg\max \left( \text{sim}(\text{context}, s_{ij}) \right) = \text{CosSim} \left( \frac{\sum_{r=1}^{n} \vec{\text{context}_r}}{n} \right) \left( \vec{s}_{ij} \right) \quad (7)$$
where $\text{context}$ denotes the vector of the word that appears in the context, and $n$ represents the number of tokens in the context, $s_{ij}$ represents the vector of synset $i$ for the word $j$. In other words, we compute the average vector of the context words around the ambiguous word. Then, the cosine similarity measure is used to compute the similarity between the context vector and the ambiguous word’s synset vectors. Figure 3.1 shows the output of the system presenting the similarity for each synset to the associated context.

```
Sentence ID: 24
TrueSysnset: %EaAm_n1AR
%EaAm_a1AR = 0.446293048225
%EaAm_a1AR = 0.401675295721
%TafaA_v1AR = 0.449631459154
%EaAm_n1AR = 0.595166309111
%EaAm_a5AR = 0.575247262483
%sabaHa_v1AR = 0.494641392052
(Correct)
```

Figure 3.1: Output of our WSD system showing each synset with its cosine similarity to the context vector.

### 3.5.2 WSD Test Data

In order to evaluate our work, we develop a WSD test dataset based on AWN version 2.0. The construction of our test dataset follows the approach used by lexical tasks in English. Such a lexical task is comprised of disambiguating a set of predefined words in context. We choose 25 ambiguous words which have a number of synsets, ranging from 15 to 4 associated with them.
We have a total of 457 samples in our test dataset. We manually find samples that represent a word’s synsets. As mentioned earlier, the AWN lacks a synset’s definition and examples, which makes it difficult to judge the appropriateness of synsets to define the meaning, and we restrict our search to the synsets that we are confident about. We use two online documented Arabic corpora as resource, namely the Watan and Khaleej corpora [66], and some samples have been extracted from the web for synsets that are not found in our used corpus.

### 3.6 Result

Table 3.4 shows our result for the WSD task using a measure which divides the number of correctly detected synsets by the total number of samples. We evaluate two cases: no stemming and Light10 vectors.

<table>
<thead>
<tr>
<th>Model</th>
<th>No Stem</th>
<th>Light10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RETROFIT</td>
<td>ArabSenEmb</td>
</tr>
<tr>
<td>SG</td>
<td>83.22%</td>
<td>78.19%</td>
</tr>
<tr>
<td>GloVe</td>
<td>73.44%</td>
<td>71.81%</td>
</tr>
</tbody>
</table>

The Skip-Gram method using the RETROFIT approach with no stemming vectors shows the best result overall. The comparison of the results between RETROFIT and the ArabSenEmb shows the need for a significant number of extra computations required by
RETROFIT. The sense-specific vector must be determined not by considering the word type of neighbors, but with the sense-specific vectors of the neighbors.

To use Light10 vectors, in our second approach we build our ontology using the Light10 stemmer. We extract each synset in AWN and then stem each neighbor of this synset. After that, we produce sense-specific vectors by extracting each neighbors’ Light10 vectors produced in our previous experiments. Even though Light10 gives either the same or slightly better results in the semantic similarity task, in the WSD task it gets a low score compared to unstemmed vectors. We believe that the reason may be that words that have different meanings may be stemmed to a common word despite that fact that Light10 is a light stemmer. RETROFIT is not applicable for the stemming process because of synset dependency issues. Stemming every entry in the ontology leads to creating new synsets that are not supposed to correspond to senses of a word.

3.7 Discussion and Conclusion

In this work, we contribute to Arabic NLP field in several dimensions. We are providing important resources in terms of a pre-processed and ready-to-use large corpus and an Arabic WSD test dataset. We also analyze two recent and important word embedding methods considering a word semantic similarity task, taking into account multiple well-known stemming algorithms. Finally, we use multiple approaches for the problem of WSD, evaluating them in a way that can be considered a downstream application.

In our work, we attempt to overcome shortcomings of previous work. Because of the fact that local context features are not enough to disambiguate words, we obtain global
context from a large corpus producing global word meanings. We restrict our work to use the AWN, the only available sense inventory, making the sense embeddings we produce a considerably useful source for real world Arabic applications.

Using recent developments in Vector Space Models (VSMs) for the problem of WSD is a good choice, and this study shows outstanding results using VSMs for another language. We believe that overcoming some of the shortcomings of AWN will lead to even better result. We notice that most of the synsets that are difficult for the WSD system to detect are the ones that do not have enough neighbors. We recommend the use of other methods to enlarge the set of gloss words for each synset rather than synonyms and hypernyms. Other methods that involve using graph measures such as Wup [46], the LCH [48], and the Li measure [50] may be used to enrich less detailed synsets.
CHAPTER IV

ATTENTION-BASED SEQUENCE LEARNING MODEL FOR

ARABIC DIACRITIC RESTORATION

Arabic Diacritic Restoration is one of the fundamental tasks in Arabic since many different Arabic NLP tasks benefit from it. In this work, we investigate the use of Recurrent Neural Networks (RNN) involving the two most popular RNN architectures: Long-Short Term Memory (LSTM) and Gated-Recurrent Units (GRU). We also examine the use of attention mechanism for the purpose of increasing the accuracy of our basic RNN models. As well as using attention-based structure involving in google transformer. We perform variety of experiments that use from one layer to multiple stacked layers as well as using bidirectional RNNs for both RNN architectures.

4.1 Introduction

Arabic is a widely used language with population of half a billion native speakers. The formal structure of Arabic orthography comprises of diacritics, which appear either above or below the alphabetic characters to alter their pronunciation or sense. The Mod-
ern Standard Arabic (MSA) is written without diacritics, which causes ambiguity in understanding the associated meanings for both humans and language modelling tasks. For example, the Arabic word (عِلم) varies greatly in meaning based on the types and positions of the usually unwritten diacritics (عِلم, science) (عَلم, Flag) (عُلم, understood) and (عَلم, teach). Absence of written diacritics in the bulk of Arabic documents causes ambiguity for many language processing tasks, including speech recognition, POS tagging, word sense disambiguation, and machine translation [67,68]. Accordingly, automatically restoring missing diacritics is a necessary preprocessing step for Arabic NLP applications.

Automatic restoration of Arabic diacritics in non-diacriticized text has been the focus of Arabic NLP community and has received significant attention. Several methods such as rule-based approaches, and finite-state machines and statistical approaches using Hidden Markov models and maximum entropy models, have been used for Arabic diacritic restoration. These methods are based on feature engineering and are expensive to develop. Moreover, these methods need expensive resources, developed specifically for the target language. We believe that diacritic restoration is better carried out in a way that generalizes to other languages such as Hebrew and Syriac. Our goal is to address this issue in a way that can be easily adapted to other languages.

Recent attempts have treated Arabic diacritic restoration as a sequence modelling problem using Bidirectional Long Short-Term Memory Networks (BiRNN) to overcome prior shortcomings [69,70]. These are classification models which receive a sequence of non-diacritic using Arabic characters and output a diacritic label. These models have produced promising results, achieving state-of-the-art performance. Because of the success of sequence modelling in general, we are inspired to explore solutions to the problem lever-
aging recent developments in attention mechanisms. A BiRNN learns to perform a task by computing a concatenated representation at a current timestep that takes into account previous as well as future values, separately. However, certain inputs, either in the forward or backward direction, might be more discriminative in predicting the missing diacritic at the current timestep. An attention mechanism assigns a score to each input element providing a degree of importance for these elements. In recent years, this method has been widely applied to achieve significant improvements for many NLP tasks such as text summarization, sentiment classification, and question answering [71].

In this work, we attempt to restore Arabic diacritics by training it as a sequence to-sequence problem that benefits from the improvements that have been obtained in such situations by using attention mechanisms. Instead of predicting the current target output at a given timestep solely based on the current BiRNN hidden state at the given timestep, we use an attention mechanism on top of the BiRNN, to capture correlation information in the whole input sequence to generate a representation of the given context, in terms of a context vector. Thus, we investigate the effectiveness of the use of an attentive BiRNN for Arabic diacritic restoration. In other words, we augment previous work which used non-attentive RNNs to work with attention mechanism. We also investigate the use of pretrained models we obtain by training google transformer. We transfer the knowledge we obtain from the transformer to be used in the BiRNN architecture. We design, analyze and experiment with multiple RNN architectures using LSTM and GRU, ranging from multiple stacked BiRNNs to attention-based BiRNN. We conduct our experiments using the benchmark dataset, Linguistic Data Consortium’s Arabic Treebank of discretized news stories (LDC’s ATB3), Part3, v3.2 [72]
4.2 Characteristics of Arabic

Arabic is unlike English in that it is written from right to left, using an alphabet of 28 letters, which are all consonants. Vowels are expressed as diacritics in Arabic, but are not explicitly written in MSA. Consequently, Arabic readers must use their linguistic knowledge to correctly unearth missing diacritics based on the context, so as to describe the proper meaning to words. The Arabic orthography system comprises a list of diacritics, which can be categorized as short vowels, case endings, gemination, and sukun.

- Three short vowels namely fatha, damma, and kasra transliterated as \( (a, u, i) \), respectively. Fatha is written as a small slash above the letter, whereas kasra is written below a letter, and damma is a small curl written above a letter. They can be located either in the beginning, middle or at the end of a word.

- Case endings (nunation) transliterated as (F, K, N) pronounced as \( an, in, \) and \( un \), respectively. They are added at the end of either a noun or an adjective for grammatical case: accusative, genitive and nominative nunation, respectively.

- Gemination, aka shaddah, transliterated as \( (\sim) \). Gemination indicates doubling or extra length of the preceding letter.

- Sukon is a silence marker which indicates an absence of a vowel, transliterated as \( (o) \).

   It should be noted that short vowels and case endings can combine with gemination, producing up to six combined vowels.

\(^1\)We use Buckwalter transliteration scheme [http://www.qamus.org/transliteration.htm](http://www.qamus.org/transliteration.htm)
4.3 Related Work

Through the years, there have been a decent amount of work on automatic diacritization. The approaches can be classified into three categories: approaches based only on statistical processing; hybrid approaches using both statistical and morphological analysis; and hybrid approaches consisting of morphological analysis, syntactic rules as well as statistical processing [73].

Gal [74] used Hidden Markov Models to statistically restore diacritics without relying on any specific language knowledge, making it generalizable to both Arabic and Hebrew. Vergyri and Kirchhof [75] used acoustic signals in combination with morphological and contextual constraints. Nelken and Shieber [76] proposed a system implementing finite state transducers combined with a word and character language model and a simple morphological model. Zitouni et al. [77] formulated diacritic restoration as a classification problem using a maximum entropy classifier using lexical, part-of-speech and segment-based features.

Habash and Rambow [78] used the BAMA morphological analyzer [79] and individual taggers to analyze a word to return a set of possible word analysis. The tables produced by individual taggers were used to choose the correct diacritic from among the possible outputs. Shalaan et al. [80] proposed a hybrid approach involving a lexicon, a bigram model and the SVM classifier. Alghamdi et al. [81] presented a statistical approach that was independent of language knowledge such as morphology and syntax. This was based on character quad-grams.
Hifny [82] used a statistical approach that was solely based on a diacritized Arabic corpus, and used dynamic programming to assign scores to possible sequences of diacritized words using a statistical n-gram language model. Bebah et al. [83] used a morphological analyzer called AlKhalil Morpho that used two Hidden Markov Models. Said et al. [84] introduced a hybrid approach consisting of auto correction, morphological analysis, and a part of speech tagger.

Pasha et al. [85] presented the MADAMIRA morphological analyzer combining some of the best aspects of two commonly used systems for Arabic processing, MADA [86] and AMIRA [87], including diacritization. Shahrour et al. [88] presented a hybrid approach integrating manual syntactic rules with morphological tagging for a machine learning model. Another hybrid approach which consisted of three levels of granularities, including word and character levels and morphological analysis, used a Viterbi decoder with backoff for stemming in addition to morphological patterns, and an SVM classifier for case endings [67, 89]. Chennoufi and Mazroui [90] presented a hybrid approach containing morphological and syntactic analysis using Alkhalil Morpho Sys, hidden Markov models, and the Viterbi algorithm.

Recently, Arabic diacritic restoration has been represented as a sequence-to-sequence problem, and solved using recurrent neural networks (RNN). Such approaches have been proposed by Abandah et al. [69] and Belinkov and Glass [70]. These approaches are language independent, which do not require any prior language knowledge such as syntax and morphology. In both works, the RNN models were trained with a sequence of non-diacriticized Arabic characters, each character mapped to a diacritic label.
4.4 Approach

We treat our problem as a classification task following previous approaches [69, 70, 91]. We conduct our experiments using BiRNNs, which already have shown good results for many NLP tasks. We are given an input sequence of Arabic non-diacriticized characters \( c = (c_1, c_2, \ldots, c_n) \), each of which is assigned a diacritic label \( l = (y_1, y_2, \ldots, y_n) \). Each character \( c \) has either 0, 1 or 2 diacritic labels \( y \). Each \( c \) is represented as a character vector denoted as \( x_c \) learned during training. The basic RNN architecture maps each embedding \( x_c \) to a hidden state \( h_c \), and then uses \( h_c \) to compute a probability distribution over all labels \( y \). In this work, we use two RNN structures, namely Long Short-Term Memory (LSTM) [92] and Gated Recurrent Unit (GRU) [93]. In the following subsections, LSTM and GRU are briefly described.

4.4.1 Long Short-Term Memory (LSTM)

The basic RNN structure has difficulty in learning long-term dependencies, but works fine when dealing with short context. When solving a task that may be influenced by longer context, the basic RNN suffers from issues such as vanishing gradient and thus is unable to learn by backpropagation. As a result, for long sequences, the basic RNN is not dependable. Thus, certain structures have been developed on top of RNNs to address this flaw. The LSTM is an augmentation of the RNN to tackle this drawback, so that it is able to learn dependencies over longer periods of time. An LSTM computes a hidden representation \( h_c \) for each input sequence \( x \) through three interactive gates: input, forget, and output gates.
The representation of an input sequence $x$ at each time step $x_t$ is computed through the following repeated process:

$$i_t = \sigma(U_{xi}x_t + U_{hi}h_{t-1} + U_{ci}c_{t-1} + b_i)$$  
(1)

$$f_t = \sigma(U_{xf}x_t + U_{hf}h_{t-1} + U_{cf}c_{t-1} + b_f)$$  
(2)

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(U_{xc}x_t + U_{hc}h_{t-1} + bc)$$  
(3)

$$o_t = \sigma(U_{xo}x_t + U_{ho}h_{t-1} + U_{co}c_t + bo)$$  
(4)

where $\sigma$ is the logistic sigmoid function, and $i, f, c, h$ and $o$ are the input gate, forget gate, cell activation, hidden state vector and output gate respectively. The weight matrix $U$ and bias vector $b$ are learned during training.

### 4.4.2 Gated-Recurrence Units (GRU)

A GRU is similar to an LSTM, but requires less computation and fewer parameters. A typical GRU structure contains two gates: a reset gate and update gate. Both gates decide what information to preserve or ignore to learn long-term dependencies. The mechanism of the GRU’s cell can be described in terms of the following formulas:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$  
(5)

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$  
(6)

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \sigma_h(W_h x_t + U_h (r_t \odot h_{t-1}) + b_h)$$  
(7)

where $z_t, r_t$, and $h_t$ are the update gate, forget gate, and hidden state vector, respectively. $W, U$ and $b$ are parameter matrices and the bias vector, respectively.
4.5 Attentive BiRNN Diacritic Restoration

Recent research suggests incorporating contextual information with LSTMs for classification tasks [28]. Our approaches are based on attention mechanisms that are able to capture deeper contextual information, and which happen to be on top of our BiRNN output. Our approach obtains contextual information from the whole input sequence. This attention structure is based on the work of Zhou et al. [94], which has shown success in a number of NLP tasks [95, 96].

We are given a sequence of hidden states \((h_1, h_2, ..., h_t)\) generated by our BiRNN, represented as a matrix \(H \in \mathbb{R}^{d \times T}\), where \(d\) denotes the dimension of the hidden states, and \(T\) denotes the input sequence length. The context vector \(cv\) is computed as the weighted sum of the produced hidden states, obtained by processing the following formulas:

\[
m = tanh(H)
\]
\[
p = softmax(w^T m)
\]
\[
cv = Hp^T
\]

where \(w\) is a parameter vector, and the dimensions of \(w, p, cv\) are \(d, T,\) and \(d,\) respectively.

We then obtain the final context vector as follow:

\[
cv^* = tanh(cv).
\]

After computing the context vector \(cv^*\), we concatenate it with the last BiRNN layer for each time step, as shown in Fig 4.1.
4.6 Sequence to Sequence Using Transformer

In this section, we use Google Transformer [33] to obtain the knowledge of language using a pre-trained model. Since our work is a character level, we must obtain the knowledge of Arabic characters. To do so, we use a portion of the big corpus we used in Chapter III. We train our Transformer on a dataset that was divided into training, dev and evaluation subsets of 100M, 10M, and 5M characters, respectively. We prepare our data in the way to have a sequence of input and output of characters as shown in Fig 4.2.
We use the default Transformer parameters as stated in the original work [33]. That is, 6 encoder stacks, where each encoder has a self attention layer, and is a simple, position-wise fully connected feed-forward network. Each Transformer layer has 512 embedding dimension, and the inner layer has dimensionality of 2048. We use Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$. We also use Residual Dropout to each of the output sublayer.

After obtaining the pre-trained embeddings of Arabic characters, we use the character embeddings and train them in our diacriticized text using the diacriticized corpus, LDC’s ATB3. We obtain pre-trained embeddings for both inputs and outputs that are Arabic characters and diacritics.

Our final model is to use a BiRNN approach using the pre-trained embeddings for both the inputs and outputs. We use our best BiRNN model using Transformer pre-trained embeddings, this is Attentive-GRU model as shown in Fig 4.3.

Figure 4.3: Arabic Diacritic Restoration Using Transformer Architecture.
4.7 Experiments and Results

The following section provides the details of the data we use and experiments we perform.

4.7.1 Data

We prepare our data as a sequence of observations that include a sequence of non-diacritic characters as inputs. We separate characters from their corresponding diacritic marks, resulting in labelling each character with either 0, 1, or 2 diacritic marks. We split our corpus into train, dev, and test sets following the previous works [69, 70, 91]. Table 4.1 shows statistics of the text data.

Table 4.1: Statistics of the corpus used.

<table>
<thead>
<tr>
<th>Data Part</th>
<th># of sequences</th>
<th># of words</th>
<th># of characters</th>
<th># of diacritic marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>8564K</td>
<td>196K</td>
<td>1.154M</td>
<td>641K</td>
</tr>
<tr>
<td>Validation</td>
<td>2141K</td>
<td>50K</td>
<td>299K</td>
<td>162K</td>
</tr>
<tr>
<td>Testing</td>
<td>1887K</td>
<td>44K</td>
<td>259K</td>
<td>144K</td>
</tr>
</tbody>
</table>

4.7.2 Implementation Details

In all of our experiments, we use 256 memory cells, resulting in 512 cells for each BiRNN layer. Dropout regularization on top of the embedding layer is used to improve generalization [97]. We initialize weights with a random uniform approach. The Stochastic Gradient Descent (SGD) optimizer is used starting with a 0.1 learning rate and a decay of $10^{-3}$ and a momentum of 0.9. The learning algorithm calculates the gradients using
backpropagation through time (BPTT) [98].

4.7.3 Evaluation Metrics

We evaluate our models based on the measurement of Diacritic Error Rate (DER) [69, 70, 99, 100]. It is the proportion of the correct predicted labels to the number of wrong label predictions. There are two variants of DER: all-diacritic, and end-diacritic. All-diacritic DER considers all diacritics in a word including word ending, and end-diacritic DER omits all diacritics except for the diacritic at word ending.

4.7.4 Results

Table 4.2 shows the values for all-diacritic and the end-diacritic DER metrics on the corpus used. We evaluate our attention model and pre-trained model against the non-attentive models. Our objective is to validate the use of our approaches compared to the non-attentive BiRNN sequence labelling approach [69, 70]. We found that using a decaying learning rate is the best choice to produce the best outcome compared to a fixed learning rate.

The use of Attentive-GRU using pre-trained embeddings produces the best result among the other models as well as the use of 3-layers. The GRU outperformed the LSTM in most of our experiments. The GRU model outperformed LSTM by 0.78% in the All DER values and by 0.96% in the End DER values when using 3-layers; likewise with the Attentive-GRU, achieving a better performance by 0.49% compared to Attentive-LSTM. Using pre-trained embeddings enhanced the performance by .12% as far as ADR-ALL
Table 4.2: The DER of all diacritic (All) and at word ending (End) for LSTM and GRU with attention (Attentive) and without attention mechanism. Bold indicates the best result among the others in the same layer, bold and underscore indicate the best result overall.

<table>
<thead>
<tr>
<th>BiRNN Layers</th>
<th>LSTM</th>
<th>GRU</th>
<th>Attentive-LSTM</th>
<th>Attentive-GRU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All</td>
<td>End</td>
<td>All</td>
<td>End</td>
</tr>
<tr>
<td>1-Layer</td>
<td>5.02</td>
<td>10.11</td>
<td>4.82</td>
<td>9.62</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2-Layers</td>
<td>4.11</td>
<td>9.97</td>
<td>3.11</td>
<td>9.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3-Layers</td>
<td>3.60</td>
<td>9.56</td>
<td>2.82</td>
<td>8.60</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transformer-3layers</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.3: The best result reported by related work using DER-All.

<table>
<thead>
<tr>
<th>LSTM Layers</th>
<th>DER-ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Belinkov &amp; Glass</td>
<td>4.85</td>
</tr>
<tr>
<td>Abandah et al.</td>
<td>2.72</td>
</tr>
</tbody>
</table>

compared to best performance reported without using pre-trained model that is involved in Attentive-GRU. Noticeably, in all models, increasing the layer sizes was always helpful, make deeper models more accurate. We tried to enhance the results by adding one more stacked layer, but it did not improve the performance.

In comparison to the two related works that used the approach of sequence modelling shown in table 4.3, our attention mechanism approach improved the accuracy by 0.19% compared to Abandah et al. [69] and by 2.32% compared to Belinkov and Glass [70]. Also, using pre-trained embeddings improved the performance by 0.31% compared to Abandah et al. and by 2.44% compared to Belinkov and Glass. We should also mention that we obtained different results using the non-attentive approach, which is a similar approach to the compared works. We might refer this to different factors such as number of epochs, the
4.8 Analysis and Conclusion

Fig 4.4 shows the heat map of our Attentive-GRU 3-layers model. Most of the diacritic marks that contribute most to the errors are the case endings. The case endings involved in F, K, N and the combination with the gemination ~N and ~K. These marks appear at the end of the words indicating grammatical cases. Arguably, the confusion with these marks is hardly surprising since the appearance of them permanently rely on the corresponding part of speech, that is either accusative, genitive or nominative.

The noticeable difference between the performance of All-DER and End-DER can be viewed as a significant sign of the importance of involving not only learning diacritic pattern by the RNN but also involving external features such as part of speech tagging. Multitask learning has been applied with success in a varity of NLP tasks, word sense disambiguation [42], relation classification [101], and sentiment analyses [102].

To sum up, in this work, we examine the use of a successful attention mechanism approach for Arabic diacritic restoration alongside with non-attentive approaches. As well as utilizing pre-trained embeddings using google-transformer architecture. The use of BiRNNs with attention mechanism showed the ability to better learn and identify existing data patterns as well as using pre-trained embeddings. Interestingly, finding Arabic diacritic patterns does not require as many parameters as in LSTMs but fewer parameters as in GRU. We might attribute this finding to the fact that our approach is a character-based
approach, which is limited to the number of Arabic characters as inputs. Otherwise, in case a word-based approach were used, the model would require more parameters to learn existing patterns thus LSTM might have performed better than GRU. For the future work, we intend to incorporate external resources involved in the part of speech tagging POS, which we believe it will provide with a better and deep pattern capture.

Figure 4.4: The confusion matrix of our 3-layers Attentive-GRU
CHAPTER V

AUTOMATIC DIACRITIC RESTORATION FOR WORD SENSE DISAMBIGUATION

Word sense disambiguation is a critical and important problem in NLP, since it is an intermediate task which helps improve many end tasks such as machine translation, and text classification. Previously, we tackled the Arabic WSD problem using global context with different learning representations methods. We had seen how these methods were used for WSD and captured sufficient information, which made it a good choice for an unsupervised WSD approach.

A way to study WSD from a different point of view is to think carefully about how Arabic is structured with vowels which play a critical role in understanding, and pronouncing Arabic words or text. Arabic vowels should be considered when attempting to realize the sense or meaning of a word in a given text. Diacritizing Arabic texts would naturally reduce the confusion in understanding texts; thus, we believe that another perspective should be added towards our ultimate goal. Previously, in Chapter IV, we presented our diacritic restoration models that were able to restore diacritics with noticeable success.
In order to perceive how diacriticizing helps mitigate the complexity of words, Table 5.1 shows the possible diacritic marks alongside their meaning for the word ()

<table>
<thead>
<tr>
<th>Arabic Diacritic Form</th>
<th>Transliteration</th>
<th>English Meaning</th>
<th>Part of Speech</th>
</tr>
</thead>
<tbody>
<tr>
<td>ﯾٌلّم</td>
<td>Eilom</td>
<td>Science</td>
<td>Noun</td>
</tr>
<tr>
<td>ﯾٌلّم</td>
<td>Ealam</td>
<td>Flag</td>
<td>Noun</td>
</tr>
<tr>
<td>ﯾٌلّم</td>
<td>Euulim</td>
<td>Understood</td>
<td>Passive Verb</td>
</tr>
<tr>
<td>ﯾٌلّم</td>
<td>Eal~am</td>
<td>Teach</td>
<td>Verb</td>
</tr>
<tr>
<td>ﯾٌلّم</td>
<td>Euul~im</td>
<td>Taught</td>
<td>Passive Verb</td>
</tr>
</tbody>
</table>

Intuitively, languages that are written without diacritic marks like Arabic can benefit broadly from restoring fully-specified diacritic marks. Automatic restoration of diacritics can be used as a pre-processing step for many NLP task; however, not all words can be totally disambiguated. Words can still exhibit ambiguity even after full diacritic restoration; for example, the fully diacritized Arabic word ( ﯾٌلّم , Eayon) can either mean an eye or water spring.

In this chapter, we investigate the use of our diacritic restoration models discussed in the previous chapter with our WSD dataset. We aim at validating if text Diacriticizing would lead to better word disambiguation compared to only using vector representations of words and synsets.

### 5.1 Diacriticizing Arabic Dataset

Our goal in this section is to perform the Arabic diacritic models developed in the previous chapter to better understand words’ senses. In the previous chapter, we solved WSD by computationally obtaining words sense’s specific vectors, which then were mapped to
the correct senses listed in the AWN. On the other hand, the approach we intend to develop completely or partially depends on trained RNN models; thus, we do not require any external resources such as a wordnet. Therefore, this approach is a unified approach which is a language independent as long as the training models and datasets are available.

This approach tends to predict the diacritic marks given a sequence of non-diacritic characters. The model reads the whole sequence in both directions (forward and backward) and then uses the learned weights (parameters) to predict the diacritic marks for each input character as shown in Algorithm 1 and Table 5.2

**Algorithm 1 Generating Diacritic Marks**

| Input: Sequence of non diacriticized characters (S1) |
| Output: Sequence of diacritic marks (S2) |
| Procedure: Predicting diacritic marks (d) |

1: **for** S1 **do**
2:  **Read forward**
3:  **Read backward**
4:  **for each** Char in S1 **do**
5:    **Extract trained parameters**
6:    **Predict** d
7:  **end for**
8: **end for**
9: **Return** S2
10:
11: **for each** Char and mark in S1 and S2 **do**
12:  **Convert from numbers to Buckwalter transliteration.**
13:  **Convert from Buckwalter transliteration to Arabic form.**
14:  **Map each character to the corresponding diacritic marks.**
15:  **end for**
16: **return** S2
17: **end for**

Lines 1-3 in Algorithm 1 reads the whole sequence forward and backward during training. We then use the trained parameters for each Arabic character and predict the as-
Table 5.2: Diacritic Procedure of the Sentence (waHat~aY Al—na taEotaqidu AlmaSAdiru >an~a >asobAba AlHAdivi qado takuwnu $axoSiy~apF )

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Input/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strip diacritics</td>
<td>wHtY Al—n tEtqd AlmSA dr &gt;n &gt; sbAb AlHA dv qd tkwn $xSyp</td>
</tr>
<tr>
<td>Convert to numbers</td>
<td>3 2 1 1 3 0 1 4 1 8 ............. 2 2 5 1 2 3 3 1 2 3 4 2 6</td>
</tr>
<tr>
<td>Output diacritics</td>
<td>5 5 1 2 1 1 1 1 1 1 5 .............1 5 1 1 6 8 1 5 1 2 1 5</td>
</tr>
<tr>
<td>Convert to labels</td>
<td>’a’ ’a’ ’o’ ’ ’ ............. ’u’ ’ ’ ’a’ ’o’ ’ a’</td>
</tr>
<tr>
<td>Combine</td>
<td>waHat<del>aY Al—na taEotaqidu AlmaSA diru &gt;an</del>a &gt;asobAba AlHA divi qado takuwnu $axoSiy~apF</td>
</tr>
</tbody>
</table>

associated diacritic mark. After each timestep, as shown in lines 4-6, we loop back from the beginning of the sequence to input the next character and predict the diacritic mark. Each character will have 0,1 or two diacritic marks as needed by the Arabic language. Eventually, we return S2 which contains characters along with their diacritic marks.

Because of the fact that an RNN does not accept categorical inputs (only numbers), during developing the RNN diacritic models, we converted each Arabic character to the corresponding Buckwalter transliteration scheme, then we mapped each of which to a unique integer. As a result, during the generation of the prediction, we must transform the output to the Arabic alphabet in order to understand it as shown in line 11-17.

### 5.2 Experiments and Evaluations

We use the WSD dataset we developed in Chapter III for our validation of Arabic diacritic restoration to improve disambiguation of words. We intend not only to use the best diacritic restoration model, but we also will explore all models obtained previously, including, multiple BiRNN stacks, LSTM, GRU, Attentive-LSTM, attentive-GRU,
and transformer-GRU. We follow the process we described in the previous section to automatically restore diacritics. The input is a sequence of characters extracted from the WSD dataset, and the outputs is a sequence of the characters with either no assigned diacritic, or one or two diacritics. Table 5.3 shows one example sentence in our evaluational dataset, which was processed by each diacritic restoration model.

In order to perfectly match the diacriticized candidate ambiguous words, we need to omit the last diacritic mark associated with the last character in a word since in Arabic, the last diacritic mark indicates the part of speech. As mentioned earlier, full restoration of diacritics reduces the level of ambiguity in words, but does not fully disambiguate words. To tackle this, when there are different word senses spelled with the same diacriticized character, we simply switch to use our approach presented previously in chapter III; this is performed by taking the context vectors of the non-diacritized version of the context words and comparing it with all sense vectors.

In the evaluation, we follow the same metric we used in Chapter III, by dividing the correctly detected senses against those that are wrong. Table 5.4 shows the WSD results taking into account all diacritic models.
Table 5.3: All models, outputs of the diacriticized sentence (faAloEamalu Alr~iyaADiy~u SaEobN wa$saA}ikN wayaHotaAju AIY jaladN waSaborN). Bold characters/diacritic labels indicate the output mistakes.

<table>
<thead>
<tr>
<th>Model</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>LSTM 1-Layer</td>
<td>faLoloEa/mala AlryaADiy~u SaEobK wa$saA}ikN</td>
</tr>
<tr>
<td></td>
<td>wayaHotaAju AIY jaludN waSaborF</td>
</tr>
<tr>
<td>LSTM 2-layers</td>
<td>faAloEamalu Alr~iyaADiy SoEobF wa$saA}ikN</td>
</tr>
<tr>
<td></td>
<td>wayaHotAju AIY jalad waSaborN</td>
</tr>
<tr>
<td>LSTM 3-layers</td>
<td>faAloEamalu Alr~iyaADiy SaEobN wa$saA}ik</td>
</tr>
<tr>
<td></td>
<td>wayaHotaAju AIY jaladN waSaborN</td>
</tr>
<tr>
<td>GRU 1-Layer</td>
<td>faAloEamalo Alr<del>iyaADiy</del>u SiEobN wa$saA}ikN</td>
</tr>
<tr>
<td></td>
<td>wayaHotaAju AIY jalidN waSaborN</td>
</tr>
<tr>
<td>GRU 2-layer</td>
<td>faAloEamalu Alr<del>ayaADiy</del>u SiEobN wa$saA}ikN</td>
</tr>
<tr>
<td></td>
<td>wayaHotaAju AIY jalidN waSaborN</td>
</tr>
<tr>
<td>GRU 3-layer</td>
<td>faAloEamalu Alr<del>iyaADiy</del>i SaEobN wa$saA}ikN</td>
</tr>
<tr>
<td></td>
<td>wayaHotaAju AIY jaladN waSaborN</td>
</tr>
<tr>
<td>Att-LSTM 1-layer</td>
<td>faAloEamala Alr~iyaADiy SaEobN wa$saA}ikN</td>
</tr>
<tr>
<td></td>
<td>wayaHotaAju AIY jaladN waSabor</td>
</tr>
<tr>
<td>Att-LSTM 2-layer</td>
<td>faLoloEamalu Alr<del>iyaADiy</del>u SaEobN wa$saA}ikN</td>
</tr>
<tr>
<td></td>
<td>wayaHotaAju AIY jaladN waSaborN</td>
</tr>
<tr>
<td>Att-LSTM 3-layer</td>
<td>faAloEamalu Alr~iyaADiy SaEobN wa$saA}ikN</td>
</tr>
<tr>
<td></td>
<td>wayaHotaAju AIY jaladN waSaborN</td>
</tr>
<tr>
<td>Att-GRU 1-layer</td>
<td>faAloEamilu Alr~iyaADiy SaEobN wa$saA}ikN</td>
</tr>
<tr>
<td></td>
<td>wayaHotaAju AIY jaladN waSabor</td>
</tr>
<tr>
<td>Att-GRU 2-layer</td>
<td>faAloEamalu Alr~iyaADiy SaEobN wa$saA}ikN</td>
</tr>
<tr>
<td></td>
<td>wayaHotaAju AIY jaladN waSaborN</td>
</tr>
<tr>
<td>Att-GRU 3-layer</td>
<td>faLoloEamalu Alr<del>iyaADiy</del>u SaEobN wa$saA}ikN</td>
</tr>
<tr>
<td></td>
<td>wayaHotaAju AIY jaladN waSaborN</td>
</tr>
<tr>
<td>Transformer-GRU 3-layer</td>
<td>faAloEamalu Alr<del>iyaADiy</del>u SaEobN wa$saA}ikN</td>
</tr>
</tbody>
</table>
Our evaluation is validated by consulting with three Arabic native speakers. Each ambiguous word in each instance that exists in our dataset is checked against the correct specified diacritic. Table 5.4 shows the result of the WSD after deploying the Arabic diacritic restoration. Obviously, the Attentive-GRU using 3-layers Transformer generates the most accurate diacritic marks on our targeted ambiguous words compared to all other models.

We do believe that the use of GRU as an RNN structures and pre-trained model using Transformer are the perfect fit compared to the other RNN structures such as LSTM, as evidenced by our diacritic restoration and WSD experiments. The best result reported in Chapter III by the RETROFIT algorithm using Skip-gram learning representation model was 83.22%, while our current approach using pre-trained embeddings with attentive-GRU reports accuracy of 87.66%, improving by 4%.

Table 5.4: Accuracy of WSD using all Arabic diacritic restoration models.

<table>
<thead>
<tr>
<th>BiRNN Layers</th>
<th>LSTM</th>
<th>GRU</th>
<th>Attentive-LSTM</th>
<th>Attentive-GRU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-Layer</td>
<td>73.11%</td>
<td>73.56%</td>
<td>75.55%</td>
<td>76.13%</td>
</tr>
<tr>
<td>2-Layers</td>
<td>75.53%</td>
<td>76.13%</td>
<td>77.12%</td>
<td>81.12%</td>
</tr>
<tr>
<td>3-Layers</td>
<td>78.01%</td>
<td>80.73%</td>
<td>83.18%</td>
<td>85.82%</td>
</tr>
<tr>
<td>3-Layers Transformer-GRU</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>87.66%</td>
</tr>
</tbody>
</table>

5.3 Discussion and Conclusion

In this chapter, we aimed at deploying both Arabic diacritic restoration models and context representations for disambiguating Arabic words. Arabic WSD has improved using full diacritic restoration, encouraging work in other languages which have the same
language orthography. We now have seen how seq2seq learning improve restoration diacritic patterns, thus, making Arabic diacritic restoration as an intermediate step toward an NLP end task should be considered.

Since this approach mainly depends on Arabic diacritic models and partially on learning representation, improving this approach requires taking closer look on both components. One way to improve this approach was to use transfer learning irrespective to the approach, whether it is for the learning representation or the seq2seq models. Since training an RNN require lots of time with less data, we use transfer learning involving in google-transformer to obtain a prior information of Arabic characters and diacritics which is beneficial regarding at exploring deeper and more inconspicuous pattern. The following chapter provides a deep and technical proposed approaches which we think will produce improvement.

Merging Arabic diacritic for words disambiguation has limited the complexity of words disambiguation. Table 5.4 shows the improvement of WSD in several models compared to only using words and context representations approach introduced in chapter III. This approach encourages to have a reliable and well structured Arabic diacritic resource, this can benefit not only WSD but also many other NLP tasks such as machine translation, text to speech, and topic modelling.
CHAPTER VI

CONCLUSION AND FUTURE WORK

In this chapter, we review the entire dissertation as well as the contributions of our work toward fulfilling the needs of the Arabic NLP community. We also suggest ideas and contributions for the future work.

6.1 Review of the Dissertation

In this dissertation, we intended to help the Arabic NLP community in many ways. First, we preprocessed and cleaned a large corpus and made it available for public use\(^1\). This process involved removing unnecessary letters such as Arabic stopwords, English characters, diacritic marks, and normalization of certain Arabic letters. This corpus can be used to perform linguistic statistical analysis such as the task of learning vector representation of words.

Second, we studied the effect of using multiple Arabic stemming algorithms in the context of word semantic similarity task. We aimed at investigating if the use of stemming would

\(^1\)https://github.com/LincLabUCCS/Arabic-Word-Sense-Disambiguation
lead to better vector representations. Third, we tackled the problem of word sense disambiguation using the best output of the previous stage. Multiple algorithms in conjunction with the AWN were used to obtain specific-sense embeddings. In order to evaluate our models, we developed a word sense disambiguation system as well as collecting and developing a specific-purpose dataset and made it available for public use.

Fourth, we developed an approach to tackle the problem of WSD in a different way. Rather than finding the appropriate of undiacritized words in context by taking into account the representation vector of the associated sentence, we detected the word sense by restoring Arabic diacritic marks. Finding the correct diacritic marks of a word reduces the complexity of the word sense detection problem. We treated the problem of Arabic diacritic restoration as a sequence to sequence problem, developing different approaches.

Finally, in the last task of our dissertation, we compared the accuracy obtained from the third task against the fourth task.

6.2 Future Work

In this section, we suggest ideas which we think might improve the models we presented in this dissertation. We worked on two Arabic NLP tasks involving in word sense disambiguation and Arabic diacritic restoration. The following subsections describe and explain the future work.
6.2.1 Word Sense Disambiguation Augmented by Semantic Similarity Measures

In our previous work, we demonstrated that new advancements in word embeddings deliver promising results. Both proposed algorithms led to encouraging improvement using the AWN. Our proposed approaches will still concentrate on the same problem, using the same resources. We intend to improve the problem of Arabic word sense disambiguation further by proposing different methodologies. We believe this will lead to even better outcomes. In the following sections, we present the general ideas and algorithms.

6.2.1.1 Observation from Previous Work

Using new advancements in word embedding, and exploiting the AWN appear to be good choices for the problem of Arabic WSD. We observed in our work that synsets which do not have enough glosses tend to give false predictions. On the other hand, synsets that have enough number of glosses mostly produce correct predictions. This is actually not a surprising observation, and needs to be overcome. A goal would be to enrich each synset with sufficient numbers of glosses to describe them in this dissertation.

In approaches in this dissertation, we used both synonms and hypernyms as features to describe each synset. Our new methodologies will incorporate other features such as similarity measures between semantic concepts in the AWN. We aim to investigate and analyze the adequacy of these measures. Each synset will absorb glosses based on surrounding semantic concepts using these similarity measures. Since the AWN is a graph based
structure, its API embodies a variety of similarity measures, namely Wu-Palmer (Wup) [46], Leacock and Chodorow (LCH) [48], Li [50] and Path measure [103].

6.2.1.2 Semantic Similarity Measures

Semantic similarity measures compute relatedness between two semantic concepts based on information obtained from a hierarchical taxonomy. The following briefly discusses each measure we may be able to use.

- The Wup similarity measure computes the similarities between two given concepts C1 and C2 considering their depth in the taxonomy, along with the depth of the Least Common Subsumer (LCS) which is the most specific concept the two concepts share as an ancestor [46].

\[
W_{up}(C1, C2) = \frac{2 \times \text{depth}(\text{lcs}(c1, c2))}{\text{depth}(c1) + \text{depth}(c2)}
\]  

(1)

- LCH is a path measure obtained from the taxonomy based on the shortest path between two concepts C1 and C2, and the maximum depth (mdepth) of the taxonomy in which both concepts occur [48].

\[
LCH(C1, C2) = - \log \frac{\text{ShortestPath}(C1, C2)}{2 \times mdepth}
\]  

(2)

- Path measure computes the similarity based on the shortest path connecting two concepts(C1,C2) by counting the number of nodes (NodeDis).
\[
\text{Path}(C1, C2) = \frac{1}{\text{NodeDis}(C1, C2)}
\]  

- Li is a path-based measure using multiple information sources extracted from the taxonomy, which involved in shortest distance between the word pairs and the depth of the words in the taxonomy [50].

6.2.1.3 Future Enhancement

In our work, we used a synset’s synonyms and hypernyms as features to describe each synset. The proposed work will benefit from the semantic relatedness between synsets and their neighbors based on similarity measures. The first and second approaches proposed will work with the RETROFIT and ArabSenEmb (Chapter 3.4.1 and 3.4.2), respectively.

6.2.1.4 Modified Version of RETROFIT Algorithm

In this approach, we use the RETROFIT algorithm described in section 3.5.1. The following takes the AWN as an input and produces an ontology \( \Omega \) which is an input to RETROFIT. We assign \( \beta_r \) the hyper-parameter as either 1.0 or .5 based on the similarity threshold. \( S_{ij} \) denotes the target synset word, \( Sid_{ij} \) denotes the target synset’s ID, \( S_{{i}'{j}'} \) denotes the neighbor synsets of the target synset, and \( Sid_{{i}'{j}'} \) denotes the neighbor synsets’ ID. The procedure of this algorithm is shown bellow.

The outer loop extracts all synsets in the AWN along with their IDs. The inner for loop extracts the neighbor synsets of the targeted synset based upon the similarity between the target synset and the neighbor synsets. The neighbor synsets, which are considered glosses of the target synset, are the ones that meet the proposed condition as shown in line
Algorithm 2 RETROFIT Modified Version

1: procedure ONTOLOGY $\Omega$ (AWN)
2: for each $S_{ij}$ do
3:   $Sid_{ij} \leftarrow$ SynsetID($S_{ij}$)
4:   for each $S_{ij}'$ do
5:     $Sim(S_{ij}, S_{ij}') \leftarrow$ ComputeSimPathLChWuLI($S_{ij}, S_{ij}'$)
6:     if $Sim(S_{ij}, S_{ij}') >$ threshold
7:        then $\beta_r \leftarrow 1.0$
8:     $Sid_{ij}' \leftarrow$ SynsetID($S_{ij}'$)
9:     if threshold > $Sim(S_{ij}, S_{ij}') >$ threshold1
10:        then $\beta_r \leftarrow 0.05$
11:     $Sid_{ij}' \leftarrow$ SynsetID($S_{ij}'$)
12:   end for
13: return Ontology $\Omega$ ($S_{ij}, Sid_{ij}, S_{ij}', Sid_{ij}', \beta_r$)
14: end for
15: end procedure

6 and 9. The purpose of the condition is to allow neighbor synsets to be glosses, either having a higher similarity than a threshold (line 6), or ranging between two thresholds (line 9).

6.2.1.5 Modified Version of ArabSenEmb Algorithm

This approach follows the same modifications as the previous one with a slight change involved. It does not extract the IDs of the synset’s neighbors. This is because of the nature of the ArabSenEmb algorithm (section 2.4.2). Following the same notation as the first algorithm, the procedure for the second algorithm is shown.

6.2.2 Combining Word-level and Character-level Sequence to Sequence process for Arabic Diacritic Restoration

In our main approach to improving Arabic diacritic restoration, we totally depended on character level granularity, considering the input as a sequence of characters, without
Algorithm 3 ArabSenEmb Modified Version

1: procedure ONTOLOGY Ω (AWN)
2: for each $S_{ij}$ do
3: $Sid_{ij} \leftarrow SynsetID(S_{ij})$
4: for each $S_{ij}'$ do
5: $Sim(S_{ij}, S_{ij}') \leftarrow ComputeSim_{Path, LCh, WuP, Li}(S_{ij}, S_{ij}')$
6: if $Sim(S_{ij}, S_{ij}') > \text{threshold}$
7: then $\beta_r \leftarrow 1.0$
8: if $\text{threshold} > Sim(S_{ij}, S_{ij}') > threshold1$
9: then $\beta_r \leftarrow 0.05$
10: end for
11: return Ontology Ω ($S_{ij}, Sid_{ij}, S_{ij}', \beta_r$)
12: end for
13: end procedure

giving heed to other possible and levels of granularity. In morphologically rich languages like Arabic, multiple levels of granularity may help capture deeper linguistic structure and hidden information. In the NLP community, there has been an increase in interest in combining word and character levels for different NLP tasks, showing promising achievements. Combining word and character levels has been carried out in a verity of NLP tasks. A hybrid approach, composing word and character levels to obtain word vector representations [104]. Another application of both word and character levels was performed for predicting the next word by concatenating word and character vector representations [105]. Relational classification on informal text task has been performed by word-character level combination [106]. Other approaches following the same concept have involved in neural machine translation [107], question answering [108], speech recognition [109], and named entity recognition [110].
6.2.2.1 Future Extension

In this section, we present an approach for Arabic diacritic restoration that combines information from both word and character embeddings as shown in Fig 5.1.

Figure 6.1: The main concept of our proposed word-character level combination.

The above digram shows the main concept behind our proposed approach. The approach attempts to use levels of granularity in both word and character. In the word level, a Bi-LSTM architecture stores information about the word sequence pattern. The word representations are concatenated with the character embeddings. Our approach with investigate if augmenting the character embeddings with the word representations will lead to better function mapping from character vectors to their associated diacritic label.
6.3 Conclusion of the Dissertation

In this dissertation, we tackled the problem of Arabic word sense disambiguation, which attempts to obtain the specific meaning of a word given its context. We first tackled this by learning the representation of all sense-agnostic word embeddings. After that, we developed WSD algorithms to obtain a sense-specific embedding for each candidate sense. Obtaining word representation in a vector space is a preprocessing step for solving many different types of NLP tasks. This technique can improve WSD as well as overcome many challenging barriers such as the need for external resources, and only depending on local context features.

With the rapid increase of interest in addressing time series problems with a seq2seq methodology, we changed our approach to solving the problem of WSD. Since Arabic vowels are expressed as diacritics in Arabic, and not written explicitly in MSA, the WSD becomes more actual. Our focus turned into treating Arabic diacritic task using seq2seq methodology. Seq2Seq models presented in this dissertation attempted to map sequence patterns of non-diacriticized Arabic characters to the desired sequence of diacritic marks. Seq2seq models, in our work, showed a robust ability to take a sequence of inputs in the nature of a time series task, and produce the corresponding output.

Our final task was to perform automatic diacritization on the dataset that was already used for the problem of WSD in our first task. We compared several configurations for Arabic diacritic restoration to see if diacriticization helps perform WSD. Our experiments show that diacriticizing an Arabic text for WSD improved the performance of WSD by 4%.
This dissertation suggests that we use Arabic diacritic restoration as a pre-processing step for WSD. One big short-coming of this approach is the lack of broad, well-structured, and regularly updated source for diacritized Arabic words dictionary. We do believe that this type of dictionary will have a positive impact on several Arabic NLP tasks. Not only Arabic, but also it is likely to benefit languages that have the similar orthography.
REFERENCES


[57] M. Banko and E. Brill, “Scaling to very very large corpora for natural language dis-
ambiguation,” in Proceedings of the 39th Annual Meeting on Association for Com-

[58] I. A. El-Khair, “1.5 billion words Arabic corpus,” arXiv preprint arXiv:1611.04033,
2016.


[60] S. Elkateb, W. Black, P. Vossen, D. Farwell, H. Rodríguez, A. Pease, and M. Alkhal-
ifa, “Arabic wordnet and the challenges of Arabic,” in Proceedings of Arabic


[63] K. Taghva, R. Elkhoury, and J. Coombs, “Arabic stemming without a root dictio-
nary,” in Information Technology: Coding and Computing, 2005. ITCC 2005. Inter-

[64] L. S. Larkey, L. Ballesteros, and M. E. Connell, “Light stemming for Arabic in-
formation retrieval,” in Arabic computational morphology, pp. 221–243, Springer,
2007.

World Academy of Science, Engineering and Technology, International Journal of
Social, Behavioral, Educational, Economic, Business and Industrial Engineering,


[73] A. Chennoufi and A. Mazroui, “Morphological, syntactic and diacritics rules for automatic diacritization of Arabic sentences,” Journal of King Saud University-


[80] K. Shaalan, H. M. Abo Bakr, and I. Ziedan, “A hybrid approach for building Ara-


