SPRING 2021 CIE 555 Sunday, June 15, 2021

Project Phase 3: Final Report

Alaa Alajmy (ID # 201700095)
Moemen Khaled (ID # 201700873)

Arabic Text Diacritization

Contents:

I. Introduction

a. Problem Definition and Motivation

Ancient as the era and spread as the Earth, Arabic is one of the fastest growing languages across the internet. The Arabic language is set apart from most other languages by its right to left writing style and its use of diacritics. With 28 letters and 8 basic diacritics, the Arabic alphabet forms the base for multiple languages besides Arabic itself. [1]

The Arabic language co-exists in two forms: colloquial, or spoken, Arabic and standard, or written, Arabic. Standard Arabic could be further separated into two forms: Classical Arabic and Modern Standard Arabic. Since the Arabic speakers read is vastly different from the one they speak and since diacritization often produces multiple very different words from the same sequence of letters, understanding undiacritized text is often problematic. Even fluent speakers are often unable to determine the proper diacritization in certain sentences. Yet, with 15 different combinations of diacritics, writers commonly find it challenging to pay enough attention to the proper grammatical rules of Arabic. Manually diacritizing text is time consuming and requires considerable linguistic expertise, thus many magazines, articles, and books are published with no sufficient Diacritization. [1]

Hence, the idea of automatic diacritization was born. In formal terms, given a non-empty sequence of partially or fully undiacritized Arabic text, we attempt to automatically find its correct diacritization. [1] Besides simplifying text deciphering on Arabic readers, automatic diacritization could be utilized to enhance text-to-speech systems or search results. [2]

b. Previous Work

There are two main approaches to Arabic language diacritization in the literature: traditional approaches and machine learning approaches. Traditional approaches, including morphological analysis and Hidden Markov Models, analyze Arabic texts and apply diacritization rules directly. This hard-coded procedure leads to a compromised flexibility of such models to treat a wide range of diacritization problems and thus, leads to a compromised accuracy. Famous traditional diacritization systems include Farasa, Harakat, MADAMIRA, and Mishkal, which is the best performance-wise. [1]

On the other hand, machine learning approaches rely on large compiled diacritized datasets that aim to model the diacritization process accurately without knowing its strict rules. Although Natural Language Processing using Machine Learning is prominent for English and Chinese, for example, it is still an emerging topic for Arabic.

Most recently, Fadel et at. created a series of feed-forward neural network (FFNN) models and RNN models to attempt the same problem. These models were all

trained using excerpts of the Holy Koran and the Tashkeela Corpus, the only freely avIn their RNN attempt, the Fadel et al. expanded their training set from 2.3 million to 22.4 million characters. This model used Bidirectional CuDNN Long Short-Term Memory (BiCuDNNLSTM) layers and their hidden units. After several iterations, the final model used 2 BiCuDNNLSTM layers and 512 hidden layers. The model was trained on Google Colab over 50 epochs using the Adam optimization algorithm with 0.001 learning rate, 0.9 beta1, 0.999 beta2, 10 epsilon, 256 batch size, and categorical cross-entropy loss −7 function. This model returned results that surpassed the Shakkala model. [2]

Two measures have been developed to measure the accuracy of a diacritization system. The first is the Diacritic Error Rate (DER) which is the percentage of misclassified Arabic characters whether the character has 0, 1, or 2 diacritics. The other is Word Error Rate (WER) which is the percentage of words which have at least one misclassified Arabic character. [1] On both measures, it has been shown that deep learning approaches are much more accurate than traditional approaches.

c. Objectives

In this project, we aim to take a deep learning approach to the problem of Arabic Diacritization. We attempt to reproduce the results by Fadel et al., using an RNN model with BiCuDNNLSTM layers. [2] At the end, we attempt to improve these results by tuning the hyperparameters.

Our main performance measures are the DER and WER. Surveying the performance of Fadel et al.'s networks, we generally aim to reach a maximum DER of 3% and a maximum WER of 8% [2].

II. Methodology

a. The Dataset

The used dataset was extracted by Fadel et al. from the Holy Koran and the Tashkeela Corpus. The only freely available Arabic diacritization dataset, Tashkeela consists of 97 Classical Arabic books and 293 Modern Standard Arabic. The dataset is split into:

- train.txt - Contains 50,000 lines of diacritized Arabic text which can be used as training dataset
- val.txt - Contains 2,500 lines of diacritized Arabic text which can be used as validation dataset
- test.txt - Contains 2,500 lines of diacritized Arabic text which can be used as testing dataset

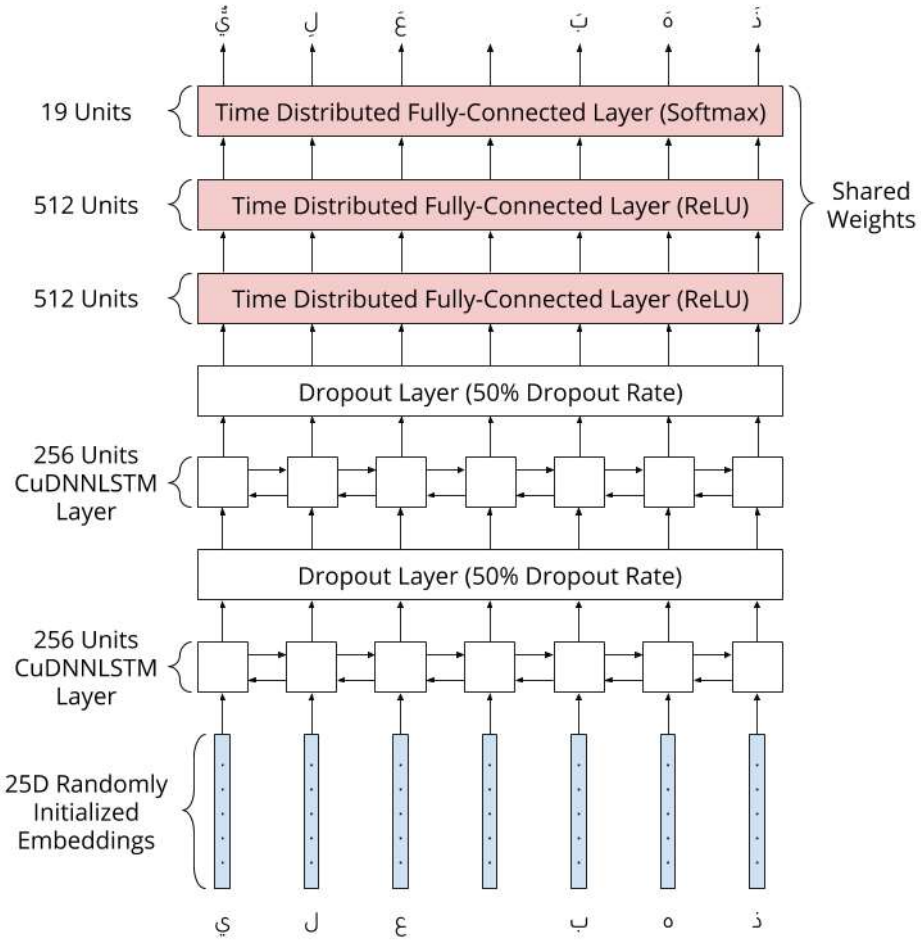
The imported dataset was compiled by Fadel et al. from the Tashkeela corpus and the Holy Koran. Fadel et al. performed the following cleaning steps to the dataset:

- Removing HTML tags using BeautifulSoup12 library
- Removing URLs using regex
- Fixing misplaced or duplicate diacritizatics
- Removing English letters
- Removing the special Arabic character Kashida as it is irrelevant to diacritization
- Seperating numbers by preceding and following whitespaces using regex
- Removing multiple whitespaces using regex

Then, in processing, the raw data was split into lines at any pranthesis or punctuation. Long lines were then split into ones shorter than 500 characters. Further processing splits each line an array of characters represented by their corresponding integers and an array of their diacritics represented by one-hot-encoded arrays. The character arrays are entered to RNN model as input and the diacritics array are the model's targeted output. The training data contains around 300,000 training lines shorter than 500 characters, while the validation data contains around 14,700 lines.

d. The Model

Our initial model is the model created by Fadel et. al with the following structure:



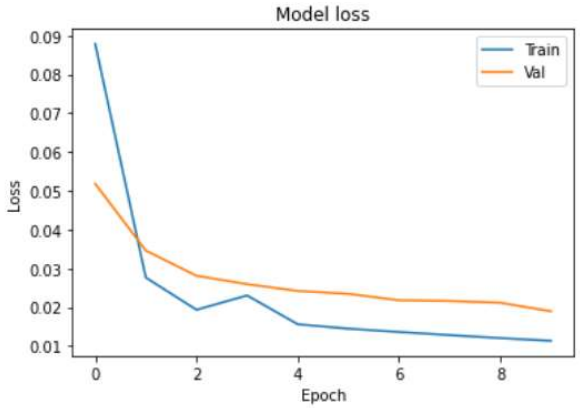
The used CuDDLSTM layers are bidirectional LTSM layers, where CuDDLTSM is a fast LSTM implementation backed by CuDNN and runable on GPU with the tensorflow backend. A bidirectional LTSM runs inputs in two ways, one from past to future and one from future to past, allowing the model to preserve information from both past and future.

We initially train the model for 20 epochs, and arrive at:

	With case ending	Without case ending	With case ending	Without case ending
DER	Including no diacritic		Excluding no diacritic	
%	3.79	3.19	4.38	3.66

	With case ending	Without case ending	With case ending	Without case ending
WER	Including no diacritic		Excluding no diacritic	
%	11.17	7.05	10.78	6.85

Observing the model's loss plot below, we decide to train the model longer for 50 epochs. We use early stopping to ensure the model does not overfit.



We then attempt to train longer, a few epochs by a few epochs. Training 4 more epochs, we achieve:

	With case ending	Without case ending	With case ending	Without case ending
DER	Including no diacritic		Excluding no diacritic	
%	3.21	2.66	3.71	3.06

	With case ending	Without case ending	With case ending	Without case ending
WER	Including no diacritic		Excluding no diacritic	
%	9.59	5.89	9.25	5.74

Finally, another 4 epochs later:

	With case ending	Without case ending	With case ending	Without case ending
DER	Including no diacritic		Excluding no diacritic	
%	3.07	2.53	3.55	2.91

	With case ending	Without case ending	With case ending	Without case ending
WER	Including no diacritic		Excluding no diacritic	
%	9.11	5.58	8.79	5.45

We additionally attempted to increase the model's depth by adding a third bidirectional LTSM layer and by adding a third bidirectional LTSM layer and halving the width of each bidirectional LTSM. The first attempt was unsuccessful in that its training required an impractical amount of time. The second attempt yielded results worse than the primary model's and was terminated short of completion.

III. Results

Our best-performing model achieves the following results, which are comparable to the results by fadel et. al. [1]

	With case ending	Without case ending	With case ending	Without case ending
DER	Including no diacritic		Excluding no diacritic	
%	3.07	2.53	3.55	2.91

	With case ending	Without case ending	With case ending	Without case ending
WER	Including no diacritic		Excluding no diacritic	
%	9.11	5.58	8.79	5.45

IV. Members Contributions

Research Topic	Moemen
Create Proposal	Moemen & Alaa
Adapt Code	Moemen
Document Code	Alaa
Train Further	Moemen & Alaa
Compile Report	Alaa

V. References

[1] A. Fadel, I. Tuffaha, B. Al-Jawarneh and M. Al-Ayyoub, "Arabic Text Diacritization Using Deep Neural Networks," 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS), 2019, pp. 1-7.

[2] Fadel, Ali, Ibraheem Tuffaha, Bara' Al-Jawarneh and M. Al-Ayyoub, “Neural Arabic Text Diacritization: State of the Art Results and a Novel Approach for Machine Translation.” WAT@EMNLP-IJCNL, 2019.

[3] Barqawiz. (n.d.). Barqawiz/Shakkala. Retrieved May 09, 2021, from https://github.com/Barqawiz/Shakkala

[4] مشكال النصوص العربية. Retrieved May 08, 2021, from https://tahadz.com/mishkal