Arabic Diacritization with Viterbi N-gram Model, Transformers, and Recurrent Neural Networks

Reyhan Abdul Quayum
New York University
rrq2003@nyu.edu

Benchik Bayor
New York University
bb2998@nyu.edu

Cheng Ji
New York University
cj2220@nyu.edu

Usaid Malik
New York University
usaid.malik@nyu.edu

Abstract

This project develops multiple systems to automate the vocalization of unvocalized Arabic texts, addressing a significant challenge for non-native speakers and enhancing readability and understanding. We leverage a combination of Natural Language Processing (NLP) techniques including an N-gram model employing the Viterbi algorithm, a Recurrent Neural Network (RNN), and a Transformer-based architecture, each selected for their strengths in contextual analysis and sequence modeling.

The core dataset comprises pre-vocalized texts from a variety of Arabic media sources such as the Qur'an, Hadith, children's literature, and beginner Arabic books, ensuring a rich variety of linguistic contexts. Our system is evaluated against a development set aiming for a Diacritic Error Rate below 10%. We define Diacritic Error rate to be the proportion of predicted diacritics that were incorrect from the actual diacritic that was on the letter.

The implementation details for the transformer based model includes a fine-tuning of the Asafaya BERT-based Arabic language model and a modified Hugging Face Model for diacritic prediction, with preprocessing steps that prepare and tokenize the input data effectively. Additionally, we used an RNN with both Bidirectional LSTM layers and Unidirectional LSTM layers.

We observed the lowest Diacritic Error Rate (DER) from our Recurrent Neural Network (RNN) model, scoring a DER of 3.25%. Succeding this error rate, we unexpectedly observed a greater error rate from our transformer model at 7.0%. Lastly, our Viterbi n-gram model attained the largest DER.

1 Introduction and Motivation

Arabic diacritization, the process of restoring diacritical marks to undiacritized text, is a significant challenge in NLP. For example, the word علم which may mean the following different words with different diacritics as seen in Table 1.

Arabic Word	IPA	Meaning
علم	ʕ_l_m_	Meaning Ambiguous
عَلِمَ	ʕalima	He knew
عَلَمْ	ʕalam	Flag
عِلْم	ʕalm	Knowledge
عُلِمَ	ʕulima	(It) was known

(1)

Despite its importance, there has been a limited focus on this task, with existing studies often relying on traditional approaches that may not generalize well across different text types. Recent advancements in NLP, including the development of sophisticated machine learning models and the increasing availability of diverse text corpora, offer new opportunities for addressing the complexities of Arabic diacritization.

This work aims to provide a comprehensive assessment of current diacritization techniques by leveraging various models, such as the Viterbi N-gram model, Recurrent Neural Networks (RNNs), and Transformer-based architectures. Our primary goal is to evaluate the performance of these models on primarily the Qur'an, as it is a fully diacritized text with widely available datasets on it that do not disagree on the vowels. We also used a variety of other Arabic media as datasets. These sources provide a broad range of linguistic contexts, which is crucial for developing robust diacritization systems.

We are particularly motivated by the need to enhance the accessibility and readability of Arabic text for non-native speakers. Automatic diacritization can significantly aid in language learning and comprehension, thereby broadening the reach of Arabic language resources. Additionally, in the context of creating new annotated language resources, identifying and adapting the most effective preprocessing tools can minimize manual correction efforts and improve the quality of automated annotations.

By providing a thorough and reproducible assessment of diacritization techniques, we aim to contribute valuable insights to the field and encourage further advancements in this important area of Arabic NLP.

2 Previous Work on Arabic Diacritization

Arabic diacritization has been approached using various techniques over the years. One of the early significant contributions was by Yaakov Gal (2002), who applied a Hidden Markov Model (HMM) to the task of vowel restoration in Arabic and Hebrew. This method utilized statistical models to predict the probability of diacritics based on the context of surrounding letters, showing promising results in handling the sequential nature of language data.

Belinkov and Glass (2015) introduced the use of Recurrent Neural Networks (RNNs) for Arabic diacritization. They treated the problem as a sequence labeling task, where each character in the input sequence is associated with a diacritic label. Their approach used Long Short-Term Memory (LSTM) units to capture long-term dependencies in the text, achieving high performance without relying on external tools. They experimented with different architectures, including unidirectional and bidirectional LSTM layers, and showed that deeper networks with multiple LSTM layers significantly improved diacritization accuracy.

In 2019, Fadel et al. proposed a novel approach using neural networks for Arabic text diacritization. They developed an end-to-end system that leveraged both character-level and word-level features to enhance the accuracy of diacritic restoration. Their method demonstrated state-of-the-art results, highlighting the importance of combining different levels of linguistic information.

The rise of transformer models, as discussed by Thomas Wolf et al. (2020), brought significant advancements in NLP, including diacritization tasks. Transformers, particularly models like BERT, have shown exceptional performance due to their ability to capture contextual information across long text spans. The pre-trained BERT model for Arabic, fine-tuned for diacritization tasks, has proven to be highly effective, as it leverages large amounts of text data to learn rich representations of language.

Additionally, CAMeL Tools, introduced by Obeid et al. (2020), is an open-source Python toolkit for Arabic NLP. It provides various tools for tasks including diacritization, which can be integrated into broader NLP pipelines. The toolkit is designed to be accessible and adaptable, supporting a wide range of applications in Arabic language processing.

Our approach seeks to compare these existing methodologies, using the Qur'an and religious texts as our corpora to offer a comprehensive comparative view into existing techniques in the field.

3 Data

We got our data from a variety of sources that utilize diacritized texts. These were a variety of Arabic media sources that were diacritized such as the Quran. The data was then parsed and split into two modes. One was the letter-based version with a letter on the left side and the corresponding diacritic on the right as tab-separated values. There were a variety of sentences however we believed context didn't matter as we were mainly focused on the grammatical structure of the language and how that affects the corresponding words. The sentences were then delimited with an end sentence character and the spaces were included to mark the end of a word. An example of the data is as follows below:

بِ سْ مِ

As is shown above the letters were separated and their corresponding diacritic was placed side by side in the text file, but in this example, they are shown on top of or below the letters.

Aside from the letter-based version we also created a word-based version following the same principles as the letter-based version also illustrated below. The diacritics were read left to right in the file with the corresponding diacritic placed on the proper letter and a blank diacritic in the sequence representing no diacritic

بسمِ ِ ْ اللّهِ ِ َ ْ الرّحمٰنِ ِ َ ْ َ ْ َ الرّحيمِ ِ ِ َ ْ

4 Viterbi

Initially, we began by constructing a probabilistic model for predicting the diacritical marks corresponding to the letter. We framed the problem similar to the problem of predicting the next POS tag and as such used the Viterbi Algorithm for predicting the next sequence of diacritical markings given the current word.

We first constructed a uni-gram version of the Viterbi algorithm and tested it not on words but rather on each letter. We split our data into a sequence of letters for each of the sentences and then we predicted the diacritical mark associated with each letter finally, we recombined all the words to evaluate the effectiveness of the model

We found by using this data the Diacritic Error Rate of the model on a 90-5-5 split was 63.82% for the validations set and 66.16% for the test set

As the accuracy was low doing it in the letter fashion we assumed this was since the letter-based Viterbi resulted in too long sequences unable to capture the proper probabilities of the sequence. We then changed this into a word-based model where we trained the Viterbi algorithm on the words and the sequence of diacritical marks instead. This improved our Diacritic Error Rate to 47.82% for the validation set and 51.69% for the test set

We also tried an n-gram-based model using $n = 2$ with the word-based data as it showed to give better results which resulted in the following diacritic error rate 43.54% on the validation set and 46.32% on the test set.

We attribute this to the fact that there were too many states for the model to evaluate. In English the amount of POS tags for the Penn Treebank of 36 POS tags. This is not many and allows the Viterbi algorithm with enough probabilistic data to get a reasonable accuracy on these tags. However in our case a single word can take on possible states and as such for 3 letters the number of possible states would be 8^3 which is an exponential number of states as such the length of the word increases, with some words reaching 7 letters in size, the number of possible states grows exponentially making it difficult for the Viterbi algorithm to accurately evaluate the proper probabilistic path to take even with an n-gram model which only marginally improves performance.

5 Recurrent Neural Network

Using a Recurrent Neural Network (RNN) model, Belinkov and Glass showed experimentally that the model can achieve high performance without relying on any external tools but solely from diacritized corpus. The problem is treated as a sequence classification task, where each character in the input sequence is associated with a label.

We use the following approach described in Belinkov and Glass, 2015:

1. Let $\mathbf{w}$ be the letter sequence mapping to vector sequence $\mathbf{x}$. Then, map $\mathbf{x}$ to a hidden sequence $\mathbf{h}$ using one or more hidden layers. In the RNN, the input layer stacks previous and future letter vectors to make the model learn contextual information.

2. Then, map $\mathbf{h}$ to output label sequence $\mathbf{l}$. The hidden layers are composed of LSTM units that capture long-term dependencies in the sequence. Belinkov and Glass experimented with different architectures, including unidirectional and bidirectional LSTM (B-LSTM) layers, as well as multiple layers to improve performance.

3. The output layer applies a softmax function to produce a probability distribution over the possible diacritic labels for each character.

The paper experimented with different types of hidden layers and achieved different levels of accuracy. Error rates are measured with Diacritic Error Rates (DERs), where it is defined as wrongly predicted diacritics over the total number of diacritics. With the corpus from Arabic treebank following the

Train/Dev/Test split, the following DERs are achieved in the paper:

	DER		
Model	All	End	# params
Feed-forward	11.76	22.90	63 K
Feed-forward (large)	11.55	23.40	908 K
LSTM	6.98	10.36	838 K
B-LSTM	6.16	9.85	518 K
2-layer B-LSTM	5.77	9.18	916 K
3-layer B-LSTM	5.08	8.14	1,498 K

(2)

The table shows clearly that the LSTM models can achieve much better performance over simple feed-forward networks. Even if we increase the number of parameters in the feed-forward model, the LSTM is still much better in performance, with 3-layer Bidirectional LSTM having the highest performance of 3.25% DER.

In conclusion, the model shows ability to learn from diacritics only, without any external resources such as the BERT pre-trained model used in the transformer model. However, the models show slight difficulty with proper names and rare words, which do require external knowledge to resolve.

6 Transformer

Using a Transformer model, we explore the application of pre-trained language models for the task of diacritic restoration in Arabic text. The approach leverages the Asafaya BERT architecture and Hugging Face's conventions for model training and evaluation. The problem here is treated as a sequence-to-sequence classification task, where each character in the input sequence is associated with a label indicating the presence and type of diacritic.

We use the following approach for our model and data:

1. Load the data and preprocess it. The data consists of an Arabic corpus using words and not letters with and without diacritics, split into input (text without vowels) and target (text with vowels) sequences.

2. Split the data into training and testing sets. Tokenize the sequences using a pre-trained BERT tokenizer.

3. Add tokens to the BERT Model and Finetune a pre-trained Model on the task of masked language modeling to predict diacritics for each character in the input sequence.

6.1 Model and Tokenizer Initialization

We initialize the tokenizer and model using the pre-trained BERT model from the Hugging Face library:

```
Load pre-trained tokenizer and model
tokenizer = BertTokenizer.from_pretrained
("asafaya/bert-base-arabic")
model = BertForMaskedLM.from_pretrained
("asafaya/bert-base-arabic")
```

6.2 Training and Evaluation

We split our data into training and testing sets and created a DataLoader instances using the scikitlearn library splitting.

We modify the tokenizer to include diacritics and resize the model's token embeddings to accommodate the vowel prediction task at hand that includes the vowels on alif and all the other characters in a shorthand arabic so we can get character recognition.

We define the training arguments and instantiate the Trainer using hugging face's transformer trainer classes with a number of epochs being 2 , learning rate of 2e-5 and a batch size of 8.

6.3 Results

After training, we evaluated the model's performance and found the DER to be 7.12%:

7 Conclusions–Future Work

On the outset, we observed the highest accuracy from the RNN model, achieving a DER of 3.5 % we attribute this to the use of the B-LSTM layer and Uni-LSTM layer alongside our architecture for the RNN. We also attribute this to the method of how our data was preprocessed and fed to the network compared to the method used by Belinkov and Glass.

This suggests that RNNs, particularly those leveraging Long Short-Term Memory (LSTM) units, are highly effective for capturing the sequential dependencies inherent in Arabic text. The Viterbi N-gram models, while useful, were hindered by the exponential increase in state spaces, leading to higher error rates

Although Transformer models showed promising results, particularly with the Asafaya BERT architecture, they still fell short of the RNN's performance we atrtibute this to the limited amount of data required to make a transformer model succesful particularly in this case. Furthermore our framing of the problem may have contributed to the error rate being high.

For future work, several avenues can be explored to enhance the accuracy and precision metrics of Arabic diacritization systems. One potential way is to integrate the approaches into a hybrid model and introduce rule-based methods based off of Arabic grammatical principles. Another way is to increase the variety of the training corpora to include more varied texts, including contemporary texts from Modern Standard Arabic in addition to Classical Arabic.

Acknowledgments

This research project was overseen by Dr. John Ortega and Dr. Adam Meyers.

This document has been adapted by Steven Bethard, Ryan Cotterell and Rui Yan from the instructions for earlier ACL and NAACL proceedings, including those for ACL 2019 by Douwe Kiela and Ivan Vulić, NAACL 2019 by Stephanie Lukin and Alla Roskovskaya, ACL 2018 by Shay Cohen, Kevin Gimpel, and Wei Lu, NAACL 2018 by Margaret Mitchell and Stephanie Lukin, ACL 2017 by Dan Gildea and Min-Yen Kan, NAACL 2017 by Margaret Mitchell, ACL 2012 by Maggie Li and Michael White, ACL 2010 by Jing-Shin Chang and Philipp Koehn, ACL 2008 by Johanna D. Moore, Simone Teufel, James Allan, and Sadaoki Furui, ACL 2005 by Hwee Tou Ng and Kemal Oflazer, ACL 2002 by Eugene Charniak and Dekang Lin, and earlier ACL and EACL formats written by several people, including John Chen, Henry S. Thompson and Donald Walker. Additional elements were taken from the formatting instructions of the *International Joint Conference on Artificial Intelligence* and the *Conference on Computer Vision and Pattern Recognition*.

References

[1] Yonatan Belinkov and James Glass. 2015. *Arabic Diacritization with Recurrent Neural Networks*

[2] Ali Fadel, Ibraheem Tuffaha, Bara Al-Jawarneh, and Mahmoud Al-Ayyoub. 2019. *Neural Arabic Text Diacritization: State of the Art Results and a Novel Approach for Machine Translation*

[3] Thomas Wolf et al. 2020. Transformers: State-of-the-Art Natural Language Processing

[4] Yaakov Gal. 2002. An HMM Approach to Vowel Restoration in Arabic and Hebrew.

[5] Ossama Obeid et al. 2020. CAMeL Tools: An Open Source Python Toolkit for Arabic Natural Language Processing