Multi-components System for Automatic Arabic Diacritization

Hamza Abbad and Shengwu Xiong(✉)

Wuhan University of Technology, Wuhan, Hubei, China
{hamza.abbad,xiongsw}@whut.edu.cn

Abstract. In this paper, we propose an approach to tackle the problem of the automatic restoration of Arabic diacritics that includes three components stacked in a pipeline: a deep learning model which is a multi-layer recurrent neural network with LSTM and Dense layers, a character-level rule-based corrector which applies deterministic operations to prevent some errors, and a word-level statistical corrector which uses the context and the distance information to fix some diacritization issues. This approach is novel in a way that combines methods of different types and adds edit distance based corrections.

We used a large public dataset containing raw diacritized Arabic text (Tashkeela) for training and testing our system after cleaning and normalizing it. On a newly-released benchmark test set, our system outperformed all the tested systems by achieving DER of 3.39% and WER of 9.94% when taking all Arabic letters into account, DER of 2.61% and WER of 5.83% when ignoring the diacritization of the last letter of every word.

Keywords: Arabic · Diacritization · Diacritics restoration · Deep learning · Rule-based · Statistical methods · Natural Language Processing

1 Introduction

Arabic is the largest Semitic language today, used by more than 422 millions persons around the world, as a first or second language, making it the fifth most spoken language in the world.

The Arabic language uses a writing system consisted of 28 letters but represented by 36 characters due to 2 letters which have more than one form[1]. Unlike Latin, Arabic is always written in a cursive style where most of the letters are joined together with no upper case letters, from right to left (RTL).

The writing system is composed of letters and other marks representing phonetic information, known as **diacritics**, which are small marks that should be placed above or below most of the letters. They are represented as additional

[1] The letter ت has another form represented as ة, and the letter ا has the following forms: أ, إ, آ, ء, ؤ, ئ, ى, depending on its pronunciation and position in the word.

J. M. Jose et al. (Eds.): ECIR 2020, LNCS 12035, pp. 341–355, 2020.
https://doi.org/10.1007/978-3-030-45439-5_23

Arabic characters in UTF-8 encoding. There are eight diacritics in the *Modern Standard Arabic* (MSA), arranged into three main groups:

Short vowels. Three marks: *Fatha*, *Damma*, *Kasra*.
Doubled case endings (Tanween). Three marks: *Tanween Fath* (*Fathatan*), *Tanween Damm* (*Dammatan*), *Tanween Kasr* (*Kasratan*).
Syllabification marks. Two marks: *Sukoon* and *Shadda* [46].

Shadda is a secondary diacritic indicating that the specified consonant is doubled, rather than making a primitive sound. The *Tanween* diacritics can appear only at the end of the word, and *Sukoon* cannot appear in the first letter. Besides, short vowels can be placed in any position. Furthermore, some characters cannot accept any diacritics at all (ex: ى), and some others cannot do that in specified grammatical contexts (ex: the definitive ال at the beginning of the word). The diacritics are essential to indicate the correct pronunciation and the meaning of the word. They are all presented on the letter د in Table 1.

Table 1. The diacritics of the Modern Standard Arabic

Diacritic	Arabic name	Transliteration	Diacritic	Arabic name	Transliteration
دَ	فَتْحَةٌ	Fatha	دُ	ضَمَّةٌ	Damma
دِ	كَسْرَةٌ	Kasra	دْ	سُكُونٌ	Sukoon
دً	تَنْوِينُ فَتْحٍ	Tanween Fath	دٌ	تَنْوِينُ ضَمٍّ	Tanween Damm
دٍ	تَنْوِينُ كَسْرٍ	Tanween Kasr	دّ	شَدَّةٌ	Shadda

These marks are dropped from almost all the written text today, except the documents intolerant to pronunciation errors, such as religious texts and Arabic teaching materials. The native speakers can generally infer the correct diacritization from their knowledge and the context of every word. However, this is still not a trivial task for a beginner learner or NLP applications [12].

The automatic diacritization problem is an essential topic due to the high ambiguity of the undiacritized text and the free word order nature of the grammar. Table 2 illustrates the differences made by the possible diacritizations of the

Table 2. The diacritizations of علم and their meanings

Diacritized form of علم	Meaning	Diacritized form of علم	Meaning
عَلِمَ	He knew	عَلَمٌ	A flag (nominative)
عَلَّمَ	He taught	عَلَمٍ	A flag (genitive)
عُلِمَ	It was known	عِلْمٌ	A science (nominative)
عُلِّمَ	It was taught	عِلْمٍ	A science (genitive)

word علم. As one might see, the diacritization defines many linguistic features, such as the part-of-speech (POS), the active/passive voice, and the grammatical case.

The full diacritization problem includes two sub-problems: *morphological diacritization* and *syntactic diacritization*. The first indicates the meaning of the word, and the second shows the grammatical case.

Two metrics are defined to calculate the quantitative performance of an automated diacritics restoration system: **Diacritization Error Rate** (DER) and **Word Error Rate** (WER). The first one measures the ratio of the number of incorrectly diacritized characters to the number of all characters. The second metric applies the same principle considering the whole word as a unit, where a word is considered incorrect if any of its characters has a wrong diacritic. Both metrics have two variants: One includes the diacritics of all characters (DER1 and WER1), and another excludes the diacritics of the last character of every word (DER2 and WER2).

We propose a new approach to restore the diacritics of a raw Arabic text using a combination of deep learning, rule-based, and statistical methods.

2 Related Works

Many works were done in the automatic restoration of the Arabic diacritics using different techniques. They can be classified into three groups.

Rule-based approaches. The used methods include cascading *Weighted Finite-State Transducers*[33], lexicon retrieval and rule-based morphological analysis [7]. One other particular work [9] used diacritized text borrowing from other sources to diacritize a highly cited text.

Statistical approaches. This type of approaches includes using *Hidden Markov Models* both on word level and on character level [8,18,21], *N-grams* models on word level and on character level as well [10], *Dynamic Programming* methods [24–26], classical Machine learning models such as *Maximum-entropy* classifier [46], and *Deep Learning* methods like the *Deep Neural Networks*, both the classical *Multi-Layer Perceptron* and the advanced *Recurrent Neural Networks*[6,14,32,36].

Hybrid approaches. They are a combination of rule-based methods and statistical methods in the same system. They include hybridization of rules and dictionary retrievals with morphological analysis, N-grams, Hidden Markov Models, Dynamic Programming and Machine Learning methods [5,15,17,20,23,31,35,37–39,42]. Some Deep Learning models improved by rules [2,3] have been developed as well.

Despite a large number of works done on this topic, the number of available tools for Arabic diacritization is still limited because most researchers do not release their source code or provide any practical application. Therefore, we will compare the performance of our system to these available ones:

Farasa [4] is a text processing toolkit which includes an automatic diacritics restoration module, in addition to other tools. It is based on the segmentation of the words based on separating the prefixes and suffixes using SVM-ranking and performing dictionary lookups.

MADAMIRA [34] is a complete morphological analyser that generates possible analyses for every word with their diacritization and uses an SVM and n-gram language models to select the most probable one.

Mishkal [44] is an application which diacritize a text by generating the possible diacritized word forms through the detection of affixes and the use of a dictionary, then limiting them using semantic relations, and finally choosing the most likely diacritization.

Tashkeela-Model [11] uses a basic N-gram language model on character level trained on the Tashkeela corpus [45].

Shakkala [13] is a character-level deep learning system made of an embedding, three bidirectional LSTM, and dense layers. It was trained on Tashkeela corpus as well. To the best of our knowledge, this is the system that achieves state-of-the-art results.

3 Dataset

In this work, the Tashkeela corpus [45] was mainly used for training and testing our model. This dataset is made of 97 religious books written in the *Classical Arabic* style, with a small part of web crawled text written in the *Modern Standard Arabic* style. The original dataset has over 75.6 million words, where over 67.2 million are diacritized Arabic words.

The structure of the data in this dataset is not consistent since its sources are heterogeneous. Furthermore, it contains some diacritization errors and some useless entities. Therefore, we applied some operations to normalize this dataset and keep the necessary text:

1. Remove the lines which do not contain any useful data (empty lines or lines without diacritized Arabic text).
2. Split the sentences at XML tags and end of lines, then discard these symbols. After that, split the new sentences at some punctuation symbols: dots, commas, semicolons, double dots, interrogation, and exclamation marks without removing them.
3. Fix some diacritization errors, such as removing the extra *Sukoon* on the declarative ال , reversing the ا + *Tanween Fath* and diacritic + *Shadda* combinations, removing any diacritic preceded by anything other than Arabic letter or *Shadda*, and keeping the latest diacritic when having more than one (excluding *Shadda* + diacritic combinations).
4. Any sentence containing undiacritized words or having less than 2 Arabic words is discarded.

After this process, the resulted dataset will be a raw text file with one sentence per line and a single space between every two tokens. This file is further shuffled then divided into a training set containing 90% of the sentences, and the rest is distributed equally between the validation and the test sets[2]. After the division, we calculated some statistics and presented them in Table 3.

Table 3. Statistics about the processed Tashkeela dataset

	Train	Val	Test
All tokens	31774001	1760397	1766844
Numbers only	80417	4462	4396
Arabic words only	27657285	1532625	1537878
Unique undiacritized Arabic words	351816	100799	101263
Unique diacritized Arabic words	626693	152752	153311

We note that the train-test Out-of-Vocabulary ratio for the unique Arabic words is 9.53% when considering the diacritics and 6.83% when ignoring them.

4 Proposed Method

Our approach is a pipeline of different components, where each one does a part of the process of the diacritization of the undiacritized Arabic text of the input sentence. Only a human-readable, fully diacritized Arabic text is needed to train this architecture, without any additional morphological or syntactic information.

4.1 Preprocessing

At first, only the necessary characters of the sentence which affect the diacritization are kept. These are Arabic characters, numbers, and spaces. The numbers are replaced by 0 since their values will most likely not affect the diacritization of the surrounding words. The other characters are removed before the diacritization process and restored at the end.

Every filtered sentence is then separated into an input and an output. The input is the bare characters of the text, and the output is the corresponding diacritics for every character. Considering that an Arabic letter can have up to two diacritics where one of them is *Shadda*, the output is represented by two vectors; one indicates the primary diacritic corresponding to every letter, and the other indicates the presence or the absence of the *Shadda*. Figure 1 illustrates this process.

The input is mapped to a set of 38 numeric labels representing all the Arabic characters in addition to 0 and the white space. It is transformed into a 2D one-hot encoded array, where the size of the first dimension equals the length of the sentence, and the size of the second equals the number of the labels.

[2] Dataset available at https://sourceforge.net/projects/tashkeela-processed/.

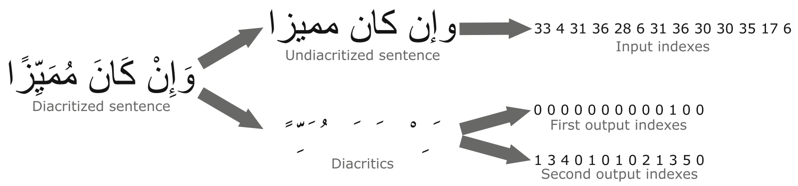
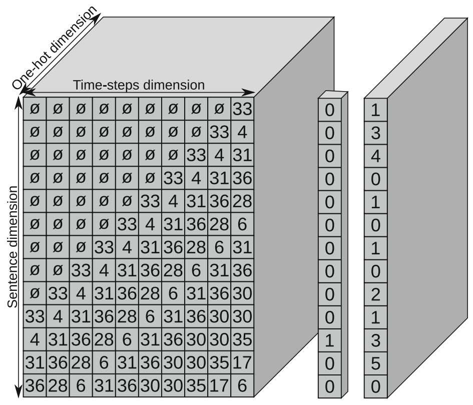


Fig. 1. Transformation of the diacritized text to the input and output labels

After that, this array is extended to 3 dimensions by inserting the time steps dimension as the second dimension and moving the dimension of the label into the third position. The time steps are generated by a sliding window of size 1 on the first dimension. The number of time steps is fixed to 10 because this number is large enough to cover most of the Arabic words, along with a part of their previous words. The output of the primary diacritics is also transformed from a vector of labels to a 2D one-hot array. The output of *Shadda* marks is left as a binary vector. Figure 2 shows a representation of the input array and the two output arrays after the preprocessing of the previous example. The ∅ represents a padding vector (all zeros), and the numbers in the input and the second output indicate the indexes of the cells of the one-hot vectors set to 1.



ø	ø	ø	ø	ø	ø	ø	ø	ø	33
ø	ø	ø	ø	ø	ø	ø	ø	33	4
ø	ø	ø	ø	ø	ø	ø	33	4	31
ø	ø	ø	ø	ø	ø	33	4	31	36
ø	ø	ø	ø	ø	33	4	31	36	28
ø	ø	ø	ø	33	4	31	36	28	6
ø	ø	ø	33	4	31	36	28	6	31
ø	ø	33	4	31	36	28	6	31	36
ø	33	4	31	36	28	6	31	36	30
33	4	31	36	28	6	31	36	30	30
4	31	36	28	6	31	36	30	30	35
31	36	28	6	31	36	30	30	35	17
36	28	6	31	36	30	30	35	17	6

0	1
0	3
0	4
0	0
0	1
0	0
0	1
0	0
0	2
0	1
1	3
0	5
0	0

Fig. 2. Input and output arrays after the transformations

4.2 Deep Learning Model

The following component in this system is an RNN model, composed from a stack of two bidirectional LSTM [22,28] layers of 64 cells in each direction, and parallel dense layers of sizes 8 and 64. All of the previous layers use *hyperbolic tangent* (Tanh) as an activation function. The first parallel layer is connected to a single perceptron having the *sigmoid* activation function, while the second is connected to 7 perceptrons having *softmax* as an activation function. The first estimates the probability that the current character has a *Shadda*, and the second generates the probabilities of the primary diacritics for that character. A schema of this network is displayed in Fig. 3. The size, type, and number of layers were determined empirically and according to the previous researches that used deep learning approaches [2,13,14,27,32].

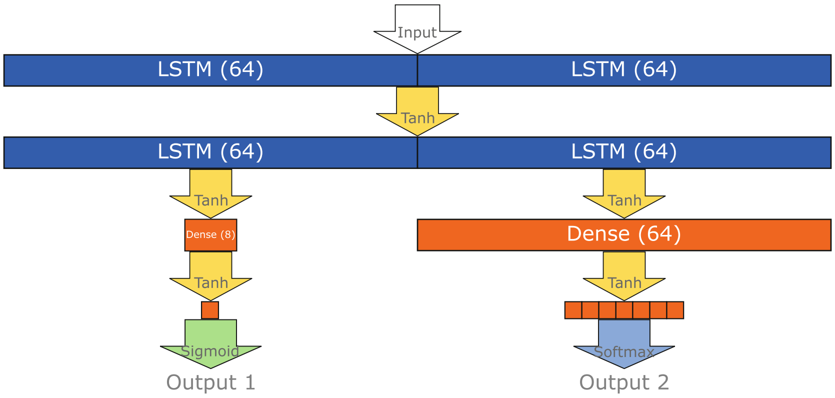


Fig. 3. Architecture of the Deep Learning model

4.3 Rule-Based Corrections

Rule-based corrections are linked to the input and output of the RNN to apply some changes to the output. These rules can select the appropriate diacritic for some characters in some contexts, or exclude the wrong choices in other contexts by nullifying their probabilities. Different sets of rules are applied to the outputs to eliminate some impossible diacritizations according to Arabic rules.

Shadda Corrections. The first output of the DL model representing the probability of the *Shadda* diacritic is ignored by nullifying its value if any of these conditions are met for the current character:

- It is a space, 0 or one of the following: ة, ا, أ, إ, آ, ء, ئ, ى.
- It is the first letter of the Arabic word.
- It has *Sukoon* as a predicted primary diacritic.

Primary Diacritics Corrections. The probabilities of the second output of the DL model are also altered by these rules when their respective conditions are met for the current character:

- If it is إ, set the current diacritic to *Kasra*, by setting the probability of its class to 1 and the others to 0.

- If it is ى or ة, set the diacritic of the previous character to *Fatha*.
- If it is ا and the last letter of the word, allow only *Fatha*, *Fathatan*, or no-diacritic choices by zeroing the probabilities of the other classes.
- If it is ا and not the last letter of the word, set *Fatha* on the previous character.
- If it is the first letter in the word, forbid *Sukoon*.
- If it is not the last character of the word, prohibit any *Tanween* diacritic from appearing on it.
- If it is the last letter, prohibit *Fathatan* unless this character is ء or ة.
- If it is a space, 0 or any of the following characters: آ, ى, ا, set the choice to no-diacritic.

4.4 Statistical Corrections

The output and the input of the previous phase are transformed and merged to generate a standard diacritized sentence. The sentence is segmented into space-delimited words and augmented by unique starting and ending entities. Every word in the sentence is checked up to 4 times in the levels of correction using the saved training data. If any acceptable correction is found at any level, the word will be corrected. Otherwise, it will be forwarded to the next level. In the case where many corrections get the same score in a single level, the first correction is chosen. If no correction is found, the predicted diacritization of the previous component is not changed.

Word Trigram Correction. In the first stage, trigrams are extracted from the undiacritized sentence and checked whether a known diacritization for its core word is available. The core word is the second one in the trigram, while the first and the third are considered previous and following contexts, respectively. If such a trigram is found, the most frequent diacritization for the core word in that context is selected. Despite its high accuracy, especially for the syntactic part, this correction rarely works since having the exact surrounding words in the test data is not common. This correction is entirely independent of the output of the DL model and the rule-based corrections. An example is shown in Fig. 4.

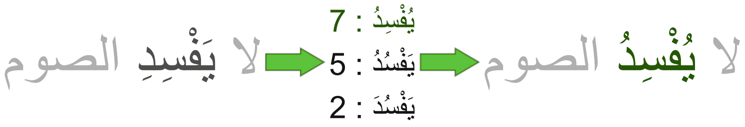


Fig. 4. Selecting the diacritization using the trigrams

Word Bigram Correction. In the second stage, the same processing as the previous one is applied for the remaining words but considering bigrams where the core word is the second one, and the first one represents the previous context. This correction works more often than the trigram-based one since it depends only on the previous word. Similarly, it does not depend on the output of the previous components.

Word Minimum Edit Distance Correction. In the third stage, when the undiacritized word has known compatible diacritizations, the Levenshtein distance [29] is calculated between the predicted diacritization and every saved diacritization for that word. The saved diacritization corresponding to the minimal edit distance is chosen, as shown in Fig. 5. Most predictions are corrected at this stage when the vocabulary of the test set is relatively similar to the training set.

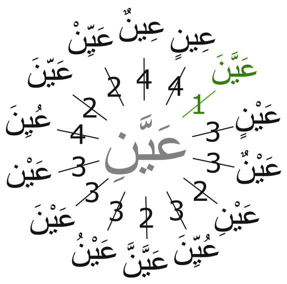


Fig. 5. Selecting the diacritization according to the minimal edit distance

Pattern Minimum Edit Distance Correction. Finally, if the word was never seen, the pattern of the predicted word is extracted and compared against the saved diacritized forms of that pattern. To generate the word pattern, the following substitutions are applied: أ, إ, ؤ, ئ, آ are all replaced by ء. ى is replaced by ا. The rest of the Arabic characters except ة and the long vowels (ا, و, ي) are substituted by the character ح. The diacritics and the other characters are not affected. The predicted diacritized pattern is compared to the saved diacritization forms of this pattern when available, and the closest one, according to the Levenshtein distance, is used as a correction, following the same idea of the previous stage. This correction is effective when the test data contains many words not seen in the training data.

5 Experiments

5.1 Implementation Details

The described architecture was developed using Python [41] 3.6 with NumPy [40] 1.16.5 and TensorFlow [1] 1.14.

The training data was transformed into NumPy arrays of input and output. The DL model was implemented using Keras, and each processed sentence of text is considered a single batch of data when fed into the DL model. The optimizer used for adjusting the model weights is ADADELTA [43] with an initial learning rate of 0.001 and ρ of 0.95.

The rule-based corrections are implemented as algebraic operations working on the arrays of the input and the output.

The statistical corrections use dictionaries as data structures, where the keys are the undiacritized n-grams/patterns, and the values are lists of the possible tuples of the diacritized form along with their frequencies in the training set.

5.2 System Evaluation

The DL model is trained for a few iterations to adjust its weights, while the dictionaries of the statistical corrections are populated while reading the training data in the first pass.

We report the accuracy of our system using the variants of the metrics DER and WER as explained in the introduction. These metrics do not have an agreed exact definition, but most of the previous works followed the definition of Zitouni et al. [46] which takes non-Arabic characters into account, while some of the new ones tend to follow the definition of Alansary et al. [7] and Fadel et al. [19] which excludes these characters. In our work, we chose the latter definition since the former can be significantly biased, as demonstrated in [19]. The calculation of these metrics should include the letters without diacritics, but they can be excluded as well, especially when the text is partially diacritized.

First, we used our testing set to measure the performances of our system. We got DER1 = 4.00%, WER1 = 12.08%, DER2 = 2.80%, and WER2 = 6.22%.

Table 4. Comparison of the performances of our system to the available baselines

System	Include no-diacritic letters				Exclude no-diacritic letters			
	DER1	WER1	DER2	WER2	DER1	WER1	DER2	WER2
Farasa	21.43%	58.88%	23.93%	53.13%	24.90%	57.28%	27.55%	51.84%
MADAMIRA	34.38%	76.58%	29.94%	59.07%	40.03%	75.39%	33.87%	57.22%
Mishkal	16.09%	39.78%	13.78%	26.42%	17.59%	35.63%	14.22%	21.92%
Tashkeela-Mode	49.96%	96.80%	52.96%	94.16%	58.50%	96.03%	60.92%	92.45%
Shakkala	3.73%	11.19%	2.88%	6.53%	4.36%	10.89%	3.33%	6.37%
Ours	**3.39%**	**9.94%**	**2.61%**	**5.83%**	**3.34%**	**7.98%**	**2.43%**	**3.98%**

The same testing data and testing method of Fadel et al. [19] were used as well in order to compare our system to the others evaluated in that work. The results are summarized in Table 4.

Results show that our system outperforms the best-reported system (Shakkala). These results can be justified as Shakkala does not perform any corrections on the output of the deep learning model, while ours includes a cascade of corrections that fix many of its errors.

When training and testing our system on the text extracted from the LDC's ATB part 3 [30], it archives DER1 = 9.32%, WER1 = 28.51%, DER2 = 6.37% and WER2 = 12.85%. Its incomplete diacritization mainly causes the higher error rates for this dataset in a lot of words, in addition to its comparatively small size, which prevents our system from generalizing well.

5.3 Error Analysis

To get a deeper understanding of the system performances, we study the effect of its different components and record the errors committed at each level. We performed the tests taking all and only Arabic characters into account on our test part of the *Tashkeela* dataset.

Contribution of the Components. In order to show the contribution in error reduction of every component, two evaluation setups were used.

Firstly, only the DL model and the static rules are enabled at first, then the following component is enabled at every step, and the values of the metrics are recalculated. Table 5a shows the obtained results.

Secondly, all the components are enabled except one at a time. The same calculations are done and displayed in Table 5b.

Table 5. Reduction of the error rates according to the enabled components

(a) Incremental enabling

Components	Metrics			
	DER1	WER1	DER2	WER2
DL + rules	23.20%	59.32%	23.75%	51.52%
+Trigrams	12.99%	32.25%	13.19%	27.46%
+Bigrams	7.09%	17.54%	6.69%	13.20%
+Unigrams	4.06%	12.18%	2.87%	6.35%
All enabled	4.00%	12.08%	2.80%	6.22%

(b) One disabled at a time

Components	Metrics			
	DER1	WER1	DER2	WER2
No trigrams	4.38%	13.39%	2.97%	6.65%
No bigrams	6.44%	19.53%	4.88%	11.46%
No unigrams	6.51%	16.57%	5.88%	11.87%
No patterns	4.06%	12.18%	2.87%	6.35%
All enabled	4.00%	12.08%	2.80%	6.22%

The contributions of the unigram and bigram corrections are the most important considering their effect on the error rates in both setups. The effect of the trigram correction is more visible on the syntactic diacritization rather than the morphological diacritization since the former is more dependant on the context.

The contribution of the pattern corrections is not very noticeable due to the position of this component in the pipeline, limiting its effect only to the OoV words.

Error Types. We use our system to generate diacritization for a subset of the sentences of our testing set. We limit our selection to 200 sentences where there is at least one word wrongly diacritized. We counted and classified a total of 426 errors manually. We present the results in Table 6.

Table 6. Diacritization errors count from 200 wrong test sentences

Error	Syntactic	Replacement	Non-existence	Prediction missing	Label missing	Total
Count	224	103	48	26	25	426

We found that 52.58% of the mistakes committed by our diacritization system are caused by the syntactic diacritization, which specify the role of the word in the sentence. The syntactic diacritization is so hard that even the Arabic native speakers often commit mistakes of this type when speaking. Since this is manual verification, we do not just suppose that the diacritic of the last character of the Arabic word is the syntactic one as what is done in the calculations of DER2 and WER2, but we select the diacritics which have a syntactic role according to Arabic rules, no matter where they appear.

A replacement error is when the system generates a diacritization that makes a valid Arabic word, but it is wrong according to the test data. 24.18% of the errors of our system are considered in this type.

Non-existence error happens when the diacritization system generates a diacritization making a word that does not exist in the standard Arabic. 11.27% of our system's errors are in this type.

The remaining error types are prediction missing and label missing, which indicate that the system has not predicted any diacritic where it should do, and the testing set has missing/wrong diacritics, respectively. These types are generally caused by the mistakes of diacritization in training and testing sets.

6 Conclusion

In this work, we developed and presented our automatic Arabic diacritization system, which follows a hybrid approach combining a deep learning model, rule-based corrections, and two types of statistical corrections. The system was trained and tested on a large part of the Tashkeela corpus after being cleaned and normalized. On our test set, the system scored DER1 = 4.00%, WER1 = 12.08%, DER2 = 2.80% and WER2 = 6.22%. These values were calculated when taking all and only Arabic words into account.

Our method establishes new state-of-the-art results in the diacritization of raw Arabic texts, mainly when the classical style is used. It performs well even on the documents that contain unseen words or non-Arabic words and symbols. We made our code publicly available as well[3].

In the next work, we will focus on improving the generalization of the system to better handle the out-of-vocabulary words, while reducing the time and memory requirements.

Acknowledgments. To Dr. Yasser Hifny for his help concerning the train and the test of our system on the diacritized text of the ATB part 3 dataset.

References

1. Abadi, M., et al.: TensorFlow: a system for large-scale machine learning. In: 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 2016), pp. 265–283 (2016)
2. Abandah, G., Arabiyat, A., et al.: Investigating hybrid approaches for Arabic text diacritization with recurrent neural networks. In: 2017 IEEE Jordan Conference on Applied Electrical Engineering and Computing Technologies (AEECT), pp. 1–6. IEEE (2017)
3. Abandah, G.A., Graves, A., Al-Shagoor, B., Arabiyat, A., Jamour, F., Al-Taee, M.: Automatic diacritization of Arabic text using recurrent neural networks. Int. J. Doc. Anal. Recogn. (IJDAR) **18**(2), 183–197 (2015)
4. Abdelali, A., Darwish, K., Durrani, N., Mubarak, H.: Farasa: a fast and furious segmenter for Arabic. In: Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations, pp. 11–16 (2016)
5. Al-Badrashiny, M., Hawwari, A., Diab, M.: A layered language model based hybrid approach to automatic full diacritization of Arabic. In: Proceedings of the Third Arabic Natural Language Processing Workshop, pp. 177–184 (2017)
6. Al Sallab, A., Rashwan, M., Raafat, H.M., Rafea, A.: Automatic Arabic diacritics restoration based on deep nets. In: Proceedings of the EMNLP 2014 Workshop on Arabic Natural Language Processing (ANLP), pp. 65–72 (2014)
7. Alansary, S.: Alserag: an automatic diacritization system for Arabic. In: Hassanien, A.E., Shaalan, K., Gaber, T., Azar, A.T., Tolba, M.F. (eds.) AISI 2016. AISC, vol. 533, pp. 182–192. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-48308-5_18
8. Alghamdi, M., Muzaffar, Z., Alhakami, H.: Automatic restoration of Arabic diacritics: a simple, purely statistical approach. Arab. J. Sci. Eng. **35**(2), 125 (2010)
9. Alosaimy, A., Atwell, E.: Diacritization of a highly cited text: a classical Arabic book as a case. In: 2018 IEEE 2nd International Workshop on Arabic and Derived Script Analysis and Recognition (ASAR), pp. 72–77. IEEE (2018)
10. Ananthakrishnan, S., Narayanan, S., Bangalore, S.: Automatic diacritization of Arabic transcripts for automatic speech recognition. In: Proceedings of the 4th International Conference on Natural Language Processing, pp. 47–54 (2005)
11. Anwar, M.: Tashkeela-model (2018). https://github.com/Anwarvic/Tashkeela-Model

[3] Available at https://github.com/Hamza5/Pipeline-diacritizer.

12. Azmi, A.M., Almajed, R.S.: A survey of automatic Arabic diacritization techniques. Nat. Lang. Eng. **21**(3), 477–495 (2015)
13. Barqawi, A., Zerrouki, T.: Shakkala, Arabic text vocalization (2017). https://github.com/Barqawiz/Shakkala
14. Belinkov, Y., Glass, J.: Arabic diacritization with recurrent neural networks. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp. 2281–2285 (2015)
15. Chennoufi, A., Mazroui, A.: Morphological, syntactic and diacritics rules for automatic diacritization of Arabic sentences. J. King Saud Univ. Comput. Inf. Sci. **29**(2), 156–163 (2017)
16. Darwish, K., Magdy, W., et al.: Arabic information retrieval. Found. Trends® Inf. Retrieval **7**(4), 239–342 (2014)
17. Darwish, K., Mubarak, H., Abdelali, A.: Arabic diacritization: stats, rules, and hacks. In: Proceedings of the Third Arabic Natural Language Processing Workshop, pp. 9–17 (2017)
18. Elshafei, M., Al-Muhtaseb, H., Alghamdi, M.: Statistical methods for automatic diacritization of Arabic text. In: The Saudi 18th National Computer Conference. Riyadh, vol. 18, pp. 301–306 (2006)
19. Fadel, A., Tuffaha, I., Al-Jawarneh, B., Al-Ayyoub, M.: Arabic text diacritization using deep neural networks. arXiv preprint arXiv:1905.01965 (2019)
20. Fashwan, A., Alansary, S.: Shakkil: an automatic diacritization system for modern standard Arabic texts. In: Proceedings of the Third Arabic Natural Language Processing Workshop, pp. 84–93 (2017)
21. Gal, Y.: An hmm approach to vowel restoration in Arabic and Hebrew. In: Proceedings of the ACL-02 Workshop on Computational Approaches to Semitic Languages, pp. 1–7. Association for Computational Linguistics (2002)
22. Graves, A., Schmidhuber, J.: Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Networks **18**(5–6), 602–610 (2005)
23. Habash, N., Rambow, O.: Arabic diacritization through full morphological tagging. In: Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers, pp. 53–56 (2007)
24. Hadj Ameur, M.S., Moulahoum, Y., Guessoum, A.: Restoration of Arabic diacritics using a multilevel statistical model. In: Amine, A., Bellatreche, L., Elberrichi, Z., Neuhold, E.J., Wrembel, R. (eds.) Computer Science and Its Applications, pp. 181–192. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19578-0_15
25. Hifny, Y.: Open vocabulary arabic diacritics restoration. IEEE Signal Process. Lett. **26**(10), 1421–1425 (2019). https://doi.org/10.1109/LSP.2019.2933721
26. Hifny, Y.: Higher order n-gram language models for Arabic diacritics restoration. In: The Twelfth Conference on Language Engineering (2012)
27. Hifny, Y.: Hybrid LSTM/MaxEnt networks for Arabic syntactic diacritics restoration. IEEE Signal Process. Lett. **25**(10), 1515–1519 (2018)
28. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
29. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions and reversals. Soviet Physics Doklady **10**, 707 (1966)
30. Maamouri, M., Bies, A., Buckwalter, T., Jin, H., Mekki, W.: Arabic treebank: Part 3 (full corpus) v 2.0 (mpg+ syntactic analysis). Linguistic Data Consortium, Philadelphia (2005)

31. Metwally, A.S., Rashwan, M.A., Atiya, A.F.: A multi-layered approach for Arabic text diacritization. In: 2016 IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA), pp. 389–393. IEEE (2016)
32. Moumen, R., Chiheb, R., Faizi, R., El Afia, A.: Arabic diacritization with gated recurrent unit. In: Proceedings of the International Conference on Learning and Optimization Algorithms: Theory and Applications, p. 37. ACM (2018)
33. Nelken, R., Shieber, S.M.: Arabic diacritization using weighted finite-state transducers. In: Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages, pp. 79–86. Association for Computational Linguistics (2005)
34. Pasha, A., et al.: MADAMIRA: a fast, comprehensive tool for morphological analysis and disambiguation of Arabic. LREC **14**, 1094–1101 (2014)
35. Rashwan, M.A., Al-Badrashiny, M.A., Attia, M., Abdou, S.M., Rafea, A.: A stochastic Arabic diacritizer based on a hybrid of factorized and unfactorized textual features. IEEE Trans. Audio Speech Lang. Process. **19**(1), 166–175 (2011)
36. Rashwan, M.A., Al Sallab, A.A., Raafat, H.M., Rafea, A.: Deep learning framework with confused sub-set resolution architecture for automatic Arabic diacritization. IEEE/ACM Trans. Audio Speech Lang. Process. (TASLP) **23**(3), 505–516 (2015)
37. Said, A., El-Sharqwi, M., Chalabi, A., Kamal, E.: A hybrid approach for Arabic diacritization. In: Métais, E., Meziane, F., Saraee, M., Sugumaran, V., Vadera, S. (eds.) NLDB 2013. LNCS, vol. 7934, pp. 53–64. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38824-8_5
38. Shaalan, K., Abo Bakr, H.M., Ziedan, I.: A hybrid approach for building Arabic diacritizer. In: Proceedings of the EACL 2009 Workshop on Computational Approaches to Semitic Languages, pp. 27–35. Association for Computational Linguistics (2009)
39. Shahrour, A., Khalifa, S., Habash, N.: Improving Arabic diacritization through syntactic analysis. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp. 1309–1315 (2015)
40. Van Der Walt, S., Colbert, S.C., Varoquaux, G.: The numpy array: a structure for efficient numerical computation. Comput. Sci. Eng. **13**(2), 22 (2011)
41. Van Rossum, G., Drake, F.L.: The Python Language Reference Manual. Network Theory Ltd., Network (2011)
42. Zayyan, A.A., Elmahdy, M., binti Husni, H., Al Ja'am, J.M.: Automatic diacritics restoration for modern standard Arabic text. In: 2016 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE), pp. 221–225. IEEE (2016)
43. Zeiler, M.D.: Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701 (2012)
44. Zerrouki, T.: Mishkal, Arabic text vocalization software (2014). https://github.com/linuxscout/mishkal
45. Zerrouki, T., Balla, A.: Tashkeela: novel corpus of Arabic vocalized texts, data for auto-diacritization systems. Data Brief **11**, 147 (2017)
46. Zitouni, I., Sorensen, J.S., Sarikaya, R.: Maximum entropy based restoration of Arabic diacritics. In: Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics, pp. 577–584. Association for Computational Linguistics (2006)