IJDAR manuscript No.
(will be inserted by the editor)

Automatic diacritization of Arabic text using recurrent neural networks

Gheith A. Abandah · Alex Graves · Balkees Al-Shagoor · Alaa Arabiyat · Fuad Jamour · Majid Al-Taee

Received: February 17, 2014 / Revised: December 13, 2014 / Accepted: February 16, 2015

Abstract This paper presents a sequence transcription approach for the automatic diacritization of Arabic text. A recurrent neural network (RNN) is trained to transcribe undiacritized Arabic text with fully diacritized sentences. We use a deep bidirectional long short-term memory (LSTM) network that builds high-level linguistic abstractions of text and exploits long-range context in both input directions. This approach differs from previous approaches in that no lexical, morphological, or syntactical analysis is performed on the data before being processed by the net. Nonetheless, when the network is post-processed with our error-correction techniques, it achieves state of the art performance, yielding an average diacritic and word error rates of 2.09% and 5.82% respectively on samples from 11 books. For the LDC ATB3 benchmark, this approach reduces the diacritic error rate by 25%, the word error rate by 20%, and the last letter diacritization error rate by 33% over the best published results.

Keywords Automatic diacritization · Arabic text · machine learning · sequence transcription · recurrent neural networks · deep neural networks · long short-term memory

G. Abandah, B. Shagoor, A. Arabiyat, M. Al-Taee
Computer Engineering Department,
The University of Jordan, Amman, 11942, Jordan
Tel.: +962-6-5355000
Fax: +962-6-5300813
E-mail: abandah@ju.edu.jo, b.alshagoor@ju.edu.jo, a.arabiyat@ju.edu.jo, altaeem@ju.edu.jo

A. Graves
Google DeepMind, London
E-mail: alex.graves@gmail.com

F. Jamour
King Abdullah University of Science and Technology
E-mail: fjamour@gmail.com

1 Introduction

The Arabic alphabet is the base alphabet used in around 27 languages, including Arabic, Persian, Kurdish, Urdu, and Jawi [26]. The Arabic language has 28 basic letters and eight basic diacritics that are encoded in the Unicode code block 0600–06FF. Figure 1 shows where these letters and diacritics are encoded in this code block. Note that this code block includes 36 variants of the 28 letters having hexadecimal codes 0621–063A and 0641–064A; the diacritics have codes 064B–0652 [1].

The Modern Standard Arabic (MSA) is commonly written without diacritics as the example shown in Fig. 2(a) (note that Arabic is written from right-to-left). Classical Arabic (CA), on the other hand, is written with diacritics as in Fig. 2(b). The Holy Quran and classical books are usually diacritized to specify the accurate pronunciation and the intended meaning.

Although the MSA writing is faster, undiacritized words often cannot be pronounced correctly from their orthographic representation only. Educated native readers can generally infer the correct pronunciation of undiacritized words from the context, and from their knowledge of the grammar and lexicon. However, the lack of diacritics causes ambiguity for children and non-native speakers, who have not mastered the language's rich derivation and inflection mechanisms.

In Arabic, many stem words can be derived from the finite Arabic *root* consonant combinations into known *patterns* [32]. Different words, with different meanings and pronunciations, often differ only by their diacritics [7].

For example, the first word in the previous example كتب is the verb "wrote" and is pronounced "kataba".

	U+062x	U+063x	U+064x	U+065x
0		ذ		ـَ
1	ء	ر	ف	ـّ
2	آ	ز	ق	ـْ
3	أ	س	ك	
4	ؤ	ش	ل	
5	إ	ص	م	
6	ئ	ض	ن	
7	ا	ط	ه	
8	ب	ظ	و	
9	ة	ع	ى	
A	ت	غ	ي	
B	ث		ً	
C	ج		ٌ	
D	ح		ٍ	
E	خ		َ	
F	د		ُ	

Fig. 1 Unicode Arabic code block, showing the 36 Arabic letter variants and eight diacritics.

(a) كتب الطالب رسالة

(b) كَتَبَ الطَّالِبُ رِسَالَةً

Fig. 2 The sentence "The student wrote a letter" in (a) Arabic without diacritics, (b) Arabic with diacritics

The same three letters can mean "books" and are pronounced "kutub" كُتُب. Nevertheless, the native reader would easily infer that the first word is the verb "kataba" كَتَبَ. As the second and third words are the nouns "the student" الطالب and "letter" رسالة, respectively, the reader would likely assume that this is a verb-subject-object sentence.

Inflection changes are phonological changes a word undergoes as it is being used in context. In Arabic, there are rules to inflect a stem word according to its tense/aspect, person, voice, mood, gender, number, case, and definiteness. In many cases, different inflections

كتب أحمد وعلي وعمر ------

(a) كُتُبُ أَحْمَدَ وَعَلِيٍ وَعُمَرَ كَثِيرَةٌ

(b) كَتَبَ أَحْمَدُ وَعَلِيٌ وَعُمَرُ الدَّرْسَ

Fig. 3 The four words in top are diacritized as shown in (a) or (b) depending of the last word of the sentence.

differ only by the diacritic of the last letter [22]. For example, the verb "kataba" كَتَبَ is inflected for masculine second person as "katabta" كَتَبْتَ (with Fatha diacritic at the end) and is inflected for feminine second person as "katabti" كَتَبْتِ (with Kasra). Another example: the noun رِسَالَة is inflected as "risalatun" رِسَالَةٌ (with Dammatan) in the subject case and as "risalatan" رِسَالَةً (with Fathatan) in the object case.

To correctly diacritize the words of a sentence, it is often required to analyze the entire sentence. Figure 3 shows two different diacritizations of four words based on the fifth (last) word. The four words are diacritized as shown in (a)"Books of Ahmad and Ali and Omar are numerous" when followed by the adjective "numerous" كثيرة, or as shown in (b) "Ahmad and Ali and Omar wrote the lesson" when followed by the noun "the lesson" الدرس. As the last word in (a) is an adjective, the word كتب must be the noun "books" كُتُبُ. Whereas the last word in (b) is a noun; so it is more likely that the word كتب is the verb "wrote" كَتَبَ.

This example illustrates that it is not possible to automatically diacritize a word based only on its past context. The successor words must often be considered to reach correct analysis and diacritization. Moreover, it is often necessary to do-long range context analysis. For example, checking only one, two, or three successor words of the word كتب is not sufficient in this example.

Adding diacritics to undiacritized or partially diacritized text is very helpful for children and non-native

speakers who are trying to learn Arabic. It is a necessary step in text-to-speech (TTS) software, as otherwise pronunciation is ambiguous, and is also useful for training automatic speech recognition (ASR) engines [25]. Moreover, indexing diacritized text instead of undiacritized text enables search engines to better exclude unwanted matches.

The best methods to date for diacritization generally analyze the input text and derive extensive morphological and contextual information, then use pattern classification techniques to select the analysis that best matches the context.

This paper proposes a novel approach, where diacritization is solved as a sequence transcription problem. We use a recurrent neural network (RNN) to transcribe raw Arabic sentences, without relying on any prior morphological or contextual analysis, into fully diacritized sentences. We use a deep bidirectional long short-term memory (LSTM) network that builds high-level linguistic abstractions of text and exploits long-range context in both input directions. We describe how Arabic sentences are encoded into sequences suitable for sequence transcription by an RNN. Moreover, we describe some post-processing corrections developed to improve the results of the sequence transcription stage. This approach, which builds on recent advances in sequence transcription using RNNs [17], yields higher accuracy than the best published results.

The problem that our approach solves is adding diacritics to undiacritized sentences. Given an input sequence $\boldsymbol{x} = (x_1, \ldots, x_T)$ that represents an Arabic sentence of T unicode letters of the set $\mathcal{U}$ in the range 0621–064A and $\boldsymbol{x} \in \mathcal{U}^T$, transcribe this sequence into the output sequence $\boldsymbol{y} = (y_1, \ldots, y_U)$ that represents the diacritized sequence and consists of U unicode letters and diacritics of the set $\mathcal{D}$ in the range 0621–0652 and $\boldsymbol{y} \in \mathcal{D}^U$. Generally, adding diacritics increases the sequence length and $U \geq T$.

The following section reviews the related work, Section 3 describes the RNN used in this work for sequence transcription, Section 4 describes our experimental setup, Section 5 describes the experiments conducted and presents their results, Section 6 discusses the results of this approach and compares its results with other leading work, and finally Section 7 presents the conclusions and suggestions for future work.

2 Literature Review

Past approaches to automatic diacritization of Arabic text can be roughly divided into rule-based, statistical and hybrid methods [5]. The earliest approaches were *rule-based* [11,3], and relied on morphological analyzers, dictionaries, and grammar modules. The major drawbacks of rule-based diacritization are the high development cost and the reliance on parsed corpora that are difficult to create. Additionally, the rules must be continuously maintained as new words and terms are generated in living languages.

More recently, there have been several *statistical*, machine-learning approaches. Gal used hidden Markov models (HMMs) to capture the contextual correlation between words [12]. His approach restores only short vowels (a subset of all diacritics).

Hifny used statistical n-gram language modeling of a large corpus [23]. He used the Tashkila diacritized Arabic text corpus described in Section 4.1. The possible diacritized word sequences of an undiacritized input sentence are assigned probability scores using the n-gram models. Then using a dynamic programming algorithm, the most likely sequence is found. Smoothing techniques were used to handle unseen n-grams in the training data. The accuracy of n-gram models depends on the order n. Larger order gives higher accuracy as it incorporate longer linguistic dependencies. However, larger order results in larger models and requires larger training data. Hifny used n-gram models of order three (*trigram*), thus, does not exploit long-range context dependencies.

To best of our knowledge, the most accurate system reported also uses a statistical approach and is due to Azim *et al.* [4]. However, this system requires the availability of speech input as it combines acoustic information from the speech input to complement text-based conditional random fields model.

Most current work in the area relies on *hybrid* approaches that combine rule-based and statistical modules. Vergyri and Kirchhoff investigated the effect of combining several knowledge sources (acoustic, morphological, and contextual) to automatically diacritize Arabic text [35]. They treated diacritization as an unsupervised tagging problem where each word is tagged as one of the many possible forms provided by the Buckwalter Arabic morphological analyzer (BAMA) [8]. They also investigated the use of Arabic dialectal speech. In this study, they used two different corpora: the Foreign Broadcast Information Service (FBIS) corpus of MSA speech and the LDC CallHome Egyptian Colloquial Arabic (ECA) corpus. However, they did not model the Shadda diacritic.

Nelken *et al.* proposed a weighted finite state machine algorithm to restore the missing diacritics [30]. Their basic module consists of a standard trigram language model that chooses the most probable diacritized word sequence that could have generated the undiacritized text. They also used several other transducers

to improve the diacritization process. This system was trained and tested on LDC's Arabic Treebank of diacritized news stories (Part 2).

Zitouni *et al.* followed a statistical model based on the framework of maximum entropy where several sources of information were used, including lexical, segment-based, and part-of-speech (POS) features [38]. Their system was trained and evaluated on Linguistic Data Consortium's Part 3 of the Arabic Treebank of diacritized news stories (LDC ATB3) [27]. They described their experimental setup in great detail, and the LDC ATB3 subsequently became a benchmark in the area. They reported that the hybrid approach has better diacritization accuracy compared with the pure statistical approach. Their maximum entropy model makes a decision for each state independent of other states. Therefore, it does not exploit context information to achieve high diacritization accuracy.

Habash and Rambow extended the use of their morphological analysis and disambiguation of Arabic system (MADA) for diacritization [22]. MADA consults BAMA to get a list of possible diacritized analyses for a word. To narrow this list to a small number, fourteen SVM predictors are used to predict morphological features of the possible analyses and to rank them. Finally, one solution is selected from the narrowed list using n-gram language models. The n-gram models used don't exploit long-range context dependencies as they are limited to three words.

The stochastic Arabic diacritizer by Rashwan *et al.* is a recent approach with excellent results [31]. The system uses two stochastic layers, the first of which predicts the most likely diacritics by choosing the sequence of unfactorized full-form Arabic word diacritizations with maximum marginal probability via A* lattice search algorithm and n-gram probability estimation. When full-form words are not found, the system falls back on the second layer, which factorizes each Arabic word into its possible morphological components (prefix, root, pattern and suffix), then uses n-gram probability estimation and A* lattice search algorithm to select among the possible factorizations to get the most likely diacritization sequence. Similar to Habash and Rambow's approach, this approach is limited by relying on trigram language models to select most probable diacritization options. However, it achieves better results through its elaborate factorization and dictionary techniques.

Said *et al.* developed a hybrid system that achieves the best results prior to this paper on LDC ATB3 without speech input [33]. The system relies on automatic correction, morphological analysis, POS tagging, and out of vocabulary diacritization. This system is similar to Rashwan *et al.*'s system but uses HMMs for morphological analyses disambiguation and resolving the syntactic ambiguity to restore the syntactic diacritic. The HMM estimates the most probable tag sequence of the alternative POS tag sequences uring Viterbi algorithm. However, this estimation is based on two local probabilities: the word likelihoods and adjacent word transition probabilities.

Bahanshal and Al-Khalifa evaluated the accuracy of three available diacritization systems using fully diacritized text from the Quran and short poems [6]. This study demonstrates that the available systems' accuracy is still not sufficient and there is still ample room for improvement.

Our pure statistical approach of sequence transcription is based on the deep bidirectional LSTM architecture [24]. This architecture has been successfully applied to many sequence transcription tasks, including automatic speech and handwriting recognition [20,15], which is currently state-of-the-art in offline Arabic handwriting recognition [28,2]. To best of our knowledge, this work is the first to use RNN sequence transcription to automatically add diacritics to Arabic text. Our experiments were carried out with the open-source software library RNNLIB [19].

3 Sequence Transcription

The basic recurrent neural network (RNN) used in this work is deep bidirectional LSTM [20]. This architecture combines long short-term memory (LSTM) [24] with bidirectional RNNs [34] and the stacked hidden layers seen in deep feedforward neural networks.

Deep LSTM networks have been previously applied to character-level text prediction, and have proven capable of modeling long-range linguistic dependencies [18].

Given an input sequence $\boldsymbol{x} = (x_1, \ldots, x_T)$, a standard RNN computes the hidden vector sequence $\boldsymbol{h} = (h_1, \ldots, h_T)$ and output vector sequence $\boldsymbol{y} = (y_1, \ldots, y_T)$ by iterating the following equations from $t = 1$ to T:

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

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

where the W terms denote weight matrices (*e.g.*, W_{ih} is the input-hidden weight matrix), the b terms denote bias vectors (*e.g.*, b_h is hidden bias vector), and $\mathcal{H}$ is the hidden layer activation function. Note the *recurrence* in Eq. 1 where h_t depends on the previous vector h_{t-1}.

3.1 Long short-term memory

$\mathcal{H}$ is usually an elementwise application of a sigmoid function. However, the long short-term memory (LSTM)

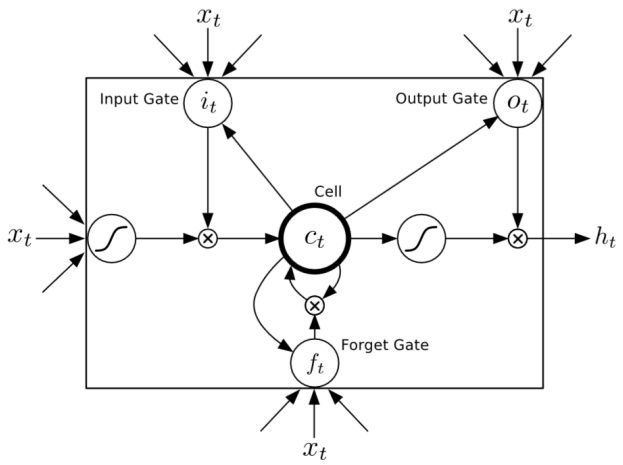

Fig. 4 Long short-term memory cell

architecture [24], which uses purpose-built *memory cells* to store information, is better at finding and exploiting long range context. Figure 4 illustrates a single LSTM memory cell. For the version of LSTM used in this paper [13], $\mathcal{H}$ is implemented by the following composite function:

$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right) \quad (3)$$
$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right) \quad (4)$$
$$c_t = f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right) \quad (5)$$
$$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right) \quad (6)$$
$$h_t = o_t \tanh(c_t) \quad (7)$$

where σ is the logistic sigmoid function, and i, f, c, and o are respectively the *input gate*, *forget gate*, *cell* activation, and *output gate* vectors, all of which are the same size as the hidden vector $\boldsymbol{h}$. The weight matrix subscripts have the obvious meaning, for example, W_{hi} is the hidden-input gate matrix, W_{xo} is the input-output gate matrix, *etc.* The weight matrices from the cell to gate vectors (e.g. W_{ci}) are diagonal, so element m in each gate vector only receives input from element m of the cell vector. The bias terms (which are added to i, f, c and o) have been omitted for clarity.

3.2 Bidirectional RNNs

One shortcoming of conventional RNNs is that they are only able to make use of previous context. In automatic diacritization, where whole sentences are transcribed at once, there is no reason not to exploit future context as well. Bidirectional RNNs (BRNNs) [34] do this by processing the data in both directions with two separate hidden layers, which are then fed forwards to the same output layer. A BRNN computes the *forward* hidden sequence $\overrightarrow{\boldsymbol{h}}$, the *backward* hidden sequence $\overleftarrow{\boldsymbol{h}}$, and the output sequence $\boldsymbol{y}$ by iterating the backward layer from $t = T$ to 1, the forward layer from $t = 1$ to T, and then updating the output layer:

$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}}x_t + W_{\overrightarrow{h}\overrightarrow{h}}\overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right) \quad (8)$$
$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}}x_t + W_{\overleftarrow{h}\overleftarrow{h}}\overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right) \quad (9)$$
$$y_t = W_{\overrightarrow{h}y}\overrightarrow{h}_t + W_{\overleftarrow{h}y}\overleftarrow{h}_t + b_o \quad (10)$$

Combing BRNNs with LSTM, gives bidirectional LSTM, which can access long-range context in both input directions [21].

3.3 Deep recurrent neural network

A crucial element of the recent success of hybrid systems is the use of *deep* architectures, which are able to build up progressively higher level representations of text. *Deep RNNs* can be created by stacking multiple RNN hidden layers on top of each other, with the output sequence of one layer forming the input sequence for the next. Assuming the same hidden layer function is used for all N layers in the stack, the hidden vector sequences $\boldsymbol{h}^n$ are iteratively computed from $n = 1$ to N and $t = 1$ to T:

$$h_t^n = \mathcal{H}\left(W_{h^{n-1}h^n}h_t^{n-1} + W_{h^n h^n}h_{t-1}^n + b_h^n\right) \quad (11)$$

where $\boldsymbol{h}^0 = \boldsymbol{x}$. The network outputs y_t are

$$y_t = W_{h^N y}h_t^N + b_o \quad (12)$$

Deep bidirectional RNNs can be implemented by replacing each hidden sequence $\boldsymbol{h}^n$ with the forward and backward sequences $\overrightarrow{\boldsymbol{h}}^n$ and $\overleftarrow{\boldsymbol{h}}^n$, and ensuring that every hidden layer receives input from both the forward and backward layers at the level below. If LSTM is used for the hidden layers, the complete architecture is referred to as deep bidirectional LSTM [20].

Two different methods were used to train the RNN to transcribe sequences of undiacritized Arabic letters with their diacritized counterparts. These two methods are described in the following two subsections.

3.4 One-to-one network

In the first method, the "one-to-one" letter encoding described in Section 4.2 was used, ensuring that the target sequences had a one-to-one correspondence with the inputs sequences. Thus, the lengths of sequences $\boldsymbol{x}$ and $\boldsymbol{y}$ are equal as implied by Eqs. 1 and 2 above.

The network was trained to individually classify each input letter with the corresponding diacritized version. As is standard for classification tasks, a softmax output layer was used to define a probability distribution over the output labels, and the network was trained to minimize the cross-entropy of this distribution with the target labels. That is, given a length T input, target sequence pair $(\boldsymbol{x}, \boldsymbol{y}^*)$, the network outputs at time t are interpreted as the probability $\Pr(k|t, \boldsymbol{x})$ of emitting (diacritized) letter k at time t and the loss function minimized by the network is defined as $\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}^*) = -\sum_{t=1}^{T} \log \Pr(\boldsymbol{y}_t^*|t, \boldsymbol{x})$.

The network is trained to minimize the loss function $\mathcal{L}$ using online gradient descent algorithm with momentum. A similar approach was previously used for bidirectional LSTM networks applied to framewise phoneme classification [21]—the main difference being that the networks in this work had more than one hidden layer. From now on, we will refer to this configuration as the "one-to-one" network.

3.5 One-to-many network

The second network transcribes undiacritized input encoded in Unicode to diacritized output using the "one-to-many" encoding described in Section 4.2. In this encoding the output sequence is usually longer than the input sequence ($U \geq T$).

The second training method used an additional, single-layer LSTM network to predict the "one-to-many" output symbols. This approach was based on the "sequence transducer" recently applied to phoneme recognition [16]; again the main modification in the current work was that the bidirectional network contained multiple hidden layers.

The sequence transducer is able to emit zero, one or many output labels for every input label, making it possible to train with input and target sequences of different lengths. Furthermore, the transducer automatically learns to align the two sequences, so there is no need to pre-define which diacritic marks correspond to which letters. We will refer to this as the "one-to-many" network.

The sequence transducer that aligns the input and target sequences consists of two separate RNNs—the *input* network, which is typically bidirectional, and the *prediction* network, which must be uni-directional—along with a feedforward *output* network used to combine the outputs of the two RNNs. The two networks are used to determine a separate distribution $\Pr(k|t, u)$ for every *combination* of input timestep t and output timestep u. Each distribution covers the K possible labels in the task plus a b. Intuitively, the network chooses what to output depending both on where it is in the input sequence and the outputs it has already emitted. For a length U target sequence $\boldsymbol{y}^*$, the complete set of TU decisions jointly determines a distribution over all possible alignments between $\boldsymbol{x}$ and $\boldsymbol{y}^*$, from which $\log \Pr(\boldsymbol{y}^*|\boldsymbol{x})$ can be efficiently determined with a forward-backward algorithm [16].

Denote by $\overrightarrow{\boldsymbol{h}}^N$ and $\overleftarrow{\boldsymbol{h}}^N$ the uppermost forward and backward hidden sequences of the input network, and by $\boldsymbol{p}$ the hidden sequence of the prediction network. At each t, u the output network is implemented by feeding $\overrightarrow{\boldsymbol{h}}^N$ and $\overleftarrow{\boldsymbol{h}}^N$ to a linear layer to generate the vector l_t, then feeding l_t and p_u to a tanh hidden layer to yield $h_{t,u}$, and finally feeding $h_{t,u}$ to a size $K+1$ softmax layer to determine $\Pr(k|t, u)$:

$$l_t = W_{\overrightarrow{h}^N l} \overrightarrow{h}_t^N + W_{\overleftarrow{h}^N l} \overleftarrow{h}_t^N + b_l \tag{13}$$

$$h_{t,u} = \tanh\left(W_{lh} l_{t,u} + W_{pb} p_u + b_h\right) \tag{14}$$

$$y_{t,u} = W_{hy} h_{t,u} + b_y \tag{15}$$

$$\Pr(k|t, u) = \frac{\exp(y_{t,u}[k])}{\sum_{k'=1}^{K} \exp(y_{t,u}[k'])}, \tag{16}$$

where $y_{t,u}[k]$ is the k^{th} element of the length $K+1$ unnormalized output vector. For simplicity we constrained all non-output layers to be the same size ($|\overrightarrow{h}_t^n| = |\overleftarrow{h}_t^n| = |p_u| = |l_t| = |h_{t,u}|$); however they could be varied independently.

More detail about this network and its training is in Ref. [16].

4 Experimental Setup

Figure 5 summarizes our experimental setup. For evaluation purposes, the diacritized Arabic sentence is first encoded in a record suitable for RNN sequence transcription. This record consists of the diacritized target sequence and the undiacritized input sequence. The RNN transcribes the input with fully diacritized output sequence. We apply post-processing corrections to overcome some transcription errors. Finally, the corrected output is compared with the target to find the diacritization accuracy.

During training the RNN, both the input and the target sequences are presented to the net. For diacritizing an undiacritized sentence, the target field is not available and, consequently, no comparison is made. The following subsections describe this method in more detail.

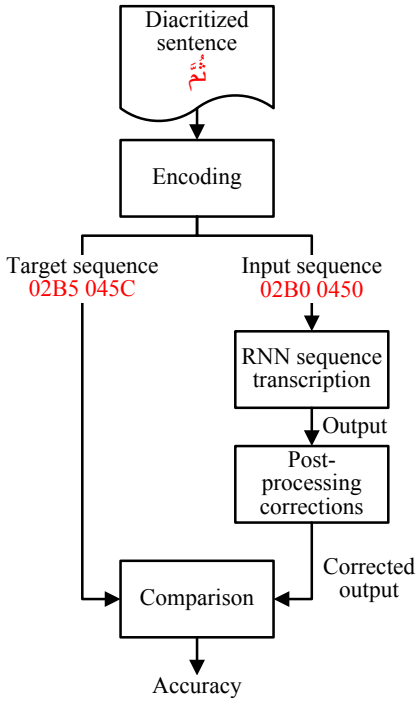

Fig. 5 Experimental setup

4.1 Data

Our experimental data were drawn from ten books of the Tashkila collection of Islamic religious heritage books [37], along with the simple version of the Holy Quran [36]. These 11 books, summarized in Table 1, are written in classical Arabic with full diacritization provided in HTML format. However, Book 1 is partially diacritized and is selected to study the effect of using partially diacritized input on the diacritization accuracy.

We also used LDC's Arabic Treebank (#LDC2010T08) of diacritized news stories, Part 3, v3.2 [27]. The LDC ATB3 is an example of the modern standard Arabic and consists of 599 distinct newswire stories from the Lebanese publication An-Nahar that were published in 2002. This treebank is used in this work in the final experiments to facilitate comparisons with previous systems.

Some of these classical books are large, multi-volume texts that take a long time to process, *e.g.*, Book 11 is 1,306-K words long (punctuation marks are not counted). As the table indicates, we randomly selected subsets of each book's sentences in our experiments. To have varying data set sizes, we selected about 3% of Book 1 on one end and 100% of Book 4 and the ATB3 on the second end.

These books have wide range of sizes and sentence length styles. They widely differ in the utilization of the punctuation marks. However, they share similarities, particular to the Arabic language, such as the average number of letters per word, and the average percentages of letters with no, one, and two diacritics. Book 1 and the ATB3 have higher values of the first percentage because they are partially diacritized.

The Tashkila books are encoded in Code Page 1256 Windows Arabic and are organized in paragraphs that each contains one or more sentence. We prepare them for training and testing purposes by first converting them from HTML to plain text files that have one sentence per line. A paragraph is split into more than one sentence when it contains sentence-ending punctuation marks such as '.', '،', '؛', ':', '؟', '≪', and '≫'. However, the punctuation marks used for interjection do not break the sentence, *e.g.*, '(' and ')'.

Note that the simple version of the Quran comes in a UTF-8 encoded text file. This file does not have punctuation marks and every Quranic verse (sentence) is in a separate line.

We extracted the diacritized version of the ATB3 sentences from the ATB3 integrated format, every sentence in a separate line. The diacritized words in this format are available in the unsplit vocalization field which is encoded in Buckwalter Arabic transliteration [8]. As some words in the ATB3 are not available in the diacritized version, we use the source undiacritized version instead for these words.

All data lines are then converted to Unicode encoding. Each line has a diacritized Arabic sentence.

4.2 Data encoding

This sections described how the sentences are encoded for sequence transcription. These sentences are converted to sentence records that has each sentence in two versions: (*i*) input sequence, without diacritics, and (*ii*) target sequence, with diacritics, comma separated. The non-diacritized version is generated from the original sentence after removing all diacritics.

Diacritics are represented in Unicode as additional characters. For example, the word "thumma" ثُمَّ has the two-field record "ثم", "ثُمَّ", and is encoded as "062B 0645", "062B 064F 0645 0651 064E". Therefore the diacritized target sequences are in general longer than

Table 1 Summary of the samples used in this study including the ID, name, size, used size, average letters per word, average words per sentence, and percentages of letters without diacritics, with one diacritic, and with two diacritics

ID	Name	Size (K Words)	Used (K Words)	Letters per word	Words per sentence	No diacritics	One diacritic	Two diacritics
1	Alahaad Walmathany	241	8	3.78	6.01	43.1%	52.6%	4.3%
2	Majma Aldamanat	218	53	4.04	14.25	21.1%	74.6%	4.3%
3	Sahyh Muslim	494	68	3.81	21.01	26.4%	67.8%	5.8%
4	The Holy Quran	76	76	4.25	12.48	22.1%	72.2%	5.7%
5	Alqwaed Labn Rajab	169	81	4.12	16.20	20.9%	74.2%	4.9%
6	Alzawajer An Aqtiraf Alkabaer	284	150	3.94	9.57	21.6%	72.3%	6.1%
7	Ghidaa Alalbab	316	189	3.99	9.28	21.9%	72.2%	5.9%
8	Aljwahrah Alnyyrah	385	205	3.99	22.77	20.7%	74.1%	5.2%
9	Almadkhal Lilabdary	361	236	4.05	13.68	21.1%	73.1%	5.8%
10	Durar Alhokam	674	407	3.83	24.22	21.5%	73.2%	5.3%
11	Moghny Almohtaj	1,306	794	3.93	9.63	20.5%	73.9%	5.6%
12	LDC ATB3	305	305	4.64	11.31	39.8%	54.8%	5.4%
	Average	402	214	4.03	14.20	25.1%	69.6%	5.4%

the non-diacritized input sequences. This "one-to-many" encoding is used with the one-to-many network.

For the one-to-one network, we encode every possible diacritization of every letter with a single symbol. Thus, the input and target sequences have same length. We use the following formula to obtain a unique code L for the letter with Unicode value l and possible diacritics d_1 and d_2:

$$L = \begin{cases} (l \wedge \text{0x00ff} \ll 4) & \text{no diacritics} \\ (l \wedge \text{0x00ff} \ll 4) \vee d_1 & \text{one diacritic} \\ (l \wedge \text{0x00ff} \ll 4) \vee d_1 \vee d_2 & \text{two diacritics} \end{cases} \quad (17)$$

The most significant eight bits of the letter's Unicode code are cleared, the masked code is shifted four bit positions to the left, then the shifted code is ORed with the bit code(s) of its diacritics d_1 and d_2 that are shown in Table 2, if any. The previous example is therefore encoded as "02B0 0450", "02B5 045C" where each input code is mapped to one target code. Note that an Arabic letter cannot have more than two diacritics, and if it has two diacritics, then one of them must be Shadda (bit code 1000).

4.3 Training parameters

The RNNs are generally trained in this work using 88% of the available sentences. The remaining 12% are randomly selected from the available data and are used for testing purposes. For the ATB3, we use the same split of data between training and testing as in previous work [38,22,31,33]. The first, in chronological order, 509 newswire stories are used for training and the last 90 newswire stories are used in testing.

Table 2 The main eight Arabic diacritics along with their shapes on the Teh Marbuta (ة) letter, sounds, hexadecimal Unicode codes, and binary bit codes.

Name	Shape	Sound	Unicode	Bit code
Fathatan	ةً	an	064B	0001
Dammatan	ةٌ	un	064C	0010
Kasratan	ةٍ	in	064D	0011
Fatha	ةَ	a	064E	0100
Damma	ةُ	u	064F	0101
Kasra	ةِ	i	0650	0110
Sukun	ةْ	None	0652	0111
Shadda	ةّ	Doubling	0651	1000

We follow previous researchers in having a single set for development and testing, rather than separate development and test sets (as is common). Zitouni *et al.* have proposed this split when they started the LDC ATB3 benchmark [38] and this split is reluctantly followed by all following researchers [22]. This adoption allows us to compare our results to theirs.

However, using the test set as the validation set in training a neural network would generally give a net that is optimized for the test set. Therefore, we also experimented with using separate development and test sets. In the separate case, the training sentences described above are randomly split into 70% training set and 30% validation test. The resulting net is then used to find the diacritization accuracy on the held-aside test set.

All networks were trained with online gradient descent (weight updates after every sequence) using a learning rate of 10^{-3} and a momentum of 0.9 and random initial weights drawn uniformly from $[-0.1, 0.1]$. The training algorithm uses an error gradient calculated with a combination of *Real Time Recurrent Learning* (RTRL) and *Back Propagation Through Time* (BPTT) [21,16]. The one-to-one network was stopped at the point of lowest label error rate on the validation set. The one-to-many network was stopped at the point of lowest log-loss.

In order to train the RNN to achieve high accuracy, we have experimented with several training options. These options include transcription method, net parameters and size, and size of the training data. The results are presented in Section 5.

4.4 Post-processing

We use the following post-processing techniques to improve the output of the sequence transcription stage.

- *Letter correction*: As the input letter sequences should not change and the main objective is to add their diacritics, any letter error in the output sequence is corrected by consulting the input sequence. For example, if for the input word كتب, we get the output "kababa" كَبَبَ which has the second letter miss-transcribed. Then this letter is corrected to get "kataba" كَتَبَ. Letter miss-transcriptions are rare; they occur when the training set is too small. However, although correcting them is necessary, this correction does not improve (or worsen) the diacritization accuracy.
- *Sukun correction*: The Sukun diacritic is used as an indication that the letter does not hold one of the three main short vowels (Fatha, Damma, and Kasra). Some writing styles use the Sukun diacritic to mark un-vowelized letters and some styles leave such letters without any diacritic. To overcome these differences when counting diacritization errors, we remove the Sukun from the output and target sequences. For example, the output "attalibu" الْطَّالِبُ is corrected to "attalibu" الطَّالِبُ.
- *Fatha correction*: The letter that precedes the letters Alef (ا), Alef Maksura (ى), or Teh Marbuta (ة) always has the short vowel Fatha (or Shadda and Fatha). If such a letter in the output sequence has a short vowel other than Fatha, it is corrected to Fatha. For example, the output "attualibu" الطُّالِبُ is corrected to "attalibu" الطَّالِبُ.
- *Dictionary correction*: A dictionary is consulted to check whether the output word is in this dictionary. This dictionary is built from the training data and is indexed by the non-diacritized version of the word. For every training word, the dictionary holds all its diacritized variants. Each output word is checked whether it is in the dictionary or not using its non-diacritized version. If the word is present and does not match any of the dictionary diacritized variants, the variant that has the smallest edit distance is selected as a correction [10] for the output word. If the word is not in the dictionary or does match one of the dictionary variants, the word is not changed. However, we have noted that when the only difference between the output word and a dictionary word is in the last letter's diacritic, it is better not to correct the output word.

4.5 Accuracy evaluation

The output of the post-processing stage is analyzed for accuracy against the target sequence using the following two metrics.

- *Diacritization error rate* (DER) which is the proportion of characters with incorrectly restored diacritics.
- *Word error rate* (WER) which is the percentage of incorrectly diacritized white-space delimited words: in order to be counted as incorrect, at least one letter in the word must have a diacritization error.

We calculate these two metrics as is done in previous related work [31,33]: (*i*) all words are counted including numbers and punctuators, (*ii*) each letter or digit in a word is a potential host for a set of diacritics, and (*iii*) all diacritics on a single letter are counted as a single binary (True or False) choice. Moreover, the target letter that is not diacritized is skipped as there is no reference to compare the output letter's diacritics with.

5 Experiments and Results

The following subsections present the experiments, and their results, that we carried out to train and tune the RNNs to automatically add diacritics on Arabic sentences.

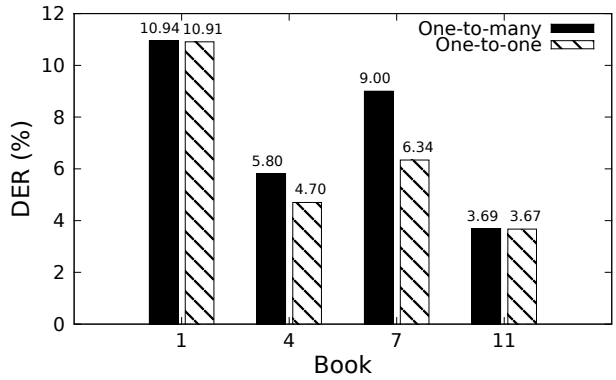

Fig. 6 The DER of the one-to-many and one-to-one networks on four books

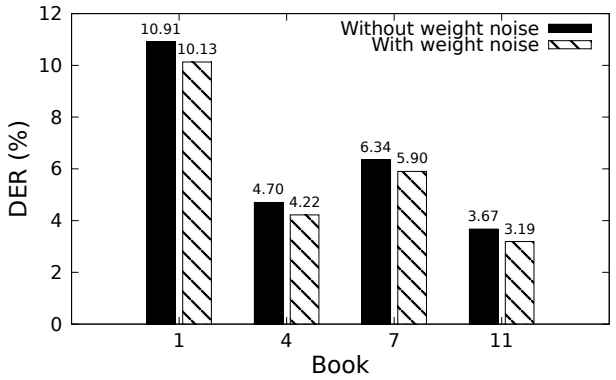

Fig. 7 The effect of using the weight noise distortion training option

5.1 One-to-many *vs.* one-to-one

We have evaluated the accuracy of the two sequence transcription networks: one-to-many and one-to-one. Figure 6 shows the diacritization error rates of these two networks for books of four representative sizes. The hidden layer size in both networks is 250 nodes and the one-to-many network has a 250-node *prediction* network. For the four books, the one-to-one network has lower error. We have also noticed that this advantage holds even when we change the network size and the training options. Therefore, we adopt the one-to-one network and use it in the following experiments.

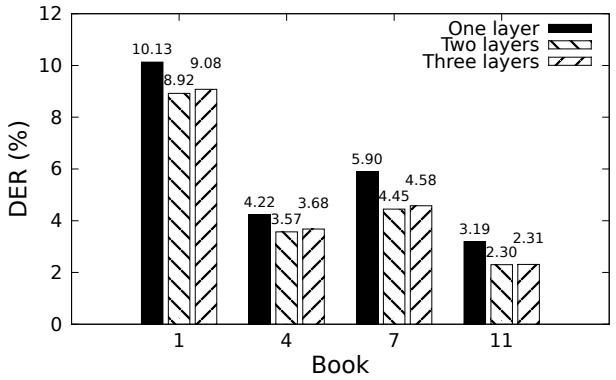

Fig. 8 The effect of changing the number of the RNN's hidden layers

5.2 Weight noise distortion

All networks were found to overfit on the training data, and we therefore used 'weight noise' regularisation [29, 14] to improve generalisation. Following previous work [14] the standard deviation of the noise was fixed at 0.075 for all network weights. Figure 7 shows the effect of training the RNN with and without weight noise. The figure shows the DER for the four representative books. These experiments were done using the one-to-one network of one hidden layer. In all the experiments that we have conducted, weight noise gave considerably superior results. Therefore we adopt this option in all the following experiments.

In fact best results are obtained when the network is trained in two steps. First the network is trained without weight noise. Then it is retrained with weight noise, starting from the weight values found in the first step.

5.3 Network size

As the neural network size and topology have a great impact on its performance, we conducted experiments to optimise these hyperparameters. Figure 8 shows the effect of changing the number of RNN hidden layers on the DER. Each hidden layer used here has 250 nodes. We have adopted the 250-node size because we noticed that using fewer nodes decreases the accuracy and using more nodes does not significantly improve it. This figure shows that the error decreases as we go from one layer to two layers. But the error increases slightly when we go from two to three. Additionally, three layers are slower than two layers. Therefore, we adopt the two-layer structure in the final system.

Similar results have been recorded in the speech recognition literature, with additional neural network hidden layers rapidly decreasing the error up to a certain point, after which performance either levels out or slightly degrades as more layers are added [9,20].

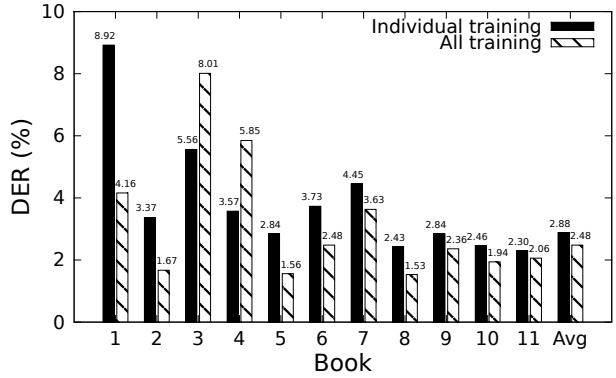

Fig. 9 The results of eleven books when each book is used in a separate training/testing experiment and the results when each testing set of a book is tested on an RNN that is trained using all the training sets of the 11 books

Table 3 Diacritization results of the LDC ATB3 samples

	Training Data	DER	WER
1	Eleven books	9.98%	30.27%
2	Eleven books and ATB3	5.82%	20.46%
3	ATB3 only	2.74%	9.31%

Table 4 The contributions of the post-processing techniques in reducing the DER in "all training" and the ATB3

Technique	Books	LDC ATB3
Sukun correction	2.5%	6.3%
Fatha correction	2.0%	1.1%
Dictionary correction	19.3%	1.3%
Total	23.8%	8.7%

5.4 Data size

We have conducted several experiments to study the effect of changing the training data size on performance. Figure 9 shows the DER for 11 books (the ten Tashkila books and the Quran) and the weighted average of their errors. This figure shows the error rates of two schemes:

Individual training Eleven RNNs are each trained and tested using the training and testing sets of one of the 11 books.

All training One RNN is trained using all the training sets of the 11 books and is tested 11 times using the testing sets of these books.

We have several observations about these experiments. The main observation is that the error rate generally decreases as the training set size increases. The weighted average DER when using all training sets is 2.48% versus 2.88% for the individual training. Moreover, the error rate generally decreases as the book size increases. Note that the books are ordered in increasing size as we go from Book 1 to Book 11, and the DER generally decreases as we go from left to right in Fig. 9.

For nine out of the 11 books, the "all training" scheme gives lower DER than the "individual training" scheme. Only Books 3 and 4 have higher error rates. These two books are Sahyh Muslim and the Quran and they seem to benefit more from the specific individual training than the "all training". It seems that their characteristics are, to some degree, different form the other books. For example, the Quran has the largest letters per word average among the used samples and Book 3 has the second smallest average (see Table 1).

The training time for the RNNs was in general very long, especially for the larger datasets. For example, it took more than three months to train the RNN using all 11 training sets (a total of 2,009 K words). However activating a trained RNN is relatively fast.

These experiments use the one-to-one netwok, two 250-node hidden layers, and the weight noise distortion training option.

This RNN setup is also used with the ATB3 samples. Table 3 shows the results of three experiments where three different training data are used: the training sets of the 11 books only, the 11 books and the ATB3, and the ATB3 only, respectively. The table shows that the worst result is got when the training is done using the classical 11 books. The results are improved when the MSA ATB3 train data is added to the 11 books. However, best DER at 2.74% and WER at 9.31% are obtained when training is done using the ATB3 train data only. This indicates that there are large differences between classical Arabic and MSA and different networks should be used with each type.

For the separate training and test sets case described in Section 4.3, we noticed that the DER is higher by less than 5% compared with the single set case. For the ATB3 samples, the DER is 2.87% (versus 2.74%) and for the "all training" samples, it is 2.57% (versus 2.48%). This increase is expected as separate training does not tune the net for the test set.

5.5 Influence of post-processing

The error rates reported in the previous subsections include the improvements due to Sukun and Fatha corrections, but not dictionary correction. Table 4 details the impact of all three post-processing techniques for the "all training" and ATB3 experiments.

The Sukun correction has more effect on the ATB3 than "all training" (6.3% *vs.* 2.5%), presumably because the ATB3 is partially diacritized.

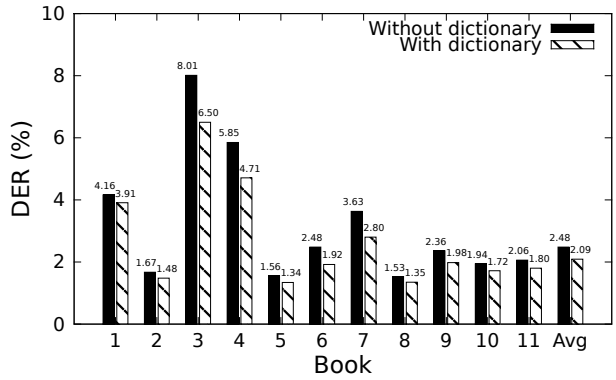

Fig. 10 The effect of dictionary corrections with "all training"

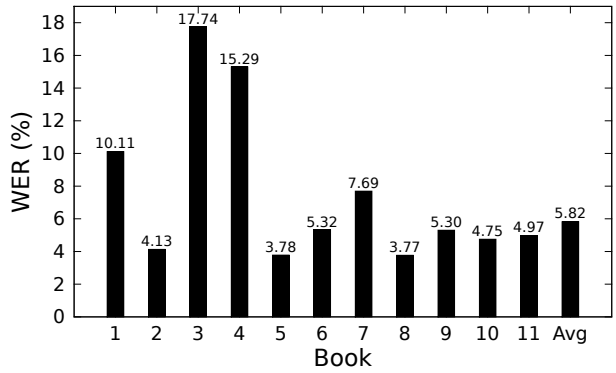

Fig. 11 The WER after dictionary corrections

The dictionary used in the "all training" case is constructed from the training sets of the 11 books and the dictionary used with the ATB3 is constructed from its training set (first 509 newswire stories). The dictionary correction gives significant error reduction of 19.3% in the "all training" case. However, it gives smaller reduction of only 1.3% with ATB3. The reason for this smaller improvement is the smaller ATB3 dictionary that is constructed from the earlier newswire stories. The last 90 stories used in the test include many new words not in the earlier stories.

In fact, dictionary correction in the "all training" case reduces the error rates for all 11 books, as shown in Figures 10, with the average DER dropping from 2.48% to 2.09%. In the ATB3 case, the DER is marginally reduced from 2.74% to 2.72%.

To conclude presenting results, we present the final WER for the "all training" case in Figure 11. The average WER at 5.82% is naturally higher than DER, but the relative standings of the 11 books is maintained. In the ATB3 case, the WER is 9.07%.

6 Discussion

Table 5 shows the diacritization results of best published systems and our system. For each system, the table shows its publication year, the database used in evaluating it, and its DER and WER. The table shows that over the last eight years, diacritization accuracy has generally improved and our system continues this trend with a significant boost over its predecessors.

The table shows two sets of results reported by Zitouni *et al.* who defined the ATB3 benchmark. The first set were obtained using a rich collection of input features, including POS tags, while the second set used only the raw undiacritized sentences, the same as our system.

Said *et al.* system, to best of our knowledge, is the system that has the best reported accuracy on the ATB3. For this database, our system provides 25% DER improvement and 20% WER improvement over Said *et al.* system. And this improvement is achieved with our approach that uses the raw undiacritized sentences only, without utilizing a hybrid approach combining statistical and rule-based techniques.

In evaluating his system, Hifny also used books from the Tashkila collection [37], similar to our work. Compared with the ATB3, Tashkila is much larger and is freely available. For the Tashkila samples, our system provides 38% DER improvement and 35% WER improvement over Hifny's system.

The results shown for the KAD system were reported by Bahanshal and Al-Khalifa [6]. Among three available automatic diacritization systems evaluated in their study (KAD, Sakhr, and Mishkal), KAD has the best results for diacritizing verses from the Holy Quran. However, our system has much better accuracy in diacritizing Quran verses, especially when it is trained only on some of the Quran verses. For Quran, our system provides 45% DER improvement over KAD. This table shows the Quran results for the "individual training" case followed by the "all training" case.

In addition to showing the error rates when all diacritization errors are counted, Table 5 shows the error rates when the errors in diacritizing the last letter of each word are ignored. Although the proportion of last letters to all letters is one to 4.64, the error rates are significantly lower when these errors are ignored. The diacritic of the last letter is generally determined by the syntax and is usually harder to get right compared with the diacritics of other letters.

The last column in Table 5 shows the difference between the All-Diacritics DER and Ignore-Last DER. This difference is the proportion of last letter diacritization errors of all letters. This column demonstrates that

Table 5 Diacritization results of related work and our system. The table shows the reported error rates when all diacritization errors are counted (All Diacritics) and when the diacritization errors of the last word letters are ignored (Ignore Last). The last column is the difference between the All-Diacritics DER and Ignore-Last DER.

		All Diacritics		Ignore Last		
System	Data	DER	WER	DER	WER	DER-Last
Zitouni *et al.* 2006 [38]	ATB3	5.5 (8.2)	18.0 (25.1)	2.5	7.9	3.0
Habash & Rambow 2007 [22]	ATB3	4.8	14.9	2.2	5.5	2.6
Rashwan *et al.* 2011 [31]	ATB3	3.8	12.5	1.2	3.1	2.6
Said *et al.* 2013 [33]	ATB3	3.6	11.4	1.6	4.4	2.0
Hifny 2012 [23]	Tashkila	-	8.9	-	3.4	
KAD System 2012 [6]	Quran	5.5	-	-	-	
This work	ATB3	**2.72**	**9.07**	1.38	4.34	**1.34**
	Tashkila	**2.09**	**5.82**	1.28	3.54	0.81
	Quran (indiv.)	**3.04**	**8.70**	1.98	5.82	1.06
	Quran (all)	4.71	15.29	3.07	10.23	1.64

Table 6 Distribution of word errors for the "all training" case

Errors per word	One	Two	Three+	Total
Last letter OK	38.3%	10.6%	1.7%	50.6%
Error in last letter	39.2%	7.6%	2.6%	49.4%
Total	77.5%	18.2%	4.3%	100.0%

our system is more successful than previous work in reducing this error. This error rate (1.34) is 33% lower than Said *et al.*'s system on the ATB3.

Now we discuss the errors of our system. Table 6 shows the distribution of word errors according to the number of diacritic errors per word and whether there is an error in the diacritic of the last letter or not. The table shows that more than three quarters of the word errors (77.7%) have one diacritic error, two errors per word comprise 18.2%, and three errors or more per word are not frequent (4.3%).

The table also shows that about half the word errors (49.4%) have an error in the last letter. As there are an average of about four letters per word, one would expect to have smaller error rate in the last letter had the errors been uniformly distributed. However, as other researchers have noticed, the diacritic of the last letter highly depends on the context and is hard to predict in the Arabic language that has rich inflection rules [38]. Moreover, last-letter errors are hard to correct using dictionaries.

We have manually inspected 200 error samples. Table 7 shows six sample sequences that have errors. The words that have errors are underlined and the table shows the target (test) sequence and the output sequence.

We have noticed that, in about 55% of the samples, the word that has diacritic error is a valid Arabic verb or noun. Sample 1, for example, shows that target stem verb "saffa" صَفَّ (past tense of the verb enqueue) is output as "sif" صِفْ (command form of the verb describe). Sample 2 shows that the noun "hubbu" حُبُّ (love) is output as "habbu" حَبُّ (seeds). Sample 3, shows a counter example where the output word يَصْنَّعَ is not a valid Arabic word.

Sample 4 shows an example where the output word is missing the diacritic of the last letter "akhiyk" أَخِيك (your brother). This word has the possessive pronoun suffix "ka" كَ. We have traced the reason of this error to the training data used. We have noticed that this pronoun suffix is not diacritized in many cases in the training data.

We also noticed that a significant fraction of the error words (23%) have prefixes or suffixes or both, as in Samples 1, 4, and 5. The error word in Sample 5 has both; it has the prefix "la" لَ, the stem verb "tarawunna" تَرَوُنَّ, and the pronoun suffix "haa" هَا. These complex words are the hardest words to be correctly diacritized especially because the inflection dicritic often goes to

Table 7 Sample sequences with errors

Sample	Target Sequence	Output Sequence
1	ثُمَّ صَفَّهُمْ	ثُمَّ صِفْهُمْ
2	وَحُبُّ الْجَاهِ عَلَى الْقَلْبِ غَالِبٌ	وَحَبُّ الْجَاهِ عَلَى الْقَلْبِ غَالِبٌ
3	إِلَّا أَنْ يَصْنَعَ مَا لَا يَصْنَعُهُ النَّاسُ	إِلَّا أَنْ يَصْنَّعَ مَا لَا يَصْنَعُهُ النَّاسُ
4	أُخْتُ أَخِيكَ	أُخْتُ أَخِيك
5	ثُمَّ لَتَرَوُنَّهَا عَيْنَ الْيَقِينِ	ثُمَّ لَتَرَونِهَا عَيْنَ الْيَقِينِ
6	يُقِيمُ عِنْدَهُ وَلَا شَيءَ لَهُ يَقْرِيهِ بِهِ	يُقِيمُ عِنْدَهُ وَلاَ شَيءَ لَهُ يَقْرِيهِ بِهِ

the last letter in the stem, and not to the last letter of the suffix.

We have estimated that about 3% of the errors are due to diacritization errors in the test samples used. For example, Sample 6 shows that the Fatha in the word "laa" لَا was mistakenly entered after the last letter (لاَ), not after the first letter.

It seems that the Shadda diacritic is harder to restore than the average diacritic; although the samples have about 6.5% Shadda diacritic relative to all diacritics, Shadda errors are about 9.8% of the DER. Note that the problematic words in Samples 1, 2, 3, and 5 have Shadda diacritic in the target or the output word.

7 Conclusions

In this paper, we have tackled the problem of adding diacritics to Arabic text as a sequence transcription problem using deep bidirectional LSTM network. Our approach is a pure machine learning approach. The network is trained on transcribing raw undiacritized Arabic text sequences into fully diacritized sequences.

We have experimentally selected the type and configuration of the used network. Best accuracy is obtained when using the one-to-one network, training with weight noise distortion, and using two hidden layers each of 250-node size.

We have used samples from ten books of the Tashkila collection and the Quran to test this approach. The experimental evidence indicates that the accuracy improves as the training set size increases. Best average results are obtained when training the network using the training sets of the 11 books.

In order to improve accuracy, we used four post-processing correction techniques: letter, Sukun, Fatha, and dictionary. These techniques reduce the average DER by 23.8% for the 11 books. The dictionary correction is responsible of 19.3% of this reduction.

We have also tested this approach on the LDC ATB3 that is widely used in related work. Although some words in the ATB3 are not, or partially, diacritized and the ATB3 is smaller than the 11 books samples, this approach achieves high accuracy on the ATB3 as well. The achieved DER of 2.72% and WER of 9.07% are better than the best published results by 25% and 20%, respectively. Also the last letter diacritization error rate is 33% lower than the best published results.

These results are mainly due to the sequence transcribing capabilities of the network used and its ability to exploit long-range context dependencies in the foreword and backward directions. Even without the post-processing techniques used, this approach has a DER better than the best published results by 18%.

This approach outperforms leading hybrid approaches. It gives better results without utilizing available rule-based techniques such as morphological analysis. However, we think that integrating such techniques as pre-processing stages before the sequence transcription stage could provide higher accuracy.

We intend to experiment with adding such techniques to this approach. This future work is motivated by the observation that significant fraction of the errors is in complex words that have prefixes, suffixes, or both. We expect that providing the morphological analysis of such words to the RNN would provide it with better information to achieve higher accuracy.

References

1. Abandah, G., Khundakjie, F.: Issues concerning code system for Arabic letters. Dirasat Engineering Sciences Journal **31**(1), 165–177 (2004)
2. Abandah, G.A., Jamour, F.T., Qaralleh, E.A.: Recognizing handwritten Arabic words using grapheme segmentation and recurrent neural networks. Int'l J. Document Analysis and Recognition **17**(3), 275–291 (2014)
3. Al-Sughaiyer, I.A., Al-Kharashi, I.A.: Arabic morphological analysis techniques: A comprehensive survey. Journal of the American Society for Information Science and Technology **55**(3), 189–213 (2004)
4. Azim, A.S., Wang, X., Sim, K.C.: A weighted combination of speech with text-based models for Arabic diacritization. In: 13th Annual Conf. Int'l Speech Communication Association, pp. 2334–2337 (2012)
5. Azmi, A.M., Almajed, R.S.: A survey of automatic Arabic diacritization techniques. Natural Language Engineering pp. 1–19 (2013). DOI 10.1017/S1351324913000284. Published online
6. Bahanshal, A., Al-Khalifa, H.S.: A first approach to the evaluation of Arabic diacritization systems. In: Int'l Conf. Digital Information Management, pp. 155–158 (2012)
7. Beesley, K.R.: Arabic finite-state morphological analysis and generation. In: 16th Conf. on Computational Linguistics, vol. 1, pp. 89–94 (1996)
8. Buckwalter, T.: Buckwalter Arabic Morphological Analyzer, v2.0 edn. Linguistic Data Consortium, Philadelphia (2004)
9. Dahl, G., Yu, D., Deng, L., Acero, A.: Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. IEEE Trans. Audio, Speech, and Language Processing **20**(1), 30–42 (2012)
10. Damerau, F.J.: A technique for computer detection and correction of spelling errors. Comm. of the ACM **7**(3), 171–176 (1964)
11. El-Sadany, T., Hashish, M.: Semi-automatic vowelization of Arabic verbs. In: 10th National Computer Conf., pp. 725–732 (1988)
12. Gal, Y.: An HMM approach to vowel restoration in Arabic and Hebrew. In: ACL-02 Workshop on Computational Approaches to Semitic Languages, pp. 1–7 (2002)
13. Gers, F., Schraudolph, N., Schmidhuber, J.: Learning precise timing with LSTM recurrent networks. Journal of Machine Learning Research **3**(1), 115–143 (2002)
14. Graves, A.: Practical variational inference for neural networks. In: Advances in Neural Information Processing Systems, pp. 2348–2356. Curran Associates, Inc. (2011)
15. Graves, A.: Offline Arabic handwriting recognition with multidimensional recurrent neural networks. In: V. Märgner, H. El Abed (eds.) Guide to OCR for Arabic Scripts, pp. 297–313. Springer, London (2012)
16. Graves, A.: Sequence transduction with recurrent neural networks. In: ICML Representation Learning Worksop (2012)
17. Graves, A.: Supervised sequence labelling with recurrent neural networks. Springer, Berlin (2012)
18. Graves, A.: Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850 (2013)
19. Graves, A.: RNNLIB: A recurrent neural network library for sequence learning problems. http://sourceforge.net/projects/rnnl/ (2013)
20. Graves, A., Mohamed, A.r., Hinton, G.: Speech recognition with deep recurrent neural networks. In: IEEE Int'l Conf. Acoustics, Speech and Signal Processing, pp. 6645–6649 (2013)
21. Graves, A., Schmidhuber, J.: Framewise phoneme classification with bidirectional LSTM and other neural network architectures. Neural Networks **18**(5-6), 602–610 (2005)
22. Habash, N., Rambow, O.: Arabic diacritization through full morphological tagging. In: Conf. North American Chapter of the Association for Computational Linguistics, pp. 53–56 (2007)
23. Hifny, Y.: Smoothing techniques for Arabic diacritics restoration. In: 12th Conf. on Language Engineering, pp. 6–12 (2012)
24. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation **9**(8), 1735–1780 (1997)
25. Kirchhoff, K., Bilmes, J., Das, S., Duta, N., Egan, M., Ji, G., He, F., Henderson, J., Liu, D., Noamany, M., et al.: Novel approaches to Arabic speech recognition: report from the 2002 Johns-Hopkins summer workshop. In: IEEE Int'l Conf. Acoustics, Speech, and Signal Processing, vol. 1, pp. 344–347 (2003)
26. Lewis, M.P. (ed.): Ethnologue: Languages of the World, 16th ed. edn. SIL International, Dallas (2009)
27. Maamouri, M., Bies, A., Buckwalter, T., Mekki, W.: The Penn Arabic treebank: Building a large-scale annotated Arabic corpus. In: NEMLAR Conf. Arabic Language Resources and Tools, pp. 102–109 (2004)
28. Märgner, V., El Abed, H.: ICDAR 2009 - Arabic handwriting recognition competition. In: Int'l Conf. Document Analysis and Recognition, pp. 1383–1387 (2009)
29. Murray, A.F., Edwards, P.J.: Enhanced MLP performance and fault tolerance resulting from synaptic weight noise during training. IEEE Trans. Neural Networks **5**(5), 792–802 (1994)
30. Nelken, R., Shieber, S.M.: Arabic diacritization using weighted finite-state transducers. In: ACL Workshop on Computational Approaches to Semitic Languages, pp. 79–86 (2005)
31. Rashwan, M., Al-Badrashiny, M., Attia, M., Abdou, S., Rafea, A.: A stochastic Arabic diacritizer based on a hybrid of factorized and unfactorized textual features. IEEE Trans. Audio Speech Language Processing **19**(1), 166–175 (2011)
32. Ryding, K.C.: A Reference Grammar of Modern Standard Arabic. Cambridge University Press, Cambridge (2005)
33. Said, A., El-Sharqwi, M., Chalabi, A., Kamal, E.: A hybrid approach for Arabic diacritization. In: E. Mtais, F. Meziane, M. Saraee, V. Sugumaran, S. Vadera (eds.) Natural Language Processing and Information Systems, *Lecture Notes in Computer Science*, vol. 7934, pp. 53–64. Springer (2013)
34. Schuster, M., Paliwal, K.K.: Bidirectional recurrent neural networks. IEEE Trans. Signal Processing **45**(11), 2673–2681 (1997)
35. Vergyri, D., Kirchhoff, K.: Automatic diacritization of Arabic for acoustic modeling in speech recognition. In: Workshop on Computational Approaches to Arabic Script-based Languages, pp. 66–73 (2004)
36. Zarrabi-Zadeh, H.: Tanzil - Quran Navigator. http://tanzil.net/download. Last accessed on Nov 27, 2014
37. Zerrouki, T.: Arabic corpora resources, Tashkila collection from the Arabic Al-Shamela library. http://aracorpus.e3rab.com. Last accessed on Nov 27, 2014
38. Zitouni, I., Sorensen, J.S., Sarikaya, R.: Maximum entropy based restoration of Arabic diacritics. In: 21st Int'l Conf. Computational Linguistics, pp. 577–584 (2006)