Deep Diacritization: Efficient Hierarchical Recurrence for Improved Arabic Diacritization

Badr AlKhamissi[1], Muhammad N. ElNokrashy[1,2], and Mohamed Gabr[2]

badr@khamissi.com, muhammad.nael@gmail.com, mohamed.gabr@hotmail.com

[1]*The American University in Cairo (AUC)*

[2]*Microsoft Egypt Development Center (EGDC)*

Abstract

We propose a novel architecture for labelling character sequences that achieves state-of-the-art results on the Tashkeela Arabic diacritization benchmark. The core is a two-level recurrence hierarchy that operates on the word and character levels separately—enabling faster training and inference than comparable traditional models. A cross-level attention module further connects the two, and opens the door for network interpretability. The task module is a softmax classifier that enumerates valid combinations of diacritics. This architecture can be extended with a recurrent decoder that optionally accepts priors from partially diacritized text, which improves results. We employ extra tricks such as sentence dropout and majority voting to further boost the final result. Our best model achieves a WER of 5.34%, outperforming the previous state-of-the-art with a 30.56% relative error reduction.

1 Introduction

The Arabic script (and similarly Hebrew, Aramaic, Pahlavi...) is an impure abjad. These writing systems represent short consonants and long vowels using full letter graphemes, but generally omit short vowels and consonant length from writing. This leaves the task of inferring the missing phonemes to the reader by using context from neighbouring words and knowledge of the language structure to determine the correct pronunciation and disambiguate the meaning of the text. Those sounds are represented by diacritical marks—small graphemes that appear usually above or below a basic letter in the abjad. Table 2 shows the diacritics considered in this work. Diacritics are usually utilized in specific domains where it is important to explicitly clear up ambiguities or where inferring the correct forms might be difficult for non-experts, such as religious texts, some literary works such as poetry, and language teaching books as novice readers have yet to build up the intuition for reading undiacritized text.

We focus in this work on diacritization of Arabic texts. However, our proposed architecture has no explicitly language-dependent components and should be adaptable for other character sequence labelling tasks. Although it is the first language of several million people, and is spoken in some of the fastest growing markets (Tinsley and Board, 2013), the Arabic language, like many others, lacks attention from the NLP community compared to established test bed languages such as English or Chinese, which both enjoy higher momentum and an abundance of established resources and techniques. The automatic restoration of diacritics to Arabic text is arguably one of the most important NLP tasks for the Arabic language. Besides direct applications like facilitating learning, diacritics are used to enhance language modeling, acoustic modeling for speech recognition, morphological analysis, machine translation, and text-to-speech systems (which need to restore the lost phonemes to render words properly) (Zitouni and Sarikaya, 2009; Azmi, 2013).

Affiliation emails: [1]{balkhamissi, m.n.elnokrashy}@aucegypt.edu
[2]{muelnokr, mogabr}@microsoft.com

* This work is not sponsored by the affiliated institutions of the authors.

** This work was accepted at the Fifth Arabic Natural Language Processing Workshop (WANLP 2020).

Proceedings of the Fifth Arabic Natural Language Processing Workshop, pages 38–48
Barcelona, Spain (Online), December 12, 2020

To illustrate this further, Table 1 shows the Arabic word *Elm*[1] in different diacritized forms with their corresponding English translations, showcasing the importance of diacritics in resolving ambiguity. Note that the MADA (Habash et al., 2009) morphological analyzer produces at least 13 different forms for this undiacritized word (Belinkov and Glass, 2015).

Arabic (diacritized)	Transliteration	English Translation
عَلِمَ	*Ealima*	He knew
عُلِمَ	*Eulima*	It was known
عَلَّمَ	*Eal∼ama*	He taught
عِلْمٌ	*Eilomu*	Knowledge
عَلَمٌ	*Ealamu*	Flag

Table 1: Subset of possible diacritized forms for *Elm* adapted from (Belinkov and Glass, 2015)

Table 2 shows the different diacritics commonly used in Arabic texts along with their phonemic symbols. They fit roughly into four kinds. (1) Ḥarakāt are diacritics for short vowels; we have three: fatḥah, kasrah, dammah. The symbols for those vowels have another form (usually a visual doubling) used at the end of a word to form a (2) tanwīn, or nunation, which is a VC sound of the ḥarakah's vowel followed by the consonant "n" ($\{a, i, u\}n$). (3) The shaddah is the gemination symbol used to indicate consonant doubling. It can be combined with one of the ḥarakāt or tanwīn on the same character. Finally, (4) the sukūn is used to indicate that the current consonant is not followed by a vowel and instead forms a cluster with the next consonant. Diacritics which appear at the end of a word are referred to as case-endings (CE); most of which are specified by the syntactic role of the word. They are harder to infer than the core-word diacritics (CW) that specify lexical selection and appear on the rest of the word (Mubarak et al., 2019).

Symbol	Name	Type	Transliteration	IPA phoneme
ـُ	dammah	ḥarakāt	*u*	/u/
ـَ	fathah	ḥarakāt	*a*	/a/
ـِ	kasrah	ḥarakāt	*i*	/i/
ـٌ	dammatain	tanwīn	*N*	/un/
ـً	fathatain	tanwīn	*F*	/an/
ـٍ	kasratain	tanwīn	*K*	/in/
ـّ	shaddah	shaddah	∼	/h:/ Gemination.
ـْ	sukūn	sukūn	*o*	No vowel.

Table 2: Primary Arabic diacritics on letter ـه

The paper is structured as follows: First we cover some of the approaches used in related works on restoring Arabic diacritics. Then we introduce our system and support it by comparing experimental results on an adapted version of the Tashkeela corpus (Zerrouki and Balla, 2017) proposed by (Fadel et al., 2019a) as a standard benchmark for Arabic diacritization systems. Each design decision will then be motivated by an ablation study. We analyze the learned attention model then discuss existing limitations in an error analysis. Finally, we offer directions for future work.

[1]This paper uses Buckwalter transliteration.

2 Related Work

The literature surrounding the automatic diacritization of Arabic text provides methods in two categories: classical rule-based solutions, and statistical modeling-based methods. Early approaches have worked on constructing a large set of language specific rules to restore the lost diacritics (Habash et al., 2009; Zitouni et al., 2006; Pasha et al., 2014; Darwish et al., 2017). Researchers have then shifted to rely more on learning-based methods that do not require extra expert systems such as morphological analyzers and part-of-speech taggers. (Belinkov and Glass, 2015) have shown that recurrent neural networks are suitable candidate models for learning the task entirely from data and can be easily extended to other languages and dialects without the use of manually engineered features. Other methods such as hidden Markov models (HMMs) (Elshafei et al., 2006), conditional random fields (CRFs) (Darwish et al., 2017), maximum-entropy models (Zitouni et al., 2006) and finite-state transducers (Nelken and Shieber, 2005) have similarly been employed. However, more recent works have started to use deep (neural-based) architectures such as sequence-to-sequence transformers and recurrent cell-based models inspired by work in Neural Machine Translation (Mubarak et al., 2019). Solutions combining both rule-based and deep learning methods appear in recently published work (Abbad and Xiong, 2020). (Zalmout and Habash, 2020) have shown that the diacritization task benefits from jointly modelling lexicalized and non-lexicalized morphological features instead of targeting only the diacritization task.

3 Approach

3.1 Datasets

We report on the cleaned version of the Tashkeela corpus (Fadel et al., 2019a)—a high quality, publicly available dataset. It is split into train (2,449k tokens), dev (119k tokens), and test (125k tokens) sets.

3.2 Architecture

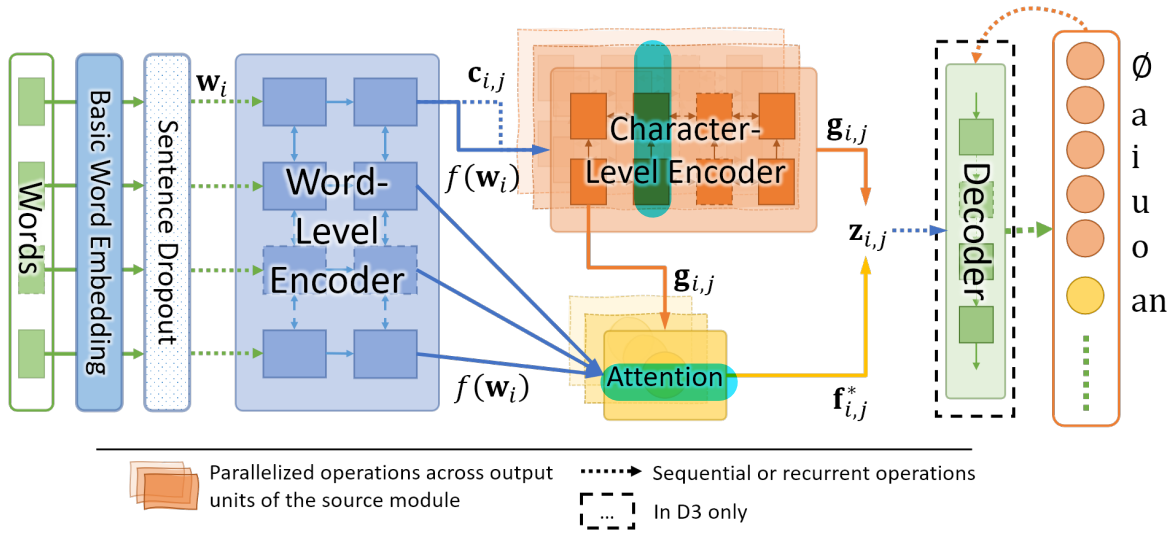


Figure 1: D3 Architecture

In this work, we propose two models: The Two-Level Diacritizer (D2) and the Two-Level Diactritizer with Decoder (D3). D3 extends D2 by allowing partially diacritized text to be taken as input—a much needed feature for neural diacritizers (Fadel et al., 2019a). D2 outperforms fully character-based models in both task and runtime performance measures.

3.2.1 Two-Level (Hierarchical) Model

Restoring diacritics can be seen as a character sequence labeling task. Each label depends on word- and character-level context. This structure motivates the hierarchy in our two-level encoder architecture—the first encoder sees the sequence of words (where words are atoms) and provides word-level context

for the second, character-level encoder. The character-level encoder is evaluated independently for each word in the sentence, enabling much faster training and inference compared to character-level recurrent cell-based models of similar structure to previous works (Belinkov and Glass, 2015; Mubarak et al., 2019; Zalmout and Habash, 2020; Fadel et al., 2019b). Let T_s be the maximum number of words allowed in a sentence and T_w the maximum number of characters allowed in a word. Then the overall character sequence length that a model in prior works would see is in the order of $(T_s \cdot T_w)$. In contrast, our approach operates on a maximum sequence length of T_s at the word level and T_w at the character level. Because character-level recurrence is independent of characters outside the current word, the serial bottleneck complexity goes from $O\left(\sum_{w \in s} |w|\right) \approx O(T_s \cdot T_w)$ down to $O\big(|s| + \max_{w \in s}\{|w|\}\big) \approx O(T_s + T_w)$, assuming adequate parallelization across word units. See Table 4 for a speed comparison.

Let $s = \{w_i\}_{i=1}^{T_s}$ denote a sequence of words. $\mathbf{w} = \{\mathbf{w}_i\}_{i=1}^{T_s}$ are the corresponding fastText features pretrained on CommonCrawl Arabic data (Bojanowski et al., 2017). Let $w_i = \{c_{i,j}\}_{j=1}^{T_w}$ denote the sequence of characters for the word at index i in sentence s. Each character is assigned a 32-dimensional learned embedding $\mathbf{c}_{i,j}$. Let $f(\mathbf{w})_i$ denote the feature vector from the word-level encoder which maps each word in s to a contextual word representation. Let $g(\cdot)_{i,j}$ denote the character-level recurrence that outputs a contextual encoding of the character relative to its parent word and sentence. See Figure 1 for an overview. Formally, the contextual embedding $\mathbf{z}_{i,j}$ of $c_{i,j}$ is

$$\mathbf{g}_{i,j} = g\left([\mathbf{c}_{i,j}\,;\, f(\mathbf{w})_i]\right) \tag{1}$$

$$\mathbf{z}_{i,j} = \left[\mathbf{g}_{i,j}\,;\, \mathbf{f}^*_{i,j}\right] \tag{2}$$

Where $\mathbf{f}^*_{i,j}$ is the attention view. Both $f(\cdot)$ and $g(\cdot)$ use Bidirectional LSTM (Bi-LSTM) layers (Graves et al., 2005) trained with backpropagation through time. We note that any similar sequence modelling architecture would be applicable (Chung et al., 2014; Vaswani et al., 2017) but leave that to future work.

3.2.2 Cross-level Attention Module

This module attends over the word-level encodings $f(\mathbf{w})$ based on the character encodings $g(\cdot)$ of each character in a word. In other words, it uses the initial contextualization of the characters ($\mathbf{g}_{i,j}$) to attend to all words in the sentence (except the current) to refine the character's representation. We use the attention formulation from (Vaswani et al., 2017). For each character $c_{i,j}$, we calculate

$$\mathbf{f}^*_{i,j} = \text{AttendReduce}\left(\mathbf{u} = \mathbf{g}_{i,j}\,;\mathbf{X} = \{...f(\mathbf{w})_{0:i-1}, f(\mathbf{w})_{i+1:T_s}...\}\right) \tag{3}$$

where

$$\text{AttendReduce}\left(\mathbf{u}; \mathbf{X}\right) = W^O\left(\underset{t}{\text{Softmax}}\left(\frac{W^Q(\mathbf{u}) \cdot W^K(\mathbf{X})_t^\top}{\sqrt{d_K}}\right) \cdot W^V(\mathbf{X})\right) \tag{4}$$

where in eq. (4), W^Q, W^K, W^V, and W^O are independent linear layers. We tried to remove W^O but faced lower performance.

3.2.3 Decoder

Used in D3, this component is a forward-only LSTM that takes as input a concatenation of the basic contextual character embedding $\mathbf{z}_{i,j}$ and a one-hot representation of the output of the previous character from the classifier module. Formally: $[\mathbf{z}_{i,j}; \hat{\mathbf{y}}_{i,j-1}]$. The $\hat{\mathbf{y}}_{i,j-1}$ signal passed to the decoder also encodes the Beginning-of-Word in addition to the previous-character diacritics. This allows the model to accept partially diacritized sentences such that the ground truth diacritic is injected in place of $\hat{\mathbf{y}}_{i,j-1}$ during inference. This feature is important as many Arabic texts contain sparse diacritics that act as hints to assist readers. Having this clean signal yields improvements as shown in Figure 2.

3.2.4 Task Objective

The final classifier optimizes a Softmax objective over an enumeration of all valid diacritic combinations. Combined, we have 3 ḥarakāt in 4 variants (with tanwīn, shadda, tanwīn and shadda, and neither), the sukūn, and the plain shadda. Thus 15 classes including the None (no diacritic) class.

3.3 Experimental Setup

3.3.1 Parameters, Hyper-parameters, and Regularization

Optimization We use the Adam optimizer (Kingma and Ba, 2014) with an initial learning rate of 0.002. The model is left to converge until the validation loss does not improve for 3 consecutive epochs where each epoch enumerates a randomly shuffled version of the training segments exactly once. The learning rate is reduced by half when the validation loss does not improve for one epoch. We train with a mini-batch size of 128 segments.

Encoders and Decoder The word and character level encoders are *each* a 2-layer stacked Bi-LSTM with 256 and 512 hidden units, respectively. We apply feature-level dropout (Srivastava et al., 2014) with probability 0.2 to the input of the character level encoder. The decoder in D3 is a one-layer forward-only LSTM with 1024 hidden units. All recurrent cells use a vertical and recurrent dropout of 0.25 each. The recurrent dropout used is untied between time-steps, in contrast to (Gal and Ghahramani, 2016).

Context Window and Voting Similar to (Mubarak et al., 2019), we use a sliding context window of size T_s on each sentence. A given sentence is split into several overlapping segments each of which is given separately to the model during training. This works well as the local context is often sufficient for correct inference. During inference, the same sequence of characters may appear in different contexts (different segments from one sentence) and potentially lead to different diacritized forms. To choose the final diacritic, we use a popularity voting mechanism and, in the case of a tie, choose one of the outputs at random. The values chosen for T_s, T_w and the *stride* are: $T_s = 10$ with *stride* $= 1$ for training and validation (a small T_s was observed to stabilize training and improve results); $T_s = 20$ with *stride* $= 2$ for evaluation/testing; and $T_w = 13$ for both training and evaluation.

Sentence Dropout We randomly dropout 20% of the words given to the word-level encoder during training. The positions of the dropped out words are preserved, and their embedding vectors $\mathbf{w}_i$ are replaced with zeros. This was observed to lead to better generalization in some cases.

3.3.2 D3 Training

This model is not trained from scratch, but uses the weights of the encoders and character embeddings learned from D2. Those weights are kept frozen and only the decoder and classifier are trained.

Ramp-up of Teacher-forcing Signal We pass the ground truth of $p\%$ of the previous-character diacritics as input to the decoder at the current time-step. This value is ramped up from $p = 0\%$ (all characters receive previous diacritics as zeros; i.e. no signal) to $p = 100\%$ (all characters receive ground truth of previous diacritic as signal). This is done over a period of $n = 10$ epochs in increments of 10%. Then the model is left to converge using the same stopping criteria as in D2. We found this to be the best approach as otherwise the model overfits early on the teacher forcing signal given from the previous time-step.

3.3.3 Source Code

The code is made open source and is available on GitHub[2]. We also provide an accompanying web application to demo the proposed models which can be found at this web address[3]. The system uses PyTorch for implementing the neural training and inference components (Paszke et al., 2019). The PyTorch LSTM cell implementation used is due to (ElNokrashy, 2020).

3.4 Results

We use the script provided by (Fadel et al., 2019a) to evaluate our results. To be consistent with prior work, we report our results in terms of both word error-rate (WER) and diacritic error-rate (DER), with and without case-endings, as well as including and excluding characters with no diacritics. Table 3 shows our results on the Tashkeela benchmark in comparison with the more recent works. We outperform state-of-the-art by 30.56% relative (2.35% absolute) error reduction on "WER with case-ending".

[2] https://github.com/bkhmsi/deep-diacritization
[3] https://deep-diacritization.herokuapp.com
[4] Results from (Fadel et al., 2019a).

DER/WER	Including 'no diacritic'		Excluding 'no diacritic'	
	w/ case ending	w/o case ending	w/ case ending	w/o case ending
(Barqawi, 2017)[4]	3.73% / 11.19%	2.88% / 6.53%	4.36% / 10.89%	3.33% / 6.37%
(Fadel et al., 2019b)	2.60% / 7.69%	2.11% / 4.57%	3.00% / 7.39%	2.42% / 4.44%
(Abbad and Xiong, 2020)	3.39% / 9.94%	2.61% / 5.83%	3.34% / 7.98%	2.43% / 3.98%
D2 (Ours)	1.85% / 5.53%	1.49% / 3.27%	2.11% / 5.26%	1.71% / 3.15%
D3 (Ours) (@0% hints)	**1.83% / 5.34%**	**1.48% / 3.11%**	**2.09% / 5.08%**	**1.69% / 3.00%**

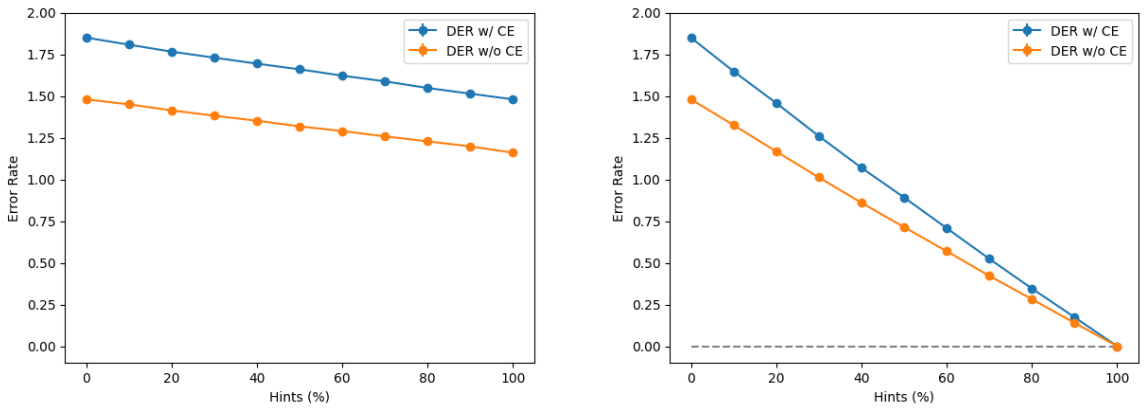
Table 3: Results on the Tashkeela benchmark

Method	**#Params**	**T/epoch**	**Convergence**	**Inference**	**Full DER/WER**
D2 – {Attn}	13.369M	**28 mins**	17 epochs	**34,996 wps**	**1.94% / 5.80%**
Flat	**13.304M**	121 mins	**13 epochs**	2,466 wps	2.20% / 6.39%

Table 4: Speed Comparison[5]

Table 4 compares our plain 2-level hierarchy design (without Attention) with a "Flat" model in task and runtime performance. The Flat model comprises a 4-layer stacked Bi-LSTM with similar implementation details as described in 3.3.1 for D2, including Sentence Dropout and the Voting mechanism. The Flat model sees each sentence as one sequence of characters.

Partially Diacritized Text Figure 2 shows the results of DER including 'no diacritic' with and without case ending when the model is supplied with partially diacritized text as input. For each character in the sentence, with some probability, we may replace the predicted output of the previous time-step with the ground truth as input to the decoder in the current step. The reported output of the previous time-step is masked to force the provided hint to be the model's "prediction" even if the inferred were different (see Figure 2b)—in contrast to Figure 2a where the final predictions are the model's unmodified outputs. The results are averaged across five runs with different seeds (i.e. injecting the ground truth signal at different characters in the sentence). Error bars represent standard deviation. Many Arabic texts already come with some hints that can improve model performance. Here we show how a neural model could be trained to leverage that.



(a) DER vs Percentage of Injected Hints (actual output)

(b) DER vs Percentage of Injected Hints (hints covering output)

Figure 2: Error Rate of Partially Diacritized Text

[5]All models use the same custom LSTM implementation and are run on a single Nvidia GeForce RTX 2080 Ti.

4 Discussion

4.1 Ablation Study

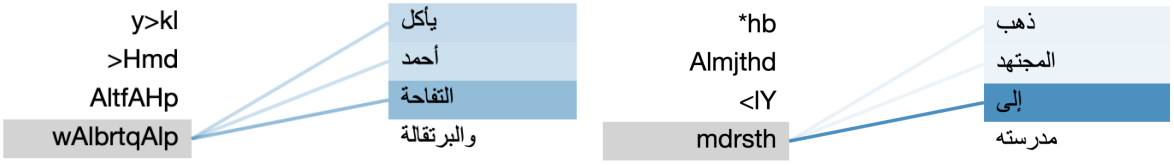
We conduct an ablation study to measure the effect of components proposed for the final model. We train and evaluate the D2 model previously detailed but with the component(s) specified removed. Table 5 shows the results after removing the sentence dropout and cross-level attention module.

DER/WER	Including 'no-diacritic'		Excluding 'no-diacritic'	
	w/ case ending	w/o case ending	w/ case ending	w/o case ending
(Fadel et al., 2019b)	2.60% / 7.69%	2.11% / 4.57%	3.00% / 7.39%	2.42% / 4.44%
D3 (@0% hints)	**1.83% / 5.34%**	**1.48% / 3.11%**	**2.09% / 5.08%**	**1.69% / 3.00%**
D2	1.85% / 5.53%	1.49% / 3.27%	2.11% / 5.26%	1.71% / 3.15%
D2 − {Attention 3.2.2}	1.94% / 5.80%	1.58% / 3.44%	2.23% / 5.52%	1.80% / 3.31%
D2 − {SDO 3.3.1}	1.91% / 5.71%	1.54% / 3.36%	2.18% / 5.43%	1.75% / 3.23%
D2 − {Attn, SDO}	1.93% / 5.78%	1.57% / 3.45%	2.21% / 5.49%	1.79% / 3.32%

Table 5: Ablation Study

4.2 Attention Analysis

The cross-level attention module allows us to gauge the contribution of each word to each output diacritic. Here we examine some examples to see whether the model was able to learn such Arabic grammar rules as a human expert would use when annotating case endings. The examples presented in this section reflect patterns we have found repeated during our analysis.

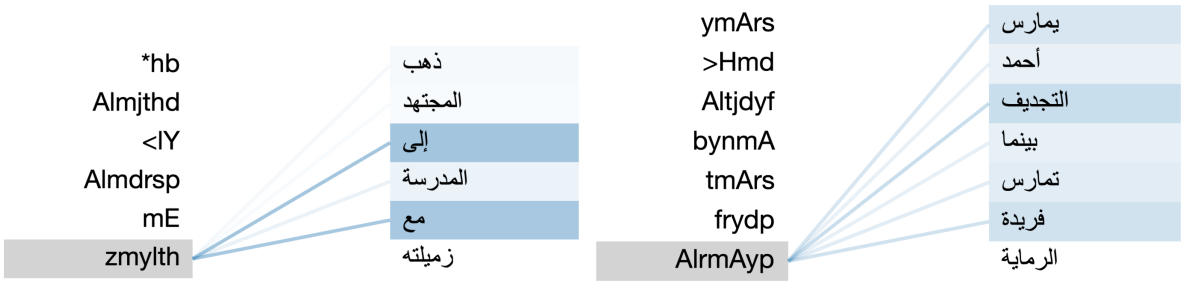


(a) Ex. of case agreement by ḥarf-atf

(b) Ex. of CE of ism-majrūr depending on ḥarf-jarr

Figure 3: Attention visualization of words correctly attending to grammatical parents.

The pattern in Figure 3a is related to the ḥarf-atf[6] rule (generally prepositions), which states that the word coming after it gets the same case as the main word of the phrase it is related to—the grammatical parent. We see indeed that the word coming after the ("w") ḥarf-atf attends the most on the word that comes before it. This is similar to what an expert would do; look at the main word in the phrase preceding the "w" in order to determine the case and case-ending of what follows.



(a) *Self* attends on a local word with similar *effect* on *Self*

(b) *Self* attends on a local word with similar *role* to *Self*

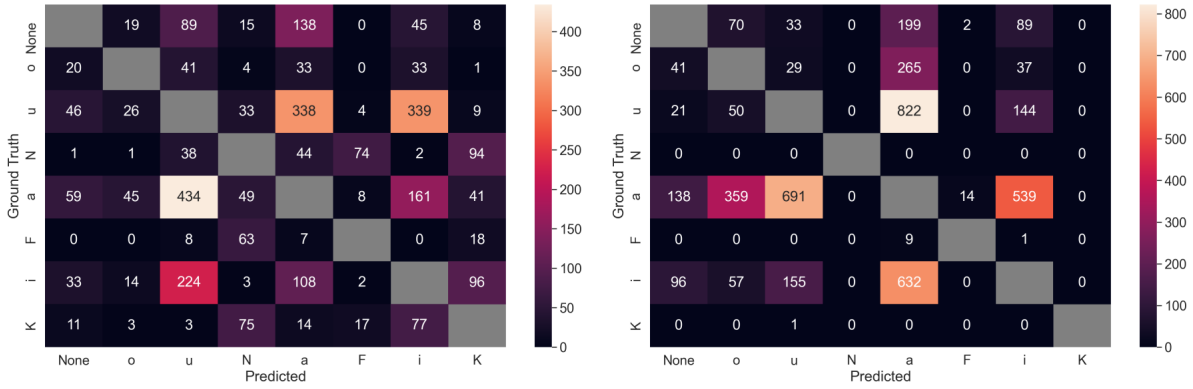
Figure 4: Attention visualization of confusion of grammatical parents.

[6] حرف عطف

Figure 3b shows another prevalent example where the word in question attends the most on the ḥarf-jarr[7] preceding it. However, in other cases where the same rule appears twice in a segment, we found that the model may choose to attend equally or more on the components of the first occurrence rather than the occurrence the current word is actually affected by—the grammatical parent. Figure 4a shows one example of this where the word ("zmylth") attends equally to two words that would affect it the same ("<lY" and "mE"), but only the second should be affecting it. In Figure 4b we see that the second maf'ool-bih[8] (roughly an object of a verb) ("AlrmAyp") attends heavily on the first occurrence of a mf'ool-bih in the segment ("Altjdyf"), rather than the verb that should be affecting it ("tmArs"). This behavior of attending on a previous word with a similar role suggests that the attention mechanism is aware of grammatical rules; it is able to group words with the same role together.

Generally, we found that not all sentences yield interpretable attention weights. We leave the task of comprehensively studying the extent of agreement of the learned weights with Arabic grammatical rules to future work.

4.3 Error Analysis



(a) Confusion Matrix for Case Endings (Errors Only)

(b) Confusion Matrix for Core Word Diacritics (Errors Only)

Figure 5: Error Confusion Matrix for CE and Core Diacritics

Figures 5a and 5b show the the confusion matrices (for visualization clarity we show errors only) of case ending and core words diacritics respectively. Many confusions are between the dammah and kasrah, and between dammah and fatḥah, and the confusion goes both ways. We analyze the errors between kasrah and dammah for case endings and try to correlate them with grammar rules.

The first error example is related to the start of a new sentence in Arabic grammar. The word "wa" composed of one letter can either mark a conjunction (e.g. in an enumeration), or mark the start of a new sentence, based on context. An example with such confusion is in the following sentence, with the ground truth: "wa yarud∼u Ealayohi >an∼a faAqida AlT∼ahuwrayoni wa naHowahu layosa lahu SalaApN <l∼aA <*aA DaAqa Alowaqotu" [9] . We see one confusion example where "wa naHow**a**h" was predicted as "wa naHow**i**h". Grammatically, the "wa" relates the next word to a "sentence starter" word ("faAqida") in a case that would make **a** the correct diacritic. Instead, we observe it follows the word immediately before the "wa" ("AlT∼ahuwr**ayoni**"), which is indeed in a grammatical case that would make **i** the correct diacritic for "naHow{**a**/i}h", were it the correct grammatical parent of this "wa".

The second example is related to the use of punctuation marks that signal an abrupt start of a new sentence or an end of one with unique context that may not be easily learnt. In the following sentence with the ground truth: "kaqaA}ilK : AloHar∼**u** >awo Alobarod**u** Al$∼adiyd**u**" [10] was predicted as

[7] حرف جر

[8] مفعول به

[9] وَيَرُدُّ عَلَيْهِ أَنَّ فَاقِدَ الطَّهُورَيْنِ وَنَحْوَهُ لَيْسَ لَهُ صَلَاةٌ إِلَّا إِذَا ضَاقَ الْوَقْتُ

[10] كَقَائِلٍ: الْحُرُّ أَوْ الْبَرْدُ الشَّدِيدُ...

“kaqaA}ilK : AloHar∼**i** >awo Alobarod**i** Al$∼adiyd**i**...”. The mark “:” here denotes the start of a new sentence, by convention, as we start a quotation. In speech, this would manifest as a brief pause or change in tone. Without “:”, the word would be an ism-majrūr that takes the kasrah ḥarakah (**i**) in this position, which the model mistakenly outputs. But because it starts a new sentence, the correct diacritic is a dammah ḥarakah (**u**). Further, the predictions for the words following “>awo” behave grammatically by following the case of the parent word (“AloHar∼**i**”) (according to “wa”), but are incorrect because the error has propagated.

One other type of errors is related to inconsistencies in the corpus—the same word with the same role in the sentence is not diacritized the same way across the dataset. For instance, the word “<lY”, which is the second top word that causes a core word error as shown in Table 6, appears multiple times in different forms: “<ilY”, “<lY”, and “<ilaY”—all correct. There are other examples that show the need to clean the dataset (at least the test set) to evaluate the published models properly.

Top words / Rank	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
With wrong CE diacritics	غير	عن	كل	بن	قلت	مثل	ثم	يوم	و	من
With wrong core diacritics	الله	إلى	ذكر	من	إلا	علم	إن	قبل	وسلّم	قوله

Table 6: Top 10 Words with CE and Core Word Errors

Looking at the top words that yield confusions in both core words and case ending diacritics, we find a notable intersection between Table 6 and the most frequent tokens in the Tashkeela corpus with an average max-normalized frequency of 0.24 (0.24 as frequent as the most frequent word).

5 Conclusion

In this work, we presented a novel architecture that outperforms previously published results on the Tashkeela Arabic diacritization benchmark. Future work may include:

- Replacing the word- and character- level Bi-LSTM encoders with transformer-based encoders.
- Using byte-pair-encoding (BPE) (Sennrich et al., 2016) to better handle suffixes and prefixes as Arabic is a moderately fusional language.
- Investigating more efficient use of injected hints to improve performance.
- Training/Evaluating this design/model on Modern Standard Arabic and dialectical benchmarks.
- Cleaning the testset of Tashkeela to remove any inconsistencies as described in the error analysis.
- Finally, achieving more interpretable attention weights through multi-task training, training on larger datasets, or otherwise.

Acknowledgements

We offer special thanks to *Khaled Essam*, as well as *Mohamed Afify* and *Ahmed Tawfik* of *Microsoft EGDC*, for many helpful discussions, suggestions and comments on the paper.

References

Hamza Abbad and Shengwu Xiong. 2020. Multi-components system for automatic arabic diacritization. In Joemon M. Jose, Emine Yilmaz, João Magalhães, Pablo Castells, Nicola Ferro, Mário J. Silva, and Flávio Martins, editors, *Advances in Information Retrieval*, pages 341–355, Cham. Springer International Publishing.

Aqil Azmi. 2013. A survey of automatic arabic diacritization techniques. *Natural Language Engineering*, 21, 10.

Zerrouki Barqawi. 2017. Shakkala, Arabic text vocalization. https://github.com/Barqawiz/Shakkala.

Yonatan Belinkov and James Glass. 2015. Arabic diacritization with recurrent neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2281–2285, Lisbon, Portugal, September. Association for Computational Linguistics.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555.

Kareem Darwish, Hamdy Mubarak, and Ahmed Abdelali. 2017. Arabic diacritization: Stats, rules, and hacks. In *Proceedings of the Third Arabic Natural Language Processing Workshop*, pages 9–17, Valencia, Spain, April. Association for Computational Linguistics.

Muhammad N. ElNokrashy. 2020. Extensible RNN cells for PyTorch. https://github.com/munael/pt-rnn.

Moustafa Elshafei, Husni Al-Muhtaseb, and Mansour Alghamdi. 2006. Statistical methods for automatic diacritization of arabic text. *The Saudi 18th National Computer Conference. Riyadh*, 18:301–306, 01.

Ali Fadel, Ibraheem Tuffaha, Bara' Al-Jawarneh, and Mahmoud Al-Ayyoub. 2019a. Arabic text diacritization using deep neural networks. In *2019 2nd International Conference on Computer Applications Information Security (ICCAIS)*, pages 1–7, May.

Ali Fadel, Ibraheem Tuffaha, Bara' Al-Jawarneh, and Mahmoud Al-Ayyoub. 2019b. Neural Arabic text diacritization: State of the art results and a novel approach for machine translation. In *Proceedings of the 6th Workshop on Asian Translation*, pages 215–225, Hong Kong, China, November. Association for Computational Linguistics.

Yarin Gal and Zoubin Ghahramani. 2016. A theoretically grounded application of dropout in recurrent neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1019–1027. Curran Associates, Inc.

Alex Graves, Santiago Fernández, and Jürgen Schmidhuber. 2005. Bidirectional LSTM Networks for Improved Phoneme Classification and Recognition. In Włodzisław Duch, Janusz Kacprzyk, Erkki Oja, and Sławomir Zadrożny, editors, *Artificial Neural Networks: Formal Models and Their Applications – ICANN 2005*, pages 799–804, Berlin, Heidelberg. Springer Berlin Heidelberg.

Nizar Habash, Owen Rambow, and Ryan Roth. 2009. MADA+TOKAN: A toolkit for Arabic tokenization, diacritization, morphological disambiguation, POS tagging, stemming and lemmatization. *Proceedings of the 2nd International Conference on Arabic Language Resources and Tools (MEDAR)*, 01.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

Hamdy Mubarak, Ahmed Abdelali, Hassan Sajjad, Younes Samih, and Kareem Darwish. 2019. Highly effective Arabic diacritization using sequence to sequence modeling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2390–2395, Minneapolis, Minnesota, June. Association for Computational Linguistics.

Rani Nelken and Stuart M. Shieber. 2005. Arabic diacritization using weighted finite-state transducers. In *Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages*, pages 79–86, Ann Arbor, Michigan, June. Association for Computational Linguistics.

Arfath Pasha, Mohamed Al-Badrashiny, Mona Diab, Ahmed El Kholy, Ramy Eskander, Nizar Habash, Manoj Pooleery, Owen Rambow, and Ryan Roth. 2014. MADAMIRA: A fast, comprehensive tool for morphological analysis and disambiguation of Arabic. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 1094–1101, Reykjavik, Iceland, May. European Language Resources Association (ELRA).

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August. Association for Computational Linguistics.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

Teresa Tinsley and Kathryn Board. 2013. *Languages for the Future*. British Council.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

Nasser Zalmout and Nizar Habash. 2020. Joint diacritization, lemmatization, normalization, and fine-grained morphological tagging. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8297–8307, Online, July. Association for Computational Linguistics.

Taha Zerrouki and Amar Balla. 2017. Tashkeela: Novel corpus of arabic vocalized texts, data for auto-diacritization systems. *Data in Brief*, 11:147 – 151.

Imed Zitouni and Ruhi Sarikaya. 2009. Arabic diacritic restoration approach based on maximum entropy models. *Computer Speech & Language*, 23:257–276, 07.

Imed Zitouni, Jeffrey S. Sorensen, and Ruhi Sarikaya. 2006. Maximum entropy based restoration of Arabic diacritics. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 577–584, Sydney, Australia, July. Association for Computational Linguistics.