Improving Arabic Diacritization with Regularized Decoding and Adversarial Training

Han Qin[♠*], **Guimin Chen**[♢*], **Yuanhe Tian**[♥*], **Yan Song**[♠♡†]
[♠]The Chinese University of Hong Kong (Shenzhen)
[♢]QTrade [♥]University of Washington
[♡]Shenzhen Research Institute of Big Data
[♠]hanqin@link.cuhk.edu.cn [♢]chenguimin@foxmail.com
[♥]yhtian@uw.edu [♠]songyan@cuhk.edu.cn

Abstract

Arabic diacritization is a fundamental task for Arabic language processing. Previous studies have demonstrated that automatically generated knowledge can be helpful to this task. However, these studies regard the auto-generated knowledge instances as gold references, which limits their effectiveness since such knowledge is not always accurate and inferior instances can lead to incorrect predictions. In this paper, we propose to use regularized decoding and adversarial training to appropriately learn from such noisy knowledge for diacritization. Experimental results on two benchmark datasets show that, even with quite flawed auto-generated knowledge, our model can still learn adequate diacritics and outperform all previous studies, on both datasets.[1]

1 Introduction

Modern standard Arabic (MSA) is generally written without diacritics, which poses a challenge to text processing and understanding in downstream applications, such as text-to-speech generation (Drago et al., 2008) and reading comprehension (Hermena et al., 2015). Restoration of such diacritics, known as diacritization, becomes an important task for Arabic natural language processing (NLP). Among different diacritization methods (Pasha et al., 2014; Shahrour et al., 2015; Zitouni et al., 2006; Habash and Rambow, 2007; Darwish et al., 2017), the neural ones (Abandah et al., 2015a; Fadel et al., 2019a,b; Zalmout and Habash, 2019, 2020; Darwish et al., 2020) achieve the best performance due to their better capability in incorporating contextual features. To further improve diacritization, automatically generated knowledge from off-the-shelf toolkits, such as morphological features, parts-of-speech tags, and automatic diacritization results, have been extensively applied to this task (Zitouni et al., 2006; Arabiyat, 2015; Darwish et al., 2017, 2020). However, current models treat such knowledge instances as gold references and always directly concatenate them with input embeddings (Arabiyat, 2015; Darwish et al., 2020), which may lead to inferior results since the knowledge may be inaccurate, especially if the toolkits were trained on data with different criteria.

Diacritization can be performed by character-based sequence labeling (Zitouni et al., 2006; Belinkov and Glass, 2015; Fadel et al., 2019b). We follow this paradigm and propose a neural approach in this paper, using regularized decoding and adversarial training to incorporate auto-generated knowledge (i.e., the diacritization results generated from off-the-shelf toolkits). Specifically, the regularized decoder treats the auto-generated knowledge as separate gold labels and learns to predict them in a separate decoding process, which is then used to update the main model. The adversarial training is applied to the encoding process by determining whether the diacritization for an input follows the gold label or the auto-generated knowledge. In doing so, our model can dynamically distinguish between auto-generated knowledge instances instead of treating them all as gold references, so as to effectively identify what knowledge should be leveraged for different inputs. Importantly, regularized decoding and adversarial training are exclusively applied to the training stage; we only need the main tagger for inference once the model has been trained. Experimental results and further analyses illustrate the effectiveness of our approach, where our model outperforms strong baselines and achieves state-of-the-art results on two benchmark datasets: Arabic Treebank (ATB) (Maamouri et al., 2004) and Tashkeela (Zerrouki and Balla, 2017).

[*]Equal contribution.
[†]Corresponding author.
[1]The code and models involved in this paper are released at https://github.com/cuhksz-nlp/AD-RDAT.

Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Short Papers), pages 534–542
August 1–6, 2021.

2 The Proposed Approach

As shown in Figure 1, our approach for diacritization follows the sequence labeling paradigm, where it has two training stages for the main tagger ($\mathcal{M}$). In the **first training stage** (presented in the orange box in Figure 1), $\mathcal{M}$ is enhanced by regularized decoding ($\mathcal{RD}$) and adversarial training ($\mathcal{AT}$) to discriminatively learn from the auto-generated labels. Specifically, given an input Arabic character sequence $\mathcal{X} = x_1 \cdots x_i \cdots x_n$, $\mathcal{M}$ and $\mathcal{RD}$ aim to predict two types of diacritization labels, $\widehat{\mathcal{Y}}$ and $\widehat{\mathcal{Y}}^K$, which follow the gold and auto-generated label criteria, respectively. $\mathcal{AT}$ ensures that the main tagger only learns useful information from either gold or auto-generated labels. Therefore, the first training stage can be conceptually formalized by

$$\widehat{\mathcal{Y}}, \widehat{\mathcal{Y}}^K = f(\mathcal{M}(\mathbf{H}^S, \mathcal{X}), \mathcal{RD}(\mathbf{H}^S, \mathcal{X}), \mathcal{AT}(\mathbf{H}^S)) \quad (1)$$

where $\mathbf{H}^S$ denotes the output vectors of the shared encoder $\mathcal{SE}$ (whose input is $\mathcal{X}$) that is designed to learn the information shared by the gold and auto-generated labels. As a result, the goal of this training stage is to minimize the loss defined by

$$\mathcal{L} = \mathcal{L}_\mathcal{M} + \mathcal{L}_\mathcal{K} + \mathcal{L}_\mathcal{A} \quad (2)$$

where $\mathcal{L}_\mathcal{M}$, $\mathcal{L}_\mathcal{K}$ and $\mathcal{L}_\mathcal{A}$ refer to the losses that come from $\mathcal{M}$, $\mathcal{RD}$, and $\mathcal{AT}$, respectively.

Afterwards, in the **second training stage** (presented in the green box in Figure 1), $\mathcal{M}$ is further trained alone on the gold labels $\widehat{\mathcal{Y}}$ without using auto-generated $\widehat{\mathcal{Y}}^K$, $\mathcal{RD}$ and $\mathcal{AT}$, to fine-tune its parameters, where all parameters in $\mathcal{SE}$ obtained through the first training stage are fixed. For inference, only $\mathcal{M}$ is used without requiring any additional input other than $\mathcal{X}$ to obtain the diacritization results. In the following sections, we first describe $\mathcal{M}$, then elaborate the details of $\mathcal{RD}$ and $\mathcal{AT}$.

2.1 The Main Tagger

The main tagger uses an encoder-decoder architecture, as shown in Figure 1, in which a shared encoder $\mathcal{SE}$ and a private encoder $\mathcal{PE}^\mathcal{M}$ are applied to model the contextual information. Particularly, $\mathcal{SE}$ is proposed to facilitate the process of leveraging auto-generated knowledge, which is excepted to learn information shared by the gold labels and the auto-generated knowledge. It takes the character embeddings of $\mathcal{X}$ (the embedding of x_i is denoted as $\mathbf{e}_i$) as input and encodes them to the shared hidden vectors (denoted as $\mathbf{h}_i^S$ for x_i) by

$$[\mathbf{h}_1^S, \cdots, \mathbf{h}_n^S] = \mathcal{SE}([\mathbf{e}_1, \cdots, \mathbf{e}_n]) \quad (3)$$

Similarly, $\mathcal{PE}^\mathcal{M}$ is also applied to the word embeddings and produces the result $\mathbf{h}_i^M$. Then, we concatenate $\mathbf{h}_i^S$ and $\mathbf{h}_i^M$ and map the resulting vector to the output space with a fully connected layer: $\mathbf{o}_i^M = \mathbf{W}^M \cdot (\mathbf{h}_i^M \oplus \mathbf{h}_i^S) + \mathbf{b}^M$, where $\oplus$ is concatenation and $\mathbf{W}^M$ and $\mathbf{b}^M$ are the trainable matrices and bias vector, respectively. Finally, a *softmax* decoder is applied to $\mathbf{o}_i^M$ to predict the label $\widehat{y}_i$:

$$\widehat{y}_i = \arg\max \frac{exp(o_i^{M,t})}{\sum_{t=1}^{|\mathcal{T}|} exp(o_i^{M,t})}, \quad (4)$$

where $\mathcal{T}$ denotes the set of all diacritization labels and $o_i^{M,t}$ is the value at dimension t in $\mathbf{o}_i^M$. Therefore the loss for $\mathcal{M}$ is

$$\mathcal{L}_\mathcal{M} = -\sum_{i=1}^{n} \log p(y_i^* | \mathcal{X}), \quad (5)$$

where $p(y_i^* | \mathcal{X})$ denotes the probability of labeling x_i by the gold label y_i^*.

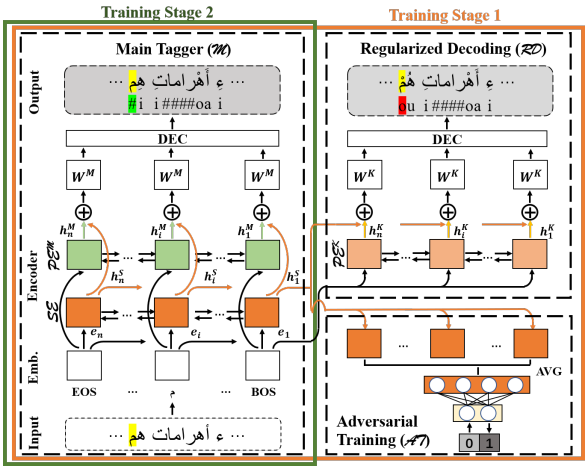

Figure 1: The architecture of our model, where the left shows the main tagger ($\mathcal{M}$) and the right shows the regularized decoding ($\mathcal{RD}$) and adversarial training ($\mathcal{AT}$) modules. The diacritization labels for an example following different criteria are illustrated in $\mathcal{M}$ and $\mathcal{RD}$, with the mismatching labels marked in green and red. E.g., for "م" (highlighted in yellow), its gold and auto-generated labels are "#" (*null*) and "o" (*sukun*).[2]

2.2 Regularized Decoding

When leveraging auto-generated knowledge, it is important to note that such knowledge may be inaccurate or follow different annotation criteria, which is required to be appropriately addressed to pre-

[2]We use a set of symbols to label different diacritization results, which are illustrated in Appendix A.

	ATB					Tashkeela				
	w/ case ending		w/o case ending		ACC	w/ case ending		w/o case ending		ACC
	DER	WER	DER	WER		DER	WER	DER	WER	
BiLSTM	2.28	6.62	1.98	4.15	93.38	2.59	7.62	2.30	5.01	92.98
+$\mathcal{RD}$	2.12	5.90	1.72	3.37	94.10	2.18	6.42	1.87	4.04	94.09
+$\mathcal{RD}$+$\mathcal{AT}$	1.87	5.17	1.59	3.09	94.83	2.10	6.08	1.82	3.88	94.40
Transformer	2.22	6.36	1.92	4.00	93.64	2.70	7.98	2.43	5.37	92.65
+$\mathcal{RD}$	2.07	5.90	1.70	3.45	94.10	2.11	6.11	1.82	3.85	94.37
+$\mathcal{RD}$+$\mathcal{AT}$	**1.83**	**5.09**	**1.56**	**3.07**	**94.91**	**2.06**	**5.98**	**1.76**	**3.75**	**94.49**

(a) AraBERT

	ATB					Tashkeela				
	w/ case ending		w/o case ending		ACC	w/ case ending		w/o case ending		ACC
	DER	WER	DER	WER		DER	WER	DER	WER	
BiLSTM	2.15	6.16	1.81	3.73	93.84	2.48	7.33	2.19	4.79	93.27
+$\mathcal{RD}$	1.97	5.65	1.69	3.48	94.35	2.08	6.02	1.83	3.91	94.50
+$\mathcal{RD}$+$\mathcal{AT}$	1.81	5.06	1.53	3.02	94.94	2.03	5.86	1.77	3.75	94.69
Transformer	2.05	5.80	1.77	3.61	94.20	2.66	7.65	2.33	4.99	92.95
+$\mathcal{RD}$	1.85	5.11	1.56	3.02	94.89	1.96	5.62	1.67	**3.37**	94.86
+$\mathcal{RD}$+$\mathcal{AT}$	**1.77**	**4.88**	**1.49**	**3.01**	**95.12**	**1.87**	**5.54**	**1.56**	3.64	**94.94**

(b) ZEN 2.0

Table 1: Experimental results (i.e., DER and WER with and without the case ending being considered and accuracy) of baselines and our models with $\mathcal{RD}$ and $\mathcal{AT}$ using AraBERT (a) and ZEN 2.0 (b) on the test sets of ATB and Tashkeela, "BiLSTM" and "Transformer" denote the encoders (i.e., $\mathcal{SE}$ and $\mathcal{PE}$) used in the models.

vent the noise in the auto-generated knowledge from significantly hurting the model performance (Tang et al., 2020; Nie et al., 2020; Chen et al., 2020; Mandya et al., 2020; Tian et al., 2020a,b, 2021a,b; Chen et al., 2021). To tackle this challenge, we propose to learn from a special decoding process, which is integrated into the main diacritization model, in order to reduce error propagation compared to directly using the knowledge instances or their features. As shown in Figure 1, the proposed regularized decoding is an extra output process separated from the main tagger and performed on another sequence of labels $\mathcal{Y}^{K^*}$, which are the auto-generated knowledge instances (diacritization labels) annotated by an existing toolkit. Therefore, the loss $\mathcal{L}_\mathcal{K}$ from $\mathcal{RD}$ is computed through

$$\mathcal{L}_\mathcal{K} = -\sum_{i=1}^{n} \log p(y_i^{K^*}|\mathcal{X}) \quad (6)$$

and in the first training stage, all trainable parameters in $\mathcal{SE}$ are updated through the information back-propagated from $\mathcal{RD}$.

2.3 Adversarial Training

Although auto-generated knowledge can be back-propagated through $\mathcal{RD}$, it could be overwhelmed by the information directly learned from the gold label. We further improve our model by balancing the information learned from both $\mathcal{M}$ and $\mathcal{RD}$ with $\mathcal{AT}$, which is proposed to equalize both sides and emphasize the shared information from them.[3] In doing so, we connect a discriminator, which is a binary classifier, to $\mathcal{SE}$. The discriminator takes all $\mathbf{h}_i^S$ from $\mathcal{SE}$, averages them by $\mathbf{h}^S = \frac{1}{n}\sum_{i=1}^{n}\mathbf{h}_i^S$, and then passes the resulting vector to a fully connected layer with a *softmax* function to compute its bias towards either type (i.e., the gold or auto-generated) of diacritization labels:

$$[p_m, p_k] = softmax(\mathbf{W}^D \cdot \mathbf{h}^S + \mathbf{b}^D) \quad (7)$$

where $\mathbf{W}^D$ and $\mathbf{b}^D$ are the trainable matrix and bias vector, respectively, that map $\mathbf{h}^S$ to a two-dimensional vector, with p_m and p_k representing normalized probabilities that satisfy $p_m + p_k = 1$ and indicating the bias of $\mathcal{SE}$ towards gold and auto-generated labels, respectively. Then we apply a negative log-likelihood loss function to the discriminator, formalized as

$$\mathcal{L}_\mathcal{D} = -\log p_m - \log p_k \quad (8)$$

and an adversarial loss to the parameters in $\mathcal{SE}$ via

$$\mathcal{L}_\mathcal{S} = -p_m \log p_m - p_k \log p_k \quad (9)$$

As a result, the goal of $\mathcal{AT}$ is to minimize the loss

$$\mathcal{L}_\mathcal{A} = \mathcal{L}_\mathcal{D} - \lambda\mathcal{L}_\mathcal{S} \quad (10)$$

[3] $\mathcal{AT}$ follows the idea that the $\mathcal{SE}$ should have no bias towards the information learned from $\mathcal{M}$ and $\mathcal{RD}$.

	ATB					Tashkeela				
	w/ case ending		w/o case ending		ACC	w/ case ending		w/o case ending		ACC
	DER	WER	DER	WER		DER	WER	DER	WER	
Fadel et al. (2019a)	-	-	-	-	-	3.73	11.19	2.88	6.53	-
Abandah and Abdel-Karim (2019)	2.46	8.12	1.24	3.81	-	1.97	5.13	1.22	3.13	-
Fadel et al. (2019b)	-	-	-	-	-	2.60	7.69	2.11	4.57	-
Alqahtani et al. (2019)	2.80	8.20	-	-	-	-	-	-	-	-
Alqahtani et al. (2020)	2.54	7.51	-	-	-	-	-	-	-	-
Zalmout and Habash (2020)	-	-	-	-	93.90	-	-	-	-	-
Farasa	19.84	68.61	20.31	68.48	31.39	22.00	58.66	24.89	53.14	45.96
Ours (AraBERT) (BiLSTM)	1.87	5.17	1.59	3.09	94.83	2.10	6.08	1.82	3.88	94.40
Ours (AraBERT) (Transformer)	**1.83**	**5.09**	**1.56**	**3.07**	**94.91**	**2.06**	**5.98**	**1.76**	**3.75**	**94.49**
Ours (ZEN 2.0) (BiLSTM)	1.81	5.06	1.53	3.02	94.94	2.03	5.86	1.77	3.75	94.69
Ours (ZEN 2.0) (Transformer)	**1.77**	**4.88**	**1.49**	**3.01**	**95.12**	**1.87**	**5.54**	**1.56**	**3.64**	**94.94**

Table 2: Comparisons of experimental results (i.e., DER, WER, and accuracy) between previous studies and our models with AraBERT and ZEN 2.0 embeddings on the test sets of the ATB and Tashkeela.

where λ is a positive coefficient that controls the influence of $\mathcal{L}_{\mathcal{S}}$ in the adversarial training, so that to minimize $\mathcal{L}_{\mathcal{D}}$ and maximize $\mathcal{L}_{\mathcal{S}}$ synchronously.

3 Experiments

3.1 Settings

In our experiments, We use two benchmark datasets, i.e., ATB (Arabic Treebank Part 1, 2, and 3) (Maamouri et al., 2004) and Tashkeela (Zerrouki and Balla, 2017), following the same settings in previous studies.[4] For implementation, we run Farasa[5] (Abdelali et al., 2016) on the two datasets and collect their diacritization results for regularized decoding. Since the quality of text representation normally dominates the model performance (Pennington et al., 2014; Song et al., 2017, 2018; Peters et al., 2018; Song and Shi, 2018; Devlin et al., 2019), in our experiments, we test two types of widely used and powerful encoders, i.e., BiLSTM and Transformer (Vaswani et al., 2017), for $\mathcal{SE}$ and $\mathcal{PE}$. For the embeddings, we use AraBERT (Antoun et al., 2020) and the large version of ZEN 2.0 (Song et al., 2021) with their default settings (i.e. 12 layers of multi-head attentions with 768 dimensional hidden vectors for AraBERT and 24 layers of multi-head attentions with 1024 dimensional hidden vectors for ZEN 2.0) to perform the initialization (we use the output of the last layer).[6] We train our model for 20 epochs in total, with the first 10 for the first training stage and the rest for the second stage. Particularly, in the second training stage, we evaluate our model on the development set for every 100 steps to locate the best performing model. For evaluation, we follow previous studies (Abandah et al., 2015b; Fadel et al., 2019b) to use diacritization error rate (DER) and word error rate (WER) with and without considering the case ending.[7] We also use diacritization accuracy following Zalmout and Habash (2017, 2019, 2020).[8]

3.2 Overall Results

In the main experiment, we run the baselines and our models using different configurations (i.e., using AraBERT or ZEN 2.0 embeddings and using BiLSTM or Transformer encoders) with and without $\mathcal{RD}$ and $\mathcal{AT}$. The experimental results (DER and WER with and without considering the case endings, and accuracy) on the test sets of ATB and Tashkeela are reported in Table 1.[9]

There are several observations. First, under different configurations (i.e., using AraBERT or ZEN 2.0 and with BiLSTM or Transformer encoders), $\mathcal{RD}$ improves the baseline on both datasets, which shows that $\mathcal{RD}$ is effective to help diacritization with auto-generated knowledge even if they follow different criterion. Second, further consistent improvement can be observed when $\mathcal{AT}$ is applied on top of $\mathcal{RD}$, with only 3K (0.015‰ of the entire model size) more trainable parameters required to achieve this effect.[10] These observations confirm the effectiveness of forcing $\mathcal{SE}$ to learn from the information shared by gold and auto-generated labels with an appropriate model design.

[4] We illustrate the dataset details in Appendix B.

[5] http://qatsdemo.cloudapp.net/farasa/

[6] We obtain the pre-trained official AraBERT model from https://github.com/aub-mind/arabert and the Arabic version of ZEN 2.0 (large) from https://github.com/sinovation/ZEN2.

[7] We show details of DER and WER in Appendix C.

[8] We report the hyper-parameter settings of different models and the best combinations of them in Appendix D.

[9] Their dev set's results and the mean and standard deviation of test set results are reported in Appendix E and F.

[10] Model sizes are reported in Appendix G.

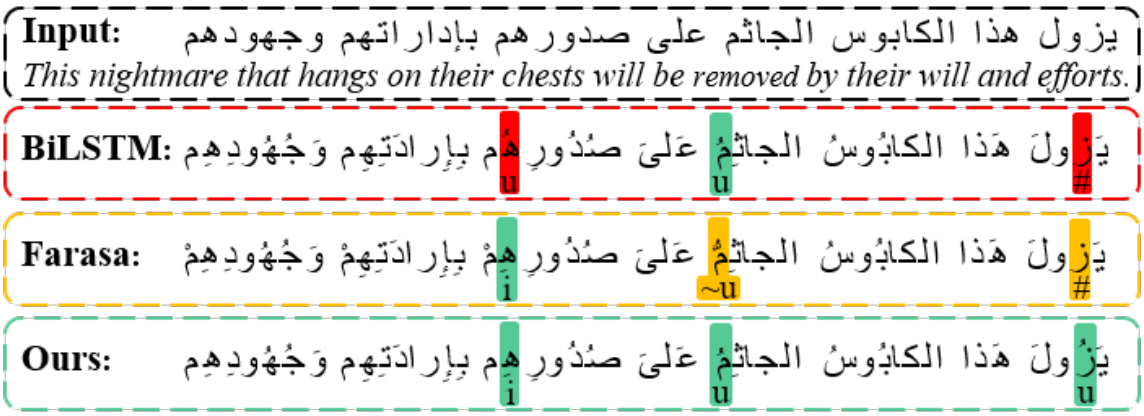

Figure 2: An example input sentence and its diacritization results ("$\sim u$", "#", "i" and "u") from Farasa, BiLSTM, and our approach (BiLSTM+$\mathcal{RD}$+$\mathcal{AT}$) with AraBERT. All results matching gold labels are highlighted in green; the mismatching results from Farasa and BiLSTM are in orange and red, respectively.

In addition, we also compare the results of our best models (with $\mathcal{RD}$ and $\mathcal{AT}$) with previous studies (including Farasa's results) on the test sets of both datasets. The results are shown in Table 2, where our model with BiLSTM encoder outperforms previous models and achieves state-of-the-art performance on both datasets.

3.3 Case Study

To explore how our approach with $\mathcal{RD}$ and $\mathcal{AT}$ leverage auto-generated knowledge, we conduct a case study on an example sentence from the test set of ATB. The input and its diacritization results from Farasa, the BiLSTM baseline, and our approach with AraBERT (BiLSTM+$\mathcal{RD}$+$\mathcal{AT}$) are illustrated in Figure 2, where the correct diacritization results are highlighted in green, and the incorrect ones from Farasa and BiLSTM are highlighted in orange and red, respectively. It is clearly observed that our approach leverages the necessary information learned from Farasa (i.e., the "$\sim u$" label) and prevents its unreliable results from affecting the final diacritics. Specifically, for the highlighted Arabic character "ه", where the Farasa output suggests the diacritic "i" (*kasra*), our approach leverages this knowledge and corrects the BiLSTM baseline. For the other two highlighted characters, although the Farasa output (i.e., "$\sim u$" (*Shadda+Damma*) for "م" and "#" (*No Diacritic*) for "ز") also produces diacritization results that are different from the BiLSTM baseline and do not match the gold standard, our approach is able to learn from their patterns and make correct predictions. Therefore, although Farasas output does not match the gold labels in most cases (see the Farasa results in Table 2), the proposed $\mathcal{RD}$ and $\mathcal{AT}$ can leverage such knowledge and improve the main tagger accordingly.

4 Conclusion

In this paper, we propose to incorporate auto-generated knowledge (diacritization labels in another criterion) for Arabic diacritization with regularized decoding and adversarial training. In detail, the regularized decoding treats the auto-generated knowledge as separate "gold" labels and learns to predict them in another decoding process; the adversarial training is used to ensure that the shared information from gold and auto-generated labels are learned to help diacritization. With the regularized decoding and adversarial training, the main tagger in our approach is able to smartly leverage auto-generated knowledge provided by an existing diacritization tagger. Experimental results on two benchmark datasets illustrate the validity and effectiveness of our model, where state-of-the-art performance is obtained on both datasets.

Acknowledgements

This work is supported by Shenzhen Institute of Artificial Intelligence and Robotics for Society under the project "Automatic Knowledge Enhanced Natural Language Understanding and Its Applications" (AC01202101001).

References

Gheith Abandah and Asma Abdel-Karim. 2019. Accurate and Fast Recurrent Neural Network Solution for the Automatic Diacritization of Arabic Text. *Jordanian Journal of Computers and Information Technology*, 6.

Gheith A Abandah, Alex Graves, Balkees Al-Shagoor, Alaa Arabiyat, Fuad Jamour, and Majid Al-Taee. 2015a. Automatic Diacritization of Arabic Text using Recurrent Neural Networks. *International Journal on Document Analysis and Recognition (IJDAR)*, 18(2):183–197.

Gheith A. Abandah, Alex Graves, Balkees Al-Shagoor, Alaa Arabiyat, Fuad T. Jamour, and Majid A. Al-Taee. 2015b. Automatic diacritization of Arabic text using recurrent neural networks. *International Journal on Document Analysis and Recognition (IJDAR)*, 18:183–197.

Ahmed Abdelali, Kareem Darwish, Nadir Durrani, and Hamdy Mubarak. 2016. Farasa: A Fast and Furious Segmenter for Arabic. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 11–16, San Diego, California.

Sawsan Alqahtani, Ajay Mishra, and Mona Diab. 2019. Efficient convolutional neural networks for diacritic restoration. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 1442–1448.

Sawsan Alqahtani, Ajay Mishra, and Mona Diab. 2020. A multitask learning approach for diacritic restoration. *arXiv preprint arXiv:2006.04016*.

Wissam Antoun, Fady Baly, and Hazem M. Hajj. 2020. AraBERT: Transformer-based Model for Arabic Language Understanding. *ArXiv*, abs/2003.00104.

Alaa Khaled Radwan Arabiyat. 2015. Automatic Arabic Text Diacritization Using Recurrent Neural Networks. Master's thesis, The University of Jordan.

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.

Guimin Chen, Yuanhe Tian, and Yan Song. 2020. Joint Aspect Extraction and Sentiment Analysis with Directional Graph Convolutional Networks. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 272–279.

Guimin Chen, Yuanhe Tian, Yan Song, and Xiang Wan. 2021. Relation Extraction with Type-aware Map Memories of Word Dependencies. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*.

Kareem Darwish, Ahmed Abdelali, Hamdy Mubarak, and Mohamed Eldesouki. 2020. Arabic Diacritic Recovery Using a Feature-Rich bilstm Model. *ArXiv*, abs/2002.01207.

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. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 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 4171–4186.

Mona Diab, Nizar Habash, Owen Rambow, and Ryan Roth. 2013. Ldc arabic treebanks and associated corpora: Data divisions manual. *arXiv preprint arXiv:1309.5652*.

Drago, Burileanu, Vladimir Popescu, Cristian Negrescu, and Aurelian Dervi. 2008. Automatic Diacritic Restoration for a Tts-based E-mail Reader Application.

Ali Fadel, Ibraheem Tuffaha, Mahmoud Al-Ayyoub, et al. 2019a. Arabic Text Diacritization Using Deep Neural Networks. In *2019 2nd International Conference on Computer Applications & Information Security (ICCAIS)*, pages 1–7.

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.

Nizar Habash and Owen Rambow. 2007. Arabic Diacritization through Full Morphological Tagging. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 53–56, Rochester, New York.

Ehab Hermena, Denis Drieghe, Sam Hellmuth, and Simon Liversedge. 2015. Processing of Arabic Diacritical Marks: Phonological-Syntactic Disambiguation of Homographic Verbs and Visual Crowding Effects. *Journal of experimental psychology. Human perception and performance*, 41.

Mohamed Maamouri, Ann Bies, Tim Buckwalter, and Wigdan Mekki. 2004. The Penn Arabic Treebank : Building a Large-Scale Annotated Arabic Corpus.

Angrosh Mandya, Danushka Bollegala, and Frans Coenen. 2020. Graph Convolution over Multiple Dependency Sub-graphs for Relation Extraction. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6424–6435.

Yuyang Nie, Yuanhe Tian, Yan Song, Xiang Ao, and Xiang Wan. 2020. Improving Named Entity Recognition with Attentive Ensemble of Syntactic Information. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4231–4245.

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.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep Contextualized Word Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana.

Anas Shahrour, Salam Khalifa, and Nizar Habash. 2015. Improving Arabic Diacritization through Syntactic Analysis. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1309–1315.

Yan Song, Chia-Jung Lee, and Fei Xia. 2017. Learning Word Representations with Regularization from Prior Knowledge. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 143–152.

Yan Song and Shuming Shi. 2018. Complementary Learning of Word Embeddings. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 4368–4374.

Yan Song, Shuming Shi, and Jing Li. 2018. Joint Learning Embeddings for Chinese Words and Their Components via Ladder Structured Networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 4375–4381.

Yan Song, Tong Zhang, Yonggang Wang, and Kai-Fu Lee. 2021. ZEN 2.0: Continue Training and Adaption for N-gram Enhanced Text Encoders. *arXiv preprint arXiv:2105.01279*.

Hao Tang, Donghong Ji, Chenliang Li, and Qiji Zhou. 2020. Dependency Graph Enhanced Dual-transformer Structure for Aspect-based Sentiment Classification. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6578–6588.

Yuanhe Tian, Guimin Chen, and Yan Song. 2021a. Aspect-based Sentiment Analysis with Type-aware Graph Convolutional Networks and Layer Ensemble. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2910–2922, Online.

Yuanhe Tian, Guimin Chen, Yan Song, and Xiang Wan. 2021b. Dependency-driven Relation Extraction with Attentive Graph Convolutional Networks. In *Proceedings of the Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*.

Yuanhe Tian, Wang Shen, Yan Song, Fei Xia, Min He, and Kenli Li. 2020a. Improving Biomedical Named Entity Recognition with Syntactic Information. *BMC Bioinformatics*, 21:1471–2105.

Yuanhe Tian, Yan Song, and Fei Xia. 2020b. Joint Chinese Word Segmentation and Part-of-speech Tagging via Multi-channel Attention of Character N-grams. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2073–2084.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008.

Nasser Zalmout and Nizar Habash. 2017. Don't throw Those Morphological Analyzers Away Just Yet: Neural Morphological Disambiguation for Arabic. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 704–713, Copenhagen, Denmark.

Nasser Zalmout and Nizar Habash. 2019. Adversarial multitask learning for joint multi-feature and multi-dialect morphological modeling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1775–1786, Florence, Italy.

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.

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, 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.

Appendix A. Diactritization Labels

Table 3 presents the 15 diacritization labels used in our study, following Fadel et al. (2019b).

Class Label	Class Name	Class Shape	Class Label	Class Name	Class Shape
#	No diacritics	بـ	~	Shadda	بّ
a	Fatha	بَ	~a	Shadda+Fatha	بَّ
i	Kasra	بِ	~i	Shadda+Kasra	بِّ
o	Sukun	بْ	~u	Shadda+Damma	بُّ
u	Damma	بُ	~K	Shadda+Kasratan	بٍّ
K	Kasratan	بٍ	~F	Shadda+Fathatan	بًّ
F	Fathatan	بً	~N	Shadda+Dammatan	بٌّ
N	Dammatan	بٌ			

Table 3: Diactritization labels used in this study.

Appendix B. Datasets

In our experiments, We use two benchmark datasets, i.e., ATB (Arabic Treebank Part 1, 2, and 3)[11] (Maamouri et al., 2004) and Tashkeela[12] (Zerrouki and Balla, 2017). For ATB, we follow the same data split policy as Diab et al. (2013), which is based on the 10-80-10 rule. That is, we firstly split each part of ATB into three portions (with each portion containing 10%, 80%, and 10% of documents, respectively). Then, we combine the first, second, and third portion of all three parts to form the development, training, and test set, respectively. For Tashkeela, we use the cleaned version[13] from Fadel et al. (2019b) with the standard train/dev/test split. The statistics of the datasets in terms of the number of words, lines, and the average number of characters in each word are reported in Table 4.

	ATB			Tashkeela		
	Train	Dev	Test	Train	Dev	Test
Word #	503K	63K	63K	2.1M	102K	107K
Line #	15.7K	1.9K	1.9K	50K	2.5K	2.5K
C/W	4.37	4.31	4.35	3.97	3.97	3.97

Table 4: Statistics of the benchmark datasets, where the number of words and lines, and average characters per word (C/W) are reported.

[11] We download the ATB part 1, 2 and 3 are from https://catalog.ldc.upenn.edu/LDC2010T13, https://catalog.ldc.upenn.edu/LDC2011T09 and https://catalog.ldc.upenn.edu/LDC2010T08.

[12] https://github.com/AliOsm/arabic-text-diacritization/tree/master/dataset

[13] We download the data from https://github.com/AliOsm/arabic-text-diacritization/tree/master/dataset.

Appendix C. Evaluation of DER and WER

It is worth noting that previous studies (Zitouni et al., 2006; Arabiyat, 2015; Fadel et al., 2019a; Abandah and Abdel-Karim, 2019; Alqahtani et al., 2019, 2020) use different methods to compute diacritic error rate (DER) for ATB and Tashikeela datasets. Therefore, we follow the schema in Zitouni et al. (2006); Arabiyat (2015); Abandah and Abdel-Karim (2019) to compute DER for ATB and follow Fadel et al. (2019a); Alqahtani et al. (2019, 2020) to compute that for Tashikeela.

Specifically, for ATB, we compute DER by: (1) all words are counted including numbers and punctuators; (2) each latter or digit in a word is a potential host for a set of diacritics; and (3) all diacritics on a single letter are counted as a single binary (True or False) choice. For Tashikeela, the schema is similar to the one for ATB but all non-Arabic letters are ignored in computing DER because they do not hold a diacritic. For word error rate (WER), the way to compute it is identical for both datasets, where the diacritization result for an Arabic word is regarded as incorrect if there is s at least one incorrectly restored diacritic. We follow previous studies (Abandah et al., 2015b; Fadel et al., 2019a) to evaluate our results in terms of diacritic error rate (DER) and word error rate (WER). We use the implementation[14] provided by Fadel et al. (2019a) to compute DER (with two criteria) and WER of different models on both datasets, where the DER and WER with and without considering the case endings are both included in our evaluation.

Appendix D. Hyper-parameter Settings

Table 5 reports the hyper-parameters tested in training our models. We test all combinations of them for each model and use the one achieving the highest F1 score in our final experiments.

Hyper-parameters	Values
Learning Rate	$1e-5, \mathbf{3e-5}, 5e-5$
Warmup Rate	**0.06**
Dropout Rate	**0.1**
Batch Size	$16, \mathbf{32}, 64$

Table 5: The hyper-parameters tested in tuning our models. The best ones used in our final experiments are highlighted in boldface.

[14] https://github.com/AliOsm/arabic-text-diacritization/blob/master/helpers/diacritization_stat.py.

Appendix E. Experimental Results on the Development Set

Table 6 reports the DER and WER (with case ending) of different models evaluated on the development set of ATB and Tashkeela.

	ATB		Tashkeela	
	DER	WER	DER	WER
BiLSTM	2.52	7.00	2.58	7.66
+$\mathcal{RD}$	2.46	6.47	2.20	6.47
+$\mathcal{RD}$+$\mathcal{AT}$	2.14	5.65	2.12	6.30
Transformer	2.46	6.79	2.71	7.96
+$\mathcal{RD}$	2.35	6.28	2.07	6.08
+$\mathcal{RD}$+$\mathcal{AT}$	2.09	5.50	2.05	6.03

(a) AraBERT

	ATB		Tashkeela	
	DER	WER	DER	WER
BiLSTM	2.41	6.67	2.51	7.45
+$\mathcal{RD}$	2.21	6.00	2.09	6.16
+$\mathcal{RD}$+$\mathcal{AT}$	2.03	5.43	2.29	6.22
Transformer	2.39	6.49	2.65	7.79
+$\mathcal{RD}$	2.03	5.40	2.21	6.09
+$\mathcal{RD}$+$\mathcal{AT}$	1.96	5.24	2.01	5.87

(b) ZEN 2.0

Table 6: DER and WER (with case ending) of models with different configurations (i.e., based on BiLSTM and Transformer) evaluated on the development set of ATB and Tashkeela.

Appendix F. Mean and Deviation of the Results

In the experiments, we test models with different configurations. For each model, we train it with the best hyper-parameter setting using five different random seeds. We report the mean (μ) and standard deviation (σ) of DER and WER (with case ending) on the test set of ATB and Tashkeela in Table 7.

Appendix G. Model Size and Running Speed

Table 8 reports the number of trainable parameters and the inference speed (lines per second) of the baseline (i.e., BiLSTM and Transformer encoder with and without regularized decoding ($\mathcal{RD}$)) and our models with both $\mathcal{RD}$ and adversarial training ($\mathcal{AT}$) on ATB and Tashkeela. All models are performed on NVIDIA Quadro RTX 6000 GPUs.

Models	ATB				Tashkeela			
	DER		WER		DER		WER	
	μ	σ	μ	σ	μ	σ	μ	σ
BiLSTM	2.33	0.0115	6.70	0.4082	2.61	0.0004	7.67	0.0020
+$\mathcal{RD}$	2.14	0.0183	5.92	0.1669	2.30	0.0114	6.79	0.1022
+$\mathcal{RD}$+$\mathcal{AT}$	1.94	0.0053	5.38	0.0589	2.21	0.0080	6.40	0.0683
Transformer	2.27	0.0413	6.58	0.5590	2.84	0.0537	8.15	0.3696
+$\mathcal{RD}$	2.10	0.0008	5.92	0.0122	2.14	0.0003	5.96	0.1934
+$\mathcal{RD}$+$\mathcal{AT}$	1.88	0.0026	5.29	0.0651	2.12	0.0024	6.20	0.0323

(a) AraBERT

Models	ATB				Tashkeela			
	DER		WER		DER		WER	
	μ	σ	μ	σ	μ	σ	μ	σ
BiLSTM	2.19	0.0374	6.23	0.0613	2.53	0.0411	7.42	0.0736
+$\mathcal{RD}$	1.99	0.0170	5.66	0.0450	2.12	0.0419	6.04	0.0510
+$\mathcal{RD}$+$\mathcal{AT}$	1.85	0.0327	5.15	0.0988	2.08	0.0408	5.95	0.0665
Transformer	2.08	0.0249	5.90	0.0860	2.69	0.0205	7.70	0.0411
+$\mathcal{RD}$	1.87	0.0205	5.15	0.0327	2.02	0.0531	5.78	0.1307
+$\mathcal{RD}$+$\mathcal{AT}$	1.80	0.0249	4.96	0.0655	1.93	0.0490	5.65	0.0829

(b) ZEN 2.0

Table 7: The mean μ and standard deviation σ of DER and WER (with case ending) of all models (i.e., based on BiLSTM or Transformer with $\mathcal{RD}$ and $\mathcal{AT}$) on the test set of ATB and Tashkeela for Arabic diacritization.

	ATB		Tashkeela	
	Para.	Speed	Para.	Speed
BiLSTM	158,840K	55.6	158,840K	54.3
+$\mathcal{RD}$	206,162K	36.4	206,162K	28.5
+$\mathcal{RD}$+$\mathcal{AT}$	206,165K	25.6	206,165K	22.9
Transformer	146,235K	70.7	146,235K	71.1
+$\mathcal{RD}$	168,335K	37.7	168,335K	34.3
+$\mathcal{RD}$+$\mathcal{AT}$	168,337K	29.0	168,337K	27.2

(a) AraBERT

	ATB		Tashkeela	
	Para.	Speed	Para.	Speed
BiLSTM	839,026K	30.2	839,026K	29.8
+$\mathcal{RD}$	872,730K	20.4	872,730K	19.7
+$\mathcal{RD}$+$\mathcal{AT}$	872,734K	14.8	872,734K	13.3
Transformer	830,612K	39.6	830,612K	37.2
+$\mathcal{RD}$	847,471K	25.8	847,471K	24.0
+$\mathcal{RD}$+$\mathcal{AT}$	847,473K	21.1	847,473K	19.4

(b) ZEN 2.0

Table 8: Numbers of trainable parameters (Para.) in different models and the inference speed (sentences per second) of these models on the test sets of both datasets. $\mathcal{RD}$ and $\mathcal{AT}$ represent the proposed regularized decoding and adversarial training, respectively.