Neural Arabic Text Diacritization: State of the Art Results and a Novel Approach for Machine Translation

Ali Fadel, Ibraheem Tuffaha, Bara' Al-Jawarneh, and Mahmoud Al-Ayyoub

Jordan University of Science and Technology, Irbid, Jordan
{aliosm1997, bro.t.1996, baraaaljawarneh, malayyoub}@gmail.com

Abstract

In this work, we present several deep learning models for the automatic diacritization of Arabic text. Our models are built using two main approaches, viz. Feed-Forward Neural Network (FFNN) and Recurrent Neural Network (RNN), with several enhancements such as 100-hot encoding, embeddings, Conditional Random Field (CRF) and Block-Normalized Gradient (BNG). The models are tested on the only freely available benchmark dataset and the results show that our models are either better or on par with other models, which require language-dependent post-processing steps, unlike ours. Moreover, we show that diacritics in Arabic can be used to enhance the models of NLP tasks such as Machine Translation (MT) by proposing the *Translation over Diacritization* (ToD) approach.

1 Introduction

In Arabic and many other languages, diacritics are added to the characters of a word (as short vowels) in order to convey certain information about the meaning of the word as a whole and its place within the sentence. Arabic Text Diacritization (ATD) is an important problem with various applications such as text to speech (TTS). At the same time, this problem is a very challenging one even to native speakers of Arabic due to the many subtle issues in determining the correct diacritic for each character from the list shown in Figure 2 and the lack of practice for many native speakers. Thus, the need to build automatic Arabic text diacritizers is high (Zitouni and Sarikaya, 2009).

The meaning of a sentence is greatly influenced by the diacritization which is determined by the context of the sentence as shown in the following example:

كلم أحمد ...
Buckwalter Transliteration: klm >Hmd ...
Incomplete sentence without diacritization.

كَلَّمَ أَحْمَدٌ صَدِيقَهُ
Buckwalter Transliteration: kal~ama >aHomadN Sadiyqahu
Translation: Ahmad talked to his friend.

كَلَمَ أَحْمَدٌ عَدُوَّهُ
Buckwalter Transliteration: kalama >aHomadN Eaduw~ahu
Translation: Ahmad wounded his enemy.

The letters كلم "klm" manifests into two different words when given two different diacritizations. As shown in this example, كَلَّمَ "kal~ama" in the first sentence is the verb 'talked' in English, while كَلَمَ "kalama" in the second sentence is the verb 'wounded' in English.

To formulate the problem in a formal manner: Given a sequence of characters representing an Arabic sentence S, find the correct diacritic class (from Figure 2) for each Arabic character S_i in S.

Despite the problem's importance, it received limited attention. One of the reasons for this is the scarcity of freely available resources for this problem. To address this issue, the Tashkeela Corpus[1] (Zerrouki and Balla, 2017) has been released to the community. Unfortunately, there are many problems with the use of this corpus for benchmarking purposes. A very recent study (Fadel et al., 2019) discussed in details these issues and provided a cleaned version of the dataset with predefined split into training, testing and validation sets. In this work, we use this dataset and provide yet another extension of it with a larger training set and a new testing set to circumvent the issue that some of the existing systems have already be-

[1] https://sourceforge.net/projects/tashkeela

Proceedings of the 6th Workshop on Asian Translation, pages 215–225
Hong Kong, China, November 4, 2019.

en trained on the entire Tashkeela Corpus.

According to (Fadel et al., 2019), existing approaches to ATD are split into two groups: traditional rule-based approaches and machine learning based approaches. The former was the main approach by many researchers such as (Zitouni and Sarikaya, 2009; Pasha et al., 2014; Darwish et al., 2017) while the latter has started to receive attention only recently (Belinkov and Glass, 2015; Abandah et al., 2015; Barqawi and Zerrouki, 2017; Mubarak et al., 2019). Based on the extensive experiments of (Fadel et al., 2019), deep learning approaches (aka neural approaches) are superior to non-neural approaches especially when large training data is available. In this work, we present several neural ATD models and compare their performance with the state of the art (SOTA) approaches to show that our models are either on par with the SOTA approaches or even better. Finally, we present a novel way to utilize diactritization in order to enhance the accuracy of Machine Translation (MT) models in what we call *Translation over Diacritization* (ToD) approach.

The rest of the paper is organized as follows. The following section discusses the dataset proposed by (Fadel et al., 2019). Sections 3 and 4 discuss our two main approaches: Feed-Forward Neural Network (FFNN) and Recurrent Neural Network (RNN), respectively. Section 5 brielfy discusses the related work and presents a comparison with the SOTA approaches while Section 6 describes our novel approach to integrate diacritization into translation tasks. The paper is concluding in Section 7 with final remarks and future directions of this work.

2 Dataset

The dataset of (Fadel et al., 2019) (which is an adaptation of the Tashkeela Corpus) consists of about 2.3M words spread over 55K lines. Basic statistics about this dataset size, content and diacritics usage are given in Table 1. Among the resources provided with this dataset are new definitions of the Diacritic Error Rate (DER), which is "the percentage of misclassified Arabic characters regardless of whether the character has 0, 1 or 2 diacritics", and the Word Error Rate (WER), which is "the percentage of Arabic words which have at least one misclassified Arabic character".[2] The redefinition of these measures is to exclude counting irrelevant characters such as numbers and punctuations, which were included in (Zitouni and Sarikaya, 2009)'s original definitions of DER and WER. It is worth mentioning that DER/WER are computed in four different ways in the literature depending on whether the last character of each word (referred to as case ending) is counted or not and whether the characters with no diacritization are counter or not.

Table 1: Statistics about the size, content and diacritics usage of (Fadel et al., 2019)'s Dataset

	Train	Valid	Test
Words Count	2,103K	102K	107K
Lines Count	50K	2.5K	2.5K
Avg Chars/Word	3.97	3.97	3.97
Avg Words/Line	42.06	40.97	42.89
0 Diacritics (%)	17.78	17.75	17.80
1 Diacritic (%)	77.17	77.19	77.22
2 Diacritics (%)	5.03	5.05	4.97
Error Diacritics (%)	0	0	0

3 The Feed-Forward Neural Network (FFNN) Approach

This is our first approach and we present three models based on it. In this approach, we consider diacritizing each character as an independent problem. To do so, the model takes a 100-dimensional vector as an input representing features for a single character in the sentence. The first 50 elements in the vector represent the 50 non-diacritic characters before the current character and the last 50 elements represent the 50 non-diacritic characters after it including the current character.

For example, the sentence 'ذَهَبَ عَلِي', the vector related to the character 'ب' is as shown in Figure 1. As the figure shows, there are two characters before the character 'ب' and four after it (including the whitespace). The special token '<PAD>' is used as a filler when there are no characters to feed. Note that the dataset contains 73 unique characters (without the diacritics) which are mapped to unique integer values from 0 to 74 after sorting them based on their unicode representations including the special padding and unknown ('<UNK>') tokens.

Each example belongs to one of the 15 classes under consideration, which are shown in Figure 2. The model outputs probabilities for each

[2] DER/WER are computed with diacritization_stat.py

$\mathcal{X}$ = [<PAD>, ..., <PAD>, ذ, ه, ب, ' ', ع, ل, ي, <PAD>, ..., <PAD>]

Figure 1: Vector representation of a FFNN example.

Class Name	Class Shape	Class Name	Class Shape
No Diacritization	بـ	Shadda	بّ
Fatha	بَ	Shadda + Fatha	بَّ
Damma	بُ	Shadda + Damma	بُّ
Kasra	بِ	Shadda + Kasra	بِّ
Fathatan	بً	Shadda + Fathatan	بًّ
Dammatan	بٌ	Shadda + Dammatan	بٌّ
Kasratan	بٍ	Shadda + Kasratan	بٍّ
Sukun	بْ		

Figure 2: The 15 classes under consideration.

class. Using a Softmax output unit, the class with maximum probability is considered as the correct output. The number of training, validation and testing examples from converting the dataset into examples as described earlier are 9,017K, 488K and 488K respectively.

Basic Model. The basic model consists of 17 hidden layers of different sizes. The activation function used in all layers is Rectified Linear Unit (ReLU) and the number of trainable parameters is about 1.5M. For more details see Appendix A. The model is trained for 300 epochs on an Nvidia GeForce GTX 970M GPU for about 16 hours using AdaGrad optimization algorithm (Duchi et al., 2011) with 0.01 learning rate, 512 batch size, and categorical cross-entropy loss function.

100-Hot Model. In this model, each integer from the 100-integer inputs is converted into its 1-hot representation as a 75-dimensional vector. Then, the 100 vectors are concatenated forming a 7,500-dimensional vector. Based on empirical exploration, the model is structured to have five hidden layers with dropout. It has close to 2M trainable parameters. For more details see Appendix A. The model is trained for 50 epochs on an Nvidia GeForce GTX 970M GPU for about 3 hours using Adam optimization algorithm (Kingma and Ba, 2014) with 0.001 learning rate, 0.9 beta1, 0.999 beta2, 512 batch size, and categorical cross-entropy loss function.

Embeddings Model. In this model, the 100-hot layer is replaced with an embeddings layer to learn feature vectors for each character through the training process. Empirically determined, the model has five hidden layers with only 728K trainable parameters. For more details see Appendix A. The model is trained with the same configurations as the 100-hot model and the training time is about 2.5 hours only.

Results and Analysis. Although the idea of diacritizing each character independently is counter-intuitive, the results of the FFNN models on the test set (shown in Table 2) are very promising with the embeddings model having an obvious advantage over the basic and 100-hot models and performing much better than the best rule-based diacritization system Mishkal[3] among the systems reviewed by (Fadel et al., 2019) (Mishakl DER: 13.78% vs FFNN Embeddings model DER: 4.06%). However, these models are still imperfect. More detailed error analysis of these models is available in Appendix A.

4 The Recurrent Neural Network (RNN) Approach

Since RNN models usually need huge data to train on and learn high-level linguistic abstractions, we prepare an external training dataset following the guidelines of (Fadel et al., 2019). The extra training dataset is extracted from the Classical Arabic (CA) part of the Tashkeela Corpus and the Holy Quran (HQ). We exclude the lines that already exist in the previously mentioned dataset. Note that, with the extra training dataset the number of unique characters goes up to 87 (without the diacritics). Table 3 shows the statistics for the extra training dataset.

The lines in the dataset are split using the following 14 punctuations ('.', '،', '؛', ':', ';', '؟' , '(', ')', '[', ']', '{', '}', '≪' and '≫'). After that, the lines with length more than 500 characters (without counting diacritics) are split into lines of length no more than 500. This step is necessary for the training phase to limit memory usage within a single batch. Note that the splitting procedure is omitted within the prediction phase, e.g., when calculating DER/WER on the validation and test sets. Moreover, four special tokens ('<SOS>', '<EOS>', '<UNK>' and '<PAD>') are used to prepare the input data before feeding it to the model. '<SOS>' and '<EOS>' are added to the start and the end of the sequences, respectively. '<UNK>' is used to represent unknown characters not seen in the training dataset. Finally, '<PAD>' is appended to pad the sequences within the same batch.

[3] https://tahadz.com/mishkal

Table 2: DER/WER comparison of the different FFNN models on the test set

DER/WER	w/ case ending	w/o case ending	w/ case ending	w/o case ending
	Including 'no diacritic'		Excluding 'no diacritic'	
Basic model	9.33% / 25.93%	6.58% / 13.89%	10.85% / 25.39%	7.51% / 13.53%
100-Hot model	6.57% / 20.21%	4.83% / 11.14%	7.75% / 19.83%	5.62% / 10.93%
Embeddings model	**5.52% / 17.12%**	**4.06% / 9.38%**	**6.44% / 16.63%**	**4.67% / 9.10%**

Table 3: Extra training dataset statistics

	Extra Train
Words Count	22.4M
Lines Count	533K
Avg Chars/Word	3.97
Avg Words/Line	42.1
0 Diacritics (%)	17.79
1 Diacritic (%)	77.16
2 Diacritics (%)	5.03
Error Diacritics (%)	0

Four equivalent special tokens are used as an output in the target sequences.

Basic Model. Several model architectures are trained without the extra training dataset. After some exploration, the best model architecture is chosen to experiment with different techniques as described in details throughout this section.

The exploration is done to tune different hyperparameters and find the structure that gives the best DER, which, in most cases, leads to better WER. Because the neural network size have a great impact on performance, we primarily experiment with the number of Bidirectional CuDNN Long Short-Term Memory (BiCuDNNLSTM) (Appleyard et al., 2016) layers and their hidden units. By using either one, two or three layers, the error significantly decreases going from one layer to two layers. However, it shows slight improvement (if any) when going from two layers to three layers while increasing the training time. So, we decide to use two BiCuDNNLSTMs in further experiments as well as 256 hidden units per layer as using less units will increase the error rate while using more units does not significantly improve it. Then, we experiment with the size and depth of the fully connected feed-forward network. The results show that the depth is not as important as the size of each layer. The best results are produced with the model using two layers with 512 hidden units each. All experiments are done using Adam optimization algorithm, because different optimizers like Stochastic Gradient Descent, Adagrad and Adadelta do not converge to the optimal minimal fast enough and RMSprop, Nadam and Adamax give the same or slightly worse results. The number of character features to learn in the embedding layer that gives the best results is 25, where more features leads to little improvement and more overfitting, and less features makes the training harder for the network. This is probably due to the input vocabulary being limited to 87 different characters. We also experiment with training the models for more than 50 epochs, but the return is very little or it makes the learning unstable and eventually causes exploding gradients leaving the network with useless predictions, unable to learn anymore. The best model is structured as shown in Figure 3.

The training is done twice: with and without the extra training dataset, in order to explore the impact of the dataset size on the training phase for the diacritization problem. This has led to reduced overfitting. A weights averaging technique over the last few epochs is applied to partially overcome the overfitting issue and obtain a better generalization.

Models in all following experiments are trained on Google Colab[4] (Carneiro et al., 2018) environment for 50 epochs using an Nvidia Tesla T4 GPU, Adam optimization algorithm with 0.001 learning rate, 0.9 beta1, 0.999 beta2, 10^{-7} epsilon, 256 batch size, and categorical cross-entropy loss function.

Conditional Random Field (CRF) Model. A CRF classifier is used in this model instead of the Softmax layer to predict the network output. CRF is usually more powerful than Softmax in terms of sequence dependencies in the output layer which exist in the diacritization problem. It is worth mentioning that CRF is considered to be "a best practice" in sequence labeling problems. However, in this particular problem, the results show that CRF

[4] http://colab.research.google.com

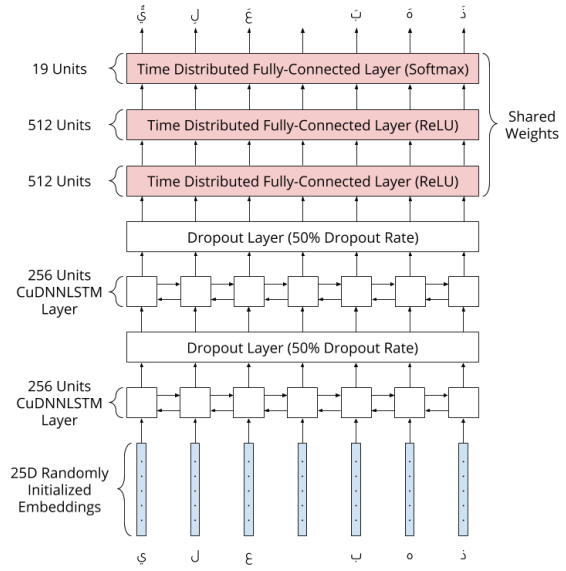

Figure 3: RNN basic model structure.

performs worse than Softmax in most cases except for WER results when training without the extra dataset which indicates that, even with worse DER results, CRF is able to make more consistent predictions within the same word.

Block-Normalized Gradient (BNG) Model. In this model, (Yu et al., 2017)'s BNG method is applied to normalize gradients within each batch. This can help accelerate the training process. According to (Yu et al., 2017), this method performs better in RNN when using optimizers with adaptive step sizes, such as Adam. It can also lead to solutions with better generalization. This coincides with our results.

Discussion and Analysis. The results of the RNN models on the test set (shown in Table 4) are much better than the FFNN models by about 67%. To show the effect of the weights averaging technique, Table 5 reports the DER/WER statistics related to the BNG model after averaging its weights over the last 1, 5, 10, and 20 epochs. Studying the confusion matrices for all the models suggests that the Shadda class and the composite classes (i.e., Shadda + another diacritic) are harder to learn for the network compared to other classes. However, with the extra training dataset, the network is able to find significantly better results compared to the results without the extra training dataset, especially for the Shadda class.

The comparison method for calculating DER/WER without case ending skips comparing the diacritization on the end of each word. This skip improves the best DER to 1.34% (vs 1.69%) and best WER to 2.91% (vs 5.09%) which is a 26% improvement in DER and 43% improvement in WER. This is because the diacritic of the last character of the word usually depends on the part of speech tag making it harder to diacritize. However, we note that the actual last character of the word may come before the end of the word if the word has some suffix added to it.

Arabic sentence	هذا هو كِتابُهُ	قرأ كِتابَهُ	مكتوب في كِتابِهِ
English translation	This is his book	He read his book	It's written in his book
Part of speech tag	subject	object	genitive

Figure 4: Case ending different diacritization with different part of speech tag.

Consider the example shown in Figure 4. The word 'كتابه' means 'his book' where the last character 'ـه' is the suffix representing the pronoun 'his', and the letter before it may take three different diacritics depending on its part of speech tagging. More detailed error analysis of these models available in Appendix B.

Furthermore, an Encoder-Decoder structure (seq2seq) was built using BiCuDNNLSTMs to encode a sequence of characters and generate a sequence of diacritics, but the model was not able to successfully learn the alignment between inputted characters and outputted diacritics. Other attempts tried encoding the sentences as sequences of words and generate a sequences of diacritics also terribly failed to learn.

The BNG model performs the best compared to other models described above. So, it is used for comparison with other systems in the following section.

5 Comparison with Existing Systems

As mentioned earlier, the efforts on building automatic ATD is limited. A recent study (Fadel et al., 2019) surveyed existing approaches and tools for ATD. After discussing the limitations in closed-source tools, they divided existing approaches to ATD into two groups: traditional rule-based approaches (Zitouni and Sarikaya, 2009; Pasha et al., 2014; Shahrour et al., 2015; Alnefaie and Azmi, 2017; Bebah et al., 2014; Azmi and Almajed, 2015; Chennoufi and Mazroui, 2017; Darwish et al., 2017; Fashwan and Alansary, 2017; Alqahtani et al., 2019) and machine learning based approaches (Belinkov and Glass, 2015; Abandah

Table 4: DER/WER comparison of the different RNN models on the test set

DER/WER	w/ case ending	w/o case ending	w/ case ending	w/o case ending
	Including 'no diacritic'		Excluding 'no diacritic'	
Without Extra Train Dataset				
Basic model	2.68% / 7.91%	2.19% / 4.79%	3.09% / 7.61%	2.51% / 4.66%
CRF model	2.67% / 7.73%	2.19% / 4.69%	3.08% / 7.46%	2.52% / 4.60%
BNG model	**2.60% / 7.69%**	**2.11% / 4.57%**	**3.00% / 7.39%**	**2.42% / 4.44%**
With Extra Train Dataset				
Basic model	1.72% / 5.16%	1.37% / 2.98%	1.99% / 4.96%	1.59% / 2.92%
CRF model	1.84% / 5.42%	1.47% / 3.17%	2.13% / 5.22%	1.69% / 3.09%
BNG model	**1.69% / 5.09%**	**1.34% / 2.91%**	**1.95% / 4.89%**	**1.54% / 2.83%**

Table 5: DER/WER comparison showing the effect of the weights averaging technique on BNG model

DER/WER	Averaged Epochs	w/ case ending	w/o case ending	w/ case ending	w/o case ending
		Including 'no diacritic'		Excluding 'no diacritic'	
Without extra train dataset	1	2.73% / 8.08%	2.21% / 4.80%	3.16% / 7.79%	2.54% / 4.68%
	5	2.64% / 7.80%	2.14% / 4.64%	3.04% / 7.49%	2.46% / 4.52%
	10	**2.60% / 7.69%**	**2.11%** / 4.57%	**3.00%/ 7.39%**	**2.42% / 4.44%**
	20	2.61% / 7.73%	**2.11% / 4.56%**	3.01% / 7.42%	**2.42%** / 7.42%
With extra train dataset	1	1.97% / 5.85%	1.61% / 3.55%	2.20% / 5.61%	1.82% / 3.45%
	5	1.73% / 5.20%	1.38% / 3.02%	1.98% / 4.98%	1.58% / 2.92%
	10	1.70% / 5.13%	1.35% / 2.94%	1.96% / 4.92%	1.55% / 2.85%
	20	**1.69% / 5.09%**	**1.34% / 2.91%**	**1.95% / 4.89%**	**1.54% / 2.83%**

et al., 2015, 2017; Barqawi and Zerrouki, 2017; Moumen et al., 2018; Mubarak et al., 2019). The extensive experiments of (Fadel et al., 2019) showed that neural ATD models are superior to their competitors especially when large training data is available. Thus, we limit our attention in this work to such models.

According to (Fadel et al., 2019), the Shakkala system (Barqawi and Zerrouki, 2017) performs the best compared to other existing systems using the test set and the evaluation metrics proposed in (Fadel et al., 2019). Considering our best model's results mentioned previously, it is clear that our model outperforms Shakkala on the testing set after splitting the lines to be at most 315 characters long (Shakkala system limit), which causes a slight drop in our best model's results. However, since Shakkala was also trained on Tashkeela Corpus, we develop an auxiliary test set extracted from three books from Al-Shamela Library[5] 'تاج العروس من جواهر القاموس', 'الفتاوى الكبرى لابن تيمية' and 'فتح الباري شرح صحيح البخاري' using the same extraction and cleaning method proposed by (Fadel et al., 2019) while keeping only lines with more than 80% "diacritics to Arabic characters" rate. The extracted lines are each split into lines of lengths no more than 315 characters (without counting diacritics) which is the input limit of the Shakkala system. This produces a test set consisting of 443K words. Table 6 shows the results comparison with Shakkala.

A comparison with the pre-trained model of (Belinkov and Glass, 2015) is also done using the test set and the evaluation metrics of (Fadel et al., 2019) while splitting the lines into lines of lengths no more than 125 characters (without counting diacritics) since any input with length more than that causes an error in their system. The results show that (Belinkov and Glass, 2015)'s model performs poorly. However, we note that (Belinkov and Glass, 2015)'s system was trained and tested on the Arabic TreeBank (ATB) dataset which consists of text in Modern Standard Arabic (MSA). So, to make a fair comparison with (Belinkov and Glass, 2015)'s system, an auxiliary dataset is built from the MSA part of the Tashkeela Corpus using the same extraction and cleaning method proposed

[5] http://shamela.ws

Table 6: Comparing the BNG model with (Barqawi and Zerrouki, 2017) in terms of DER/WER on the test set

DER/WER	w/ case ending	w/o case ending	w/ case ending	w/o case ending
	Including 'no diacritic'		Excluding 'no diacritic'	
(Fadel et al., 2019) Testing Dataset Results				
Our best model	**1.78% / 5.38%**	**1.39% / 3.04%**	**2.05% / 5.17%**	**1.60% / 2.96%**
Barqawi, 2017	3.73% / 11.19%	2.88% / 6.53%	4.36% / 10.89%	3.33% / 6.37%
Auxiliary Testing Dataset Results				
Our best model	**5.98% / 15.72%**	5.21% / 11.07%	**5.54% / 13.21%**	**4.85% / 9.02%**
Barqawi, 2017	6.41% / 17.52%	**5.12% / 10.91%**	6.82% / 15.92%	5.32% / 9.65%

Table 7: Comparing the BNG model with (Belinkov and Glass, 2015) in terms of DER/WER on the test set

DER/WER	w/ case ending	w/o case ending	w/ case ending	w/o case ending
	Including 'no diacritic'		Excluding 'no diacritic'	
Classical Arabic Testing Dataset Results				
Our best model	**1.99% / 6.10%**	**1.48% / 3.25%**	**2.30% / 5.88%**	**1.70% / 3.17%**
Belinkov, 2015	31.26% / 75.29%	29.66% / 59.46%	35.78% / 74.37%	33.67% / 57.66%
Modern Standard Arabic Testing Dataset Results				
Our best model	**8.05% / 23.56%**	**6.85% / 16.12%**	**8.29% / 21.10%**	**7.16% / 14.41%**
Belinkov, 2015	31.77% / 75.02%	29.21% / 59.40%	37.13% / 73.93%	33.82% / 58.03%

by (Fadel et al., 2019) keeping only lines with more than 80% "diacritics to Arabic characters" rate. This test set consists of 111K words. The results are reported in Table 7. In addition to the poor results of (Belinkov and Glass, 2015)'s system, its output has a large number of special characters inserted randomly. These characters are removed manually to make the evaluation of the system possible.

Finally, we compare our model with (Abandah et al., 2015)'s model which, to our best knowledge, is the most recent deep-learning work announcing the best results so far. To do so, we employ a similar comparison method to (Chennoufi and Mazroui, 2017)'s by using the 10 books from the Tashkeela Corpus and the HQ that were excluded from (Abandah et al., 2015)'s test set. The sentences used for testing our best model are all sentences that are not included in the training dataset of (Fadel et al., 2019) or extra training dataset on which our model is trained. To make the comparison fair, we use the same evaluation metric as (Abandah et al., 2015), which is (Zitouni and Sarikaya, 2009)'s. Moreover, the characters with no diacritics in the original text are skipped similarly to (Abandah et al., 2015). The results are shown in Table 8. It is worth mentioning that the results of (Abandah et al., 2015) include post-processing techniques, which improved DER by 23.8% as reported in (Abandah et al., 2015). It can be easily shown that, without this step, our model's results are actually superior.

All codes related to the diacritization work are publicly available on GitHub,[6] and are also implemented into a web application[7] for testing purposes.

6 Translation over Diacritization (ToD)

Word's diacritics can carry various types of information about the word itself, like its part of speech tag, the semantic meaning and the pronunciation. Intuitively, providing such extra features in NLP tasks has the potential to improve the results of any system. In this section, we show how we benefit from the integration of diacritics into Arabic-English (Ar-En) Neural Machine Translation (NMT) creating what we call Translation over Diacritization (ToD).

Dataset Extraction and Preparation. Due to the lack of free standardized benchmark datasets for Ar-En MT, we create a mid-size dataset using the following corpora: GlobalVoices v2017q3, MultiUN v1, News-Commentary v11, Tatoeba v2, TED2013 v1.1, Ubuntu v14.10, Wikipedia v1.0 (Tiedemann, 2012) downloaded from

[6] https://github.com/AliOsm/shakkelha
[7] https://shakkelha.herokuapp.com

Table 8: Comparing the BNG model with (Abandah et al., 2015) in terms of DER/WER on the test set

	DER		WER	
	w/ case ending	w/o case ending	w/ case ending	w/o case ending
Our best model	2.18%	1.76%	**4.44%**	**2.66%**
Abandah, 2015	**2.09%**	**1.28%**	5.82%	3.54%

Table 9: Vocab size for all sequences types before and after BPE step

Language	Vocab Size	
	Before BPE	After BPE
English	113K	31K
Original Arabic	224K	32K
Diacritized Arabic	402K	186K
Diacritics Forms	41K	15K

the OPUS[8] project. The dataset contains 1M Ar-En sentence pairs split into 990K pairs for training and 10K pairs for testing. The extracted 1M pairs follow these conventions: (i) The maximum length for each sentence in the pair is 50 tokens, (ii) Arabic sentences contain Arabic letters only, (iii) English sentences contain English letters only, and (iv) the sentences do not contain any URLs.

The Arabic sentences in the training and testing datasets are diacritized using the best BNG model. After that, Byte Pair Encoding (BPE)[9] (Sennrich et al., 2015) is applied separately on both English and original (undiacritized) Arabic sequences to segment the words into subwords. This step overcomes the Out Of Vocabulary (OOV) problem and reduces the vocabulary size. Then, diacritics are added to Arabic subwords to create the diacritized version. Table 9 shows the number of tokens before and after BPE step for English, Original Arabic and Diacritized Arabic as well as the Diacritics forms when removing the Arabic characters.

Model Structure The model used in the experiments is a basic Encoder-Decoder sequence to sequence (seq2seq) model that consists of a BiCuDNNLSTM layer for encoding and a CuDNNLSTM layer for decoding with 512 units each (256 per direction for the encoder) while applying additive attention (Bahdanau et al., 2014) on the outputs of the encoder. As for the embeddings layer, a single randomly initialized embeddings layer with vector size 64 is used to represent the subwords when training without diacritics. Another layer with the same configuration is used to represent subwords' diacritics, which is concatenated with the subwords embeddings when training with diacritics. The model structure shown in Figure 5.

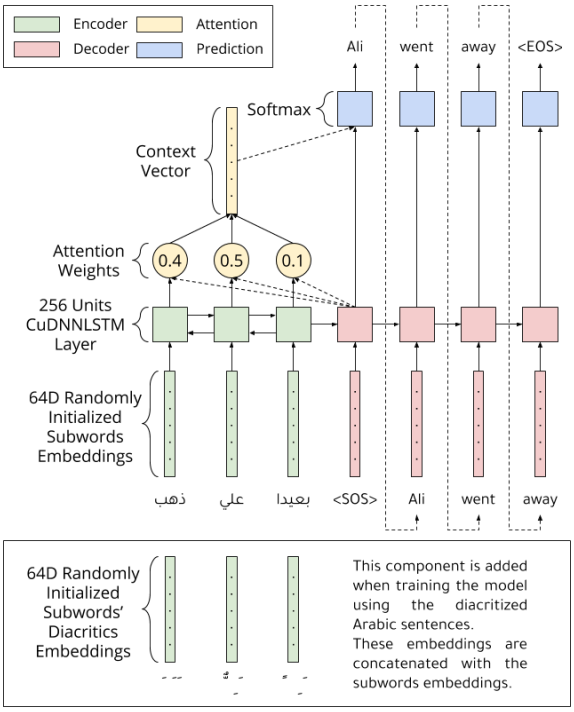

Figure 5: ToD model structure.

Results and Discussion To explore the effect of the Arabic diacritization on the NMT task, we experiment with training both with and without diacritics. The models are trained for 50 epochs using an Nvidia Titan Xp GPU, Adam optimization algorithm with 0.001 learning rate, 0.9 beta1, 0.999 beta2, 10^{-7} epsilon and 256 batch size.

The structure for training the model with diacritics may vary. We experiment with two variations where the first one uses the diacritized version of the sequences, while the other one uses the original sequences and the diacritics sequences in parallel. When merging diacritics with their sequences, we get more variations of each word depending on its different forms of diacritization, therefore expanding the vocabulary size. On the

[8] http://opus.nlpl.eu

[9] https://github.com/rsennrich/subword-nmt

other hand, when separating diacritics from their sequences, the vocab size stays the same, and diacritics are added separately as extra input.

The results in Table 10 show that training the model with diacritization compared to without diacritization improves marginally by 0.31 BLEU score[10] when using the 'with diacritics (merged)' data and improves even more when using the 'with diacritics (separated)' data by 1.33 BLEU score. Moreover, the training time and model size increases by about 20.6% and 41.4%, respectively, for using the 'with diacritics (merged)' data, while they only increase by about 3.4% and 4.5%, respectively, for using the 'with diacritics (separated)' data. By observing Figure 6, which reports the BLEU score on all three models every 5 epochs, it is clear that, although the 'with diacritics (merged)' model converges better at the start of the training, it starts diverging after 15 epochs, which might be due to the huge vocab size and the training data size.

By analysing Figure 6, we find that BLUE score converges faster when training with diacritics (merged) compared to the other two approaches. However, it starts diverging later on due to vocabulary sparsity. As for with diacritics (separated), the BLUE score has higher convergence compared to without diacritics while also maintaining stability compared to with diacritics (merged). This is because separating diacritics solves the vocabulary sparsity issue while also providing the information needed to disambiguate homonym words.

We note that, concurrently to our work, another work on utilizing diacritization for MT has recently appeared. (Alqahtani et al., 2019) used diacritics with text in three downstream tasks, namely Semantic Text Similarity (STS), NMT and Part of Speech (POS) tagging, to boost the performance of their systems. They applied different techniques to disambiguate homonym words through diacritization. They achieved 27.1 and 27.3 BLUE scores without and with diacritics, respectively, using their best disambiguation technique. This is a very small improvement of 0.74% compared to our noticeable improvement of 4.03%. Moreover, our approach is simpler and it does not require to drop any diacritical information.

All codes related to the ToD work are publicly available on GitHub[11].

[10]BLEU scores are computed with multi-bleu.perl

[11]https://github.com/AliOsm/translation-over-diacritization

Table 10: Translation over Diacritization (ToD) results on the test set

Model	Training Time	Model Size	Best BLEU Score
Without	**29 Hours**	**285MB**	33.01
Merged	35 Hours	403MB	33.32
Separated	30 Hours	298MB	**34.34**

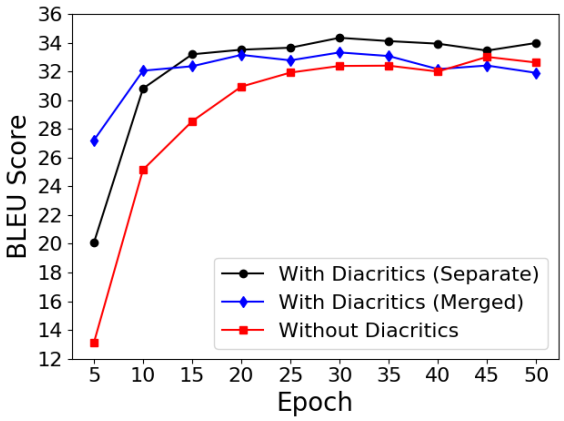

Figure 6: Testing dataset BLEU score while training.

7 Conclusion

In this work, we explored the ATD problem. Our models, which follow two main approaches: FF-NN and RNN, proved to be very effective as they performed on par with or better than SOTA approaches. In the future, we plan on investigating the sequence to sequence models such as RNN Seq2seq, Conv Seq2seq and Transformer. In another contribution of this work, we showed that diacritics can be integrated into other systems to attain enhanced versions in NLP tasks. We used MT as a case study and showed how our idea of ToD improved the results of the SOTA NMT system.

Acknowledgments

We gratefully acknowledge the support of the Deanship of Research at the Jordan University of Science and Technology for supporting this work via Grant #20180193 in addition to NVIDIA Corporation for the donation of the Titan Xp GPU that was used for this research.

References

Gheith Abandah, Alaa Arabiyat, et al. 2017. Investigating hybrid approaches for arabic text diacritization with recurrent neural networks. In *2017 IEEE Jordan Conference on Applied Electrical Engineering*

and Computing Technologies (AEECT), pages 1–6. IEEE.

Gheith A Abandah, Alex Graves, Balkees Al-Shagoor, Alaa Arabiyat, Fuad Jamour, and Majid Al-Taee. 2015. Automatic diacritization of arabic text using recurrent neural networks. *International Journal on Document Analysis and Recognition (IJDAR)*, 18(2):183–197.

Rehab Alnefaie and Aqil M Azmi. 2017. Automatic minimal diacritization of arabic texts. *Procedia Computer Science*, 117:169–174.

Sawsan Alqahtani, Hanan Aldarmaki, and Mona Diab. 2019. Homograph disambiguation through selective diacritic restoration. In *Proceedings of the Fourth Arabic Natural Language Processing Workshop*, pages 49–59.

Jeremy Appleyard, Tomas Kocisky, and Phil Blunsom. 2016. Optimizing performance of recurrent neural networks on gpus. *arXiv preprint arXiv:1604.01946*.

Aqil M Azmi and Reham S Almajed. 2015. A survey of automatic arabic diacritization techniques. *Natural Language Engineering*, 21(3):477–495.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Ahmad Barqawi and Taha Zerrouki. 2017. Shakkala, arabic text vocalization.

Mohamed Bebah, Chennoufi Amine, Mazroui Azzeddine, and Lakhouaja Abdelhak. 2014. Hybrid approaches for automatic vowelization of arabic texts. *arXiv preprint arXiv:1410.2646*.

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.

Tiago Carneiro, Raul Victor Medeiros Da Nóbrega, Thiago Nepomuceno, Gui-Bin Bian, Victor Hugo C De Albuquerque, and Pedro Pedrosa Reboucas Filho. 2018. Performance analysis of google colaboratory as a tool for accelerating deep learning applications. *IEEE Access*, 6:61677–61685.

Amine Chennoufi and Azzeddine Mazroui. 2017. Morphological, syntactic and diacritics rules for automatic diacritization of arabic sentences. *Journal of King Saud University-Computer and Information Sciences*, 29(2):156–163.

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.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Ali Fadel, Ibraheem Tuffaha, Bara' Al-Jawarneh, and Mahmoud Al-Ayyoub. 2019. Arabic text diacritization using deep neural networks. In *ICCAIS*.

Amany Fashwan and Sameh Alansary. 2017. Shakkil: an automatic diacritization system for modern standard arabic texts. In *Proceedings of the Third Arabic Natural Language Processing Workshop*, pages 84–93.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

Rajae Moumen, Raddouane Chiheb, Rdouan Faizi, and Abdellatif El Afia. 2018. Evaluation of gated recurrent unit in arabic diacritization. *International Journal of Advanced Computer Science and Applications (IJACSA)*, 9(11).

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.

Arfath Pasha, Mohamed Al-Badrashiny, Mona T 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 *LREC*, volume 14, pages 1094–1101.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

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.

Jörg Tiedemann. 2012. Parallel data, tools and interfaces in opus. In *Lrec*, volume 2012, pages 2214–2218.

Adams Wei Yu, Lei Huang, Qihang Lin, Ruslan Salakhutdinov, and Jaime Carbonell. 2017. Block-normalized gradient method: An empirical study for training deep neural network. *arXiv preprint arXiv:1707.04822*.

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

Imed Zitouni and Ruhi Sarikaya. 2009. Arabic diacritic restoration approach based on maximum entropy models. *Computer Speech & Language*, 23(3):257–276.